WebSphere® Application Server - Express for IBM i, Version 7.0

Securing WebSphere applications

Note

Before using this information, be sure to read the general information under “Notices” on page 737.

Compilation date: September 24, 2008

© Copyright International Business Machines Corporation 2008.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments v

Changes to serve you more quickly vii

Chapter 1. Web applications 1
Securing Web applications using an assembly tool. 1
Security constraints 3
Security settings 4
Security role references 4
Securing applications during assembly and deployment 5
Assigning users and groups to roles 6
Updating and redeploying secured applications 16
Deploying secured applications 17

Chapter 2. SIP applications 19
Securing SIP applications 19
Configuring security for the SIP container 19

Chapter 3. EJB applications 23
Securing enterprise bean applications 23
Configuring security for EJB 2.1 message-driven beans 25

Chapter 4. Client applications 27
Accessing secure resources using SSL and applet clients 27
Applet client security requirements 27

Chapter 5. Web services 29
Configuring a Web services client to access resources using a Web proxy 29
Provide HTTP endpoint URL information 30
Specify endpoint URL prefixes for Web services 31
Select default HTTP URL prefix 31
Select custom HTTP URL prefix 31
Securing Web services applications at the transport level 31
HTTP transport custom properties for Web services applications 33
Configuring HTTP outbound transport level security with the administrative console 38
Configuring HTTP outbound transport level security using Java properties 40
Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console 41
Configuring additional HTTP transport properties using the wsadmin command-line tool 42
Configuring additional HTTP transport properties for JAX-RPC Web services with an assembly tool 44
Configuring HTTP outbound transport level security with an assembly tool 45
Authenticating Web services clients using HTTP basic authentication 46
Configuring HTTP basic authentication for JAX-RPC Web services with the administrative console 47
Configuring HTTP basic authentication for JAX-RPC Web services programmatically 48
Configuring HTTP basic authentication for JAX-RPC Web services with an assembly tool 49
Custom property settings. 50
Name 50
Value 50
Securing Web services applications using message level security. 50
What is new for securing Web services 52
Web services security configuration considerations 81
Default bindings and runtime properties for Web services security 83
Web services security provides message integrity, confidentiality, and authentication 85

© Copyright IBM Corp. 2008

Securing JAX-WS Web services using message-level security . . . 145
Securing JAX-RPC Web services using message level security . . . 328
Enabling hardware cryptographic devices for Web Services Security . . . 499
Securing Web services for Version 5.x applications based on WS-Security . . . 502
Enabling security for WSIF . . . 655
Configuring UDDI registry security . . . 656
Configuring the UDDI registry to use WebSphere Application Server security . . . 656
Configuring the UDDI registry to use UDDI security . . . 658
Access control for UDDI registry interfaces . . . 659
UDDI registry security additional considerations . . . 660
Security API for the UDDI Version 3 registry . . . 662

Chapter 6. Service integration . . . 663
Security . . . 663
Securing buses . . . 663
Enabling client SSL authentication . . . 672
Adding unique names to the bus authorization policy . . . 673
Administering authorization permissions . . . 674
Administering permitted transports for a bus . . . 700
Securing messages between messaging buses . . . 703
Securing access to a foreign bus . . . 704
Securing links between messaging engines . . . 704
Controlling which foreign buses can link to your bus . . . 705
Securing database access . . . 705
Securing mediations . . . 705
Auditing the service integration security infrastructure . . . 707
Securing bus-enabled Web services . . . 709
Configuring bus-enabled Web services to use an authentication alias to access a secure service integration bus . . . 710
Configuring secure transmission of SOAP messages using WS-Security . . . 712
Working with password-protected components . . . 713
Invoking outbound services over HTTPS . . . 720
Securing WS-Notification . . . 721
Configuring secure access to WS-Notification service points using SOAP over HTTPS . . . 722

Chapter 7. Messaging resources . . . 725
Configuring authorization security for a Version 5 default messaging provider . . . 725
Authorization settings for Version 5 default JMS resources . . . 727
Configuring security for EJB 2.1 message-driven beans . . . 729

Chapter 8. Mail, URLs, and other J2EE resources . . . 731
JavaMail security permissions best practices . . . 731

Chapter 9. Learn about WebSphere programming extensions . . . 733
Scheduler . . . 733
Securing scheduler tasks . . . 733

Appendix. Directory conventions . . . 735

Notices . . . 737

Trademarks and service marks . . . 739

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2008

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with http:// work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

© Copyright IBM Corp. 2008

Chapter 1. Web applications

Securing Web applications using an assembly tool

You can use three types of Web login authentication mechanisms to configure a Web application: basic authentication, form-based authentication and client certificate-based authentication. Protect Web resources in a Web application by assigning security roles to those resources.

About this task

To secure Web applications, determine the Web resources that need protecting and determine how to protect them.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

The following steps detail securing a Web application using an assembly tool:

1. In an assembly tool, import your Web archive (WAR) file or an application archive (EAR) file that contains one or more Web modules.
2. In the Project Explorer folder, locate your Web application.
3. Right-click the deployment descriptor and click **Open With > Deployment Descriptor Editor**. The Deployment Descriptor window opens. To see online information about the editor, press F1 and click the editor name. If you select a Web archive (WAR) file, a Web deployment descriptor editor opens. If you select an enterprise application (EAR) file, an application deployment descriptor editor opens.
4. Create security roles either at the application level or at the Web module level. If a security role is created at the Web module level, the role also displays in the application level. If a security role is created at the application level, the role does not display in all of the Web modules. You can copy and paste a security role at the application level to one or more Web module security roles.
   - Create a role at a Web-module level. In a Web deployment descriptor editor, click the Security tab. Under **Security Roles**, click **Add.**. Enter the security role name, describe the security role, and click **Finish**.
   - Create a role at the application level. In an application deployment descriptor editor, click the Security tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role and then click **Finish**.
5. Create security constraints. Security constraints are a mapping of one or more Web resources to a set of roles.
   a. On the Security tab of a Web deployment descriptor editor, click **Security Constraints**. On the Security Constraints tab, you can do the following actions:
      - Add or remove security constraints for specific security roles.
      - Add or remove Web resources and their HTTP methods.
      - Define which security roles are authorized to access the Web resources.
      - Specify None, Integral, or Confidential constraints on user data.

        None The application does not require transport guarantees.

        Integral
        Data cannot be changed in transit between the client and the server.

        Confidential
        Data content cannot be observed while it is in transit.

        Integral and Confidential usually require the use of SSL. When deploying applications that are available over public networks, specify Confidential for your Web Applications constraints
   b. Under Security Constraints, click **Add**.
   c. Under Constraint name, specify a display name for the security constraint and click **Next**.

© Copyright IBM Corp. 2008

d. Type a name and description for the Web resource collection.
e. Select one or more HTTP methods. The HTTP method options are: GET, PUT, HEAD, TRACE, POST, DELETE, and OPTIONS.
f. Beside the Patterns field, click **Add**.
g. Specify a URL Pattern. For example, type - `/*`, `*.jsp`, `/hello`. Consult the Servlet specification Version 2.4 for instructions on mapping URL patterns to servlets. The security runtime uses the exact match first to map the incoming URL with URL patterns. If the exact match is not present, the security runtime uses the longest match. The wild card (`*.`,`*.jsp`) URL pattern matching is used last.
h. Click **Finish**.
i. Repeat these steps to create multiple security constraints.

6. Map security-role-ref and role-name elements to the role-link element. During the development of a Web application, you can create the security-role-ref element. The security-role-ref element contains only the role-name field. The role-name field contains the name of the role that is referenced in the servlet or JavaServer Pages (JSP) code to determine if the caller is in a specified role. Because security roles are created during the assembly stage, the developer uses a logical role name in the Role-name field and provides enough description in the Description field for the assembler to map the role actual. The Security-role-ref element is at the servlet level. A servlet or JavaServer Pages (JSP) file can have zero or more security-role-ref elements.
   a. Go to the References tab of a Web deployment descriptor editor. On the References tab, you can add or remove the name of an enterprise bean reference to the deployment descriptor. You can define five types of references on this tab:
      - EJB reference
      - Service reference
      - Resource reference
      - Message destination reference
      - Security role reference
      - Resource environment reference
   b. Under the list of Enterprise JavaBeans™ (EJB) references, click **Add**.
   c. Specify a name and a type for the reference in the **Name** and **Ref Type** fields.
   d. Select either **Enterprise Beans in the workplace** or **Enterprise Beans not in the workplace**.
   e. Optional: If you select **Enterprise Beans not in the workplace**, select the type of enterprise bean in the **Type** field. You can specify either an entity bean or a session bean.
   f. Optional: Click **Browse** to specify values for the local home and local interface in the **Local home** and **Local** fields before you click **Next**.
   g. Map every role-name that is used during development to the role using the previous steps. Every role name that is used during development maps to the actual role.
7. Specify the RunAs identity for servlets and JSP files. The RunAs identity of a servlet is used to invoke enterprise beans from within the servlet code. When enterprise beans are invoked, the RunAs identity is passed to the enterprise bean for performing an authorization check on the enterprise beans. If the RunAs identity is not specified, the client identity is propagated to the enterprise beans. The RunAs identity is assigned at the servlet level.
   a. On the Servlets tab of a Web deployment descriptor editor, under **Servlets and JSP**, click **Add**. The Add Servlet or JSP wizard opens.
   b. Specify the servlet or JavaServer Pages (JSP) file settings, including the name, initialization parameters, and URL mappings and click **Next**.
   c. Specify the class file destination.
   d. Click **Next** to specify additional settings or click **Finish**.
   e. Click **Run As** on the **Servlets** tab, select the security role and describe the role.
   f. Specify a RunAs identity for each servlet and JSP file that is used by your Web application.

8. Configure the login mechanism for the Web module. This configured login mechanism applies to all the servlets, JavaServer Pages (JSP) files and HTML resources in the Web module.
   a. Click the **Pages** tab of a Web deployment descriptor editor and click **Login**. Select the required authentication method. Available method values include: Unspecified, Basic, Digest, Form, and Client-Cert.
   b. Specify a realm name.
   c. If you select the Form authentication method, select a login page and an error page Web address. For example, you might use `/login.jsp` or `/error.jsp`. The specified login and error pages are present in the `.war` file.
   d. Install the client certificate on the browser or Web client and place the client certificate in the server trust keyring file, if `ClientCert` is selected.
9. Close the deployment descriptor editor and, when prompted, click **Yes** to save the changes.

Results

After securing a Web application, the resulting Web archive (WAR) file contains security information in its deployment descriptor. The Web module security information is stored in the `web.xml` file. When you work in the Web deployment descriptor editor, you also can edit other deployment descriptors in the Web project, including information on bindings and IBM® extensions in the `ibm-web-bnd.xmi` and `ibm-web-ext.xmi` files.

What to do next

After using an assembly tool to secure a Web application, you can install the Web application using the administrative console. During the Web application installation, complete the steps in “Deploying secured applications” on page 17 to finish securing the Web application.

Security constraints

Security constraints determine how Web content is to be protected.

These properties associate security constraints with one or more Web resource collections. A constraint consists of a Web resource collection, an authorization constraint and a user data constraint.

- A Web resource collection is a set of resources (URL patterns) and HTTP methods on those resources. All requests that contain a request path that matches the URL pattern described in the Web resource collection are subject to the constraint. If no HTTP methods are specified, then the security constraint applies to all HTTP methods.
- An authorization constraint is a set of roles that users must be granted in order to access the resources described by the Web resource collection. If a user who requests access to a specified Uniform Resource Identifier (URI) is not granted at least one of the roles specified in the authorization constraint, the user is denied access to that resource.

  Previously the http-methodType schema limited HTTP methods to DELETE, GET, HEAD, OPTIONS, POST, PUT and TRACE. The http-methodType schema has changed. The http-methodType schema has been changed as follows:

```
<xsd:simpleType name="http-methodType">
<xsd:annotation>
<xsd:documentation>
A HTTP method type as defined in HTTP 1.1 section 2.2.
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:token">
<xsd:pattern value="[\p{L}-[\p{Cc}\p{Z}]]+"/>
</xsd:restriction>
</xsd:simpleType>
```

This requires elements to be a token. Based upon the pattern value, tokens can contain any character except for control characters and separators.
- A user data constraint indicates that the transport layer of the client or server communications process must satisfy the requirement of either guaranteeing content integrity (preventing tampering in transit) or guaranteeing confidentiality (preventing reading while in transit).

Security settings

Use the administrative console to modify the security settings for all applications.

You can enable security for applications by selecting the **Enable application security** option on the Global security panel.

Note that:
- Global settings apply to existing and future applications and cannot be customized.
- Default settings apply only to future applications and can be customized.

The default settings are used as a template or starting point for configuring individual applications. The administrator should still explicitly configure security settings for each application.

The following security settings are specified during application assembly:

Security role settings

When using the Assembly Toolkit at an application level (Enterprise Archive (EAR) file), security roles are synchronized with the security roles defined for the embedded modules of the application.

If a security role is manually added to the EAR file, it can be automatically removed when the file is saved if an embedded module does not reference the role, or the role is in conflict with an existing role. In this case, remove the manually added role, but then all roles with the same name are removed.

The role is automatically added again when the file is saved if it is still referenced in an embedded module file. If a duplicate role is added in an embedded module file, delete all roles with the same name and manually read the correct role.

Security constraints

Security constraints declare how to protect Web content. These properties associate security constraints with one or more Web resource collections. A *constraint* consists of a Web resource collection, an authorization constraint, and a user data constraint.

Security constraints are set when configuring a Web application in the Assembly Toolkit.

Security role references

Web application developers or Enterprise JavaBeans (EJB) providers must use a role-name in the code when using the available programmatic security Java™ Platform, Enterprise Edition (Java EE) application programming interfaces (APIs) isUserInRole(String roleName) and isCallerInRole(String roleName).

The roles used in the deployed run-time environment might not be known until the Web application and EJB components (for example, Web archive (WAR) files and `ejb-jar.xml` files) are assembled into an enterprise archive (EAR) file. Therefore, the role names used in the Web application or EJB component code are logical role names which the application assembler maps to the actual run-time environment roles during application assembly. The security role references provide a level of indirection that insulate Web application component and EJB developers from having to know the actual roles in the run-time environment.

The definition of the logical roles and the mapping to the actual run-time environment roles are specified in the security-role-ref element of both the Web application and the EJB JAR file deployment descriptors,

web.xml and ejb-jar.xml respectively. Use the assembly tools to define the role names and map them to the actual run-time roles in the environment with the role-link element.

The following code sample is an example of a security-role-ref from an EJB ejb-jar.xml deployment descriptor.

```
... <enterprise-beans>
... <entity>
<ejb-name>AardvarkPayroll</ejb-name>
<ejb-class>com.aardvark.payroll.PayrollBean</ejb-class>
...
<security-role-ref>
<description>
```

This role should be assigned to the employees of the payroll department. Members of this role have access to the payroll record of everyone. The role has been linked to the payroll-department role. This role should be assigned to the employees of the payroll department. Members of this role have access to all payroll records. The role has been linked to the payroll-department role.

```
</description> <role-name>payroll</role-name>
<role-link>payroll-department</role-link>
</security-role-ref>
 ...
</entity>
 ...
</enterprise-beans>
```

In the previous example, the string payroll, which appears in the <role-name> element, is what the EJB provider uses as the argument to the isCallerInRole() API. The <role-link> element is what ties the logical role to the actual role used in the run-time environment.

Note that for enterprise beans, the security-role-ref element must appear in the deployment descriptor even if the logical role name is the same as the actual role name in the environment.

The rules Web application components are slightly different. If no security-role-ref element matching a security-role element is declared, the container must default to checking the role-name element argument against the list of security-role elements for the Web application. The isUserInRole method references the list to determine whether the caller is mapped to a security role. The developer must be aware that the use of this default mechanism can limit the flexibility in changing role names in the application without having to recompile the servlet making the call.

See the EJB Version 2.0 and Servlet Version 2.3 specification in the Security: Resources for Learning article for complete details on this specification.

Securing applications during assembly and deployment

Several assembly tools exist that are graphical user interfaces for assembling enterprise or Java Platform, Enterprise Edition (Java EE) applications. You can use these tools to assemble an application and secure Enterprise JavaBeans (EJB) and Web modules in that application.

About this task

An EJB module consists of one or more beans. You can enforce security at the EJB method level. A Web module consists of one or more Web resources: an HTML page, a JavaServer Pages (JSP) file, or a servlet. You can also enforce security for each Web resource.

Note: For information about the tools that WebSphere® Application Server supports, see Assembly tools.

To secure an EJB module, a Java archive (JAR) file, a Web module, a Web archive (WAR) file, or an application enterprise archive (EAR) file, you can use an assembly tool. You can create an application, an

EJB module, or a Web module and secure them using an assembly tool or development tools such as the IBM Rational® Application Developer.

1. Secure EJB applications using an assembly tool. For more information, see “Securing enterprise bean applications” on page 23.
2. Secure Web applications using an assembly tool. For more information, see “Securing Web applications using an assembly tool” on page 1.
3. Add users and groups-to-roles while assembling a secured application using an assembly tool. For more information, see “Adding users and groups to roles using an assembly tool” on page 8.
4. Map users to RunAs roles using an assembly tool. For more information, see “Mapping users to RunAs roles using an assembly tool” on page 13.
5. Adding the was.policy file to applications.
6. Assemble the application components that you secured using an assembly tool. For more information, see Assembling applications.

Results

After securing an application, the resulting `.ear` file contains security information in its deployment descriptor. The EJB module security information is stored in the `ejb-jar.xml` file and the Web module security information is stored in the `web.xml` file. The `application.xml` file of the application EAR file contains all the roles that are used in the application. The user and group-to-roles mapping is stored in the `ibm-application-bnd.xmi` file of the application EAR file.

The `was.policy` file of the application EAR contains the permissions granted for the application to access system resources protected by Java 2 security.

This task is required to secure EJB modules and Web modules in an application. This task is also required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not be able to access system resources.

What to do next

After securing an application, you can install an application using the administrative console. When you install a secured application, refer to “Deploying secured applications” on page 17 to complete this task.

Assigning users and groups to roles

This topic describes how to assign users and groups to roles if you are using WebSphere Application Server authorization for Java Platform, Enterprise Edition (Java EE) roles.

Before you begin

Before you perform this task:

- Secure the Web applications and Enterprise JavaBeans (EJB) applications where new roles are created and assigned to Web and enterprise bean resources.
- Create all the roles in your application.
- Verify that you have properly configured the user registry that contains the users that you want to assign. It is preferable to have security turned on with the user registry of your choice before beginning this process.
- Make sure that if you change anything in the security configuration you save the configuration and restart the server before the changes become effective. For example, enable security or change the user registry.

About this task

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the Security role to user/group mapping link during application installation and also during application management, as a link in the Additional properties section.

1. Access the administrative console.

   Type `http://server_name:port_number/ibm/console` in a Web browser.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name*** .
3. Under Detail properties, click **Security role to user/group mapping**. A list of all the roles that belong to this application is displayed. If the roles already have users or All Authentication or Everyone special subjects assigned, they display here.
4. To assign the special subjects, select either the **Everyone** or the **All Authenticated in Application's Realm** option for the appropriate roles.
5. To assign users or groups, select the role. You can select multiple roles at the same time, if the same users or groups are assigned to all the roles.
6. Click **Look up users** or **Look up groups**.
7. Get the appropriate users and groups from the user registry by completing the Limit and the Search string fields and by clicking **Search**. The Limit field limits the number of users that are obtained and displayed from the user registry. The pattern is a searchable pattern matching one or more users and groups. For example, `user*` lists users like user1, user2. A pattern of asterisk (*) indicates all users or groups.

   Use the limit and the search strings cautiously so as not to overwhelm the user registry. When you use large user registries such as Lightweight Directory Access Protocol (LDAP) where information on thousands of users and groups resides, a search for a large number of users or groups can make the system slow and can make it fail. When more entries exist than requests for entries, a message displays on top of the panel. You can refine your search until you have the required list.

   If the search string you are using has no matches, a NULL error message is displayed. This message is informational and does not necessarily indicate an error, as it is valid to have no entries matching your selected criteria.
8. Select the users and groups to include as members of these roles from the **Available** field and click **>>** to add them to the roles.
9. To remove existing users and groups, select them from the **Selected** field and click **<<**. When removing existing users and groups from roles, use caution if those same roles are used as RunAs roles.

   For example, if the user1 user is assigned to the role1 RunAs role and you try to remove the user1 user from the role1 role, the administrative console validation does not delete the user. A user can only be part of a RunAs role if the user is already in a role either directly or indirectly through a group. In this case, the user1 user is in the role1 role. For more information on the validation checks that are performed between RunAs role mapping and user and group mapping to roles, see "Assigning users to RunAs roles" on page 11.
10. Click **OK**. If any validation problems exist between the role assignments and the RunAs role assignments, the changes are not committed and an error message that indicates the problem displays at the top of the panel. If a problem exists, make sure that the user in the RunAs role is also a member of the regular role. If the regular role contains a group that contains the user in the RunAs role, make sure that the group is assigned to the role using the administrative console. Follow steps 4 and 5. Avoid using any process where the complete name of the group, host name, group name, or distinguished name (DN) is not used.

Results

The user and group information is added to the binding file in the application. This information is used later for authorization purposes.

What to do next

This task is required to assign users and groups to roles, which enables the correct users and groups to access a secured application. If you are installing an application, complete your installation. After the application is installed and running you can access your resources according to the user and group mapping that you did in this task. If you manage applications and modify the users and groups to role mapping, make sure you save, stop, and restart the application so that the changes become effective. Try accessing the Java EE resources in the application to verify that the changes are effective.

Note: In a distributed environment, when you install WebSphere Application Server with samples, enable security using federated repositories, and start the server1 server with sample applications, the server might create exceptions. However, the server starts successfully. The deployment manager did not create user and group samples when it created the deployment manager profile. To resolve exceptions caused by the samples failing to load, create your own sample users and groups. In the administrative console, do the following:

1. Click **Users and Groups > Manage Users**.
2. Create the `samples` user and the `sampadmn` group. The `samples` user is a member of the `sampadmn` group.

For more assistance, refer to the ″Managing users″ help topic by clicking **More information about this page** at the top right of the Manage Users panel.

Adding users and groups to roles using an assembly tool

After creating new roles and assigning them to enterprise bean and Web resources, use this task to add users and groups to roles with an assembly tool.

Before you begin

Before you perform this task, you already completed the steps in “Securing Web applications using an assembly tool” on page 1 and “Securing enterprise bean applications” on page 23 where you created new roles and assigned those roles to enterprise bean and Web resources. Complete these steps during application installation. The environment user registry under which the application is running is not known until deployment.

About this task

If you already know the environment in which the application is running and the user registry that is used, you can use an assembly tool to assign users and groups to roles. Using the administrative console to assign users and groups to roles is recommended.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To add users and groups to roles using an assembly tool, follow these steps:

1. In the Project Explorer view of an assembly tool, right-click an enterprise application project, or Enterprise Archive (EAR) file, and click **Open With** > **Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
2. Click the **Security** tab and, under the main panel, click **Add**.

3. In the Add Security Role wizard, name and describe the security role. Click **Finish**.
4. Under WebSphere Bindings, select the user or group extension properties for the security role. Available values include: Everyone, All authenticated users, and Users/Groups.
5. If you selected Users/Groups, click **Add** beside the Users or Groups panes. In the wizard that opens, specify a user or group name and click **Finish**. Repeat this step until you added all the users and groups to which the security role applies.
6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

Results

The `ibm-application-bnd.xmi` or `ibm-application-bnd.xml` file in the application contains the users and groups-to-roles mapping table, which is the *authorization table*. For Java EE Version 5 applications, the `ibm-application-bnd.xml` file contains the authorization table.

What to do next

After securing an application, install the application using the administrative console.

Security role to user or group mapping

Use this page to specify the users and groups that are mapped to the security roles that are used with the enterprise application.

To view this administrative console page, click **Applications > Application types > WebSphere enterprise applications >***application_name*. Under Detail Properties, click **Security role to user/group mapping**.

Button	Resulting action
Map Users	Lists the users that are mapped to the specified role within this application.
Map Groups	Lists the groups that are mapped to this specified role within this application.
Map Special Subjects	This choice appears if multiple realms are being used. It enables you to map any of the following Special Subjects to a selected role: • **All authenticated in application's realm**: All authenticated users that are in the applications's realm, which specifies whether to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role. This selection also applies to all authenticated users regardless of the realm. • **Everyone**: map everyone to the selected role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security. • **None**: Do not map anyone to the selected role **Note:** • For all users in the trusted realms. If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm. • If the secured realm cannot be reached, the left list is replaced with 3 text fields (that is, `name`, `realm`, and `uid`). You can add the user when the secured realm is not available. It is not possible to map two subjects to the same role in this release of WebSphere Application Server.

Role:

Lists the specific capabilities to a user. Role privileges give users and groups permission to run as specified.

For example, you might map the user Joe to the administrator role, which enables user Joe to perform all of the tasks associated with the administrator role.

The authorization policy is only enforced when global security is enabled.

Mapped users:

Lists the users that are mapped to the specified role within this application.

Special subjects:

Lists which special subjects are mapped to the security role when an application uses multiple realms.

Mapped groups:

Lists the groups that are mapped to this specified role within this application.

Look up users

Use this page to select and to map users, groups and special subjects for security roles.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application types > WebSphere enterprise applications >** *application_name*.
2. Under Detail Properties, click **Security role to user/group mapping**.
3. Select the role and click either **Map users...**, **Map groups...** or **Map Special Subjects**.

Note: Once you click **OK** after making any changes, you must also click **OK** on the previous panel for the changes to be accepted.

Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users and groups are associated with roles defined in an application when the application is installed or configured. Use the Search pattern field to display users in the Available list. Click **>>** to add users from the Available list to the Selected list.

Map users...:

Lists the users that are mapped to the specified role within this application.

Map groups...:

Lists the groups that are mapped to this specified role within this application.

Map Special Subjects:

This choice appears if multiple realms are being used. It enables you to map any of the following to selected roles:

- All authenticated users that are in the applications's realm, which specifies whether to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role.
- All authenticated users regardless of the realm.

- Everyone, which specifies whether to map everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.
- All users in the trusted realms.

  If trusted realms are configured, a drop-down list of realms to search is displayed. Users from the non-default realm are displayed as user@realm.

Note: If the secured realm cannot be reached, the left list is replaced with 3 text fields (that is, `name`, `realm`, and `uid`). You can add the user when the secured realm is not available.

It is not possible to map two subjects to the same role in this release of WebSphere Application Server.

Limit:

Specifies the maximum number of users or groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users or groups that match the pattern. You can either increase the limit or refine the search pattern to get all the entries.

Data type	Integer
Units	User name
Default	20
Range	0 or more

Search string:

Indicates the search pattern used to search for the entries in a user registry.

The Search string field contains the search pattern that is used to search for the user or group entries. For example, bob* will search all users or groups starting with bob. A limit of zero (0) retrieves all of the entries that match the pattern. Use a limit of zero (0) only when a small number users or groups match that pattern in the user registry. If the user registry contains more entries that match the pattern than requested for, a message shows in the administrative console to indicate that there are more entries in the user registry.

Data type	String
Units	Number of users
Default	20
Range	A-Z with *

Assigning users to RunAs roles

This article explains how to assign users to the RunAs roles for your application.

Before you begin

Complete the following tasks:
- Secure the Web applications and the EJB applications where new RunAs roles are created and assigned to Web and EJB resources.
- Create all the RunAs roles in your application. The user in the RunAs role can only be entered if that user or a group to which that user belongs is already part of the regular role.
- Assign users and groups to security roles. Refer to “Assigning users and groups to roles” on page 6 for more information.
- Verify that the user registry requirements are met. These requirements are the same as those discussed in “Assigning users and groups to roles” on page 6. For example, if the role1 role is a role that is also

used as a RunAs role, then the user1 user can be added to the RunAs role. The administrative console checks this logic when **Apply** or **OK** is clicked. If the check fails, the change is not made and an error message is displayed at the top of the panel.

When a user ID and password is assigned to a RunAs role, validation occurs using the current active user registry that is configured. By default, the local operating system registry is set as the active user registry. Therefore, when an application is installed and security is disabled on the server, the local operating system registry is used to validate the user ID and password that is assigned to the RunAs Role. If the intended registry for the application is not local operative system, the validation fails. Therefore, map RunAs roles to users when the security is enabled on the server. However, if the active user registry and the intended registry after enabling security are the same, you can assign the user to a RunAs role when security is disabled.

If the Everyone or All Authenticated special subjects are assigned to a role, validation does not occur for that role.

Validation is done every time you click **Apply** in this panel or when you click **OK** in the Security role to user/group mapping panel. The check verifies that all the users in all the RunAs roles do exist directly or indirectly through a group in those roles in the Security role to user/group mappings panel. If a role is assigned both a user and a group to which that user belongs, you can delete either the user or the group from the Security role to user/group mapping panel.

If the RunAs role user belongs to a group and if that group is assigned to that role, make sure that the assignment of this group to the role is done through the administrative console and not through an assembly tool or other method. When using the administrative console, the full name of the group is used (for example, `hostname\groupName` in Windows® systems and distinguished names (DN) in Lightweight Directory Access Protocol (LDAP)). During the check, all the groups to which the RunAs role user belongs are obtained from the user registry. Because the list of groups that are obtained from the user registry are the full names of the groups, the check works correctly. If the short name of a group is entered using an assembly tool, for example `group1` instead of `CN=group1, o=myCompany.com`, this check fails.

About this task

These steps are common to both installing an application and modifying an existing application. If the application contains RunAs roles, you see the User RunAs roles link during application installation and also during managing applications as a link in the Additional properties section.

1. Click **Applications > Enterprise Applications > *application_name***.
2. Under Detail Properties, click **Security role to user/group mapping**. A list of all the RunAs roles that belong to this application display. If the roles already have users assigned, they display here.
3. To assign a user, select the role. You can select multiple roles at the same time if the same user is assigned to all the roles.
4. Enter the user's name and password in the designated fields. The user name entered can be either the short name, which is preferred, or the full name, as seen when getting users and groups from the user registry.
5. Click **Apply**. The user is authenticated using the active user registry. If authentication is successful, a check is made to verify that this user or group is mapped to the role in the Map security roles to users and groups panel. If authentication fails, verify that the user and password are correct and that the active registry configuration is correct.
6. To remove a user from a RunAs role, select the roles and click **Remove**.

Results

The RunAs role user is added to the binding file in the application. This file is used for delegation purposes when accessing Java EE resources. This step is required to assign users to RunAs roles so that

during delegation the appropriate user is used to invoke the EJB methods.

What to do next

If you are installing the application, complete installation. After the application is installed and running, you can access your resources according to the RunAS role mapping. Save the configuration.

If you manage applications and modify User RunAs roles, make sure you save, stop, and restart the application so that the changes become effective. Try accessing your Java Platform, Enterprise Edition (Java EE) resources to verify that the new changes are in effect.

Mapping users to RunAs roles using an assembly tool:

RunAs roles are used for delegation. A servlet or enterprise bean component uses the RunAs role to invoke another enterprise bean by impersonating that role.

Before you begin

Before you perform this task:
- Secure the Web application and enterprise bean applications, including creating and assigning new roles to enterprise bean and Web resources. For more information, see “Securing Web applications using an assembly tool” on page 1 and “Securing enterprise bean applications” on page 23.
- Assign users and groups to roles. For more information, see “Adding users and groups to roles using an assembly tool” on page 8. Complete this step during the installation of the application. The environment or user registry under which the application is going to run is not known until deployment. If you already know the environment in which the application is going to run and you know the user registry, then you can use an assembly tool to assign users to RunAs roles.

About this task

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To define RunAs roles when a servlet or an enterprise bean in an application is configured with RunAs settings, perform these steps:

1. In the Project Explorer view of an assembly tool, right-click an enterprise application project or Enterprise Archive (EAR) file and click **Open With** > **Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
2. On the Security tab, under Security Role Run As Bindings, click **Add**.
3. Click **Add** under RunAs Bindings.
4. In the Security Role wizard, select one or more roles and click **Finish**.
5. Repeat steps 3 through 5 for all the RunAs roles in the application.
6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

Results

The `ibm-application-bnd.xmi` file in the application contains the user to RunAs role mapping table.

What to do next

After securing an application, you can install the application using the administrative console. You can change the RunAs role mappings of an installed application. For more information, see “User RunAs collection” on page 16.

Ensure all unprotected 1.x methods have the correct level of protection:

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 1.*x* methods have the correct level of protection before you map users to roles.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Application > New application > New Enterprise Application** . The panel is displayed as *Ensure all unprotected 1.x methods have the correct level of protection* in the application deployment steps. On this administrative console panel, you can specify whether users can access specific EJB modules.

EJB module:

Specifies the EJB module name.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Deny all access:

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Default: Cleared

Ensure all unprotected 2.x methods have the correct level of protection:

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 2.*x* methods have the correct level of protection before you map users to roles.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Applications > New application** *application_name*. The panel is displayed as *Ensure all unprotected 2.x methods have the correct level of protection* in the application deployment steps. On this administrative console panel, you can specify whether users can access specific EJB modules.

To use this administrative console page, select the **Uncheck**, **Exclude**, or **Role** option, the check box next to the EJB module, and click **Apply**. If you select **Role** option, select the appropriate role for the EJB module before you click **Apply**.

Uncheck:

Select this option if you do not want the application server to verify the access permissions for the EJB module. Everyone can access the EJB module.

Default: Selected

Exclude:

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Default: Deselected

Role:

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the module, then it is added to the modules or Java archive (JAR) files.

Default: Deselected

EJB module:

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Protection type:

Specifies the level of protection that is assigned to a particular module name.

After you select the **Uncheck**, **Exclude**, or **Role** option and click **Apply**, the selected protection option is displayed in this column.

Correct use of the system identity:

Use this page to manage the system identity properties for the Enterprise JavaBeans (EJB) method in your application.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Application > New application > New Enterprise Application** . The panel is displayed as *Correct use of System Identity* in the application deployment steps.

To use this panel, complete the following steps:

1. Select the check box next to the EJB method.
2. Select a role that is defined for this enterprise bean.
3. Specify a user name and password for the RunAs role. The user name must be defined in your user registry.
4. Click **Apply**.
   The specified user will be assigned to the specified RunAs role for the EJB method that you selected.

Role:

Specifies the RunAs role that is used for this EJB method.

Username:

Specifies the user name that is assigned to the RunAs role for this EJB method.

The user name is used in conjunction with the RunAs role that you select for the Role.

Password:

Specifies the password that is associated with the user name in the user registry.

User RunAs collection:

Use this page to map a specified user identity and password to a RunAs role. This panel enables you to specify application-specific privileges for individual users to run specific tasks using another user identity.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >** *application_name*.
2. Under Detail properties, click **Security role to user/group mapping**.

The enterprise beans that you install contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

Username:

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role. After you specify the user name and password for the user and select a RunAs role, click **Apply**.

Data type: String

Password:

Specifies the password for the RunAs user.

Data type: String

Role:

Maps specific capabilities to a user.

The authorization policy is only enforced when global security is enabled.

Updating and redeploying secured applications

This section addresses the way to update existing applications.

Before you begin

Before you perform this task, secure Web applications, secure Enterprise JavaBeans (EJB) applications, and deploy them in WebSphere Application Server.

1. Use the administrative console to modify the existing users and groups mapping to roles. For information on the required steps, see “Assigning users and groups to roles” on page 6.
2. Use the administrative console to modify the users for the RunAs roles. For information on the required steps, see “Assigning users to RunAs roles” on page 11.
3. Complete and save the changes.
4. Stop and restart the application for the changes to become effective.
5. Use an assembly tool. For more information, see Assembling applications.
6. Use an assembly tool to modify roles, method permissions, auth-constraints, data-constraints and so on. For more information, see Assembling applications.

7. Save the enterprise archive (EAR) file, uninstall the old application, deploy the modified application and start the application to make the changes effective.

Results

The applications are modified and redeployed. This step is required to modify existing secured applications.

What to do next

If information about roles is modified, make sure that you update the user and group information using the administrative console. After the secured applications are modified and either restarted or redeployed, verify that the changes are effective by accessing the resources in the application.

Deploying secured applications

Deploying applications that have security constraints (secured applications) is not much different than deploying applications that do not contain any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application. The secured application requires that you have the correct active user registry.

Before you begin

Before you perform this task, verify that you already designed, developed, and assembled an application with all the relevant security configurations. For more information on these tasks refer to Developing applications that use programmatic security and “Securing applications during assembly and deployment” on page 5. In this context, deploying and installing an application are considered the same task.

To deploy a newly secured application click **Applications > Install New Application** and follow the prompts to complete the installation steps. One of the required steps to deploy secured applications is to assign users and groups to roles that are defined in the application.

- If you are installing a secured application, roles will be defined in the application.
- If delegation is required in the application, you will be defining RunAs roles also.

During the installation of a new application, the role definition is completed as part of the step that maps security roles to users and groups. If this assignment has already been completed by using an assembly tool, you can still confirm the mapping by going through this installation step. You can add new users and groups and modify existing information during this step.

If the application supports delegation, a RunAs role will already be defined in the application. If the delegation policy is set to `Specified Identity` during assembly, the intermediary invokes a method by using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user `bob` and the client `alice` is invoking a servlet, with delegation set that calls the enterprise beans, the method on the enterprise beans is invoked with `bob` as the identity.

As part of the new application installation and deployment process, one of the steps is to map or modify users to the RunAs roles. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to `Specified Identity`.

About this task

Note that the steps are common whether you are installing an application or modifying an existing application.

To install and deploy the application, complete the following steps.

1. Click **Applications > Install New Application**. Complete the required steps until you see the step for mapping security roles to users and groups.
2. If the application contains roles, assign users and groups to roles. At this step during the installation, under Additional Properties, click **Map security roles to users and groups**. For more information, see “Assigning users and groups to roles” on page 6.
3. If RunAs roles exist in the application, assign users to RunAs roles. At this step during the installation, under Additional Properties, click **Map RunAs roles to users**. For more information, see “Assigning users to RunAs roles” on page 11.
4. Optional: Click **Correct use of System Identity** to specify RunAs roles, if needed. Complete this action if the application has delegation set to use system identity, which is applicable to enterprise beans only. System identity uses the WebSphere Application Server security server ID to invoke downstream methods. Using system identity is not recommended as this ID has more privileges than other identities in accessing WebSphere Application Server internal methods. This task is provided to make sure that the deployer is aware that the methods listed in the panel have system identity set up for delegation and to correct them if necessary. When the internalServerId feature is used, runAs with system identity is not supported; you must specify RunAs roles here.
5. Complete the remaining non-security related steps to finish installing and deploying the application.

What to do next

After a secured application is deployed, verify that you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure only the users that you assigned to the roles can use the application.

Chapter 2. SIP applications

Securing SIP applications

You can apply digest authentication and Trust Association Interceptor (TAI) for a SIP application by applying Lightweight Directory Access Protocol (LDAP) security to the application.

Before you begin

Before you can apply security, you must first deploy an application that has been developed to support security (with the web.xml file configured for security) and roles. The following software must also be installed:

1. Install a supported LDAP server. For a list of supported LDAP servers, see the IBM Web site for WebSphere Application Server supported hardware, software, and APIs.
2. Set up and activate Lightweight Third Party Authentication. For more information, see theConfiguring the Lightweight Third Party Authentication mechanism topic.

About this task

To apply LDAP security to a SIP application, click **Applications** → **Enterprise Applications** → ***applicationName*** and complete the following steps:

1. Click **Detail Properties** → **Security role to user/group mapping**.
2. Check **All Authenticated**.
3. Save all changes.
4. Restart the server.

Configuring security for the SIP container

This section provides instructions specific to security for the SIP container.

Before you begin

Before you can configure security for your SIP container, you will need to:

1. Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.
2. Install a supported LDAP server.

You may also need to:

- Adjust key group settings. Refer to Lightweight Third Party Authentication key sets and key set groups for LTPA key information.
- Establish and configure Trust Association Interceptor (TAI) settings. Refer to Trust association interceptor settings.

About this task

You must know the name of the key set group and the management scope where the key set group is defined in order to activate and secure LTPA with keys. Refer to Activating Lightweight Third Party Authentication key versions for the setup and activation procedures.

To configure security based on the Lightweight Directory Access Protocol (LDAP), you can configure digest authentication for your supported LDAP server.

- To configure digest authentication and TAI on WebSphere Application Server for Tivoli, select “Configuring digest authentication and TAI for SIP” on page 20.

© Copyright IBM Corp. 2008

- To configure digest authentication on WebSphere Application Server for Oracle Internet Directory, select “Configuring digest authentication for Oracle Internet Directory” on page 21.

To define an LDAP connection between WebSphere Application Server and LDAP, use the security wizard. It can also be defined by selecting it from available realms and defining the proper connection properties to connect LDAP.

To set up a TAI, you must specify the trust information for any reverse security proxy servers. See Trust association interceptor settings to configure TAI settings.

Configuring digest authentication and TAI for SIP

You can configure digest authentication and Trust Association Interceptor (TAI) for the Session Initiation Protocol (SIP).

Before you begin

Before you can configure digest authentication and TAI, you must either install a supported LDAP server, or configure digest TAI to work without LDAP.

To configure digest TAI to work without LDAP, complete these steps:

1. Create a class that implements the interface: com.ibm.ws.sip.security.digest.DigestPasswordServer.
2. In the administrative console, click **Global security > Digest authentication > Custom Properties > New**, and enter `DigestPasswordServerClass` in the **Name** field, and the name of the class that you created in the **Value** field.
3. Ensure that all users that implement the impl class are declared in the user registry configured for WebSphere Application Server security.

LDAP servers automatically provide password support. Unless you enable the LDAP server to use hashed values, the LDAP server stores user passwords and then the request processing component uses these passwords to validate a request. Because this method of authentication exposes user passwords to potential internet theft, you should enable the use of hashed credentials to authenticate a request.

When you enable the use of hashed credentials, the LDAP server stores a hash value for the user, password and realm information. The SIP container then requests this hash value from the LDAP server instead of asking for a user password. This methodology protects the passwords even if the hash data is compromised through internet theft. However, this methodology has the following limitations:

- The LDAP attribute must store a byte value or a string value. Other attribute types are not supported.
- All of your applications must share the same realm, or you must define a different attribute for each realm.
- The hash function might be different than MD5. In this situation, the SIP container sends a algorithm that is different from the calculated value for the attribute. When this situation occurs, user authentication might fail even if the user provided the proper credentials.

To enable the LDAP server to use hashed credentials, you must define the following two custom properties:

- hashedCredentials=*value*, where *value* is the name of LDAP attribute which stores the hash value for user, password, realm
- hashedCredentialsRealms=*value*, where *value* is the realm, on which the hashed value is calculated.

About this task

The SIP container supports digest authentication. When this type of authentication is used, the client does not send a clear text password to the server. Instead, SIP authenticates each request using user data from

LDAP. Typically, a component that uses LDAP for authentication, verifies that the response that the client provides equals the response that the component calculates using LDAP data, the component authorizes the request. However,

Howto define: One should d

Complete the following procedure to configure digest authentication and TAI for the SIP container.

1. In the administrative console, click **Security** → **Global security** → **Authentication mechanisms** to verify that **Lightweight Third Party Authentication (LTPA)** is configured for use on your server.

   In the **Configuration** tab on the **Authentication mechanisms and expiration** page you should see the **Password** field already filled in.
2. Click **Security** → **Global security**.
   a. Under **Authentication**, expand **Web security** and click **Trust association**.
   b. On the **Configuration** tab, in the General properties section, verify that the **Enable trust association** box is selected, and then click **Apply**.
3. On the **Interceptors** page of the administration console look for `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** list.
   a. If this class name in not present, click **New** to open the Configuration tab and enter `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** field, and then click **Apply**.
   b. If this interceptor class is present, click **com.ibm.ws.sip.security.digest.DigestTAI** → **Custom Properties** to set up a realm in digest authentication.
   c. Click **OK**.
4. Click **Security** → **Global security** → **Authentication mechanisms and expiration**, and then click the **Configuration** tab.
   a. In the **Key generation** section, click **Generate keys**. You do not have to import or export the key.
   b. In the Cross-cell Single Sign-on section, specify values in the **Password** fields and the **Internal server ID** field.
   c. Click **OK**.
5. Click **Security** → **Global security**.
   a. If the box **Use Java 2 security to restrict application access to local resources** is selected, click to deselect it.
   b. In the **User account repository** section of the page, select your LDAP registry from the **Available realm definitions** list.
   c. Click **Set as current**, and then click**Apply**.
6. Save all changes.
7. Restart the server.
8. Verify that the following message appears in the SystemOut.log file after the server restarts:

   ```
   SECJ0121I: Trust Association Init class com.ibm.ws.sip.security.digest.DigestTAI loaded successfully
   ```

   If this message does not appear in the log file, digest authentication is not active

Configuring digest authentication for Oracle Internet Directory

You can configure digest authentication for Oracle Internet Directory, an implementation of the Lightweight Directory Access Protocol (LDAP) that uses the Oracle database as a repository for directory entries.

Before you begin

To configure digest authentication for Oracle Internet Directory, you will need to:

- Install Oracle Internet Directory version 9.0.2.

- Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.

About this task

Complete the following procedure to configure digest authentication for Oracle Internet Directory on WebSphere Application Server:

1. To set up digest authentication, verify that **Lightweight Third Party Authentication (LTPA)** is configured for use on your server by selecting **Security** → **Global security** → **Authentication mechanisms**. In the **Configuration** tab on the **Authentication mechanisms and expiration** page you should see the **Password** field already filled in.
2. In the administrative console, click **Security** → **Global security**.
   a. Under **Authentication**, expand **Web security** and click on **Trust association**.
   b. On the **Configuration** tab, under **General properties**, make sure the **Enable trust association** box is checked. Then click **Apply**.
3. On the **Interceptors** page of the administration console look for `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** list:
   a. If this class name in not present, click **New** to open the Configuration tab and enter `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** field and click **Apply**. Then proceed to the following steps.
   b. If this interceptor class is present, you may set up custom properties for it. To do this, click **com.ibm.ws.sip.security.digest.DigestTAI** → **Custom Properties**:
   c. Click **OK**.
4. Navigate through **Security** → **Global security** → **Authentication mechanisms and expiration** to the **Configuration** tab.
   a. In the **Key generation** section, click **Generate Keys**. (No import or export of the key is necessary.)
   b. Under the Cross-cell single sign-on section fill in the **Password** fields.
   c. Fill in the **Internal server ID** field.
   d. Click **OK**.
5. Click to **Security** → **Global security**.
   a. If the box **Use Java 2 security to restrict application access to local resources** is checked, then Java 2 security is enabled. Click the box if you want to disable Java 2 security.
   b. In the **User account repository** section of the page, select your LDAP registry from the **Available realm definitions** drop-down box.
   c. Click **Set as current** and then click**Apply**.
6. Save all changes.
7. Restart the server.
8. Be sure you see the following message appear in the SystemOut.log after the server has restarted:

   ```
   SECJ0121I: Trust Association Init class
    com.ibm.ws.sip.security.digest.DigestTAI loaded successfully
   ```

   If this message does not appear in the log, digest authentication has not been activated.

Chapter 3. EJB applications

Securing enterprise bean applications

You can protect enterprise bean methods by assigning security roles to them. Before you assign security roles, you need to know which Enterprise JavaBeans (EJB) methods need protecting and how to protect them.

About this task

You can assign a set of EJB methods to a set of roles. When an EJB method is secured by associating a set of roles, grant at least one role in that set so that you can access that method. To exclude a set of EJB methods from access, mark the set **excluded**. You can give everyone access to a set of enterprise beans methods by clearing those methods. You can run enterprise beans as a different identity, using the runAs identity, before invoking other enterprise beans.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To secure enterprise bean applications, follow these steps:

1. In an assembly tool, import your Enterprise JavaBean (EJB) Java Archive (JAR) file or an application archive (EAR) file that contains one or more Web modules.

   See the information about importing an EJB JAR file or importing an enterprise application EAR file in the Rational Application Developer documentation.
2. In the Project Explorer, click **EJB Projects** directory and click the name of your application.
3. Right-click the deployment descriptor and click **Open with > Deployment Descriptor Editor**. If you selected an enterprise bean `.jar` file, an EJB deployment descriptor editor opens. If you select an application `.ear` file, an application deployment descriptor editor opens. To see online information about the editor, press **F1** and click the editor name.
4. Create security roles. You can create security roles at the application level or at the EJB module level. If you create a security role at the EJB module level, the role displays in the application level. If a security role is created at the application level, the role does not display in all the EJB modules. You can copy and paste one or more EJB module security roles that you create at application level:
   - Create a role at an EJB module level. In an EJB deployment descriptor editor, click the **Assembly** tab. Under Security Roles, click **Add**. In the Add Security Role wizard, name and describe the security role and click **Finish**.
   - Create a role at the application level. In an application deployment descriptor editor, select the **Security** tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
5. Create method permissions. Method permissions map one or more methods to a set of roles. An enterprise bean has four types of methods: home methods, remote methods, LocalHome methods and local methods. You can add permissions to enterprise beans on the method level. You cannot add a method permission to an enterprise bean unless you already have one or more security roles defined. For Version 2.0 EJB projects, an unselected option specifies that the selected methods from the selected beans do not require authorization to run. To add a method permission to an enterprise bean:
   a. On the **Assembly** tab of an EJB deployment descriptor editor, under **Method Permissions**, click **Add**. The Add Method Permission wizard is opened.
   b. Select a security role from the list of roles found and click **Next**.
   c. Select one or more enterprise beans from the list of beans found. You can click **Select All** or **Deselect All** to select or clear all of the enterprise beans in the list. Click **Next**.

© Copyright IBM Corp. 2008

d. Select the methods that you want to bind to your security role. The Method elements page lists all the methods that are associated with the enterprise beans. You can click **Apply to All** or **Deselect All** to quickly select or clear multiple methods. The selection affects the default (*) method for each bean only. Creating a method permission for the exact method signature overrides the default (*) method permission setting. The default (*) method represents all the methods within the bean. There are default (*) methods for each interface as well. By not selecting all of the individual methods in the tree, you can set other permissions on the remaining methods.

e. Click **Finish**.

After the method permission is created, you can see the new method permission in the tree. Expand the tree to see the bean and the methods that are defined in the method permission.

6. Exclude user access to methods. Users cannot access excluded methods. Any method in the enterprise beans that is not assigned to a role or that is not excluded, is cleared during the application installation by the deployer.
   a. On the **Assembly** tab of an EJB deployment descriptor editor, under **Excludes List**, click **Add**. The Exclude List wizard is opened.
   b. Select one or more enterprise beans from the list of beans found and click **Next**.
   c. Select one or more of the method elements for the security identity and click **Finish**.
7. Map the security-role-ref and role-name to the role-link. When developing enterprise beans, you can create the security-role-ref element. The security-role-ref element contains only the role-name field. The role-name field determines if the caller is in a specified role(isCallerInRole()) role and contains the name of the role that is referenced in the code. Because you create security roles during the assembly stage, the developer uses a *logical role name* in the **role-name** field and provides enough information in the **Description** field for the assembler to map the actual role (role-link). The security-role-ref element is located at the EJB level. Enterprise beans can have zero or more security-role-ref elements.
   a. On the **Reference** tab of an EJB deployment descriptor editor, under the list of references, click **Add**. The Add Reference wizard is opened.
   b. Select **Security role reference** and click **Next**.
   c. Name the security role reference, select a security role to link the reference to, describe the security role reference, and click **Finish**.
   d. Map every role-name that is used during development to the role (role-link) using the previous steps.
8. Specify the RunAs identity for enterprise bean components. The RunAs identity of the enterprise bean is used to invoke the next enterprise beans in the chain of EJB invocations. When the next enterprise beans are invoked, the RunAsIdentity identity passes to the next enterprise beans for performing an authorization check on the next enterprise bean. If the RunAs identity is not specified, the client identity is propagated to the next enterprise bean. The RunAs identity can represent each of the enterprise beans or can represent each method in the enterprise beans.
   a. On the **Access** tab of an EJB deployment descriptor editor, next to the **Security Identity (Bean Level)** field, click **Add**. The Add Security Identity wizard is opened.
   b. Select the appropriate run as mode, describe the security identity, and click **Next**. Select the **Use identity of caller** mode to instruct the security service to not make changes to the credential settings for the principal. Select the **Use identity assigned to specific role (below)** mode to use a principal that is assigned to the specified security role for running the bean methods. This association is part of the application binding in which the role is associated with the user ID and password of a user who is granted that role. If you select the **Use identity assigned to specific role (below)** mode , you must specify a role name and role description.
   c. Select one or more enterprise beans from the list of beans found and click **Next**. If Next is unavailable, click **Finish**.
   d. Optional: On the Method elements page, select one or more of the method elements for the security identity and click **Finish**.
9. Close the deployment descriptor editor and, when prompted, click **Yes** to save the changes.

Results

After securing an EJB application, the resulting `.jar` file contains security information in its deployment descriptor. The security information of the EJB modules is stored in the `ejb-jar.xml` file.

What to do next

After securing an EJB application using an assembly tool, you can install the EJB application using the administrative console. During the installation of a secured EJB application, follow the steps in the “Deploying secured applications” on page 17 article to complete the task of securing the EJB application.

Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for Enterprise JavaBeans (EJB) Version 2.1 message-driven beans.

About this task

The association between connection factories, destinations, and message-driven beans is provided by listener ports. A listener port allows a deployed message-driven bean associated with the port to retrieve messages from the associated destination. You create listener ports by specifying their administrative name, the connection factory JNDI name, and the destination name (other optional properties are also configurable). Listener ports provide simplified administration of the associations between connection factories, destinations and message-driven beans, and are managed by a listener manager. The listener manager is provided by the message listener service to control and monitor the JMS listeners that are monitoring JMS destinations on behalf of deployed message-driven beans. For more information about listener ports, see Message-driven beans - listener port components

Messages handled by message-driven beans have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

Connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define an authentication alias on the J2C activation specification that the message-driven bean is configured with. If defined, the message-driven bean uses the authentication alias for its JMSConnection security credentials instead of any application-managed alias.

To set the authentication alias, you can use the administrative console to complete the following steps. This task description assumes that you have already created an activation specification. If you want to create a new activation specification, see the related tasks.

- For a message-driven bean listening on a JMS destination of the default messaging provider, set the authentication alias on a JMS activation specification.
  1. To display the JMS activation specification settings, click **Resources** → **JMS** → **JMS providers**
  2. In the content pane, click the name of a default messaging provider.
  3. In the content pane, under Additional Resources, click **Activation specifications**.
  4. If you have already created a JMS activation specification, click its name in the list displayed. Otherwise, click **New** to create a new JMS activation specification.

5. Set the **Authentication alias** property.
6. Click **OK**
7. Save your changes to the master configuration.

- For a message-driven bean listening on a destination (or endpoint) of another Java EE Connector Architecture (JCA) provider, set the authentication alias on a J2C activation specification.
  1. To display the J2C activation specification settings, click **Resources** → **Resource Adapters** → **adapter_name** → **J2C Activation specifications** → **activation specification_name**
  2. Set the **Authentication alias** property.
  3. Click **OK**
  4. Save your changes to the master configuration.

Chapter 4. Client applications

Accessing secure resources using SSL and applet clients

By default, the applet client is configured to have security enabled. If you have administrative security turned on at the server from which you are accessing resources, then you can use secure sockets layer (SSL) when needed.

About this task

If you decide that the security requirements for the applet differ from other application client types, then create a new version of the `sas.client.props` and `ssl.client.props` files.

1. Make a copy of the following files so that you can use them for an applet:
   - `<app_client_root>\properties\sas.client.props`
   - `<app_client_root>\properties\ssl.client.props`
2. Edit the copies of the `sas.client.props` and `ssl.client.props` files that you made with your changes.
3. Click **Start** > **Control panel** > select the product Java plug-in to open the Java control panel. To use the files you created in step 1, modify the following values:
   - `-Dcom.ibm.CORBA.ConfigURL=file:<app_client_root>\properties\sas.client.props`
   - `-Dcom.ibm.SSL.ConfigURL=file:<app_client_root>\properties\ssl.client.props`

   For more information on the `sas.client.props` and `ssl.client.props` files and WebSphere Application Server security, see the Security section of the information center.

Applet client security requirements

When code is loaded, it is assigned permissions based on the security policy in effect. This policy specifies the permissions that are available for code from various locations. You can initialize this policy from an external policy file.

By default, the client uses the `<app_server_root>/properties/client.policy` file. You must update this file with the following permission:

SocketPermission grants permission to open a port and make a connection to a host machine, which is your WebSphere Application Server. In the following example, `yourserver.yourcompany.com` is the complete host name of your WebSphere Application Server:

```
permission java.util.PropertyPermission "*", "read";
permission java.net.SocketPermission "yourserver.yourcompany.com ,"connect";
```

© Copyright IBM Corp. 2008

Chapter 5. Web services

Configuring a Web services client to access resources using a Web proxy

You can configure a Web services client to access resources through a Web proxy server.

About this task

You can configure a Web services client to access resources by connecting to a Web proxy server either with or without requiring authentication, just like other HTTP client applications. You can configure HTTP transport properties for a Web service acting as a client to another Web service. The HTTP transport values you configure are used at runtime. Configure the HTTP transport values in one of the following ways:

- Configure the properties using the Java Virtual Machine (JVM) custom property panel in the administrative console.
- Configure the properties using the **wsadmin** command-line tool.
- Configure the properties with an assembly tool.
- Configure the properties programmatically using the application programming model

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model or the Java API for XML Web Services (JAX-WS) programming model, review the JAX-RPC or JAX-WS specifications.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

For Java API XML-based Remote Procedure Call (JAX-RPC) Web services, the HTTP transport values take the following precedence order with the programmatic method being the most significant:

1. values specified programmatically on the Call object
2. values defined in the deployment descriptors in each `portQNameBinding` attribute using an assembly tool
3. values defined as JVM system properties

For Java API for XML Web Services (JAX-WS) Web services, the HTTP transport values you specify in your policy set definitions take precedence over the values defined programmatically. Subsequently, the HTTP transport values you define programmatically take precedence over the values defined as JVM system properties. For JAX-WS applications, deployment descriptors are not supported. Use annotations to specify deployment information.

1. Configure the HTTP or HTTPS `proxyHost` and `proxyPort` transport properties for the Web services in one of the following ways:
   - using the Java Virtual Machine (JVM) custom property panel in the administrative console
   - using the **wsadmin** command-line tool
   - using assembly tools
   - programmatically using the application programming model

   To access the Web proxy over HTTP:
   - **http.proxyHost**
   - **http.proxyPort**

   To access the Web proxy over HTTPS:
   - **https.proxyHost**

© Copyright IBM Corp. 2008

- **https.proxyPort**

2. If HTTP proxy authentication is required for your Web services client, then additionally configure the HTTP or HTTPS `proxyUser` and `proxyPassword` transport properties using one of the methods specified in the previous step.

   To access the Web proxy over HTTP:

   - **http.proxyUser**
   - **http.proxyPassword**

   To access the Web proxy over HTTPS:

   - **https.proxyUser**
   - **https.proxyPassword**

3. If you are specifying the HTTP or HTTPS properties programmatically, set the properties in the Stub or Call instance to configure the HTTP proxy authentication.

   a. You can set the HTTP or HTTPS properties programmatically using the following Web services constants:

   ```
   com.ibm.wsspi.webservices.Constants.HTTP_PROXYHOST_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTP_PROXYPORT_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTP_PROXYUSER_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTP_PROXYPASSWORD_PROPERTY

   com.ibm.wsspi.webservices.Constants.HTTPS_PROXYHOST_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTPS_PROXYPORT_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTPS_PROXYUSER_PROPERTY
   com.ibm.wsspi.webservices.Constants.HTTPS_PROXYPASSWORD_PROPERTY
   ```

Results

You have configured your Web services client to use a Web proxy server to access resources.

You can optionally set the `http.nonProxyHosts` property to specify the host names of machines to which requests will not be sent through the proxy server. Any requests invoked by the client application that are sent to a host whose name is contained in this property will not pass through the proxy server. This property applies for both HTTP and HTTPS connections. To learn more about the `http.nonProxyHosts` property and other HTTP properties that you can configure, read about HTTP transport custom properties for Web services applications.

Example

Configuring the HTTP proxy programmatically

The following code allows you to configure the HTTP proxy programmatically:

```
import com.ibm.wsspi.webservices.Constants
Properties prop = new Properties();
InitialContext ctx = new InitialContext(prop);
Service service = (Service)ctx.lookup("java:comp/env/service/StockQuoteService");
QName portQname = new QName("http://httpchannel.test.wsfvt.ws.ibm.com", "StockQuoteHttp");
StockQuote sq = (StockQuote)service.getPort(portQname, StockQuote.class);
((javax.xml.rpc.Stub) sq)._setProperty(Constants.HTTP_PROXYHOST_PROPERTY, "proxyHost1.ibm.com");
((javax.xml.rpc.Stub) sq)._setProperty(Constants.HTTP_PROXYPORT_PROPERTY, "80");
```

Provide HTTP endpoint URL information

Use this page to specify endpoint URL prefix information for Web services accessed by HTTP. Prefixes are used to form complete endpoint addresses included in published Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Provide HTTP endpoint URL information**.

You can specify a portion of the endpoint URL to be used in each Web service module. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's `soap:address` element.

This administrative console panel applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

Specify endpoint URL prefixes for Web services

Specifies the *protocol* (either http or https), *host_name*, and *port_number* to be used in the endpoint URL.

You can select a prefix from a predefined list using the **HTTP URL prefix** or **Custom HTTP URL prefix** field.

The URL prefix format is *protocol://host_name:port_number*, for example, http://*myHost*:9045. The actual endpoint URL that is contained in a published WSDL file consists of the prefix followed by the module's context-root and the Web service url-pattern, for example, http://*myHost*:9045/services/*myService*.

Select default HTTP URL prefix

Specifies the drop down list associated with a default list of URL prefixes. This list is the intersection of the set of ports for the module's virtual host and the set of ports for the module's application server. Use items from this list if the Web services application server is accessed directly.

To set an HTTP endpoint URL prefix, select **Select default HTTP URL prefix** and select a value from the drop down list. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP URL prefix** field of any module whose check box is selected.

Select custom HTTP URL prefix

Specifies the *protocol*, *host*, and *port_number* of the intermediate service if the Web services in a module are accessed through an intermediate node, for example the Web services gateway or an IHS server.

To set a custom HTTP endpoint URL prefix, you must also configure the custom JVM property, `com.ibm.ws.webservices.enableHTTPPrefix` in the administrative console and set the value to `true`. Setting this custom JVM property is required so the custom HTTP URL is correctly populated in the URL field of the WSDL file that is returned to the client. If this custom JVM property is not configured, the custom HTTP URL prefix is not in the URL field in the copy of the WSDL file that the service returns to the client. To learn how to configure this custom JVM property, see the documentation on configuring additional HTTP transport properties using the JVM custom property panel in the administrative console. You must restart the application server after this custom property has been defined so that this property is used by the system.

After the `com.ibm.ws.webservices.enableHTTPPrefix` custom JVM property is configured, select **Select custom HTTP URL prefix** and enter a value. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP endpoint URL prefix** field of any module whose check box is selected.

Securing Web services applications at the transport level

Transport-level security is a well-known and often used mechanism to secure HTTP Internet and intranet communications. Transport level security can be used to secure Web services messages. Transport-level security functionality is independent from functionality that is provided by message-level security (WS-Security) or HTTP basic authentication.

Before you begin

You can use either message-level security (WS-Security) or transport-level security:

- Use message-level security when security is essential to the Web service application. HTTP basic authentication uses a user name and password to authenticate a service client to a secure endpoint. The basic authentication is encoded in the HTTP request that carries the SOAP message. When the application server receives the HTTP request, the user name and password are retrieved and verified using the authentication mechanism specific to the server.
- Use transport-level security to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Transport-level security provides minimal security. You can use this configuration when a Web service is a client to another Web service.

About this task

Transport-level security is based on Secure Sockets Layer (SSL) or Transport Layer Security (TLS) that runs beneath HTTP. HTTP, the most used Internet communication protocol, is currently also the most popular protocol for Web services. HTTP is an inherently insecure protocol because all information is sent in clear text between unauthenticated peers over an insecure network. To secure HTTP, transport-level security can be applied.

Transport level security can be used to secure Web services messages. However, transport-level security functionality is independent from functionality that is provided by WS-Security or HTTP Basic Authentication.

SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections. To run with HTTPS, the service port address must be in the form `https://`. The integrity and confidentiality of transport data, including SOAP messages and HTTP basic authentication, is confirmed when you use SSL and TLS.

WebSphere Application Server uses the Java Secure Sockets Extension (JSSE) package to support SSL and TLS.

This task is one of several ways that you can configure the HTTP outbound transport level security for a Web service acting as a client to another Web service server. You can also configure the HTTP outbound transport level security with an assembly tool or by using the Java properties. If you do not configure the HTTP outbound transport level security, the Web services runtime defers to the Java Platform, Enterprise Edition (Java EE) security runtime in the WebSphere product for an effective Secure Sockets Layer (SSL) configuration. If there is no SSL configuration with the Java EE security runtime in the WebSphere product, the Java Secure Socket Extension (JSSE) system properties are used.

You can define additional HTTP transport properties for Web services applications. Use the additional properties to manage the connection pool for HTTP outbound connections, configure the content encoding of the HTTP message, enable HTTP persistent connection, and resend the HTTP request when a timeout occurs.

- Select one of the following methods to configure HTTP outbound transport level security. There are three ways that you can configure HTTP outbound transport level security:
  - Configure HTTP outbound transport level security by using the administrative console.
  - Configure HTTP outbound transport level security by using an assembly tool.
  - Configure HTTP outbound transport-level security by using Java properties.
  -
- Select one of the following methods to define additional HTTP transport properties for Web services applications.
  - Configure additional HTTP transport properties by using the JVM custom property panel in the administrative console.

– Configure additional HTTP transport properties by using an assembly tool.

Results

By completing these steps, you have secured Web services applications at the transport level.

What to do next

HTTP transport custom properties for Web services applications

Use HTTP transport properties for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services to manage the connection pool for HTTP outbound connections, configure the content encoding of the HTTP message, enable HTTP persistent connection, and resend the HTTP request when a timeout occurs.

Establishing a connection is an expensive operation. Connection pooling improves performance by avoiding the overhead of creating and disconnecting connections. When an application invokes a Web service over an HTTP transport, the HTTP outbound connector for the Web service locates and uses an existing connection from a pool of connections. When the response is received, the connector returns the connection to the connection pool for reuse. The overhead to create and disconnect the connection is avoided.

com.ibm.websphere.webservices.http.connectionTimeout

This property specifies the interval, in seconds, that a connection request times-out and the `WebServicesFault( "Connection timed out" )` error occurs.

The value affects all of the HTTP connection requests made by the HTTP outbound connector. The wait time is needed when the maximum number of connections in the connection pool is reached. For example, if the property is set to 300 and the maximum number of connections is reached, the connector waits for 300 seconds until a connection is available. After 300 seconds, the `WebServicesFault( "Connection timed out" )` error occurs if a connection is not available. If the property is set to 0 (zero), the connector waits until a connection is available.

If the `WebServicesFault( "Connection timed out")` error occurs in the application, set the com.ibm.websphere.webservices.http.connectionTimeout property value higher. Also, review the application usage. If the com.ibm.websphere.webservices.http.maxConnection property value is set to 0 (zero), and is enabled for an unlimited number of connections, the com.ibm.websphere.webservices.http.connectionTimeout property value is ignored.

Data type	Integer
Units	Seconds
Default	300
Range	0 (zero) to the maximum integer

For information about how to configure these properties see Configuring additional HTTP transport properties using the administrative console.

Note: This property only can be configured as a JVM custom property that manages the connection pool for HTTP outbound connections for Web services applications.

com.ibm.websphere.webservices.http.maxConnection

This property specifies the maximum number of connections that are created in the HTTP outbound connector connection pool.

It affects all of the Web services HTTP connections that are made within one JVM. When the maximum number of connections is reached, no new connection are created and the HTTP connector waits for a current connection to return to the connection pool. If the HTTP connector does not wait for a current connection because of a connection request timeout, the `WebServicesFault( "Connection timed out" )` error occurs. For example, if the property is set to 5, and there are 5 connections in use, the HTTP connector waits for the specified time set in the com.ibm.websphere.webservices.http.connectionTimeout property for a connection to become available.

Note: For performance reasons, it is a best practice to set the maximum size of the com.ibm.websphere.webservices.http.maxConnection property to less than or equal to half of the size of Web container threadpool. The default size for the Web container threadpool is 50. As a result, the default size of the com.ibm.websphere.webservices.http.maxConnection property is set to 25 by the Web services runtime.

Data type	Integer
Default	25
Range	5 to the maximum integer, which is less than or equal to half of the size of Web container threadpool

For information about how to configure these properties see Configuring additional HTTP transport properties using the administrative console.

Note: This property only can be configured as a JVM custom property that manages the connection pool for HTTP outbound connections for Web services applications.

com.ibm.websphere.webservices.http.connectionPoolCleanUpTime

This property specifies the interval, in seconds, between runs of the connection pool maintenance thread.

You can configure the property only as a JVM custom property. This property affects all HTTP connections for Web Services made within one JVM. For example, if the property is set to 180, the pool maintenance thread runs every 180 seconds. When the pool maintenance thread runs, the connector discards any connections in the clean up queue.

Data type	Integer
Units	Seconds
Default	180
Range	0 (zero) to the maximum integer

For information about how to configure these properties see Configuring additional HTTP transport properties using the administrative console.

Note: This property only can be configured as a JVM custom property that manages the connection pool for HTTP outbound connections for Web services applications.

com.ibm.websphere.webservices.http.connectionIdleTimeout

This property specifies the interval, in seconds, after an idle connection is discarded.

The connection is added to the clean up queue only after a new connection is formed. You can configure the property only as a JVM custom property. For example, if the property is set to 120, the pool maintenance thread discards any connection that remains idle for 2 minutes. This property affects all Web services HTTP connections made within one JVM.

Data type	Integer

Units	Seconds
Default	5
Range	0 (zero) to the maximum integer

For information about how to configure these properties see Configuring additional HTTP transport properties using the administrative console.

Note: This property only can be configured as a JVM custom property that manages the connection pool for HTTP outbound connections for Web services applications.

com.ibm.websphere.webservices.http.SocketTimeout

This property specifies the amount of time, in seconds, to wait for the outbound socket to be established with the remote server.

You can configure the property only as a JVM custom property. This property affects all Web services HTTP connections made within one JVM. If an invalid value is provided, the default value overrides the invalid value.

Data type	Integer
Units	Seconds
Default	180
Range	0 (zero) to the maximum integer

For information about how to configure these properties see Configuring additional HTTP transport properties using the administrative console.

Note: This property only can be configured as a JVM custom property that manages the connection pool for HTTP outbound connections for Web services applications.

com.ibm.websphere.webservices.http.requestContentEncoding

This property specifies the type of encoding to use in the message of each HTTP outbound request. It is an HTTP transport property you can configure for Web services applications.

Supported encoding formats follow the HTTP 1.1 protocol specification including gzip, x-gzip, and deflate. If this property is configured, the headers ″Content-Encoding″ and ″Accept-Encoding″ in the HTTP request are also set to the same value. For example, if the property is set to gzip, the headers become Content-Encoding: gzip and Accept-Encoding: gzip. However, if the property is not set, the HTTP request message is not encoded. The default is no encoding.

You should check if the target Web server is capable of decoding the configured coding format. For example, if the property is set to gzip, the target Web server must also support the gzip encoding. Otherwise, a failure can occur and a status code of `415 Unsupported Media Type` might display.

The compress encoding format is not supported and x-gzip encoding is equivalent to gzip encoding.

Data type	String
Valid values	gzip, x-gzip, and deflate

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

com.ibm.websphere.webservices.http.responseContentEncoding

This property specifies the type of encoding to be used in the message of each HTTP response. It is an HTTP transport property you can configure for Web services applications.

Supported encoding formats follow the HTTP 1.1 protocol specification including gzip, x-gzip, and deflate. If this property is configured, the headers ″Content-Encoding″ in the HTTP response is set to the same value. If the property is not set, the HTTP response message content is not encoded. The default value is `no encoding`.

If the property is set, the request client must also support the same encoding. Otherwise, a failure can occur and a `WebServicesFault()` error displays.

The compress encoding format is not supported and x-gzip encoding is equivalent to gzip encoding.

Data type	String
Valid values	gzip, x-gzip, or deflate

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

com.ibm.websphere.webservices.http.connectionKeepAlive

This property specifies whether the connector should maintain a live or persistent HTTP connection. It is an HTTP transport property you can configure for Web services applications.

If the property is set to `true`, the connector keeps the connection in the connection pool and reuses the connection for subsequent HTTP requests. However, the connection is closed if syncTimeout(Read timeout) is reached or the server has dropped the connection. Also, an idle connection is closed by the pool maintenance thread if the idle time has passed the connection idle time-out. If the property is set to `false`, the connection is closed after the HTTP request is sent. If a new request is ready to send and the connection does not exist, the HTTP connector creates one.

Data type	String
Default	True
Valid values	True, false

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

com.ibm.websphere.webservices.http.requestResendEnabled

This property tells the HTTP connector to resend the SOAP message over HTTP request after a `java.net.ConnectException: read timed out` error is logged. It is an HTTP transport property you can configure for Web services applications.

This property tells the HTTP connector to resend the SOAP message over HTTP request after a `java.net.ConnectException: read timed out` error is logged. The `java.net.ConnectException` is caused by a socket time-out, or when a server shuts down while the request is being sent. If the property is enabled, the connector tries to reconnect one time only and resends the same SOAP message over HTTP. Otherwise, the connector stops sending the SOAP message and a `WebServicesFault` error is logged.

Problems can occur with the application this property is enabled. The HTTP request that is resent can be received twice by the server and can cause an unexpected result.

Data type	String
Default	False
Valid values	True, false

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

com.ibm.ws.webservices.enableHTTPPrefix

This property specifies whether the `hosthame:port` value that is defined as a custom HTTP URL prefix in the Provide HTTP endpoint URL panel in the administrative console is populated to the URL field in the copy of the WSDL file that is returned from the service to the client. It is an HTTP transport property you can configure for Web services applications.

Configure this property with the value of `true` so the specified custom HTTP URL prefix is correctly specified in the WSDL file that is returned to the client. If this property is not configured with the value of `true`, then the specified custom `hosthame:port` of the server node that responds to the request is not populated in the URL field in the copy of the WSDL file that is returned to the client.

You must restart the application server after this custom property has been defined so that this property is used by the system.

Data type	String
Default	False
Valid values	True, false

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

http.proxyHost

desc

This property specifies the host name of an HTTP proxy. It is an HTTP transport property you can configure for Web services applications.

Data type	String

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

http.proxyPort

This property specifies the port of an HTTP proxy. It is an HTTP transport property you can configure for Web services applications.

Data type	String

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

https.proxyHost

This property specifies the host name of an HTTPS proxy. It is an HTTP transport property you can configure for Web services applications.

Data type	String

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

https.proxyPort

This property specifies the port of an HTTPS proxy. It is an HTTP transport property you can configure for Web services applications.

Data type	String

For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

http.nonProxyHosts

This JVM system property acts as an override to the `http.proxyHost` and `https.proxyHost` properties and specifies the host names of machines to which requests will not be sent through the proxy server. It is an HTTP transport property you can configure for Web services applications.

Any requests invoked by the client application that are sent to a host whose name is contained in this property will not pass through the proxy server. Separate each host name in the list with a vertical bar (″|″). You can optionally use an asterisk (*) as a wildcard character.

The `http.nonProxyHosts` property applies for both HTTP and HTTPS connections.

In the following example,

```
http.proxyHost="myproxy.mycompany.com"
http.nonProxyHosts="host1.company1.com|host*.company2.com|*.company3.com"
```

all requests will be routed through the proxy server, *myproxy.mycompany.com*, except for the HTTP requests destined for the following hosts:

- a single host named host1.company1.com
- any host in the company2.com domain whose name starts with *host*
- any host in the company3.com domain

Note: When using a Web services client through a web proxy, it is a best practice to set the `http.nonProxyHosts` property to include the local host if any Web services are hosted on the same system. For example, if the local host is named myHost.myCorp.com, then set the `http.nonProxyHosts` property to `localhost|myHost.myCorp.com` or `localhost|*.myCorp.com`. If you do not set the `http.nonProxyHosts` property to include the local host, then Web services requests made to the local host will go out to the Web proxy and then return back to the local host.

Data type	String

You can only configure this property as a JVM custom property. For information about how to configure this property, see Configuring additional HTTP transport properties using wsadmin, and Configuring additional HTTP transport properties using an assembly tool.

Configuring HTTP outbound transport level security with the administrative console

You can configure HTTP outbound transport level security with the administrative console.

Before you begin

This task is one of several ways that you can configure the HTTP outbound transport level security for a Web service acting as a client to another Web service server. You can also configure the HTTP outbound transport level security with an assembly tool or by using the Java properties. If you do not configure the HTTP outbound transport level security, the Web services runtime defers to the Web Services for Java Platform, Enterprise Edition (Java EE) security runtime in the WebSphere product for an effective Secure Sockets Layer (SSL) configuration. If there is no SSL configuration with the Java EE security runtime in the WebSphere product, the Java Secure Socket Extension (JSSE) system properties are used.

About this task

If you choose to configure the HTTP outbound transport level security with the administrative console or an assembly tool, the Web services security binding information is modified. You can use the administrative console to configure the Web services client security bindings if you have deployed or installed the Web services application into WebSphere Application Server. If you have not installed the Web services application, you can configure the HTTP SSL configuration with an assembly tool. This task assumes that you have deployed the Web services application into the WebSphere product.

If you configure the HTTP outbound transport level security using the standard Java properties for JSSE, the properties are configured as system properties. The configuration specified in the binding takes precedence over the Java properties. However, the configurations that are specified by the Java EE security programming model , or that are associated the Dynamic selection , have higher precedence.

Review the topic Secure communications using Secure Sockets Layer for more information.

Configure the HTTP outbound transport level security with the following steps provided in this task section.

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* . Under Web Services Security Properties, click **Web Services: Client security bindings**.
3. Under the heading, HTTP SSL Configuration, click **Edit** to access the HTTP SSL configuration panel. Select the **Centrally-managed** radio button so that the system runtime chooses the SSL configuration that is based on the current context. Select the **Specific to this Web service port** radio button if you want to choose the SSL configuration in the HTTP SSL configuration drop down box.

Results

You have configured the HTTP outbound transport level security for a Web service acting as a client to another Web service with the administrative console.

HTTP SSL Configuration collection

Use this page to configure transport-level Secure Sockets Layer (SSL) security. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable HTTP SSL (or HTTPS). Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name*** .
2. Click **Manage modules > *URI_file_name* > Web Services: Client Security Bindings** .
3. Under HTTP SSL Configuration, click **Edit**.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

SSL configuration: Select the **Centrally-managed** radio button so that the system runtime chooses the SSL configuration that is based on the current context. Select the **Specific to this Web service port** radio button if you want to choose the SSL configuration in the **HTTP SSL configuration** drop down box.

HTTP SSL configuration: The **HTTP SSL configuration** drop down box lists the SSL configurations used with the HTTP transport for a port. Use this drop down box if you want to select the SSL configuration rather than using the SSL configuration that the runtime automatically selects. To use the drop down box, select the **Specific to the Web service port** radio button that is located in the **SSL configuration** field. After you select the radio button, you can click the drop down box to view and select an SSL configuration.

Configuring HTTP outbound transport level security using Java properties

You can configure the HTTP outbound transport level security for a Web service using Java properties

Before you begin

This task is one of three ways that you can configure HTTP outbound transport-level security for a Web service that is acting as a client to another Web service. You can also configure the HTTP outbound transport level security with the administrative console or an assembly tool. However, you can also use this task to configure the HTTP outbound transport-level security for a Web service client.

About this task

If you choose to configure the HTTP outbound transport-level security with the administrative console or an assembly tool, the Web services security binding information is modified.

If you configure the HTTP outbound transport-level security using Java properties, the properties are configured as system properties. However, the configuration specified in the binding takes precedence over the Java properties.

You can configure the HTTP outbound transport-level security using WebSphere SSL properties or JSSE SSL properties. However, the WebSphere SSL properties take precedence over the JSSE SSL properties.

Configure the HTTP outbound transport-level security with the following steps provided in this task section.

1. Create a property file that includes the following properties:

   ```
   com.ibm.ssl.protocol
   com.ibm.ssl.keyStoreType
   com.ibm.ssl.keyStore
   com.ibm.ssl.keyStorePassword
   com.ibm.ssl.trustStoreType
   com.ibm.ssl.trustStore
   com.ibm.ssl.trustStorePassword
   ```

2. Set the `com.ibm.webservices.sslConfigURL` Java system property to the absolute path of the created property file. If no WebSphere SSL properties are defined, the JSSE SSL properties are used. Set the JSSE SSL properties as JVM custom properties. See Secure transports with JSSE and JCE programming interfaces for more information about setting the JSSE SSL properties.

Results

You have configured the HTTP outbound transport-level security for a Web service acting as a client to another Web service.

Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console

You can configure additional HTTP transport properties for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services with the JVM custom properties panel in the administrative console.

About this task

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configure the properties with an assembly tool
- Configure the properties using the **wsadmin** command-line tool

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model or the Java API for XML Web Services (JAX-WS) programming model, review the JAX-RPC or JAX-WS specifications.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

For more information about the following HTTP properties that you can configure, read about HTTP custom properties for Web services applications:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding
- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- com.ibm.websphere.webservices.http.SocketTimeout
- com.ibm.ws.webservices.enableHTTPPrefix
- http.proxyHost
- http.proxyPort
- https.proxyHost
- https.proxyPort
- http.nonProxyHosts - You can only configure this property as a JVM custom property. This property applies for both HTTP and HTTPS connections.

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

1. Open the administrative console.
   a. Click **Servers** > **Application Servers** > *server* > **Java and Process Management** > **Process Definition** > **Java Virtual Machine** > **Custom Properties**.
2. (Optional) If the property is not listed, create a new property name.
3. Enter the name and value.
4. (Optional) Accept the redirection of the HTTP request to a different URI in HTTPS.

   A redirection of the HTTP request to a different URI in HTTPS can occur if the transport guarantee of CONFIDENTIAL or INTEGRAL is configured in the application. To accept the redirection, you can do either of the following tasks:

   - Set the `com.ibm.ws.webservices.HttpRedirectEnabled` Java system property to `true`.

- Programmatically set the `com.ibm.wsspi.webservices.Constants.HTTP_REDIRECT_ENABLED` property to a java.lang.Boolean object in the Stub or Call object before invoking the service. For example, use any of the following java.lang.Boolean values to set the property to `true`:
  - `Boolean.TRUE`
  - `new Boolean(true)`
  - `new Boolean("true")`

Results

You have configured HTTP transport properties for a Web services application.

Configuring additional HTTP transport properties using the wsadmin command-line tool

You can configure additional HTTP transport properties for Java API for XML-based RPC (JAX-RPC) Web services with the **wsadmin** command-line tool.

Before you begin

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages. For more information about the wsadmin tool options, review Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands

About this task

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configure the properties with an assembly tool
- Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model or the Java API for XML Web Services (JAX-WS) programming model, review the JAX-RPC or JAX-WS specifications. For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

For more information about the following HTTP properties that you can configure, read about HTTP custom properties for Web services applications:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding
- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- com.ibm.websphere.webservices.http.SocketTimeout
- com.ibm.ws.webservices.enableHTTPPrefix
- http.proxyHost
- http.proxyPort
- https.proxyHost
- https.proxyPort

- http.nonProxyHosts - You can only configure this property as a JVM custom property. This property applies for both HTTP and HTTPS connections.

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

Configure the additional HTTP properties with the wsadmin tool by following steps provided in this task section:

1. Launch a scripting command.
2. At the **wsadmin** command prompt, enter the command syntax. You can use install, installInteractive, edit, editInteractive, update, and updateInteractive commands.
3. If you are configuring the `com.ibm.websphere.webservices.http.responseContentEncoding` property, use the **WebServicesServerCustomProperty** command option.
4. Configure all other properties using the **WebServicesClientCustomProperty** command option.
5. Save the configuration changes with the **$AdminConfig save** command.

Results

You have configured HTTP transport properties for a Web services application.

Example

The following illustrates an example of the Jython script syntax:

```
AdminApp.edit ( 'PlantsByWebSphere', '[ -WebServicesClientCustomProperty [[PlantsByWebSphere.war ""
service/FrontGate_SEIService FrontGate http.proxyHost+http.proxyPort myhost+80]]]')
AdminConfig.save()

AdminApp.edit ( 'WebServicesSamples', '[ -WebServicesServerCustomProperty
[[AddressBookW2JE.jarAddressBookService AddressBook http.proxyHost+http.proxyPort myhost+80]]]')
AdminConfig.save()
```

The following illustrates an example of the Jacly script syntax:

```
$AdminApp edit PlantsByWebSphere {  -WebServicesClientCustomProperty {{PlantsByWebSphere.war {}
service/FrontGate_SEIService FrontGate  http.proxyHost+http.proxyPort myhost+80 }}}
$AdminConfig save

$AdminApp edit WebServicesSamples {-WebServicesServerCustomProperty {{AddressBookW2JE.jar
AddressBookService AddressBook http.proxyHost+http.proxyPort myhost+80}}}
$AdminConfig save
```

To convert these examples from **edit** to **install**, add `.ear` to form a file name, and add any extra keywords for deployment, like `-usedefaultbindings` and `-deployejb`.

Related tasks

“Configuring additional HTTP transport properties for JAX-RPC Web services with an assembly tool”
This topic explains how to configure additional HTTP transport properties for Java API for XML-based RPC (JAX-RPC) Web services with an assembly tool. The assembly tool is used to configure the `ibm-webservicesclient-bnd.xmi` deployment descriptor binding file.

“Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console” on page 41
You can configure additional HTTP transport properties for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services with the JVM custom properties panel in the administrative console.

“Configuring a Web services client to access resources using a Web proxy” on page 29
You can configure a Web services client to access resources through a Web proxy server.

Related reference

“HTTP transport custom properties for Web services applications” on page 33
Use HTTP transport properties for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services to manage the connection pool for HTTP outbound connections, configure the content encoding of the HTTP message, enable HTTP persistent connection, and resend the HTTP request when a timeout occurs.

Web services specifications and APIs

Configuring additional HTTP transport properties for JAX-RPC Web services with an assembly tool

This topic explains how to configure additional HTTP transport properties for Java API for XML-based RPC (JAX-RPC) Web services with an assembly tool. The assembly tool is used to configure the `ibm-webservicesclient-bnd.xmi` deployment descriptor binding file.

Before you begin

You can configure additional HTTP transport properties with assembly tools provided with WebSphere Application Server.

About this task

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console
- Configure the properties using the **wsadmin** command-line tool.

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model or the Java API for XML Web Services (JAX-WS) programming model, review the JAX-RPC or JAX-WS specifications.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

For more information about the following HTTP properties that you can configure, read about HTTP custom properties for Web services applications:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding

- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- com.ibm.websphere.webservices.http.SocketTimeout
- com.ibm.ws.webservices.enableHTTPPrefix
- http.proxyHost
- http.proxyPort
- https.proxyHost
- https.proxyPort
- http.nonProxyHosts - You can only configure this property as a JVM custom property. This property applies for both HTTP and HTTPS connections.

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

Configure the additional HTTP properties with an assembly tool with the following steps provided in this task section:

1. Start an assembly tool. Read about starting the assembly tool in the Rational Application Developer documentation.
2. If you have not done so already, configure the assembly tool so that it works on Java EE modules. You need to make sure that the **Java EE** and **Web** categories are enabled. Read about configuring the assembly tool in the Rational Application Developer documentation.
3. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to the Rational Application Developer assembly tool. To migrate files, import your WAR files to the assembly tool. Read about migrating code artifacts to an assembly tool in the Rational Application Developer documentation.
4. Configure the additional HTTP transport properties. Create and specify the name/value pair in the **Web Services Client Port Binding** page for a Web service client. The Web Services Client Port Binding page is available after double-clicking the client deployment descriptor file. Read about configuring HTTP transport properties in the Rational Application Developer documentation.

Results

You have configured additional HTTP transport properties for a Web services application.

Configuring HTTP outbound transport level security with an assembly tool

You can configure the HTTP outbound transport level security with an assembly tool.

Before you begin

You can configure HTTP outbound transport level security with assembly tools provided with WebSphere Application Server.

This task is one of several ways that you can configure the HTTP outbound transport level security for a Web Service acting as a client to another Web service server. You can also configure the HTTP outbound transport level security with the administrative console or by using the Java properties. If you do not configure the HTTP outbound transport level security, the Web services runtime defers to the Java Platform, Enterprise Edition (Java EE) security runtime in the WebSphere product for an effective Secure Sockets Layer (SSL) configuration. If there is no SSL configuration with the Java EE security runtime in the WebSphere product, the Java Secure Socket Extension (JSSE) system properties are used.

About this task

If you configure the HTTP outbound transport level security with assembly tool or with the administrative console, the Web services security binding information is modified. If you have not yet installed the Web services application into WebSphere Application Server, you can configure the HTTP SSL configuration with an assembly tool. This task assumes that you have not deployed the Web services application into the WebSphere product.

If you configure the HTTP outbound transport level security using the standard Java properties for JSSE, the properties are configured as system properties. The configuration that is specified in the binding takes precedence over the Java properties. However, the configurations that are specified by the Java EE security programming model, or are associated with the Dynamic selection, have a higher precedence.

Review the topic Secure communications using Secure Sockets Layer for more information.

1. Start an assembly tool. Read about starting the assembly tool in the Rational Application Developer documentation.
2. If you have not done so already, configure the assembly tool so that it works on Java EE modules. You need to make sure that the **Java EE** and **Web** categories are enabled. Read about configuring the assembly tool in the Rational Application Developer documentation.
3. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to the Rational Application Developer assembly tool. To migrate files, import your WAR files to the assembly tool. Read about migrating code artifacts to an assembly tool in the Rational Application Developer documentation.
4. Configure the HTTP outbound transport level security. Read about enabling Web service endpoints in the Rational Application Developer documentation.

Results

You have configured the HTTP outbound transport level security for a Web Service acting as a client to another Web service with an assembly tool.

Authenticating Web services clients using HTTP basic authentication

A simple way to provide authentication data for the service client is to authenticate to the protected service endpoint by using HTTP basic authentication. *HTTP basic authentication* uses a user name and password to authenticate a service client to a secure endpoint.

Before you begin

You can use either message-level security (WS-Security) or transport-level security:

- Use *message-level security* when security is essential to the Web service application. HTTP basic authentication uses a user name and password to authenticate a service client to a secure endpoint. The basic authentication is encoded in the HTTP request that carries the SOAP message. When the application server receives the HTTP request, the user name and password are retrieved and verified using the authentication mechanism specific to the server.
- Use *transport-level security* to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Transport-level security provides minimal security. You can use this configuration when a Web service is a client to another Web service.

About this task

WebSphere Application Server can have several resources, including Web services, protected by a Java Platform, Enterprise Edition (Java EE) security model.

HTTP basic authentication is orthogonal to the security support provided by WS-Security or HTTP Secure Sockets Layer (SSL) configuration.

A simple way to provide authentication data for the service client is to authenticate to the protected service endpoint using HTTP basic authentication. The basic authentication is encoded in the HTTP request that carries the SOAP message. When the application server receives the HTTP request, the user name and password are retrieved and verified using the authentication mechanism specific to the server.

Although the basic authentication data is base64-encoded, sending data over HTTPS is recommended. The integrity and confidentiality of the data can be protected by the SSL protocol.

- Use the assembly tools that are provided with WebSphere Application Server if you have not deployed the Web services application into the WebSphere product.
- Use the administrative console if you have deployed or installed the Web services application into the WebSphere Application Serve product. If you choose to configure HTTP basic authentication with the administrative console, the Web services security binding information is modified.
- Modify the HTTP properties programmatically if you want the values that are set programmatically to take precedence over the values that are defined in the binding. If you configure HTTP basic authentication programmatically, the properties are configured in the Stub or Call instance. However, you only can programmatically configure HTTP proxy authentication.

Configuring HTTP basic authentication for JAX-RPC Web services with the administrative console

You can configure HTTP basic authentication for Java API for XML-based RPC (JAX-RPC) Web services with the administrative console.

Before you begin

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with an assembly tool or by modifying the HTTP properties programmatically.

If you choose to configure HTTP basic authentication with the administrative console or an assembly tool, the Web services security binding information is modified. You can use the administrative console to configure HTTP basic authentication if you have deployed or installed the Web services application into WebSphere Application Server. If you have not installed the Web services application, then you can configure the security bindings with an assembly tool. This task assumes that you have deployed the Web services application into the WebSphere product.

If you configure HTTP basic authentication programmatically, the properties are configured in the Stub or Call instance. The values set programmatically take precedence over the values defined in the binding.

About this task

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

Configure HTTP basic authentication with the following steps provided in this task section.

Open the administrative console.

1. Click **Applications** > **Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* > **Web services: Client security bindings**.
2. Click **HTTP Basic Authentication** to access the HTTP basic authentication panel. Enter the values in the HTTP Basic Authentication panel.

Results

You have configured the HTTP basic authentication.

HTTP basic authentication collection

Use this page to specify a user name and password for transport-level basic authentication security for this port. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name*** .
2. Click **Manage modules >** ***URI_file_name*** **> Web Services: Client Security Bindings** .
3. Under HTTP basic authentication, click **Edit**.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Basic authentication ID:

The user name for the HTTP basic authentication for this port is set in this field.

Use the Basic authentication ID field to specify the user name for the HTTP basic authentication for this port.

Basic authentication password:

The password for the HTTP basic authentication for this port is set in this field.

Use the Basic authentication password field to specify the password for the HTTP basic authentication for this port.

Configuring HTTP basic authentication for JAX-RPC Web services programmatically

You can configure HTTP basic authentication for Java API for XML-based RPC (JAX-RPC) Web services by programmatically modifying HTTP properties.

Before you begin

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with an assembly tool or with the administrative console.

If you programmatically configure HTTP basic authentication, the properties are configured in the Stub or Call instance. If you choose to configure HTTP basic authentication with the administrative console or an assembly tool, the Web services security binding information is modified. The values that are set programmatically take precedence over the values defined in the binding.

About this task

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

Configure HTTP basic authentication programmatically with the following steps.

Set the properties in the Stub or Call instance for a Web service or a Web service client. You can set properties with the following constant names:

```
javax.xml.rpc.Call.USERNAME_PROPERTY
javax.xml.rpc.Call.PASSWORD_PROPERTY
javax.xml.rpc.Stub.USERNAME_PROPERTY
javax.xml.rpc.Stub.PASSWORD_PROPERTY
```

Example

The following code enables you to configure basic authentication programmatically:

```
Properties prop = new Properties();
InitialContext ctx = new InitialContext(prop);
Service service = (Service)ctx.lookup("java:comp/env/service/StockQuoteService");
QName portQname = new QName("http://httpchannel.test.wsfvt.ws.ibm.com", "StockQuoteHttp");
StockQuote sq = (StockQuote)service.getPort(portQname, StockQuote.class);
((javax.xml.rpc.Stub) sq)._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY, "myUser");
((javax.xml.rpc.Stub) sq)._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY, "myPwd");
```

Configuring HTTP basic authentication for JAX-RPC Web services with an assembly tool

You can configure HTTP basic authentication for Java API for XML-based RPC (JAX-RPC) Web services with an assembly tool.

Before you begin

You can configure HTTP basic authentication with assembly tools provided with WebSphere Application Server.

About this task

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with the administrative console or by modifying the HTTP properties programmatically.

If you choose to configure the HTTP basic authentication with an assembly tool or with the administrative console , the Web services security binding information is modified. You can use an assembly tool to configure HTTP basic authentication before you deploy or install the Web services application into WebSphere Application Server. This task assumes that you have not deployed the Web services application into the WebSphere product.

If you configure HTTP basic authentication programmatically, the properties are configured in the Stub or Call instance. The values set programmatically take precedence over the values defined in the binding.

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

To configure HTTP basic authentication, use the WebSphere Application Server tools to modify the binding information.

1. Start an assembly tool. Read about starting the assembly tool in the Rational Application Developer documentation.
2. If you have not done so already, configure the assembly tool so that it works on Java EE modules. You need to make sure that the **Java EE** and **Web** categories are enabled. Read about configuring the assembly tool in the Rational Application Developer documentation.

3. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to the Rational Application Developer assembly tool. To migrate files, import your WAR files to the assembly tool. Read about migrating code artifacts to an assembly tool in the Rational Application Developer documentation.
4. Configure the HTTP basic authentication in the Web Services Client Port Binding page for a Web service or a Web service client. The Web Services Client Port Binding page is available after double-clicking the client deployment descriptor file. Read about Web Services Client Port Bindings in the Rational Application Developer documentation.

Custom property settings

Use this page to configure name-value pairs for custom binding properties, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. You can specify custom properties that apply to both inbound and outbound messages, or properties that apply only to inbound messages, or only to outbound messages.

To view this administrative console page, click **Applications** → **Application types** → **Websphere enterprise applications** → ***application name*** → **Service provider policy sets and bindings** → ***binding name*** → **WS-Security** → **Custom properties**.

Click **New** to create a new custom property, or click the check box for an existing property name. Click **Delete** to delete an existing property.

Name

Specifies the name, or key, for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the product.

Value

Specifies the value paired with the specified name.

Securing Web services applications using message level security

Web services security standards and profiles describe how to provide security and protection for SOAP messages that are exchanged in a Web services environment.

Before you begin

The Organization for the Advancement of Structured Information Standards (OASIS) Web services security (WS-Security) specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Web services security is a message-level standard based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. WebSphere Application Server Version 7 supports Version 1.1 of the Web Services Security specification, including features such as encrypted header, thumbprint and signature configuration, username token profile and X.509 token profile. In addition, limited security scenario support is provided for the Kerberos Version 1.1 token profile, WS-SecureConversation Version 1.3, WS-Trust Version 1.3, and WS-SecurityPolicy Version 1.2.

About this task

To secure Web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today's business environment is the ability to inter-operate between formerly incompatible security technologies in heterogeneous environments. The complete Web services security protocol stack and technology roadmap is described in Security in a Web Services World: A Proposed Architecture and Roadmap.

The Web Services Security SOAP Message Security 1.1 specification outlines a standard set of SOAP 1.1 extensions that you can use to build secure Web services. These standards provide integrity and confidentiality protection, which are generally implemented with digital signature and encryption technologies. In addition, Web services security provides a general purpose mechanism for associating security tokens with messages. A typical example of the security token is a username token, in which a user name and password are included as text. Web services security defines how to encode binary security tokens using methods such as X.509 certificates. However, the required security tokens are not defined in the SOAP Message Security 1.1 specification. Instead, the tokens are defined in separate profiles such as the Username token profile, the X.509 token profile, and so on.

It is important to note that while Web services security can be used to provide message level integrity and confidentiality protection for normal SOAP message requests from a client to a service, and normal SOAP message responses from a service to a client, Web services security cannot be used to protect SOAP fault messages.

Compatibility between WS-Security Draft 13 and WS-Security standard Versions 1.0 and 1.1

The WS-Security standard has evolved over the years, from a draft to an OASIS standard. WebSphere Application Server Version 5.02 introduced support for the WS-Security Draft 13, and support for WS-Security 1.0 was introduced beginning with WebSphere Application Server Version 6.0. WS-Security Version 1.1 is supported by WebSphere Application Server Version 6.1 Feature Pack for Web Services, using the JAX-WS runtime only. The topic Web services security specification - a chronology provides more details about the evolution of this support.

It is important to note that a WS-Security Draft 13 client is not compatible with providers that use WS-Security Version 1.0 or Version 1.1. You must use Draft 13 client to communicate with a Draft 13 Web services provider. You cannot use a Draft 13 client to communicate with a WS-Security Version 1.0 provider, or a Version 1.1 provider. This issue arises because the SOAP message format for the WS-Security header and namespace is different between a WS-Security Draft 13–enabled application and a WS-Security Version 1.0 or Version 1.1–enabled application.

The version of the WS-Security standard that is used also has implications for the required version of the Java Platform, Enterprise Edition (Java EE) application:

- Java EE Version 1.3 is used only with WS-Security Draft 13.
- Java EE Version 1.4 and later is used with WS-Security Version 1.0 (JAX-RPC and JAX-WS), and also WS-Security Version 1.1 (JAX-WS).

To secure Web services with WebSphere Application Server, you must specify several different configurations. Although there is not a specific sequence in which you must specify these different configurations, some configurations reference other configurations. See "Web services security configuration considerations" on page 81.

Because of the relationship between the different Web services security configurations, it is recommended that you specify the configurations on each level of the configuration in the order described by the following sections. You can choose to configure Web services security for the application level or the

server level as it depends upon your environment and security needs.

- To secure Web services using the Java API for XML-Based Web Services (JAX-WS) programming model, begin with the topic “Securing JAX-WS Web services using message-level security” on page 145.
- To secure Web services using the Java API for XML-based RPC (JAX-RPC) programming model, begin with the topic “Securing JAX-RPC Web services using message level security” on page 328

What is new for securing Web services

In WebSphere Application Server Version 7, there are many security enhancements for Web services. The enhancements include supporting sections of the Web Services Security (WS-Security) specifications and providing architectural support for plugging in and extending the capabilities of security tokens.

Enhancements from the supported Web Services Security specifications

Since September 2002, the Organization for the Advancement of Structured Information Standards (OASIS) has been developing the Web Services Security (WS-Security) for SOAP message standard.

In April 2004, OASIS released the Web Services security Version 1.0 specification, which is a major milestone for securing Web services. In Feburary 2006, the specification was updated to Version 1.1. This specification is the foundation for other Web services security specifications and is also the basis for the Basic Security Profile (WS-I BSP) Version 1.0 specification, which was approved in March 2007.See the Basic Security Profile Web page for more information.

Web Services Security Version 1.1 is a strategic move towards Web services security inter-operability, and an important part of the Web services security roadmap. For more information on the Web services security roadmap, see Security in a Web Services World: A Proposed Architecture and Roadmap.

WebSphere Application Server Version 7 supports the following OASIS specifications and WS-I profiles:

- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.1
- OASIS: Web Services Security: Kerberos Token Profile 1.1
- OASIS: WS-SecurityPolicy 1.2
- OASIS: WS-SecureConversation 1.3
- OASIS: WS-Trust 1.3
- Basic Security Profile (WS-I BSP) 1.0

For details on what parts of the previous specifications are supported in WebSphere Application Server, see “Supported functionality from OASIS specifications” on page 59.

High level features overview in WebSphere Application Server

In WebSphere Application Server, the Web Services Security for SOAP Message Version 1.1 specification is designed to be flexible and accommodate the requirements of Web services. For example, the specification does not have a mandatory security token definition. Instead, the specification defines a generic mechanism to associate the security token with a SOAP message. The use of security tokens is defined in the various Version 1.0 and 1.1 security token profiles, such as:

- The Username Token Profile
- The X.509 Token Profile
- The Kerberos Token Profile

For more information on security token profile development at OASIS, see Organization for the Advancement of Structured Information Standards.

The Web Services Security for SOAP Message Version 1.1 updates the Web Services Security for SOAP Message core specification and the various security token profiles. For this release, WebSphere Application Server implements the Username Token Profile 1.1 and the X.509 Token Profile 1.1, which includes support for the Thumbprint type of security token reference. In addition, it supports the signature confirmation and encrypted header portions of the Web Services Security Version 1.1 standard.

Note: The wire format (such as namespaces) in the WS-SecureConversation and WS-Trust 1.3 specification has changed. WebSphere Application Server version 7 tolerates requests formatted according to both the Submission Drafts and version 1.3 specifications, but you must ensure that the correct version is used when version 7 clients are communicating with a Web Services Feature Pack service provider. You can disable tolerance of the older format for WS-SecureConversation and WS-Trust 1.3 endpoints. Submission Drafts requests are not interoperable with version 1.3 standards.

Support for pluggable security tokens has been available since WebSphere Application Server Version 5.0.2. However, in WebSphere Application Server Versions 6.x and 7, the pluggable architecture is enhanced to support the Web services security specifications, other profiles, and other Web services security specifications. You can learn more about the pluggable security token framework for JAX-RPC Web Services, and associating custom security tokens with SOAP messages, by reading these articles on the IBM developerWorks® Web site:

- Security for JAX-RPC Web services, Part 1: Generating custom tokens
- Security for JAX-RPC Web services, Part 2: Consuming custom tokens

WebSphere Application Server Version 7 includes the following key enhancements:

- Support for the LTPA version 2 token
- Support for configuration of multiple callers, and an order attribute on the caller to determine which caller is used for the WebSphere credential
- Support for the published WS-SecurityPolicy version 1.2 specification embedded in WSDL
- Support for the WS-SecureConversation version 1.3 specification and the WS-Trust version 1.3 specification (used by WS-SecureConversation)
- Support for Kerberos token as defined in the WS-Kerberos Token Profile version 1.1 specification

For more information on some of these enhancements, see “Web services security enhancements” on page 55.

Configuration of Web services security

WebSphere Application Server uses the policy set model for implementing the Web Services Security Version 1.1 specification, including the Username token Version 1.1 profile, support for the Kerberos and LTPA v2 tokens, and the X.509 token version 1.1 profile. Policy sets combine configuration settings, including those for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, WS-SecureConversation, and WS-Security. For more information on policy sets, refer to the topic Managing policy sets using the administrative console.

You can use the administrative console to configure the Web services security binding of a deployed application with Web services security constraints that are defined in the policy set.

For the X.509 Certificate Token Profile, one new type of security token reference is the Thumbprint reference, which is specified in the binding. WebSphere Application Server now supports creating and authenticating a security token by using a security token reference (STR) with a key identifier and a Thumbprint in the <KeyInfo> element. The Thumbprint key information type requires that there be a keystore with the public and private key pair instead of a shared key. To use the Thumbprint of the specified certificate, specify the keyInfo type THUMBPRINT in the bindings.

For example, a decryption key is referenced by means of the thumbprint of an associated certificate. The certificate is not included in the message. Instead, the <ds:KeyInfo> element contains a <wsse:SecurityTokenReference> element that specified the thumbprint of the specified certificate by means of the http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1 attribute of the <wsse:KeyIdentifier> element.

To take advantage of implementations associated with the Web services security Version 1.1 specification, you must:

- Ensure that your applications use the Java API for XML Web Services (JAX-WS) programming model.
- Re-configure the Web services security constraints in the new policy set and binding format.

WebSphere Application Server provides the following tools that you can use to edit the policy set file and the binding file:

IBM assembly tools
You can use IBM assembly tools to develop Web services and configure the policy set and the binding file for Web services security. The tools enable you to assemble both Web and Enterprise JavaBeans (EJB) modules. The assembly tools do not support direct editing of policy sets, but can import policy sets from the application server, and then attach the modified policy sets to the service. For more information on assembly tools, see the topic Assembly tools.

Note: You can use policy sets only with Java API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

WebSphere Application Server administrative console
You can use the administrative console to configure the Web services security binding of a deployed application with Web services security constraints that are defined in the policy set.

What is not supported

Web service security is still fairly new and some of the standards are still being defined or standardized. The following functionality is not supported in WebSphere Application Server:

- JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification). See the standard documentation for more information: JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification).
- Application programming interfaces (API) do not exist for Web services security in WebSphere Application Server Versions 6.0.x and later.
- SAML token profile is not supported out of the box.
- REL token profile is not supported.
- SwA profile is not supported

The following standards exist for the Java application programming interface for XML security and Web services security:

- JSR-105 (Java API for XML-Signature XPath Filter Version 2.0
  W3C Recommendation, November 2002
- JSR-106 (Java API for XML Encryption Syntax and Processing)
  W3C Recommendation, December 2002

For information on what is supported for Web services security in WebSphere Application Server, see “Supported functionality from OASIS specifications” on page 59.

Web services security enhancements

Version 6 and later applications

WebSphere Application Server includes a number of enhancements for securing Web services. For example, policy sets are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later, to simplify security configuration for Web services.

Building your applications

The Web services security run time implementation used by WebSphere Application Server Version 7 is based on the Java API for XML Web Services (JAX-WS) programming model. The JAX-WS run time is based on Apache Open Source Axis2, and the data model is AXIOM. Instead of deployment descriptor and bindings, a policy set is used for configuration. You can use the WebSphere Application Server administrative console to edit the application binding files associated with the policy sets. The JAX-WS run time is supported for the WebSphere Application Server V6.1 Feature Pack for Web Services, and later.

The JAX-RPC programming model, which uses deployment descriptors and bindings, is still supported. Read the topic Securing JAX-RPC Web services using message level security for more information.

Using policy sets

Use policy sets to simplify your Web service Quality of Service configuration.

Note: Policy sets can only be used with JAX-WS applications, in WebSphere Application Server V6.1 Feature Pack for Web Services, and later. Policy sets cannot be used for JAX-RPC applications.

Policy sets combine configuration settings, including those for transport and message level configuration, such as Web Services Addressing (WS-Addressing), Web Services Reliable Messaging (WS-ReliableMessaging), and Web Services Security (WS-Security), which includes Secure Conversation (WS-SecureConversation).

Managing trust policies

Web Services Security Trust (WS-Trust) provides the ability for an endpoint to issue a security context token for Web Services Secure Conversation (WS-SecureConversation). The token issuing support is limited to the security context token. Trust policy management defines a policy for each of the trust service operations, such as issuing, cancelling, validating, and renewing a token. A client's bootstrap policies must correspond to the WebSphere Application Server trust service policies.

Securing session-based messages

Web Services Secure Conversation provides a secured session for long running message exchanges and leveraging symmetric cryptographic algorithm. WS-SecureConversation provides the basic security for securing session-based messages exchange patterns, such as Web Services Security Reliable Messaging (WS-ReliableMessaging).

Updating message-level security

Web Services Security (WS-Security) Version 1.1 supports the following functions that update the message-level security.

- Signature confirmation
- Encrypted headers

Signature confirmation enhances the protection of XML digital signature security. The <SignatureConfirmation> element indicates that the responder has processed the signature in the request, and the signature confirmation ensures that the signature is indeed processed by the intended recipient. To process signature confirmation correctly, the initiator must preserve the signatures during the request generation processing and later must retrieve the signatures for confirmation checks even with the stateless nature of Web Service and the different message exchange patterns. You enable signature confirmation by configuring the policy.

The encrypted header element provides a standard way of encrypting SOAP headers, which helps inter-operability. As defined in the SOAP message security specification, the <EncryptedHeader> element indicates that a specific SOAP header (or set of headers) must be protected. Encrypting SOAP headers and parts helps to provide more secure message-level security. The EncryptedHeader element ensures compliance with the SOAP mustUnderstand processing guidelines and prevents disclosure of information contained in attributes on a SOAP header block.

Using identity assertion

In a secured environment such as an intranet, a secure sockets layer (SSL) connection or through a Virtual Private Network (VPN), it is useful to send the requester identity only without credentials, such as password, with other trusted credentials, such as the server identity. WebSphere Application Server supports the following types of identity assertions:

- A username token without a password
- An X.509 Token for a X.509 certificate

For more information about identity assertion, read the topic Trusted ID evaluator.

Signing or encrypting data with a custom token

For the JAX-RPC programming model, the key locator, or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface, is enhanced to support the flexibility of the specification. The key locator is responsible for locating the key. The local JAAS Subject is passed into the KeyLocator.getKey() method in the context. The key locator implementation can derive the key from the token, which is created by the token generator or the token consumer, to sign a message, to verify the signature within a message, to encrypt a message, or to decrypt a message. The com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface is different from the version in WebSphere Application Server Version 5.x. The com.ibm.wsspi.wssecurity.config.KeyLocator interface from Version 5.x is deprecated. There is no automatic migration for the key locator from Version 5.x to Versions 6 and later. You must migrate the source code for the Version 5.x key locator implementation to the key locator programming model for Version 6 and later.

For the JAX-WS programming model, the pluggable token framework reuses the same framework from the WSS API. The same implementation for creating and validating a security token can be used in both the Web services security run time and the WSS API application. This simplifies the SPI programming model and makes it easier to add new or custom security token types. The redesigned SPI consists of the following interfaces:

- The JAAS CallbackHandler and JAAS Login Module create security tokens on the generator side and validate, or authenticate, security tokens on the consumer side.
- The Security Token interface, com.ibm.websphere.wssecurity.wssapi.token.SecurityToken, represents the security token that has methods to get the identity, XML format and cryptographic keys.

When using JAX-WS, the following interfaces are no longer required:

- Token Generator (com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent)
- Token Consumer (com.ibm.wsspi.wssecurity.token.TokenConsumerComponent)
- Key Locator (com.ibm.wsspi.wssecurity.keyinfo.KeyLocator)

You can learn more about custom security tokens by reading these articles on the IBM developerWorks Web site:

- Security for JAX-RPC Web services, Part 1: Generating custom tokens
- Security for JAX-RPC Web services, Part 2: Consuming custom tokens

Signing or encrypting any XML element

An XPath expression is used for selecting which XML element to sign or encrypt. However, an envelope signature is used when you sign the SOAP envelope, SOAP header, or Web services security header. In JAX-RPC Web services, the XPath expression is specified in the application deployment descriptor. In JAX-WS Web services, the XPath expression is specified in the WS-Security policy of the policy set.

The JAX-WS programming model uses policy sets to indicate the message parts where security should be applied. For example, the <Body> assertion is used to indicate that the body of the SOAP message is signed or encrypted. Another example is the <Header> assertion, where the QName of the SOAP header to be signed or encrypted is specified.

Signing or encrypting SOAP headers

The OASIS Web Service Security (WS-Security) Version 1.1 support provides for a standard way of encrypting and signing SOAP headers. To sign or encrypt SOAP messages, specify the QName to select header elements in the SOAP header of the SOAP message.

You can configure policy sets for signing or encrypting either by using the administrative console or by using Web Services Security APIs (WSS APIs). For more details, see the topic Securing message parts using the administrative console.

For signing, specify the following:

Name This optional attribute indicates the local name of the SOAP header to be integrity protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

Namespace
This required attribute indicates the namespace of the SOAP headers to be integrity protected.

For encrypting, specify the following:

Name This optional attribute indicates the local name of the SOAP header to be confidentiality protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.

Namespace
This required attribute indicates the namespace of the SOAP header(s) to be confidentiality protected.

This results in an <EncryptedHeader> element that contains the <EncryptedData> element.

For Web Services Security Version 1.0 behavior, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.0 property with a value of `true` in EncryptionInfo in the bindings. Specifying this property results in an <EncryptedData> element.

For Web Services Security Version 1.1 behavior that is equivalent to WebSphere Application Server versions prior to version 7.0, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.1.pre.V7 property with a value of true on the <encryptionInfo> element in the binding. When this property is specified, the <EncryptedHeader> element includes a wsu:Id parameter and the <EncryptedData> element omits the Id parameter. This property

should only be used if compliance with Basic Security Profile 1.1 is not required and it is necessary to send <EncryptedHeader> elements to a client or server that uses the WebSphere Application Server Version 5.1 Feature Pack for Web Services.

Supporting LTPA

Lightweight Third Party Authentication (LTPA) is supported as a binary security token in Web services security. Web services security supports both LTPA (version 1) and LTPA version 2 tokens. The LTPA version 2 token, which is more secure than version 1, is supported in WebSphere Application Server version 7.0.

Extending the support for timestamps

You can insert a timestamp in other elements during the signing process besides the Web services security header. This timestamp provides a mechanism for adding a time limit to an element. This support is an extension for WebSphere Application Server. Other vendor implementations might not have the ability to consume a message that is generated with an additional timestamp that is inserted in the message.

Extending the support for nonce

You can insert a nonce, which is a randomly generated value, in other elements beside the Username token. The nonce is used to reduce the chance of a replay attack. This support is an extension for WebSphere Application Server. Other vendor implementations might not have the ability to consume messages with a nonce that is inserted into elements other than a Username token.

Supporting distributed nonce caching

Distributed nonce caching is a new feature for Web services in WebSphere Application Server Versions 6 and later that enables you to replicate nonce data between servers in a cluster. For example, you might have application server A and application server B in cluster C. If application server A accepts a nonce with a value of `X`, then application server B creates a SoapSecurityException if it receives the nonce with the same value within a specified period of time.

Caching the X.509 certificate

WebSphere Application Server caches the X.509 certificates it receives, by default, to avoid certificate path validation and improve its performance. However, this change might lead to security exposure. You can disable X.509 certificate caching by using the following steps:

On the server level:

- Click **Servers > Application servers >** ***server_name*** .
- Under Security, click **Web services: Default bindings for Web services security**.
- Under Additional properties, click **Properties > New**.
- In the Property name field, type `com.ibm.ws.wssecurity.config.token.certificate.useCache`.
- In the Property value field, type `false`.

Providing support for a certificate revocation list

The certificate revocation list (CRL) in WebSphere Application Server is used to enhance certificate path validation. You can specify a CRL in the collection certificate store for validation. You can also encode a CRL in an X.509 token using PKCS#7 encoding. However, WebSphere Application Server Version 6 and later do not support X509PKIPathv1 CRL encoding in a X.509 token.

Note: The PKCS#7 encoding was tested with the IBM certificate path (IBM CertPath) provider only. The encoding is not supported for other certificate path providers.

Supported functionality from OASIS specifications

Version 6 and later applications

The application server supports the Organization for the Advancement of Structured Information (OASIS) Web Services Security (WS-Security) specifications.

WebSphere Application Server supports these OASIS Web Services Security Version 1.0 specifications.

- OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.0
- OASIS: Web Services Security X.509 Certificate Token Profile 1.0

In WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later, support for the OASIS standards has been updated to the latest versions of Web Service Security (WS-Security) specifications and tokens. Web Services Security Version 1.1 provides better security verification for signature, a standard way of encrypting SOAP headers, and meets the requirement from some of the inter-operability scenarios that use features from Web Service Security Version 1.1.

- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security UsernameToken Profile 1.1 (Standard Specification, 1 February 2006)
- OASIS: Web Services Security X.509 Certificate Token Profile 1.1 (Standard Specification, 1 February 2006)

The following standards are supported only in WebSphere Application Server Version 7.0.

- WS-Security Kerberos Token Profile 1.1
- WS-SecureConversation Version 1.3
- WS-Trust Version 1.3
- WS-SecurityPolicy Version 1.2

WS-SecurityPolicy support is only available for Web Services Metadata Exchange (WS-MetadataExchange) scenarios where the assertions are embedded in the WSDL file. For more information, read the WS-MetadataExchange requests topic.

In 2007, the OASIS Web Services Secure Exchange Technical Committee (WS-SX) produced and approved the following specifications. Portions of these specifications are supported by WebSphere Application Server Version 7.

- WS-SecureConversation
- WS-Trust
- WS-SecurityPolicy

OASIS: Web Services Security SOAP Message Security 1.0 and 1.1

The following table shows the aspects of the OASIS: Web Services Security: SOAP Message Security 1.0 and 1.1 specifications that are supported in WebSphere Application Server Versions 6 and later.

Supported topic	Specific aspect that is supported
Security header	• @S11:actor (for an intermediary) • @S11:mustUnderstand • @S12:mustUnderstand • @S12:role (S12 is the namespace prefix for http://www.w3.org/2003/05/soap-envelope when using SOAP Version 1.2)

Supported topic	Specific aspect that is supported
Security tokens	• Username token (user name and password) • Binary security token (X.509 and Lightweight Third Party Authentication (LTPA) • Custom token – Other binary security token – XML token **Note:** WebSphere Application Server does not provide an implementation, but you can use an XML token with plug-in point.
Token references	• Direct reference • Key identifier • Key name • Embedded reference
Signature	Signature confirmation

Supported topic	Specific aspect that is supported
Signature algorithms	• **Digest** **SHA1** http://www.w3.org/2000/09/xmldsig#sha1 **SHA256** http://www.w3.org/2001/04/xmlenc#sha256 **SHA512** http://www.w3.org/2001/04/xmlenc#sha512 • **MAC** **HMAC-SHA1** http://www.w3.org/2000/09/xmldsig#hmac-sha1 • **Signature** **DSA with SHA1** http://www.w3.org/2000/09/xmldsig#dsa-sha1 Do not use this algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP) **RSA with SHA1** http://www.w3.org/2000/09/xmldsig#rsa-sha1 • **Canonicalization** **Canonical XML (with comments)** http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments **Canonical XML (without comments)** http://www.w3.org/TR/2001/REC-xml-c14n-20010315 **Exclusive XML canonicalization (with comments)** http://www.w3.org/2001/10/xml-exc-c14n#WithComments **Exclusive XML canonicalization (without comments)** http://www.w3.org/2001/10/xml-exc-c14n# • **Transform** **STR transform** http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage-security-1.0#STR-Transform **XPath** http://www.w3.org/TR/1999/REC-xpath-19991116 Do not use the original XPATH transform if you want your configured application to be in compliance with the Basic Security Profile (BSP). **Note:** When referring to an element in a SECURE_ENVELOPE that does not carry an attribute of type ID from a ds:Reference in a SIGNATURE, you must use the XPATH Filter 2.0 Transform, http://www.w3.org/2002/06/xmldsig-filter2 **Enveloped signature** http://www.w3.org/2000/09/xmldsig#enveloped-signature **XPath Filter2** http://www.w3.org/2002/06/xmldsig-filter2 **Note:** When referring to an element in a SECURE_ENVELOPE that does not carry an ID attribute type from a ds:Reference in a SIGNATURE, you must use the XPATH Filter 2.0 Transform, http://www.w3.org/2002/06/xmldsig-filter2 **Decryption transform** http://www.w3.org/2002/07/decrypt#XML

Supported topic	Specific aspect that is supported
Signature signed parts for JAX-RPC only	• WebSphere Application Server key words: – body, which signs the SOAP message body – timestamp, which signs all of the time stamps – securitytoken, which signs all of the security tokens – dsigkey, which signs the signing key – enckey, which signs the encryption key – messageid, which signs the `wsa :MessageID` element in WS-Addressing. – to, which signs the `wsa:To` element in WS-Addressing – action, which signs the `wsa:Action` element in WS-Addressing – relatesto, which signs the `wsa:RelatesTo` element in WS-Addressing wsa is the namespace prefix of http://schemas.xmlsoap.org/ws/2004/08/addressing – wscontext, which specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services. – wsafrom, which specifies the `<wsa:From>` WS-Addressing From element in the SOAP header. – wsareplyto, which specifies the `<wsa:ReplyTo>` WS-Addressing ReplyTo element in the SOAP header. – wsafaultto, which specifies the `<wsa:FaultTo>` WS-Addressing FaultTo element in the SOAP header. – wsaall, which specifies all of the WS-Addressing elements in the SOAP header. • XPath expression to select an XML element in a SOAP message. For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Signature message parts for JAX-WS only	• Body (which signs the SOAP message body) • Header (which signs one or more SOAP headers within the main SOAP header) • XPath expression to select an XML element in a SOAP message. – For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Encryption	EncryptedHeader element

Supported topic	Specific aspect that is supported
Encryption algorithms	• Data encryption – Triple DES in CBC: http://www.w3.org/2001/04/xmlenc#tripledes-cbc – AES128 in CBC: http://www.w3.org/2001/04/xmlenc#aes128-cbc – AES192 in CBC: http://www.w3.org/2001/04/xmlenc#aes192-cbc This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings: Message parts” on page 382. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). – AES256 in CBC: http://www.w3.org/2001/04/xmlenc#aes256-cbc This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings: Message parts” on page 382. • Key encryption – Key transport (public key cryptography) - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p. **Note:** • When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5. • Use of the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine does not support this transport algorithm. - RSA Version 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5 – Symmetric key wrap (private key cryptography) - Triple DES key wrap: http://www.w3.org/2001/04/xmlenc#kw-tripledes - AES key wrap (aes128): http://www.w3.org/2001/04/xmlenc#kw-aes128 - AES key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192 This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings: Message parts” on page 382. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). - AES key wrap (aes256): http://www.w3.org/2001/04/xmlenc#kw-aes256 This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings: Message parts” on page 382. • Manifests-xenc is the namespace prefix of http://www.w3.org/TR/xmlenc-core – xenc:ReferenceList – xenc:EncryptedKey Advanced Encryption Standard (AES) is designed to provide stronger and better performance for symmetric key encryption over Triple-DES (data encryption standard). Therefore, it is recommended that you use AES, if possible, for symmetric key encryption.

Supported topic	Specific aspect that is supported
Encryption message parts for JAX-RPC only	• WebSphere Application Server keywords – bodycontent, which is used to encrypt the SOAP body content – usernametoken, which is used to encrypt the username token – digestvalue, which is used to encrypt the digest value of the digital signature – signature, which is used to encrypt the entire digital signature – wscontextcontent, which encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services. • XPath expression to select the XML element in the SOAP message – XML elements – XML element contents
Encryption message parts for JAX-WS only	• Body (which encrypts the SOAP message body content) • Header (which encrypts one or more SOAP headers within the main SOAP header, resulting in the EncryptedHeader element) • XPath expression to select an XML element in a SOAP message – For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.
Time stamp	• Within Web services security header • WebSphere Application Server is extended to allow you to insert time stamps into other elements so that the age of those elements can be determined.
Error handling	SOAP faults • New `failure` SOAP fault with faultcode • `The message has expired` text has been added

OASIS: Web Services Security UsernameToken Profile 1.0

The following table shows the aspects of the OASIS: Web Services Security Username Token Profile 1.0 specification that is supported in WebSphere Application Server.

Supported topic	Specific aspect that is supported
Password types	Text
Token references	Direct reference

OASIS: Web Services Security UsernameToken Profile 1.1

The following table shows the aspects of the OASIS: Web Services Security Username Token Profile 1.1 specification that is supported in WebSphere Application Server. Items that were previously supported for Web Services Security UsernameToken Profile 1.0 are not listed but are still supported, unless noted otherwise.

Supported topic	Specific aspect that is supported
Password types	Text
Token references	Direct reference

OASIS: Web Services Security X.509 Certificate Token Profile 1.0

The following table shows the aspects of the OASIS: Web Services Security X.509 Certificate Token Profile specification that are supported in WebSphere Application Server Versions 6 and later.

Supported topic	Specific aspect that is supported
Token types	• X.509 Version 3: Single certificate http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509v3 • X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL) http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509PKIPathv1 • X.509 Version 3: PKCS7 with or without CRLs. The IBM software development kit (SDK) supports both. The Sun Java SE Development Kit 6 (JDK 6) supports PKCS7 without CRL only.
Token references	• Key identifier – subject key identifier • Direct reference • Custom reference – issuer name and serial number

OASIS: Web Services Security X.509 Certificate Token Profile 1.1

The following table shows the aspects of the OASIS: Web Services Security X.509 Certificate Token Profile 1.1 specification that are supported in WebSphere Application Server. Items that were previously supported for Web Services Security X.509 Certificate Token Profile 1.0 are not listed but are still supported, unless noted otherwise.

Supported topic	Specific aspect that is supported
Token types	X.509 Version 1: Single certificate
Token references	Key identifier – subject key identifier • Can only reference an X.509v3 certificate • Can specify the thumbprint of the specified certificate by using the http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1 attribute of the wsse:KeyIdentifier> element.

OASIS: Web Services Security Kerberos Token Profile 1.1

The following table shows the aspects of the OASIS: Web Services Kerberos Token Profile 1.1 specification that are supported in WebSphere Application Server.

Supported topic	Specific aspect that is supported
Token types	• GSS_API Kerberos v5 token http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ • GSS_API Kerberos v5 token per RFC1510 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510 • GSS_API Kerberos v5 token per RFC4120 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120 • Kerberos v5 token http://docs.oasis-open.org/wss/oasiswss- kerberos-token-profile-1.1#Kerberosv5_AP_REQ • Kerberos v5 token per RFC1510 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510 • Kerberos v5 token per RFC4120 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ412
Token references	• Security token reference • Key identifier, which is used after the initial Kerberos v5 token is consumed • Derived key token based on the Kerberos key

OASIS: Web Services Security WS-Secure Conversation Draft and Version 1.3

The following table shows the aspects of the OASIS: WS-SecureConversation specification that are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later. Support for Version 1.3 of the specification is provided in WebSphere Application Server Version 7.0.

Supported topic	Specific aspect that is supported
Token types	• Security Context Token draft version: http://schemas.xmlsoap.org/ws/2005/02/sc/sct • Security Context Token Version 1.3: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
Token references	Direct reference
Security context establishment	Security context token created by a security token service that is embedded in the WebSphere Application Server.
Renewing context	Automatic renewal of the token when its about to expire.
Cancelling context	Explicit cancel request support.
Derived keys	The following information is used to derive the keys using a shared secret from a security context: • /wsc:DerivedKeyToken/wsse:SecurityTokenReference • /wsc:DerivedKeyToken/wsc:Label • /wsc:DerivedKeyToken/wsc:Nonce • /wsc:DerivedKeyToken/wsc:Length

Supported topic	Specific aspect that is supported
Error handling	SOAP faults, including: • wsc:BadContextToken • wsc:UnsupportedContextToken • wsc:RenewNeeded • wsc:UnableToRenew

OASIS: Web Services Security WS-Trust Version 1.0 Draft and Version 1.3

The following tables show the aspects of the OASIS: Web Services Security: WS-Trust Version 1.0 Draft and Version 1.3 specifications that are supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later.

Table 1. Supported topics for WS-Trust Version 1.0 Draft

Supported topic	Specific aspect that is supported
Namespace	http://schemas.xmlsoap.org/ws/2005/02/trust
Request header	/wsa:Action Valid options include: • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Renew • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Cancel • http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Validate

Table 1. Supported topics for WS-Trust Version 1.0 Draft (continued)

Supported topic	Specific aspect that is supported
Request elements and attributes	/wst:RequestSecurityToken /wst:RequestSecurityToken/@Context /wst:RequestSecurityToken/wst:RequestType • Valid options include: – http://schemas.xmlsoap.org/ws/2005/02/trust/Issue – http://schemas.xmlsoap.org/ws/2005/02/trust/Renew – http://schemas.xmlsoap.org/ws/2005/02/trust/Cancel – http://schemas.xmlsoap.org/ws/2005/02/trust/Validate /wst:RequestSecurityToken/wst:TokenType • Valid options include: – for http://schemas.xmlsoap.org/ws/2005/02/sc/sct - /wst:RequestSecurityToken/wsp:AppliesTo - /wst:RequestSecurityToken/wst:Entropy - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type – for http://schemas.xmlsoap.org/ws/2005/02/trust/Nonce - /wst:RequestSecurityToken/wst:Lifetime - /wst:RequestSecurityToken/wst:Lifetime/wsu:Created - /wst:RequestSecurityToken/wst:Lifetime/wsu:Expires - /wst:RequestSecurityToken/wst:KeySize - /wst:RequestSecurityToken/wst:KeyType – for http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey - /wst:RequestSecurityToken/wst:RenewTarget - /wst:RequestSecurityToken/wst:Renewing - /wst:RequestSecurityToken/wst:Renewing/@Allow - /wst:RequestSecurityToken/wst:Renewing/@OK - /wst:RequestSecurityToken/wst:CancelTarget - /wst:RequestSecurityToken/wst:ValidateTarget - /wst:RequestSecurityToken/wst:Issuer
Response header	/wsa:Action Valid options include: • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Renew • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Cancel • http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Validate

Table 1. Supported topics for WS-Trust Version 1.0 Draft (continued)

Supported topic	Specific aspect that is supported
Response elements and attributes	/wst:RequestSecurityTokenResponse /wst:RequestSecurityTokenResponse/@Context /wst:RequestSecurityTokenResponse/wst:TokenType /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wsp:AppliesTo /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wst:RequestedAttachedReference /wst:RequestSecurityTokenResponse/wst:RequestedUnattachedReference /wst:RequestSecurityTokenResponse/wst:RequestedProofToken /wst:RequestSecurityTokenResponse/wst:Entropy /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret/@Type /wst:RequestSecurityTokenResponse/wst:Lifetime /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Created /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Expires /wst:RequestSecurityTokenResponse/wst:RequestedProofToken/wst:ComputedKey /wst:RequestSecurityTokenResponse/wst:KeySize /wst:RequestSecurityTokenResponse/wst:Renewing /wst:RequestSecurityTokenResponse/wst:Renewing/@Allow /wst:RequestSecurityTokenResponse/wst:Renewing/@OK /wst:RequestSecurityTokenResponse/wst:RequestedTokenCancelled /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status/wst:Code • Valid responses include: – http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid – http://schemas.xmlsoap.org/ws/2005/02/trust/status/invalid /wst:RequestSecurityTokenResponse/wst:Status/wst:Reason

Table 1. Supported topics for WS-Trust Version 1.0 Draft (continued)

Supported topic	Specific aspect that is supported
Error handling	wst:InvalidRequest wst:FailedAuthentication wst:RequestFailed wst:InvalidSecurityToken wst:AuthenticationBadElements wst:BadRequest wst:ExpiredData wst:InvalidTimeRange wst:InvalidScope wst:RenewNeeded wst:UnableToRenew

Table 2. Supported topics for WS-Trust Version 1.3

Supported topic	Specific aspect that is supported
Namespace	http://docs.oasis-open.org/ws-sx/ws-trust/200512
Request header	/wsa:Action Valid options include: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Renew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Cancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Validate • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchIssue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchCancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchRenew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchValidate

Table 2. Supported topics for WS-Trust Version 1.3 (continued)

Supported topic	Specific aspect that is supported
Request elements and attributes	/wst:RequestSecurityToken /wst:RequestSecurityToken/@Context /wst:RequestSecurityToken/wst:RequestType • Valid options include: – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchIssue – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchRenew – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchCancel – http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchValidate /wst:RequestSecurityToken/wst:TokenType • Valid options include: – for http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct - /wst:RequestSecurityToken/wsp:AppliesTo - /wst:RequestSecurityToken/wst:Entropy - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret - /wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type – for http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce - /wst:RequestSecurityToken/wst:Lifetime - /wst:RequestSecurityToken/wst:Lifetime/wsu:Created - /wst:RequestSecurityToken/wst:Lifetime/wsu:Expires - /wst:RequestSecurityToken/wst:KeySize - /wst:RequestSecurityToken/wst:KeyType – for http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey - /wst:RequestSecurityToken/wst:RenewTarget - /wst:RequestSecurityToken/wst:Renewing - /wst:RequestSecurityToken/wst:Renewing/@Allow - /wst:RequestSecurityToken/wst:Renewing/@OK - /wst:RequestSecurityToken/wst:CancelTarget - /wst:RequestSecurityToken/wst:ValidateTarget - /wst:RequestSecurityToken/wst:Issuer
Response header	/wsa:Action Valid options include: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/RenewFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/CancelFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/ValidateFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/CancelFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/RenewFinal • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/ValidateFinal

Table 2. Supported topics for WS-Trust Version 1.3 (continued)

Supported topic	Specific aspect that is supported
Response elements and attributes	/wst:RequestSecurityTokenResponse /wst:RequestSecurityTokenResponse/@Context /wst:RequestSecurityTokenResponse/wst:TokenType /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wsp:AppliesTo /wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken /wst:RequestSecurityTokenResponse/wst:RequestedAttachedReference /wst:RequestSecurityTokenResponse/wst:RequestedUnattachedReference /wst:RequestSecurityTokenResponse/wst:RequestedProofToken /wst:RequestSecurityTokenResponse/wst:Entropy /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret /wst:RequestSecurityTokenResponse/wst:Entropy/wst:BinarySecret/@Type /wst:RequestSecurityTokenResponse/wst:Lifetime /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Created /wst:RequestSecurityTokenResponse/wst:Lifetime/wsu:Expires /wst:RequestSecurityTokenResponse/wst:RequestedProofToken/wst:ComputedKey /wst:RequestSecurityTokenResponse/wst:KeySize /wst:RequestSecurityTokenResponse/wst:Renewing /wst:RequestSecurityTokenResponse/wst:Renewing/@Allow /wst:RequestSecurityTokenResponse/wst:Renewing/@OK /wst:RequestSecurityTokenResponse/wst:RequestedTokenCancelled /wst:RequestSecurityTokenResponse/wst:Status /wst:RequestSecurityTokenResponse/wst:Status/wst:Code • Valid responses include: – http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid – http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/invalid /wst:RequestSecurityTokenResponse/wst:Status/wst:Reason

Table 2. Supported topics for WS-Trust Version 1.3 (continued)

Supported topic	Specific aspect that is supported
Error handling	wst:InvalidRequest wst:FailedAuthentication wst:RequestFailed wst:InvalidSecurityToken wst:AuthenticationBadElements wst:BadRequest wst:ExpiredData wst:InvalidTimeRange wst:InvalidScope wst:RenewNeeded wst:UnableToRenew

Functionality that is not supported by WebSphere Application Server

The following list shows the functionality that is supported in the OASIS specifications, OASIS drafts, and other recommendations but is not supported by WebSphere Application Server Version 6 and later:

- Security Assertion Markup Language (SAML) token profile
- Web services security SOAP Messages with Attachments (SwA) profile 1.0

  Note: When using the JAX-WS programming model, securing the SOAP Message Transmission Optimization Mechanism (MTOM) attachment is supported. See the topic Enabling MTOM for JAX-WS Web services for more information.
- XrML token profile
- XML enveloping digital signature
- XML enveloping digital encryption
- The following WS-SecureConversation functionality is not supported by WebSphere Application Server:
  - Two methods for establishing security context are not supported: 1) security context token created by one of the communicating parties and propagated with a message; and 2) security context token created through negotiation or exchanges.
  - SCT propagation
  - Amending security contexts
- The following transform algorithms for digital signatures are not supported:
  - XSLT: http://www.w3.org/TR/1999/REC-xslt-19991116
  - SOAP Message Normalization

    See SOAP Version 1.2 Message Normalization for information, such as an empty header or header entry with `mustUnderstand=false` is removed, and so forth.
  - Decryption transform
- The following key agreement algorithm for encryption is not supported:
  - Diffie-Hellman: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html#sec-DHKeyValue

- The following canonicalization algorithm for encryption, which is optional in the XML encryption specification, is not supported:
  - Canonical XML with or without comments
  - Exclusive XML Canonicalization with or without comments
- DSA digital signature is not supported.
- Pre-agreed symmetric key data encryption is not supported.
- Auditing for nonrepudiation for digital signatures is not supported.
- In both versions of the Username Token Profile specification, the digest password type is not supported.
- In the Username Token Version 1.1 Profile specification, the key derivation based on a password is not supported.

Unsupported function for WS-Trust Version 1.0 Draft and Version 1.3

The following tables show the aspects of the OASIS: Web Services Security: WS-Trust Version 1.0 Draft and Version 1.3 specifications that are **not** supported in WebSphere Application Server Version 6.1 Feature Pack for Web Services, and later.

Table 3. Unsupported topics for WS-Trust Version 1.0 Draft

Unsupported topic	**Specific aspect that is not supported**
Elements and attributes	/wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type Unsupported request options: • for http://schemas.xmlsoap.org/ws/2005/02/trust/AsymmetricKey and http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey – /wst:RequestSecurityToken/wst:Claims – /wst:RequestSecurityToken/wst:AllowPostdating – /wst:RequestSecurityToken/wst:OnBehalfOf – /wst:RequestSecurityToken/wst:AuthenticationType – /wst:RequestSecurityToken/wst:KeyType • for http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey – /wst:RequestSecurityToken/wst:SignatureAlgorithm – /wst:RequestSecurityToken/wst:EncryptionAlgorithm – /wst:RequestSecurityToken/wst:CanonicalizationAlgorithm – /wst:RequestSecurityToken/wst:ComputedKeyAlgorithm – /wst:RequestSecurityToken/wst:Encryption – /wst:RequestSecurityToken/wst:ProofEncryption – /wst:RequestSecurityToken/wst:UseKey – /wst:RequestSecurityToken/wst:UseKey/@Sig – /wst:RequestSecurityToken/wst:SignWith – /wst:RequestSecurityToken/wst:EncryptWith – /wst:RequestSecurityToken/wst:DelegateTo – /wst:RequestSecurityToken/wst:Forwardable – /wst:RequestSecurityToken/wst:Delegatable – /wst:RequestSecurityToken/wsp:Policy – /wst:RequestSecurityToken/wsp:PolicyReference
Response elements and attributes	/wst:RequestSecurityTokenResponseCollection /wst:RequestSecurityTokenResponseCollection/wst:RequestSecurityTokenResponse

Table 4. Unsupported topics for WS-Trust Version 1.3

Unsupported topic	Specific aspect that is not supported
Elements and attributes	/wst:RequestSecurityToken/wst:Entropy/wst:BinarySecret/@Type Unsupported request options: • for http://docs.oasis-open.org/ws-sx/ws-trust/200512/AsymmetricKey and http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey – /wst:RequestSecurityToken/wst:Claims – /wst:RequestSecurityToken/wst:AllowPostdating – /wst:RequestSecurityToken/wst:OnBehalfOf – /wst:RequestSecurityToken/wst:AuthenticationType – /wst:RequestSecurityToken/wst:KeyType • for http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey and http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer – /wst:RequestSecurityToken/wst:SignatureAlgorithm – /wst:RequestSecurityToken/wst:EncryptionAlgorithm – /wst:RequestSecurityToken/wst:CanonicalizationAlgorithm – /wst:RequestSecurityToken/wst:ComputedKeyAlgorithm – /wst:RequestSecurityToken/wst:Encryption – /wst:RequestSecurityToken/wst:ProofEncryption – /wst:RequestSecurityToken/wst:UseKey – /wst:RequestSecurityToken/wst:UseKey/@Sig – /wst:RequestSecurityToken/wst:SignWith – /wst:RequestSecurityToken/wst:EncryptWith – /wst:RequestSecurityToken/wst:DelegateTo – /wst:RequestSecurityToken/wst:Forwardable – /wst:RequestSecurityToken/wst:Delegatable – /wst:RequestSecurityToken/wsp:Policy – /wst:RequestSecurityToken/wsp:PolicyReference
Response header	/wsa:Action Unsupported Responses: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Issue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Renew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Cancel • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Validate

Web services security specification - a chronology

The development of the Web services security specification includes information on the Organization for the Advancement of Structured Information Standards (OASIS) Web services security specification. The OASIS Web services security specification serves as a basis for securing Web services in WebSphere Application Server.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based

annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Advantages of using the JAX-WS programming model in WebSphere Application Server version 7.0 include:

- The configuration of qualities of service (QoS) is simplified when using policy sets. Policy sets combine configuration settings, including those for transport and message-level configuration. Policy sets and general bindings can be reused across multiple applications, making Web services QoS more consumable.
- WS-Security for JAX-WS is supported in both a managed environment, such as a Java EE container, and unmanaged environments, such as Java Platform, Standard Edition (Java SE 6). In addition, there is an API for enabling WS-Security in the JAX-WS client.

Non-OASIS activities

Web services is gaining rapid acceptance as a viable technology for interoperability and integration. However, securing Web services is one of the paramount quality of services that makes the adoption of Web services a viable industry and commercial solution for businesses. IBM and Microsoft® jointly published a security white paper on Web services entitled Security in a Web Services World: A Proposed Architecture and Roadmap. The white paper discusses the following initial and subsequent specifications in the proposed Web services security roadmap:

Web service security
This specification defines how to attach a digital signature, use encryption, and use security tokens in SOAP messages.

WS-Policy
This specification defines the language that is used to describe security constraints and the policy of intermediaries or endpoints.

WS-Trust
This specification defines a framework for trust models to establish trust between Web services.

WS-Privacy
This specification defines a model of how to express a privacy policy for a Web service and a requester.

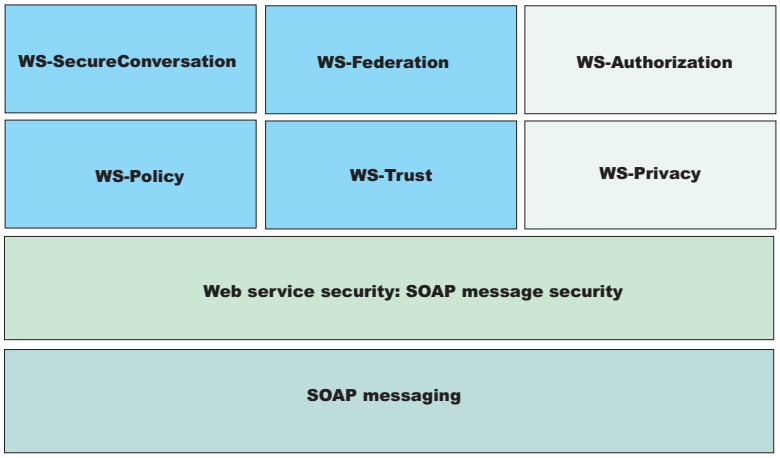
WS-SecureConversation
This specification defines how to exchange and establish a secured context, which derives session keys between Web services.

WS-Authorization

This specification defines the authorization policy for a Web service. However, the WS-Authorization specification has not been published. The existing implementation of Web services security is based upon the Web Services for Java Platform, Enterprise Edition (Java EE) or Java Specification Requirements (JSR) 109 specification. The implementation of Web services security leverages the Java EE role-based authorization checks. For conceptual information on role-based authorization, see Role-based authorization. If you develop a Web service that requires method-level authorization checks, then you must use stateless session beans to implement your Web service. For more information, see “Securing enterprise bean applications” on page 23.

If you develop a Web service that is implemented as a servlet, you can use coarse-grained or URL-based authorization in the Web container. However, in this situation, you cannot use the identity from Web services security for authorization checks. Instead, you can use the identity from the transport. If you use SOAP over HTTP, then the identity is in the HTTP transport.

This following figure shows the relationship between these specifications:



In April 2002, IBM, Microsoft, and VeriSign proposed the Web Services Security (WS-Security) specification on their Web sites as depicted by the green box in the previous figure. This specification included the basic ideas of a security token, XML digital signature, and XML encryption. The specification also defined the format for user name tokens and encoded binary security tokens. After some discussion and an interoperability test based on the specification, the following issues were noted:

- The specification requires that the Web services security processors understand the schema correctly so that the processor distinguishes between the ID attribute for XML digital signature and XML encryption.
- The freshness of the message, which indicates whether the message complies with predefined time constraints, cannot be determined.
- Digested password strings do not strengthen security.

In August 2002, IBM, Microsoft, and VeriSign published the *Web Services Security Addendum*, which attempted to address the previously listed issues. The following solutions were addressed in the addendum:

- Require a global ID attribute for XML signature and XML encryption.
- Use time stamp header elements that indicate the time of the creation, receipt, or expiration of the message.
- Use password strings that are digested with a time stamp and nonce, which is a randomly generated token.

The specifications for the blue boxes in the previous figure have been proposed by various industry vendors and various interoperability events have been organized by the vendors to verify and refine the proposed specifications.

OASIS activities

In June 2002, OASIS received a proposed Web services security specification from IBM, Microsoft, and VeriSign. The Web Services Security Technical Committee (WSS TC) was organized at OASIS soon after the submission. The technical committee included many companies including IBM, Microsoft, VeriSign, Sun Microsystems, and BEA Systems.

In September 2002, WSS TC published its first specification, Web Services Security Core Specification, Working Draft 01. This specification included the contents of both the original Web services security specification and its addendum.

The coverage of the technical committee became larger as the discussion proceeded. Because the Web Services Security Core Specification allows arbitrary types of security tokens, proposals were published as profiles. The profiles described the method for embedding tokens, including Security Assertion Markup

Language (SAML) tokens and Kerberos tokens embedded into the Web services security messages. Subsequently, the definitions of the usage for user name tokens and X.509 binary security tokens, which were defined in the original Web Services Security Specification, were divided into the profiles.

WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1 support the following specifications:

- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)
- Web Services Security: Username Token Profile Draft 2

In April 2004, the Web service security specification (officially called Web Services Security: SOAP Message Security Version 1.0) became the Version 1.0 OASIS standard. Also, the Username token and X.509 token profiles are Version 1.0 specifications. WebSphere Application Server 6 and later support the following Web services security specifications from OASIS:

- Web Services Security: SOAP Message Security 1.0 specification
- Web Services Security: Username Token 1.0 Profile
- Web Services Security: X.509 Token 1.0 Profile

In February 2006, the core Web service security specification was updated and became the Version 1.1 OASIS standard. Also, the Username token, X.509 token profile, and Kerberos token profile were updated to the Version 1.1 specifications. Portions of the following Web services security specifications from OASIS are supported in WebSphere Application Server, specifically signature confirmation, encrypted header, and thumbprint references:

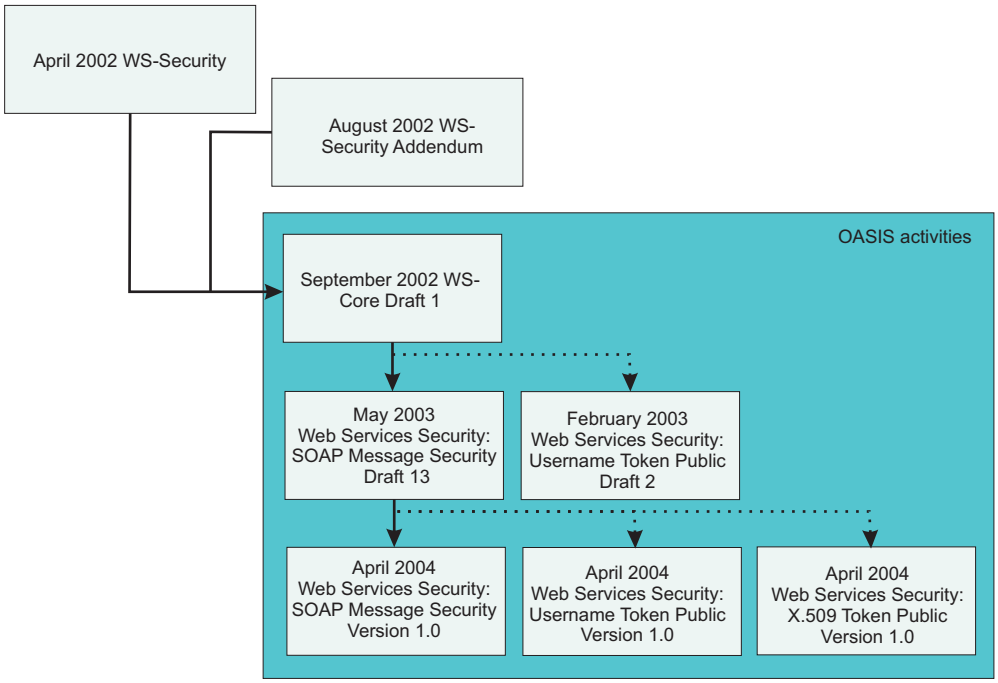
- OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security UsernameToken Profile 1.1 OASIS Standard Specification, 1 February 2006
- OASIS: Web Services Security X.509 Certificate Token Profile 1.1 OASIS Standard Specification, 1 February 2006

The following specification describes the use of Kerberos tokens with respect to the Web services security message security specifications. The specification defines how to use a Kerberos token to support authentication and message protection: OASIS: Web Services Security Kerberos Token Profile 1.1 OASIS Standard Specification, 1 February 2006.

In 2007, the OASIS Web Services Secure Exchange Technical Committee (WS-SX) produced and approved the following specifications. Portions of these specifications are supported by WebSphere Application Server Version 7.

- WS-SecureConversation
- WS-Trust
- WS-SecurityPolicy

The following figure shows the various Web services security-related specifications.



WebSphere Application Server also provides plug-in capability to enable security providers to extend the runtime capability and implement some of the higher level specifications in the Web service security stack. The plug-in points are exposed as Service Provider Programming Interfaces (SPI). For more information on these SPIs, see “Default implementations of the Web services security service provider programming interfaces” on page 114.

Web services security specification 1.0 development

The OASIS Web services security specification is based upon the following World Wide Web Consortium (W3C) specifications. Most of the W3C specifications are in the standard body recommended status.

- XML-Signature Syntax and Processing
  W3C recommendation, February 2002 (Also, IETF RFC 3275, March 2002)
- Canonical XML Version 1.0
  W3C recommendation, March 2001
- Exclusive XML Canonicalization Version 1.0
  W3C recommendation, July 2002
- XML-Signature XPath Filter Version 2.0
  W3C Recommendation, November 2002
- XML Encryption Syntax and Processing
  W3C Recommendation, December 2002
- Decryption Transform for XML Signature
  W3C Recommendation, December 2002

These specifications are supported in WebSphere Application Server in the context of Web services security. For example, you can sign a SOAP message by specifying the integrity option in the deployment descriptors. There is a client side application programming interface (API) that an application can use to enable Web services security for securing a SOAP message.

The OASIS Web services security Version 1.0 specification defines the enhancements that are used to provide message integrity and confidentiality. It also provides a general framework for associating the security tokens with a SOAP message. The specification is designed to be extensible to support multiple

security token formats. The particular security token usage is addressed with the security token profile.

Specification and profile support in WebSphere Application Server

OASIS is working on various profiles. For more information, see Organization for the Advancement of Structured Information Standards Committees.

The following list includes of the published draft profiles and OASIS Web services security technical committee work in progress.

WebSphere Application Server does not support these profiles:

- Web Services Security: SAML token profile 1.0
- Web Services Security: Rights Expression Language (REL) token profile 1.0
- Web Services Security: SOAP Messages with Attachments (SwA) profile 1.0

Note: Support for Web services security draft 13 and Username token profile draft 2 is deprecated in WebSphere Application 5.0.2, 5.1.0 and 5.1.1. For migration information, see “Migrating JAX-RPC Web services security applications to Version 7.0 applications” on page 329.

The wire format of the SOAP message with Web services security in Web services security Version 1.0 has changed and is not compatible with previous drafts of the OASIS Web services security specification. Interoperability between OASIS Web services security Version 1.0 and previous Web services security drafts is not supported. However, it is possible to run an application that is based on Web services security draft 13 on WebSphere Application Server Version 6 and later. The application can interoperate with an application that is based on Web services security draft 13 on WebSphere Application Server Version 5.0.2, 5.1 or 5.1.1.

WebSphere Application Server supports both the OASIS Web services security draft 13 and the OASIS Web services security 1.0 specification. But in WebSphere Application Server Version 6 and later, the support of OASIS Web services security draft 13 is deprecated. However, applications that were developed using OASIS Web services security draft 13 on WebSphere Application Server 5.0.2, 5.1.0 and 5.1.1 can run on WebSphere Application Server Version 6 and later. OASIS Web services security Version 1.0 support is available only for Java Platform, Enterprise Edition (Java EE) Version 1.4 and later applications. The configuration format for the deployment descriptor and the binding is different from previous versions of WebSphere Application Server. You must migrate the existing applications to Java EE 1.4 and migrate the Web services security configuration to the WebSphere Application Server Version 6 format.

Other Web services security specifications development

The most recently updated versions of the following OASIS Web services security specifications are supported in WebSphere Application Server Version 7.0 in the context of Web services security:

- WS-Trust Version 1.3

  The Web Services Trust Language (WS-Trust) uses the secure messaging mechanisms of Web services security to define additional primitives and extensions for the issuance, exchange and validation of security tokens. WS-Trust enables the issuance and dissemination of credentials within different trust domains. This specification defines ways to establish, assess the presence of, and broker trust relationships.
- WS-SecureConversation Version 1.3

  The Web Services Secure Conversation Language (WS-SecureConversation) is built on top of the WS-Security and WS-Policy models to provide secure communication between services. WS-Security focuses on the message authentication model but not a security context, and thus is subject several forms of security attacks. This specification defines mechanisms for establishing and sharing security

contexts, and deriving keys from security contexts, to enable a secure conversation. By using the SOAP extensibility model, modular SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment.

- WS-SecurityPolicy Version 1.2

  Web Services Security Policy (WS-Policy) provides a general purpose model and syntax to describe and communicate the policies of a Web service. WS-Policy assertions express the capabilities and constraints of a particular Web service. WS-PolicyAttachments defines several methods for associating the WS-Policy expressions with Web services (such as WSDL). The Web services security specifications have been updated following the re-publication of WS-Security Policy in July 2005, to reflect the constraints and capabilities of Web services that are using WS-Security, WSTrust and WS-SecureConversation. WS-ReliableMessaging Policy has also been re-published in 2005 to express the capabilities and constraints of Web services implementing WS-ReliableMessaging.

Web Services Interoperability Organization (WS-I) activities

Web Services Interoperability Organization (WS-I) is an open industry effort to promote Web services interoperability across vendors, platforms, programming languages and applications. The organization is a consortium of companies across many industries including IBM, Microsoft, Oracle, Sun, Novell, VeriSign, and Daimler Chrysler. WS-I began working on the basic security profile (BSP) in the spring of 2003. BSP consists of a set of non-proprietary Web services specifications that clarifies and amplifies those specifications to promote Web services security interoperability across different vendor implementations. As of June 2004, BSP is a public draft. For more information, see the Web Services Interoperability Organization Web page.

Specifically, see Basic Security Profile Version 1.0 for details about the BSP. WebSphere Application Server supports compliance with the BSP draft, but Web services security does not support the BSP Version 1.1 draft. See “Basic Security Profile compliance tips” on page 142 for the details to configure your application in compliance with the BSP draft.

Web services security configuration considerations

Version 6 and later applications

To secure Web services security for WebSphere Application Server, you must specify several different configurations. Although there is not a specific sequence in which you must specify these different configurations, some configurations reference other configurations.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

You can configure Web services security on the application level and the server level. The following table shows an example of the relationships between each of the configurations that apply to just the application, to an entire server. However, the requirements for the bindings depend upon the deployment descriptor. Some binding information depends upon other information in the binding or server-level configuration. Within the table, the configurations in the Referenced configurations column are referenced by the configuration listed in the Configuration name column. For example, the token generator on the application-level for the request generator references the collection certificate store, the nonce, time stamp, and callback handler configurations.

Table 5. The relationship between the configurations.

Configuration level	Configuration name	Referenced configurations
Application-level request generator	Token generator	• Collection certificate store • Nonce • Timestamp • Callback handler
Application-level request generator	Key information	• Key locator • Key name • Token
Application-level request generator	Signing information	• Key information
Application-level request generator	Encryption information	• Key information
Application-level request consumer	Token consumer	• Trust anchor • Collection certificate store • Trusted ID evaluators • Java Authentication and Authorization Service (JAAS) configuration
Application-level request consumer	Key information	• Key locator • Token
Application-level request consumer	Signing information	• Key information
Application-level request consumer	Encryption information	• Key information
Application-level response generator	Token generator	• Collection certificate store • Callback handler
Application-level response generator	Key information	• Key locator • Token
Application-level response generator	Signing information	• Key information
Application-level response generator	Encryption information	• Key information
Application-level response consumer	Token consumer	• Trust anchor • Collection certificate store • JAAS configuration
Application-level response consumer	Key information	• Key locator • Key name • Token
Application-level response consumer	Signing information	• Key information
Application-level response consumer	Encryption information	• Key information
Server-level default generator bindings	Token generator	• Collection certificate store • Callback handler
Server-level default generator bindings	Key information	• Key locator • Token
Server-level default generator bindings	Signing information	• Key information

Table 5. The relationship between the configurations. (continued)

Configuration level	Configuration name	Referenced configurations
Server-level default generator bindings	Encryption information	• Key information
Server-level default consumer bindings	Token consumer	• Trust anchor • Collection certificate store • Trusted ID evaluator • JAAS configuration
Server-level default consumer bindings	Key information	• Key locator • Token
Server-level default consumer bindings	Signing information	• Key information
Server-level default consumer bindings	Encryption information	• Key information

When multiple applications will use the same binding information, consider configuring the binding information on the server level. For example, you might have a global key locator configuration that is used by multiple applications. Configuration information for the application-level precedes similar configuration information on the server-level.

Default bindings and runtime properties for Web services security

Use this page to configure the settings for nonce on the server level and to manage the default bindings for the signing information, encryption information, key information, token generators, token consumers, key locators, collection certificate store, trust anchors, trusted ID evaluators, algorithm mappings, and login mappings.

Displayed options and the panel title depend on your server configuration and version.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

Read the Web services documentation before you begin defining the default bindings for Web services security.

Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens.

In WebSphere Application Server and WebSphere Application Server Express, you must specify values for the Nonce cache timeout, Nonce maximum age, and Nonce clock skew fields for the server level.

Nonce cache timeout

Version 5.x or 6 application

Specifies the timeout value, in seconds, for the nonce cached on the server. Nonce is a randomly generated value.

The Nonce cache timeout field is required on the server level.

If you make changes to the value for the Nonce cache timeout field, you must restart the application server for the changes to take effect.

Default	600 seconds
Minimum	300 seconds

Nonce maximum age

Version 5.x or 6 application

Specifies the default time, in seconds, before the nonce timestamp expires. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce cache timeout field for the server level.

The Nonce maximum age field is required on the server level.

Default	300 seconds
Range	300 to the value that is specified, in seconds, in the Nonce cache timeout field.

Nonce clock skew

Version 5.x or 6 application

Specifies the default clock skew value, in seconds, to consider when the application server checks the timeliness of the message. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce maximum age field.

The Nonce clock skew field is required.

Default	0 seconds
Range	0 to the value that is specified, in seconds, in the Nonce maximum age field.

Enable cryptographic operations on hardware device

Enables cryptographic operations on hardware devices. Enabling this feature might improve the performance, depending on the hardware device.

Cryptographic hardware configuration name

Specifies the name of the hardware device configuration name that is defined in the keystore settings in the secure communications.

This value is necessary only if **Hardware acceleration** has been selected.

Custom properties

The linked Properties panel specifies additional properties for the security runtime configuration.

Web services security provides message integrity, confidentiality, and authentication

OASIS Web Services Security (WS-Security) is a flexible standard that is designed to secure Web services within a wide variety of security models. You can secure SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. Web services implements security using technology that includes transport-level Secure Sockets Layer (SSL).

The Web services security specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Message-level security, or securing Web services at the message level, addresses the same security requirements as for traditional Web security. These security requirements include: identity, authentication, authorization, integrity, confidentiality, nonrepudiation, basic message exchange, and so forth. Both traditional Web and message-level security share many of the same mechanisms for handling security, including digital certificates, encryption, and digital signatures. While HTTPS and SSL transport-level technology may be used for securing Web services, some security scenarios are addressed more effectively by message-level security.

Traditional Web security mechanisms, such as HTTPS, might be insufficient to manage the security requirements of all Web service scenarios. For example, when an application sends a document with JAX-RPC using HTTPS, the message is secured only for the HTTPS connection, meaning during the transport of the document between the service requester (the client) and the service. However, the application might require that the document data be secured beyond the HTTPS connection, or even beyond the transport layer. By securing Web services at the message level, message-level security is capable of meeting these expanded requirements.

Message-level security applies to XML documents that are sent as SOAP messages. Message-level security makes security part of the message itself by embedding all required security information in the SOAP header of a message. In addition, message-level security can apply security mechanisms, such as encryption and digital signature, to the data in the message itself.

With message-level security, the SOAP message itself either contains the information needed to secure the message or it contains information about where to get that information to handle security needs. The SOAP message also contains information relevant to the protocols and procedures for processing the specified message-level security. However, message-level security is not tied to any particular transport mechanism. Because the security information is part of the message, it is independent of a transport protocol, such as HTTPS.

The client adds to the SOAP message header security information that applies to that particular message. When the message is received, the Web service endpoint, using the security information in the header, verifies the secured message and validates it against the policy. For example, the service endpoint might verify the message signature and check that the message has not been tampered with. It is possible to add signature and encryption information to the SOAP message headers, as well as other information such as security tokens for identity (for example, an X.509 certificate) that are bound to the SOAP message content.

For WebSphere Application Server Versions 6 and later, Web services security can be applied as transport-level security and as message-level security. You can architect highly secure client and server designs by using these security mechanisms. Transport-level security refers to securing the connection between a client application and a Web service with Secure Sockets Layer (SSL).

You can apply various scenarios of Web services security according to the characteristics of each Web service application. You have choices of how to protect your information when using Web services security. The authentication mechanism, integrity, and confidentiality can be applied at the message level and at the transport level. When message-level security is applied, you can protect the SOAP message with a security token, digital signature, and encryption.

Without Web services security, the SOAP message is sent in clear text, and personal information such as a user ID or an account number is not protected. Without applying Web services security, there is only a SOAP body under the SOAP envelope in the SOAP message. By applying features from the WS-Security specification, the SOAP security header is inserted under the SOAP envelope in the SOAP message when the SOAP body is signed and encrypted.

To maintain the integrity or confidentiality of the message, digital signatures and encryption are typically applied.

- *Confidentiality* specifies the confidentiality constraints that are applied to generated messages. This includes specifying which message parts within the generated message must be encrypted, and the message parts to attach encrypted Nonce and time stamp elements to.
- *Integrity* is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption. Multiple signatures and encryptions are supported. In addition, both signing and encryption can be applied to the same parts, such as the SOAP body.

You can add an authentication mechanism by inserting various types of security tokens, such as the Username token (<UsernameToken> element). When the Username token is received by the Web service server, the user name and password are extracted and verified. Only when the user name and password combination is valid, will the message be accepted and processed at the server. Using the Username token is just one of the ways of implementing authentication. This mechanism is also known as *basic authentication*.

In addition to digital signatures, encryption, and basic authentication, other forms of authentication include identity assertion, LTPA tokens, Kerberos tokens, and custom tokens. These other forms of authentication are also extensions of WebSphere Application Server. You can configure these authentication mechanisms using the assembly tools to implement authentication.

With updates to Web Services Security in the Version 1.1 specification, it is possible to layer additional functionality on top of these basic mechanisms. Some Version 1.1 mechanisms are extensions of WebSphere Application Server, such as signature confirmation and the encrypted header. The security token profiles that are supported by WebSphere Application Server include the Username token profile, the X.509 token profile, and the Kerberos profile. In this case, when the message is received, the Web service endpoint, using the security information in the header, applies the appropriate security mechanisms to the message. For example, the service endpoint might add signature and encryption information to the SOAP message headers, as well as other information, such as security tokens, that are bound to the SOAP message content. You can implement these new mechanisms by using a policy set.

WS-SecureConversation was introduced in WebSphere Application Server Version 6.1 with the Feature Pack for Web Services. Secure Conversation uses a session key to protect SOAP messages more efficiently, particularly when multiple SOAP messages are transmitted in a session.

Other enhancements, added in WebSphere Application Server Version 7.0, include:

- The Kerberos token, which is used for both authentication and for subsequent message protection.
- Dynamic policy, which allows the client to retrieve the provider policy through a WSDL request, or using Web Services MetadataExhange (WS-MEX), to simplify Web services client deployment.

High-level architecture for Web services security

Version 6 and later applications

The Web services security policy is specified in the IBM extension of the Web services deployment descriptors when using the JAX-RPC programming model, and in policy sets when using the JAX-WS programming model. A stand-alone JAX-WS client application may specify Web Services security policy programmatically. Binding data that supports the Web Services security policy are stored in the IBM extension of the Web services deployment descriptors for both the JAX-RPC and JAX-WS programming

models. The Web Services security run time enforces the security assertions that are specified in the policy document, or in the application program, in that order.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

WebSphere Application Server uses the Java Platform, Enterprise Edition (Java EE) Version 1.4 or later Web services deployment model to implement Web services security. One of the advantages of deployment model is that you can define the Web services security requirements outside of the application business logic. With the separation of roles, the application developer can focus on the business logic and the security expert can specify the security requirement.

There are two sets of configurations on both the client side and the server side:

Request generator
This client-side configuration defines the Web services security requirements for the outgoing SOAP message request. These requirements might involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request generator was known as the *request sender*.

Request consumer
This server-side configuration defines the Web services security requirements for the incoming SOAP message request. These requirements might involve verifying that the required integrity parts are digitally signed; verifying the digital signature; verifying that the required confidential parts were encrypted by the request generator; decrypting the required confidential parts; validating the security tokens, and verifying that the security context is set up with the appropriate identity. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request consumer was known as the *request receiver*.

Response generator
This server-side configuration defines the Web services security requirements for the outgoing SOAP message response. These requirements might involve generating the SOAP message response with Web services security; including digital signature; and encrypting and attaching the security tokens, if necessary. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response generator was known as the *response sender*.

Response consumer
This client-side configuration defines the Web services security requirements for the incoming SOAP response. The requirements might involve verifying that the integrity parts are signed and the signature is verified; verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response consumer was known as the *response receiver*.

WebSphere Application Server does not include security policy negotiation or exchange between the client and server. This security policy negotiation, as defined by the WS-Policy, WS-PolicyAssertion, and WS-SecurityPolicy specifications, are not supported in WebSphere Application Server.

Note: The Web services security requirements that are defined in the request generator must match the request consumer. The requirements that are defined in the response generator must match the response consumer. Otherwise, the request or response is rejected because the Web services security constraints cannot be met by the request consumer and response consumer.

The format of the Web services security deployment descriptors and bindings are IBM proprietary. However, the following tools are available to edit the deployment descriptors and bindings:

IBM assembly tools
Use IBM assembly tools to edit the Web services security deployment descriptor and binding. Use the tools to assemble both Web and Enterprise JavaBeans (EJB) modules. For more information on assembly tools, see Assembly tools.

WebSphere Application Server Administrative Console
Use this tool to edit the Web services security binding of a deployed application.

Security model mixture: **Version 6 and later applications**

There can be multiple protocols and channels in the WebSphere Application Server Version 6 and later programming environments. Each of these applications serve different business needs.

For example, you might access:

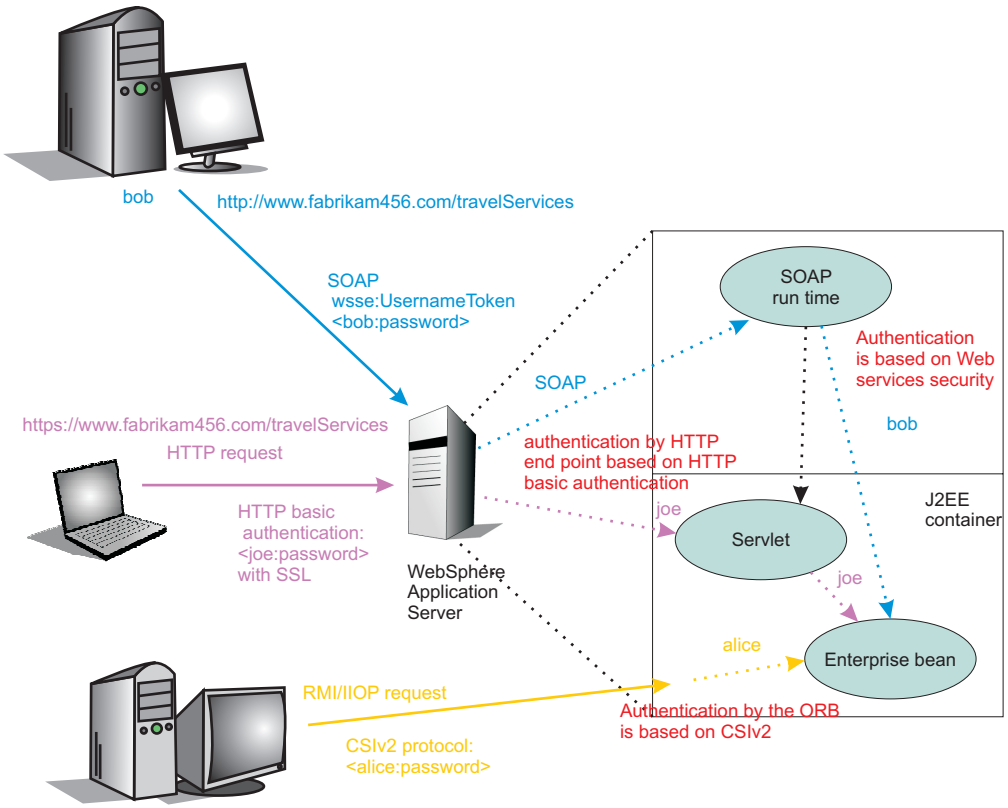
- A Web-based application through the HTTP transport such as a servlet, JavaServer Pages (JSP) file, HTML and so on.
- An enterprise application through the Remote Method Invocation (RMI) over the Internet Inter-ORB (RMI/IIOP) protocol.
- A Web service application through the SOAP over HTTP, SOAP over the Java Message Service (JMS), or SOAP over the RMI/IIOP protocol.

More importantly, Web services are often implemented as servlets with a Enterprise JavaBeans (EJB) file. Therefore, you can mix and match the Web services security model with the Java Platform, Enterprise Edition (Java EE) security model for Web and EJB components. It is intended that Web service security complement the Java EE role-based security and the security run time for WebSphere Application Server Version 6 and later.

Web services security also can take advantage of the security features in Java EE and the security run time for WebSphere Application Server Version 6 and later. For example, Web services security can use the following security features to provide an end-to-end security deployment:

- Use the local OS, Lightweight Directory Access Protocol (LDAP), and custom user registries for authenticating the username token
- Propagate the Lightweight Third Party Authentication (LTPA) security token in the SOAP message
- Use identity assertion
- Use a trust association interceptor (TAI)
- Enable security attribute propagation
- Use Java EE role-based authorization
- Use a Java Authorization Contract for Containers (JACC) authorization provider, such as Tivoli® Access Manager

The following figure shows that different security protocols are used to send authentication information to the application server. For a Web service, you might use either HTTP basic authentication with Secure Sockets Layer (SSL) or a Web services security username token with signing and encryption. In the following figure, when identity *bob* from Web services security is authenticated and set as the caller identity of the SOAP message request, the Java EE Enterprise JavaBeans container performs authorization using *bob* before the call is dispatched to the service implementation, which, in this case, is the enterprise bean.



You can secure a Web service using the transport layer security. For example, when you are using SOAP over HTTP, HTTPS can be used to secure the Web service. However, transport layer security provides point-to-point security only. This layer of security might be adequate for certain scenarios. However, when the SOAP message must travel through intermediary servers (multi-hop) before it is consumed by the target endpoint, you might use SOAP over the Java Message Service (JMS). The usage scenarios and security requirements dictate how to secure Web services. The requirements depend upon the operating environment and the business needs. However, one key advantage of using Web services security is that it is transport layer independent; the same Web services security constraints can be used for SOAP over HTTP, SOAP over JMS, or SOAP over RMI/IIOP.

Overview of platform configuration and bindings: **Version 6 and later applications**

The Web services security policy is specified in the IBM extension of the Web services deployment descriptors when using the JAX-RPC programming model, and in policy sets when using the JAX-WS programming model. Binding information to support the Web Services security policy is stored in the IBM extension of the Web services deployment descriptors for both the JAX-RPC and JAX-WS programming models.

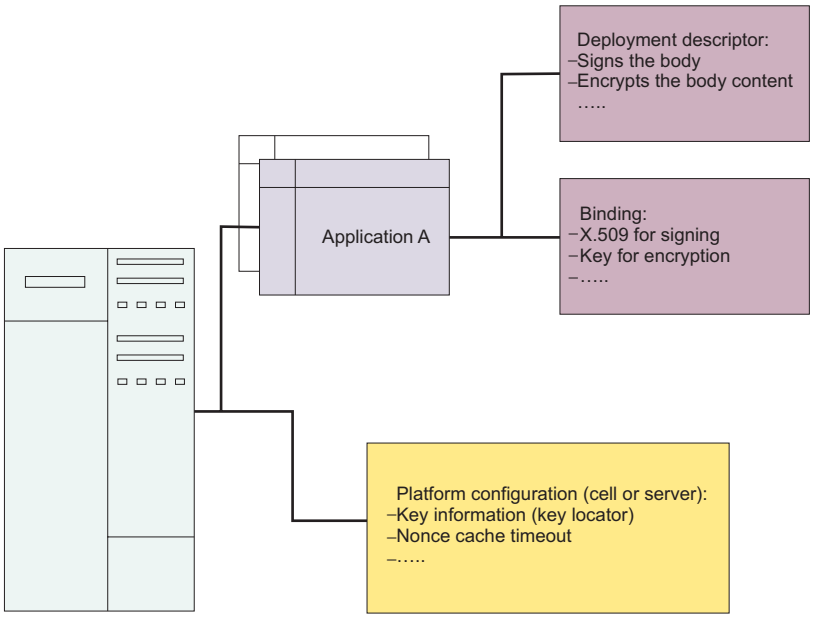
Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Due to the complexity of these files, it is not recommended that you edit the deployment descriptor and binding files manually with a text editor because they might cause errors. It is recommended, however, that you use the tools provided by IBM to configure the Web services security constraints for an application. These tools are the WebSphere Application Server administrative console, or an assembly tool. For more information about IBM assembly tools, see the topic Assembly tools.

You can use the policy set function of the WebSphere Application Server to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. Policy sets are assertions about how quality of services is defined. A policy set incorporates policy types, and their settings.

In addition to the application deployment descriptor and binding files, WebSphere Application Server Versions 6 and later have a server level Web Security Services configuration. These configurations are global for all applications. Because WebSphere Application Server Version 6 and later support 5.x applications, some of the configurations are valid for Version 5.x applications only and some are valid for Version 6 and later applications only.

The following figure represents the relationship of the application deployment descriptor and binding files to the cell (Network Deployment only) or server level configuration.



WebSphere Application Server

Platform configuration

The following options are available in the administrative console:

Nonce cache timeout
This option, which is found on the cell level (Network Deployment only) and server level, specifies the cache timeout value for a nonce in seconds.

Nonce maximum age
This option, which is found on the cell level (Network Deployment only) and server level, specifies the default life span for the nonce in seconds.

Nonce clock skew
This option, which is found on the cell level (Network Deployment only) and server level, specifies the default clock skew to account for network delay, processing delay, and so on. It is used to calculate when the nonce expires. Its unit of measurement is seconds.

Distribute nonce caching
This feature enables you to distribute the cache for the nonce to different servers in a cluster. It is available for WebSphere Application Server Version 6.0.x and later.

The following features can be referenced in the application binding:

Key locator
This feature specifies how the keys are retrieved for signing, encryption, and decryption. The implementation classes for the key locator are different in WebSphere Application Server Versions 6 and later and Version 5.*x*.

Collection certificate store
This feature specifies the certificate store for certificate path validation. It is typically used for validating X.509 tokens during signature verification or constructing the X.509 token with a certificate revocation list that is encoded in the PKCS#7 format. The certificate revocation list is supported for WebSphere Application Server Version 6.x and later applications only.

Trust anchors
This feature specifies the trust level for the signer certificate and is typically used in the X.509 token validation during signature verification.

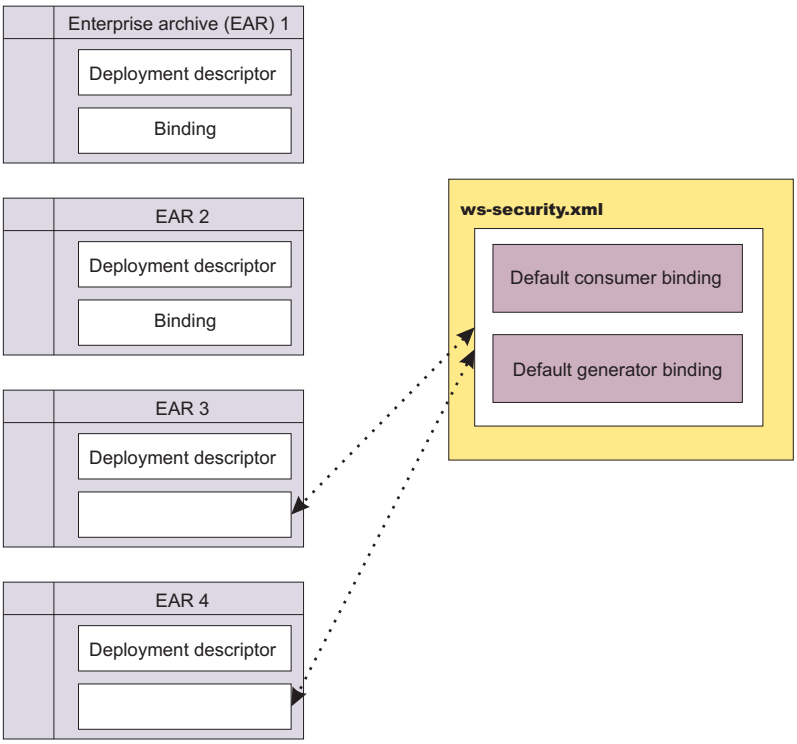
Trusted ID evaluators
This feature specifies how to verify the trust level for the identity. The feature is used with identity assertion.

Login mappings
This feature specifies the login configuration binding to the authentication methods. This feature is used by WebSphere Application Server Version 5.*x* applications only and it is deprecated.

Default bindings

The configuration of the default cell level and default server level bindings has changed in WebSphere Application Server version 7.0. Previously, you could configure only one set of default bindings for the cell, and optionally configure one set of default bindings for each server. In version 7.0, you can configure one or more general provider bindings and one or more general client bindings. However, only one general provider binding and one general client binding can be designated as the default.

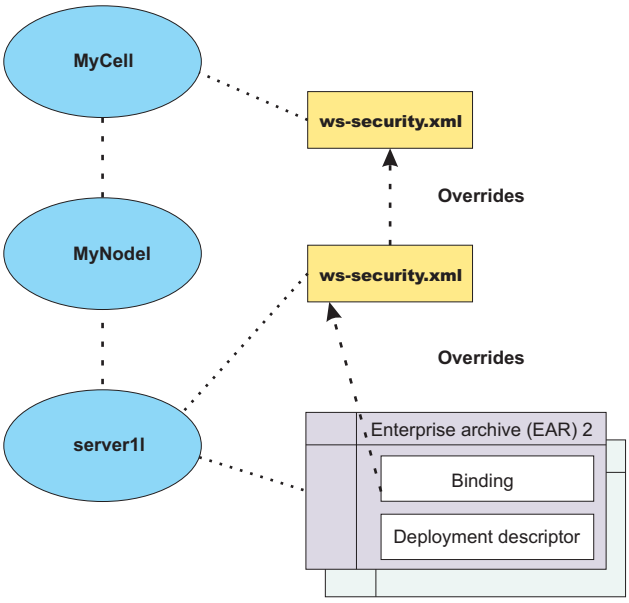
The following figure shows the relationship between the application enterprise archive (EAR) file and the `ws-security.xml` file.



Applications EAR 1 and EAR 2 have specific bindings in the application binding file. However, applications EAR 3 and EAR 4 do not have a binding in the application binding file; it must be referenced to use the default bindings defined in the `ws-security.xml` file. The configuration is resolved by nearest configuration in the hierarchy. For example, there might be key locators named `mykeylocator` that is defined in the application binding file and the server level.

If `mykeylocator` is referenced in the application binding, then the key locator that is defined in the application binding is used. The visibility scope of the data depends upon where the data is defined. If the data is defined in the application binding, then its visibility is scoped to that particular application. If the data is defined on the server level, then the visibility scope is all of the applications deployed on that server. In general, if data is not meant to be shared by other applications, define the configuration in the application binding level.

The following figure shows the relationship of the bindings on the application, server, and cell (Network Deployment only) levels.



General bindings

General bindings are used as the default bindings at the cell level or server level. The general bindings that are shipped with WebSphere Application Server are initially set as the default bindings, but you can choose a different binding as the default, or change the level of binding that should be used as the default, for example, from cell level binding to server level binding.

In version 7.0, there are two types of bindings: application specific bindings, and general bindings. Both types of bindings are supported for WS-Security policy sets. General bindings can be shared across multiple applications and for trust service attachments. There are two types of general bindings: one for service providers and one for service clients. Multiple general bindings can be defined for the provider and also for the client.

Keys: **Version 6 and later applications**

Use keys for XML digital signature and encryption.

There are two predominant kinds of keys used in the current Web services security implementation:

- Public key - such as Rivest Shamir Adleman (RSA) encryption and Digital Signature Algorithm (DSA) encryption
- Secret key - such as triple-strength DES (3DES) encryption

In public key-based signature, a message is signed using the sender private key and is verified using the sender public key. In public key-based encryption, a message is encrypted using the receiver public key and is decrypted using the receiver private key. In secret key-based signature and encryption, the same key is used by both parties.

While the current implementation of Web services security can support both kinds of keys, the format of the message differs slightly between public key-based encryption and secret key-based encryption.

Key locator: **Version 6 and later applications**

A key locator is an abstraction of the mechanism that retrieves the key for digital signature and encryption. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side.

Retrieve keys from one of the following sources, depending upon your implementation:

- Java keystore file
- Database
- Kerberos KDC server (WebSphere Application Server Version 7 using JAX-WS only)
- Trust service can provide a security context token and key (WebSphere Application Server Version 7 using JAX-WS only)

Key locators search for the key using some type of a clue. The following types of clues are supported:

- A string label of the key, which is explicitly passed through the application programming interface (API). The relationship between each key and its name (string label) is maintained inside the key locator.
- The implementation context of the key locator; explicit information is not passed to the key locator. A key locator determines the appropriate key according to the implementation context.

WebSphere Application Server Versions 6 and later support a secret key-based signature called HMAC-SHA1. If you use HMAC-SHA1, the SOAP message does not contain a binary security token. In this case, it is assumed that the key information within the message contains the key name that is used to specify the secret key within the keystore.

Because the key locators support the public key-based signature, the key for verification is embedded in the X.509 certificate as a `<BinarySecurityToken>` element in the incoming message. For example, key locators can obtain the identity of the caller from the context and can retrieve the public key of the caller for response encryption.

This section describes the usage scenarios for key locators.

Signing

The name of the signing key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. The corresponding X.509 certificate also can be returned.

Verification

By default, WebSphere Application Server Versions 6 and later supports the following types of key locators:

KeyStoreKeyLocator
: Uses the keystore to retrieve the key that is used for digital signature and verification or encryption and decryption.

X509CertKeyLocator
: Uses an X.509 certificate within a message to retrieve the key for verification or decryption.

SignerCertKeyLocator
: Uses the X.509 certificate within the request message to retrieve the key that is used for encryption in the response message.

Encryption

The name of the encryption key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. On the server side, you can use the SignerCertKeyLocator to retrieve the key for encryption in the response message from the X.509 certificate in the request message.

Decryption

The Web services security specification recommends using the key identifier instead of the key name. However, while the algorithm for computing the identifier for the public keys is defined in Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280, there is no agreed-upon algorithm for the secret keys. Therefore, the current implementation of Web services security uses the identifier only when public key-based encryption is performed. Otherwise, the ordinal key name is used.

When you use public key-based encryption, the value of the key identifier is embedded in the incoming encrypted message. Then, the Web services security implementation searches for all of the keys managed by the key locator and decrypts the message using the key whose identifier value matches the one in the message.

When you use secret key-based encryption, the value of the key name is embedded in the incoming encrypted message. The Web services security implementation asks the key locator for the key with the name that matches the name in the message and decrypts the message using the key.

Trust anchor: **Version 6 and later applications**

A trust anchor specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the SOAP message.

When using WebSphere Application Server with the JAX-RPC programming model, key stores are implemented with the following message points to validate the X.509 certificate that is used for digital signature or XML encryption:

- Request consumer, as defined in the `ibm-webservices-bnd.xmi` file.
- Response consumer, as defined in the `ibm-webservicesclient-bnd.xmi` file when a Web service is acting as a client to another Web service.

For WebSphere Application Server Version 7.0, using JAX-WS, key stores are used by the following message points to validate the X.509 certificate that is used for digital signature or XML encryption:

- Request consumer, as defined in the inbound keys and certificates of the WS-Security bindings.
- Response consumer, as defined in the inbound keys and certificates of the WS-Security bindings when a Web service is acting as a client to another Web service.

Key stores are critical to the integrity of the digital signature validation. If the key stores are tampered with, the result of the digital signature verification is doubtful and compromised. Therefore, it is recommended that you secure the key stores. The binding configuration specified for the consumer must match the binding configuration for the generator.

The trust anchor is defined as java.security.cert.TrustAnchor in the Java CertPath application programming interface (API). The Java CertPath API uses the trust anchor and the certificate store to validate the incoming X.509 certificate that is embedded in the SOAP message. The Web services security implementation in WebSphere Application Server supports this trust anchor. In WebSphere Application Server, the trust anchor is represented as a Java key store object. The type, path, and password of the key store are passed to the implementation through the administrative console or by scripting.

Trusted ID evaluator: **Version 6 and later applications**

A trusted ID evaluator is the mechanism that evaluates whether a given ID name is trusted.

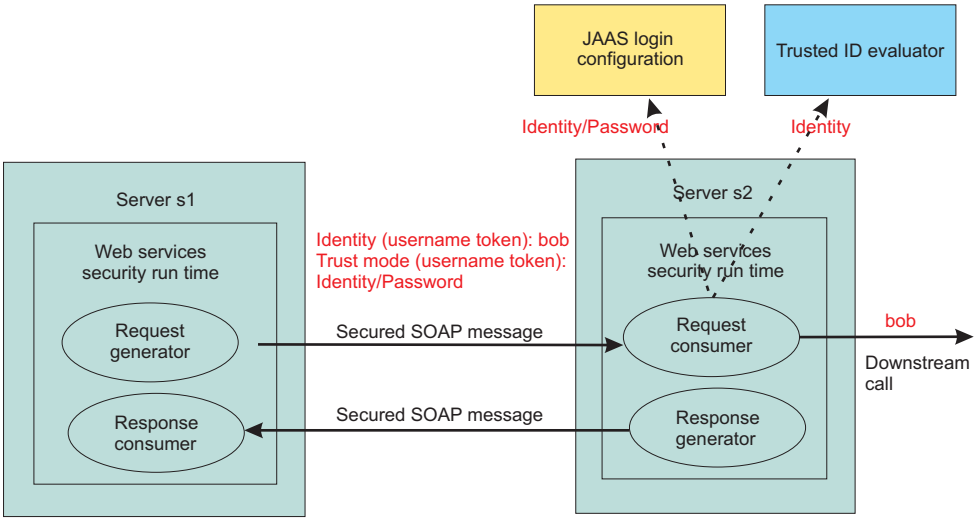
Using the trusted ID evaluator with the JAX-RPC programming model

In the JAX-RPC programming model, the trusted ID evaluator, com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, is an abstraction of the mechanism that evaluates whether a given ID name is trusted. There are two trust modes for validating the trust of the upstream server when using JAX-RPC:

Basic authentication (username token)
The upstream server sends a username token with a user name and password to a downstream server. The consumer or receiver of the message authenticates the username token and validates the trust based upon the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface.

Signature
The upstream server signs the message, which can be any message part such as the SOAP body. The upstream server sends the X.509 token to a downstream server. The consumer or receiver of the message verifies the signature and validates the X.509 token. The identity or the distinguished name from the X.509 token that is used in the digital signature is validated based on the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface. For the X.509 certificate, WebSphere Application Server uses the distinguished name in the certificate as a requester identity.

The following figure demonstrates the identity assertion trust process.



In this figure, server s1 is the upstream server and identity assertion is set up between server s1 and server s2. The s1 server authenticates the identity called *bob*. Server s1 wants to send *bob* to the s2 server with a password. The trust mode is an s1 credential that contains the identity and a password. Server s2 receives the request, authenticates the user using a Java Authentication and Authorization Service (JAAS) login module, and uses the trusted ID evaluator to determine whether to trust the identity. If the identity is trusted, *bob* is used as the caller that invokes the service. If authorization is required, *bob* is the identity that is used for authorization verification.

The identity can be asserted as the RunAs (invocation) identity of the current security context. For example, the Web services gateway authenticates a requester using a secure method such as password

authentication and then sends the requester identity only to a back-end server. You might also use identity assertion for interoperability with another Web services security implementation.

Depending upon the implementation of JAX-RPC, you can use various types of infrastructure to store a list of the trusted IDs, such as:

- Plain text file
- Database
- Lightweight Directory Access Protocol (LDAP) server

The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment. The Web services security implementation invokes the trusted ID evaluator and passes the identity name of the intermediary as a parameter. If the identity is evaluated and deemed trustworthy, the procedure continues. Otherwise, an exception is created and the procedure is stopped.

Using the trusted ID evaluator with the JAX-WS programming model

In the JAX-WS programming model, the same concepts are supported for the trusted ID evaluator, although the implementation is different. For the JAX-WS run time, use the administrative console to select the **Use identity assertion** option on the caller binding panel. This defines the trusted identity token type, and then defines a list of one or more trusted identities. The trusted ID evaluator validates the trusted identity token against the list of trusted identities. For more information about the list of trusted identities, read the topic Changing the order of the callers for a token or message part.

For WebSphere Application Server Version 6.1 and later, the Caller and TrustMethod elements are used to support the requestor login. The requestor sends a message to an intermediary, and the message is dispatched to the service. Based on the security information, the service performs a login for the requestor. In some cases, there are multiple security tokens, so the service has to decide which one to use. When the requestor ID is included as an ID assertion, the service can specify how to trust the intermediary. The following intermediary scenarios are supported:

<BasicAuth, null, null>
The requestor username and password is used for authentication. In this case, authentication is performed with requestor properties, therefore a password is required for authentication.

<Signature, null, null>
The requestor signature is used for authentication.

<IDAssertion, Username, null>
The requestor username (without a password) is used to identify the requestor. The UsernameToken token is used as the ID assertion, therefore no password is required to accompany the username. In this case, the service trusts the intermediary unconditionally.

<IDAssertion, Username, Username>
The requestor username (without a password) is used to identify the requestor, and the username and password of the intermediary is used to authenticate the intermediary. The UsernameToken token, when used to establish trusted identity, always requires a password because the purpose of the token is to establish trust between the intermediary and the service.

<IDAssertion, Username, X509>
The requestor username (without a password) is used to identify the requestor, and the signature of the intermediary is used to authenticate the intermediary. In this case, the trusted identity for the signature of the intermediary must be established using an X.509 certificate.

<IDAssertion, X509, null>
The identity of the requestor is established using an X.509 certificate. In this case, the X.509 certificate from the requestor does not provide a signature to prove possession of the certificate, and therefore the service trusts the intermediary unconditionally.

<IDAssertion, X509, Username>
The identity of the requestor is established using an X.509 certificate, and the username and password of the intermediary is used to authenticate the intermediary. The UsernameToken token, when used to establish trusted identity, always requires a password because the purpose of the token is to establish trust between the intermediary and the service.

<IDAssertion, X509, X509>
The identity of the requestor is established using an X.509 certificate, and the signature of intermediary is used to authenticate the intermediary.

Hardware cryptographic device support for Web Services Security:

Version 6 and later applications

In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

Enabling cryptographic operations on hardware devices

You can enable cryptographic operations on hardware devices. The keys that are used can be stored in a Java keystore file; it is not necessary to store them on the hardware device. The decision to use enable cryptographic operations on hardware devices is made at the server level only, not at the application level.

If cryptographic operations on hardware device is enabled, the Web service security run time first attempts to use the hardware device for cryptographic operations. If the attempt to use the hardware device fails or if the algorithm is not supported by the hardware device, the run time uses a software provider from the security providers list.

Enabling this feature might improve the performance, depending on the hardware device. For more information on how to enable cryptographic operations on hardware devices, see “Configuring hardware cryptographic devices for Web Services Security” on page 500.

Secure keys

Cryptographic keys can be stored on the hardware cryptographic device and never leave the device. These secure keys are confined to the hardware cryptographic device for security considerations rather than performance considerations. The option to select whether to use keys that are stored in a hardware cryptographic device or a Java keystore file can be made at the application level.

If the keystore reference is specified to be a hardware device configuration, the Web services security run time first attempts to obtain the cryptographic algorithm from the hardware device. If the algorithm is not supported or fails, the run time uses a software provider from the security providers list.

See further information about how to enable secure keys, see “Enabling cryptographic keys stored in hardware devices in Web Services Security” on page 501.

Limitations

The hardware cryptographic device support for Web Services Security currently has the following limitations:

- There is no support for a Web services client running as a Java Platform, Enterprise Edition (Java EE) Application Client.
- There is no support for hardware cryptographic devices on iSeries®.

- Only Version 6.1 and later, Web services security applications can take advantage of the hardware cryptographic support.

  Note: Versions 5.*x* and 6.0.*x* Web services security applications can run in a Version 6.1 WebSphere Application Server, but these versions cannot take advantage of the hardware cryptographic support.

Long-term usage of session keys

You can configure WebSphere Application Server to use the hardware keystore, or you can configure the hardware acceleration card to allow the long-term usage of session keys. Session keys might be insecure.

If you are concerned about insecure session keys, configure WebSphere Application Server to use the hardware keystore. See the information about how to enable cryptographic keys that are stored in hardware devices in Web services security.

To configure the hardware acceleration card to allow the long-term usage of session keys, see the manufacturer's documentation for the specific hardware acceleration card. For example:

1. For the nCipher nforce 1600 server Version 2.23.6, follow the nCipher documentation instructions.
2. You can set the `CKNFAST_SECURITY_ASSURANCES_OVERRIDE=longterm` parameter in the `cknfastrc` configuration file. This configuration change eliminates the time limit that is associated with session keys.
3. Follow the documentation for Cipher to restart the nCipher server.
4. Restart WebSphere Application Server.

Default configuration: **Version 6 and later applications**

You can use sample configurations with the administrative console for testing purposes.

The information in the following sections describes sample default bindings, sample general bindings, and samples for key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators. You can develop Web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7, using the Java API for XML-Based Web Services (JAX-WS) programming model. Samples that are provided with WebSphere Application Server differ depending on which programming model you use.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Do not use these sample configurations in a production environment as they are for sample and testing purposes only. To make modifications to these sample configurations, it is recommended that you use the administrative console provided by WebSphere Application Server.

Detailed information on the sample general bindings for the JAX-WS programming model is available in the topic General sample bindings for JAX-WS applications.

Information on configuring default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators for the JAX-RPC programming model is available in the topic Default sample configurations for JAX-RPC.

General sample bindings for JAX-WS applications:

You can use sample bindings with the administrative console for testing purposes.

WebSphere Application Server Version 7.0 includes provider and client sample bindings for testing purposes. In the bindings, the product provides sample values for supporting tokens for different token types, such as the X.509 token, the username token, the LTPA token, and the Kerberos token. The bindings also include sample values for message protection information for token types such as X.509 and secure conversation. Both provider and client sample bindings can be applied to the applications attached with a system policy set, or application policy set, from the default local repository.

This information describes the general sample bindings for the Java API for XML-Based Web Services (JAX-WS) programming model. You can develop Web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7.0, using the Java API for XML-Based Web Services (JAX-WS) programming model. Sample general bindings may differ depending on which programming model you use.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Do not use these provider and client sample bindings in their default state in a production environment. You must modify the bindings to meet your security needs before using them in a production environment by making a copy of the bindings and then modifying the copy. For example, you must change the key and keystore settings to ensure security, and modify the binding settings to match your environment.

One set of general default bindings is shared by the applications to make application deployment easier. The default binding information for the server level can be overridden by the binding information on the application level. The product searches for binding information for an application on the application level before searching the server level.

General client sample bindings

- The sample configuration for signing information generation, called `asymmetric-signingInfoRequest`, contains the following configuration:
  - References the `gen_signkeyinfo` signing key information.
  - The part reference configuration, which contains the transform configuration using the `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
  - The signing key information, `gen_signkeyinfo`, which contains this configuration:
    - The security token reference.
    - The `gen_signx509token` protection token asymmetric signature generator, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` value type for the local part value.
      - Contains the `wss.generate.x509` JAAS login

- The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsigsender.ks, with these characteristics:
  - The keystore type is JKS.
  - The keystore password is client.
  - The alias name of the trusted certificate is soapca.
  - The alias name of the personal certificate is soaprequester.
  - The key password client issued by the intermediary certificate authority Int CA2, which is in turn issued by soapca.

– The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
– The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.

- The sample configuration for signing information generation called symmetric-signingInfoRequest contains the following configuration:
  – References the gen_signsctkeyinfo signing key information.
  – The part reference configuration, which contains the transform configuration using the http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  – The signing key information, gen_signsctkeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, as follows:
      - Contains the Secure Conversation Token Version 1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type as the local part value.
      - Contains wss.generate.sct JAAS login
    - The WS-Trust Callback Handler.
  – The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
  – The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information generation, called asymmetric-encryptionInfoRequest, contains the following configuration:
  – References the gen_enckeyinfo encryption key information.
  – Encryption key information, named gen_enckeyinfo, which contains this configuration:
    - The key identifier.
    - The gen_encx509token protection token asymmetric encryption generator, as follows:
      - Keystore type is JCEKS.
      - Keystore password is client.
      - Alias name of the trusted certificate is soapca.
      - Alias name of the personal certificate is bob.
      - Key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks.
  – The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.

- The sample configuration for encryption information generation, called symmetric-encryptionInfoRequest, contains the following configuration:
  - References the gen_encsctkeyinfo encryption key information.
  - The encryption key information, gen_encsctkeyinfo, which contains this configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The gen_scttoken protection token generator, which contains the following configuration:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains wss.generate.sct JAAS login.
    - The WS-Trust Callback Handler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for signing information consumption, called asymmetric-signingInfoResponse, contains the following configuration:
  - References the con_signkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_signkeyinfo, which contains the following configuration:
    - The con_signx509token protection token asymmetric signature consumer, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.consume.x509 JAAS login.
    - The X.509 Callback Handler, as follows:
      - References a certificate store named DigSigCertStore.
      - References a trusted anchor store named DigSigTrustAnchor.
  - The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information consumption, called symmetric-signingInfoResponse, contains the following configuration:
  - References the con_sctsignkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named con_sctsignkeyinfo, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.

- The con_scttoken protection token consumer, as follows:
  - Contains the Secure Conversation Token v1.3 token type.
  - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
  - Contains the wss.consume.sct JAAS login.
- The WS-SecureConversation Callback Handler.

– The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
– The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.

- The sample configuration for encryption information consumption, called asymmetric-encryptionInfoResponse, which contains the following configuration:
  – References the dec_keyinfo encryption key information.
  – The encryption key information, named dec_keyinfo, which contains the following configuration:
    - The con_encx509token protection token asymmetric encryption consumer, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.consume.x509 JAAS login.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks, with the follow characteristics:
      - The keystore type is JCEKS.
      - The keystore password is client.
      - The alias name of the trusted certificate is soapca.
      - The alias name of the personal certificate is alice.
      - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
  – The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information consumption, called symmetric-encryptionInfoResponse, contains the following configuration:
  – References the dec_sctkeyinfo encryption key information.
  – The encryption key information, named dec_sctkeyinfo, contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The con_scttoken protection token consumer, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains the wss.consume.sct JAAS login.
    - The WS-SecureConversation Callback Handler.
  – The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for authentication token generation, called gen_signkrb5token, contains the following configuration:

  - The custom token type for the Kerberos v5 token, which uses `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` for the local part value.
  - The `wss.generate.KRB5BST` JAAS login.
  - The following custom properties:
    - `com.ibm.wsspi.wssecurity.krbtoken.targetServiceName`, the target Kerberos service name.
    - `com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost`, the host name associated with the target Kerberos service name,

      You must provide the correct values for your environment before using this configuration.
  - The custom Kerberos token callback handler. You must provide the correct values for the Kerberos client principal and password.
- The sample configuration for authentication token generation, called `gen_signltpaproptoken`, contains the following configuration:
  - The token type LTPA propagation token, as follows:
    - Contains `LTPA_PROPAGATION` for the local part value.
    - Contains `http://www.ibm.com/websphere/appserver/tokentype` for the Namespace URI value.
  - Contains the `wss.generate.ltpaProp` JAAS login.
  - Uses the LTPA token callback handler.
- The sample configuration for authentication token generation, called `gen_signltpatoken`, contains the following configuration:
  - The token type of LTPA Token v2.0, as follows:
    - Contains `LTPA_PROPAGATION` for the local part value.
    - Contains `http://www.ibm.com/websphere/appserver/tokentype` for the Namespace URI value.
  - The `wss.generate.ltpa` JAAS login.
  - The LTPA token callback handler.
- The sample configuration for authentication token generation, called `gen_signunametoken`, contains the following configuration:
  - The token type of Username Token v1.0, which uses `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` for the local part value.
  - The `wss.generate.unt` JAAS login.
  - The Username token callback handler, as follows:
    - Contains basic authentication fields. You must provide the correct values for your environment for client principal and password.
    - Contains the following custom properties:
      - `com.ibm.wsspi.wssecurity.token.username.addNonce` for adding the nonce value.
      - `com.ibm.wsspi.wssecurity.token.username.addTimestamp` for adding the time stamp value.

General provider sample bindings

- The sample configuration for signing information consumption, called `asymmetric-signingInfoRequest`, contains the following configuration:
  - References the `con_signkeyinfo` signing key information.
  - The part reference configuration, which uses the transform configuration `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
  - The signing key information, named `con_signkeyinfo`, which contains the following configuration:
    - The `con_signx509token` protection token asymmetric signature consumer, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` value type for the local part value.

        - Contains the wss.consume.x509 JAAS login.
      - The X.509 Callback Handler, as follows:
        - References a certificate store named DigSigCertStore.
        - References a trusted anchor store named DigSigTrustAnchor.
    – The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
    – The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information consumption, called symmetric-signingInfoRequest, contains the following configuration:
    – References the con_sctsignkeyinfo signing key information.
    – The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
    – The signing key information, named con_sctsignkeyinfo, which contains the following configuration:
      - The derived key, as follows:
        - Requires explicit derived key token.
        - WS-SecureConversation as the client label.
        - WS-SecureConversation as the service label.
        - Key length of 16 bytes.
        - Nonce length of 16 bytes.
      - The con_scttoken protection token generator, as follows:
        - Contains the Secure Conversation Token v1.3 token type.
        - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
        - Contains the wss.consume.sct JAAS login.
      - The WS-SecureConversation Callback Handler.
    – The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
    – The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information consumption, called asymmetric-encryptionInfoRequest, contains the following configurations:
    – References the dec_keyinfo encryption key information.
    – The encryption key information, named dec_keyinfo, which contains the following configuration:
      - The con_encx509token protection token asymmetric encryption consumer, as follows:
        - Contains the X.509 V3 Token v1.0 token type.
        - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
        - Contains the wss.consume.x509 JAAS login.
      - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks, with the following characteristics:
        - The keystore type is JCEKS.
        - The keystore password is client.
        - The alias name of the trusted certificate is soapca.
        - The alias name of the personal certificate is bob.
        - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
    – The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information consumption, called symmetric-encryptionInfoRequest, contains the following configuration:

  - References the dec_sctkeyinfo encryption key information.
  - The encryption key information, named dec_sctkeyinfo, which contains the following configuration:
    - The derived key, as follows:
      - Requires explicit derived key token.
      - WS-SecureConversation as the client label.
      - WS-SecureConversation as the service label.
      - Key length of 16 bytes.
      - Nonce length of 16 bytes.
    - The con_scttoken protection token consumer, as follows:
      - Contains the Secure Conversation Token v1.3 token type.
      - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
      - Contains the wss.consume.sct JAAS login.
    - The WS-SecureConversation Callback Handler.
  - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for signing information generation, called asymmetric-signingInfoResponse, contains the following configuration:
  - References the gen_signkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named gen_signkeyinfo, which contains the following configuration:
    - The security token reference.
    - The gen_signx509token protection token asymmetric signature generator, as follows:
      - Contains the X.509 V3 Token v1.0 token type.
      - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
      - Contains the wss.generate.x509 JAAS login.
    - The X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks, with the following characteristics:
      - The keystore type is JKS.
      - The keystore password is client.
      - The alias name of the trusted certificate is soapca.
      - The alias name of the personal certificate is soapprovider.
      - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
  - The signature method http://www.w3.org/2000/09/xmldsig#rsa-sha1.
  - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for signing information generation, called symmetric-signingInfoResponse, contains the following configuration:
  - References the gen_signsctkeyinfo signing key information.
  - The part reference configuration, which uses the transform configuration http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
  - The signing key information, named gen_signsctkeyinfo, which contains the following configuration:
    - The security token reference.
    - The derived key, as follows:
      - Requires explicit derived key token.

        - WS-SecureConversation as the client label.
        - WS-SecureConversation as the service label.
        - Key length of 16 bytes.
        - Nonce length of 16 bytes.
      - The gen_scttoken protection token generator, as follows:
        - Contains the Secure Conversation Token v1.3 token type.
        - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
        - Contains the wss.generate.sct JAAS login.
      - The WS-Trust Callback Handler.
    - The signature method http://www.w3.org/2000/09/xmldsig#hmac-sha1.
    - The canonicalization method http://www.w3.org/2001/10/xml-exc-c14n#.
- The sample configuration for encryption information generation, called asymmetric-encryptionInfoResponse, contains the following configuration:
    - References the gen_enckeyinfo encryption key information.
    - The encryption key information, named gen_enckeyinfo, contains the following configuration
      - The key identifier.
      - The gen_encx509token protection token asymmetric encryption generator, as follows:
        - Contains the X.509 V3 Token v1.0 token type.
        - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 value type for the local part value.
        - Contains the wss.generate.x509 JAAS login.
      - Uses X.509 Callback Handler. The callback handler calls the custom keystore in ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks, with the following characteristics:
        - The keystore type is JCEKS.
        - The keystore password is client.
        - The alias name of the trusted certificate is soapca.
        - The alias name of the personal certificate is alice.
        - The key password client issued by intermediary certificate authority Int CA2, which is in turn issued by soapca.
    - The key encryption method http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- The sample configuration for encryption information generation, called symmetric-encryptionInfoResponse, contains the following configuration:
    - References the gen_encsctkeyinfo encryption key information.
    - The encryption key information, named gen_encsctkeyinfo, contains the following configuration:
      - The security token reference.
      - The derived key, as follows:
        - Requires explicit derived key token.
        - WS-SecureConversation as the client label.
        - WS-SecureConversation as the service label.
        - Key length of 16 bytes.
        - Nonce length of 16 bytes.
      - The gen_scttoken protection token generator, as follows:
        - Contains the Secure Conversation Token v1.3 token type.

        - Contains the http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct value type for the local part value.
        - Contains the wss.generate.sct JAAS login.
      - The WS-Trust Callback Handler.
    - The data encryption method http://www.w3.org/2001/04/xmlenc#aes128-cbc.
- The sample configuration for authentication token consumption, called con_krb5token, contains the following configuration:
  - The custom token type for Kerberos v5 token, which uses http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ for the local part value.
  - The wss.consume.KRB5BST JAAS login.
  - The custom Kerberos token callback handler.
- The sample configuration for authentication token consumption, called con_ltpaproptoken, contains the following configuration:
  - The token type LTPA propagation token.
  - The wss.consume.ltpaProp JAAS login.
  - The LTPA token callback handler.
- The sample configuration for authentication token consumption, called con_ltpatoken, contains the following configuration:
  - The token type LTPA Token v2.0, with the following characteristics:
    - Contains LTPAv2 for the local part value.
    - Contains http://www.ibm.com/websphere/appserver/tokentype for the Namespace URI value.
  - The wss.consume.ltpa JAAS login
  - The LTPA token callback handler.
- The sample configuration for authentication token consumption, called con_unametoken, contains the following configuration:
  - Token type Username Token v1.0, which uses http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken for the local part value.
  - The wss.consume.unt JAAS login.
  - The Username token callback handler, with the following custom properties:
    - com.ibm.wsspi.wssecurity.token.username.verifyNonce for verifying the nonce value.
    - com.ibm.wsspi.wssecurity.token.username.verifyTimestamp for verifying the time stamp value.

Default sample configurations for JAX-RPC: **Version 6 and later applications**

Use sample configurations with the administrative console for testing purposes.

This information describes the sample default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluators for WebSphere Application Server using the API for XML-based RPC (JAX-RPC) programming model. You can develop Web services using the Java API for XML-based RPC (JAX-RPC) programming model, or for WebSphere Application Server Version 7, using the Java API for XML-Based Web Services (JAX-WS) programming model. Sample default bindings, key stores, key locators, collection certificate store, trust anchors, and trusted ID evaluator may differ depending on which programming model you use.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based

annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Do not use these configurations in a production environment as they are for sample and testing purposes only. To make modifications to these sample configurations, it is recommended that you use the administrative console provided by WebSphere Application Server.

For a Web services security-enabled application, you must correctly configure a deployment descriptor and a binding. In WebSphere Application Server, one set of default bindings is shared by the applications to make application deployment easier. The default binding information for the server level can be overridden by the binding information on the application level. The Application Server searches for binding information for an application on the application level before searching the server level.

The following sample configurations are for WebSphere Application Server using the API for XML-based RPC (JAX-RPC) programming model.

Default generator binding

WebSphere Application Server provides a sample set of default generator bindings. The default generator bindings contain both signing information and encryption information.

The sample signing information configuration is called `gen_signinfo` and contains the following configurations:

- Uses the following algorithms for the `gen_signinfo` configuration:
  - Signature method: `http://www.w3.org/2000/09/xmldsig#rsa-sha1`
  - Canonicalization method: `http://www.w3.org/2001/10/xml-exc-c14n#`
- References the `gen_signkeyinfo` signing key information. The following information pertains to the `gen_signkeyinfo` configuration:
  - Contains a part reference configuration that is called `gen_signpart`. The part reference is not used in default binding. The signing information applies to all of the Integrity or Required Integrity elements within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the `gen_signpart` configuration:
    - Uses the transform configuration called `transform1`. The following transforms are configured for the default signing information:
      - Uses the `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm
      - Uses the `http://www.w3.org/2000/09/xmldsig#sha1` digest method
  - Uses the security token reference, which is the configured default key information.
  - Uses the `SampleGeneratorSignatureKeyStoreKeyLocator` key locator. For more information on this key locator, see “Default sample configurations for JAX-RPC” on page 108.
  - Uses the `gen_signtgen` token generator, which contains the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the `gen_signtgen_vtype` value type URI.
    - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509` value type local name value.
  - Uses X.509 Callback Handler. The callback handler calls the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store.
    - The key store password is `client`.
    - The alias name of the trusted certificate is `soapca`.
    - The alias name of the personal certificate is `soaprequester`.

- The key password client issued by intermediary certificate authority `Int CA2`, which is in turn issued by `soapca`.

The sample encryption information configuration is called `gen_encinfo` and contains the following configurations:

- Uses the following algorithms for the `gen_encinfo` configuration:
  - Data encryption method: `http://www.w3.org/2001/04/xmlenc#tripledes-cbc`
  - Key encryption method: `http://www.w3.org/2001/04/xmlenc#rsa-1_5`
- References the `gen_enckeyinfo` encryption key information. The following information pertains to the `gen_enckeyinfo` configuration:
  - Uses the key identifier as the default key information.
  - Contains a reference to the `SampleGeneratorEncryptionKeyStoreKeyLocator` key locator. For more information on this key locator, see "Default sample configurations for JAX-RPC" on page 108.
  - Uses the `gen_signtgen` token generator, which has the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the `gen_enctgen_vtype` value type URI.
    - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509` value type local name value.
  - Uses X.509 Callback Handler. The callback handler calls the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` key store.
    - The key store password is `storepass`.
    - The secret key `CN=Group1` has an alias name of `Group1` and a key password of `keypass`.
    - The public key `CN=Bob, O=IBM, C=US` has an alias name of `bob` and a key password of `keypass`.
    - The private key `CN=Alice, O=IBM, C=US` has an alias name of `alice` and a key password of `keypass`.

Default consumer binding

WebSphere Application Server provides a sample set of default consumer binding. The default consumer binding contain both signing information and encryption information.

The sample signing information configuration is called `con_signinfo` and contains the following configurations:

- Uses the following algorithms for the `con_signinfo` configuration:
  - Signature method: `http://www.w3.org/2000/09/xmldsig#rsa-sha1`
  - Canonicalization method: `http://www.w3.org/2001/10/xml-exc-c14n#`
- Uses the `con_signkeyinfo` signing key information reference. The following information pertains to the `con_signkeyinfo` configuration:
  - Contains a part reference configuration that is called `con_signpart`. The part reference is not used in default binding. The signing information applies to all of the Integrity or RequiredIntegrity elements within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the `con_signpart` configuration:
    - Uses the transform configuration called `reqint_body_transform1`. The following transforms are configured for the default signing information:
      - Uses the `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm.
      - Uses the `http://www.w3.org/2000/09/xmldsig#sha1` digest method.
  - Uses the security token reference, which is the configured default key information.
  - Uses the `SampleX509TokenKeyLocator` key locator. For more information on this key locator, see "Default sample configurations for JAX-RPC" on page 108.

  - References the con_signtcon token consumer configuration. The following information pertains to the con_signtcon configuration:
    - Uses the X.509 Token Consumer, which is configured as the consumer for the default signing information.
    - Contains the signtconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST that references the following information:
    - Trust anchor: SampleClientTrustAnchor
    - Collection certificate store: SampleCollectionCertStore

The encryption information configuration is called con_encinfo and contains the following configurations:

- Uses the following algorithms for the con_encinfo configuration:
  - Data encryption method: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - Key encryption method: http://www.w3.org/2001/04/xmlenc#rsa-1_5
- References the con_enckeyinfo encryption key information. This key actually decrypts the message. The following information pertains to the con_enckeyinfo configuration:
  - Uses the key identifier, which is configured as the key information for the default encryption information.
  - Contains a reference to the SampleConsumerEncryptionKeyStoreKeyLocator key locator. For more information on this key locator, see “Default sample configurations for JAX-RPC” on page 108.
  - References the con_enctcon token consumer configuration. The following information pertains to the con_enctcon configuration:
    - Uses the X.509 token consumer, which is configured for the default encryption information.
    - Contains the enctconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST.

Sample key store configurations

The following sample key stores are for testing purposes only; do not use these key stores in a production environment:

- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks
  - The key store format is JKS.
  - The key store password is client.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soaprequester alias name and a client key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks
  - The key store format is JKS.
  - The key store password is server.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soapprovider alias name and a server key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks
  - The key store format is JCEKS.

  - The key store password is `storepass`.
  - The `CN=Group1` DES secret key has a `Group1` alias name and a `keypass` key password.
  - The `CN=Bob, O=IBM, C=US` public key has a `bob` alias name and a `keypass` key password.
  - The `CN=Alice, O=IBM, C=US` private key has a `alice` alias name and a `keypass` key password.
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks`
  - The key store format is `JCEKS`.
  - The key store password is `storepass`.
  - The `CN=Group1` DES secret key has a `Group1` alias name and a `keypass` key password.
  - The `CN=Bob, O=IBM, C=US` private key has a `bob` alias name and a `keypass` key password.
  - The `CN=Alice, O=IBM, C=US` public key has a `alice` alias name and a `keypass` key password.
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`
  - The intermediary certificate is signed by `soapca` and it signs both the `soaprequester` and the `soapprovider`.

Sample key locators

Key locators are used to locate the key for digital signature, encryption, and decryption. For information on how to modify these sample key locator configurations, see the following articles:

- "Configuring the key locator using JAX-RPC for the generator binding on the application level" on page 463
- "Configuring the key locator using JAX-RPC for the consumer binding on the application level" on page 470
- "Configuring the key locator using JAX-RPC on the server level" on page 472

Version 5.x application

SampleClientSignerKey
This key locator is used by the request sender for a Version 5.*x* application to sign the SOAP message. The signing key name is `clientsignerkey`, which is referenced in the signing information as the signing key name.

SampleServerSignerKey
This key locator is used by the response sender for a Version 5.*x* application to sign the SOAP message. The signing key name is `serversignerkey`, which can be referenced in the signing information as the signing key name.

SampleSenderEncryptionKeyLocator
This key locator is used by the sender for a Version 5.*x* application to encrypt the SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` key store and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator key store key locator. The implementation is configured for the DES secret key. To use asymmetric encryption (RSA), you must add the appropriate RSA keys.

SampleReceiverEncryptionKeyLocator
This key locator is used by the receiver for a Version 5.*x* application to decrypt the encrypted SOAP message. The implementation is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` key store and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator key store key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). To use RSA, you must add the private key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`.

SampleResponseSenderEncryptionKeyLocator
This key locator is used by the response sender for a Version 5.*x* application to encrypt the SOAP response message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/`

enc-receiver.jceks key store and the com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator key store key locator. This key locator maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key alice, and you must change WebSphere Application Server to the appropriate user. The SampleResponseSenderEncryptionKeyLocator key locator also can set a default key for encryption. By default, this key locator is configured to use public key alice.

Version 6 and later applications

SampleGeneratorSignatureKeyStoreKeyLocator
This key locator is used by generator to sign the SOAP message. The signing key name is SOAPRequester, which is referenced in the signing information as the signing key name. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks key store and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key store key locator.

SampleConsumerSignatureKeyStoreKeyLocator
This key locator is used by the consumer to verify the digital signature in the SOAP message. The signing key is SOAPProvider, which is referenced in the signing information. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks key store and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key store key locator.

SampleGeneratorEncryptionKeyStoreKeyLocator
This key locator is used by the generator to encrypt the SOAP message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks key store and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key store key locator.

SampleConsumerEncryptionKeyStoreKeyLocator
This key locator is used by the consumer to decrypt an encrypted SOAP message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks key store and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key store key locator.

SampleX509TokenKeyLocator
This key locator is used by the consumer to verify a digital certificate in an X.509 certificate. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks key store and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key store key locator.

Sample collection certificate store

Collection certificate stores are used to validate the certificate path. For information on how to modify this sample collection certificate store, see the following articles:

- “Configuring the collection certificate store for the generator binding on the application level” on page 481
- “Configuring the collection certificate store for the consumer binding on the application level” on page 490
- “Configuring the collection certificate on the server level” on page 492

SampleCollectionCertStore
This collection certificate store is used by the response consumer and the request generator to validate the signer certificate path.

Sample trust anchors

Trust anchors are used to validate the trust of the signer certificate. For information on how to modify the sample trust anchor configurations, see the following articles:

- “Configuring trust anchors for the generator binding on the application level” on page 474

- “Configuring trust anchors for the consumer binding on the application level” on page 478
- “Configuring trust anchors on the server level” on page 480

Version 5.x application

SampleClientTrustAnchor
This trust anchor is used by the response consumer to validate the signer certificate. This trust anchor is configure to access the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store.

SampleServerTrustAnchor
This trust anchor is used by the request consumer to validate the signer certificate. This trust anchor is configure to access the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` key store.

Sample trusted ID evaluators

Trusted ID evaluators are used to establish trust before asserting the identity in identity assertion. For information on how to modify the sample trusted ID evaluator configuration, see “Configuring trusted ID evaluators on the server level” on page 494.

SampleTrustedIDEvaluator
This trusted ID evaluator uses the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl implementation. The default implementation of com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator contains a list of trusted identities. This list, which is used for identity assertion, defines the key name and value pair for the trusted identity. The key name is in the form trustedId_* and the value is the trusted identity. For more information, see the example in “Configuring trusted ID evaluators on the server level” on page 494.

Complete the following steps to define this information for the server level in the administrative console:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security.**
3. Under Additional properties, click **Trusted ID evaluators** > **SampleTrustedIDEvaluator**.

Default implementations of the Web services security service provider programming interfaces:

Version 6 and later applications

This information describes the default implementations of the service provider interfaces (SPI) for Web services security within WebSphere Application Server. The default implementation classes and their functionality for both the JAX-RPC run time and the JAX-WS run time are discussed. You can use this information to create or modify the Web services security binding configuration.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

The default implementations of the service provider interfaces for WebSphere Application Server Version 5.x are not described in this document. Instead, see “Securing Web services for Version 5.x applications based on WS-Security” on page 502 for the Version 5.x implementations that are deprecated in Version 6.0.x and later.

Default implementations for the JAX-RPC run time

com.ibm.wsspi.wssecurity.token.X509TokenGenerator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. It is responsible for creating the X.509 token object from the X.509 certificate, which is returned by the com.ibm.wsspi.wssecurity.auth.callback.{X509,PKCS7,PkiPath}CallbackHandler interface. Encode the token using the base 64 format and insert its XML representation into the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface and it retrieves the X.509 certificate from the keystore file.

com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. It is responsible for creating the username token object from user name and password that is returned by a javax.security.auth.callback.CallbackHandler implementation such as the following callback handler:

com.ibm.wsspi.wssecurity.auth.callback{GUIPrompt,NonPrompt,StdinPrompt}CallbackHandler.

It also inserts the XML representation of the token into the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class retrieves the keys from the keystore files for digital signature and encryption.

com.ibm.wsspi.wssecurity.token.X509TokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the X.509 token from the binary security token. This class decodes the Base64 encryption within the X.509 token and then invokes the system.wssecurity.X509BST Java Authentication and Authorization Service (JAAS) Login Configuration with the com.ibm.wsspi.wssecurity.auth.module.X509LoginModule login module to validate the X.509 token. An object of the com.ibm.wsspi.wssecurity.auth.token.X509Token is created for the validated X.509 token and stored in JAAS Subject.

com.ibm.wsspi.wssecurity.token.IDAssertionUsernameTokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the username token for identity assertion (IDAssertion), which does not have a password element. This interface invokes the system.wssecurity.IDAssertionUsernameToken JAAS login configuration with the com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule login module to validate the IDAssertion user name token. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule
This class implements the javax.security.auth.spi.LoginModule interface and checks whether the username value is not empty. The login module assumes that the UsernameToken is valid if the username value is not empty.

com.ibm.wsspi.wssecurity.token.LTPATokenGenerator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. This class is responsible for Base 64 encoding the LTPA token object obtained from the com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler callback handler. The object is inserted into the Web services security header within the SOAP message, if necessary.

com.ibm.wsspi.wssecurity.token.LTPATokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the LTPA token from the binary security token, and decodes the Base64 encoding within the LTPA token. An object of the com.ibm.wsspi.wssecurity.auth.token.LTPAToken class is created for the validated LTPA token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.auth.module.X509LoginModule
This class implements the javax.security.auth.spi.LoginModule interface and validates the X.509 Certificate based on the trust anchor and the collection certification store configuration.

com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. This class processes the username token, extracts the user name and password, and then invokes the system.wssecurity.UsernameToken JAAS login configuration using the com.ibm.wsspi.wssecurity.auth.module.UsernameLoginModule login module to validate the user name and password. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class is used to retrieve a public key from a X.509 certificate. The X.509 certificate is stored in the X.509 token (com.ibm.wsspi.wssecurity.auth.token.X509Token) in the JAAS Subject. The X.509 token is created by the X.509 Token Consumer (com.ibm.wsspi.wssecurity.tokenX509TokenConsumer).

com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator
The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side. This class is used to retrieve a public key from the X.509 certificate of the request signer and encrypt the response. You can use this key locator in the response generator binding configuration only.

Note: This implementation assumes that only one signer certificate is used in the request.

com.ibm.wsspi.wssecurity.auth.token.UsernameToken
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the username token.

com.ibm.wsspi.wssecurity.auth.token.X509Token
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the X.509 binary security token (X.509 certificate).

com.ibm.wsspi.wssecurity.auth.token.LTPAToken
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class as a wrapper to the LTPA token that is extracted from the binary security token.

com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface and is responsible for creating a certificate and binary data with or without a certificate revocation list (CRL) using the PKCS#7 encoding. The certificate and the binary data is passed back to the

com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate and binary data without a CRL using the PkiPath encoding. The certificate and binary data is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate from the keystore file. The X.509 token certificate is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
This implementation generates a Lightweight Third Party Authentication (LTPA) token in the Web services security header as a binary security token. If basic authentication data is defined in the application binding file, it is used to perform a login, to extract the LTPA token from the WebSphere Application Server credentials, and to insert the token in the Web services security header. Otherwise, it extracts the LTPA security token from the invocation credentials (run as identity) and inserts the token in the Web services security header.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This implementation reads the basic authentication data from the application binding file. You might use this implementation on the server side to generate a username token.

com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
This implementation presents you with a login prompt to gather the basic authentication data. Use this implementation on the client side only.

com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
This implementation collects the basic authentication data using a standard in (stdin) prompt. Use this implementation on the client side only.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator
This interface is used to evaluate the level of trust for identity assertion. The default implementation is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, which enables you to define a list of trusted identities.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl
This default implementation enables you to define a list of trusted identities for identity assertion.

com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorException
This exception class is used by an implementation of the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator to communicate the exception and errors to the Web services security run time.

Default implementations for the JAX-WS run time

com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenGenerator
This implementation invokes the JAAS CallbackHandler and JAAS login configuration that are specified in the binding to create the SecurityToken at run time on the outbound SOAP message.

com.ibm.websphere.wssecurity.callbackhandler.X509GenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and retrieves the X.509 certificate. The following properties may be specified:

- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed. This property takes a boolean value, and the default value is false.

- com.ibm.wsspi.wssecurity.token.cert.useRequestorCert. This property takes a boolean value, and the default value is false.

com.ibm.ws.wssecurity.wssapi.token.impl.X509GenerateLoginModule
The wss.generate.x509 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.X509GenerateLoginModule. X509GenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML Username token structure, and also a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7GenerateLoginModule
The wss.generate.pkcs7 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7GenerateLoginModule. PKCS7GenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathGenerateLoginModule
The wss.generate.pkiPath JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathGenerateLoginModule. PkiPathGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the binding configuration and user name and password authentication data. The following properties may be specified. These properties take a boolean value, and the default value is false.

- com.ibm.wsspi.wssecurity.token.username.addNonce
- com.ibm.wsspi.wssecurity.token.username.addTimestamp
- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed
- com.ibm.wsspi.wssecurity.token.IDAssertion.useRunAsIdentity
- com.ibm.wsspi.wssecurity.token.IDAssertion.sendRealm
- com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm

com.ibm.ws.wssecurity.wssapi.token.impl.UNTGenerateLoginModule
The wss.generate.unt JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl. UNTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML Username token structure and also a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has a the value of true, the generated username token does not contain a password. When com.ibm.wsspi.wssecurity.token.IDAssertion.sendRealm has the value of true, the user name is qualified by the local realm name. When com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm has the value of true, the user name field contains both the user name and a registry-dependent unique identifier for the user. Both the user name and the unique identifier are qualified by the local realm name.

com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the Kerberos user name and password, along with other binding configuration properties. The following properties may be specified. The properties take a string that specifies the target service name as part of a service principal name (SPN), in the form of service_name/host_name@Kerberos_realm_name.

- com.ibm.wsspi.wssecurity.krbtoken.targetServiceName

- com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost
- com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm

com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule
The wss.generate.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. KRBGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule
The wss.generate.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. DKTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time when the **Requires derived keys** option is enabled.

com.ibm.websphere.wssecurity.callbackhandler.LTPAGenerateCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the user name and password binding data if they are specified.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAGenerateLoginModule
The wss.generate.ltpa JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAGenerateLoginModule. LTPAGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. The security token contains an LTPA token that is generated from the user name and password if they are defined in the binding data, or the LTPA authentication token from the RunAs Subject, in that order.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationGenerateLoginModule
The wss.generate.ltpaProp JAAS system login configuration contains com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationGenerateLoginModule. LTPAPropagationGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time. The security token contains the serialized RunAs Subject.

com.ibm.ws.wssecurity.impl.auth.callback.WSTrustCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves security context token configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule
The wss.generate.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. SCTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the security context token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule
The wss.generate.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTGenerateLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTGenerateLoginModule. DKTGenerateLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for generating an XML token structure and a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the token at run time when the **Requires derived keys** option is enabled.

com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer
This implementation invokes the JAAS CallbackHandler and JAAS login configuration that are specified in the binding to extract the security token from the inbound SOAP message and to create the SecurityToken object at run time.

com.ibm.websphere.wssecurity.callbackhandler.X509ConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on SOAP message inbound to retrieve the trust store and certificate file information that are required to validate the X.509 certificate.

com.ibm.ws.wssecurity.wssapi.token.impl.X509ConsumeLoginModule
The wss.consume.x509 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.X509ConsumeLoginModule. X509ConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7ConsumeLoginModule
The wss.consume.pkcs7 JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PKCS7ConsumeLoginModule PKCS7ConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathConsumeLoginModule
The wss.consume.pkiPath JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.PkiPathConsumeLoginModule. PkiPathConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the X.509 certificate. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the X.509 token at run time.

com.ibm.websphere.wssecurity.callbackhandler.UNTConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on SOAP message inbound to retrieve binding configuration data. The following properties may be specified. These properties take a boolean value and the default value is false.

- com.ibm.wsspi.wssecurity.token.username.verifyTimestamp
- com.ibm.wsspi.wssecurity.token.username.verifyNonce
- com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed
- com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm

com.ibm.ws.wssecurity.wssapi.token.impl.UNTConsumeLoginModule
The wss.consume.unt JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.UNTConsumeLoginModule. UNTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the username token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the username token at run time. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has the value of false, UNTConsumeLoginModule validates the username and password against the local user registry. An incorrect user name or incorrect or missing password will cause the token validation to fail. When com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed has a value of true, and com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm has a value of false, the user name is validated against the local user registry. There should be no password in the username token. When both com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed and com.ibm.wsspi.wssecurity.token.IDAssertion.trustedRealm have a value of true, the user name

field must contain a realm-qualified user name and unique user identifier data, and the realm must be one of the trusted realms in the multiple security domain inbound trust configuration.

com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the inbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsu
The wss.consume.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. KRBConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the Kerberos AP_REQ token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the AP_REQ token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule
The wss.consume.KRB5BST JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.KRBConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. DKTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving the derived key when a derived key is required.

com.ibm.websphere.wssecurity.callbackhandler.LTPAConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the inbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAConsumeLoginModule
The wss.consume.ltpa JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAConsumeLoginModule. LTPAConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the LTPA v2 or LTPA token. It creates a com.ibm.websphere.wssecurity.wssapi.token.SecurityToken that represents the LTPA v2 or LTPA token at run time.

com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationConsumeLoginModule
The wss.consume.ltpaProp JAAS system login configuration contains the class com.ibm.ws.wssecurity.wssapi.token.impl.LTPAPropagationConsumeLoginModule. LTPAPropagationConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving, deserializing, and validating the propagation token and reconstructing the security context.

com.ibm.ws.wssecurity.impl.auth.callback.SCTConsumeCallbackHandler
This class implements the javax.security.auth.callback.CallbackHandler interface on the outbound SOAP message, and it retrieves the binding configuration data.

com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule
The wss.consume.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. SCTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving and validating the security context token.

com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule
The wss.consume.sct JAAS system login configuration contains the classes com.ibm.ws.wssecurity.wssapi.token.impl.SCTConsumeLoginModule, and com.ibm.ws.wssecurity.wssapi.token.impl.DKTConsumeLoginModule. DKTConsumeLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for retrieving the derived key when a derived key is required.

com.ibm.ws.wssecurity.impl.auth.module.PreCallerLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.PreCallerLoginModule. PreCallerLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for validating whether it has received any security token that may be used to establish caller identity or trusted identity.

com.ibm.ws.wssecurity.impl.auth.module.UNTCallerLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.UNTCallerLoginModule. UNTCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. UNTCallerLoginModule also determines if the user identity is authorized to make an identity assertion if the username is configured to be a trusted identity, or if there is exactly one caller identity if the username token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.X509CallerLoginModule
The wss.caller JAAS system login configuration contains com.ibm.ws.wssecurity.impl.auth.module.X509CallerLoginModule. X509CallerLoginModule implements the javax.security.auth.spi.LoginModule interface. X509CallerLoginModule checks to see if the user identity is authorized to make an identity assertion if the X509 token is configured to be a trusted identity, or if there is exactly one caller identity if the X509 token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.LTPACallerLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.LTPACallerLoginModule. LTPACallerLoginModule implements the javax.security.auth.spi.LoginModule interface. LTPACallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the LTPA token is configured to be a trusted identity, or if there is exactly one caller identity if the LTPA token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.LTPAPropagationCallerLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.LTPAPropagationCallerLoginModule. LTPAPropagationCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. LTPAPropagationCallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the propagation token is configured to be a trusted identity, or if there is exactly one caller identity if the propagation token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.KRBCallerLoginModule
The wss.caller JAAS system login configuration contains com.ibm.ws.wssecurity.impl.auth.module.KRBCallerLoginModule. KRBCallerLoginModule implements the javax.security.auth.spi.LoginModule interface. KRBCallerLoginModule also checks to see if the user identity is an authorized to make an identity assertion if the Kerberos token is configured to be a trusted identity, or if there is exactly one caller identity if the Kerberos token is configured to be a caller identity. It sets the validated caller and trusted identity into the shared state.

com.ibm.ws.wssecurity.impl.auth.module.WSWSSLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.wssecurity.impl.auth.module.WSWSSLoginModule. WSWSSLoginModule implements the javax.security.auth.spi.LoginModule interface and is responsible for asserting the caller identity to the ltpaLoginModule and the wsMapDefaultInboundLoginModule to establish the caller security context.

com.ibm.ws.security.server.lm.ltpaLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.security.server.lm.ltpaLoginModule.

com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule
The wss.caller JAAS system login configuration contains the class com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule.

XML digital signature

Version 6 and later applications

XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

XML digital signature does not introduce new cryptographic algorithms. WebSphere Application Server uses XML digital signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method.

XML canonicalization (c14n) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. For example, although their octet representations are different, the following examples are identical:

- `<person first="John" last="Smith"/>`
- `<person last="Smith" first="John"></person>`

C14n is a process that is used to canonicalize XML information. Select an appropriate c14n algorithm because the information that is canonicalized is dependent upon this algorithm. One of the major c14n algorithms, Exclusive XML Canonicalization, canonicalizes the character encoding scheme, attribute order, namespace declarations, and so on. The algorithm does not canonicalize white space outside tags, namespace prefixes, or data type representation.

XML signature in the Web Services Security-Core specification

The Web Services Security-Core (WSS-Core) specification defines a standard way for SOAP messages to incorporate an XML signature. You can use almost all of the XML signature features in WSS-Core except enveloped signature and enveloping signature. However, WSS-Core has some recommendations such as exclusive canonicalization for the c14n algorithm and some additional features such as SecurityTokenReference and KeyIdentifier.

The KeyIdentifier is the value of the SubjectKeyIdentifier field within the X.509 certificate. For more information on the KeyIdentifier, see ″Reference to a Subject Key Identifier″ within the OASIS Web Services Security X.509 Certificate Token Profile documentation.

By including XML signature in SOAP messages, the following issues are realized:

Message integrity
A message receiver can confirm that attackers or accidents have not altered parts of the message after these parts are signed by a key.

Authentication
You can assume that a valid signature is *proof of possession*. A message with a digital certificate that is issued by a certificate authority and a signature in the message that is validated successfully by a public key in the certificate, is proof that the signer has the corresponding private key. The receiver can authenticate the signer by checking the trustworthiness of the certificate.

Collection certificate store

Version 6 and later applications

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

A collection certificate store is used when WebSphere Application Server is processing a received SOAP message. For JAX-RPC applications, this collection is configured in the Request Consumer Service Configuration Details section of the binding file for servers and in the Response Consumer Configuration section of the binding file for clients. You can configure these two sections using one of the assembly tools provided by WebSphere Application Server. See the assembly tools information in the topic Assembly tools.

For JAX-WS applications, this collection is configured using the administrative console in the Keys and certificates panel of the WS-Security policy set bindings.

A collection certificate store is one kind of certificate store. A certificate store is defined as javax.security.cert.CertStore in the Java CertPath application programming interface (API). The Java CertPath API defines the following types of certificate stores:

Collection certificate store
A collection certificate store accepts the certificates and CRLs as Java collection objects.

Lightweight Directory Access Protocol certificate store
The Lightweight Directory Access Protocol (LDAP) certificate store accepts certificates and CRLs as LDAP entries.

The CertPath API uses the certificate store and the trust anchor to validate the incoming X.509 certificate that is embedded in the SOAP message. The Web services security implementation in the WebSphere Application Server supports the collection certificate store. Each certificate and CRL is passed as an encoded file.

Certificate revocation list

Version 6 and later applications

A *certificate revocation list* is a time-stamped list of certificates that have been revoked by a certificate authority (CA).

A certificate that is found in a certificate revocation list (CRL) might not be expired, but is no longer trusted by the certificate authority that issued the certificate. The certificate authority creates the CRL that contains the serial number and issuing CA distinguished name of the certificate that has been revoked. The CA might add the certificate to the certificate revocation list if it believes that the client certificate is compromised. The certificate revocation list is maintained and issued by the certificate authority.

XML encryption

Version 6 and later applications

XML encryption is a specification that was developed by World Wide Web (WWW) Consortium (W3C) in 2002 and that contains the steps to encrypt data, the steps to decrypt encrypted data, the XML syntax to represent encrypted data, the information to be used to decrypt the data, and a list of encryption algorithms, such as triple DES, AES, and RSA.

You can apply XML encryption to an XML element, XML element content, and arbitrary data, including an XML document. For example, suppose that you need to encrypt the <CreditCard> element that is shown in example 1.

Example 1: Sample XML document

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Example 2: XML document with a common secret key

Example 2 shows the XML document after encryption. The <EncryptedData> element represents the encrypted <CreditCard> element. The <EncryptionMethod> element describes the applied encryption algorithm, which is triple DES in this example. The <KeyInfo> element contains the information that is needed to retrieve a decryption key, which is a <KeyName> element in this example. The <CipherValue> element contains the cipher text that is obtained by serializing and encrypting the <CreditCard> element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
       Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
       <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
         <KeyName>John Smith</KeyName>
       </KeyInfo>
       <CipherData>
         <CipherValue>ydUNqHkMrD...</CipherValue>
       </CipherData>
  </EncryptedData>
</PaymentInfo>
```

Example 3: XML document encrypted with the public key of the recipient

In example 2, it is assumed that both the sender and recipient have a common secret key. If the recipient has a public and private key pair, which is commonly the case, the <CreditCard> element can be encrypted as shown in example 3. The <EncryptedData> element is the same as the <EncryptedData> element found in Example 2. However, the <KeyInfo> element contains an <EncryptedKey> element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
```

```
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XML Encryption in the WSS-Core

The WSS-Core specification is under development by Organization for the Advancement of Structured Information Standards (OASIS). The specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The message confidentiality is realized by encryption based on XML Encryption.

The WSS-Core specification supports encryption of any combination of body blocks, header blocks, their substructures, and attachments of a SOAP message. When you encrypt parts of a SOAP message, the specification also requires that you prepend a reference from the security header block to the encrypted parts of the message. The reference can be a clue for a recipient to identify which encrypted parts of the message to decrypt.

The XML syntax of the reference varies according to what information is encrypted and how it is encrypted. For example, suppose that the <CreditCard> element in example 4 is encrypted with either a common secret key or the public key of the recipient.

Example 4: Sample SOAP Version 1.1 message

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` so the example changes to the following:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The resulting SOAP messages are shown in Examples 5 and 6. In these example, the <ReferenceList> and <EncryptedKey> elements are used as references, respectively.

Example 5: SOAP Version 1.1 message encrypted with a common secret key

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
```

```
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` and the example changes to the following:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 6: SOAP message encrypted with the public key of the recipient

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
```

```
        <KeyName>Sally Doe</KeyName>
      </KeyInfo>
      <CipherData>
        <CipherValue>yMTEyOTA1M...</CipherValue>
      </CipherData>
      <ReferenceList>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </EncryptedKey>
  </Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <PaymentInfo xmlns='http://example.org/paymentv2'>
    <Name>John Smith</Name>
    <EncryptedData Id='ed1'
      Type='http://www.w3.org/2001/04/xmlenc#Element'
      xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <EncryptionMethod
        Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
      <CipherData>
        <CipherValue>ydUNqHkMrD...</CipherValue>
      </CipherData>
    </EncryptedData>
  </PaymentInfo>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Version 1.2 does not support `encodingStyle` and the example changes to the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Relationship to digital signature

The WSS-Core specification also provides message integrity, which is realized by a digital signature that is based on the XML-Signature specification.

A combination of encryption and digital signature over common data introduces cryptographic vulnerabilities.

Symmetric versus asymmetric encryption

For XML encryption, the application server supports two types of encryption:

- *Symmetric encryption*

  In releases of the application server prior to WebSphere Application Server Version 7, including the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services, by default the KeyName reference was used to refer to the shared key outside of the SOAP message. However, the Web Services Security (WS-Security) Version 1.1 standard does not recommend using the KeyName reference. Because KeyName is not supported by the security policy, it is not supported in the application server.

  The Web Services Secure Conversation (WS-SecureConversation) standard defines how to exchange the shared key between the client and the service and how to refer to the shared key in the message. The use of Kerberos with Web Services Security, as described in the Kerberos Token Profile, also defines how to use a Kerberos session key or key derived from the session key to perform symmetric encryption. Therefore, you can use symmetric encryption by using WS-SecureConversation or Kerberos. WebSphere Application Server supports DerivedKeyToken when using WS-SecureConversation. When using Kerberos, WebSphere Application Server supports both the use of DerivedKeyToken and the use of the Kerberos session key directly.
- *Asymmetric encryption*

  For asymmetric encryption, XML Encryption introduces the idea of key wrapping. The data, such as the contents of the SOAP body element, is encrypted with a shared key that is dynamically generated while processing. Then, the generated shared key is encrypted with the public key of the receiver. WebSphere Application Server supports the X509Token for asymmetric encryption.

Security token

Version 6 and later applications

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

A specific type of security token is not required by Web services security. Web services security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification. However, the security token usage for Web services security is defined in separate profiles such as the Username token profile, the X.509 token profile, the Security Assertion Markup Language (SAML) token profile, the eXtensible rights Markup Language (XrML) token profile, the Kerberos token profile, and so on.

A security token is embedded in the SOAP message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the WebSphere Application Server Web Services security handler authenticates the security token and sets up the caller identity on the running thread.

WebSphere Application Server contains an enhanced security token that has the following features:

- The client can send multiple tokens to downstream servers.

- The receiver can determine which security token to use for authorization based upon the type or signed part for X.509 tokens.
- You can use the custom token or derived key token for digital signing or encryption.

LTPA and LTPA Version 2 tokens

Web services security supports both LTPA (Version 1) and LTPA Version 2 tokens. The LTPA Version 2 token, which is more secure than Version 1, is supported in WebSphere Application Server Version 7.0.

The Lightweight Third Party Authentication (LTPA) token is a specific type of binary security token. The Web services security implementation for WebSphere Application Server, Version 6 and later supports the Version 1 level of LTPA token, while WebSphere Application Server Version 7 added support for Version 2 of LTPA.

Although the same LTPAToken assertion is used in the policy for both LTPA Version 1 and LTPA Version 2, the URI for the Version 2 token is different than Version 1. When LTPA Token v2.0 is selected as the token type for the default policy set bindings, the URI value is set to http://www.ibm.com/websphere/appserver/tokentype, and this value is not editable.

To allow interoperability between servers running different versions of WebSphere Application Server, Web services security can successfully consume an LTPA Version 1 token when the policy is configured to expect an LTPA Version 2 token. Likewise, if a LTPA Version 1 token is expected, a Version 2 token can be consumed. A custom property can be configured to enforce a specific version of the LTPA token. If an LTPA Version 1 token is configured for the token generator, the single sign-on interoperability mode must be enabled in global security. For more information on the custom property or the single sign-on interoperability mode, see the topic Enabling single-sign on interoperability mode for the LTPA token.

Related concepts

“Binary security token” on page 133
The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 6 and later supports LTPA,, LTPA version 2, and X.509 certificate binary security tokens.

Related tasks

“Enabling or disabling single sign-on interoperability mode for the LTPA token” on page 424
You can set an interoperability flag on the token generator to determine whether an LTPA Version 1 token or an LTPA Version 2 token is retrieved when a request message is received.

Username token

You can use the <UsernameToken> element to propagate a user name and, optionally, password information. Also, you can use this token type to carry basic authentication information. Both a user name and a password are used to authenticate the SOAP message.

OASIS: Web Services Security UsernameToken Profile 1.0

A UsernameToken element containing the user name is used in identity assertion. Identity assertion establishes the identity of the user based on the trust relationship.

The following example shows the syntax of the <UsernameToken> element:

```
<wsse:UsernameToken wsu:Id="Example-1">
   <wsse:Username>
   ...
   </wsse:Username>
   <wsse:Password Type="...">
   ...
   </wsse:Password>
```

```
   <wsse:Nonce EncodingType="...">
   ...
   </wsse:Nonce>
   <wsu:Created>
   ...
   </wsu:Created>
</wsse:UsernameToken>
```

The Web services security specification defines the following password types:

wsse:PasswordText (default)
This type is the actual password for the user name.

wsse:PasswordDigest
The type is the digest of the password for the user name. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

WebSphere Application Server supports the default PasswordText type. However, it does not support password digest because most user registry security policies do not expose the password to the application software.

The following example illustrates the use of the <UsernameToken> element:

```
<S:Envelope
       xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Joe</wsse:Username>
                <wsse:Password>ILoveJava</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
</S:Envelope>
```

OASIS: Web Services Security UsernameToken Profile 1.1

WebSphere Application Server supports both Username Token Profile 1.0 and Version 1.1 standards.

WebSphere Application Server does not support the following functions:

- In both versions of the Username Token Profile specification, the digest password type is not supported
- In both versions of the Username Token Profile specification, key derivation based on a password is not supported.

You can use policy sets to configure the UsernameToken using the administrative console. Also, you can use the Web Services Security APIs to attach the Username token to the SOAP message. The following figure describes the creation and validation of the Username token:

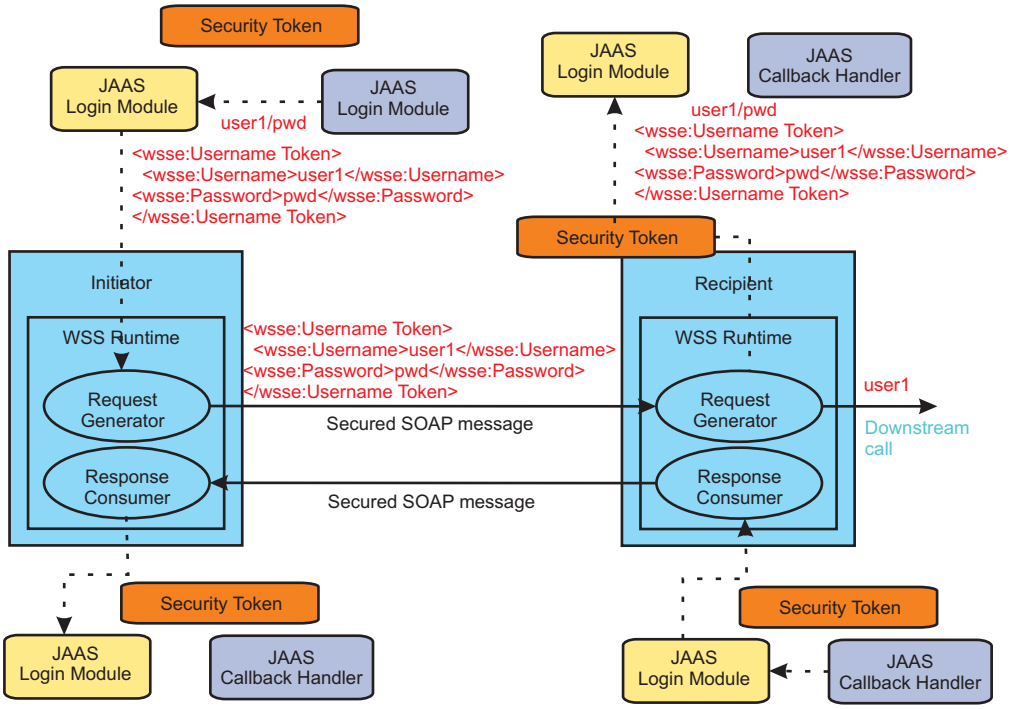


Figure 1. Creating and validating the Username token using the JAAS Login Module and the JAAS CallbackHandler

Note: The WSS API is available only when you are using the Java API for XML-Based Web Services (JAX-WS) programming model.

On the generator side, the Username token is created by using the JAAS LoginModule and by using the JAAS CallbackHandler to pass the authentication data. The JAAS LoginModule creates the UsernameToken object and passes it to the Web Service Security run time.

On the consumer side, the Username Token XML format is passed to the JAAS LoginModule for validation or authentication, and the JAAS CallbackHandler is used to pass the authentication data from the Web Service Security run time to the JAAS LoginModule. After the token is authenticated, a UsernameToken object is created and is passed to the Web service security run time.

The following example provides sample code for creating Username tokens:

```
WSSFactory factory = WSSFactory.getInstance();
   WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Attach the username token to the message.
   UNTGenerationCallbackHandler ugCallbackHandler =
      newUNTGenerationCallbackHandler("alice", "ecila");
   SecurityToken ut = factory.newSecurityToken(ugCallbackHandler,
                                               UsernameToken.class);
   gencont.add(ut);

// Generate the WS-Security header
gencont.process(msgctx);
```

XML token

Version 6 and later applications

XML tokens are offered in two well-known formats called Security Assertion Markup Language (SAML) and eXtensible rights Markup Language (XrML).

In WebSphere Application Server Versions 6 and later, you can plug in your own implementation. By using extensibility of the <wsse:Security> header in XML-based security tokens, you can directly insert these security tokens into the header. SAML assertions are attached to Web services security messages using Web services by placing assertion elements inside the <wsse:Security> header. The following example illustrates a Web services security message with a SAML assertion token.

```
<S:Envelope xmlns:S="...">
<S:Header>
      <wsse:Security xmlns:wsse="...">
          <saml:Assertion
                    MajorVersion="1"
                    MinorVersion="0"
                    AssertionID="SecurityToken-ef375268"
                       Issuer="elliotw1"
                       IssueInstant="2002-07-23T11:32:05.6228146-07:00"
                     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                     ...
           </saml:Assertion>
      </wsse:Security>
 </S:Header>
 <S:Body>
 ...
 </S:Body>
</S:Envelope>
```

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Binary security token

Version 6 and later applications

The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 6 and later supports LTPA,, LTPA version 2, and X.509 certificate binary security tokens.

A binary security token has the following attributes that are used for interpretation:

- Value type
- Encoding type

The following example depicts an LTPA binary security token in a Web services security message header:

```
<wsse:BinarySecurityToken xmlns:ns7902342339871340177=
     "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
     EncodingType="wsse:Base64Binary"
     ValueType="ns7902342339871340177:LTPA">
            MIZ6LGPt2CzXBQfio9wZTo1VotWov0NW3Za6lU5K7Li78DSnIK6iHj3hxXgrUn6p4wZI
            8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbFQJOqh5VcRvI+AKEaNmnEgEV65jUYAC9
            C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
            msu0sewsOKfl/WPsjW0bR/2g3NaVvBy18VlTFBpUbGFVGgzHRjBKAGo+ctkl80nlVLIk
            TUjt/XdYvEpOr6QoddGi4okjDGPyyoDxcvKZnReXww5UsoqlpfXwN4KG9as=
</wsse:BinarySecurityToken>
</wsse:Security>
</soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

The following example depicts an LTPA version 2 binary security token:

```
<wsse:BinarySecurityToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsst="http://www.ibm.com/websphere/appserver/tokentype" wsu:Id="ltpa_20"
    ValueType="wsst:LTPAv2">bRYI0Z59k/P1gIkgSaxeJIQoI1BdojxjdoD+6qMmiH37lqS6U90Wx6EArMA05FHVyTmxvIJACGD
    UVfqVcPDQCdPlWAn9Brhz/bXw9OEVx0wx/eNYQuiBvEVNam7urd8SxZkqppOZyeN6APZ4Z4Rox0MjqQv9lFIB/AKBpJyaK8V9Z9
    gFO8k6J5HmE/G9jdBov9Su6hXlfF50Bhy6tx8BEm4Zn/pkeNc1H1d+tOxwDOfS0ORWH0tjzDCTFpAMPjMmfR0/o7o3DivONtZG61
    ylbcwB4hx01iQC/FN5DJwrEy8kCwCeFywubKVVt5pyM1k6uVXI8ik5Pjf9aU1ei86y5iXc9CirhvqosXiZvjObHTYKZSjtGiMYw3
    q9NKbZxsSzfCuAdht8sjGfaVo43i0iz7CuFYAywqVldUPjwSTvCGNtmWB/3MRtBDrmq3fqYSomjw5ZWDFex/n98ZaOz8mUjNHinJ
    c4APTtEx6S10CxUkUc8b8hoCdqbcOGdZcGqYF7xgcFXvsezsXw0eRmhra54x6gCJs1skMMNvi0vF2pic1cg4GClQ74NKxV1oTrDZ
    PaQPTikYGJOLKHBPYnbPda0hPkX+iCOYN0IIRBaVwjj1T0G+Y/MgokiNJRGwUQ7VHXEo0+Q2HsmCkmAFrIp4lZc9fGcFyVY/EUBB
    pkGchL0eKNv4DoVJW6EhFXWZdeiVk8
</wsse:BinarySecurityToken>
```

Kerberos token

IBM WebSphere Application Server provides Kerberos token support for Web services message-level security. The support is based on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Kerberos Token Profile Version 1.1. Use this topic to understand the Kerberos support that is available for Web services.

Kerberos token profile version 1.1

Kerberos Version 5 is a mature, open standard that provides a secure third-party authentication mechanism. The OASIS Web Services SOAP Message Security specification references the Kerberos token in the SOAP message. Web services applications can use the Kerberos token to send identities and protect messages more securely. Overall, Kerberos support involves Kerberos support in Java Platform, Enterprise Edition (Java EE) security and the Kerberos token support in Web services security. This topic covers the Kerberos token support in Web services security only.

In WebSphere Application Server Version 7.0, Web services security introduces support for the Kerberos token, which is based on OASIS WS-Security Kerberos Token Profile Version 1.1 specification. The Kerberos token is a binary security token for Web services message-level security. Web services security provides SOAP message-level security, such as security token propagation, message signature, and message encryption. The Kerberos token is used for message security, specifically with the SOAP message security specification for Web services, and is another supported token, such as the username token and the secure conversation token.

For more information, see the Web Services Security Kerberos Token Profile Version 1.1 specification. The specification explains how to use Kerberos security with the Web services security and how the Kerberos token is propagated and used to secure the SOAP message through signing and encryption.

Kerberos token profile enablement

The WebSphere Application Server configuration model leverages existing tools and frameworks for the Kerberos token profile configuration of authentication and message protection, such as:

- Policy set and binding configuration to enable the Kerberos token profile for Java API for XML-Based Web Services (JAX-WS) applications
- Deployment descriptor and binding configuration to enable the Kerberos token profile for JAX-RPC applications
- Token profile enablement with a Kerberos token for JAX-WS applications
- Minimal client configuration to enable the Kerberos token profile using the JAX-WS programming model

For JAX-WS client applications, the design updates the application programming interfaces (APIs) for Web services security and enforces a Web services security policy with a Kerberos token, which is based on the OASIS token profile. To enable a Kerberos token profile by using a policy set, you must first establish the Web services security policy and binding files by using a custom token. For more information, see the ″Kerberos configuration models for Web services″ topic.

Kerberos support

The following Kerberos-related function is supported by Web services in WebSphere Application Server:

- Client programming models for JAX-WS applications with Web services security APIs
- Interoperability with Web Services Enhancements (WSE) Version 3.5 and Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET
- Recovery of Web services message security tokens for JAX-WS applications
- Kerberos token profile enablement
- Integration with the base security for the application server
- Kerberos token generation for the client and service
- Kerberos consumption at the service
- Clustering and high-availability for JAX-WS applications
- Kerberos token profile configuration of authentication and message protection for JAX-WS applications
- Integration in a single realm with either a Microsoft or z/OS® operating system Key Distribution Center (KDC).
- Kerberos token profile configuration of authentication for JAX-RPC applications
-

The application server does not support the following function:

- Key name references
- Message protection using session keys for JAX-RPC applications
- Message protection using derived keys for JAX-RPC applications
- Generation of SHA1 keys for JAX-RPC applications

Kerberos message protection for Web services:

Message-level security is based on the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Kerberos Token Profile Version 1.1 specification. Use this topic to gain an overall understanding of how message protection is implemented with a Kerberos token for Web services.

Message protection

The application server can interoperate with other Web services technology because of the implementation of the OASIS Web services Kerberos token profile. This specification defines the standards for securing a SOAP message with the Kerberos token. However, mutual authentication is not defined by the token profile. The OASIS Web Services SOAP Message Security specification describes how to secure a SOAP message through signing and encryption by using and referencing a Kerberos token. Specifically, the OASIS specification defines how the Kerberos token, as a wrapped or unwrapped AP_REQ packet, is encoded and attached to the SOAP message. The token that is described in the OASIS Kerberos token profile is limited to the AP_REQ packet, which consists of a service ticket and an authenticator. The AP_REQ packet is obtained from the Key Distribution Center (KDC), which serves as the third-party authentication service.

Multiple formats exist for the Kerberos token, as defined in the OASIS Web Services Security Kerberos Token Profile 1.1. The `@ValueType` attribute is used to specify the token format. You must specify one of the following `<@ValueType>` attributes for the element:

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120

The resulting AP_REQ token can be either GSS-API framed (wrapped) or raw (unwrapped). The token must be Base-64 encoded.

Kerberos usage overview for Web services:

You can use a Kerberos token to complete similar functions that you might currently complete with other binary security tokens, such as Lightweight Third Party Authentication (LTPA) and Secure Conversation tokens.

Token generator

After the Kerberos token is created from the Key Distribution Center (KDC), the Web services security generator encodes and inserts the token into the SOAP message and propagates the token for token consumption or acceptance. If a message integrity or confidentiality key is required, a Kerberos sub-key or a Kerberos session key from the Kerberos ticket is used. A key can be derived from either the Kerberos sub-key or the Kerberos session key. Web services security uses the key from the Kerberos token to sign and encrypt the message parts as described in the OASIS Web Services Security Kerberos Token Profile Version 1.1 specification. The type of key to use is predetermined by the Web services security configuration or policy. Also, the size of the derived key is configurable.

The value of the signature or encryption key is constructed from the value of one of the following keys:
- The Kerberos sub-key when it is present in the authenticator
- A session key directly from the ticket if the sub-key is absent
- A key that is derived from either of the previous keys

When the Kerberos token is referenced as a signature key, the signature algorithm must be a hashed message authentication code, which is http://www.w3.org/2000/09/xmldsig#hmac-sha1. When the Kerberos token is referenced as an encryption key, you must use one of the following symmetric encryption algorithms:
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc
- http://www.w3.org/2001/04/xmlenc#tripledes-cbc

Note:
- The Application Server supports Kerberos Version 5 only.
- You can use a AES-type symmetric algorithm suite in Web services security when the Kerberos ticket complies with RFC-4120 only.
- A Kerberos key with the RC4-HMAC 128–bit key type only is used when the KDC is on a Microsoft Windows 2003 server.
- A Kerberos key with AES 128–bit or 256–bit key types is used when the KDC is on a Microsoft Windows 2008 server.
- A Kerberos ticket must be forwardable and not contain an address when the service provider is running in a cluster.
- You must import an unrestricted Java security policy when you use an AES 256–bit encryption algorithm.

Token consumer

The Web services security consumer receives and extracts the Kerberos token from the SOAP message. The consumer then accepts the Kerberos token by validating the token with its own secret key. The secret

key of the service is stored in an exported keytab file. After acceptance, the Web services security consumer stores the associated request token information into the context Subject. You can also derive the corresponding key to the request token. The key is used to verify and decrypt the message. If the request token is forwardable and does not contain an address, the application server can use the stored token for downstream calls.

Token format and reference

For JAX-WS applications, use the existing custom policy set or administrative command scripts for the custom policy to specify the Kerberos token type, the message signing, and message encryption. The JAX-WS programming model for WebSphere Application Server provides minimal configuration to enable the Kerberos token profile with the Kerberos token.

For JAX-RPC applications, use the deployment descriptor to specify that the custom token use the Kerberos token. You can use the Kerberos token for authentication, but you cannot use it for message signing or encryption.

WebSphere Application Server supports the following callback handler classes for the Kerberos Version 5 token:

- com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler

  This class is a callback handler for Kerberos Version 5 token on the consumer side. This instance is used to generate the WSSVerification and WSSDecryption objects to validate a Kerberos binary security token.
- com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler

  This class is a callback handler for Kerberos Version 5 token on the generator side. This instance is used to generate the WSSSignature object and the WSSEncryption object to generate a Kerberos binary security token.

The OASIS Web Services Security Kerberos Token Profile Version 1.1 specification states that the Kerberos token is attached to the SOAP message with the <wsse:BinarySecurityToken> element. The following example shows the message format. The boldface type shows delineates the binary security token information from the other parts of the example.

```
<S11:Envelope xmlns:S11="..." xmlns:wsu="...">
   <S11:Header>
      <wsse:Security xmlns:wsse="...">
          <wsse:BinarySecurityToken
              EncodingType="http://docs.oasis-open.org/wss/2004/01/
                 oasis-200401-wss-soap-message-security-1.0#Base64Binary"
              ValueType=" http://docs.oasis-open.org/wss/
                 oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ"
              wsu:Id="MyToken">boIBxDCCAcCgAwIBBaEDAgEOogcD...
          </wsse:BinarySecurityToken>
          ...
      </wsse:Security>
    </S11:Header>
    <S11:Body>
        ...
    </S11:Body>
</S11:Envelope>
```

The Kerberos token is referenced by the <wsse:SecurityTokenReference> element. The <wsu:Id> element, which is specified within the <wsse:BinarySecurityToken> element and is shown within the following example in boldface type, directly references the token in the <wsse:SecurityTokenReference> element.

The @wsse:TokenType attribute value within the <wsse:SecurityTokenReference> element matches the ValueType attribute value of the <wsse:BinarySecurityToken> element. The Reference/@ValueType attribute is not required. However, if the attribute is specified, its value must be equivalent to the @wsse11:TokenType attribute.

The following example shows the message format, the correlation between the<wsu:Id> and <wsse:SecurityTokenReference> elements, and the relationship between the @wsse:TokenType and ValueType attribute values.

```
<S11:Envelope xmlns:S11="..." xmlns:wsu="...">
   <S11:Header>
      <wsse:Security xmlns:wsse="...">
          <wsse:BinarySecurityToken
               EncodingType="http://docs.oasis-open.org/wss/2004/01/
                    oasis-200401-wss-soap-message-security-1.0#Base64Binary"
               ValueType=" http://docs.oasis-open.org/wss/
                    oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ"
               wsu:Id="MyToken">boIBxDCCAcCgAwIBBaEDAgEOogcD...
          </wsse:BinarySecurityToken>
          </wsse:Security>
   </S11:Header>
</S11:Envelope>
      <wsse:Security>
       </wsse:Security>
          <wsse:SecurityTokenReference
               TokenType="http://docs.oasis-open.org/wss/
                    oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ">
               <wsse:Reference URI="#MyToken"
                    ValueType="http://docs.oasis-open.org/wss/
                        oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ">
              </wsse:Reference>
          </wsse:SecurityTokenReference>
         ...
      <wsse:Security>
      </wsse:Security>
    <S11:Header>
    </S11:Header>
    <S11:Body>
       ...
    </S11:Body>
 <S11:Envelope>
 </S11:Envelope>
```

The <wsse:KeyIdentifier> element is used to specify an identifier for the Kerberos token. The value of the identifier is a SHA1 hash value of the encoded Kerberos token in the previous message. The element must have a ValueType attribute with a #Kerberosv5APREQSHA1 value. The KeyIdentifier reference mechanism is used on subsequent message exchanges after the initial Kerberos token is accepted. The following example shows the key identifier information in boldface type:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
    <S11:Header>
      <wsse:Security>
         ...
         <wsse:SecurityTokenReference
               wsse11:TokenType=http://docs.oasis-open.org/wss/
                   oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ>
               <wsse:KeyIdentifier
                   ValueType="http://docs.oasis-open.org/wss/
                       oasis-wss-kerberos-token-profile-1.1#Kerberosv5APREQSHA1">
                       GbsDt+WmD9XlnUUWbY/nhBveW8I=
               </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
          ...
      </wsse:Security>
   </S11:Header>
   <S11:Body>
       ...
    </S11:Body>
</S11:Envelope>
```

Multiple references to the Kerberos token

The client is not required to send a Kerberos token in every request after the Kerberos identity is validated and accepted by the service. The OASIS Web Services Security Kerberos Token Profile Version 1.1

specification suggests that you use a SHA1 encoded key with the <wsse:KeyIdentifier> element within the <wsse:SecurityTokenReference> element for every subsequent message after the initial AP_REQ packet is accepted. However, the runtime environment for Web services security must map the key identifier to a cached Kerberos token for further processing. IBM WebSphere Application Server 7.0 supports this SHA1 caching as described in the profile, by default. However, the application server also provides the ability to generate new AP_REQ tokens for each request with the existing service Kerberos ticket. When you interoperate with Microsoft .NET, do not use pSHA1 caching; generate an AP_REQ packet for each request.

Kerberos configuration models for Web services:

The IBM WebSphere Application Server configuration model leverages existing frameworks.

The configuration model features include:

- Deployment descriptors and bindings configuration to enable the Kerberos token profile for Java API for XML-based RPC (JAX-RPC) applications
- Policy sets and bindings configuration to enable the Kerberos token profile for Java Architecture for XML Web Services (JAX-WS) applications
- Web Services Security APIs for JAX-WS applications
- Administrative command scripts
- Interoperability with Microsoft Web Services Enhancements (WSE) Version 3.5

Following are some examples of possible configurations when using the Kerberos token:

- A JAX-WS client on Windows operating systems
- A JAX-RPC client on Windows operating systems
- A Windows JAX-RPC client on z/OS operating systems
- Web services security APIs on Windows operating systems
- A Microsoft .NET WSE 3.5 client on Windows operating systems
- A Microsoft .NET WSE 3.5 client on z/OS operating systems

JAX-WS configuration model

For JAX-WS applications, the WebSphere Application Server client configuration model uses the policy set and leverages a custom policy set for the Kerberos token. You can specify the Kerberos token type and message signing and the encryption by using the custom policy set. The Web Services Security (WS-Security) policy is the security policy that is used to secure the application messages.

Using the administrative console, you can specify the Kerberos token type, message signing, and message encryption by using an existing custom policy set. Kerberos token generation and consumption includes the Kerberos token generation for unmanaged JAX-WS clients.

The JAX-WS programming model also provides capabilities to enable the Kerberos token profile and identity assertion by configuring the Kerberos token using policy sets, Web services security APIs, and administrative command scripts.

For JAX-WS applications, you can use administrative commands to configure the policy set as an alternative to using the administrative console.

JAX-RPC configuration model

JAX-RPC applications are configured using a deployment model. The deployment descriptor specifies the custom token to use for the Kerberos token. A JAX-RPC client can generate the specified Kerberos token.

A JAX-RPC Web service can successfully authenticate the Kerberos token by using a custom or the default Kerberos identity mapping login module.

API configuration model

A set of APIs is provided by WebSphere Application Server. To successfully use these APIs, application developers must have knowledge about the OASIS Web Services Security Version 1.0 and 1.1 specifications. When you use these APIs, the application server assumes that a policy set is not attached to the client resources; however, a warning is still issued when the application server detects any policy set information.

For JAX-WS client applications, the APIs include and enforce Web services security policy for the Kerberos token, which is based on the OASIS token profile. To enable the Kerberos token profile with the policy set, you must first configure the WS-Security policy and the binding files with the custom token.

For JAX-RPC applications, APIs for Web services security are not provided. You must use the deployment descriptor to specify the custom token to use the Kerberos token. You can use the custom token panels within an assembly tool, such as Rational Application Developer, to configure the deployment information.

Kerberos clustering for Web services:

Clusters are groups of servers that are managed together and participate in workload management.

In a clustered environment, the Kerberos token needs to be distributed and recoverable. The Web services security configuration saves and distributes Kerberos tokens among the cluster members. The Kerberos tokens that are created or validated in one server are available to the other cluster members. The distributed cache or database repository need to be configured as the caching mechanism.

Security considerations for Web services

Version 6 and later applications

When you configure Web services security, you should make every effort to verify that the result is not vulnerable to a wide range of attack mechanisms. There are possible security concerns that arise when you are securing Web services.

In WebSphere Application Server, when you enable integrity, confidentiality, and the associated tokens within a SOAP message, security is not guaranteed. This list of security concerns is not complete. You must conduct your own security analysis for your environment.

- Ensuring the message freshness

  Message freshness involves protecting resources from a replay attack in which a message is captured and resent. Digital signatures, by themselves, cannot prevent a replay attack because a signed message can be captured and resent. It is recommended that you allow message recipients to detect message replay attacks when messages are exchanged through an open network. You can use the following elements, which are described in the Web services security specifications, for this purpose:

  Timestamp

  You can use the timestamp element to keep track of messages and to detect replays of previous messages. The WS-Security 2004 specification recommends that you cache time stamps for a given period of time. As a guideline, you can use five minutes as a minimum period of time to detect replays. Messages that contain an expired timestamp are rejected.

  Nonce

  A nonce is a child element of the <UsernameToken> element in the UsernameToken profile. Because each nonce element has a unique value, recipients can detect replay attacks with relative ease.

Note: Both the time stamp and nonce element must be signed. Otherwise, these elements can be altered easily and, therefore, cannot prevent replay attacks.

- Using XML digital signature and XML encryption properly to avoid a potential security hole

  The Web Services Security 2004 specification defines how to use XML digital signature and XML encryption in SOAP headers. Therefore, users must understand XML digital signature and XML encryption in the context of other security mechanisms and their possible threats to an entity. For XML digital signature, you must be aware of all of the security implications resulting from the use of digital signatures in general and XML digital signature in particular. When you build trust into an application based on a digital signature, you must incorporate other technologies such as certification trust validation based upon the Public Key Infrastructure (PKI). For XML encryption, the combination of digital signing and encryption over a common data item might introduce some cryptographic vulnerabilities. For example, when you encrypt digitally signed data, you might leave the digital signature in plain text and leave your message vulnerable to plain text guessing attacks. As a general practice, when data is encrypted, encrypt any digest or signature over the data. For more information, see http://www.w3.org/TR/xmlenc-core/#sec-Sign-with-Encrypt.
- Protecting the integrity of security tokens

  The possibility of a token substitution attack exists. In this scenario, a digital signature is verified with a key that is often derived from a security token and is included in a message. If the token is substituted, a recipient might accept the message based on the substituted key, which might not be what you expect. One possible solution to this problem is to sign the security token (or the unique identifying data from which the signing key is derived) together with the signed data. In some situations, the token that is issued by a trusted authority is signed. In this case, there might not be an integrity issue. However, because application semantics and the environment might change over time, the best practice is to prevent this attack. You must assess the risk assessment based upon the deployed environment.
- Verifying the certificate to leverage the certificate path verification and the certificate revocation list

  It is recommended that you verify that the authenticity or validity of the token identity that is used for digital signature is properly trusted. Especially for an X.509 token, this issue involves verifying the certificate path and using a certificate revocation list (CRL). In the Web services security implementation in WebSphere Application Server Version 6 and later, the certificate is verified by the <TokenConsumer> element. WebSphere Application Server provides a default implementation for the X.509 certificate that uses the Java CertPath library to verify and validate the certificate. In the implementation, there is no explicit concept of a CRL. Rather, proper root certificates and intermediate certificates are prepared in files only. For a sophisticated solution, you might develop your own TokenConsumer implementation that performs certificate and CRL verification using the online CRL database or the Online Certificate Status Protocol (OCSP).
- Protecting the username token with a password

  It is recommended that you do not send a password in a username token to a downstream server without protection. You can use transport-level security such as SSL (for example, HTTPS) or use XML encryption within Web services security to protect the password. The preferred method of protection depends upon your environment. However, you might be able to send a password to a downstream server as plain text in some special environments where you are positive that you are not vulnerable to an attack.

Securing Web services involves more work than just enabling XML digital signature and XML encryption. To properly secure a Web service, you must have knowledge about the PKI. The amount of security that you need depends upon the deployed environment and the usage patterns. However, there are some basic rules and best practices for securing Web services. It is recommended that you read some books on PKI and also read information on the Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP).

Nonce, a randomly generated token: **Version 6 and later applications**

Nonce is a randomly-generated, cryptographic token that is used to prevent replay attacks. Although nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

Without nonce, when a UsernameToken is passed from one machine to another machine using a nonsecure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same password might be reused when the user name token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption. However, nonce alone, used in a non-secure transport, cannot adequately address the replay problem. Nonce is most useful when the SOAP message is transmitted via a communication channel that is secured, either at the transport level, or at the message level.

To help eliminate these replay attacks, the <wsse:Nonce> and <wsu:Created> elements are generated within the <wsse:UsernameToken> element and used to validate the message. The server checks the freshness of the message by verifying that the difference between the nonce creation time, which is specified by the <wsu:Created> element, and the current time falls within a specified time period. Also, the server checks a cache of used nonces to verify that the user name token in the received SOAP message has not been processed within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

To add a nonce for the UsernameToken, you can specify it in the token generator for the user name token. When the token generator for the UsernameToken is specified, you can select the **Add nonce** option if you want to include nonce in the user name token.

Basic Security Profile compliance tips: **Version 6 and later applications**

The Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP) 1.0 promotes interoperability by providing clarifications and amplifications to a set of nonproprietary Web services specifications. WebSphere Application Server Web Services Security provides configuration options to ensure that the BSP recommendations and security considerations can be enabled to ensure interoperability. The degree to which you follow these recommendations is then a measure of how well the application you are configuring complies with the Basic Security Profile (BSP).

Support for applications to comply to the Basic Security Profile (BSP) is new in WebSphere Application Server Version 7.0. For more information on the Basic Security Profile, see Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP), Basic Security Profile Version 1.0.

You can use either a predefined list of keywords or XPath expressions to comply to the BSP. Both the keywords and the XPath expressions are specified in the deployment descriptor configuration file and are configured using an assembly tool.

Basic Security Profile recommendations

Follow these recommendations to ensure that your configured applications are Basic Security Profile (BSP) compliant.

- Do not use the original XPath transform, http://www.w3.org/TR/1999/REC-xpath-19991116

  When you refer to an element in a SECURE_ENVELOPE that does not carry an ID attribute type from a ds:Reference in a SIGNATURE element, you must use the XPath Filter 2.0 transform, http://www.w3.org/2002/06/xmldsig-filter2 to refer to that element.

  Any ds:Transform/@Algorithm attribute in a SIGNATURE element must have one of these values:

  – http://www.w3.org/2001/10/xml-exc-c14n#
  – http://www.w3.org/2002/06/xmldsig-filter2
  – http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform

  - http://www.w3.org/2000/09/xmldsig#enveloped-signature
  - http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform
  - http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform
- Do not use the http://www.w3.org/2000/09/xmldsig#dsa-sha1 signature algorithm.

  Any ds:SignatureMethod/@Algorithm element in a SIGNATURE that is based on a symmetric key must have one of the following values:
  - http://www.w3.org/2000/09/xmldsig#rsa-sha1
  - http://www.w3.org/2000/09/xmldsig#hmac-sha1
- Do not specify the digestvalue keyword for the message part to encrypt. Instead, use the signature keyword.

  If the value of a ds:DigestValue element in a SIGNATURE element requires encryption, the entire parent ds:Signature element must be encrypted. A SIGNATURE must not have any xenc:EncryptedData elements among its descendants.
- Do not use the KEYNAME key information type

  KEYNAME references can be ambiguous and compliance with the BSP disallows the use of KEYNAME.

  A SECURITY_TOKEN_REFERENCE must not use a key name to reference a SECURITY_TOKEN. The child element of a ds:KeyInfo element in an ENCRYPTED_KEY must be either a SECURITY_TOKEN_REFERENCE or a ds:MgmtData element. Using a KEYNAME key information type for an encryption key results in a KeyName child element of a ds:KeyInfo element and is disallowed for BSP compliance.
- Do not use the http://www.w3.org/2001/04/xmlenc#aes192-cbc bit data encryption algorithm.

  Any xenc:EncryptionMethod/@Algorithm attribute in an ENCRYPTED_DATA element must have one of these values:
  - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - http://www.w3.org/2001/04/xmlenc#aes128-cbc
  - http://www.w3.org/2001/04/xmlenc#aes256-cbc
- Do not use the advanced encryption standard (AES) key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192 key encryption algorithm.

  When used for key wrap, any xenc:EncryptionMethod/@Algorithm attribute in an ENCRYPTED_KEY element must have one of these values:
  - http://www.w3.org/2001/04/xmlenc#kw-tripledes
  - http://www.w3.org/2001/04/xmlenc#kw-aes128
  - http://www.w3.org/2001/04/xmlenc#kw-aes256

Configuration Options for BSP Compliance

You achieve BSP compliance when certain configuration choices are made. The assembly tool assists you in using appropriate choices when configuring the application by issuing warning messages. The following configuration descriptions comprise these warnings:

- When configuring the ds:Transforms element in a signature, the list of transforms must include as its last child element http://www.w3.org/2001/10/xml-exc-c14n# or http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
- Add a wsse:Nonce or wsse:Created element to a Username token to prevent replay. After the element is added, sign the Username token to prevent undetected alteration of these fields; otherwise, replay can occur.

Distributed nonce cache:

In previous releases of WebSphere Application Server, the nonce was cached locally. WebSphere Application Server Versions 6 and later use distributed nonce caching. The distributed nonce cache makes it possible to replicate nonce data among servers in a WebSphere Application Server cluster.

If nonce elements are in a SOAP header, all nonce values are cached by the server in the cluster. If the distributed nonce cache is enabled, the cached nonce values are copied to other servers in the same cluster. Then, if the message with the same nonce value is sent to (one of) other servers, the message is rejected. A received nonce cache value is cached and replicated in a push manner among other servers in the cluster with the same replication domain. The replication is an out-of-process call and, in some cases, is a remote call. Therefore, there is latency when the content of the cache in the cluster is updated.

For example, you might have application server A and application server B in cluster C.

- A SOAP client sends a message with nonce `abc` to application server A.
- The server caches the value and pushes it to the other application server B.
- If the client sends the message with nonce `abc` to application server B after a certain time frame, the message is rejected and if the application server B receives the nonce with the same value within a specified period of time, a SoapSecurityException is thrown by application server B.

  For more information, see the information that explains nonce cache timeout, nonce maximum age, and nonce clock skew in “Token generator configuration settings” on page 417.
- If the client sends the message with another nonce value of `xyz`, the message is accepted, the value is cached by application server B and is copied into the other application servers within the same cluster.

Custom security token propagation

Version 6 and later applications

Web services security has the ability to send security tokens in the security header of a SOAP message. These security tokens can be used to sign, verify, encrypt or decrypt message parts. These security tokens can also be sent as standalone security tokens and set as the caller on the request consumer. *Custom security token propagation* is used to propagate these custom security tokens by using Web services security.

Web services security supports the Username, X.509 and Lightweight Third-Party Authentication (LTPA) security token types. When you use security token propagation, the propagation token is sent in the wsse:BinarySecurityToken element in the security header of the SOAP message. Web services security uses the same propagation token format as used by the Security attribute propagation feature.

Configuring this option is similar to the configuration for sending and receiving LTPA tokens. The same token generator and token consumer implementations are used, for example:

- com.ibm.wsspi.wssecurity.token.LTPATokenGenerator
- com.ibm.wsspi.wssecurity.token.LTPATokenConsumer

But, the token type Uniform Resource Identifier (URI) and local name for the token generator and token consumer are different. For custom token properties, use the following values:

- **Token type URI:** http://www.ibm.com/websphere/appserver/tokentype
- **Token type local name:** LTPA_PROPAGATION

By default, the custom token propagation uses the following JAAS login configuration entries:

- **Inbound:** WSS_INBOUND
- **Outbound:** WSS_OUTBOUND

You can use the com.ibm.ws.webservices.wssecurity.constants.jaasConfig custom property to specify a different JAAS login configuration for the generator. You can do this configuration on the CallbackHandler

configuration panel. To specify a different JAAS login configuration on the consumer side, use the JAAS configuration name field in the Token consumer panel.

Securing JAX-WS Web services using message-level security

Web services security standards and profiles address how to provide message-level protection for messages that are exchanged in a Web service environment.

Before you begin

Before you begin this task, you must develop and deploy a JAX-WS application. See the JAX-WS topic for more information.

About this task

Java API for XML-Based Web Services (JAX-WS) is the next generation Web services programming model complimenting the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, development of Web services and clients is simplified with greater platform independence for Java applications through the use of dynamic proxies and Java annotations. JAX-WS simplifies application development through support of a standard, annotation-based model to develop Web service applications and clients. A required part of the Java Platform, Enterprise Edition 5 (Java EE 5), JAX-WS is also known as JSR 224.

JAX-WS applications can be secured with Web services security in one of two ways. The application can be secured using policy sets, or through the use of the Web Services Security API (WSS API). The WSS API can only be used to secure a JAX-WS client application. The following sections describe both methods.

Securing JAX-WS applications using the WSS API

To secure JAX-WS client applications with message-level security programmatically, using the WSS API, see the topic Securing Web services applications using the WSS APIs at the message level.

Securing JAX-WS applications using policy sets

1. Select, create, or copy and modify a policy set to specify the message-level protection required. The policy specifies what protection will be applied, for example, what message parts to sign or encrypt and the token types and algorithms to use.
   a. Signing and encrypting message parts.
   b. Specify security tokens using the token type settings, such as:
      - Kerberos token
      - Username token
      - LTPA token
      - X.509 token

   For complete information about policy sets, read the topic Managing policy sets using the administrative console.
2. Configure the default Web services security bindings.
   a. Configure the token consumer.
   b. Configure the token generator.

   For more information about bindings, read the topic Defining and managing policy set bindings.

Configuring policy sets through metadata exchange (WS-MetadataExchange)

In WebSphere Application Server Version 7.0, using JAX-WS, you can enable the Web Services Metadata Exchange (WS-MetadataExchange) protocol so that the policy configuration of the service provider is included in the WSDL and is available to a WS-MetadataExchange GetMetadata request. One advantage of using the WS-MetadataExhange protocol is that you can apply message-level security to WS-MetadataExchange GetMetadata requests by using a suitable system policy set. Another advantage is that the client does not have to match the provider configuration, or have a policy set attached. The client only needs the binding information, and then the client can operate based on the provider policy, or based on the intersection of the client and provider policies. You can configure a service provider to share its policy configuration using the administrative console. For more information, read the following topics:

- Configuring security for a WS-MetadataExchange request
- Configuring a service provider to share its policy configuration
- Transformation of policy and binding assertions for WSDL

Migration of JAX-WS Web services security bindings from Version 6.1 to Version 7.0

You can migrate Web services security bindings from an older version to Version 7.0 of WebSphere Application Server. Migration of the JAX-WS bindings in Version 6.1 Feature Pack for Web Services takes place during the product migration to Version 7.0.

The product migration handles most of the WS-Security migration process, but your input and action is required for specific configurations in order to complete the migration. Following are some examples of configurations that require manual migration steps:

- Using callers on Version 6.1 Feature Pack for Web Services.

  WebSphere Application Server Version 7.0 introduces the capability to specify a preference order for callers. In the situation where only a single caller is present in the bindings, product migration automatically assigns an order of **1** to the single caller that is present. However, if there are multiple callers, warnings are logged during migration, and you must manually assign the caller preference order. Set the order attribute for each caller using the administrative console, or the administration commands, after product migration is completed. See the topics Caller collection and Configuring the callers for general and default bindings for instructions on setting the caller order using the administrative console. See the topic WS-Security policy and bindings properties for information on using the administration commands.
- Using multiple username tokens in the default bindings.

  During product migration for Version 6.1 default bindings, all UsernameToken generators and consumers in the bindings will be migrated. However, if multiple username token generators, or consumers, are used, a warning is logged during migration. The warning states that a maximum of two username token generators and two username token consumers are allowed, and that one of each pair of token generators or consumers must have the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property set to true. You must manually select which username token consumer or generator to keep, and which token has the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property, using the administrative console, or administration commands.
  - To use the administrative console to set the property, access the WS-Security 6.1 default bindings as instructed in the Policy set bindings settings for WS-Security topic and click the **Authentication and Protection** link. Add, remove or edit the username token consumers and generators as needed, following the instructions in the WS-Security authentication and protection for general bindings topic.
  - To use the administration commands to set the property, and to add, delete and edit the username token generators or consumers as needed, refer to the WS-Security policy and bindings properties topic for instructions.
- Using multiple token generators and token consumers of the same type in Version 6.1 default bindings.

  During product migration for Version 6.1 default bindings, all token generators and token consumers for supporting tokens are migrated. In the situation where multiple token generators and token consumers of the same supporting token type are found a warning is logged during migration. This warning states that only a single token consumer and token generator for each supporting token type must be present.

You must manually select which token consumer or generator to use for the repeating supporting token type, using the administrative console or administration commands. To use the administrative console to select the token, access the WS-Security 6.1 default bindings as instructed in the Policy set bindings settings for WS-Security topic, and select the **Authentication and Protection** link. Add, edit or remove token consumers and generators as needed, following the instructions in the WS-Security authentication and protection for general bindings topic.

Securing Web services using policy sets

Policy sets are assertions about how services are defined. They are used to simplify the quality of service configuration for Web services.

About this task

Policy sets combine configuration settings, including those for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, and WS-Security. There are two main types of policy sets; application policy sets and system policy sets. Application policy sets are used for business-related assertions. These assertions are related to the business operations that are defined in the Web Services Description Language (WSDL) file. System policy sets, on the other hand, are used for non-business-related system messages. These messages are not related to the business operations that are defined in the WSDL, but instead refer to messages that are defined in other specifications which apply qualities of service (QoS). Such QoS are the request security token (RST) messages that are defined in WS-Trust, or create sequence messages that are defined in WS-Reliable Messaging metadata exchange messages of the WS-MetadataExchange.

Note: You can use policy sets only with Java™ API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

Policies are defined based on a quality of service. Policy definition is typically based on WS-Policy standard language, for example, the WS-Security policy is based on the current WS-SecurityPolicy from the Organization for the Advancement of Structured Information Standards (OASIS) standards.

Policy sets do not include environment or platform-specific information, such as keys for signing, keystore information, or persistent store information. This type of information is defined in the binding. A policy set attachment defines how a policy set is attached to service resources and bindings. The attachment definition is outside the policy set definition and is defined as meta-data associated with application data.

To secure JAX-WS Web services with message-level security using policy sets, follow these steps:

1. Select, create, or copy and modify a policy set to specify the message-level protection required. The policy specifies what protection will be applied, for example, what message parts to sign or encrypt and the token types and algorithms to use.
   - Select one of the default policy sets.
   - Manage policy sets using the administrative console. Create, copy, modify, import, export or delete policy sets.
2. Attach the policy set to the application.
3. Create or select the policy set bindings to be used. The bindings are then attached to the application along with the policy set. The bindings used can either be general bindings that can be shared among applications or application specific bindings.
4. If WS-SecureConversation is being used, specify the trust service system policy sets and bindings on the application server.

Example: Configuring the message-level WS-Security policy set and bindings:

This example shows how to configure the message-level WS-Security policy set and bindings to send a Username token in a JAX-WS request, and to encrypt the Username token using asymmetric encryption.

Before you begin

Make a copy of the **Username WSSecurity** default policy set and give it a unique name. This example illustrates how to modify a copy of the default policy set. For more information on how to copy a policy set, see the topic Copy of default policy set and bindings settings.

About this task

By default, the Username WSSecurity policy set signs the WS-Addressing headers and body in the request and the response, and encrypts the body and signature in the request and the response. However, in this example, the goal is to encrypt only the Username token in the request from the client to the service, but not to encrypt any part of the response from the service to the client. In addition, no part of the request or the response will be signed. Therefore, the policy set must be modified to remove several message protection parts. You must also configure the client and server bindings.

First, configure the policy set by modifying your copy of the Username WSSecurity default policy set.

1. From the administrative console, click **Services > Policy sets > Application policy sets > *policy_set_name***. In the Policy set settings panel, you can specify information about the policy set, such as the description.
2. Remove the following message protection parts: request:app_signparts, response:app_signparts and response:app_encparts.
   a. Click **Application policy sets > *policy_set_name* > WS-Security > Main policy > Response message part protection**.
   b. Click on **app_encparts** in the Encrypted parts box, then click the **Delete** button.
   c. Click on **app_signparts** in the Signed parts box, then click the **Delete** button.
   d. Click **Application policy sets > *policy_set_name* > WS-Security > Main policy > Request message part protection**.
   e. Click on **app_signparts** in the Signed parts box, then click the **Delete** button.
3. Update the protection part specified for request:app_encparts. By default, this message protection part encrypts the body and signature elements, and must be modified to encrypt the Username token.
   a. Click **Application policy sets > *policy_set_name* > WS-Security > Main policy > Request message part protection > Encrypted part - app_encparts > Edit**.
   b. Delete the existing elements in the Elements in part panel, then add two XPath expressions for encrypting the Username token.

      Expression 1:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
        and local-name()='Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
        and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
        and local-name()='Security']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
        and local-name()='UsernameToken']
```

      Expression 2:

```
/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope'
        and local-name()='Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope'
        and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
        and local-name()='Security']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
        and local-name()='UsernameToken']
```

What to do next

The second part of the process is to configure the client and server bindings.

1. Configure the client binding, as follows:
   a. Attach the policy to a service resource and create a new binding for that resource that includes the WSSecurity policy.

b. Click on **WSSecurity** in the new binding to display the main WSSecurity binding panel. For example, click **Enterprise Applications > WSSampleServiceSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security**.
c. Click **Authentication and protection**.
d. Click **AsymmetricBindingRecipientEncryptionToken0** under Protection tokens.
e. Click **Apply**.
f. Click **Callback handler**.
g. Select **Custom** from the Keystore menu.
h. Click **Custom keystore configuration**.
i. Enter the keystore path. For example: ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks.
j. Select **JCEKS** for the Type.
k. Enter the password in the Password and Confirm password fields. For example, **storepass**.
l. Enter a Key Name. For example, **CN=Bob, O=IBM, C=US**.
m. Enter a Key Alias. For example, **bob**.
n. Enter the password for the keypass in the Password and Confirm password fields.
o. Click **OK**.
p. Click **OK** again.
q. Click **OK** one more time to return to the **Enterprise Applications > WSSampleServicesSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security > Authentication and protection** panel.
r. The status of AsymmetricBindingRecipientEncryptionToken0 should display as **Configured**.

2. Modify the encrypted parts settings for the client binding, as follows:
a. Click **request:app_encparts** under Request message signature and encryption protection.
b. Enter a Name. For example, **MyEncPart**.
c. Click **New** under Key information.
d. Fill in a Name. For example, **MyEncKeyInfo**.
e. Click **OK**.
f. Select **MyEncKeyInfo** (or the name that you specified for the encrypted part) from the Available box and click **Add**. MyEncKeyInfo appears in the **Assigned** box.
g. Click **OK** to return to the **Enterprise Applications > WSSampleServicesSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security > Authentication and protection** panel.
h. The status of request:app_encparts should display as **Configured**.

3. Configure the Username token settings in the client binding, as follows:
a. Click **request:myUserNameToken** under Authentication tokens.
b. Click **Apply**.
c. Click **Callback handler**.
d. Specify the User name. For example, **LDAPSunuser6**.
e. Specify the password, and confirm the password.
f. Click **OK**.
g. Click **OK** again.
h. The status of request:myUserNameToken should now display as **Configured**.
i. Click **Save** to save your client bindings.

4. Configure the server binding, as follows:
a. Attach the policy to a service resource and create a new binding for that resource that includes the WSSecurity policy.

b. Click on **WSSecurity** in the new binding to display the main WSSecurity binding panel. For example, click **Enterprise Applications > WSSampleServiceSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security**.
c. Click **Authentication and protection**.
d. Click **AsymmetricBindingRecipientEncryptionToken0** under Protection tokens.
e. Click **Apply**.
f. Click **Callback handler**.
g. Select **Custom** from the Keystore menu.
h. Click **Custom keystore configuration**.
i. Enter the keystore path. For example: ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks.
j. Select **JCEKS** for the Type.
k. Enter the password in the Password and Confirm password fields. For example, **storepass**.
l. Enter a Key Name. For example, **CN=Bob, O=IBM, C=US**.
m. Enter a Key Alias. For example, **bob**.
n. Enter the password for the keypass in the Password and Confirm password fields.
o. Click **OK**.
p. Click **OK** again.
q. Click **OK** one more time to get return to the **Enterprise Applications > WSSampleServicesSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security > Authentication and protection** panel.
r. The status of AsymmetricBindingRecipientEncryptionToken0 should display as **Configured**.

5. Modify the encrypted parts settings for the server binding, as follows:
   a. Click **request:app_encparts** under Request message signature and encryption protection.
   b. Enter a Name. For example, **MyEncPart**.
   c. Click **New** under Key information.
   d. Fill in a Name. For example, **MyEncKeyInfo**.
   e. Click **OK**.
   f. Select **MyEncKeyInfo** (or the name that you specified for the encrypted part) from the Available box and click **Add**. MyEncKeyInfo appears in the **Assigned** box.
   g. Click **OK** to return to the **Enterprise Applications > WSSampleServicesSei > Service provider policy sets and bindings >** ***binding_name*** **> WS-Security > Authentication and protection** panel.
   h. The status of request:app_encparts should display as **Configured**.
6. Configure the Username token settings in the server binding, as follows:
   a. Click **request:myUserNameToken** under Authentication tokens.
   b. Click **Apply**.
   c. Click **Callback handler**.
   d. Click **OK**.
   e. Click **OK** again.
   f. The status of request:myUserNameToken should display as **Configured**.
   g. Click **Save** to save the server bindings.

Configuring the username and password for WS-Security Username or LTPA token authentication

When using the Username WSSecurity default policy set, you must configure the username and password for username token authentication separately from the security settings defined in the bindings.

About this task

When you install a JAX-WS application and attach the default Username WSSecurity default policy set, the next step is to configure the general provider sample binding for the JAX-WS provider, and the general client sample binding for the JAX-WS client. However, the binding file for the default client sample binding does not include a username or password for token authentication. Since the username and password is not available from the target deployed system, you must specify a valid username and password in your environment using the administrative console.

1. Log in to the administrative console, then click **Services** → **Policy sets** → **General client policy set bindings**.
2. Click **Client sample** to edit the binding.
3. Click **WS-Security**.

   Add basic authentication information, such as username and password, to the general client sample bindings for any policy set that uses a Username token or LTPA token, including:

   - Username SecureConversation
   - Username WS-I RSP
   - LTPA SecureConversation
   - LTPA WS-I RSP
   - LTPA WSSecurity default
4. Click **Authentication and protection**.
5. In the Authentication tokens table, click **gen_signunametoken** to edit the username token settings.
6. Click **Callback handler** in the Additional Bindings section.
7. Enter the appropriate username and password information for your environment in the **User name** and **Password** fields.
8. Enter the password a second time in the **Confirm Password** field, then click **Apply**.
9. Repeat steps 5 through 8 for the gen_signltpatoken LTPA token generator.

Results

Note: This administrative console panel applies only to Java™ API for XML Web Services (JAX-WS) Web services.

Related reference

Callback handler settings
Use this page to configure callback handler settings, which determine how security tokens are acquired from messages headers.

Configuring default Web services security bindings

WebSphere Application Server provides support for a set of default Web services security bindings for applications. A set of bindings is a named object that is associated with a specific policy set and service resource attached to the policy set.

About this task

Bindings contain environment and platform specific information, such as the following types of information:

- Keys used for signature and encryption
- Keystore information
- Authentication information
- Persistent information

In WebSphere Application Server Version 7.0, there are two types of bindings, application specific bindings and general bindings. Typically, bindings are specific to the application or the platform, and they are not shared.

General bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings. The general bindings that are shipped with WebSphere Application Server are initially set as the default bindings, but you can choose a different binding as the default, or change the level of binding that should be used as the default, for example, from cell level binding to server level binding. Default bindings are used when no application specific binding or trust service binding has been assigned to a policy set attachment. For more information, see the topic General JAX-WS default bindings for Web services security. For a description of the general sample bindings that are included with WebSphere Application Server, and used with the JAX-WS programming model, read the topic General sample bindings for JAX-WS applications.

To create general bindings:

1. Log in to the administrative console and navigate to the general provider policy set and bindings panel, or the general client policy set and bindings panel
   - Click **Services > Policy sets > General provider policy set bindings**.
   - Click **Services > Policy sets > General client policy set bindings**.
2. Click **New**.

Results

Policy set bindings contain platform-specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Each policy set attachment to a service provider or service client must have exactly one binding. When you create a policy set attachment, the general default bindings are used initially. When general bindings are used in association with a policy set attachment, the cell-level general bindings are applied at run time. If application server level bindings exist, the server-level general bindings override the cell-level definition. General bindings specify configuration for both service client and service provider attachments and the general bindings are not tailored to a specific policy set or application. When you define server-level general bindings, the binding begins in a completely unconfigured state. You must add the policy, and then fully configure the bindings for each added policy.

An application specific binding is a named binding that you create. Application specific bindings enable you to provide platform-specific configuration information for specific policy set attachments. When you create an application specific binding, the available binding configuration options are tailored to the definitions in the attached policy set. You can reuse application specific bindings for multiple service resources within an application. For example, if you create a trust service specific binding, that binding can be reused only for trust service attachments. When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. For each policy, such as WS-Security or HTTP Transport, where you want to override the general binding, you must add the policy, and then fully configure the bindings for each added policy.

Note: Only use the sample default bindings in a testing environment. Do not use sample default bindings in a production environment. Default bindings contain sample key files that must be customized before use in a production environment.

See the topic Defining and managing service client or provider bindings for more information about bindings.

General JAX-WS default bindings for Web services security

General bindings are used as the default bindings at the cell level or server level, or for multiple domains, at the domain level. The general bindings that are included with WebSphere Application Server are initially set as the default bindings. However, you can choose a different binding as the default, or change the level of binding that is used as the default, for example, from cell-level binding to server-level binding.

Policy set bindings contain platform-specific information, such as keystore, authentication information or persistent information, required by a policy set attachment. In WebSphere Application Server Version 7.0, there are two types of bindings: application-specific bindings, and general bindings. Both types of bindings are supported for WS-Security policy sets. General bindings can be used as default bindings, and can also be shared across multiple applications and for trust service attachments. There are two types of general bindings: one for service providers and one for service clients. You can define multiple general bindings for the provider and also for the client.

Note: The configuration of the default cell level and default server level bindings has changed in WebSphere Application Server Version 7.0. Previously, you could configure only one set of default bindings for the cell, and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general provider bindings and one or more general client bindings. However, only one general provider binding and one general client binding can be designated as the default.

Default bindings are used when no application-specific binding or trust service binding has been assigned to a policy set attachment. You can choose the general provider and general client bindings, which are used as the default bindings for the cell. These are the global security settings. Likewise, you can choose the general provider and general client bindings, which are used as the default bindings for a server. For specific information about selecting bindings, see the topic Defining and managing policy set bindings.

In an environment with multiple security domains, you can also choose the general provider and general client bindings, which are used as the default bindings for a domain. If you do not choose a binding to be the default for a server, the default bindings for the domain in which the server resides are used. If you do not choose a binding to be the default for a domain, the default bindings for the cell (global security) are used. You must choose default provider and default client bindings for the cell.

The general bindings that are included with WebSphere Application Server are initially set as the cell default bindings. You cannot delete a binding that has been selected as the default binding for server, a domain, or the cell. Before you delete a binding that is selected as the default, you must select a different default binding, or specify that the defaults for the cell (global security) should be used.

The following default bindings shipped are included with WebSphere Application Server Version 7.0:

- Provider sample
- Client sample
- Version 6.1 default policy set bindings

  The Version 6.1 bindings are used only if a WebSphere Application Server Version 6.1 Feature Pack for Web Services application is installed within the WebSphere Application Server Version 7.0 environment. For more information on these bindings, see the topic Version 6.1 default policy set bindings.

Note: Do not use the provider and client sample bindings that are included with WebSphere Application Server in their current state in a production environment. You must modify these bindings to meet your security needs before using them in a production environment by making a copy of the bindings and then modifying the copy. For example, change the key and keystore settings to ensure security, and modify the binding settings to match your environment.

For a detailed description of the general sample bindings, see the topic General sample bindings for JAX-WS applications.

To define and manage general bindings, in the administrative console click **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**. To manage bindings for the cell or the domain, click **Services > Policy sets > Default policy set bindings**. The general service provider and client bindings have independent settings that you can customize to meet the needs of your environment. To learn more about general bindings, read the topic Defining and managing policy set bindings.

In addition to choosing default bindings for the cell (global security), you can also choose the general provider and general client bindings that you want to use as the default bindings for a server. When are using the JAX-WS programming model and want to specify the server default bindings, log on to the administrative console and click **Servers > Server Types > WebSphere application servers > *server_name***. In the Security section of the console page, click **Default policy set bindings**.

Web services security API programming model

The application server programming model provides Web Services Security programming application programming interfaces (WSS API) for securing SOAP messages.

The API programming model is an interface-based programming model that is based on Web Services Security Version 1.1 standards, but the design also includes support for Web Services Security Version 1.0 for securing SOAP messages. The WSS API programming model implementation is a simplified version, which is based on an early draft proposal of JSR-183, which is the JSR for defining Java API binding for Web Service Security. By design, because the application code is programmed to the interface, any application code that is programmed with the open source implementation should be able to run on the WebSphere Application Server with minimal changes or no changes at all.

The configuration model for Web services has also been redesigned from a deployment descriptor model to a policy set model. Web Service Security can be enabled by either using a policy set that is configured by using the administrative console, or by using the WSS API for configuration. The functions provided by the policy set configurations are the same as the functions supported by the WSS API for the Web Service Security run time. However, the security policy that is defined using policy sets has a higher priority over the WSS API. When the WSS API and the policy set are both used in the application, the default behavior is for the security policy from the policy set to be enforced and the WSS API to be ignored. To use the WSS API in the application, you must make sure that there is no policy set attached to the application or to the application resources, or make sure there is no security policy in the attached policy set.

You can still use your existing JAX-RPC applications with Web Service Security; however, those applications cannot take advantage of the Web Services Security Version 1.1 functions, such as configuring the security policy using a policy set, OM filter performance improvements, WSS API, Web Services Secure Conversation (WS-SecureConversation), Kerberos token and the associated SHA-1 key for message protection and identity propagation, and Web Services Trust (WS-Trust) features.

In order to take advantage of the Web Services Security Version 1.1 functions, you must rewrite an existing JAX-RPC application as a JAX-WS application, manually re-configure the security constraints to a policy set, and perform code migration of the DOM-based SPIs to the OM-based SPIs.

For example, when using the JAX-WS programming model, the improved design of the pluggable token framework allows the same security implementation to be used for both the API and policy sets. The framework uses the JAAS Login Module and JAAS Callback Handler for token creation and token validation.

What is supported when using the WSS APIs

The WSS API can only be used on the client. You can use the Java SE 6 client, the J2EE Application client, or a server client (a service provider acting as client) using the API to secure SOAP message with message-level security.

You should have Web Services Security (WSS) knowledge to use the WSS APIs. Before using the WSS API, keep in mind that the WSS API:

- Are Java-based interfaces.
- Are implemented by using a factory model (WSSFactory).
- Supports the WS-Security Version 1.0 and 1.1 standards, which include the Username and X.509 token profiles, Versions 1.0 and 1.1.
- Are very XML centric.
- Include an object-oriented design which simplifies the APIs.
- Are task oriented and allow common usage scenarios, such as: signing the body and encrypting the SOAP message body content.
- Are flexible and extensible, and they let you to extend the token type support.
- Are based on the provider framework and allow the use of different data models to be used, such as: AXIOM or DOM.
- Provides application programmer with better control and flexibility in applying WSS in their applications.

The default values for the WSS API are predefined and are part of the Web services security run time. Default values are provided for:

- The duration of the timestamp
- The signing algorithm, canonicalization algorithm, digest method, transform algorithm, security token reference method and signed parts such as the SOAP body, Web Services Addressing headers and the time stamp.
- The key encryption algorithm, data encryption algorithm, security token reference method, and encrypted parts such as the SOAP body content.

The signature validation has similar default values as the signature (signing information). Similarly, decryption has similar default values as encryption.

What is not supported when using the WSS APIs

The WSS API provided with the application server does not support the following function:

- The application programming model is JAX-WS, meaning JAX-RPC (JSR-109) applications are not supported.
- The WSS API is available in the synchronous message exchange of the JAX-WS client application. However, the WSS API are not supported for the asynchronous client.
- WSS API support is available only for the requester and not for the provider.
- The identity assertion semantic programming model is not supported in the WSS API because identity assertion is not part of the Web Service Security Version 1.0 standard. However, you can use the WSS API to add Identity Assertion semantic in the token processing.

WS-Trust and WS-SecureConversation scenarios

There are several ways to secure the WS-Trust SOAP messages:

- Using the bootstrap policy defined in the policy set.
- Using the WSS API, which supports WS-SecureConversation.
- Enabling dynamic policy for the provider so that the client can retrieve the provider-side policy at run time.

An application would use the WSS API to acquire a security context token for programmatic API-based secure conversation. The WebSphere Application Server trust service provides an application the ability to request a security token for access to a service. The scope and focus of the trust service is only for a WebSphere Application Server Security Context Token (SCT) for WS-SecureConversation.

The WS-SecureConversation and WS-Trust scenarios focus on the inter-operability functions, such as the configuration and runtime interaction of various components. You would use the WSS API to secure the bootstrap RST and RSTR to acquire the security context token from the trust service. After acquiring the security context token, a Derived Key Token is created by using the WSS API. Then the Derived Key Token can be used for signature and encryption.

See the examples that describe the following security context token usage scenarios:

- How to establish the security context token to secure WS-SecureConveration
- How to establish the security context token to secure WS-ReliableMessaging

There are two conditions when using the WSS API to secure the SOAP message with Web Service Security:

- Generation of the secure SOAP message, which is in the request generator application code.
- Consuming of the secured SOAP message, which is in the response consumer application code.

In both cases, a Java exception class com.ibm.websphere.wssecurity.wssapi.WSSException is provided if an error is encountered.

Web services client security context

When the JAX-WS client invokes Web services, the current security context that is constructed by the security handler is stored in the RequestContext object. By default, the security context in the JAX-WS Web services client runtime environment is reconstructed for the next Web services request invocation. You can preserve the security context for subsequent Web services invocations. An example of this is a scenario where the security policy requires the client to send a username security token with the user name and password. When the client sends the first request to invoke the service, you are prompted to enter the required user name and password. The user name and password is saved in a Username SecurityToken token in a Subject in the security context. To avoid being prompted to enter the same user name and password again in subsequent request invocations, you can preserve the security context. There are two methods to preserve the security context: 1) configure the client run time to automatically preserve the client security context for subsequent request invocations; or 2) preserve the security context manually.

To configure the JAX-WS client run time environment to automatically preserve the security context, set the Java system property `com.ibm.websphere.wssecurity.context.management` to true. When this system property is true, the JAX-WS client run time copies the security context constructed by the security handler to the RequestContext automatically, and the context is used for subsequent request invocations.

To manually preserve the security context, use the following sample code:

```
// First request
Service svc = Service.create(...);
svc.addPort(...);
Dispatch<String> dispatch = svc.createDispatch(...);
Map<String, Object> requestContext = dispatch.getRequestContext();
String response = dispatch.invoke(body.toString());

Object securityContext = requestContext.get(com.ibm.wsspi.websvcs.Constants.WEBSPHERE_SECURITY_CONTEXT);

// Subsequent request

Dispatch<String> dispatch = svc.createDispatch(...);
Map<String, Object> requestContext = dispatch.getRequestContext();
Object securityContext = requestContext.put(com.ibm.wsspi.websvcs.Constants.WEBSPHERE_SECURITY_CONTEXT, securityContext);
```

Service Programming Interfaces (SPI)

The Web Services Security service programming interface (WSS SPI) provides programming interfaces for securing Web services security.

The Web services security specification provides a flexible framework for building secure Web services to implement message content integrity and confidentiality. The specification does not define specific token

formats, but instead associates separate profile documents that define various security token formats and semantics for using those tokens. The Web services security service programming model supports the flexible framework by providing extension points to integrate with new token formats, and with methods to obtains keys needed for message protection. Web services security uses this programming model to implement support for the standard X.509 token profile, the Username token profile, and the Kerberos token profiles. The programming model is also used to implement support for the LTPA security token, and for new security token types.

The Web service security run time token generation and token consuming Service Programming Interfaces (SPI) have been redesigned so that the same security token interface and JAAS Login Module implementation can be used for both the WSS API and the SPI. The WSS SPI for the service provider extends the security token types and provides keys and deriving keys for signing, signature verification, encryption and decryption.

The Web services security service programming model provides mechanisms to process custom security tokens, to use custom token in signing and encryption, and to retrieve encryption and signing keys. The Web services security service programming interfaces for the JAX-RPC run time, and for the JAX-WS run time, are similar, but not identical.

JAX-RPC run time

The plug-in programming interfaces for the JAX-RPC run time consist of the TokenGenerator, KeyLocator, and JAAS CallbackHandler for outbound message processing, and the TokenConsumer, KeyLocator, and JAAS LoginModule for inbound message processing.

Token Generator, KeyLocator, and Callback Handler
: The TokenGenerator class is responsible for formatting the security token to the XML element. This class calls the CallbackHandler class that is specified in the TokenGeneratorConfig object, which obtains the security token input data, and then stores the resulting security token in the Subject object private credentials.

Token Consumer, KeyLocator and JAAS LoginModule
: The KeyLocator class is responsible for obtaining the required key for signing and encrypting SOAP message elements from a key store that is specified by the KeyStoreConfig and the KeyLocatorConfig configuration. The TokenConsumer class extracts the token data from the XML security token representation, and stores it in the JAAS Subject using a JAAS LoginModule. The specified KeyLocator class is invoked to find the required key for verifying the digital signature and decrypting the SOAP message elements.

JAX-WS run time

The plug-in programming interfaces for the JAX-WS run time are based on the JAAS programming model for both inbound and outbound SOAP message processing. The JAAS LoginModule and CallbackHandler are responsible for processing the security tokens in SOAP messages. The Login Module and Callback Handler both retrieve and generate tokens, and store the SecurityToken objects in the run time. They replace the functionality of the TokenGenerator, TokenConsumer, and KeyLocator interfaces.

Due to the differences in the programming models, any WebSphere Application Server or custom SPI implementation from the Web Service Security Version 6.1 run time is not supported to run on the Web Service Security run time with the Version 6.1 Feature Pack for Web Services, or the Version 7.0 Web Service Security runtime. However, the Web Service Security Version 6.1 run time is supported simultaneously with the Version 6.1 Feature Pack for Web Services, meaning the Version 6.1 SPI implementations are still supported through the original run time. Before using the new Web Service Security run time, a code migration is required to reprogram the Version 6.1 DOM-based SPIs to the AXIOM-based SPIs in the Feature Pack for Web Services, before the SPI can be used.

Securing Web services applications using the WSS APIs at the message level

Standards and profiles address how to provide protection for messages that are exchanged in a Web service environment. Web services security is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens.

Before you begin

To secure Web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today's business environment is the ability to interoperate between formerly incompatible security technologies in heterogeneous environments. The complete Web services security protocol stack and technology roadmap is described in the Web services roadmap.

About this task

The Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security: SOAP Message Security Version 1.1 specification is the basic messaging transport for all Web services. SOAP 1.2 adds extensions to the existing SOAP 1.1 extensions so that you can build secure Web services. Attachments can be added to SOAP messages by using Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) instead of the SOAP with Attachments (SWA) profile.

The OASIS Web Services Security (WS-Security) Version 1.1 specification is the building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models. Web services security for WebSphere Application Server is based on specific standards that are included in the OASIS Web Services Security Version 1.1 specification and profiles.

The Version 1.1 specification defines additional facilities for protecting the integrity and confidentiality of a message. The Version 1.1 specification also provides the mechanisms for associating security-related claims with the message. The Web Services Security Version 1.1 standards that are supported by WebSphere Application Server include the signature confirmation, encrypted header elements, the Username Token Profile and the X.509 Token Profile. The Username Token Profile and the X.509 Token Profile have been updated as Version 1.1 profiles. For the X.509 Certificate Token Profile, one new type of security token reference is the Thumbprint reference, which is specified in the binding.

XML Schema, Part 1 and Part 2 are specifications that explain how schemas are organized in XML documents. The two WS-Security Version 1.0 schemas have been updated to the Version 1.1 specifications plus a new Version 1.1 schema has been added. Note that the Version 1.1 schema does not replace the Version 1.0 schema but instead builds upon it by defining an additional set of capabilities within a Version 1.1 namespace.

You can use the following methods to configure Web services security and to define policy types to secure the SOAP messages:

- **Use the administrative console to configure policy sets.**

  This method uses the bootstrap policy that is defined in the policy set. You can use policy sets, or assertions about how services are defined, to simplify your security configuration for Web services. You can use the administrative console to create, modify, and delete custom policy sets. A set of default policy sets are available.

  For example, you can define the bootstrap policy in the policy set to secure the Web Services Trust (WS-Trust) SOAP messages.

  You can also use the administrative console to perform policy set management tasks and to secure Web Services using encryption, signing information, and security tokens.

The following steps high-level steps describe how to configure WebSphere Application Server to use WS-Security and to secure the SOAP messages using the administrative console. The generator and consumer tasks that are discussed in the following steps use WS-Security Versions 1.0 and 1.1.

- Create and configure the application policy sets or the system policy sets for trust service.
- Define the policy types to be used to secure the SOAP messages when creating and configuring the policy sets.
- Configure the policy set binding. Select either the symmetric or asymmetric binding assertion to describe the token type and the algorithm to be used for message protection.
- Assemble your Web services security-enabled application by using an assembly tool.

- **Use the Web Services Security APIs (WSS API) to configure the SOAP message context (only for the client)**

  WebSphere Application Server uses a new API programming model. In addition to the existing JAX-RPC programming model, a new programming model, Java API for XML Web Services (JAX-WS), has been added. The JAX-WS programming standard aligns with the document-centric messaging model and replaces the remote procedure call programming model defined by the Java API for XML-based RPC (JAX-RPC) specification.

  For example, an application could create system policy sets and then use the WebSphere Application Server WSS API to acquire the security context token for programmatic API-based Web Services Secure Conversation (WS-SecureConversation).

  You can also use the administrative console to perform the encryption, signing, and token configuration tasks that the WSS APIs perform to secure Web services.

  The following high-level steps describe how to configure WebSphere Application Server to use WS-Security and to secure the SOAP messages using the WSS APIs. The generator and consumer tasks that are discussed in the following steps use WS-Security Versions 1.0 and 1.1.

  - Use the WSSSignature API to configure the signing information for the request generator (client side) binding.

    Different message parts can be specified in the message protection for a request on the generator side. The default required parts are BODY, ADDRESSING_HEADERS and TIMESTAMP.

    The WSSSignature API also specifies the different algorithm methods to be used with the signature for message protection. The default signature method is RSA_SHA1. The default canonicalization method is EXC_C14N.
  - Use the WSSSignPart API if you want to change the digest method and the transform method.

    The default signed parts are WSSSignature.BODY, WSSSignature.ADDRESSING_HEADERS and WSSSignature.TIMESTAMP.

    The WSSSignPart API also specifies the different algorithm methods to be used if you added or changed the signed parts. The default digest method is SHA1. The default transform method is TRANSFORM_EXC_C14N. For example, use the WSSSignPart API if you want to generate the signature for the SOAP message using the SHA256 digest method instead of the default value of SHA1.
  - Use the WSSEncryption API to configure the encryption information on the request generator side.

    The encryption information on the generator side is used for encrypting an outgoing SOAP message for the request generator (client side) bindings. The default targets of encryption are BODY_CONTENT and SIGNATURE.

    The WSSEncryption API also specifies the different algorithm methods to be used to protect message confidentiality. The default data encryption method is AES128. The default key encryption method is KW_RSA_OAEP.
  - Use the WSSEncryptPart API if you want to set the transform method only.

    For example, if you want to change the data encryption method from the default value of AES128 to TRIPLE_DES.

    No algorithm methods are required for encrypted parts.

- Use the WSS API to configure the token on the generator side.

  The requirements for the security token depend on the token type. The JAAS Login Module and the JAAS CallbackHandler are responsible for creating the security token on the generator side. Different standalone tokens can be sent in request and response. The default token is the X509Token. The other token that can be used for signing is the DerivedKeyToken, which is used only with Web Services Secure Conversation (WS-SecureConversation).
- Use the WSSVerification API to verify the signature for the response consumer (client side) binding.

  Different message parts can be specified in the message protection for a response on the consumer side. The required targets for verification are BODY, ADDRESSING_HEADERS and TIMESTAMP.

  The WSSVerification API also specifies the different algorithm methods to be used for verifying the signature and for message protection. The default signature method is RSA_SHA1. The default canonicalization method is EXC_C14N.
- Use the WSSVerifyPart API to change the digest method and the transform method. The required verify parts are WSSVerification.BODY, WSSVerification.ADDRESSING_HEADERS and WSSVerification.TIMESTAMP.

  The WSSVerifyPart API also specifies the different algorithm methods to be used if you added or changed the verification parts. The default digest method is SHA1. The default transform method is TRANSFORM_EXC_C14N.
- Use the WSSDecryption API to configure the decryption information for the response consumer (client side) binding.

  The decryption information on the consumer side is used for decrypting an incoming SOAP message. The targets of decryption are BODY_CONTENT and SIGNATURE. The default key encryption method is KW_RSA_OAEP.

  No algorithm methods are required for decryption.
- Use the WSSDecryptPart API if you want to set the transform method only.

  For example, if you want to change the data encryption method from the default value of AES128 to TRIPLE_DES.

  No algorithm methods are required for decrypted parts.
- Use the WSS API to configure the token on the consumer side.

  The requirements for the security token depend on the token type. The JAAS Login Module and the JAAS CallbackHandler are responsible for validating (authenticating) the security token on the consumer side. Different standalone tokens can be sent in request or response.

  The WSS API adds the information for the candidate token that is used for decryption. The default token is X509Token.

- **Use the wsadmin administrative scripting tool to configure policy sets.**

  This method allows you to create, manage, and delete policy sets from the command-line or to create scripts to automate your tasks. You can use the wsadmin tool and the PolicySetManagement command group to manage default policy sets, create custom policy sets, configure policies, and manage attachments and bindings. For more information, use the policy set scripting topics in the information center.

To secure Web services with WebSphere Application Server, you must configure the generator and the consumer security constraints. You must specify several different configurations. Although there is no specific sequence to specify these different configurations, some configurations reference other configurations. For example, decryption configurations reference encryption configurations.

Results

After completing these high-level steps for WebSphere Application Server, you have secured Web services by configuring policy sets and by using the WSS API to configure encryption and decryption, the signature and signature verification information, and the consumer and generator tokens.

Securing messages at the request generator using WSS APIs:

You can secure SOAP messages by configuring signing information, encryption, and generator tokens to protect message integrity, confidentiality, and authenticity, respectively. This request (client-side) generator configuration defines the Web services security requirements for the outgoing SOAP message request.

Before you begin

To secure Web services with WebSphere Application Server, you must configure the generator and the consumer security constraints. Therefore, in addition to securing messages at the request generator level, you must also secure messages at the response consumer level.

About this task

The request (client-side) generator configuration requirements involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches security tokens.

To secure Web service applications, you must specify several different configurations. Although there is no specific sequence to specify these different configurations, some configurations reference other configurations. For example, decryption configurations reference encryption configurations.

You can use the following interfaces to configure Web services security and to define policy types to secure the SOAP messages:

- Use the administrative console to configure policy sets.
- Use the Web Services Security APIs (WSS API) to configure the SOAP message context (only for the client)

The following high-level steps use the WSS APIs:

- Configure generator signing to protect message integrity.
- Configure encryption to protect message confidentiality.
- Attach generator tokens to protect message authenticity.

Results

After completing these procedures, you have secured messages at the request generator level.

What to do next

Next, if not already configured, secure messages with signature verification, decryption, and consumer tokens at the response consumer (client-side) level.

Configuring encryption to protect message confidentiality using the WSS APIs:

You can configure encryption information for the client-side request generator (sender) bindings. Encryption information is used to specify how the generators (senders) encrypt outgoing SOAP messages. To configure encryption, specify which message parts to encrypt and specify which algorithm methods and security tokens are to be used for encryption.

Before you begin

Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the message flowing across the Internet. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the correct target. Prior to configuring encryption, familiarize yourself with XML encryption.

About this task

For encryption, you must specify the following:

- Which parts of the message are to be encrypted.
- Which encryption algorithms to specify.

To configure encryption and encrypted parts on the client side, use the WSSEncryption and WSSEncryptPart APIs, or configure policy sets using the administrative console.

WebSphere Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

WebSphere Application Server uses encryption information for the default generator to encrypt parts of the SOAP message. The WSSEncryption API configures the following required parts as encrypted parts.

Table 6. Required encrypted parts

Encryption parts	Description
Keywords	Keywords are used to add the encrypted parts to the SOAP message.
XPath expression	An XPath expression is used to add the encrypted parts to the SOAP message.
WSSEncryptPart object	This object adds the encrypted parts to the SOAP message.
WSSSignature object	This object adds the signature component as an encrypted part.
Header	This part adds the header in the SOAP header, specified by QName, as an encryption part.
Security token object	This object adds the security token as an encryption part.

Web Services Security API (WSS API) supports symmetric encryption, by using a shared key, only when Web Services Secure Conversation (WS-SecureConversation) is used.

The WSS APIs allow the use of either keywords or an XPath expression to specify the parts of the message that are to be encrypted. WebSphere Application Server supports the use of the following keywords:

Table 7. Supported encryption keywords

Keyword	References
BODY_CONTENT	The keyword for the contents of the SOAP message body as an encryption target.
SIGNATURE	The keyword for the signature component as an encryption target.

If configuring using the WSS APIs, the WSSEncryption and WSSEncryptPart APIs complete these high-level steps:

1. Use the WSSEncryption API to configure encryption. The WSSEncryption API performs these tasks by default:
   a. Generates the callback handler.
   b. Generates the generator security token object.
   c. Adds the security token reference type.
   d. Adds the signature component.
   e. Adds the WSSEncryptPart object.
   f. Adds the parts to be encrypted. Adds the default parts as targets of encryption by using keywords and XPath expressions.

g. Adds the header in the SOAP message, specified by QName.
h. Sets the default data encryption method.
i. Specifies whether the key is to be encrypted using a Boolean value.
j. Sets the default key encryption method.
k. Selects a part reference.
l. Sets the MTOM optimization Boolean value.

2. Use the WSSEncryptPart API to configure encrypted parts or add a transform method. The WSSEncryptPart API performs these tasks by default:
   a. Sets the encrypted parts specified by using keywords or an XPath expression.
   b. Sets the encrypted parts specified by an XPath expression.
   c. Sets the signature component object, WSSSignature.
   d. Sets the header in the SOAP message, specified by QName.
   e. Sets the generator security token.
   f. Adds the transform method, if needed.
3. Change from the default values for algorithm or message parts, as needed. For example: you could change one or more of the following items:
   - Change the data encryption algorithm from the default value of AES 128.
   - Change the key encryption algorithm from the default value of KW_RSA_OAEP.
   - Specify to not encrypt the key (false).
   - Change the security token type from default of X.509 token.
   - Change the security token reference type from the default value of SecurityToken.REF_STR.
   - Only use BODY_CONTENT as an encryption part and not use SIGNATURE also.
   - Turn MTOM optimization on (true).

Results

The encryption information is configured for the generator binding.

Example

The following is an example of the WSSEncryption API:

```
WSSFactory factory = WSSFactory.getInstance();
WSSGenerationContext gencont = factory.newWSSGenerationContext();

X509GenerateCallbackHandler callbackhandler = generateCallbackHandler();
SecurityToken token = factory.newSecurityToken(X509Token.class, callbackHandler);
WSSEncryption enc = factory.newWSSEncryption(token);

gencont.add(enc);
```

What to do next

You must configure similar decryption information for the client-side response consumer (receiver) bindings, if you have not already configured the information.

Next, review the WSSEncryption API process.

Encrypting the SOAP message using the WSSEncryption API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web Services Security APIs (WSS API). To configure the client for request encryption on the generator side,

use the WSSEncryption API to encrypt the SOAP message. The WSSEncryption API specifies which request SOAP message parts to encrypt when configuring the client.

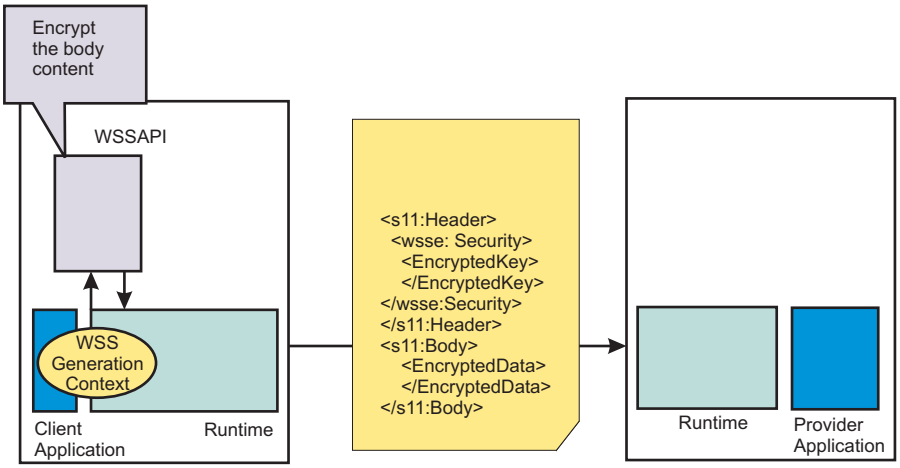
Before you begin

You can use the WSS API or use policy sets on the administrative console to enable encryption and add generator security tokens in the SOAP message. To secure SOAP messages, use the WSS APIs to complete the following encryption tasks, as needed:

- Configure encryption and choose the encryption methods using the WSSEncryption API.
- Configure the encrypted parts, as needed, using the WSSEncryptPart API.

About this task

The encryption information on the generator side is used for encrypting an outgoing SOAP message for the request generator (client side) bindings. The client generator configuration must match the configuration for the provider consumer.



Confidentiality settings require that confidentiality constraints be applied to generated messages. These constraints include specifying which message parts within the generated message must be encrypted, and which message parts to attach encrypted Nonce and timestamp elements to.

The following encryption parts can be configured:

Table 8. Encryption parts

Encryption parts	Description
part	Adds the WSSEncryptPart object as a target of the encryption part.
keyword	Adds the encryption parts using keywords. WebSphere Application Server supports the following keywords: • BODY_CONTENT • SIGNATURE
xpath	Adds the encryption part using an XPath expression.
signature	Adds the WSSignature component as a target of the encrypted part.
header	Adds the SOAP header, specified by QName, as a target of the encrypted part.
securityToken	Adds the SecurityToken object as a target of the encrypted part.

For encryption, certain default behaviors occur. The simplest way to use the WSSEncryption API is to use the default behavior (see the example code).

WSSEncryption provides defaults for the key encryption algorithm, the data encryption algorithm, the security token reference method, and the encryption parts such as the SOAP body content and the signature. The encryption default behaviors include:

Table 9. Encryption decisions

Encryption decisions	Default behavior
Which SOAP message parts to encrypt using keywords	Sets the encryption parts that you can add using keywords. The default encryption parts are the BODY_CONTENT and SIGNATURE. WebSphere Application Server supports using these keywords: • WSSEncryption.BODY_CONTENT • WSSEncryption.SIGNATURE
Which data encryption method to choose (algorithm)	Sets the data encryption method. Both data and key encryption methods can be specified. The default data encryption algorithm method is AES 128. WebSphere Application Server supports these data encryption methods: • WSSEncryption.AES128: http://www.w3.org/2001/04/xmlenc#aes128-cbc • WSSEncryption.AES192: http://www.w3.org/2001/04/xmlenc#aes192-cbc • WSSEncryption.AES256: http://www.w3.org/2001/04/xmlenc#aes256-cbc • WSSEncryption.TRIPLE_DES: http://www.w3.org.2001/04/xmlenc#tripledes-cbc
Whether to encrypt the key (isEncrypt)	Specifies whether to encrypt the key. The values are `true` or `false`. The default value is to encrypt the key (`true`).
Which key encryption method to choose (algorithm)	Sets the key encryption method. Both data and key encryption methods can be specified. The default key encryption algorithm method is key wrap RSA OAEP. WebSphere Application Server supports these key encryption methods: • WSSEncryption.KW_AES128: http://www.w3.org/2001/04/xmlenc#kw-aes128 • WSSEncryption.KW_AES192: http://www.w3.org/2001/04/xmlenc#kw-aes192 • WSSEncryption.KW_AES256: http://www.w3.org/2001/04/xmlenc#kw-aes256 • WSSEncryption.KW_RSA_OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p • WSSEncryption.KW_RSA15: http://www.w3.org/2001/04/xmlenc#rsa-1_5 • WSSEncryption.KW_TRIPLE_DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes
Which security token to specify (securityToken)	Sets the SecurityToken. The default security token type is the X509Token. WebSphere Application Server provides the following pre-configured consumer token types: • Derived key token • X.509 tokens
Which token reference to use (refType)	Sets the type of the security token reference. The default token reference is SecurityToken.REF_KEYID. WebSphere Application Server supports the following token reference types: • SecurityToken.REF_KEYID • SecurityToken.REF_STR • SecurityToken.REF_EMBEDDED • SecurityToken.REF_THUMBPRINT
Whether to use MTOM (mtomOptimize)	Sets Message Transmission Optimization Mechanism (MTOM) optimization for the encrypted part.

1. To encrypt the SOAP message using the WSSEncryption API, first ensure that the application server is installed.

2. The WSS API process for encryption performs these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance
   b. Creates the WSSGenerationContext instance from the WSSFactory instance.
   c. Creates the SecurityToken from WSSFactory used for encryption.
   d. Creates WSSEncryption from the WSSFactory instance using the SecurityToken. The default behavior of WSSEncryption is to encrypt the body content and the signature.
   e. Adds a new part to be encrypted in WSSEncryption if the existing part is not appropriate. After addEncryptPart(), addEncryptHeader(), or addEncryptPartByXPath() is called, the default part is cleared.
   f. Calls the encryptKey(false) if the key is not to be encrypted.
   g. Sets the data encryption method if the default method is not appropriate.
   h. Sets the key encryption method if the default method is not appropriate.
   i. Sets the token reference if the default token reference is not appropriate.
   j. Adds WSSEncryption to WSSConsumingContext.
   k. Calls WSSGenerationContext.process() with the SOAPMessageContext.

Results

If there is an error condition during encryption, a WSSException is provided. If successful, the API calls the WSSGenerationContext.process(), the WS-Security header is generated, and the SOAP message is now secured using Web services security.

Example

The following example provides sample code using methods that are defined in WSSEncryption:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance (step: a)
WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSGenerationContext instance (step: b)
WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Generate the callback handler
X509GenerateCallbackHandler callbackHandler = new
         X509GenerateCallbackHandler(
     "",
     "enc-sender.jceks",
     "jceks",
     "storepass".toCharArray(),
     "bob",
     null,
     "CN=Bob, O=IBM, C=US",
     null);

// Generate the security token used for encryption (step: c)
SecurityToken token = factory.newSecurityToken(X509Token.class , callbackHandler);

// Generate WSSEncryption instance (step: d)
WSSEncryption enc = factory.newWSSEncryption(token);

// Set the part to be encrypted (step: e)
// DEFAULT: WSSEncryption.BODY_CONTENT and WSSEncryption.SIGNATURE

// Set the part specified by the keyword (step: e)
   enc.addEncryptPart(WSSEncryption.BODY_CONTENT);

// Set the part in the SOAP Header specified by QName (step: e)
   enc.addEncryptHeader(new QName("http://www.w3.org/2005/08/addressing",
                                  "MessageID"));
```

```
// Set the part specified by WSSSignature (step: e)
   SecurityToken sigToken = getSecurityToken();
      WSSSignature sig = factory.newWSSSignature(sigToken);
   enc.addEncryptPart(sig);

// Set the part specified by SecurityToken (step: e)
   UNTGenerateCallbackHandler untCallbackHandler =
          new UNTGenerateCallbackHandler("Chris", "sirhC");
      SecurityToken unt = factory.newSecurityToken(UsernameToken.class,
                                                   untCallbackHandler);
   enc.addEncryptPart(unt, false);

// sSt the part specified by XPath expression (step: e)
   StringBuffer sb = new StringBuffer();
      sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         and local-name()='Envelope']");
      sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         and local-name()='Body']");
      sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
         and local-name()='Ping']");
      sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
         and local-name()='Text']");
   enc.addEncryptPartByXPath(sb.toString());

// Set whether the key is encrypted (step: f)
// DEFAULT: true
   enc.encryptKey(true);

// Set the data encryption method (step: g)
// DEFAULT: WSSEncryption.AES128
   enc.setEncryptionMethod(WSSEncryption.TRIPLE_DES);

// Set the key encryption method (step: h)
// DEFAULT: WSSEncryption.KW_RSA_OAEP
   enc.setEncryptionMethod(WSSEncryption.KW_RSA15);

// Set the token reference (step: i)
// DEFAULT: SecurityToken.REF_KEYID
  enc.setTokenReference(SecurityToken.REF_STR);

// Add the WSSEncryption to the WSSGenerationContext (step: j)
  gencont.add(enc);

// Process the WS-Security header (step: k)
gencont.process(msgcontext);
```

Note: The X509GenerationCallbackHandler does not need the key password because the public key is used for encryption. You do not need a password to obtain the public key from the Java keystore.

What to do next

If you have not previously specified which encryption methods to choose, use the WSS API or configure the policy sets using the administrative console to choose the data and key encryption algorithm methods.

Choosing the encryption methods for the generator binding:

To configure the client for request encryption for the generator binding, you must specify which encryption methods to use when the client encrypts the SOAP messages.

Before you begin

Prior to completing these steps, read the XML encryption information to become familiar with encrypting and decrypting SOAP messages.

To specify which algorithm methods are to be used when the client encrypts the SOAP messages, complete the following tasks:

- Use the WSSEncryption API to configure the data encryption algorithm and the key encryption algorithm methods.
- Use the WSSEncryptPart API to configure a transform algorithm method, if needed. The default is no transform algorithm.

About this task

Some of the encryption-related definitions are based on the XML-Encryption specification. The following information defines some data encryption-related terms:

Data encryption method algorithm

Data encryption algorithms specify the algorithm uniform resource identifier (URI) of the data encryption method. This algorithm encrypts and decrypts data in fixed size, multiple octet blocks.

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files.

For the AES256-cbc and the AES192-cbc algorithms, you must download the unrestricted Java™ Cryptography Extension (JCE) policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

Key encryption method algorithm

Key encryption algorithms specify the algorithm uniform resource identifier (URI) of the method to encrypt the key that is used to encrypt data. The algorithm represents public key encryption algorithms that are specified for encrypting and decrypting keys.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property.

The property name is: com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod. The property value is one of the following URIs of the digest method:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmlenc#sha512

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

For the KW-AES256 and the KW-AES192 key encryption algorithms, you must download the unrestricted JCE policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

Table 10. Encryption usage types

Usage types	Description
Data encryption	Specifies the algorithm URI that is used for both encrypting and decrypting data. Encrypts and decrypts data in fixed size, multiple octet blocks.
Key encryption	Specifies the algorithm URI that is used for encrypting and decrypting the encryption key.

Data encryption

WebSphere Application Server supports the following pre-configured data encryption algorithms:

Table 11. Data encryption algorithms

Data encryption name	Algorithm URI
WSSEncryption.AES128 (the default value)	A URI of data encryption algorithm, AES 128: http://www.w3.org/2001/04/xmlenc#aes128-cbc
WSSEncryption.AES192	A URI of data encryption algorithm, AES 192: http://www.w3.org/2001/04/xmlenc#aes192-cbc
WSSEncryption.AES256	A URI of data encryption algorithm, AES 256: http://www.w3.org/2001/04/xmlenc#aes256-cbc
WSSEncryption.TRIPLE_DES	A URI of data encryption algorithm, 3DES: http://www.w3.org.2001/04/xmlenc#tripledes-cbc

Key encryption

WebSphere Application Server supports the following pre-configured key encryption algorithms:

Table 12. Key encryption algorithms

Key encryption name	Algorithm URI
WSSEncryption.KW_AES128	A URI of key encryption algorithm, key wrap AES 128: http://www.w3.org/2001/04/xmlenc#kw-aes128
WSSEncryption.KW_AES192	A URI of key encryption algorithm, key wrap AES 192: http://www.w3.org/2001/04/xmlenc#kw-aes192 **Note:** Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
WSSEncryption.KW_AES256	A URI of key encryption algorithm, key wrap AES 256: http://www.w3.org/2001/04/xmlenc#kw-aes256
WSSEncryption.KW_RSA_OAEP (the default value)	A URI of key encryption algorithm, key wrap RSA OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
WSSEncryption.KW_RSA15	A URI of key encryption algorithm, key wrap RSA 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5
WSSEncryption.KW_TRIPLE_DES	http://www.w3.org/2001/04/xmlenc#kw-tripledes

To configure the encryption and encrypted part algorithm methods, use the WSSEncryption API, or configure policy sets using the administrative console.

Note: Policy sets do not support symmetric key encryption. If you are using the WSS API for symmetric key encryption, you will not be able to interoperate with Web services endpoints that use policy sets.

The WSS API process completes the following high-level steps to specify which encryption methods to use when configuring the client for request encryption:

1. Using the WSSEncryption API, adds the required data encryption algorithm. The data encryption algorithm is used for encrypting or decrypting parts of a SOAP message. Data encryption algorithms specify the algorithm uniform resource identifier (URI) of the data encryption method.

   The client generator configuration must match the configuration for the provider consumer.

The default data encryption algorithm is AES 128. The data encryption name is AES128, and the URI of the data encryption algorithm, is http://www.w3.org/2001/04/xmlenc#aes128-cbc. WebSphere Application Server supports the following pre-configured data encryption algorithms:

- AES 128: http://www.w3.org/2001/04/xmlenc#aes128-cbc

  The AES 128 algorithm is the default data algorithm method.
- AES 192: http://www.w3.org/2001/04/xmlenc#aes192-cbc

  Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

  To use this AES 192-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- AES 256: http://www.w3.org/2001/04/xmlenc#aes256-cbc

  To use this AES 256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- TRIPLEDES: http://www.w3.org/2001/04/xmlenc#tripledes-cbc

2. As needed, changes the WSSEncryption API method to specify another data encryption algorithm. For example, you might add the following code to change from the default AES 128 algorithm to the Triple DES algorithm:

```
// Default data encryption algorithm: AES128
WSSEncryption enc = factory.newWSSEncryption(x509t);
   enc.setEncryptionMethod(EncryptionMethod.TRIPLEDES_CBC);
gencont.add(enc);
```

3. Using the WSSEncryption API, adds the required key encryption algorithm. The key encryption algorithm is used for encrypting the key that is used for encrypting the message parts within the SOAP message. If the encryption key, which is the key that is used for encrypting the message parts, is not encrypted, then the decryption API selects **false** to match the encryption key.

   The client generator configuration must match the configuration for the provider consumer.

   The default key encryption algorithm value is key wrap RSA OAP. The key encryption name is KW_RSA_OAEP, and the URI of the key encryption algorithm is http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p. WebSphere Application Server supports the following pre-configured key encryption algorithms:

   - KW AES128: http://www.w3.org/2001/04/xmlenc#kw-aes128
   - KW AES192: http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use this key wrap AES 192 algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).KW AES 256: http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this key wrap AES 256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - KW RSA OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

     The KW RSA OAEP algorithm is the default key algorithm method.

     When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this algorithm. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5. See more information at http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
   - KW RSA15: http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - KW TRIPLE DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes

Note: For Web Services Secure Conversation, the WSSEncryption API might specify addition key-related information, such as the:
- algorithmName
- keyLength

Results

If there is an error condition, a WSSException is provided. If successful, the API calls the WSSGenerationContext.process(), the WS-Security header is generated, and the SOAP message is now secured using Web services security.

Example

The following example provides sample WSS API code using WSSEncryption.setEncryptionMethod() and WSSEncryption.setKeyEncryptionMethod().

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSGenerationContext instance
   WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Generate callback handler
   X509GenerateCallbackHandler callbackHandler = new
       X509GenerateCallbackHandler(
                                   "",
                                   "enc-sender.jceks",
                                   "jceks",
                                   "storepass".toCharArray(),
                                   "bob",
                                   null,
                                   "CN=Bob, O=IBM, C=US",
                                   null);

// Generate the security token used for encryption
   SecurityToken token = factory.newSecurityToken(X509Token.class , callbackHandler);

// Generate WSSEncryption instance
   WSSEncryption enc = factory.newWSSEncryption(token);

// Set the data encryption method
// DEFAULT: WSSEncryption.AES128
      enc.setEncryptionMethod(WSSEncryption.TRIPLE_DES);

// Set the key encryption method
// DEFAULT: WSSEncryption.KW_RSA_OAEP
      enc.setEncryptionMethod(WSSEncryption.KW_RSA15);

// Add the WSSEncryption to the WSSGenerationContext
   gencont.add(enc);

// Generate the WS-Security header
gencont.process(msgcontext);
```

What to do next

Next, if you want to add a transform algorithm, review the WSSEncryptPart API process task.

Encryption methods:

For request generator binding settings, the encryption methods include specifying the data and key encryption algorithms to use to encrypt the SOAP message. The WSS API for encryption (WSSEncryption)

specifies the algorithm name and the matching algorithm uniform resource identifier (URI) for the data and key encryption methods. If the data and key encryption algorithms are specified, only elements that are encrypted with those algorithms are accepted.

Data encryption algorithms

The data encryption algorithm is used to encrypt parts of the SOAP message, including the body and the signature. Data encryption algorithms specify the algorithm uniform resource identifier (URI) for each type of data encryption algorithms.

The following pre-configured data encryption algorithms are supported:

Table 13. Data encryption algorithms

Data encryption algorithm name	Algorithm URI
WSSEncryption.AES128 (the default value)	A URI of data encryption algorithm, AES 128: http://www.w3.org/2001/04/xmlenc#aes128-cbc
WSSEncryption.AES192	A URI of data encryption algorithm, AES 192: http://www.w3.org/2001/04/xmlenc#aes192-cbc
WSSEncryption.AES256	A URI of data encryption algorithm, AES 256: http://www.w3.org/2001/04/xmlenc#aes256-cbc
WSSEncryption.TRIPLE_DES	A URI of data encryption algorithm, TRIPLE DES: http://www.w3.org/2001/04/xmlenc#tripledes-cbc

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files.

For the AES256-cbc and the AES192-CBC algorithms, you must download the unrestricted Java™ Cryptography Extension (JCE) policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

The data encryption algorithm configured for encryption for the generator side must match the data encryption algorithm that is configured for decryption for the consumer side.

Key encryption algorithms

This algorithm is used to encrypt and decrypt keys. This key information is used to specify the configuration that is needed to generate the key for digital signature and encryption. The signing information and encryption information configurations can share the key information. The key information on the consumer side is used for specifying the information about the key that is used for validating the digital signature in the received message or for decrypting the encrypted parts of the message. The request generator is configured for the client.

Note: Policy sets do not support symmetric key encryption. If you are using the WSS API for symmetric key encryption, you will not be able to interoperate with Web services endpoints using the policy sets.

Key encryption algorithms specify the algorithm uniform resource identifier (URI) of the key encryption method. The following pre-configured key encryption algorithms are supported:

Table 14. Supported pre-configured key encryption algorithms

WSS API	URI
WSSEncryption.KW_AES128	A URI of key encryption algorithm, key wrap AES 128: http://www.w3.org/2001/04/xmlenc#kw-aes128

Table 14. Supported pre-configured key encryption algorithms (continued)

WSS API	URI
WSSEncryption.KW_AES192	A URI of key encryption algorithm, key wrap AES 192: http://www.w3.org/2001/04/xmlenc#kw-aes192 **Note:** Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
WSSEncryption.KW_AES256	A URI of key encryption algorithm, key wrap AES 256: http://www.w3.org/2001/04/xmlenc#kw-aes256
WSSEncryption.KW_RSA_OAEP (the default value)	A URI of key encryption algorithm, key wrap RSA OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
WSSEncryption.KW_RSA15	A URI of key encryption algorithm, key wrap RSA 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5
WSSEncryption.KW_TRIPLE_DES	A URI of key encryption algorithm, key wrap TRIPLE DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes

For Secure Conversation, additional key-related information must be specified, such as:

- algorithmName
- keyLength

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod. The property value is one of the following URIs of the digest method:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmlenc#sha512

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

For the KW-AES256 and the KW-AES192 key encryption algorithms, you must download the unrestricted JCE policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

The key encryption algorithm for the generator must match the key decryption algorithm that is configured for the consumer.

This example provides sample code for encryption to use the Triple DES for the data encryption method and to use RSA1.5 for the key encryption method:

```
// get the message context
Object msgcontext = getMessageContext();

// generate WSSFactory instance
WSSFactory factory = WSSFactory.getInstance();

// generate WSSGenerationContext instance
WSSGenerationContext gencont = factory.newWSSGenerationContext();
```

```
// generate callback handler
X509GenerateCallbackHandler callbackHandler = new X509GenerateCallbackHandler(
  "",
  "enc-sender.jceks",
  "jceks",
  "storepass".toCharArray(),
  "bob",
  null,
  "CN=Bob, O=IBM, C=US",
  null);

// generate the security token used to the encryption
SecurityToken token = factory.newSecurityToken(X509Token.class,
     callbackHandler);

// generate WSSEncryption instance to encrypt the SOAP body content
WSSEncryption enc = factory.newWSSEncryption(token);
enc.addEncryptPart(WSSEncryption.BODY_CONTENT);

// set the data encryption method
// DEFAULT: WSSEncryption.AES128
enc.setEncryptionMethod(WSSEncryption.TRIPLE_DES);

// set the key encryption method
// DEFAULT: WSSEncryption.KW_RSA_OAEP
enc.setEncryptionMethod(WSSEncryption.KW_RSA15);

// add the WSSEncryption to the WSSGenerationContext
gencont.add(enc);

// generate the WS-Security header
gencont.process(msgcontext);
```

Adding encrypted parts using the WSSEncryptPart API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web Services Security APIs (WSS API). To configure encrypted parts for the request generator (client side) bindings, use the WSSEncryptPart API to define and add to the listing of elements in the encrypted part. WSSEncryptPart is an interface that is part of the com.ibm.websphere.wssecurity.wssapi.encryption package.

Before you begin

You can use the WSS APIs or configure policy sets using the administrative console to enable the encrypted parts. To secure SOAP messages, use the WSS APIs to complete the following encryption tasks, as needed:

- Configure encryption and choose the encryption methods using the WSSEncryption API.
- Configure the encrypted parts using the WSSEncryptpart API, as needed.

About this task

Confidentiality settings require that confidentiality constraints be applied to generated messages. These constraints include specifying which message parts within the generated message must be encrypted, and which message parts to attach encrypted elements to. The encryption information on the generator side is used for encrypting an outgoing SOAP message. The request generator is configured for the client.

The WSSEncryptPart API specifies information related to encrypted parts and sets the encrypted parts that have been added for message confidentiality protection. Use the WSSEncryptPart to set the transform method and to specify the part to which the transform method is to be applied. Sets the transform method only if using SOAP with Attachments. The WSSEncryptPart is usually not needed except, in some case for tasks such as setting the transform method.

The encrypted parts and related information displayed in the following table are used to protect the confidentiality of messages.

Table 15. Encrypted parts

Encrypted parts	Description
part	Adds the WSSEncryptPart object as a target of the encryption part.
keyword	Adds the encrypted parts using keywords. The default encryption parts that you can add using keywords are the BODY_CONTENT and SIGNATURE. WebSphere Application Server supports using these keywords: • BODY_CONTENT • SIGNATURE
xpath	Adds the encrypted part by using an XPath expression.
signature	Adds the WSSSignature component as a target of the encrypted part. WSSSignature is applicable only if the SOAP message contains a signature element.
header	Adds the SOAP header, specified by QName, as a target of the encrypted part.
securityToken	Adds the SecurityToken object as a target of the encrypted part.

For encrypted parts, certain default behaviors occur. The simplest way to use the WSSEncryptPart API is to use the default behavior. The WSSEncryptPart API provides defaults for specifying the transform algorithm, setting objects as targets, specifying the encrypted parts, such as: the SOAP body content and the signature.

The encryption default behaviors include:

Table 16. Encrypted part decisions

Encrypted part decisions	Default behavior
Which SOAP message parts to encrypt using keywords	Specifies which keywords to use for the encrypted parts. WebSphere Application Server sets the following SOAP message parts by default for encryption: • WSSEncryption.BODY_CONTENT • WSSEncryption.SIGNATURE
Which transform method to add	WebSphere Application Server does not specify any transform method by default. Specify a transform method only if using SOAP with Attachments.

1. To encrypt the SOAP message parts using the WSSEncryptPart API, first ensure that the application server is installed.
2. The WSS API process using WSSEncryptPart follows these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSGenerationContext instance from the WSSFactory instance.
   c. Creates the SecurityToken from WSSFactory to configure the encryption.
   d. Creates WSSEncryption from the WSSFactory instance using SecurityToken.
   e. Creates WSSEncryptPart from WSSFactory.
   f. Adds the parts to be encrypted and to be applied with the transform in WSSEncryptPart. WebSphere Application Server sets these encrypted parts by default for WSSEncryptPart: the BODY_CONTENT and SIGNATURE. After you add other encrypted parts, the default values are no longer valid. For example, if you call addEncryptPart(securityToken, false), only the security token is encrypted, and not the signature and body content. So if you want to encrypt the security token, the signature, and the body content, you must call addEncryptPart(securityToken, false),

addEncryptPart(WSSEncryption.SIGNATURE), and
addEncryptPart(WSSEncrypyion.BODY_CONTENT).

g. Sets the transform method.
h. Adds WSSEncryptPart to WSSEncryption.
i. Adds WSSEncryption to WSSGenerationContext.
j. Calls WSSGenerationContext.process() with the SOAPMessageContext.

Results

If there is an error condition during encryption of the message parts, a WSSException is provided. If successful, the API calls the WSSGenerationContext.process(), the WS-Security header is generated, and the SOAP message is now secured using Web services security.

What to do next

After enabling encrypted parts for the request generator (client side) binding, you must specify the same parts to be decrypted for the response consumer (client side) bindings. Next, to configure decryption and decrypted parts, use the WSS APIs or configure policy sets using the administrative console.

Configuring generator signing information to protect message integrity using the WSS APIs:

You can configure the signing information to protect message integrity for the request (client side) generator binding. Signing information includes the signature and the signed parts. To keep the integrity of the message, digital signatures are typically applied.

Before you begin

In addition to using a digital signature and configuring the signing information, the following tasks should also be performed:

- Verify the signing information.
- Incorporate encryption.
- Attach security tokens.

About this task

Integrity refers to digital signature while confidentiality refers to encryption. Integrity is provided by applying a digital signature to a SOAP message. To configure the signing information to protect message integrity, you must first digitally sign and then verify the signature for the SOAP messages. Integrity decreases the risk of data modification when you transmit data across a network.

Also, message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using the signature algorithm methods. The WSS APIs specify which algorithm is to be used to sign the certificate. The signature algorithms specify the Uniform Resource Identifiers (URI) of the signature method. WebSphere Application Server supports several pre-configured request signing algorithm methods.

You can use the following interfaces to configure Web services security and to protect SOAP message integrity:

- Use the administrative console to configure policy sets for the signing information.
- Use the Web Services Security APIs (WSS API) to configure the SOAP message context (only for the client).

Perform the following signing tasks, using the WSS APIs, to configure the signing information and to protect message integrity for the generator binding.

- Configure the signing information using the WSSSignature API. Configure the signing information for the generator binding using the WSSSignature API. Signing information is used to sign parts of a message including the SOAP body, the time stamp, and the WS-Addressing headers. Both signing and encryption can be applied to the same message parts, such as the SOAP body.
- Add or change signed parts using the WSSSignPart API.
- Configure the client for request signing methods using the WSSSignature or WSSSignPart APIs. To configure the client for request signing, choose the signing methods. The request signing methods include the signature, the canonicalization, the digest, and the transform methods. Use the WSSSignature API to configure the signature and canonicalization methods. Use the WSSSignPart API to configure the digest and transform methods.

Results

The WSS APIs also specify the security token for the generator (client) binding and set the type of token reference to protect message authenticity. By completing the steps in these tasks, you have configured generator signing to protect the integrity of the SOAP message.

What to do next

Next, verify the consumer signing information by using the WSS APIs or by configuring policy sets using the administrative console.

Configuring the signing information using the WSS APIs:

You can configure the signing information for the client-side request generator (sender) bindings. Signing information is used to sign and validate parts of a message including the SOAP body, the timestamp information, and the Username token. To configure the client for request signing, specify which message parts to digitally sign when configuring the client.

Before you begin

WebSphere Application Server uses XML digital signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method. Prior to completing these steps, familiarize yourself with XML digital signature for signing and verifying digital signatures for digital content.

About this task

By including XML signature in SOAP messages, the following issues are realized: message integrity and authentication. *Integrity* refers to digital signature whereas confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet. WebSphere Application Server uses the signing information for the default generator to sign parts of the message, such as the body, time stamp, and Username token.

For the signing information, you must specify the following:

- Which parts of the message are to be signed.
- The key information that is referenced by the key information for the signing keys.
- The signing algorithms.

WebSphere Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

The WSSSignature API configures the following parts as signature parts:

Table 17. Pre-configured signature parts

Security token object	This object authenticates the client. If this option is specified, then the message is signed. You can digitally sign the message using a security token if a login configuration authentication method is selected.
WSSTimestamp object	This object adds a time stamp to a message. The time stamp determines if the message is valid based on the time that the message is sent and then received.
WSSSignature Part object	This object adds the signature parts to a message.
SOAP header and the QName as a target	This signature part adds the header, specified by QName, as a verification part.

The WSS APIs allow the use of keywords or an XPath expression to specify which parts of the message are to be signed. WebSphere Application Server supports the use of the following keywords:

Table 18. Supported signature keywords

Keyword	**References**
ADDRESSING_HEADERS	The Web Services Addressing (WS-Addressing) headers.
BODY	The SOAP message body. The body is the user data portion of the message.
TIMESTAMP	The creation and expiration timestamp information.

The Web Services Security API (WSS API) are used to configure the signing information for the request generator (client side) section of the bindings file. To configure the signing information on the client side, use the WSS APIs or configure policy sets for signing using the administrative console.

If configuring using the WSS APIs, the WSSSignature and WSSSignPart APIs complete the following steps to specify which message parts to digitally sign when configuring the client for request generator signing:

1. The WSSSignature API adds the required parts of the SOAP message to digitally sign. Either a keyword or an XPath expression can be used to specify the required encryption parts.
2. The WSSSignature API sets the signature method algorithm. The default signature method is RSA_SHA1. WebSphere Application Server supports the following pre-configured algorithms:
   - RSA SHA1: http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - HMAC SHA1http://www.w3.org/2000/09/xmldsig#hmac-sha1

   WebSphere Application Server does not support the following algorithm for DSA-SHA1: http://www.w3.org/2000/09/xmldsig#dsa-sha1. You cannot use the DSA-SHA1 algorithm if you want to be compliant with the Basic Security Profile (BSP).

   Any ds:SignatureMethod/@Algorithm element in a signature is based on a symmetric key and must have a value of RSA-SHA1 or HMAC-SHA1.

   The algorithm that is specified for the request generator configuration must match the algorithm that is specified for the request consumer configuration.
3. The WSSSignature API sets the canonicalization method. The default signature method is EXC_C14N. WebSphere Application Server supports the following pre-configured algorithms:
   - The URI of the exclusive canonicalization algorithm, EXC_C14N: http://www.w3.org/2001/10/xml-exc-c14n#.
   - The URI of the inclusive canonicalization algorithm, C14N: http://www.w3.org/2001/10/xml-c14n#.

   The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer.
4. The WSSSignature API adds a security token. The API adds information about the security token that is to be used for the signature, such as:

- The class for security token.
- The callback handler
- The name of the JAAS login configuration.

5. The WSSSignature API sets the type of security token and sets the type of token reference. WebSphere Application Server supports the following pre-configured token references:
   - SecurityToken.REF_STR

     Represents the security token reference as a token reference type.
   - SecurityToken.REF_KEYID

     Represents the key identifier reference as a token reference type.
   - SecurityToken.REF_EMBEDDED

     Represents the embedded reference as a token reference type.
   - SecurityToken.REF_THUMBPRINT

     Represents the thumbprint reference as a token reference type.
6. If SecurityToken.REF_KEYID is set as the type of token reference, the WSSSignature API sets the key information signature type and configures the key information that is referenced by the key information references. WebSphere Application Server supports the following:
   - Specifying that the KeyInfo element is not signed.
   - Specifying that the entire <KeyInfo> element is signed.
   - Specifying that the child elements <Keyinfochildelements> of the <KeyInfo> element are signed.

   If you do not specify one of the previous signature types, WebSphere Application Server specifies that the entire <KeyInfo> element is signed, by default.

   If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.

   The key information signature type for the generator must match the signature type for the consumer.
7. The WSSSignature API specifies whether to require signature confirmation. The OASIS Web Services Security (WS-Security) Version 1.1 specification defines the use of signature confirmation. If you are using WS-Security Version 1.0, this function is not available.

   The signature confirmation value is stored in order to validate the signature confirmation with it after the receiving message is returned. This method is called if the response message is expected to attach the signature confirmation into the SOAP message.
8. The WSSSignPart API specifies the part reference. The part reference specifies which parts of the message to digitally sign.

   The part reference refers to the message part that is digitally signed. The part attribute refers to the name of the <Integrity> element when the <PartReference> element is specified for the signature. You can specify multiple <PartReference> elements within the <SigningInfo> element. The <PartReference> element has two child elements when it is specified for the signature verification: <DigestTransform> and <Transform>.
9. The WSSSignPart API specifies the digest method algorithm. The digest method algorithm specified within the <DigestMethod> element is used in the <SigningInfo> element.

   WebSphere Application Server supports the following pre-configured digest algorithms:
   - http://www.w3.org/2000/09/xmldsig#sha1
   - http://www.w3.org/2001/04/xmlenc#sha256
   - http://www.w3.org/2001/04/xmlenc#sha512
10. The WSSSignPart API specifies the transform algorithm. The transform algorithm is that is specified within the <Transform> element and specifies the transform algorithm for the signature. WebSphere Application Server supports the following pre-configured transform algorithms:
    - http://www.w3.org/2001/10/xml-exc-c14n#

- http://www.w3.org/TR/1999/REC-xpath-19991116

  Do not use this transform algorithm if you want to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.
- http://www.w3.org/2002/06/xmldsig-filter2
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
- http://www.w3.org/2002/07/decrypt#XML
- http://www.w3.org/2000/09/xmldsig#enveloped-signature

The transform algorithm that you select for the generator must match the transform algorithm that you select for the consumer.

Note: If both of the following conditions are true, WebSphere Application Server signs the referenced token:
- You previously selected the Keyinfo or the Keyinfochildelements option
- You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

11. If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. Configure policy sets using the administrative console to configure the same actor strings for the Web service on the server, which processes the request and sends the response back.

    The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

Results

After the WSSSignature and WSSSignPart APIs complete these steps, the signing information is configured for the generator sections of the bindings files.

Example

The following example shows WSS API sample code to configure the signature, to generate the callback handler, and to specify the X.509 token type as the security token:

```
WSSFactory factory = WSSFactory.getInstance();
// Instantiate a generation context
WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Generate the callback handler and specify the X.509 token
   X509GenerateCallbackHandler callbackhandler = generateCallbackHandler();
   SecurityToken token = factory.newSecurityToken(X509Token.class,
                                                  callbackHandler);

// Set the signature information
WSSSignature sig = factory.newWSSSignature(token);
// Add the header using QName
   sig.addSignHeader(new QName("http://www.w3.org/2005/08/addressing", "To"));
   sig.addSignHeader(new QName("http://www.w3.org/2005/08/addressing", "MessageID"));
   sig.addSignHeader(new QName("http://www.w3.org/2005/08/addressing", "Action"));
// Apply the signature
```

```
   gencont.add(sig);

// Secure the message
gencont.process(msgctx);
```

What to do next

You must configure similar signature information for the client-side request consumer (receiver) bindings by completing the following verification tasks:

- Verify the signature
- Choose the signature algorithm methods.
- Change or add signed parts, as needed.

If signature verification is already configured, configure the encryption and decryption information, or configure the consumer and generator tokens.

Configuring the signature information using the WSSSignature API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web services security APIs (WSS API). To configure the signature information for the generator binding sections for the client-side request, use the WSSSignature API. The WSSSignature API is part of the com.ibm.websphere.wssecurity.wssapi.signature package.

Before you begin

Either you can use the WSS API or you can configure the policy sets by using the administrative console to enable the signing information. To secure SOAP messages, you must complete the following signature tasks:

- Configure the signature information.
- Choose the signature methods.
- Add or change signed parts, as needed.

About this task

WebSphere Application Server uses the signing information for the default generator to sign parts of the message, and uses XML digital signature with existing algorithms such as RSA-SHA1 and HMAC-SHA1.

XML signature defines many methods for describing key information and enables the definition of a new method. XML canonicalization (C14N) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. The C14N process is used to canonicalize XML information. Select an appropriate C14N algorithm because the information that is canonicalized depends on this algorithm.

The signing information specifies the integrity constraints that are applied to generated messages. The constraints include specifying which message parts within the generated message must be digitally signed, and the message parts to attach digitally signed Nonce and timestamp elements to. The following signature and related signature part information are configured:

Table 19. signature parts information

signature parts	Description
keyword	Adds a signature part using keywords. Use the following keywords for the signature parts: • ADDRESSING_HEADERS • BODY • TIMESTAMP The WS-Addressing headers are not encrypted but can be signed.
xpath	Adds a signature part by using an XPath expression.
part	Adds a WSSSignPart object as a target of the signature part.
timestamp	Adds a WSSTimestamp object as a target of the signature part. When specified, the timestamp information also specifies when the message is generated and when it expires.
header	Adds the header, specified by QName, as a target of the signature part.
securityToken	Adds a SecurityToken object as a target of the signature part.

For signing information, certain default behaviors occur. The simplest way to use the WSSSignature API is to use the default behavior (see the example code). The default values are defined by the WSS API for the signing method, the canonicalization method, the security token references, and the signature parts.

Table 20. Signature default behaviors

Signature decisions	Default behavior
Which keywords to use	Sets the keywords. WebSphere Application Server supports the following keywords by default: • ADDRESSING_HEADERS • BODY • TIMESTAMP
Which signature method to use	Sets the signature algorithm. The default signature method is RSA SHA1. WebSphere Application Server supports the following pre-configured signature methods: • WSSSignature.RSA_SHA1: http://www.w3.org/2000/09/xmldsig#rsa-sha1 • WSSSignature.HMAC_SHA1: http://www.w3.org/2000/09/xmldsig#hmac-sha1 The DSA-SHA1 digital signature method (http://www.w3.org/2000/09/xmldsig#dsa-sha1) is not supported.
Which canonicalization method to use	Sets the canonicalization algorithm. The default canonicalization method is EXC C14N. WebSphere Application Server supports the following pre-configured canonicalization methods: • WSSSignature.EXC_C14N; http://www.w3.org/2001/10/xml-exc-c14n# • WSSSignature.C14N: http://www.w3.org/2001/10/xml-c14n#
Whether signature confirmation is required	Sets whether to require signature confirmation. The default value is **false**. Signature confirmation is defined in the OASIS Web Services Security Version 1.1 specification. If required, the value of your signature confirmation is stored in order to use it to validate the signature confirmation after receiving back the message that generated the signature confirmation in the response message. This method is for the requestor side.

Table 20. Signature default behaviors (continued)

Signature decisions	Default behavior
Which security token to use	Sets the SecurityToken. The token type specifies which type of token to use for signing and validating messages. The X.509 token is the default token type. WebSphere Application Server provides the following pre-configured consumer token types: • Derived Key Token • X509 tokens You can also create custom token types, as needed.
Which token reference to set	Sets the refType. SecurityToken.REF_STR is the default value for the type of token reference. WebSphere Application Server supports these pre-configured token references types: • SecurityToken.REF_STR • SecurityToken.REF_KEYID • SecurityToken.REF_EMBEDDED • SecurityToken.REF_THUMBPRINT

If WSSSignature.requireSignatureConfirmation() is called, then the WSSSignature API expects that the response message will include the signature confirmation.

1. To configure the signing information in a SOAP message by using the WSS API, first ensure that the application server is installed.
2. Use the WSSSignature API to sign the message parts and specify the algorithms in a SOAP message. The WSS API process for signature follows these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSGenerationContext instance from the WSSFactory instance. WSSGenerationContext must be called in a JAX-WS client application.
   c. Creates the SecurityToken from WSSFactory to configure the key for signing.
   d. Creates WSSSignature from the WSSFactory instance using the SecurityToken. The default behavior of WSSSignature is to sign these signature parts: BODY, ADDRESSING_HEADERS, and TIMESTAMP.
   e. Adds the part to be signed, if the default part is not appropriate. If the digest method or transform method is changed, creates WSSSignPart and add it to WSSSignature.
   f. Creates WSSSignaturePart to WSSSignature. Calls the requiredSignatureConfirmation() method, if the signature confirmation is to be applied.
   g. Sets the canonicalization method, if the default is not appropriate.
   h. Sets the signature method, if the default is not appropriate.
   i. Sets the token reference, if the default is not appropriate.
   j. Adds WSSSignature to WSSGenerationContext.
   k. Calls WSSGenerationContext.process() with the SOAPMessageContext.

Results

You have completed the steps to configure the signature for the generator section of the bindings. If there is an error condition when signing the message parts, a WSSException is provided. If successful, the WSSGenerationContext.process() is called, and Web Services Security is applied to the SOAP message.

Example

The following example provides sample code that uses methods that are defined in the WSSignature API.

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance  (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSGenerationContext instance  (step: b)
   WSSGenerationContext gencont = factory.newWSSGenerationContext();

// Generate the callback handler
   X509GenerateCallbackHandler callbackHandler = new
       X509GenerateCallbackHandler(
       "",
       "dsig-sender.ks",
       "jks",
       "client".toCharArray(),
       "soaprequester",
       "client".toCharArray(),
       "CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP", null);

// Generate the security token to be used for the signature  (step: c)
   SecurityToken token = factory.newSecurityToken(X509Token.class,
        callbackHandler);

// Generate the WSSSignature instance (step: d)
   WSSSignature sig = factory.newWSSSignature(token);

// Set the part to be signed  (step: e)
// DEFAULT: WSSSignature.BODY, WSSSignature.ADDRESSING_HEADERS,
//          and WSSSignature.TIMESTAMP.

// Set the part in the SOAP Header specified by QName  (step: e)
      sig.addSignHeader(new
                        QName("http://www.w3.org/2005/08/addressing",
                        "MessageID"));

// Set the part specified by the keyword  (step: e)
      sig.addSignPart(WSSSignature.BODY);

// Set the part specified by SecurityToken  (step: e)
   UNTGenerateCallbackHandler untCallbackHandler = new
      UNTGenerateCallbackHandler("Chris", "sirhC");
      SecurityToken unt = factory.newSecurityToken(UsernameToken.class,
         untCallbackHandler);
      sig.addSignPart(unt);

// Set the part specified by WSSSignPart  (step: e)
   WSSSignPart sigPart = factory.newWSSSignPart();
      sigPart.setSignPart(WSSSignature.TIMESTAMP);
      sigPart.setDigestMethod(WSSSignPart.SHA256);
      sig.addSignPart(sigPart);

// Set the part specified by WSSTimestamp  (step: e)
   WSSTimestamp timestamp = factory.newWSSTimestamp();
      sig.addSignPart(timestamp);

// Set the part specified by XPath expression  (step: e)
   StringBuffer sb = new StringBuffer();
      sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         and local-name()='Envelope']");
      sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         and local-name()='Body']");
      sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
         and local-name()='Ping']");
      sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
         and local-name()='Text']");
      sig.addSignPartByXPath(sb.toString());
```

```
// Set to apply the signature confirmation  (step: f)
      sig.requireSignatureConfirmation();

// Set the canonicalization method  (step: g)
// DEFAULT: WSSSignature.EXC_C14N
      sig.setCanonicalizationMethod(WSSSignature.C14N);

// Set the signature method  (step: h)
// DEFAULT: WSSSignature.RSA_SHA1
      sig.setSignatureMethod(WSSSignature.HMAC_SHA1);

// Set the token reference  (step: i)
// DEFAULT: SecurityToken.REF_STR
      sig.setTokenReference(SecurityToken.REF_KEYID);

// Add the WSSSignature to WSSGenerationContext  (step: j)
   gencont.add(sig);

// Generate the WS-Security header  (step: k)
gencont.process(msgctx);
```

Note: The X509GenerationCallbackHandler needs the key password because the private key is used for signing.

What to do next

Next, chose the algorithm methods if you want a method that is different from the default values. If the algorithm methods do not need to be changed, next use the WSSVerification API to verify the signature and specify the algorithm methods in the consumer section of the binding. Note that the WSSVerification API is only supported on the response consumer (client side).

Adding signed parts using the WSSSignPart API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web Services Security APIs (WSS API). To configure parts to be signed for the request generator (client side) bindings, use the WSSSignPart API to protect the integrity of messages and to configure the digest and transform algorithm methods. The WSSSignPart API is part of the com.ibm.websphere.wssecurity.wssapi.signature package.

Before you begin

Either you can use the WSS API or you can configure the policy sets by using the administrative console to configure the signing information. To secure SOAP messages using the signing information, you must complete one of the following tasks:

- Configure the signature information
- Configure signed parts, as needed.

About this task

WebSphere Application Server uses the signing information for the default generator to sign parts of the message, and uses XML digital signature with existing digest and transform algorithms (for example, SHA1 or TRANSFORM_EXC_C14N).

The signing information specifies the integrity constraints that are applied to generated messages. The signed parts are used to protect the integrity of messages. You can specify the signed parts to add for message integrity protection.

The following table shows the required signed parts when the digital signature security constraint (integrity) is defined:

Table 21. Signed parts information

Signed parts	Description
keyword	Adds signed parts using keywords. WebSphere Application Server supports the following keywords for signed parts: • BODY • ADDRESSING_HEADERS • TIMESTAMP The WS-Addressing headers are not encrypted but can be signed.
xpath	Adds the required signed parts by using an XPath expression.
header	Adds the header, specified by QName, as a signed part.
timestamp	Adds a WSSTimestamp object as a signed part. If specified, the timestamp information specifies when the message is generated and when it expires.

Different message parts can be specified in the message protection for request on the generator side. WSSSignPart allows for adding a transform algorithm, setting a digest method, setting objects as targets, specifying whether an element, and the signed parts, such as: the SOAP body, the WS-Addressing header, and timestamp information.

For signing information, certain default behaviors occur. The simplest way to use the WSSSignPart API is to use the default behavior (see the example code). The signed parts default behaviors include:

Signature decisions	Default behavior
Which SOAP message parts to sign	WebSphere Application Server supports the following SOAP message parts to be signed and used for message protection: • WSSSignature.BODY • WSSSignature.ADDRESSING_HEADERS • WSSSignature.TIMESTAMP
Which digest method to use	Sets the digest algorithm method. The digest method algorithm that is specified within the <DigestMethod> element is used in the <SigningInfo> element. WebSphere Application Server supports the following pre-configured digest methods: • WSSSignPart.SHA1 (the default value): http://www.w3.org/2000/09/xmldsig#sha1 • WSSSignPart.SHA256: http://www.w3.org/2001/04/xmlenc#sha256 • WSSSignPart.SHA512: http://www.w3.org/2001/04/xmlenc#sha512

Signature decisions	Default behavior
Which transform algorithms to use	Adds the transform method. The transform algorithm is specified within the <Transform> element and specifies the transform algorithm for the signature. WebSphere Application Server supports the following pre-configured transform algorithms: • WSSSignPart.TRANSFORM_EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n# • WSSSignPart.TRANSFORM_XPATH2_FILTER: http://www.w3.org/2002/06/xmldsig-filter2 Use this transform method to ensure compliance with the Basic Security Profile (BSP). • WSSSignPart.TRANSFORM_STRT10: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform • WSSSignPart.TRANSFORM_ENVELOPED_SIGNATURE: http://www.w3.org/2000/09/xmldsig#enveloped-signature

1. To enable Web Service Security by using the WSS API (WSSSignPart), first ensure that the application server is installed.
2. Use the WSSSignPart API to sign the message parts and specify the algorithms in a SOAP message. The WSS API process for signed parts follows these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSGenerationContext instance from the WSSFactory instance.
   c. Creates the SecurityToken from WSSFactory to configure the key for signing.
   d. Creates WSSSignature from the WSSFactory instance using the SecurityToken.
   e. Creates WSSSignPart from the WSSFactory instance.
   f. Sets the part to be signed and the digest method or transform method specified by step g or step h if the default is not appropriate.
   g. Sets the digest method if the default is not appropriate.
   h. Sets the transform method if the default is not appropriate.
   i. Adds WSSSignPart to WSSSignature. After any WSSSignPart is set to WSSSignature, the default parts to be signed, which are specified in WSSSignature, are ignored.
   j. Adds WSSSignature to WSSGenerationContext.
   k. Calls WSSGenerationContext.process() with the SOAPMessageContext.

Results

You have completed the steps to configure the signed parts for the generator section of the bindings files. If there is an error condition, a WSSException is provided. If successful, the WSSGenerationContext.process() is called, and Web services security is applied to the SOAP message.

Example

The following example provides sample code that uses all of methods that are defined in the WSSSignPart API:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance  (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate WSSGenerationContext instance  (step: b)
   WSSGenerationContext gencont = factory.newWSSGenerationContext();
```

```
// Generate callback handler
   X509GenerateCallbackHandler callbackHandler = new
      X509GenerateCallbackHandler
      "",
      "dsig-sender.ks",
      "jks",
      "client".toCharArray(),
      "soaprequester",
      "client".toCharArray(),
      "CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP", null);

// Generate the security token used to the signature  (step: c)
   SecurityToken token = factory.newSecurityToken(X509Token.class, callbackHandler);

// Generate WSSSignature instance (step: d)
   WSSSignature sig = factory.newWSSSignature(token);

// Set the part specified by WSSSignPart  (step: e)
   WSSSignPart sigPart = factory.newWSSSignPart();

// Set the part specified by WSSSignPart  (step: f)
   sigPart.setSignPart(WSSSignature.BODY);

// Set the digest method specified by WSSSignPart  (step: g)
   sigPart.setDigestMethod(WSSSignPart.SHA256);

// Set the transform method specified by WSSSignPart  (step: h)
   sigPart.setTransformMethod(WSSSignPart.TRANSFORM_STRT10);

// Add the part specified by WSSSignPart  (step: i)
   sig.addSignPart(sigPart);

// Add the WSSSignature to the WSSGenerationContext  (step: j)
   gencont.add(sig);

// Generate the WS-Security header  (step: k)
gencont.process(msgcontext);
```

Note: The X509GenerationCallbackHandler needs the key password because the private key is used for signing.

What to do next

Use the WSSVerifyPart API or configure policy sets using the administrative console to verify the signed parts on the consumer side.

Configuring the client for request signing methods:

Use the WSSSignature and WSSSignPart APIs to choose the signing methods. The request signing methods include the signature, canonicalization, digest, and transform methods.

Before you begin

First, you must have specified which parts of the message sent by the client must be digitally signed using the WSS APIs or configuring policy sets using the administrative console.

About this task

The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web site http://www.w3.org/TR/xmldsig-core.

Table 22. Signing methods

Name of method	Description
Canonicalization algorithm	Canonicalizes the <SignedInfo> element before the information is digested as part of the signature operation.
Signature algorithm	Calculates the signature value of the canonicalized <SignedInfo> element. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.
Transform method	Transforms the parts to be signed before the information is digested as part of the signature operation.
Digest method	Calculates the digest value of the transformed parts. The algorithm selected for the client request sender configuration must match the algorithms selected in the server request receiver configuration.

You can use the WSS APIs or configure policy sets using the administrative console to configure the signing algorithm methods. If using the WSS APIs, use the WSSSignature and WSSSignPart APIs to specify which message parts to digitally sign when configuring the client for request signing.

The WSSSignature and WSSSignPart APIs complete the following steps to configure the signature and signed part algorithm methods:

1. For the generator binding, the WSSSignature API specifies the signature method. WebSphere Application Server supports the following pre-configured signature methods:
   - WSSSignature.RSA_SHA1 (the default value): http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - WSSSignature.HMAC_SHA1: http://www.w3.org/2000/09/xmldsig#hmac-sha1

   For the WSS APIs, WebSphere Application Server does not support the DSA-SHA1 digital signature method, http://www.w3.org/2000/09/xmldsig#dsa-sha1.
2. For the generator binding, the WSSSignature API specifies the canonicalization method. WebSphere Application Server supports the following pre-configured canonicalization algorithms:
   - WSSSignature.EXC_C14N (the default value): The exclusive canonicalization algorithm, http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSSignature.C14N: The inclusive canonicalization algorithm, http://www.w3.org/2001/10/xml-c14n#
3. For the generator binding, the WSSSignPart API specifies the digest method. WebSphere Application Server supports the following pre-configured digest methods:
   - WSSSignPart.SHA1 (the default value): http://www.w3.org/2000/09/xmldsig#sha1
   - WSSSignPart.SHA256: http://www.w3.org/2001/04/xmlenc#sha256
   - WSSSignPart.SHA512: http://www.w3.org/2001/04/xmlenc#sha512
4. For the generator binding, the WSSSignPart API specifies the transform method. WebSphere Application Server supports the following pre-configured transform algorithms:
   - WSSSignPart.TRANSFORM_EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSSignPart.TRANSFORM_XPATH2_FILTER: http://www.w3.org/2002/06/xmldsig-filter2
   - WSSSignPart.TRANSFORM_STRT10: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - WSSSignPart.TRANSFORM_ENVELOPED_SIGNATURE: http://www.w3.org/2000/09/xmldsig#enveloped-signature

   For the WSS APIs, WebSphere Application Server does not support the following transform algorithms:
   - http://www.w3.org/TR/1999/REC-xpath-19991116
   - http://www.w3.org/2002/07/decrypt#XML

Results

Using the WSS APIs, you have specified which algorithm methods are used to digitally sign a message when the client sends a message to a server.

Example

The following example is sample code for specifying the signature information, HMAC_SHA1 as signature method, C14N as a canonicalizaion method, SHA256 as a digest method, and EXC_C14N and TRANSFORM_STRT10 as the transform methods:

```
//get the message context
Object msgcontext = getMessageContext();

//generate WSSFactory instance
WSSFactory factory = WSSFactory.getInstance();

//generate WSSGenerationContext instance
WSSGenerationContext gencont = factory.newWSSGenerationContext();

//generate callback handler
X509GenerateCallbackHandler callbackHandler = new X509GenerateCallbackHandler(
  "",
  "dsig-sender.ks",
  "jks",
  "client".toCharArray(),
  "soaprequester",
  "client".toCharArray(),
  "CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP",
  null);

//generate the security token used to the signature
SecurityToken token = factory.newSecurityToken(X509Token.class, callbackHandler);

//generate WSSSignature instance
WSSSignature sig = factory.newWSSSignature(token);

//set the canonicalization method
// DEFAULT: WSSSignature.EXC_C14N
sig.setCanonicalizationMethod(WSSSignature.C14N);

//set the signature method
// DEFAULT: WSSSignature.RSA_SHA1
sig.setSignatureMethod(WSSSignature.HMAC_SHA1);

//set the part specified by WSSSignPart
WSSSignPart sigPart = factory.newWSSSignPart();

//set the digest method
 // DEFAULT: WSSSignPart.SHA1
sigPart.setDigestMethod(WSSSignPart.SHA256);

//add the transform method
// DEFAULT: WSSSignPart.TRANSFORM_EXC_C14N
sigPart.addTransformMethod(WSSSignPart.TRANSFORM_EXC_C14N);
sigPart.addTransformMethod(WSSSignPart.TRANSFORM_STRT10);

 // add the WSSSignPart to the WSSSignature
 sig.addSignPart(sigPart);

//add the WSSSignature to the WSSGenerationContext
gencont.add(sig);

//generate the WS-Security header
gencont.process(msgcontext);
```

What to do next

After you configure the client to digitally sign the message and to choose the algorithm methods, you must configure the server to verify the digital signature for request signing and to choose the algorithm methods.

Configure policy sets using the administrative console to configure the signature verification information and methods on the server.

Digital signing methods using the WSSSignature API:

You can configure the signing information for the generator binding using the WSS API. To configure the client for request signing, choose the digital signing methods. The algorithm methods include the signing and canonicalization methods.

You must configure generator signing information to protect message integrity by digitally signing SOAP messages. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network.

After you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message.

Methods

Methods that are used for the signing information include the:

Signature method
Sets the signature algorithm method.

Canonicalization method
Sets the canonicalization algorithm method.

Signature algorithms

The signature algorithms specify the algorithm that is used to sign the certificate. The signature algorithms specify the Uniform Resource Identifiers (URI) of the signature method. WebSphere Application Server supports the following pre-configured algorithms:

Table 23. Signature algorithms

Algorithm	Description
WSSSignature.HMAC_SHA1	A URI of the signature algorithm, HMAC: http://www.w3.org/2000/09/xmldsig#hmac-sha1
WSSSignature.RSA_SHA1 (the default value)	A URI of the signature algorithm, RSA: http://www.w3.org/2000/09/xmldsig#rsa-sha1

For the WSS APIs, WebSphere Application Server does not support the DSA-SHA1 algorithm, http://www.w3.org/2000/09/xmldsig#dsa-sha1

The signing algorithm that is specified for the request generator configuration must match the algorithm that is specified for the request consumer configuration.

Canonicalization algorithms

The canonicalization algorithms specify the Uniform Resource Identifiers (URI) of the canonicalization method. WebSphere Application Server supports the following pre-configured algorithms:

Table 24. Signature canonicalization algorithms

Algorithm	Description
WSSSignature.EXC_C14N (the default value)	A URI of the exclusive canonicalization algorithm EXC_C14N: http://www.w3.org/2001/10/xml-exc-c14n#
WSSSignature.C14N	A URI of the inclusive canonicalization algorithm, C14N: http://www.w3.org/2001/10/xml-c14n#

The canonicalization algorithm that is specified for the request generator configuration must match the algorithm that is specified for the request consumer configuration.

The following example provides sample WSS API code that specifies the HMAC_SHA1 as a signature method and C14n as a canonicalization method:

```
//generate WSSFactory instance
WSSFactory factory = WSSFactory.getInstance();

//generate WSSGenerationContext instance
WSSGenerationContext gencont = factory.newWSSGenerationContext();

//generate callback handler
X509GenerateCallbackHandler callbackHandler = new
     X509GenerateCallbackHandler(
  "",
  "dsig-sender.ks",
  "jks",
  "client".toCharArray(),
  "soaprequester",
  "client".toCharArray(),
  "CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP",
  null);

//generate the security token used to the signature
SecurityToken token = factory.newSecurityToken(X509Token.class,
     callbackHandler);

//generate WSSSignature instance
WSSSignature sig = factory.newWSSSignature(token);

//set the canonicalization method
// DEFAULT: WSSSignature.EXC_C14N
sig.setCanonicalizationMethod(WSSSignature.C14N);

//set the signature method
// DEFAULT: WSSSignature.RSA_SHA1
sig.setSignatureMethod(WSSSignature.HMAC_SHA1);

//add the WSSSignature to the WSSGenerationContext
gencont.add(sig);

//generate the WS-Security header
gencont.process(msgcontext);
```

Signed parts methods using the WSSSignPart API:

You can configure the signed parts information for the generator binding using the WSS API. The algorithms include the digest and transform methods.

You can protect message integrity by configuring signed parts and key information. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network.

Methods

Methods that are used for the signed parts include the:

Digest method
Sets the digest algorithm method.

Transform algorithm
Sets the transform algorithm method.

Digest algorithms

The digest method algorithm specified within the element is used in the element. WebSphere Application Server supports the following pre-configured algorithms:

Table 25. Signed parts digest methods

Digest method	Description
WSSSignPart.SHA1 (the default value)	A URI of the digest algorithm, SHA1: http://www.w3.org/2000/09/xmldsig#sha1
WSSSignPart.SHA256	A URI of the digest algorithm, SHA256: http://www.w3.org/2001/04/xmlenc#sha256
WSSSignPart.SHA512	A URI of the digest algorithm, SHA256: http://www.w3.org/2001/04/xmlenc#sha512

Transform algorithms

The transform method algorithm specified within the element is used in the element. WebSphere Application Server supports the following pre-configured algorithms:

Table 26. Signed parts transform methods

Digest method	Description
WSSSignPart.TRANSFORM_ENVELOPED_SIGNATURE	A URI of the transform algorithm, enveloped signature: http://www.w3.org/2000/09/xmldsig#enveloped-signature
WSSSignPart.TRANSFORM_STRT10	A URI of the transform algorithm, STR-Transform: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
WSSSignPart.TRANSFORM_EXC_C14N (the default value)	A URI of the transform algorithm, Exc-C14N: http://www.w3.org/2001/10/xml-exc-c14n#
WSSSignPart.TRANSFORM_XPATH2_FILTER	A URI of the transform algorithm, XPath2 filter: http://www.w3.org/2002/06/xmldsig-filter2

The transform algorithm is specified within the <Transform> element and specifies the transform algorithm for the signed part.

For the WSS APIs, WebSphere Application Server does not support the following transform algorithms:

- http://www.w3.org/TR/1999/REC-xpath-19991116
- http://www.w3.org/2002/07/decrypt#XML

The following example provides sample WSS API code for specifying the signature and signed parts, setting the signing key and adding the STR-Transform transform algorithm as signed parts:

```
//get the message context
Object msgcontext = getMessageContext();

//generate WSSFactory instance
```

```
WSSFactory factory = WSSFactory.getInstance();

//generate WSSGenerationContext instance
WSSGenerationContext gencont = factory.newWSSGenerationContext();

//generate callback handler
X509GenerateCallbackHandler callbackHandler = new
      X509GenerateCallbackHandler(
  "",
  "dsig-sender.ks",
  "jks",
  "client".toCharArray(),
  "soaprequester",
  "client".toCharArray(),
  "CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP",
  null);

//generate the security token used to the signature
SecurityToken token = factory.newSecurityToken(X509Token.class,
     callbackHandler);

//generate WSSSignature instance
WSSSignature sig = factory.newWSSSignature(token);

//set the part specified by WSSSignPart
WSSSignPart sigPart = factory.newWSSSignPart();

//set the part specified by WSSSignPart
sigPart.setSignPart(WSSSignature.BODY);

//set the digest method specified by WSSSignPart
sigPart.setDigestMethod(WSSSignPart.SHA256);

//set the transform method specified by WSSSignPart
sigPart.addTransform(WSSSignPart.TRANSFORM_STRT10);

//set the part specified by WSSSignPart
sig.addSignPart(sigPart);

//add the WSSSignature to the WSSGenerationContext
gencont.add(sig);

//generate the WS-Security header
gencont.process(msgcontext);
```

Attaching the generator token using WSS APIs to protect message authenticity:

When you specify the token generator, the information is used on the generator side to generate the security token.

Before you begin

The token processing and pluggable token architecture in the Web Service Security run time reuses the same security token interface and Java Authentication and Authorization Service (JAAS) Login Module from the Web Services Security APIs (WSS API). The same implementation of token creation and validation can be used in both the WSS API and the WSS SPI in the Web Service Security run time.

Note: The com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface is not used with JAX-WS Web services. If you are using JAX-RPC Web services, this interface is still valid.

Note that the key name (KeyName) element is not supported in the application server because there is no KeyName policy assertion defined in the current OASIS Web Services Security draft specification. For similar reasons, a SAML token is not supported out of the box.

About this task

The JAAS callback handler (CallbackHandler) and the JAAS login module (LoginModule) are responsible for creating the security token on the generator side and validating (authenticating) the security token on the consumer side.

For example, on the generator side, the Username token is created by the JAAS LoginModule and using the JAAS CallbackHandler to pass the authentication data. The JAAS LoginModule creates the Username SecurityToken object and passes it to the Web services security run time.

Then, on the consumer side, the Username Token XML format is passed to the JAAS LoginModule for validation or authentication and the JAAS CallbackHandler is used to pass authentication data from the Web services security run time to the LoginModule. After the token is authenticated, a Username SecurityToken object is created and passed it to the Web Service Security run time.

Note: WebSphere Application Server does not support a stackable login module with the WebSphere Application Server default login module implementation, meaning adding the login module before or after the WebSphere Application Server login module implementation. If you want to stack the login module implementations, you must develop the required login modules because there is no default implementation.

The com.ibm.websphere.wssecurity.wssapi.token package provided by WebSphere Application Server includes support for these classes:

- Security token (SecurityTokenImpl)
- Binary security token (BinarySecurityTokenImpl)

In addition, WebSphere Application Server provides the following pre-configured sub-interfaces for security tokens:

- Derived key token
- Security context token (SCT)
- Username token
- LTPA token propagation
- LTPA token
- X509PKCS7 token
- X509PKIPath token
- X509v3 token
- Kerberos v5 token

The Username token, the X.509 tokens, and the LTPA tokens are used by default for message authenticity. The derived key token and the X.509 tokens are used by default for signing and encryption.

The WSS API and WSS SPI are only supported on the client. To specify the security token type on the generator side, you can also configure policy sets using the administrative console. You can also use the WSS APIs or policy sets for matching consumer security tokens.

The default Login Module and Callback implementations are designed to be used as a pair, meaning both a generator and a consumer part. To use the default implementations, select the appropriate generator and consumer security token in a pair. For example, select `system.wss.generate.x509` in the token generator and `system.wss.consume.x509` in the token consumer when the X.509 token is required.

To configure the generator-side security token, use the appropriate pre-configured token generator interface from the WSS APIs to complete the following token configuration process steps:

1. Generate the wssFactory instance.

2. Generate the wssGenerationContext instance.

   The WSSGenerationContext interface stores the components for generating Web Services Security (WS-Security), such as the signing and encryption information, the security token, and the time stamp. When the generate() method is called, all of these components are generated.
3. Create the generator-side components, such as the WSSSignature and the WSSEncryption objects.
4. Specify a JAAS configuration by specifying the name of the JAAS login configuration. The Java Authentication and Authorization Service (JAAS) configuration specifies the name of the JAAS configuration. The JAAS configuration specifies how the token logs in on the consumer side. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module.
5. Specifiy a token generator class name. The token generator class name specifies the required information to generate the SecurityToken. The Username token, the X.509 tokens, and the LTPA tokens are used by default for message authenticity.
6. Specifiy the settings for the callback handler by specifying a callback handler class name and also specifies the callback handler keys. This class name is the name of the callback handler implementation class that is used for the plug-in to the security token framework.

   The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is a plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the WSSGenerationContext interface.

   WebSphere Application Server provides the following default callback handler implementations for the generator side:

   com.ibm.websphere.wssecurity.callbackhandler.PropertyCallback
   This class is a callback for handling the name-value pair in elements in the Web Services Security (WS-Security) configuration XMI files.

   com.ibm.websphere.wssecurity.callbackhandler.UNTGUIPromptCallbackHandler
   This class is a callback handler for the Username token with the GUI prompt on the generator side. This instance is used to set the WSSGenerationContext object to generate a Username token.

   com.ibm.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
   This class is a callback handler for the Username token on the generator side. This instance is used to set into WSSGenerationContext object to attach a Username token. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

   com.ibm.websphere.wssecurity.callbackhandler.X509GenerateCallbackHandler
   This class is a callback handler that is used to generate the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token on the generator side. This instance is used to generate the WSSSignature and WSSEncryption objects, set the objects into the WSSGenerationContext object to generate the X.509 binary security tokens. A keystore and a key definition are required for this callback handler. If you use this implementation, a key store password, path, and type must be provided on the generator side.

   com.ibm.websphere.wssecurity.callbackhandler.LTPAGenerateCallbackHandler
   This class is a callback handler for the Lightweight Third Party Authentication (LTPA) tokens on the generator side. This instance is used to generate WSSSignature object and WSSEncryption object to generate a LTPA token.

   This callback handler is used to validate the LTPA security token inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run

As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a Java EE application client. If you use this implementation, a basic authentication user ID and password must have been provided on the generator side.

com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler
This class is a callback handler for the Kerberos v5 token on the generator side. This instance is used to set the WSSGenerationContext object to generate the Kerberos v5 AP-REQ as a binary security token. The instance is also used to generate the WSSSignature and WSSEncryption objects to use the Kerberos session key or derived key in the SOAP message signature and encryption.

7. If a X.509 token is specified, additional token information is also specified.

storeRef	The reference name of the keystore.
storePath	The keystore file path from which the keystore is loaded, if needed. It is recommended that you use the `${USER_INSTALL_ROOT}` in the path name as this variable expands to the WebSphere Application Server path on your machine. This path is required when you use the X.509 tokens callback handler implementations.
storePassword	The password that is used to check the integrity of the keystore, or the keystore password that is used to unlock the keystore and to access the keystore file. The keystore and its configuration are used for some of the default callback handler implementations that are provided by WebSphere Application Server.
storeType	The keystore type of keystore that is used for the key locator. This selection indicates the format that is used by the keystore file. The following values are available for selection: **JKS** Use this option if the keystore uses the Java Keystore (JKS) format. **JCEKS** Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption. **JCERACFKS** Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only). **PKCS11KS (PKCS11)** Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection. **PKCS12KS (PKCS12)** Use this option if your keystore uses the PKCS#12 file format.
alias	The key alias name. The key alias is used by the key locator to find the key within the keystore file.
keyPassword	The key password that is used for recovering the key. This password is needed to access the key object within the keystore file.
keyName	The name of the key. For digital signatures, the key name is used by the request generator or response consumer signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption. The key name must be a fully qualified, distinguished name (DN). For example, `CN=Bob,O=IBM,C=US`.
certStores	A list of certificate stores. A collection certificate store includes a list of untrusted, intermediary certificates and certificate revocation lists (CRLs). This step configures a collection certificate store and certificate revocation lists for the generator bindings.

identityAssertion	Specifies whether identity assertion is used. Selects this item if identity assertion is defined. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For an X.509 token generator, the application server sends the original signer certification only.
requestorCertificate	Specifies whether the certificate of the requestor is used.

The following can be specified for a X.509 token:

a. Without any keystore.

b. With a trust anchor. A trust anchor specifies a list of keystore configurations that contain trusted root certificates. These configurations are used to validate the certificate path of incoming X.509-formatted security tokens. For example, when you select the trust anchor or the certificate store of a trusted certificate, you must configure the trust anchor and the certificate store before setting the certificate path.

c. With a keystore that is used for the key locator.

First, you must have created the keystore file, by using a key tool utility, for example. The keystore is used to retrieve the X.509 certificate. This entry specifies the password that is used to access the keystore file. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trustworthiness of a certificate chain.

d. With keystore that is used for the key locator and the trust anchor.

e. With a map that includes key-value pairs. For example, you might specify the value type name and the value type Uniform Resource Identifier (URI). The value type specifies the namespace URI of the value type for the generated token, and represents the token type of this class:

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

Specifies an X.509 certificate token.

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

Specifies X.509 certificates in a public key infrastructure (PKI) path. This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used. If you use this implementation, you must provide a key store password, path, and type on this panel.

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Specifies a list of X.509 certificates and certificate revocation lists in a PKCS#7 format. This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You can specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. If you use this implementation, you must provide a key store password, path, and type.

For some tokens, WebSphere Application Server provides a predefined local name for the value type. When you specify the following local name, you do not need to specify a value type URI:

ValueType: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

For an LTPA token, you can use `LTPA` for the value type local name. This local name causes `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` to be specified for the value type Uniform Resource Identifier (URI).

ValueType: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

For LTPA token propagation, you can use `LTPA_PROPAGATION` for the value type local name. This local name causes`http://www.ibm.com/websphere/appserver/tokentype` to be specified for the value type Uniform Resource Identifier (URI).

8. If the Username token is specified as the token generator class name, the following token information can be specified:
   a. Whether to use IdentityAssertion option. This option is selected if identity assertion is defined. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server sends only the user name of the original caller for a Username token generator.
   b. Whether to use RunAsSubject identity option. This option is used if an identity assertion is defined and you want to use the Run As identity instead of the initial caller identity for identity assertion in a downstream call. This option is valid only if you have configured the Username token as the token generator.
   c. Whether to use sendRealm.
   d. Whether to specify the nonce.

      This option indicates whether a Nonce is included for the token generator. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of Username tokens. Nonce is valid only when the generated token type is a Username token, and it is available only for the request generator binding.
   e. Specifies the keyword of the time stamp. This option indicates whether to verify a time stamp in the Username token. The time stamp is valid only when the incorporated token type is a Username token.
   f. Specifies a map that includes key-value pairs. For example, you might specify the value type name and the value type Uniform Resource Identifier (URI). The value type specifies the namespace URI of the value type for the generated token, and represents the token type of this class:

      URI value type: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

      Specifies a Username token.
9. If the Kerberos v5 token is specified as the token generator class name, the following token information can be specified:

Token Information	Description	Default Value
name	Kerberos client principal name	
password	Kerberos client password	
realm	Kerberos realm associated with the Kerberos client	Default realm name in Kerberos configuration file. Specify null to use the default value.
targetService	Kerberos service name associated with the target Web Services.	
targetHost	Kerberos realm name associated with the Kerberos service name.	
tokenValueType	Kerberos token value type in QName defined by Oasis Kerberos Token Profile v1.1 specification.	http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ
targetRealm	Kerberos realm name associated with the Kerberos service name.	Default realm name in the Kerberos configuration file
prompt	A boolean value to enable the login prompt.	false

Token Information	Description	Default Value
supportTokenRequireSHA1	A boolean value to require a SHA1 key that is used in subsequent request messages when the Kerberos token is used as a supporting token.	false SHA1 key is consumed only if the supporting Kerberos token is protected. If set to true, the SHA1 key is always consumed.
alwaysAPREQ	A boolean value to indicate that the client should always send the Kerberos AP_REQ token in the request messages.	false The SHA1 key is used instead in the subsequent messages. If set to true, the Kerberos AP_REQ token is always used.
requireDKT	A boolean value to require a derived key for message protection.	false
clabel	The client label for the derived key.	WS-SecureConversation Specify null to use the default value.
slabel	The service label for the derived key.	WS-SecureConversation Specify null to use the default value.
keylen	The length of the derived key.	16 Specify zero to use the default value
noncelen	The length of the nonce.	16 Specify zero to use the default value
encComponent	An instance of WSSEncryption.	Set encComponent and sigComponent to null to initialize this first for either the encryption or signature component. Then, use the initialized component only in the callback handler constructor for the second component.
sigComponent	An instance of WSSSignature.	Set encComponent and sigComponent to null to initialize this first for either the encryption or signature component. Then, use the initialized component only in the callback handler constructor for the second component.

Additional token value types are defined in the OASIS Kerberos Token Profile v1.1 specification. Specify the token value type as the local name. It is not necessary to specify the value type URI for the Kerberos v5 token.

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120

10. If Secure Conversation is used for message protection, the following information must be specified:

Information	Description
bootstrapWSSGenerationContext	The bootstrap configuration used to secure the RequestSecurityToken (RST) token.
bootstrapWSSConmingContext	The bootstrap configuration used for consuming a secured RequestSecurityTokenResponse (RSTR).
ENDPOINT_URL	The service end point URL.
EncryptionAlgorithm	This determines the key size.
cLabel	The client label used when creating the derived key.
sLabel	The server label used when creating the derived key.

11. Set the components into the wssGenerationContext object.
12. Invoke the wssGenerationContext.process() method.

Results

Using the Web Services Security API (WSS API) process, you can configured the token generator.

What to do next

Next, you must specify a similar token consumer configuration.

Configuring the generator security tokens using the WSS API:

You can secure the SOAP messages, without using policy sets, by using the Web Services Security APIs. To configure the token on the generator side, use the Web Services Security APIs (WSS API). The generator security tokens are part of the com.ibm.websphere.wssecurity.wssapi.token interface package.

Before you begin

The pluggable token framework in WebSphere Application Server has been redesigned so that the same framework from the WSS API can be reused. The same implementation of creating and validating security token can be used both for the Web Services Security runtime and for the WSS API application code. The redesigned framework also simplifies the SPI programming model and will make it easier to add security token types.

You can use the WSS API or you can configure the tokens by using the administrative console. To configure tokens, you must complete the following token tasks:

- Configure the generator tokens.
- Configure the consumer tokens.

About this task

The JAAS CallbackHandler and JAAS LoginModule are responsible for creating the security token on the generator side.

On the generator side, the token is created by using the JAAS LoginModule and by using JAAS CallbackHandler to pass authentication data. Then, the JAAS LoginModule creates the securityToken object, such as the UsernameToken, and passes it to the Web Service Security run time.

On the consumer side, the XML format is passed to the JAAS LoginModule for validation or authentication. then the JAAS CallbackHandler is used to pass authentication data from the Web Service Security runtime to the LoginModule. After the token is authenticated, a security token object is created, and the token is passed it to the Web Service Security runtime.

When using the WSS API for generator token creation, certain default behaviors occur. The simplest way to use the WSS API is to use the default behavior (see the example code). The WSS API provide default values for the token type, the token value, and the JAAS confirmation name. The default token behaviors include:

Generator token decisions	Default behavior
Which token type to use	The token type specifies which type of token to use for message integrity, message confidentiality, or message authenticity. WebSphere Application Server provides the following pre-configured generator token types for message integrity and message confidentiality: • Derived key token • X509 tokens You can also create custom token types, as needed. WebSphere Application Server also provides the following pre-configured generator token types for the message authenticity: • Username token • LTPA tokens • X509 tokens You can also create custom token types, as needed.
What JAAS login configuration name to specify	The JAAS login configuration name specifies which JAAS login configuration name to use.
Which configuration type to use	The JAAS login module specifies the configuration type. Only the pre-configured generator configuration types can be used for generator token types.

The SecurityToken class (com.ibm.websphere.wssecurity.wssapi.token.SecurityToken) is the generic token class and represents the security token that has methods to get the identity, the XML format, and the cryptographic keys. Using the SecurityToken class, you can apply both the signature and encryption to the SOAP message. However, to apply both, you must have two SecurityToken objects, one for the signature and one for encryption, respectively.

The following tokens types are subclasses of the generic security token class:

Table 27. Subclasses of the SecurityToken

Token type	JAAS login configuration name
Username token	system.wss.generate.unt
Security context token	system.wss.generate.sct
Derived key token	system.wss.generate.dkt

The following tokens types are subclasses of the binary security token class:

Table 28. Subclasses of the BinarySecurityToken

Token type	JAAS login configuration name
LTPA token	system.wss.generate.ltpa
LTPA propagation token	system.wss.generate.ltpaProp

Table 28. Subclasses of the BinarySecurityToken (continued)

Token type	JAAS login configuration name
X.509 token	system.wss.generate.x509
X.509 PKI Path token	system.wss.generate.pkiPath
X.509 PKCS7 token	system.wss.generate.pkcs7

Notes®:

- For each JAAS login token generator configuration name, there is a respective token consumer configuration name. For example, for the Username token, the respective token consumer configuration name is system.wss.consume.unt.
- The LTPA and LTPA propagation tokens are only available to a requester that is running as a server-based client. The LTPA and LTPA propagation tokens are not supported for the Java SE 6 or Java EE application client.

1. To configure the securityToken package, com.ibm.websphere.wssecurity.wssapi.token, first ensure that the application server is installed.
2. Use the Web Services Security token generator process to configure the tokens. For each token type, the process is similar to the following process that demonstrates the UsernameToken token generator process:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSGenerationContext instance from the WSSFactory instance.
   c. Creates a JAAS CallbackHandler. The authentication data, such as the user name and password are specified as part of the CallbackHandler. For example, the following code specifies Chris as the user name and sirhC as the password: UNTGenerationCallbackHandler(″Chris″, ″sirhC″);
   d. Calls any JAAS CallbackHandler parameters and reviews the token class information for which parameters are required or optional. For example, for the UsernameToken, the following parameters can be configured also:

      Nonce
      Indicates whether a nonce is included in the user name token for the token generator. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The nonce value is valid only when the generated token type is a UsernameToken and only when it applies to the request generator binding.

      Created timestamp
      Indicates whether to insert a time stamp into the UsernameToken. The timestamp value is valid only when the generated token type is a UsernameToken and only when it applies to the request generator binding.

   e. Creates the SecurityToken from WSSFactory.

      By default, the UsernameToken API specifies the ValueType as: ″http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken″

      By default, the UsernameToken API provides the QName of this class and specifies the NamespaceURI as http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd and also specifies the LocalPart as UsernameToken.
   f. Optional: Specifies the JAAS login module configuration name. On the generator side, the configuration type is always `generate` (for example, system.wss.generate.unt).
   g. Adds the SecurityToken to the WSSGenerationContext.
   h. Calls WSSGenerationContext.process() and generates the WS-Security header.

Results

If there is an error condition, a WSSException is provided. If successful, the WSSGenerationContext process() is called, and the security token for the generator binding is attached.

Example

The following example code shows how the WSS API process creates a Username security token, attaches the Username token to the SOAP message, and configures the Username token in the generator binding.

```
// get the message context
   Object msgcontext = getMessageContext();

// generate WSSFactory instance
   WSSFactory factory = WSSFactory.getInstance();

// generate WSSGenerationContext instance
   WSSGenerationContext gencont = factory.newWSSGenerationContext();

// generate callback handler
   UNTGenerateCallbackHandler untCallbackHandler =
 new UNTGenerateCallbackHandler("Chris", "sirhC");

// generate the username token
   SecurityToken unt = factory.newSecurityToken(UsernameToken.class, untCallbackHandler);

// add the SecurityToken to the WSSGenerationContext
   gencont.add(unt);

// generate the WS-Security header
gencont.process(msgcontext);
```

The following example shows how to use secure conversation with the WSS APIs to configure the generator tokens, as well as the consumer tokens. In this example, the SecurityContextToken token is created using the WS-SecureConversation draft namespace: `http://schemas.xmlsoap.org/ws/2005/02/sc/sct`. To use the WS-SecureConversation version 1.3 namespace, `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct`, specify SecurityContextToken13.class instead of SecurityContextToken.class.

```
WSSGenerationContext bootstrapGenCon =
                          wssFactory.newWSSGenerationContext();

  // Create a Timestamp
                ..
  //add Timestamp
                ..


  // Sign the SOAP Body, WS-Addressing headers, and Timestamp
  X509GenerateCallbackHandler btspReqSigCbHandler = new X509GenerateCallbackHandler(
    ...);
  SecurityToken btspReqSigToken = wssFactory.newSecurityToken(
    X509Token.class, btspReqSigCbHandler);
  WSSSignature bootstrapReqSig = wssFactory.newWSSSignature(btspReqSigToken);
  bootstrapReqSig.setCanonicalizationMethod(WSSSignature.EXC_C14N);
                //add Sign Parts
  ..
                bootstrapGenCon.add(bootstrapReqSig);


  // Encrypt the SOAP Body and the Signature
  X509GenerateCallbackHandler btspReqEncCbHandler = new X509GenerateCallbackHandler(
    ...);
  SecurityToken btspReqEncToken = wssFactory.newSecurityToken(X509Token.class, btspReqEncCbHandler);
  WSSEncryption bootstrapReqEnc = wssFactory.newWSSEncryption(btspReqEncToken);
  bootstrapReqEnc.setEncryptionMethod(WSSEncryption.AES128);
  bootstrapReqEnc.setKeyEncryptionMethod(WSSEncryption.KW_RSA15);
                // add Encryption parts
  ..
                ..
                bootstrapGenCon.add(bootstrapReqEnc);

                WSSConsumingContext bootstrapConCon = wssFactory.newWSSConsumingContext();
                X509ConsumeCallbackHandler btspRspVfyCbHandler = new X509ConsumeCallbackHandler(....);
```

```
            WSSVerification bootstrapRspVfy = wssFactory.newWSSVerification(X509Token.class, btspRspVfyCbHandler);
            bootstrapRspVfy.addAllowedCanonicalizationMethod(WSSVerification.EXC_C14N);

            //add Verify parts
            ..
            ..
            bootstrapConCon.add(bootstrapRspVfy);

            X509ConsumeCallbackHandler btspRspDecCbHandler = new X509ConsumeCallbackHandler(...);
            WSSDecryption bootstrapRspDec = wssFactory.newWSSDecryption(X509Token.class, btspRspDecCbHandler);
            bootstrapRspDec.addAllowedEncryptionMethod(WSSDecryption.AES128);
bootstrapRspDec.addAllowedKeyEncryptionMethod(WSSDecryption.KW_RSA15);
            // add Decryption parts
            ..
            ..
            bootstrapConCon.add(bootstrapRspDec);

            SCTGenerateCallbackHandler sctgch =
 new SCTGenerateCallbackHandler(bootstrapGenCon, bootstrapConCon,
   ENDPOINT_URL, WSSEncryption.AES128);

            SecurityToken[] scts = wssFactory.newSecurityTokens(new Class[]{SecurityContextToken.class}, sctgch);
            SecurityContextToken sct = (SecurityContextToken)scts[0];


             // Use the SCT to generate DKTs for Secure Conversation
             //Signature algorithm and client and service labels
DerivedKeyToken dktSig = sct.getDerivedKeyToken(WSSSignature.HMAC_SHA1, "WS-SecureConversation", "WS-SecureConversation");
              //Encryption algorithm and client and service labels
DerivedKeyToken dktEnc = sct.getDerivedKeyToken(WSSEncryption.AES128, "WS-SecureConversation", "WS-SecureConversation");

            // Create the application generation context for the request message
WSSGenerationContext applicationGenCon = wssFactory.newWSSGenerationContext();
// Create and add Timestamp
..
..
// add the derived key token and Sign the SOAP Body and WS-Addressing headers
WSSSignature appReqSig = wssFactory.newWSSSignature(dktSig);

appReqSig.setSignatureMethod(WSSSignature.HMAC_SHA1);
appReqSig.setCanonicalizationMethod(WSSSignature.EXC_C14N);
..
            ..
applicationGenCon.add(appReqSig);
// add the derived key token and Encrypt the SOAP Body and the Signature
WSSEncryption appReqEnc = wssFactory.newWSSEncryption(dktEnc);

appReqEnc.setEncryptionMethod(WSSEncryption.AES128);
appReqEnc.setTokenReference(SecurityToken.REF_STR);
appReqEnc.encryptKey(false);
..
            ..
applicationGenCon.add(appReqEnc);

            // Create the application consuming context for the response message
WSSConsumingContext applicationConCon = wssFactory.newWSSConsumingContext();
            //client and service labels and decryption algorithm
SCTConsumeCallbackHandler sctCbHandler =
            new SCTConsumeCallbackHandler("WS-SecureConversation", "WS-SecureConversation", WSSDecryption.AES128);
// Derive the token from SCT and use it to Decrypt the SOAP Body and the Signature
WSSDecryption appRspDec = wssFactory.newWSSDecryption(SecurityContextToken.class, sctCbHandler);
appRspDec.addAllowedEncryptionMethod(WSSDecryption.AES128);
appRspDec.encryptKey(false);

            ..
applicationConCon.add(appRspDec);
//Derived the token from SCT and use it to Verify the signature on the SOAP Body, WS-Addressing headers, and Timestamp
WSSVerification appRspVfy = wssFactory.newWSSVerification(SecurityContextToken.class, sctCbHandler);
..
            ..
applicationConCon.add(appRspVfy);

            ..
            applicationGenCon.process(messageContext);
applicationConCon.process(messageContext);
```

What to do next

For each token type, configure the token using the WSS APIs or using the administrative console. Next, specify the similar consumer tokens if you have not done so.

If both the generator and consumer tokens are configured, continue securing SOAP messages either by signing the SOAP message or by encrypting the message, as needed. You can use either the WSS APIs or the administrative console to secure the SOAP messages.

Securing messages at the response consumer using WSS APIs:

You can secure SOAP messages with signature verification, decryption, and consumer tokens to protect message integrity, confidentiality, and authenticity, respectively. The response consumer (client-side) configuration defines the Web services security requirements for the incoming SOAP response.

About this task

To secure Web services with WebSphere Application Server, you must configure the generator and the consumer security constraints. You must specify several different configurations. Although there is no specific sequence to specify these different configurations, some configurations reference other configurations. For example, decryption configurations reference encryption configurations.

The response consumer (client-side) configuration requirements involve verifying that the integrity parts are signed and that the signature is verified, verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security tokens.

You can use the following methods to configure Web services security and to define policy types to secure the SOAP messages:

- Use the administrative console to configure policy sets.
- Use the Web Services Security APIs (WSS API) to configure the SOAP message context (only for the client)

The following high-level steps use the WSS APIs:

- Verify consumer signing information to protect message integrity.
- Configure decryption to protect message confidentiality.
- Validate consumer tokens to protect message authenticity.

Results

After completing these procedures, you have secured messages at the response consumer level.

What to do next

Next, if not already configured, secure messages with signing information, encryption, and generator tokens at the response (client-side) generator level.

Configuring decryption to protect message confidentiality using the WSS APIs:

You can configure decryption information for the response consumer (client side) section of the binding file. Decryption information is used to specify how the consumers (receivers) decrypt incoming SOAP messages. To configure decryption, specify which message parts to decrypt and specify which algorithm methods and security tokens are to be used for decryption.

Before you begin

Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the message flowing across the Internet. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the correct target. Prior to configuring decryption, familiarize yourself with XML encryption.

About this task

For decryption, you must specify the following:

- Which parts of the message are to be decrypted.

- Which decryption algorithms to specify.

To configure decryption and decrypted parts on the client side, use the WSSDecryption and WSSDecryptPart APIs, or configure policy sets using the administrative console.

WebSphere Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

WebSphere Application Server uses decryption information for the default consumer to decrypt parts of the SOAP message. The WSSDecryption API configures the following required parts as decrypted parts.

Table 29. Required decrypted parts

Decryption parts	Description
Keywords	Keywords are used to add the decrypted parts to the SOAP message.
XPath expression	XPath expressions are used to add the decrypted parts to the SOAP message.
WSSDencryptPart object	This object adds the decrypted parts to the SOAP message.
WSSVerification object	This object adds the signature verification component as a decrypted part.
Header	This part adds the header in the SOAP header, specified by QName, as a decrypted part.
Security token object	This object adds the security token as a decrypted part.

Web Services Security API (WSS API) supports symmetric encryption, by using a shared key, only when Web Services Secure Conversation (WS-SecureConversation) is used.

The WSS APIs allow the use of either keywords or an XPath expression to specify the parts of the SOAP message that are to be decrypted. WebSphere Application Server supports the use of the following keywords:

Table 30. Supported decryption keywords

Keyword	References
BODY_CONTENT	The keyword for the body contents of the SOAP message body as a decryption target.
SIGNATURE	The keyword for the signature element as a decryption target.
USERNAME_TOKEN,	The keyword for the Username token element as a decryption target.

If configuring using the WSS APIs, the WSSDecryption and WSSDecryptPart APIs complete these high-level steps:

1. Use the WSSDecryption API to configure encryption. The WSSDecryption API performs these tasks by default:
   a. Generates the callback handler.
   b. Generates the consumer security token object.
   c. Adds the security token reference type.
   d. Adds the WSSEncryptPart object.
   e. Adds the parts to be encrypted. Adds the default parts for decryption by using keywords and XPath expressions.
   f. Adds the verification component.
   g. Adds the header in the SOAP message, specified by QName.

h. Sets the default data encryption method.
i. Specifies whether the key is to be decrypted using a Boolean value. Calls this method when the shared key is encrypted.
j. Sets the default key encryption method.

2. Use the WSSEncryptPart API to configure encrypted parts or add a transform method. The WSSEncryptPart API performs these tasks by default:
   a. Sets the encrypted parts specified by using keywords or an XPath expression.
   b. Sets the encrypted parts specified by an XPath expression.
   c. Sets the signature component object, WSSSignature.
   d. Sets the header in the SOAP message, specified by QName.
   e. Sets the generator security token.
   f. Adds the transform method, if needed.
3. Change from the default values for algorithm or message parts, as needed. For example: you could change one or more of the following items:
   - Add USERNAME_TOKEN as a target of decryption.
   - Change the data encryption algorithm from the default value of AES 128.
   - Change the key encryption algorithm from the default value of KW_RSA_OAEP.
   - Specify to not encrypt the encryption key (false).
   - Change the security token type from the default value of X.509 token.
   - Only use BODY_CONTENT as an encryption part and not use SIGNATURE also.

Results

The decryption information is configured for the consumer binding.

Example

The following is an example of the WSSDecryption API:

```
WSSFactory factory = WSSFactory.getInstance();
WSSConsumingContext concont = factory.newWSSConsumingContext();
    X509ConsumeCallbackHandler callbackhandler = generateCallbackHandler();
// see X509ConsumeCallbackHandler
    WSSDecryption dec = factory.newWSSDecryption(X509Token.class,
        callbackhandler);

 concont.add(dec);
```

What to do next

You must configure similar encryption information for the client-side request generator (sender) bindings, if you have not already configured the information.

Next, review the WSSDecryption API process.

Decrypting the SOAP message using the WSSDecryption API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web Services Security APIs (WSS API). To configure the client for decryption on the response (client) consumer side, use the WSSDecryption API to decrypt the SOAP messages. The WSSDecryption API specifies which request SOAP message parts to decrypt when configuring the client.

Before you begin

You can use the WSS API or use policy sets on the administrative console to enable decryption and add consumer security tokens in the SOAP message. To secure SOAP messages, you must have completed the following decryption tasks:

- Encrypted the SOAP message.
- Chosen the decryption method.

About this task

The decryption information on the consumer side is used for decrypting an incoming SOAP message for the response consumer (client side) bindings. The client consumer configuration must match the configuration for the provider generator.

Confidentiality settings require that confidentiality constraints be applied to generated messages.

The following decryption parts can be configured:

Table 31. Decryption parts

Decryption parts	Description
part	Adds the WSSDecryptPart object as a target of the decryption part.
keyword	Adds the decryption part using keywords. WebSphere Application Server supports the following keywords: • BODY_CONTENT • SIGNATURE • USERNAME_TOKEN
xpath	Adds the decryption part using an XPath expression.
verification	Adds the WSSVerification instance as a target of the decryption part.
header	Adds the SOAP header, specified by QName, as a target of the decryption part.

For decryption, certain default behaviors occur. The simplest way to use the WSS API for decryption is to use the default behavior (see the example code). WSSDecryption provides defaults for the key encryption algorithm, the data encryption algorithm, and the decryption parts such as the SOAP body content and the signature. The decryption default behaviors include:

Table 32. Decryption decisions

Decryption decisions	Default behavior
Which parts to decrypt	The default decryption parts are the BODY_CONTENT and SIGNATURE. WebSphere Application Server supports using these keywords: • WSSDecryption.BODY_CONTENT • WSSDecryption.SIGNATURE • WSSDecryption.USERNAME_TOKEN After you specify which message parts to decrypt, you must specify which method to use when decrypting the consumer request message. For example, if both signature and body content are applied for encryption, then the SOAP message parts that are decrypted include the same parts.
Whether to encrypt the key (isEncrypt)	The default value is to encrypt the key (`true`).

Table 32. Decryption decisions (continued)

Decryption decisions	Default behavior
Which data decryption algorithm to choose (method)	The default data decryption algorithm method is AES128. WebSphere Application Server supports these data encryption methods: • WSSDecryption.AES128: http://www.w3.org/2001/04/xmlenc#aes128-cbc • WSSDecryption.AES192: http://www.w3.org/2001/04/xmlenc#aes192-cbc • WSSDecryption.AES256: http://www.w3.org/2001/04/xmlenc#aes256-cbc • WSSDecryption.TRIPLE_DES: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Which key decryption method to choose (algorithm)	The default key decryption algorithm method is key wrap RSA OAEP. WebSphere Application Server supports these key encryption methods: • WSSDecryption.KW_AES128: http://www.w3.org/2001/04/xmlenc#kw-aes128 • WSSDecryption.KW_AES192: http://www.w3.org/2001/04/xmlenc#kw-aes192 • WSSDecryption.KW_AES256: http://www.w3.org/2001/04/xmlenc#kw-aes256 • WSSDecryption.KW_RSA_OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p • WSSDecryption.KW_RSA15: http://www.w3.org/2001/04/xmlenc#rsa-1_5 • WSSDecryption.KW_TRIPLE_DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes
Which security token to specify	The default security token type is the X509 token. WebSphere Application Server provides the following pre-configured consumer token types: • Derived key token • X509 tokens

1. To decrypt the SOAP message using the WSSDecryption API, first ensure that the application server is installed.
2. The WSS API process for decryption performs these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSConsumingContext instance from the WSSFactory instance. The WSSConsumingContext must always be called in a JAX-WS client application.
   c. Creates the callback handler for the consumer side.
   d. Creates WSSDecryption with the class for the security token and the callback handler from the WSSFactory instance. The default behavior of WSSDecryption is to assume that the body content and the signature are encrypted.
   e. Adds the parts to be decrypted, if the default is not appropriate.
   f. Adds the candidates of the data encryption methods to use for decryption.
   g. Adds the candidates of the key encryption methods to use for decryption.
   h. Adds the candidates of the security token to use for decryption.
   i. Calls WSSDecryption.encryptKey(false) if the application does not want the key to be encrypted in the incoming message.
   j. Adds WSSDecryption to WSSConsumingContext.
   k. Calls WSSConsumingContext.process() with the SOAPMessageContext

Results

If there is an error condition during decryption, a WSSException is provided. If successful, the WSSConsumingContext.process() is called, and Web services security is applied to the SOAP message.

Example

The following example provides sample code for decrypting the SOAP message body content:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance (step: b)
   WSSConsumingContext gencont = factory.newWSSConsumingContext();

// Generate the callback handler (step: c)
   X509ConsumeCallbackHandler callbackHandler = new
       X509ConsumeCallbackHandler(
                                  "",
                                  "enc-sender.jceks",
                                  "jceks",
                                  "storepass".toCharArray(),
                                  "alice",
                                  "keypass".toCharArray(),
 "CN=Alice, O=IBM, C=US");

// Generate the WSSDecryption instance (step: d)
   WSSDecryption dec = factory.newWSSDecryption(X509Token.class,
                                                callbackHandler);

// Set the part to be encrypted (step: e)
// DEFAULT: WSSEncryption.BODY_CONTENT and WSSEncryption.SIGNATURE

// Set the part to be encrypted (step: e)
// DEFAULT: WSSEncryption.BODY_CONTENT and WSSEncryption.SIGNATURE

// Set the part specified by the keyword (step: e)
      dec.addRequiredDecryptPart(WSSDecryption.BODY_CONTENT);

// Set the part in the SOAP Header specified by QName (step: e)
    dec.addRequiredDecryptHeader(new
       QName("http://www.w3.org/2005/08/addressing",
       "MessageID"));

// Set the part specified by WSSVerification  (step: e)
   X509ConsumeCallbackHandler verifyCallbackHandler =
      getCallbackHandler();
   WSSVerification ver = factory.newWSSVerification(X509Token.class,
                                                    verifyCallbackHandler);
      dec.addRequiredDecryptPart(ver);

// Set the part specified by XPath expression (step: e)
      StringBuffer sb = new StringBuffer();
         sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
            and local-name()='Envelope']");
         sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
            and local-name()='Body']");
         sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
            and local-name()='Ping']");
         sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
            and local-name()='Text']");
      dec.addRequiredDecryptPartByXPath(sb.toString());

// Set the part in the SOAP header to be decrypted specified by QName (step: e)
      dec.addRequiredDecryptHeader(new
          QName("http://www.w3.org/2005/08/addressing",
          "MessageID"));

// Set the candidates for the data encryption method (step: f)
// DEFAULT : WSSDecryption.AES128
      dec.addAllowedEncryptionMethod(WSSDecryption.AES128);
      dec.addAllowedEncryptionMethod(WSSDecryption.AES192);

// Set the candidates for the key encryption method (step: g)
```

```
// DEFAULT : WSSDecryption.KW_RSA_OAEP
      dec.addAllowedKeyEncryptionMethod(WSSDecryption.KW_TRIPLE_DES);

// Set the candidate security token to used for the decryption (step: h)
   X509ConsumeCallbackHandler callbackHandler2 = getCallbackHandler2();
      dec.addToken(X509Token.class, callbackHandler2);

// Set whether or not the key should be encrypted in the incoming SOAP message (step: i)
// DEFAULT: true
      dec.encryptKey(true);

// Add the WSSDecryption to the WSSConsumingContext (step: j)
   concont.add(dec);

// Validate the WS-Security header (step: k)
concont.process(msgcontext);
```

What to do next

Next, use the WSSDecryptPart API or configure the policy sets using the administrative console to add decrypted parts for the consumer message.

Choosing the decryption methods for the consumer binding:

To configure the client for response decryption for the consumer binding, specify which data and transform algorithm methods to use when the client decrypts the SOAP messages.

Before you begin

Prior to completing these steps, read the XML encryption information to become familiar with encrypting and decrypting SOAP messages.

To complete decryption configuration to secure SOAP messages, you must complete the following tasks:

- Configure decryption of the SOAP message parts
- Specify the decryption methods.

You can configure the decryption methods using the WSSDecryption and WSSDecryptPart APIs. Or you can also configure policy sets using the administrative console to configure the decryption methods.

About this task

Some of the encryption-related definitions are based on the XML-Encryption specification. The following information defines some data encryption-related terms:

Data encryption method algorithm

Data encryption algorithms specify the algorithm uniform resource identifier (URI) of the data encryption method. This algorithm encrypts and decrypts data in fixed size, multiple octet blocks.

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files.

For the AES256-cbc and the AES192-cbc algorithms, you must download the unrestricted Java™ Cryptography Extension (JCE) policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

Key encryption method algorithm

Key encryption algorithms specify the algorithm uniform resource identifier (URI) of the key encryption method. The algorithm represents public key encryption algorithms that are specified for encrypting and decrypting keys.

By default, the RSA_OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod. The property value is one of the following URIs of the digest method:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmlenc#sha512

By default, the RSA_OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

For the KW_AES256 and the KW_AES192 key encryption algorithms, you must download the unrestricted JCE policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

To complete the decryption configuration, you must specify the algorithm uniform resource identifier (URI) and its usage type. If the URI is used for multiple usage types, then you must define the URI to each usage type. WebSphere Application Server supports the following decryption usage types:

Table 33. Decryption usage types

Usage types	Description
Data encryption	Specifies the algorithm URI that is used for both encrypting and decrypting data. Encrypts and decrypts data in fixed size, multiple octet blocks.
Key encryption	Specifies the algorithm URI that is used for encrypting and decrypting the encryption key.

To configure the decryption and decrypted part algorithms, use the WSSDecryption and WSSDecryptPart APIs, or configure policy sets using the administrative console.

Note: Policy sets do not support symmetric key encryption. If you are using the WSS API for symmetric key encryption, you will not be able to interoperate with Web services endpoints that use policy sets.

If you are using the WSS APIs, the WSSDecryption and WSSDecryptPart APIs specify which algorithm methods are used when the client decrypts the SOAP messages.

- Use the WSSDecryption API to configure the data encryption algorithm and the key encryption algorithm methods.
- Use the WSSDecryptPart API to configure a transform algorithm method.

The WSS API process completes the following high-level steps to specify which decryption and decrypted part algorithm methods to use when configuring the client for response decryption:

1. Using the WSSDecryption API, adds the required data encryption algorithm. The data encryption algorithm is used for encrypting or decrypting parts of a SOAP message. Data decryption algorithms specify the algorithm uniform resource identifier (URI) of the data encryption method.

   The default data encryption algorithm is AES 128. The data encryption name is AES128, and the URI of the data encryption algorithm, is http://www.w3.org/2001/04/xmlenc#aes128-cbc. WebSphere Application Server supports the following pre-configured data decryption algorithms:

   - AES128: http://www.w3.org/2001/04/xmlenc#aes128-cbc

The AES 128 algorithm is the default data algorithm method.

- AES256: http://www.w3.org/2001/04/xmlenc#aes256-cbc

  To use this AES 256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- AES192: http://www.w3.org/2001/04/xmlenc#aes192-cbc

  Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

  To use this AES 192-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- TRIPLE_DES: http://www.w3.org/2001/04/xmlenc#tripledes-cbc

2. As needed, changes the WSSEncryption API method to specify another data encryption algorithm. For example, you might add the following code to change from the default AES 128 algorithm to the Triple DES algorithm:

   ```
   dec.addAllowedKeyEncryptionMethod(WSSDecryption.TRIPLE_DES);
   ```

3. Using the WSSDecryption API, adds the required key encryption algorithm. The key encryption algorithm is used for encrypting the key that is used for encrypting the message parts within the SOAP message. If no key for encrypting the data is needed, then you must specify WSSDecryption.encryptKey(false).

   The key encryption algorithm that you select for the consumer side must match the key encryption method that you select for the generator side.

   The default key encryption algorithm value is key wrap RSA_OAEP. The key encryption name is KW_RSA_OAEP, and the URI of the key encryption algorithm is http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p. WebSphere Application Server supports the following pre-configured key encryption algorithms:

   - KW_AES128: http://www.w3.org/2001/04/xmlenc#kw-aes128
   - KW_AES192: http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use this key wrap AES 192 algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
   - KW_AES256: http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this key wrap AES 256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - KW_RSA_OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

     The KW_RSA_OAEP algorithm is the default key algorithm method.

     When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this algorithm. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5. See more information at http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
   - KW_RSA_15: http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - KW_TRIPLE_DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes

   Note: For Web Services Secure Conversation, the WSSEncryption API might specify addition key-related information, such as the:
   - algorithmName
   - keyLength

4. As needed, uses the WSSDecryption API method to change to other key encryption algorithms. For example, you might add the following code to change from the default key encryption algorithm KW_RSA_OAEP to the TRIPLE_DES algorithm:

   ```
   dec.addAllowedKeyEncryptionMethod(WSSDecryption.KW_TRIPLE_DES);
   ```

5. Using the WSSDecryptPart API, adds a transform algorithm, as needed. There is no default transform algorithm. However, WebSphere Application Server provides a pre-configured decrypted part, WSSDecryptPart.TRANSFORM_ATTACHMENT_CIPHERTEXT, that can be added.

Results

If there is an error condition, a WSSException is provided. If successful, the API calls the WSSConsumerContext.process(), the WS-Security header is validated, and the SOAP message is now secured using Web services security.

Example

The following example provides sample WSS API code for decrypting the body content as well as changing the data encryption and key encryption algorithms from the default values:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance
   WSSConsumingContext gencont = factory.newWSSConsumingContext();

// Generate the callback handler
   X509ConsumeCallbackHandler callbackHandler = new
      X509ConsumeCallbackHandler(
                                 "",
                                 "enc-sender.jceks",
                                 "jceks",
                                 "storepass".toCharArray(),
                                 "alice",
                                 "keypass".toCharArray(),
                                 "CN=Alice, O=IBM, C=US");

// Generate WSSDecryption instance
   WSSDecryption dec = factory.newWSSDecryption(X509Token.class,
                                                callbackHandler);

// Set the candidates for the data encryption method
// DEFAULT : WSSDecryption.AES128
   dec.addAllowedEncryptionMethod(WSSDecryption.AES128);
   dec.addAllowedEncryptionMethod(WSSDecryption.AES192);

// Set the candidates for the key encryption method
// DEFAULT : WSSDecryption.KW_RSA_OAEP
   dec.addAllowedKeyEncryptionMethod(WSSDecryption.KW_TRIPLE_DES);

// Add the WSSDecryption to WSSConsumingContext
   concont.add(dec);

// Validate the WS-Security header
concont.process(msgcontext);
```

Adding decrypted parts using the WSSDecryptPart API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web services security APIs (WSS API). To configure decrypted parts for the response consumer (client side) bindings, use the WSSDecryptPart API to define and add to the listing of elements in the decrypted part. WSSDecryptPart is an interface that is part of the com.ibm.websphere.wssecurity.wssapi.decryption package.

Before you begin

You can use either the WSS APIs or configure the policy sets using the administrative console to configure and add new encrypted parts. To secure SOAP messages using the WSSDecryptPart APIs, you must configure the decrypted parts for the response consumer bindings.

About this task

Confidentiality settings require that confidentiality constraints be applied to generated messages. These constraints include specifying which message parts within the generated message must be encrypted and decrypted, and which message parts to attach encrypted elements to.

The WSSDecryptPart API specifies information related to decryption and sets the decrypted parts that have been added for message confidentiality protection. Use the WSSDecryptPart to set the transform method and to specify the part to which the transform method is to be applied. Sets the transform method only if using SOAP with Attachments. The WSSDecryptPart is usually not needed except, in some case for tasks such as setting the transform method.

The decrypted parts displayed in the following table are used to protect the confidentiality of messages.

Table 34. Decrypted Parts

Decrypted parts	Description
keyword	Sets the decrypted part using keywords. The default decrypted parts that you can add using keywords are the BODY_CONTENT and SIGNATURE. WebSphere Application Server supports the following keywords: • BODY_CONTENT • SIGNATURE • USERNAME_TOKEN
xpath	Sets the decrypted part by using an XPath expression.
verification	Sets the WSSVerification component as a decrypted part. The WSSVerification part is applicable only if the SOAP message contains a signature element.
header	Sets the header, specified by QName, as a decrypted part.

For decrypted parts, certain default behaviors occur. The simplest way to use the WSSDecryptPart API is to use the default behavior (see the example code).

WSSDecryptPart provides defaults for setting the transform algorithm, adding a transform method, setting objects as targets, whether an element, and the encrypted parts, such as: the SOAP body content and the signature.

Table 35. Decrypted part decisions

Decryption decisions	Default behavior
Which SOAP message parts to decrypt using keywords	Specifies which keywords to use for the decrypted parts. WebSphere Application Server sets the following SOAP message parts by default for decryption: • WSSDecryption.BODY_CONTENT • WSSDecryption.SIGNATURE
Which transform algorithm to use (algorithm)	WebSphere Application Server does not specify any transform algorithm by default. Specify a transform method only if using SOAP with Attachments.

1. To decrypt the SOAP message parts using the WSSDecryptPart API, first ensure that the application server is installed.
2. The WSS API process using WSSDecryptPart follows these steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSConsumingContext instance from the WSSFactory instance. Note that the WSSConsumingContext must always be called in a JAX-WS client application.
   c. Creates the SecurityToken from WSSFactory to configure decryption.
   d. Creates WSSDecryption from the WSSFactory instance using SecurityToken.
   e. Creates WSSDecryptPart from the WSSFactory instance. The default behavior of WSSDecryptPart is to assume that the body content and signature are encrypted.
   f. Adds the parts to be decrypted and to be applied with the transform in WSSDecryptPart. WebSphere Application Server sets these encrypted parts by default for WSSDecryptPart: the BODY_CONTENT and SIGNATURE. After you add other decrypted parts, the default values are no longer valid. For example, if you call addDecryptPart(securityToken, false), only the security token is encrypted, and not the signature and body content. So if you want to decrypt the security token, the signature, and the body content, you must call addDecryptPart(securityToken, false), addDecryptPart(WSSDecryption.SIGNATURE), and addDecryptPart(WSSDecryption.BODY_CONTENT).
   g. Sets the transform method.
   h. Adds WSSDecryptPart to WSSDecryption.
   i. Adds WSSDecryption to WSSConsumingContext.
   j. Calls WSSConsumingContext.process() with the SOAPMessageContext

Results

If there is an error condition when decrypting the message, a WSSException is provided. If successful, the API calls the WSSConsumingContext.process(), the WS-Security header is generated, and the SOAP message is now secured using Web services security.

What to do next

After enabling decrypted parts for the response consumer (client side) binding, specify the generator and consumer tokens, if the security tokens have not already been specified.

Decryption methods:

The decryption algorithms specify the data and key encryption algorithms that are used to decrypt the SOAP message. The WSS API for decryption (WSSDecryption) specifies the algorithm uniform resource identifier (URI) of the data and key encryption methods. The WSSDecryption interface is part of the com.ibm.websphere.wssecurity.wssapi.decryption package.

Data encryption algorithms

The data encryption algorithms are the algorithms that are used to encrypt and decrypt data. This algorithm type is used for encrypting data to encrypt and decrypt various parts of the message, including the body content and the signature.

Data decryption algorithms specify the algorithm uniform resource identifier (URI) of the data encryption method. WebSphere Application Server supports the following pre-configured data decryption algorithms:

Table 36. Supported pre-configured data decryption algorithms

WSS API	URI
WSSDecryption.AES128 (the default value)	A URI of data encryption algorithm, AES 128: http://www.w3.org/2001/04/xmlenc#aes128-cbc
WSSDecryption.AES192	A URI of data encryption algorithm, AES 192: http://www.w3.org/2001/04/xmlenc#aes192-cbc
WSSDecryption.AES256	A URI of data encryption algorithm, AES 256: http://www.w3.org/2001/04/xmlenc#aes256-cbc
WSSDecryption.TRIPLE_DES	A URI of data encryption algorithm, TRIPLE DES: http://www.w3.org/2001/04/xmlenc#tripledes-cbc

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files.

For the AES256-cbc and the AES192-cbc algorithms, you must download the unrestricted Java™ Cryptography Extension (JCE) policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

The data encryption algorithm must match the data decryption algorithm that is configured for the consumer.

Key encryption algorithms

The key encryption algorithms are the algorithms that are used to encrypt and decrypt keys.

This key information is used to specify the configuration that is needed to generate the key for digital signature and encryption. The signing information and encryption information configurations can share the key information. The key information on the consumer side is used for specifying the information about the key that is used for validating the digital signature in the received message or for decrypting the encrypted parts of the message. The request generator is configured for the client.

Key encryption algorithms specify the algorithm uniform resource identifier (URI) of the key encryption method. WebSphere Application Server supports the following pre-configured key encryption algorithms:

Table 37. Supported pre-configured key encryption algorithms

WSS API	URI
WSSDecryption.KW_AES128	A URI of key encryption algorithm, key wrap AES 128: http://www.w3.org/2001/04/xmlenc#kw-aes128
WSSDecryption.KW_AES192	A URI of key encryption algorithm, key wrap AES 192: http://www.w3.org/2001/04/xmlenc#kw-aes192 **Note:** Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
WSSDecryption.KW_AES256	A URI of key encryption algorithm, key wrap AES 256: http://www.w3.org/2001/04/xmlenc#kw-aes256
WSSDecryption.KW_RSA_OAEP (the default value)	A URI of key encryption algorithm, key wrap RSA OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
WSSDecryption.KW_RSA15	A URI of key encryption algorithm, key wrap RSA 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5
WSSDecryption.KW_TRIPLE_DES	A URI of data encryption algorithm, key wrap TRIPLE DES: http://www.w3.org/2001/04/xmlenc#kw-tripledes

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod. The property value is one of the following URIs of the digest method: http://www.w3.org/2001/04/xmlenc#sha256 http://www.w3.org/2001/04/xmlenc#sha512

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

For the kw-aes256 and the kw-aes192 key encryption algorithms, you must download the unrestricted JCE policy files from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

The key encryption algorithm for the generator and the consumer must match.

The following example provides a sample of the WSS API code for the default algorithms that are used for WebSphere Application Server decryption:

```
WSSFactory factory = WSSFactory.getInstance();
WSSConsumingContext concont = factory.newWSSConsumingContext();

// Required to attach username token into the message.
X509ConsumeCallbackHandler callbackHandler =
       new X509ConsumeCallbackHandler("",
                                      "enc-sender.jceks",
                                      "JCEKS",
                                      "storepass".toCharArray(),
                                      "alice",
                                      "keypass".toCharArray(),
                                      "CN=Alice, O=IBM, C=US");
// Set the decrypt component.
// Default encrypted part: Body-Content
// Default data encryption algorithm: AES128
// Default key encryption algorithm: KW-RSA-OAEP
WSSDecryption dec = factory.newWSSDecryption(X509Token.Type,
callbackHandler);
concont.add(dec);

// validate the WS-Security header.
concont.process(msgctx);
```

Verifying consumer signing information to protect message integrity using WSS APIs:

You can verify the signing information to protect message integrity for the response (client side) consumer binding. Signing information includes the signature and the signed parts for the generator side as well as signature verification and verify parts for the consumer side. To keep the integrity of the message, digital signatures are typically applied.

Before you begin

Ensure that the signature and signed parts information has been configured. The signature verification information must match what was configured on the generator side.

About this task

Integrity refers to digital signature while confidentiality refers to encryption. Integrity is provided by applying a digital signature to a SOAP message. To configure the signing information to protect message integrity,

you must first digitally sign and then verify the signature for the SOAP messages. Integrity decreases the risk of data modification when you transmit data across a network.

Also, message integrity is provided by verifying the digitally signed body, time stamp, and WS-Addressing headers using the signature verification algorithm methods. The WSS APIs specify which algorithm is to be used to verify the certificate. The signature algorithms specify the Uniform Resource Identifiers (URI) of the signature verification method. WebSphere Application Server supports several pre-configured verification algorithm methods.

You can use the following interfaces to configure Web services security and to protect SOAP message integrity:

- Use the administrative console to configure policy sets for signature verification.
- Use the Web Services Security APIs (WSS API) to configure the SOAP message context (only for the client)

Perform the following verification tasks, using the WSS APIs, to configure the signing information and to protect message integrity for the consumer binding.

- Configure the signing information using the WSSSignature API. Configure the signature verification information for the consumer binding using the WSSVerification API. Signature verification information is used to verify parts of a message including the SOAP body, the time stamp, and the WS-Addressing headers. Both verifying and decryption can be applied to the same message parts, such as the SOAP body.
- Add or change verify parts using the WSSVerifyPart API.
- Configure the client for request signing methods using the WSSVerification or WSSVerifyPart APIs. To configure the client for response verification, choose the verification methods. Use the WSSVerification API to configure the canonicalization and signature methods. Use the WSSVerifyPart API to configure the digest and transform methods.

Results

By completing the steps in these tasks, you have configured the consumer verification information to protect the integrity of messages.

Verifying the signature information for the consumer binding using the WSS APIs:

You can configure the signing information for the client-side response consumer (receiver) bindings. Signing information is used to sign and validate parts of a message including the SOAP body, the timestamp information, and the Username token.

Before you begin

WebSphere Application Server uses XML digital signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method. Prior to completing these steps, read the information about XML digital signature to become familiar with signing and verifying digital signatures for digital content.

By including XML signature in SOAP messages, the following issues are realized: message integrity and authentication. *Integrity* refers to digital signature whereas confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet.

Before you can verify the signature and SOAP message signed parts, you must have completed the following tasks:

- Configured the signature.
- Added signed parts, as needed.

- Chosen the signature and signed parts methods.

About this task

Use the Web Services Security APIs (WSS API) to configure the signing verification information for the response consumer (client side) section of the bindings file. Use the WSSVerification or WSSVerifyPart APIs to configure the client for request signature verification and to specify which digitally signed message parts to verify.

WebSphere Application Server uses the signing information on the consumer side to verify the integrity of the received SOAP message by validating that the message parts (such as the body, time stamp, and Username token) are signed.

On the client side, use the WSS APIs, or configure policy sets using the administrative console to specify which parts of the message are signed and to configure the key information that is referenced by the key information references. To verify the signature and signed parts, use the WSSVerification and WSSVerifyPart APIs.

WebSphere Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

The WSSVerification and WSSVerifyPart APIs complete the following steps to specify which digitally signed message parts to verify when configuring the client for response consumer signing:

1. The WSSVerification API adds the required verify parts of the SOAP message.

   The part reference refers to the message part that is digitally signed. The part attribute refers to the name of the <Integrity> element when the <PartReference> element is specified for the signature. You can specify multiple <PartReference> elements within the <SigningInfo> element. The <PartReference> element has two child elements when it is specified for the signature: <DigestTransform> and <Transform>.

   The WSSVerification API configures the following parts as verification parts:

Security token	Adds information for the security token that is used for the signature verification.
SOAP header and the QName as a target	Adds the SOAP header, specified by QName, as a verification part.

   The WSS APIs allow the use of keywords or an XPath expression to specify which parts of the message are to be verified. WebSphere Application Server supports the use of the following keywords:

Keyword	**References**
WSSVerification.ADDRESSING_HEADERS	The Web Services Addressing (WS-Addressing) headers.
WSSVerification.BODY	The SOAP message body. The body is the user data portion of the message.
WSSVerification.TIMESTAMP	The creation and expiration timestamp information.

2. The WSSVerification API adds the required header to the SOAP message. The header, specified by QName, is a required verification header.
3. The WSSVerification API adds a security token. Adds information about the security token that is to be used for the signature verification, such as:
   - The class for security token.
   - The callback handler
   - The name of the JAAS login configuration.

4. The WSSVerification API adds the signature method algorithm. The signature method is the algorithm that is used to convert the canonicalized <SignedInfo> element in the binding file into the <SignatureValue> element. The algorithm that is specified for the consumer, which is the response consumer configuration, must match the algorithm specified for the request generator configuration. WebSphere Application Server supports the following pre-configured signature algorithms:
   - WSSVerification.RSA_SHA1:http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - WSSVerification.HMAC_SHA1:http://www.w3.org/2000/09/xmldsig#hmac-sha1

   WebSphere Application Server does not support the following algorithm for DSA-SHA1: http://www.w3.org/2000/09/xmldsig#dsa-sha1. You cannot use the DSA-SHA1 algorithm if you want to be compliant with the Basic Security Profile (BSP).
5. The WSSVerification API adds a canonicalization method. The canonicalization method algorithm is used to canonicalize the <SignedInfo> element before it is incorporated as part of the digital signature operation. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer.

   WebSphere Application Server supports the following pre-configured canonicalization algorithms:
   - WSSVerification.EXC_C14N: http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSVerification.C14N: http://www.w3.org/TR/xml-c14n
6. The WSSVerification API verifies whether a signature confirmation is required. The OASIS Web Services Security (WS-Security) Version 1.1 specification defines the use of signature confirmation. If you are using WS-Security Version 1.0, this function is not available.

   The signature confirmation value is stored in order to validate the signature confirmation with it after the receiving message is returned. This method is called if the response message is expected to attach the signature confirmation into the SOAP message.
7. The WSSVerifyPart API adds a digest method. For each part reference in the signing information, the API specifies both a digest method algorithm and a transform algorithm.

   WebSphere Application Server supports the following pre-configured digest algorithms:
   - WSSVerifyPart.SHA1: http://www.w3.org/2000/09/xmldsig#sha1
   - WSSVerifyPart.SHA256: http://www.w3.org/2001/04/xmlenc#sha256
   - WSSVerifyPart.SHA512: http://www.w3.org/2001/04/xmlenc#sha512
8. The WSSVerifyPart API adds a transform method. For each part reference in the signing information, the API specifies both a digest method algorithm and a transform algorithm.

   WebSphere Application Server supports the following pre-configured transform algorithms:
   - WSSVerifyPart.TRANSFORM_EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSVerifyPart.TRANSFORM_XPATH2_FILTER: http://www.w3.org/2002/06/xmldsig-filter2
   - WSSVerifyPart.TRANSFORM_STRT10: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - WSSVerifyPart.TRANSFORM_ENVELOPED_SIGNATURE: http://www.w3.org/2000/09/xmldsig#enveloped-signature

   For the WSS APIs, WebSphere Application Server does not support these algorithms:
   - http://www.w3.org/2002/07/decrypt#XML
   - http://www.w3.org/TR/1999/REC-xpath-19991116

   The transform algorithm for the consumer must match the transform algorithm for the generator.

Results

You have completed the steps to configure the signing information for the client-side response consumer sections of the bindings files.

Example

The following example shows WSS API sample code to verify the signature and to verify the X.509 token type as the security token:

```
WSSFactory factory = WSSFactory.getInstance();
WSSConsumingContext concont = factory.newWSSConsumingContext();
// Generate the X.509 Callback Handler on the consumer side
    X509ConsumeCallbackHandler callbackhandler = generateCallbackHandler();
    WSSVerification ver = factory.newWSSVerification(X509Token.class,
        callbackhandler);
concont.add(ver);
```

What to do next

If not already configured, specify a similar signing information configuration for the generator bindings.

Next, if already configured, configure the encryption and decryption information, or configure the consumer and generator tokens.

Verifying the signature using the WSSVerification API:

You can secure the SOAP messages, without using policy sets for configuration, by using the Web services security APIs (WSS API). To verify the signing information for the consumer binding sections for the client side request, use the WSSVerification API. You must also specify which algorithm methods and which signature parts of the SOAP message are to be verified. The WSSVerification API is part of the com.ibm.websphere.wssecurity.wssapi.verification package.

Before you begin

Use the WSS APIs, or configure the policy sets by using the administrative console to verify the signing information. To secure SOAP messages, you must complete the following signature tasks:

- Configure the signature information.
- Choose the algorithm methods for signature and signature verification.
- Verify the signature information.

About this task

WebSphere Application Server uses the signing information for the default generator to sign parts of the message, and uses XML digital signature with existing algorithms such as RSA-SHA1 and HMAC-SHA1.

XML signature defines many methods for describing key information and enables the definition of a new method. XML canonicalization (C14N) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. The C14N process is used to canonicalize XML information. Select an appropriate C14N algorithm because the information that is canonicalized depends on this algorithm.

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined.

Table 38. Signature verification parts

Verification parts	Description
keywords	Adds required signature parts as targets of verification by using keywords . Different message parts can be specified in the message protection for request on the generator side. Use the following keywords for the required signature verification parts: • ADDRESSING_HEADERS • BODY • TIMESTAMP The WS-Addressing headers are not encrypted but can be signed.
xpath	Adds verification parts by using an XPath expression.
part	Adds the WSSVerifyPart object as a verification part.
header	Adds the header, specified by QName, as a verification part.

For signature verification information, certain default behaviors occur. The simplest way to use the WSSVerification API is to use the default behavior.

The default values are defined by the WSS API for the digest method, the transform method, the security token, and the required verification parts.

Table 39. Signature verification default behaviors

Signature verification decisions	Default behavior
Which signature method to use (algorithm)	Sets the signature algorithm method. Both the data encryption and the signature and the canonicalization can be specified. The default signature method is RSA SHA1. WebSphere Application Server supports the following pre-configured signature methods: • WSSVerification.RSA_SHA1: http://www.w3.org/2000/09/xmldsig#rsa-sha1 • WSSVerification.HMAC_SHA1: http://www.w3.org/2000/09/xmldsig#hmac-sha1 The DSA-SHA1 digital signature method (http://www.w3.org/2000/09/xmldsig#dsa-sha1) is not supported.
Which canonicalization method to use (algorithm)	Sets the canonicalization algorithm method. Both the data encryption and the signature and the canonicalization can be specified. The default signature method is EXC_C14N. WebSphere Application Server supports the following pre-configured canonicalization methods: • WSSVerification.EXC_C14N: http://www.w3.org/2001/10/xml-exc-c14n# • WSSVerification.C14N: http://www.w3.org/2001/10/xml-c14n#
Whether signature confirmation is required	If the WSSSignature API specifies that signature confirmation is required, then the WSSVerification API verifies the signature confirmation value in the response message that has the signature confirmation value attached to it when received. Signature confirmation is defined in the OASIS Web Services Security Version 1.1 specification. The default signature confirmation is false.

Table 39. Signature verification default behaviors (continued)

Signature verification decisions	Default behavior
Which security token to specify (securityToken)	Adds the securityToken object as a signature part. WebSphere Application Server sets the token information to use for verification. WebSphere Application Server supports the following pre-configured tokens for signing: • X.509 Token • Derived Key Token Information required for tokens include the class for the token, the callback handler information, and the name of the JAAS login module.

1. To verify the signature in a SOAP message by using the WSSVerification API, first ensure that the application server is installed.
2. Use the WSSVerification API to set the message parts to be verified and to specify the algorithms in a SOAP message. The WSS API process for signature verification follows these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSConsumingContext instance from the WSSFactory instance.
   c. Ensures that WSSConsumingContext is called in the JAX-WS Provider implementation class. Due to the nature of the JAX-WS programming model, a JAX-WS provider needs to be implemented and must call the WSSConsumingContext to verify the SOAP message signature.
   d. Creates WSSVerification from the WSSFactory instance.
   e. Adds the part to be verified. If the digest method or the transform method are changed, create WSSVerifyPart and set it into WSSVerification.
   f. Sets the candidates of the canonicalization method, if the default is not appropriate.
   g. Sets the candidates of the signature method, if the default is not appropriate.
   h. Sets the candidate security token, if the default is not appropriate.
   i. Calls the requireSignatureConfirmation(), if the signature confirmation is applied.
   j. Adds WSSVerification to WSSConsumingContext.
   k. Calls WSSConsumingContext.process() with the SOAP message context.

Results

You have completed the steps to verify the signature for the consumer section of the bindings. If there is an error condition, a WSSException is provided. If successful, the WSSConsumingContext.process() is called, and Web Services Security is applied to the SOAP message.

Example

The following example provides sample code that uses methods that are defined in the WSSVerification API:

```
// Get the message context
  Object msgcontext = getMessageContext();

// Generate the WSSFactory instance (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance (step: b)
   WSSConsumingContext concont = factory.newWSSConsumingContext();


// Generate the certificate list
  String certpath = "c:/WebSphere/AppServer/etc/ws-security/samples/intca2.cer";
// The location of the X509 certificate file
  X509Certificate x509cert = null;
```

```
     try {
      InputStream is = new FileInputStream(certpath);
      CertificateFactory cf = CertificateFactory.getInstance("X.509");
      x509cert = (X509Certificate)cf.generateCertificate(is);
       } catch(FileNotFoundException e1){
        throw new WSSException(e1);
       } catch (CertificateException e2) {
        throw new WSSException(e2);
       }
      Set<Object> eeCerts = new HashSet<Object>();
       eeCerts.add(x509cert);
// Create the certificate store
   java.util.List<CertStore> certList = new java.util.ArrayList<CertStore>();
   CollectionCertStoreParameters certparam = new CollectionCertStoreParameters(eeCerts);
   CertStore cert = null;
   try {
    cert = CertStore.getInstance("Collection", certparam, "IBMCertPath");
   } catch (NoSuchProviderException e1) {
    throw new WSSException(e1);
   } catch (InvalidAlgorithmParameterException e2) {
    throw new WSSException(e2);
   } catch (NoSuchAlgorithmException e3) {
    throw new WSSException (e3);
   }
   if(certList != null ){
    certList.add(cert);
   }
// Generate the callback handler
   X509ConsumeCallbackHandler callbackHandler = new X509ConsumeCallbackHandler(
     "dsig-receiver.ks",
     "jks",
     "server".toCharArray(),
     certList,
     java.security.Security.getProvider("IBMCertPath")
   );

// Generate the WSSVerification instance (step: d)
  WSSVerification ver = factory.newWSSVerification(X509Token.class, callbackHandler);

// Set the part to be verified (step: e)
// DEFAULT: WSSVerification.BODY, WSSSignature.ADDRESSING_HEADERS,
// and WSSSignature.TIMESTAMP.

// Set the part in the SOAP header to be specified by QName (step: e)
     ver.addRequiredVerifyHeader(new QName("http://www.w3.org/2005/08/addressing", "MessageID"));

// Set the part to be specified by the keyword (step: e)
     ver.addRequiredVerifyPart(WSSVerification.BODY);

// Set the part to be specified by WSSVerifyPart (step: e)
  WSSVerifyPart verPart = factory.newWSSVerifyPart();
     verPart.setRequiredVerifyPart(WSSVerification.BODY);
     verPart.addAllowedDigestMethod(WSSVerifyPart.SHA256);
     ver.addRequiredVerifyPart(verPart);

// Set the part specified by XPath expression (step: e)
  StringBuffer sb = new StringBuffer();
        sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
            and local-name()='Envelope']");
        sb.append("/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
            and local-name()='Body']");
        sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
            and local-name()='Ping']");
        sb.append("/*[namespace-uri()='http://xmlsoap.org/Ping'
            and local-name()='Text']");
     ver.addRequiredVerifyPartByXPath(sb.toString());


// Set one or more canonicalization method candidates for verification (step: f)
// DEFAULT : WSSVerification.EXC_C14N
    ver.addAllowedCanonicalizationMethod(WSSVerification.C14N);
    ver.addAllowedCanonicalizationMethod(WSSVerification.EXC_C14N);
```

```
// Set one or more signature method candidates for verification (step: g)
// DEFAULT : WSSVerification.RSA_SHA1
    ver.addAllowedSignatureMethod(WSSVerification.HMAC_SHA1);

// Set the candidate security token to used for the verification (step: h)
  X509ConsumeCallbackHandler callbackHandler2 = getCallbackHandler2();
    ver.addToken(X509Token.class, callbackHandler2);


// Set the flag to require the signature confirmation (step: i)
   ver.requireSignatureConfirmation();

// Add the WSSVerification to the WSSConsumingContext (step: j)
  concont.add(ver);

//Validate the WS-Security header (step: k)
concont.process(msgcontext);
```

What to do next

After verifying the signature and setting algorithm methods for the SOAP message, you can set either the digest method or the transform method. If you want to set these methods, use the WSSVerifyPart API, or configure policy sets using the administrative console.

Verifying the signed parts using the WSSVerifyPart API:

To secure SOAP messages on the consumer side, use the Web Services Security APIs (WSS API) to configure the verify parts information for the consumer binding on the response consumer (client side). You can specify which algorithm methods and which parts of the SOAP message are to be verified. Use the WSSVerifyPart API to change the digest method or the transform method. The WSSVerifyPart API is part of the com.ibm.websphere.wssecurity.wssapi.verification package.

Before you begin

To secure SOAP messages using the signing verification information, you must complete one of the following tasks:

- Configure the signature verification information using the WSSVerification API.
- Configure verify parts using the WSSVerifyPart API, as needed.

The WSSVerifyPart is used for specify the transform or digest methods for the verification. Use the WSSVerifyPart API or configure policy sets using the administrative console.

About this task

WebSphere Application Server uses the signing information for the default consumer to verify the signed parts of the message. The WSSVerifyPart API is only supported on the response consumer (requester).

The following table shows the required verification parts when the digital signature security constraint (integrity) is defined:

Table 40. Verify parts information

Verify parts information	Description
keyword	Sets the verify parts using the following keywords: • BODY • ADDRESSING_HEADERS • TIMESTAMP The WS-Addressing headers are not decrypted but can be signed and verified.
xpath	Sets the verify parts using an XPath expression.
header	Sets the header, specified by QName, as a required verify part.

For signature verification, certain default behaviors occur. The simplest way to use the WSSVerification API is to use the default behavior (see the example code). The default values are defined by the WSS API for the signing algorithm and the canonicalization algorithm, and the verify parts.

Table 41. Verify parts default behaviors

Verify parts decisions	Default behavior
Which keywords to specify	The different SOAP message parts to be signed and used for message protection. WebSphere Application Server supports the following keywords: • WSSVerification.BODY • WSSVerification.ADDRESSING_HEADERS • WSSVerification.TIMESTAMP
Which transform method to use (algorithm)	Adds the transform method. The transform algorithm is specified within the <Transform> element and specifies the transform algorithm for the signature. The default transform method is TRANSFORM_EXC_C14N. WebSphere Application Server supports the following pre-configured transform algorithms: • WSSVerifyPart.TRANSFORM_EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n# • WSSVerifyPart.TRANSFORM_XPATH2_FILTER: http://www.w3.org/2002/06/xmldsig-filter2 Use this transform method to ensure compliance with the Basic Security Profile (BSP). • WSSVerifyPart.TRANSFORM_STRT10: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform • WSSVerifyPart.TRANSFORM_ENVELOPED_SIGNATURE: http://www.w3.org/2000/09/xmldsig#enveloped-signature
Which digest method to use (algorithm)	Sets the digest algorithm method. The digest method algorithm that is specified within the <DigestMethod> element is used in the <SigningInfo> element. The default digest method is SHA1. WebSphere Application Server supports the following digest method algorithms: • WSSVerifyPart.SHA1: http://www.w3.org/2000/09/xmldsig#sha1 • WSSVerifyPart.SHA256: http://www.w3.org/2001/04/xmlenc#sha256 • WSSVerifyPart.SHA512: http://www.w3.org/2001/04/xmlenc#sha512

1. To verify signed parts by using the WSSVerifyPart API, first ensure that the application server is installed.

2. Use the Web Services Security API to verify the verification in a SOAP message. The WSS API process for verifying the signature follows these process steps:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSConsumingContext instance from the WSSFactory instance. Ensures that WSSConsumingContext is called in the JAX-WS Provider implementation class. Due to the nature of the JAX-WS programming model, a JAX-WS provider needs to be implemented and must call the WSSConsumingContext to verify the SOAP message signature.
   c. Creates the CallbackHandler to use for verification.
   d. Create the WSSVerification object from the WSSFactory instance.
   e. Creates WSSVerifyPart from the WSSFactory instance.
   f. Sets the part to be verified, if the default is not appropriate.
   g. Sets the candidates for the digest method, if the default is not appropriate.
   h. Sets the candidates for the transform method, if the default is not appropriate.
   i. Adds WSSVerifyPart to WSSVerification.
   j. Adds WSSVerification to WSSConsumingContext.
   k. Calls WSSConsumingContext.process() with the SOAPMessageContext.

Results

You have completed the steps to verify to verify the signed parts on the consumer side. If there is an error condition when verifying the signing information, a WSSException is provided. If successful, the WSSConsumingContext.process() is called, and Web Services Security is verified for the SOAP message.

Example

The following example provides sample code for the WSSVerification API process for verifying the signing information in a SOAP message:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance (step: b)
   WSSConsumingContext concont = factory.newWSSConsumingContext();

// Generate the certificate list
   String certpath =
   "c:/WebSphere/AppServer/etc/ws-security/samples/intca2.cer";
// The location of the X509 certificate file
   X509Certificate x509cert = null;
      try {
          InputStream is = new FileInputStream(certpath);
          CertificateFactory cf = CertificateFactory.getInstance("X.509");
          x509cert = (X509Certificate)cf.generateCertificate(is);
      } catch(FileNotFoundException e1){
            throw new WSSException(e1);
      } catch (CertificateException e2) {
            throw new WSSException(e2);
      }

      Set<Object> eeCerts = new HashSet<Object>();
      eeCerts.add(x509cert);
// create certStore
      java.util.List<CertStore> certList = new
          java.util.ArrayList<CertStore>();
      CollectionCertStoreParameters certparam = new
          CollectionCertStoreParameters(eeCerts);
      CertStore cert = null;
      try {
```

```
         cert = CertStore.getInstance("Collection",
              certparam, "IBMCertPath");
      } catch (NoSuchProviderException e1) {
           throw new WSSException(e1);
      } catch (InvalidAlgorithmParameterException e2) {
           throw new WSSException(e2);
      } catch (NoSuchAlgorithmException e3) {
           throw new WSSException (e3);
      }
      if(certList != null ){
      certList.add(cert);
      }

// generate callback handler (step: c)
   X509ConsumeCallbackHandler callbackHandler = new
       X509ConsumeCallbackHandler(
                                  "dsig-receiver.ks",
                                  "jks",
                                  "server".toCharArray(),
                                  certList,
       java.security.Security.getProvider("IBMCertPath")
);

// Generate the WSSVerification instance (step: d)
   WSSVerification ver = factory.newWSSVerification(X509Token.class,
       callbackHandler);

// Set the part to be specified by WSSVerifyPart (step: e)
   WSSVerifyPart verPart = factory.newWSSVerifyPart();

// Set the part to be specified by the keyword (step: f)
   verPart.setRequiredVerifyPart(WSSVerification.BODY);

// Set the candidates for the digest method for verification (step: g)
// DEFAULT : WSSVerifyPart.SHA1
   verPart.addAllowedDigestMethod(WSSVerifyPart.SHA256);

// Set the candidates for the transform method for verification (step: h)
// DEFAULT : WSSVerifypart.TRANSFORM_EXC_C14N : String
   verPart.addAllowedTransform(WSSVerifyPart.TRANSFORM_STRT10);

// Set WSSVerifyPart to WSSVerification (step: i)
   ver.addRequiredVerifyPart(verPart);

// Add WSSVerification to WSSConsumingContext (step: j)
   concont.add(ver);

//Validate the WS-Security header (step: k)
concont.process(msgcontext);
```

What to do next

You have completed configuring the signed part to be verified.

Configuring the client for response signature verification methods:

Use the WSSVerification and WSSVerifyPart APIs to choose the signing verification methods. The request signing verification methods include the digest algorithm and the transport methods.

Before you begin

To complete configuration of the signature verification information to secure SOAP messages, you must perform the following algorithm tasks:

- Use the WSSVerification API to configure the canonicalization and signature methods.
- Use the WSSVerifyPart API to configure the digest and transform methods.

to configure the algorithm methods to use when configuring the client for request signing.

About this task

The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web site http://www.w3.org/TR/xmldsig-core.

Name of method	Purpose
Digest algorithm	Applies to the data after transforms are applied, if specified, to yield the <DigestValue> element. Signing the <DigestValue> element binds the resource content to the signer key. The algorithm selected for the client request sender configuration must match the algorithm selected in the client request receiver configuration.
Transform algorithm	Applies to the <Transform> element.
Signature algorithm	Specifies the Uniform Resource Identifiers (URI) of the signature verification method.
Canonicalization algorithm	Specifies the Uniform Resource Identifiers (URI) of the canonicalization method.

After configuring the client to digitally sign the message, you must configure the client to verify the digital signature. You can use the WSS APIs or configure policy sets using the administrative console to verify the digital signature and to choose the verification and verify part algorithms. If using the WSS APIs to configure, use the WSSVerification and WSSVerifyPart APIs to specify which digitally signed message parts to verify and to specify which algorithm methods to use when configuring the client for request signing.

The WSSVerification and WSSVerifyPart APIs perform the following steps to configure the signature verification and verify parts algorithm methods:

1. For the consumer binding, the WSSVerification API specifies the signature methods to allow for the signature verification. WebSphere Application Server supports the following pre-configured signature methods:
   - WSSVerification.RSA_SHA1 (the default value): http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - WSSVerification.HMAC_SHA1: http://www.w3.org/2000/09/xmldsig#hmac-sha1

   The DSA-SHA1 digital signature method (http://www.w3.org/2000/09/xmldsig#dsa-sha1) is not supported.
2. For the consumer binding, the WSSVerification API specifies the canonicalization method to allow for the signature verification. WebSphere Application Server supports the following pre-configured canonicalization methods by default:
   - WSSVerification.EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSVerification.C14N: http://www.w3.org/2001/10/xml-c14n#
3. For the consumer binding, the WSSVerifyPart API specifies the digest method, as needed. WebSphere Application Server supports the following digest method algorithms for signed parts verification:
   - WSSVerifyPart.SHA1 (the default value): http://www.w3.org/2000/09/xmldsig#sha1
   - WSSVerifyPart.SHA256: http://www.w3.org/2001/04/xmlenc#sha256
   - WSSVerifyPart.SHA512: http://www.w3.org/2001/04/xmlenc#sha512
4. For the consumer binding, the WSSVerifyPart API specifies the transform method. WebSphere Application Server supports the following transform algorithms for verify parts:
   - WSSVerifyPart.TRANSFORM_EXC_C14N (the default value): http://www.w3.org/2001/10/xml-exc-c14n#
   - WSSVerifyPart.TRANSFORM_XPATH2_FILTER: http://www.w3.org/2002/06/xmldsig-filter2
   - WSSVerifyPart.TRANSFORM_STRT10: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform

- WSSVerifyPart.TRANSFORM_ENVELOPED_SIGNATURE: http://www.w3.org/2000/09/xmldsig#enveloped-signature

For the WSS APIs, WebSphere Application Server does not support these algorithms:

- http://www.w3.org/2002/07/decrypt#XML
- http://www.w3.org/TR/1999/REC-xpath-19991116

Results

You have specified which method to use when verifying a digital signature when the client sends a message.

Example

The following example provides sample WSS API code that specifies the verification information, the body as a part to be verified, the HMAC_SHA1 as a signature method, C14N and EXC_C14N as the candidates of canonicalization methods, TRANSFORM_STRT10 as a transform method, and SHA256 as a digest method.

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance
   WSSConsumingContext concont = factory.newWSSConsumingContext();

// Generate the certificate list
   String certpath = "intca2.cer";
// The location of the X509 certificate file
      X509Certificate x509cert = null;
      try {
            InputStream is = new FileInputStream(certpath);
            CertificateFactory cf = CertificateFactory.getInstance("X.509");
           x509cert = (X509Certificate)cf.generateCertificate(is);
      } catch(FileNotFoundException e1){
          throw new WSSException(e1);
      } catch (CertificateException e2) {
          throw new WSSException(e2);
      }

      Set<Object> eeCerts = new HashSet<Object>();
      eeCerts.add(x509cert);
// Create the certStore
   java.util.List<CertStore> certList = new
        java.util.ArrayList<CertStore>();
   CollectionCertStoreParameters certparam = new
        CollectionCertStoreParameters(eeCerts);
   CertStore cert = null;
      try {
            cert = CertStore.getInstance("Collection",
                                         certparam,
                                        "IBMCertPath");
      } catch (NoSuchProviderException e1) {
           throw new WSSException(e1);
      } catch (InvalidAlgorithmParameterException e2) {
           throw new WSSException(e2);
      } catch (NoSuchAlgorithmException e3) {
           throw new WSSException (e3);
      }
      if(certList != null ){
            certList.add(cert);
      }

// Generate the callback handler
   X509ConsumeCallbackHandler callbackHandler = new
      X509ConsumeCallbackHandler(
```

```
                                "dsig-receiver.ks",
                                "jks",
                                "server".toCharArray(),
                                certList,
                                java.security.Security.getProvider(
                                      "IBMCertPath")
                                );

// Generate the WSSVerification instance
   WSSVerification ver = factory.newWSSVerification(X509Token.class,
                                                    callbackHandler);

// Set one or more candidates of the signature method used for
//    verification (step. 1)
// DEFAULT : WSSVerification.RSA_SHA1
      ver.addAllowedSignatureMethod(WSSVerification.HMAC_SHA1);

// Set one or more candidates of the canonicalization method used for
//    verification (step. 2)
// DEFAULT : WSSVerification.EXC_C14N
      ver.addAllowedCanonicalizationMethod(WSSVerification.C14N);
      ver.addAllowedCanonicalizationMethod(WSSVerification.EXC_C14N);

// Set the part to be specified by WSSVerifyPart
   WSSVerifyPart verPart = factory.newWSSVerifyPart();

// Set the part to be specified by the keyword
      verPart.setRequiredVerifyPart(WSSVerification.BODY);

// Set the candidates of digest methods to use for verification (step. 3)
// DEFAULT : WSSVerifypart.TRANSFORM_EXC_C14N : String
      verPart.addAllowedTransform(WSSVerifyPart.TRANSFORM_STRT10);

// Set the candidates of digest methods to use for verification (step. 4)
// DEFAULT : WSSVerifyPart.SHA1
      verPart.addAllowedDigestMethod(WSSVerifyPart.SHA256);

// Set WSSVerifyPart to WSSVerification
      ver.addRequiredVerifyPart(verPart);

// Add the WSSVerification to the WSSConsumingContext
   concont.add(ver);

// Validate the WS-Security header
concont.process(msgcontext);
```

What to do next

You have completed configuring the signature verification algorithms. Next, configure the encryption or decryption algorithms, if not already configured. Or, configure the security token information, as needed.

Signature verification methods using the WSSVerification API:

You can verify the signing or signature information using the WSS API for the consumer binding. The signature and canonicalization algorithm methods are used for the generator binding. The WSSVerification API is provided in the com.ibm.websphere.wssecurity.wssapi.verification package.

To configure consumer signing information to protect message integrity, you must first digitally sign and then verify the signature for the SOAP messages. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network.

Methods

Methods that are used for the signature verification include the:

Signature method
Sets the signature algorithm method.

Canonicalization method
Sets the canonicalization algorithm method.

The algorithm that is specified for the request generator configuration must match the algorithm that is specified for the response consumer configuration.

Signature algorithms

The signature algorithms specify the signature verification algorithm that is used to sign the certificate. The signature algorithms specify the Uniform Resource Identifiers (URI) of the signature verification method. WebSphere Application Server supports the following pre-configured algorithms:

Table 42. Signature verification algorithms

Algorithm	Description
WSSVerification.HMAC_SHA1	A URI of the signature algorithm, HMAC: http://www.w3.org/2000/09/xmldsig#hmac-sha1
WSSVerification.RSA_SHA1 (the default value)	A URI of the signature algorithm, RSA: http://www.w3.org/2000/09/xmldsig#rsa-sha1

WebSphere Application Server does not support the algorithm for DSA-SHA1: http://www.w3.org/2000/09/xmldsig#dsa-sha1

Canonicalization algorithms

The canonicalization algorithms specify the Uniform Resource Identifiers (URI) of the canonicalization method. WebSphere Application Server supports the following pre-configured algorithms:

Table 43. Verification canonicalization algorithms

Algorithm	Description
WSSVerification.C14N	A URI of the inclusive canonicalization algorithm, C14N: http://www.w3.org/2001/10/xml-c14n#
WSSVerification.EXC_C14N (the default value)	A URI of the exclusive canonicalization algorithm EXC_C14N: http://www.w3.org/2001/10/xml-exc-c14n#

The following example provides sample WSS API code that specifies the X.509 token security token for signature verification:

```
WSSFactory factory = WSSFactory.getInstance();
WSSConsumingContext concont = factory.newWSSConsumingContext();

// X509ConsumeCallbackHandler
X509ConsumeCallbackHandler callbackHandler = new
   X509ConsumeCallbackHandler("dsig-receiver.ks",
                              "jks",
                              "server".toCharArray(),
                              certList,
   java.security.Security.getProvider("IBMCertPath")46 );

// Set the verification component
// DEFAULT verification parts: Body, WS-Addressing header, and Timestamp
// DEFAULT data encryption algorithm: RSA-SHA1
// DEFAULT digest algorithm: SHA1
// DEFAULT canonicalization algorithm: exc-c14n
WSSVerification ver = factory.newWSSVerification(X509Token.class,
                                                 callbackhandler);
```

```
concont.add(ver);

// Validate the WS-Security header
concont.validate(msgctx);
```

Choosing the verify parts methods using the WSSVerifyPart API:

You can configure the signing verification information for the consumer binding using the WSS API. The transform algorithm and digest methods are used for the consumer binding. Use the WSSVerifyPart API to configure the algorithm methods. The WSSVerifyPart API is provided in the com.ibm.websphere.wssecurity.wssapi.verification package.

To configure consumer verify parts information to protect message integrity, you must first digitally sign and then verify the signature and signed parts for the SOAP messages. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network.

Methods

Methods that are used for the signing information include the:

Digest method
Sets the digest method.

Transform method
Sets the transform algorithm method.

Digest algorithms

The digest method algorithm is specified within the element is used in the <Digest> element. WebSphere Application Server supports the following pre-configured digest algorithms:

Table 44. Verify parts digest methods

Digest method	Description
WSSVerifyPart.SHA1 (the default value)	A URI of the digest algorithm, SHA1: http://www.w3.org/2000/09/xmldsig#sha1
WSSVerifyPart.SHA256	A URI of the digest algorithm, SHA256: http://www.w3.org/2001/04/xmlenc#sha256
WSSVerifyPart.SHA512	A URI of the digest algorithm, SHA256: http://www.w3.org/2001/04/xmlenc#sha512

Transform algorithms

The transform algorithm is specified within the <Transform> element and specifies the transform algorithm for the signed part. WebSphere Application Server supports the following pre-configured transform algorithms:

Table 45. Verify parts transform methods

Digest method	Description
WSSVerifyPart.TRANSFORM_ENVELOPED_SIGNATURE	A URI of the transform algorithm, enveloped signature: http://www.w3.org/2000/09/xmldsig#enveloped-signature
WSSVerifyPart.TRANSFORM_STRT10	A URI of the transform algorithm, STR-Transform: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform

Table 45. Verify parts transform methods (continued)

Digest method	Description
WSSVerifyPart.TRANSFORM_EXC_C14N (the default value)	A URI of the transform algorithm, Exc-C14N: http://www.w3.org/2001/10/xml-exc-c14n#
WSSVerifyPart.TRANSFORM_XPATH2_FILTER	A URI of the transform algorithm, XPath2 filter: http://www.w3.org/2002/06/xmldsig-filter2

For the WSS APIs, WebSphere Application Server does not support the following transform algorithms:

- http://www.w3.org/TR/1999/REC-xpath-19991116
- http://www.w3.org/2002/07/decrypt#XML

The following example provides sample WSS API code that verifies the body using SHA256 as the digest method and TRANSFORM_EXC_14N and TRANSFORM_STRT10 as the transform methods:

```
// get the message context
Object msgcontext = getMessageContext();

// generate WSSFactory instance
WSSFactory factory = WSSFactory.getInstance();

// generate WSSConsumingContext instance
WSSConsumingContext concont = factory.newWSSConsumingContext();


// generate the cert list
String certpath = "intca2.cer";// The location of the X509
   certificate file X509Certificate x509cert = null;
try {
 InputStream is = new FileInputStream(certpath);
 CertificateFactory cf = CertificateFactory.getInstance("X.509");
 x509cert = (X509Certificate)cf.generateCertificate(is);
} catch(FileNotFoundException e1){
 throw new WSSException(e1);
} catch (CertificateException e2) {
 throw new WSSException(e2);
}

Set<Object> eeCerts = new HashSet<Object>();
eeCerts.add(x509cert);
// create certStore
java.util.List<CertStore> certList = new java.util.ArrayList<CertStore>();
CollectionCertStoreParameters certparam = new
      CollectionCertStoreParameters(eeCerts);
CertStore cert = null;
try {
 cert = CertStore.getInstance("Collection", certparam, "IBMCertPath");
} catch (NoSuchProviderException e1) {
 throw new WSSException(e1);
} catch (InvalidAlgorithmParameterException e2) {
 throw new WSSException(e2);
} catch (NoSuchAlgorithmException e3) {
 throw new WSSException (e3);
}
if(certList != null ){
 certList.add(cert);
}

// generate callback handler
X509ConsumeCallbackHandler callbackHandler = new
      X509ConsumeCallbackHandler(
  "dsig-receiver.ks",
  "jks",
  "server".toCharArray(),
  certList,
  java.security.Security.getProvider("IBMCertPath")
);
```

```
//generate WSSVerification instance
WSSVerification ver = factory.newWSSVerification(X509Token.class,
     callbackHandler);

//set one or more candidates of the signature method used for the
 //verification (step. 1)
// DEFAULT : WSSVerification.RSA_SHA1
ver.addAllowedSignatureMethod(WSSVerification.HMAC_SHA1);

//set one or more candidates of the canonicalization method used
 //for the verification (step. 2)
// DEFAULT : WSSVerification.EXC_C14N
ver.addAllowedCanonicalizationMethod(WSSVerification.C14N);
ver.addAllowedCanonicalizationMethod(WSSVerification.EXC_C14N);

//set the part to be specified by WSSVerifyPart
WSSVerifyPart verPart = factory.newWSSVerifyPart();

//set the part to be specified by the keyword
verPart.setRequiredVerifyPart(WSSVerification.BODY);

//set the candidates of digest methods to use for verification (step. 3)
// DEFAULT : WSSVerifypart.TRANSFORM_EXC_C14N
verPart.addAllowedTransform(WSSVerifyPart.TRANSFORM_EXC_C14N);
verPart.addAllowedTransform(WSSVerifyPart.TRANSFORM_STRT10);

//set the candidates of digest methods to use for verification (step. 4)
// DEFAULT : WSSVerifyPart.SHA1
verPart.addAllowedDigestMethod(WSSVerifyPart.SHA256);

//set WSSVerifyPart to WSSVerification
ver.addRequiredVerifyPart(verPart);

//add the WSSVerification to the WSSConsumingContext
concont.add(ver);

//validate the WS-Security header
concont.process(msgcontext);
```

Validating the consumer token to protect message authenticity:

The token consumer information is used on the consumer side to incorporate and validate the security token. The Username token, X509 tokens, and LTPA tokens by default are used for message authenticity.

Before you begin

The token processing and pluggable token architecture in the Web Service Security run time reuses the same security token interface and Java Authentication and Authorization Service (JAAS) Login Module from the Web Services Security APIs (WSS API). The same implementation of token creation and validation can be used in both the WSS API and the WSS SPI in the Web Service Security run time.

Note: The com.ibm.wsspi.wssecurity.token.TokenConsumingComponent interface is not used with JAX-WS Web services. If you are using JAX-RPC Web services, this interface is still valid.

Note that the key name (KeyName) element is not supported because there is no KeyName policy assertion defined in the current OASIS Web Services Security draft specification. For similar reasons, a SAML token is not supported.

About this task

The JAAS callback handler (CallbackHandler) and the JAAS login module (LoginModule) are responsible for creating the security token on the generator side and validating (authenticating) the security token on the consumer side.

For example, on the generator side, the Username token is created by the JAAS LoginModule and using the JAAS CallbackHandler to pass the authentication data. The JAAS LoginModule creates the Username SecurityToken object and passes it to the Web services security run time.

Then, on the consumer side, the Username Token XML format is passed to the JAAS LoginModule for validation or authentication and the JAAS CallbackHandler is used to pass authentication data from the Web services security run time to the LoginModule. After the token is authenticated, a Username SecurityToken object is created and passed it to the Web Service Security run time.

Note: WebSphere Application Server does not support a stackable login module with the WebSphere Application Server default login module implementation, meaning adding the login module before or after the WebSphere Application Server login module implementation. If you want to stack the login module implementations, you must develop the required login modules because there is no default implementation.

The com.ibm.websphere.wssecurity.wssapi.token package provided by WebSphere Application Server includes support for these classes:

- Security token (SecurityTokenImpl)
- Binary security token (BinarySecurityTokenImpl)

In addition, WebSphere Application Server provides the following pre-configured sub-interfaces for security tokens:

- Derived key token
- Security context token (SCT)
- Username token
- LTPA token propagation
- LTPA token
- X509PKCS7 token
- X509PKIPath token
- X509v3 token
- Kerberos v5 token

The Username token, the X.509 tokens, and the LTPA tokens are used by default for message authenticity. The derived key token and the X.509 tokens are used by default for signing and encryption.

The WSS API and WSS SPI are only supported on the client. To specify the security token type on the consumer side, you can also configure policy sets using the administrative console. You can also use the WSS APIs or policy sets for matching generator security tokens.

The default Login Module and Callback implementations are designed to be used as a pair, meaning both a generator and a consumer part. To use the default implementations, select the appropriate generator and consumer security token in a pair. For example, select `system.wss.generate.x509` in the token generator and `system.wss.consume.x509` in the token consumer when the X.509 token is required.

To configure the consumer-side security token, use the appropriate pre-configured token consumer interface from the WSS APIs to complete the following token configuration process steps:

1. Generate the wssFactory instance.
2. Generate the wssConsumingContext instance.

   The WSSConsumingContext interface stores the components for consuming Web Services Security (WS-Security), such as verification, decryption, the security token, and the time stamp. When the validate() method is called, all of these components are validated.
3. Create the consumer-side components, such as the WSSVerification and the WSSDecryption objects.

4. Specify a JAAS configuration by specifying the name of the JAAS login configuration. The Java Authentication and Authorization Service (JAAS) configuration specifies the name of the JAAS configuration. The JAAS configuration specifies how the token logs in on the consumer side. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module.
5. Specify a token consumer class name. The token consumer class name specifies the required information to validate the SecurityToken. The Username token, the X.509 tokens, and the LTPA tokens are used by default for message authenticity.
6. Specify the settings for the callback handler by specifying a callback handler class name and also specifies the callback handler keys. This class name is the name of the callback handler implementation class that is used for the plug-in to the security token framework.

   WebSphere Application Server provides the following default callback handler implementations for the consumer side:

   com.ibm.websphere.wssecurity.callbackhandler.PropertyCallback
   This class is a callback for handling the name-value pair in elements in the Web Services Security (WS-Security) configuration XMI files.

   ccom.ibm.websphere.wssecurity.callbackhandler.UNTConsumeCallbackHandler
   This class is a callback handler for the Username token on the consumer side. This instance is used to set into WSSConsumingContext object to validate a Username token. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

   com.ibm.websphere.wssecurity.callbackhandler.X509ConsumeCallbackHandler
   This class is a callback handler that is used to validate the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token on the consumer side. This instance is used to generate the WSSVerification object and WSSDecryption objects, set the objects into WSSConsumingContext object to validate the X.509 binary security tokens. A keystore and a key definition are required for this callback handler. If you use this implementation, a key store password, path, and type must have been provided on the generator side.

   com.ibm.websphere.wssecurity.callbackhandler.LTPAConsumeCallbackHandler
   This class is a callback handler for the Lightweight Third Party Authentication (LTPA) tokens on the consumer side. This instance is used to generate the WSSVerification and WSSDecryption objects to validate an LTPA token.

   This callback handler is used to validate the LTPA security token inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a Java EE application client. If you use this implementation, a basic authentication user ID and password must have been provided on the generator side.

   com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler
   This class is a callback handler for the Kerberos v5 token on the consumer side. This instance is used to set the WSSConsumingContext object to consume the Kerberos v5 AP-REQ as a binary security token. The instance is also used to generate the WSSVerification and WSSDecryption objects to use the Kerberos session key or derived key in the SOAP message verification and decryption.
7. If a X.509 token is specified, additional token information is also specified.

keyStoreRef	The reference name of the keystore that is used for the key locator.

keyStorePath	The keystore file path from which the keystore is loaded, if needed. It is recommended that you use the `${USER_INSTALL_ROOT}` in the path name as this variable expands to the WebSphere Application Server path on your machine. This path is required when you use the X.509 tokens callback handler implementations.
keyStorePassword	The password that is used to check the integrity of the keystore, or the keystore password that is used to unlock the keystore and to access the keystore file. The keystore and its configuration are used for some of the default callback handler implementations that are provided by WebSphere Application Server.
keyStoreType	The keystore type of keystore that is used for the key locator. This selection indicates the format that is used by the keystore file. The following values are available for selection: **JKS** Use this option if the keystore uses the Java Keystore (JKS) format. **JCEKS** Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption. **JCERACFKS** Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only). **PKCS11KS (PKCS11)** Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might contain encrypt keys that use cryptographic hardware to ensure protection. **PKCS12KS (PKCS12)** Use this option if your keystore uses the PKCS#12 file format.
alias	The key alias name. The key alias is used by the key locator to find the key within the keystore file.
keyPassword	The key password that is used for recovering the key. This password is needed to access the key object within the keystore file.
keyName	The name of the key. For digital signatures, the key name is used by the request generator or response consumer signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption. The key name must be a fully qualified, distinguished name (DN). For example, `CN=Bob,O=IBM,C=US`.
trustAnchorPath	The file path from which the trust anchor is loaded.
trustAnchorType	The type of trust anchor.
trustAnchorPassword	The password that is used to check the integrity of the trust anchor or the password used to unlock the keystore.
certStores	A list of certificate stores. A collection certificate store includes a list of untrusted, intermediary certificates and certificate revocation lists (CRLs). The collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens.
provider	The security provider.

The following can be specified for a X.509 token:

a. Without any keystore.

b. With a trust anchor. A trust anchor specifies a list of keystore configurations that contain trusted root certificates. These configurations are used to validate the certificate path of incoming X.509-formatted security tokens. For example, when you select the trust anchor or the certificate store of a trusted certificate, you must configure the trust anchor and the certificate store before setting the certificate path.

c. With a keystore that is used for the key locator.

First, you must have created the keystore file, by using a key tool utility, for example. The keystore is used to retrieve the X.509 certificate. This entry specifies the password that is used to access the keystore file. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trustworthiness of a certificate chain. The names of the trust anchor and the collection certificate store are created in the certificate path under your token consumer.

d. With a keystore that is used for the key locator and the trust anchor.

e. With a map that includes key-value pairs. For example, you might specify the value type name and the value type Uniform Resource Identifier (URI). The value type specifies the namespace URI of the value type for the consumer token, and represents the token type of this class:

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

Specifies an X.509 certificate token.

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

Specifies X.509 certificates in a public key infrastructure (PKI) path. This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used. If you use this implementation, you must provide a key store password, path, and type on this panel.

ValueType: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Specifies a list of X.509 certificates and certificate revocation lists in a PKCS#7 format. This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You can specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. If you use this implementation, you must provide a key store password, path, and type.

For some tokens, WebSphere Application Server provides a predefined local name for the value type. When you specify the following local name, you do not need to specify a value type URI:

ValueType: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

For an LTPA token, you can use `LTPA` for the value type local name. This local name causes `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` to be specified for the value type Uniform Resource Identifier (URI).

ValueType: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

For LTPA token propagation, you can use `LTPA_PROPAGATION` for the value type local name. This local name causes`http://www.ibm.com/websphere/appserver/tokentype` to be specified for the value type Uniform Resource Identifier (URI).

8. If the Username token is specified as the token consumer class name, the following token information can be specified:

a. Whether to specify the nonce.

This option indicates whether a Nonce is included for the token consumer. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of Username tokens. Nonce is valid only when the validating token type is a Username token, and it is available only for the response consumer binding.

b. Specifies the keyword of the time stamp. This option indicates whether to verify a time stamp in the Username token. The time stamp is valid only when the incorporated token type is a Username token.

c. Specifies a map that includes key-value pairs. For example, you might specify the value type name and the value type Uniform Resource Identifier (URI). The value type specifies the namespace URI of the value type for the consumer token, and represents the token type of this class:

URI value type: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

Specifies a Username token.

9. If a Kerberos v5 token is specified as the token generator class name, the following token information can be specified:

Token Information	Description	Default Value
tokenValueType	Kerberos token value type in QName defined by Oasis Kerberos Token Profile v1.1 specification.	http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ
requireDKT	A boolean value to require a derived key for message protection.	false
clabel	The client label for the derived key.	WS-SecureConversation Specify null to use the default value.
slabel	The service label for the derived key.	WS-SecureConversation Specify null to use the default value.
keylen	The length of the derived key.	16 Specify zero to use the default value
supportTokenRequireSHA1	A boolean value to require a SHA1 key that is used in subsequent request messages when the Kerberos token is used as a supporting token.	false SHA1 key is consumed only if the supporting Kerberos token is protected. If set to true, the SHA1 key is always consumed.
decComponent	An instance of WSSDecryption .	Set decComponent and verComponent to null to initialize this first for either the decryption or verification component. Then, use the initialized component only in the callback handler constructor for the second component.
verComponent	An instance of WSSVerfication.	Set decComponent and verComponent to null to initialize this first for either the decryption or verification component. Then, use the initialized component only in the callback handler constructor for the second component.

Additional token value types are defined in the OASIS Kerberos Token Profile v1.1 specification. Specify the token value type as the local name. It is not necessary to specify the value type URI for the Kerberos v5 token.

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120

10. If secure conversation is used for message protection, then the following information must be specified:

Information	Description
EncryptionAlgorithm	This determines the key size.
cLabel	The client label used when creating the derived key.
sLabel	The server label used when creating the derived key.

11. Set the components into the wssConsumingContext object.
12. Invoke the wssConsumingContext.process() method.

Results

Using the WSS APIs, you have configured the token consumer.

What to do next

You must specify a similar token generator configuration, if not already completed.

Configuring the consumer security tokens using the WSS API:

You can secure the SOAP messages, without using policy sets, by using the Web Services Security APIs. To configure the token on the consumer side, use the Web Services Security APIs (WSS API). The consumer security tokens are part of the com.ibm.websphere.wssecurity.wssapi.token interface package.

Before you begin

The pluggable token framework in WebSphere Application Server has been redesigned so that the same framework from the WSS API can be reused. The same implementation of creating and validating security token can be used both for the Web Service Security run time and for the WSS API application code. The redesigned framework also simplifies the SPI programming model and will make it easier to add security token types.

You can use the WSS API or you can configure the tokens by using the administrative console. To configure tokens, you must have completed the following token task: configure the generator tokens, as needed.

About this task

On the generator side, the JAAS CallbackHandler and JAAS LoginModule are responsible for creating the security token. The token is created by using the JAAS LoginModule and by using JAAS CallbackHandler to pass authentication data. Then, the JAAS LoginModule creates the securityToken object, such as the UsernameToken, and passes it to the Web Service Security run time.

On the consumer side, the XML format is passed to the JAAS LoginModule for validation or authentication. then the JAAS CallbackHandler is used to pass authentication data from the Web Service Security run time to the LoginModule. After the token is authenticated and a security token object is created, then the token is passed it to the Web Service Security run time.

When using the WSS API for consumer token validation, certain default behaviors occur. The simplest way to use the WSS API is to use the default JAAS login module and callback handler. The example uses the default for them so the example does not specify the JAAS login module name.

The simplest way to use the WSS API is to use the default behavior (see the example code). The WSS API provide defaults for the token type, the token value, and the JAAS configuration name. The default token behaviors include:

Table 46. Default token behaviors

Consumer token decisions	Default behavior
Which token type to use	The token type specifies which type of token to use for signing and validating messages. The X.509 token is the default token type. WebSphere Application Server provides the following pre-configured consumer token types: • Security context token • Derived key token • X509 tokens You can also create custom token types, as needed.
What JAAS login configuration name to specify	The JAAS login configuration name specifies which JAAS login configuration name to use.
Which configuration type to use	The JAAS login module configuration type. Only the pre-configured consumer configuration types can be used for consumer token types.

The SecurityToken class (com.ibm.websphere.wssecurity.wssapi.token.SecurityToken) is the generic token class and represents the security token that has methods to get the identity, XML format, and cryptographic keys. Using the SecurityToken class, you can apply both the signature and encryption to the SOAP message. However, to apply both, you must have two SecurityToken objects, one for the signature and one for encryption, respectively.

The following token types are subclasses of the generic security token class:

Table 47. Subclasses of the SecurityToken

Token type	JAAS login configuration name
Security context token	system.wss.consume.sct
Derived key token	system.wss.consume.dkt

The following token types are subclasses of the binary security token class:

Table 48. Subclasses to the BinarySecurityToken

Token type	JAAS login configuration name
X.509 token	system.wss.consume.x509
X.509 PKI Path token	system.wss.consume.pkiPath
X.509 PKCS7 token	system.wss.consume.pkcs7

Notes:

- For each JAAS login token consumer configuration name, there is a respective token generator configuration name. For example, for the X509Token, the respective token generator configuration name is system.wss.generate.x509.

- The LTPA and LTPA propagation tokens are only available to a requester that is running as a server-based client. The LTPA and LTPA propagation tokens are not supported for the Java SE 6 or Java EE application client.

To validate the X509Token to the SOAP message on the consumer side, the <X509Token> element must be in the <wsse:Security> element.

1. To validate the securityToken package, com.ibm.websphere.wssecurity.wssapi.token, first ensure that the application server is installed.
2. *If using the default values*, configures the tokens for the Web Services Security token consumer process. , for each token type, the process is similar to the following token consumer process:
   a. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   b. Creates the WSSConsumingContext instance from the WSSFactory instance. Note that the WSSConsumingContext must always be called in a JAX-WS client application.
   c. Creates a JAAS CallbackHandler with information that is required to validate the security token. Review the token class information for which parameters are required or optional. For example, for an X.509 token, you could configure the following:

keyStoreRef	Indicates the reference name of the keystore that is stored in the cryptographic card. It can be specified when the card is set to the hardware.
keyStorePath	Indicates the path of the keystore file. It is not necessary to specify the keyStorePath if the keyStoreRef is set.
keyStorePassword	Indicates the password of the keystore file.
keyStoreType	Indicates the type of keystore file.
alias	Indicates the alias of the key.
keyPassword	Indicates the password of the key.
keyName	Indicates the subject name of the key.

   d. Sets the callback handler into WSSDecryption, WSSVerification, or WSSConsumingContext.
   e. If the callback handler is set into the WSSDecryption or WSSVerification, adds either one into WSSConsumingContext.
   f. Calls WSSConsumingContext.process().
3. *If using other than the default values*, configures the tokens for the Web Services Security token consumer process. For each token type, the process is similar to the following token consumer process:
   a. If you do not use the default JAAS login module and callback handler, you need to prepare a custom one and register the name of JAAS login configuration using the administrative console in advance.
   b. Uses WSSFactory.getInstance() to get the WSS API implementation instance.
   c. Creates the WSSConsumingContext instance from the WSSFactory instance. Note that the WSSConsumingContext must always be called in a JAX-WS client application.
   d. Creates a callback handler with information that is required to validate the security token. Review the token class information for which parameters are required or optional. For example, for a X.509 token, you can configure the following:

keyStoreRef	Indicates the reference name of the keystore that is stored in the cryptographic card. It can be specified when the card is set to the hardware.
keyStorePath	Indicates the path of the keystore file. It is not necessary to specify the keyStorePath if the keyStoreRef is set.
keyStorePassword	Indicates the password of the keystore file.

keyStoreType	Indicates the type of keystore file.
alias	Indicates the alias of the key.
keyPassword	Indicates the password of the key.
keyName	Indicates the subject name of the key.

e. Sets JAAS configuration name and callback handler into WSSDecryption or WSSVerification, or WSSConsumingContext.

f. If JAAS configuration name and callback handler are set into the WSSDecryption or WSSVerification, adds either one into WSSConsumingContext.

g. Calls WSSConsumingContext.process().

Results

If there is an error condition, a WSSException is provided. If successful, the WSSConsumingContext.process() is called, and the security token on the consumer side is validated (authenticated).

Example

The following sample code provides the WSS API example code for decryption using the default JAAS login module and callback handler:

```
// Get the message context
   Object msgcontext = getMessageContext();

// Generate the WSSFactory instance (step: a)
   WSSFactory factory = WSSFactory.getInstance();

// Generate the WSSConsumingContext instance (step: b)
   WSSConsumingContext gencont = factory.newWSSConsumingContext();

// Generate the callback handler (step: c)
   X509ConsumeCallbackHandler callbackHandler = new
        X509ConsumeCallbackHandler(
                                   "",
                                   "enc-sender.jceks",
                                   "jceks",
                                   "storepass".toCharArray(),
                                   "alice",
                                   "keypass".toCharArray(),
                                   "CN=Alice, O=IBM, C=US");

// Generate the WSSDecryption instance (step: d)
   WSSDecryption dec = factory.newWSSDecryption(X509Token.class,
                                                callbackHandler);


// Add WSSDecryption to WSSConsumingContext (step: e)
   concont.add(dec);

// Validate the WS-Security header (step: f)
concont.process(msgcontext);
```

What to do next

For each token type, configure the token using the WSS APIs or using the administrative console. Next, specify the similar generator tokens if you have not done so.

If both the generator and consumer tokens are configured, continue securing SOAP messages at the response consumer using the WSS APIs or configure the tokens using the administrative console.

If both the generator and consumer tokens are configured, continue securing SOAP messages either by verifying the signature or by decrypting the message, as needed. You can use either the WSS APIs or the administrative console to secure the SOAP messages.

Configuring Web services security using the WSS APIs:

The Web Services Security application programming interfaces (WSS API) provide support for securing SOAP message.

Before you begin

Web Service Security supports the following programming models:

- Programming API for securing SOAP message with Web Service Security (WSS API).

  The API programming model design has been redesigned. The new design is an interface-based programming model and is based on Web Services Security Version 1.1 standards but the design also includes support for Web Services Security Version 1.0 for securing the SOAP message. The WSS API programming model implementation is a simplified version, which is based on an early draft proposal of JSR-183, which is the JSR for defining Java API binding for Web Service Security. By design, because the application code is programmed to the interface, any application code that is programmed with the open source implementation should be able to run on the WebSphere Application Server with minimal changes or no changes at all.
- Service Programming Interfaces (SPI) for a service provider

  Similarly, the Web Service Security run time token generation and token consuming SPI have been redesign so that the same security token interface and JAAS Login Module implementation can be used for both the WSS API and the SPI. The WSS SPI for the service provider extend the security token types and provide keys and deriving keys for signing, signature verification, encryption and decryption.

About this task

These programming models extend the following functions :

- Security token types and deriving keys for signing
- Signature and verification
- Encryption and decryption

The following figure demonstrates how to use the simplified WSS APIs to secure a SOAP message by using XML digital signature and XML encryption.

The configuration model for Web services has also been redesigned from a deployment descriptor model to a policy set model. The configuration programming model is based on configuring policy sets using a security policy to specify security constraints.

The functions provided by the policy set configurations are the same as the functions supported by the WSS API for the Web Service Security run time. However, the security policy that is defined using policy sets has a higher priority over the WSS API. When the WSS API and the policy set are both used in the application, the default behavior is for the security policy from the policy set to be enforced and the WSS API to be ignored. To use the WSS API in the application, you must make sure that there is no policy set attached to the application or to the application resources, or make sure there is no security policy in the attached policy set.

Web Service Security can be enabled by either using a policy set that is configured by using the administrative console, or by using the WSS API for configuration.

Using the WSS API, complete the following high-level steps to secure the SOAP message:

1. Use the WSSSignature API to configure the signing information for the request generator (client side) binding. Different message parts can be specified in the message protection for a request on the generator side. The default required parts are BODY, ADDRESSING_HEADERS, and TIMESTAMP.

   The WSSSignature API also specifies the different algorithm methods to be used with the signature for message protection. The default signature method is RSA_SHA1. The default canonicalization method is EXC_C14N.
2. Use the WSSSignPart API if you want to add or change the signed parts to be used for message protection. The default signed parts are WSSSignature.BODY, WSSSignature.ADDRESSING_HEADERS, and WSSSignature.TIMESTAMP.

   The WSSSignPart API also specifies the different algorithm methods to be used if you added or changed the signed parts. The default digest method is SHA1. The default transform method is TRANSFORM_EXC_C14N. For example, use the WSSSignPart API if you want to generate the signature for the SOAP message using the SHA256 digest method instead of the default value of SHA1.
3. Use the WSSEncryption API to configure the encryption information on the request generator side. The encryption information on the generator side is used for encrypting an outgoing SOAP message for the request generator (client side) bindings. The default targets of encryption are BODY_CONTENT and SIGNATURE

   The WSSEncryption API also specifies the different algorithm methods to be used to protect message confidentiality. The default data encryption method is AES128. The default key encryption method is KW_RSA_OAEP.
4. Use the WSSEncryptPart API if you want to add or change the encrypted parts to be used for message confidentiality. For example, if you want to change the data encryption method from the default value of AES128 to TRIPLE_DES.

   No algorithm methods are required for encrypted parts.
5. Use the WSS API to attach the token on the generator side. The requirements for the security token depend on the token type. The JAAS Login Module and the JAAS CallbackHandler are responsible for creating the security token on the generator side. Different standalone tokens can be sent in request or response. The default token is the X509Token. The other token that can be used for signing is the DerivedKeyToken, which is used only with Web Services Secure Conversation (WS-SecureConversation).
6. Use the WSSVerification API to verify the signature for the response consumer (client side) binding. Different message parts can be specified in the message protection for a response on the consumer side. The required targets for verification are BODY, ADDRESSING_HEADERS, and TIMESTAMP.

   The WSSVerification API also specifies the different algorithm methods to be used for verifying the signature and for message protection. The default signature method is RSA_SHA1. The default canonicalization method is EXC_C14N.
7. Use the WSSVerifyPart API to add or change the verify signed parts to be used for message protection. The required verify parts are WSSVerification.BODY, WSSVerification.ADDRESSING_HEADERS, and WSSVerification.TIMESTAMP.

   The WSSVerifyPart API also specifies the different algorithm methods to be used if you added or changed the verification parts. The default digest method is SHA1. The default transform method is TRANSFORM_EXC_C14N.
8. Use the WSSDecryption API to configure the decryption information for the response consumer (client side) binding. The decryption information on the consumer side is used for decrypting an incoming SOAP message. The default targets of decryption are BODY_CONTENT and SIGNATURE. The default data encryption method is AES128. The default key encryption method is KW_RSA_OAEP.

   No algorithm methods are required for decryption.
9. Use the WSSDecryptPart API if you want to add or change the decrypted parts to be used for message confidentiality. For example, if you want to change the data encryption method from the default value of AES128 to TRIPLE_DES.

   No algorithm methods are required for decrypted parts.

10. Use the WSS API to configure the token on the consumer side. The requirements for the security token depend on the token type. The JAAS Login Module and the JAAS CallbackHandler are responsible for validating (authenticating) the security token on the consumer side. Different standalone tokens can be sent in request or response.

    The WSS API adds the information for the candidate token that is used for decryption. The default token is X509Token.

Results

What to do next

The Web Service Security run time token generation and token consuming Service Programming Interfaces (SPI) have been redesign so that the same Security Token interface and JAAS Login Module implementation can be used in both the WSS API and the SPI. See the SPI information for detail descriptions.

Web services security APIs:

The Web services security programming model provides application programming interfaces (WSS API) for securing the SOAP message. The WSS API model is based on Web Services Security Version 1.1 standards but also includes support for Web Services Security Version 1.0.

The Web services security APIs (WSS APIs) can generate and process the following SOAP-related bindings for XML security:

- XML signature and signature verification
- XML encryption and decryption

The token processing and pluggable token architecture in the Web service security run time has been redesign to reuse the same Security Token interface and the JAAS Login Module as those used for the WSS APIs.

The following table lists the WSS API interfaces that are provided with WebSphere Application Server and used to configure signing and encryption information in the SOAP bindings for the generator and consumer bindings.

Table 49. WSS API interfaces

WSS API interfaces	Description
WSSDecryption	Package: com.ibm.websphere.wssecurity.wssapi.decryption This interface is responsible for specifying decryption. The default values for decryption include: • Targets: BODY_CONTENT, SIGNATURE • Data encryption method: AES128 • Key encryption method: KW_RSA_OAEP • Security token: X.509
WSSDecryptPart	Package: com.ibm.websphere.wssecurity.wssapi.decryption This interface is responsible for adding decrypted parts, as needed. If specified, the default values for decrypted parts include: • Security token: X.509 • Transform method: N/A (not applicable)

Table 49. WSS API interfaces (continued)

WSS API interfaces	Description
WSSEncryption	Package: com.ibm.websphere.wssecurity.wssapi.encryption This interface is responsible for the encryption component. The default values for encryption include: • Targets: BODY_CONTENT, SIGNATURE • Data encryption method: AES128 • Key encryption method: KW_RSA_OAEP • Security token: X.509 • refType: SecurityToken.REF_KEYID • mtomOptimize: false
WSSEncryptPart	Package: com.ibm.websphere.wssecurity.wssapi.encryption This interface is responsible for adding encrypted parts, as needed. If specified, the default values for encrypted parts include: • Transform method: N/A (not applicable)
WSSSignature	Package: com.ibm.websphere.wssecurity.wssapi.signature This interface is responsible for specifying the signature. The default values for signature include: • Targets: BODY, ADDRESSING_HEADERS, TIMESTAMP • Signature method: RSA_SHA1 • Canonicalization method: EXC_C14N • Security token: X.509 • Type of token reference: SecurityToken.REF_STR
WSSSignPart	Package: com.ibm.websphere.wssecurity.wssapi.signature This interface is responsible for adding signed parts, as needed. If specified, the default values for signed parts include: • Transform method : TRANSFORM_EXC_C14N • Digest method: SHA1
WSSVerification	Package: com.ibm.websphere.wssecurity.wssapi.verification This interface is responsible for specifying the signature verification. The default values for verification include: • Targets: BODY, ADDRESSING_HEADERS, TIMESTAMP • Signature method: RSA_SHA1 • Canonicalization method: EXC_C14N • Security token: X.509
WSSVerifyPart	Package: com.ibm.websphere.wssecurity.wssapi.verification This interface is responsible for adding verify parts, as needed. If specified, the default values for verify parts include: • Digest method: SHA1 • Transform method: TRANSFORM_EXC_C14N

Also see the information about pre-configured generator and consumer tokens.

Web services security configuration considerations when using the WSS API:

To secure Web services security for WebSphere Application Server, you can specify several different configurations using the Web Services Security APIs (WSS API). The Web services security specification provides a flexible way to secure Web services messages using XML digital signature, XML encryption, and attaching security tokens. You can enable Web services security by either configuring a policy set or by using the Web services security APIs (WSS API). The implementation for WSS API has default values for which message parts are to be signed or encrypted. The default values for the WSS APIs help end users to enable Web services security quickly.

Different message parts can be specified in the message protection for request or response, and different standalone tokens can be sent in request or response. However, there is only one symmetric or one asymmetric binding assertion to describe the token type and the algorithm that is used for message protection.

Using the WSS API, you can override any default values. However, when you alter the protection parts, note that all the default protection parts are cleared. For example, if you specify that you want to encrypt the Username token instead of the default X.509 token, all the default values of the encrypting protection parts are cleared.

The following table shows an example of the relationships between each of the configurations:

Table 50. Request generator and response consumer configurations

Type of configuration	Configuration name	Configurations and default values
Request generator	Signing information	• Canonicalization method: WSSSignature.EXC_C14N • Signature method: WSSSignature.RSA_SHA1 • Digest method: WSSSignPart.SHA1 • Transform method: WSSSignPart.TRANSFORM_EXC_C14N • Signed part - Body: WSSSignature.BODY • Signed part - Addressing: WSSSignature.ADDRESSING_HEADERS • Signed part - Timestamp: WSSSignature.TIMESTAMP • Token reference: SecurityToken.REF_STR • Token - Value type: X509Token.ValueType • Token - JAAS login configuration name: system.wss.generate.x509
Response consumer	Signature verification information	• Canonicalization method: WSSVerification.EXC_C14N • Signature method: WSSVerification.RSA_SHA1 • Transform method: WSSVerifyPart.TRANSFORM_EXC_C14N • Signed part - Body: WSSVerification.BODY • Signed part - Addressing: WSSVerification.ADDRESSING_HEADERS • Signed part - Timestamp: WSSVerification.TIMESTAMP • Token - Value type: X509Token.ValueType • Token - JAAS login configuration name: system.wss.consume.x509
Request generator	Encryption information	• Encrypted key: true • Key encryption method: WSSEncryption.KW_RSA_OAEP • Data encryption method: WSSEncryption.AES128 • Encryption part: WSSEncryption.BODY_CONTENT • Token reference: SecurityToken.REF_KEYID • Token - Value type: X509Token.ValueType • Token - JAAS login configuration name: system.wss.generate.x509

Table 50. Request generator and response consumer configurations (continued)

Type of configuration	Configuration name	Configurations and default values
Response consumer	Decryption information	• Encrypted key: true • Key decryption method: WSSDecryption.KW_RSA_OAEP • Data decryption method: WSSDecryption.AES128 • Decryption part: WSSDecryption.BODY_CONTENT • Token - Value type: 509Token.ValueType • Token - JAAS login configuration name: system.wss.consume.x509

Encrypted SOAP headers:

The encrypted header element provides a standard way of encrypting SOAP headers. As one of the extensions to the OASIS SOAP message security specification, the encrypted header element indicates that the responder has processed the request. Encrypting SOAP headers and parts help to provide more secure message-level security.

The EncryptedHeader or <wsse11:EncryptedHeader> element is a part of the updated Web Services Security Version 1.1 standard and enables interoperability with other vendors that support the Version 1.1 standards, such as Microsoft .NET.

Use the EncryptedHeader element for encrypting SOAP header blocks. The EncryptedHeader element allows Web Services Security to be compliant with the SOAP mustUnderstand processing guidelines and to prevent disclosure of information that is contained in attributes on a SOAP header block.

The <wsse11:EncryptedHeader> element must contain one <xenc:EncryptedData> element. Only one <xenc:EncryptedData> element per encrypted header element is permitted.

Encrypted data element

Normally, the programming model, such as JAX-WS, deserializes the SOAP message to a Java binding object before dispatching the call to the application code. However, if the SOAP message is encrypted, the deserialization fails because, before encryption, the original content is replaced with the EncryptedData XML element from the XML Encryption standard.

In certain cases, it might be desirable for the token that is included in the <wsse:Security> header to be encrypted for the recipient processing role.

Follow these guidelines when using the EncryptedData element:

- The EncryptedHeader element must contain one EncryptedData element.
- The <xenc:EncryptedData> element may be used to contain a security token and include it in the <wsse:Security> header.
- The <xenc:EncryptedData> must not include an XML ID for referencing the contained security token.
- All <xenc:EncryptedData> tokens must either have an embedded encryption key or must be referenced by a separate encryption key.
- If compliance with Basic Security Profile 1.1 is desired, the <xenc:EncryptedData> element must have an Id attribute.

Policy assertion for encrypted parts

The EncryptedParts policy assertion specifies which header is to be encrypted in the security policy. The following table describes the elements and attributes that can be used for EncryptedParts.

Table 51. Attributes and elements of the EncryptedParts element

Element or attribute	Description
/sp:EncryptedParts/sp:Header	Optional. Presence of this optional element indicates that a specific SOAP header (or set of such headers) must be protected. You can have multiple sp:Header elements within a single EncryptedParts element. Each header (or set of headers) must be encrypted, and this encryption will encrypt the elements by using Web Services Security Version 1.1 encrypted headers. As such, if WS-Secuirty 1.1 Encrypted Headers are not supported by a service, then the headers cannot be encrypted by using message-level security. If multiple SOAP headers with the same local name but different namespace names are to be encrypted, multiple sp:Header elements are required, either as part of a single sp:EncryptedParts assertion or as part of separate sp:EncryptedParts assertions.
/sp:EncryptedParts/sp:Header/@Name	Optional. This attribute indicates the local name of the SOAP header to be confidentiality protected. If this attribute is not specified, all SOAP headers whose namespace matches the Namespace attribute are to be protected.
/sp:EncryptedParts/sp:Header/ @Namespace	Required. This attribute indicates the namespace of the SOAP headers to be confidentiality protected.

The following message example shows what the EncryptedHeader element looks like on a message where the EncryptedParts policy assertion for the encrypted header has been specified on the policy:

```
<S:Envelope xmlns:S="..." xmlns:wsse="..." xmlns:wsse11="..." xmlns:wsu="..."
   xmlns:xenc="..." xmlns:ds="...">
   <S:Header>
     <wsse:Security>
      <!-- Tokens etc. -->
       <xenc:EncryptedKey>
         <xenc:EncryptionMethod Algorithm="..."/>
           <ds:KeyInfo>
             ... ...
           </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
          </xenc:CipherData>
          <xenc:ReferenceList>
            <xenc:DataReference URI="#hdrID"/>
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
     </wsse:Security>
     <wsse11:EncryptedHeader wsu:Id="hdrID">
       <xenc:EncryptedData Id="encDataID">
         <xenc:CipherData>
           <xenc:CipherValue>...</xenc:CipherValue>
         </xenc:CipherData>
         ...
       </xenc:EncryptedData>
     </wsse11:EncryptedHeader>
   </S:Header>
   <S:Body>
     ...
   </S:Body>
</S:Envelope>
```

To encrypt headers in the Web Services Security Version 1.0 specification format, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.0 property with a value of `true` on the

<encryptionInfo> element in the binding. When this property is specified, the target header for encryption is replaced by an <EncryptedData> element, instead of an <EncryptedHeader> element that contains an <EncryptedData> element.

For Web Services Security Version 1.1 behavior that is equivalent to WebSphere Application Server versions prior to version 7.0, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.1.pre.V7 property with a value of `true` on the <encryptionInfo> element in the binding. When this property is specified, the <EncryptedHeader> element includes a wsu:Id parameter and the <EncryptedData> element omits the Id parameter. This property should only be used if compliance with Basic Security Profile 1.1 is not required.

For complete information about the EncryptedHeader element and the EncryptedData element, see the Web Services Security Version 1.1 specification.

Signature confirmation:

Web services security signature confirmation is an enhanced XML digital signature, and it is included in the Web services security standard. XML digital signature is used for signing elements of the SOAP envelope.

As one of the extensions to the OASIS SOAP message security specification, the signature confirmation element incorporates the elements that are needed within the response message in order to confirm the signature that is contained in a request message. XML digital signature and signature confirmation help to provide more secure message-level security.

Web Services Security Version 1.0 for SOAP message security did not provide any guidance on how to confirm mutual understanding of the request that prompted this response. The SignatureConfirmation or <wsse11:SignatureConfirmation> element has been added to the Web Services Security Version 1.1 specification. The <wsse11:SignatureConfirmation> element ensures that the signature is processed by the intended recipient and indicates that the responder has processed the signature in the request. The signature confirmation element is part of the updated Web Services Security standard and enables interoperability with other vendors that support the Version 1.1 standards, such as Microsoft .NET.

Because of the stateless nature of Web services and due to different message exchange patterns (MEPs), consider the following assumptions:

- Assume that session affinity is enabled if a cluster is enabled for the clients that are running in WebSphere Application Server. When session affinity is enabled, it implies that the response is sent back to the initiating client of the server.
- Assume WS-Addressing is enabled for asynchronous message exchange patterns. When WS-Addressing is enabled, it allows the run time to relate the response back to the request. An asynchronous response is sent back to the application of the initiating WebSphere Application Server.

Syntax

The SignatureConfirmation element indicates that the responder has processed the signature in the request. When this element is not present in a response, the initiator interprets that the responder is not compliant.

The format for the signature confirmation element is as follows:

```
<wsse11:SignatureConfirmation wsu:Id="..." Value="..." />
```

where:

wsu:Id
The identifier that is used when referencing this element in the <ds:SignedInfo> reference list of

the signature of the associated response message. This attribute is required so that unambiguous references are made to this <wsse11:SignatureConfirmation> element.

Value This attribute is optional and contains the contents of a <ds:SignatureValue> that is copied from the associated request. If the request is unsigned, this attribute must not be present. If this attribute is specified without a value (empty), the initiator interprets this as incorrect behavior and processes it accordingly. When this attribute is not present, the initiator interprets this to mean that the response is based on a request that was not signed.

Configuration

To configure signature confirmation, configure the policy file using the administrative console, and select **Require signature confirmation**. To process Signature Confirmation correctly, the initiator of the request needs to preserve the signatures during request generator processing and later needs to retrieve the signatures for confirmation checks.

Response generation rules

Additional SOAP security elements for the SOAP responder are used to confirm that the response is in relationship to a particular request. The responder must include the contents of the <ds:SignatureValue> element of the request signature as the value of the @Value attribute of the <wsse11:SignatureConfirmation> element.

The following response generation rules apply when using the SignatureConfirmation policy assertion:

- If there are no signatures on the request, the response contains one SignatureConfirmation element, without a value. For MEPs where there are multiple requests (all without signatures) and one response, the response contains one SignatureConfirmation element without a value.
- If there are signatures on the request, the response contains a SignatureConfirmation element for each signature, with a value that matches the signature value on the request. For MEPs where there are multiple requests, with at least one containing a signature, and one response, the response contains a SignatureConfirmation element for each signature that is found on the requests, with a value that matches the signature value on the request.
- For MEPs where there is one request and multiple responses, each response contains the appropriate SignatureConfirmation elements as noted in the first and second bullets.
- If the SOAP request contains multiple signatures, the requester will find all of the signature confirmation elements contained in the response, and will check the values of the value fields of the signature confirmation elements against the values of the signatures in the original SOAP request.

Securing requests to the trust service using system policy sets

WebSphere Application Server provides message-level protection for its security token service, known as the WebSphere Application Server trust service. For the trust service, you must use a special class of policy sets known as system policy sets.

Before you begin

You can secure requests to the trust service by using two different configuration methods:

- Use the administrative console to define and attach a system policy set and binding to a trust service operation that is associated with an endpoint.
- Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure system policy sets for the trust service. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

About this task

For WebSphere Application Server trust service security, you must configure the system policy sets, the bindings, the trust service attachments, and the security cache.

Perform the following high-level steps. The order of the tasks is not important but all high-level required steps must be performed to complete the trust configuration.

1. Define a new system policy set or manage existing system policy sets. To manage system policy sets, you can perform the following tasks:
   a. Define the system policy set and binding. The system policy set can be a new or existing policy set. If you create a new system policy set, you must specify and configure the policy types. A default binding configuration is associated with each policy type.
   b. Modify the system policy set, as needed.

      Other optional policy set-related tasks that you can perform include:
      - Add, edit, or remove policy set attachments.
      - Edit, enable, disable or remove policy types
      - Create a system policy set by selecting and copying an existing system policy set. When copying an existing system policy set, you also specify whether to move the existing attachments to this new system policy set.
      - Delete system policy sets. You cannot delete pre-configured system policy sets that are provided by WebSphere Application Server by default.
      - Archive a system policy set by selecting and exporting an existing system policy set. When exporting an existing system policy set, you create a .zip archive file. The .zip file for exporting the policy set is provided for downloading. For example, if you have a policy set named ABC_ps and you want to export and move the archive file from ServerA to ServerB, first use the export function to create the .zip file. Then, manually transfer the archive file to ServerB.
2. Create and manage explicit attachments. You can perform the following trust service attachment tasks:
   a. Attach the system policy set and assign a binding to an endpoint. For an endpoint, you can create explicit attachments for each of the four trust service operations to the respective Trust Service Defaults policy sets and bindings. After you have created these initial attachments, you can view and further modify existing policy set and binding configurations.
   b. Modify existing policy set attachment and binding configurations, as needed.. The system policy set can be a new or existing policy set. If you create a new system policy set, you must specify and configure the policy types. A default binding configuration is associated with each policy type.

      The system policy set that is attached to issue and renew must correspond to the client and endpoint's bootstrap policy set and the system policy set attached to validate and cancel must correspond to the client and endpoint's application policy set. The bootstrap policy set for the endpoint service is only required if the endpoint service makes issue and renew requests to the trust service.

      Other optional attachment-related tasks that you can perform include:
      - Change the system policy set and binding configurations.
      - Create custom system policy sets and bindings.
      - Attach each of the four default trust service operations to a system policy set and binding.
      - Attach each of the four trust service operations associated with a specific endpoint to a system policy set and binding.
      - Specify that the selected trust service operations for an endpoint inherit the respective default trust service policy set and binding.
      - Assign the Default binding or a custom binding configuration to the selected policy set attachment.
      - Update the trust service runtime configuration.

3. Manage the security context token provider that the trust service provides. You can perform the following trust service token provider tasks:
   a. Modify the configuration of the Security Context Token provider, as needed..

      Other optional token provider-related tasks that you can perform include:

      - Update the trust service runtime configuration for any token provider configuration changes.
4. Manage the trust service default token provider and any endpoints that have an explicitly assigned token (rather than inheriting from the default). Targets are endpoints that are assigned a specific token provider. You can perform the following trust service target tasks:
   a. Create a new trust service target by explicitly assigning a service endpoint URL to the default token provider.. Performing this task creates an explicit assignment to the default trust service token provider, the Security Context Token. All other endpoints inherit the trust service default token provider.
   b. Configure a target. WebSphere Application Server defines one default supported token provider, the Security Context Token. Other tasks that you can perform for existing targets include:
      - Modifying one or more endpoints that have a security context token provider explicitly assigned.
      - Changing the token provider for an endpoint from inherited to explicitly assigned. Therefore, the token provider for the endpoint does not change as the default trust service token provider changes.
      - Changing the token provider for an endpoint from explicitly assigned to inherited. Therefore, the token provider for the endpoint is the default trust service token provider and changes as the default changes.
      - Updating the trust service runtime configuration.
5. Configure the security cache. You can change the behavior of client-side security caching.
6. Update the trust service runtime configuration. You must update the runtime configuration whenever one or all of the following trust-related items are created or changed:
   - Trust service attachments
   - Token providers
   - Targets

Results

After the configurations are completed and the trust service runtime configuration has been updated, you have used the administrative console to secure requests to the trust service by using system policy sets.

Enabling secure conversation:

Use secure conversation to secure Web services application messages.

Before you begin

Applications that contain Web services must have been deployed.

About this task

The Organization for the Advancement of Structured Information Standards (OASIS) Web Services Secure Conversation (WS-SecureConversation) draft specification describes ways to establish a secure session between the initiator and recipient of SOAP messages. The WS-SecureConversation draft specification also defines how to use the OASIS Web Services Trust (WS-Trust) protocol to establish a security context token (SCT). For complete information, see the OASIS Web Services Secure Conversation specification.

WebSphere Application Server supports the ability of an endpoint to issue a security context token for WS-SecureConversation, and thereby provides a secure session between the initiator and recipient of SOAP messages.

The following figure describes the flow that is required to establish a secured context and to use session-based security.

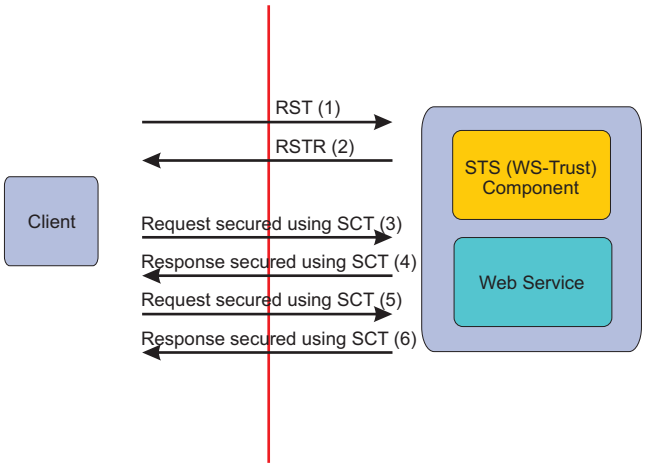


Figure 2. Displaying the flow between the client and the Web service and security token service

In the WS-SecureConversation specification, a security context is represented by the <wsc:SecurityContextToken> security token. The following example represents the assertion syntax for a <wsc:SecurityContextToken> element.

```
<wsc:SecurityContextToken wsu:Id="..." ...>
    <wsc:Identifier>...</wsc:Identifier>
    <wsc:Instance>...</wsc:Instance>
    ...
</wsc:SecurityContextToken>
```

The security context token does not support references to it by using key identifiers or key names. All references must either use an ID (to a wsu:Id attribute) or a <wsse:Reference> to the <wsc:Identifier> element.

WebSphere Application Server provides these pre-configured secure conversation-related polices:

- The **SecureConveration** policy set follows the WS-SecureConversation and WS-Security specifications and provides a policy set with secure conversation enabled and using keys derived from security context token for signing and encrypting the application messages.
- The **Username SecureConversation** policy set follows the WS-SecureConversation and WS-Security specifications and adds authentication using the Username token.
- The **LTPA SecureConversation** policy follows the WS-SecureConversation and WS-Security specifications and provides authentication using the Lightweight Third Party Authentication (LTPA) tokens.

In this example, the default SecureConversation policy set, and the default WS-Security binding and TrustServiceSecurityDefault binding are used to achieve the task of enabling secure conversation. The default SecureConversation policy set has both the application policy (symmetricBinding) and the bootstrap policy (asymmetricBinding). The application policy is used to secure application messages and the bootstrap policy is used to secure the RequestSecurityToken (RST) messages.

A trust service that issues a security context token is configured with the TrustServiceSecurityDefault system policy and the TrustServiceSecurityDefault binding. The trust policy is responsible for securing RequestSecurityTokenResponse (RSTR) messages. If the bootstrap policy is modified, the trust policy has to be modified to match both of the configurations.

Note: The following steps are to be used only in development and test environments.

The Web Services Security (WS-Security) default bindings that are used here contain sample key files and must be customized before use in a production. For the production environment, use of custom bindings is advised. Also note that, if the profile is created by using the choice of **Create the server using the development template**, you can skip steps 2 and 3.

To configure secure conversation, configure the policy set, and add a policy assertion to the policy, complete the following steps:

1. Make a copy of a default secure conversation policy so you can customize the policy set for your own environment.
   a. Launch the administrative console, and click **Services > Policy sets > Application policy sets**.
   b. Select the check box next to an existing policy set that follows the WS-SecureConversation specifications. For example, you might click the check box next to **SecureConversation**. This policy set is one of the pre-configured secure conversation-related application policy sets that is listed in the table. The SecureConversation policy set has a bootstrap policy to match the default policy set for the trust service to issue and renew tokens.
   c. Click **Copy**.
   d. Enter a unique name for the new copy of the SecureConversation application policy set. For example: CopyOfSCPolicySet
   e. Optional: Change the description, as needed, for your customized version of this policy set.
2. Attach the policy set and binding to the application.
   a. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application name***.
   b. Click either **Service provider policy sets and bindings** or **Service client policy sets and bindings** to attach resources to the CopyOfSCPolicySet policy set. The general binding is assigned automatically as the default.
   c. You can use the **Attach Policy Set** and **Assign Binding** menu lists to select a different policy set or binding.

Results

After completing these steps, you have configured secure conversation.

What to do next

Next, review the example scenario about how to establish a security context token to secure a secure conversation.

Enable distributed cache and session affinity when using Secure Conversation:

WebSphere Application Server provides message-level protection in a cluster environment. You can use Web Services Secure Conversation (WS-SecureConversation) for message-level protection of Java API for XML Web Services 2.0 (JAX-WS) Web services in a cluster environment.

Before you begin

A Web services request that is protected with a Security Context Token (SCT) is routed to one server in a cluster, but that SCT might have been issued or renewed by a different server in the cluster. If the

WebSphere Application Server distributed cache is not configured to replicate or does not replicate quickly enough, the server processing the request might not have access to the SCT.

About this task

Perform the following high-level steps to enable distributed cache and session affinity when using secure conversation for message-level protection in a cluster environment.

1. Enable the distributed cache for the Security Context Token. Perform the following steps:
   a. In the Administrative Console for IBM WebSphere Application Server, click **Services > Trust service > Trust Providers > Security Context Token**.
   b. Select the **Distributed cache** check box.
   c. Click **OK** and then click **Save** to save the configuration.
2. Create a replication domain. Perform the following steps:
   a. In the Administrative Console, click **Environment > Replication domains > New**.
   b. Enter a name. For example, `ABCDomain`.
   c. Under Number of replicas, select the **Entire Domain** option.
   d. Click **OK** and then click **Save** to save the configuration.
3. Enable the dynamic cache. Perform the following steps for each server in the cluster:
   a. In the Administrative Console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Container Services** → **Dynamic Cache Service**.
   b. Select the **Enable service at server startup** option.
   c. Select the **Enable cache replication** option.
   d. Select the replication domain name that you created. For example, `ABCDomain`.
   e. Select the replication type as **Both push and pull**.
   f. Click **OK** and then click **Save** to save the configuration.
4. Specify the distributed cache batch update interval as 100 milliseconds. Perform the following steps for each server in the cluster:
   a. In the Administrative Console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Java and Process Management** → **Process Definition** → **Java Virtual Machine** → **Custom Properties** → **New**.
   b. Enter the `com.ibm.ws.cache.CacheConfig.batchUpdateInterval` property name.
   c. Enter `100` as the property value.
   d. Click **OK** and then click **Save** to save the configuration.
5. Install and configure a Web server or proxy server that supports session affinity. The IBM HTTP Server and WebSphere Application Server proxy server support session affinity. In the WebSphere Application Server Information Center, see Communicating with Web servers for information on installing and configuring the IBM HTTP Server.
6. Configure the client systems to send the Web services requests to the host and port where the Web server or proxy server is running. The Web server or proxy server then routes the requests to the proper cluster member.
7. On the services that are receiving the Web services requests, which are protected by using Web Services Secure Conversation, select the HTTP transport Session enabled policy option. Complete the policy set configuration by following these steps:
   a. Add the HTTP Transport policy to the policy set that is being used by the services.
   b. In the configuration panel for the HTTP Transport policy, select **Session enabled**.
   c. Click **OK** and then click **Save** to save the configuration.

8. On the client systems that are sending the Web services requests and are protected by Secure Conversation, enable the HTTP transport Maintain session property. Complete the policy set configuration or set the property programmatically. If you are using a policy set with your configuration, follow these steps:
   a. Add the HTTP Transport policy to the policy set that is being used by the clients.
   b. At the HTTP Transport policy configuration panel, select the **Session enabled** option.
   c. Click **OK** and then click **Save** to save the configuration.

Results

After the configurations are completed, you have enabled the distributed cache and session affinity when using secure conversation in a cluster environment. If the server processing the request does not have access to the SCT, it will fail the request with the error of `Either null SCT or invalid SCT`.

Example

The following example, which is a code snippet, demonstrates how to programmatically set the Maintain session property on the correct JAX-WS object:

```
Map<String> rc = ((BindingProvider) port).getRequestContext();
...
rc.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);
... </String>
```

Example: Establishing a security context token to secure a secure conversation:

This example describes the flow of how the initiator establishes the security context token (SCT) by using the WS-Trust protocol for session-based security with the recipient. After establishing the security context token, derived keys from the security context token are used to sign and encrypt the SOAP message to provide message-level protection. This examples focuses on the message exchanges using the security context token in the overall flow of the SOAP messages.

The Organization for the Advancement of Structured Information Standards (OASIS) Web Services Secure Conversation (WS-SecureConversation) specification describes ways to establish a secure session between the initiator and recipient of SOAP messages. The WS-SecureConversation specification also defines how to use Web Services Trust (WS-Trust) protocol to establish a security context token. WebSphere Application Server Version 7 supports both Version 1.3, and the draft version, of the WS-SecureConversation specification.

WebSphere Application Server supports the ability of an endpoint to issue a security context token for WS-SecureConversation and thereby provides a secure session between the initiator and recipient of SOAP messages.

The following figure describes the flow that is required to establish a secured context and to use session-based security.

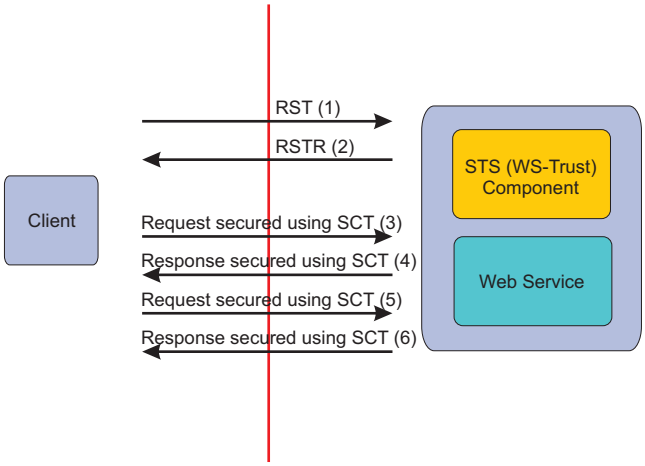


Figure 3. Displaying the flow between the client and the Web service and security token service

Exchanging messages between the initiator and the recipient

The following figure shows how the messages are exchanged between the initiator and the recipient to establish the security context token. The two WS-Trust protocols, RequestSecurityToken (RST) and RequestSecurityTokenResponse (RSTR), are used to request the security context token from the recipient endpoint.

The bootstrap policy is used to secure the RST and validate the RSTR request, which is typically different from the application security policy.

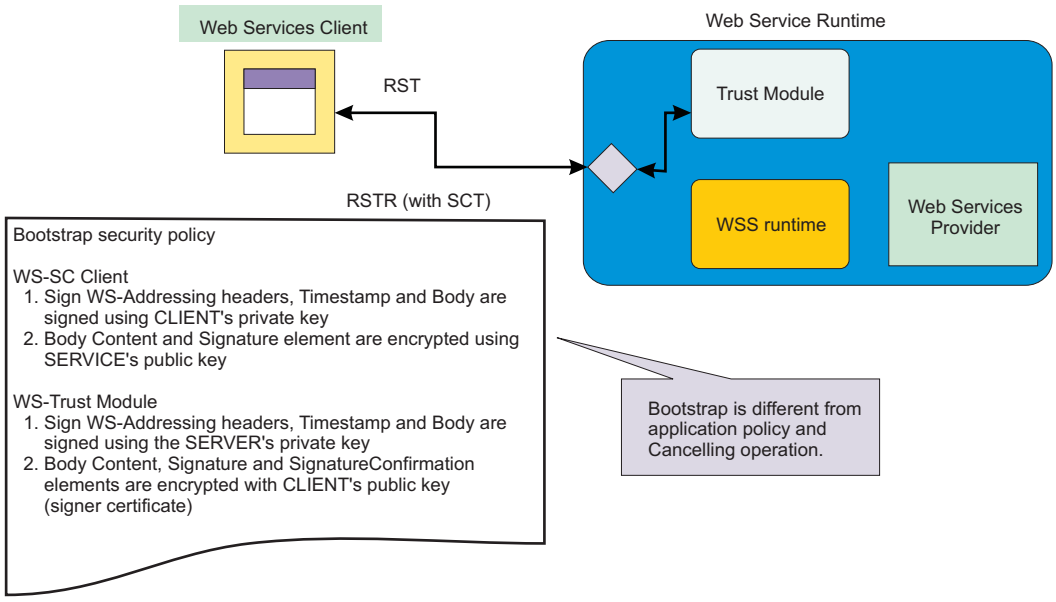


Figure 4. Using WS-Trust protocols RST and RSTR to establish the SCT between the initiator and the recipient

Scenario describing how to use secure conversation

Typically, to use secure conversation, the following steps are involved;

1. The client sends a RequestSecurityToken (RST) trust request for a security context token to an application endpoint with its secret key (entropy and target key size) and requests the target service secret key.

This request is typically secured with asymmetric Web service security that is defined in the bootstrap policy.

2. The RST is processed by the trust service and, if the request is trusted based on the security policy, the trust service returns the security context token with the target service secret key by using a WS-Trust RequestSecurityTokenResponse (RSTR).

   This request is typically secured with asymmetric Web service security. The client verifies whether the RSTR can be trusted, based on the bootstrap policy.

3. If the RequestSecurityTokenResponse is trusted, the client secures (signs and encrypts) the subsequent application messages by using the session keys.

   The session keys are derived from secret of the security context token that is obtained from the initial WS-Trust RequestSecurityToken and RequestSecurityTokenResponse messages that are exchanged between the initiator and the recipient.

4. The specification defines an algorithm of how to derive the key based on the initial secret. The target Web service calculates the derive key from the metadata contained in the security header of the SOAP message and the initial secret.
5. The target Web service uses the derived key to verify and decrypt the message based on the application security policy.
6. The target Web Service uses the derived key from the secret to sign and encrypt the response based on the application security policy.
7. Repeat of steps 3 through 6 until the message exchange has completed.

Using keys that are derived from the secret of the security context token

After the security context token is established, the application messages are secured with message protection by using keys that are derived from the secret of the security context token. The derived keys are used to secure the application messages by signing and encrypting the application messages. The security context token contains a UUID, which is used as identification of a shared secret. The token UUID can be used in the SOAP message to identify the security context token for the message exchanges. The secret must be kept in memory by the session participants (in this case the initiator and the recipient) and protected. Compromising the secret undermines the secure conversation between the participants.

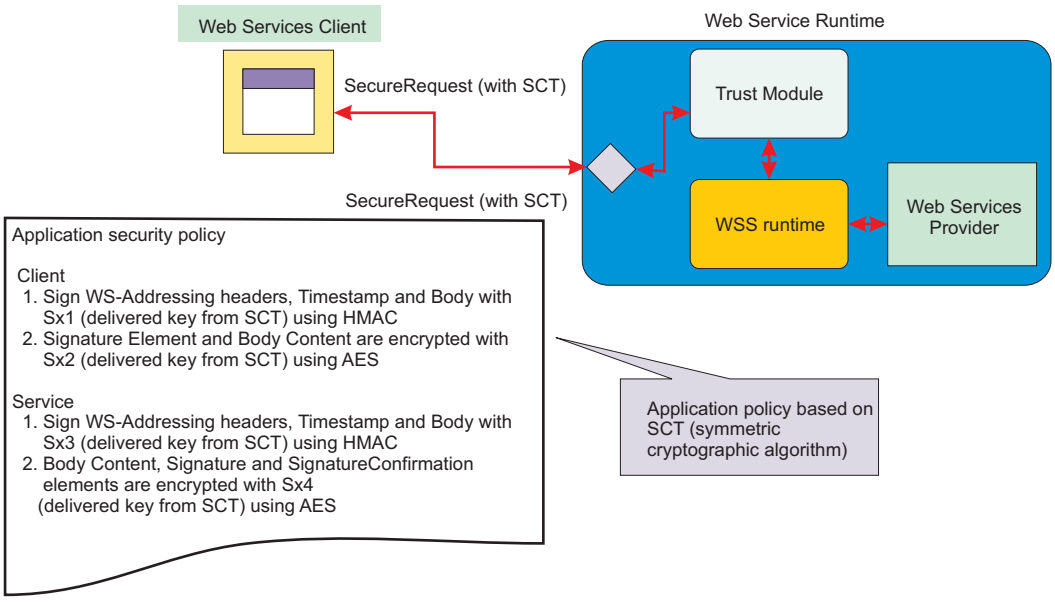


Figure 5. Securing application messages with keys derived from secret of the security context token

A similar scenario except with Web Services Reliable Messaging (WS-ReliableMessaging) is possible from the WS-SecureConversation prospective. See the example for establishing a security context token to secure reliable messaging.

Example: Establishing a security context token to secure reliable messaging:

This example scenario includes functions that are required for the composite scenario of Web Services Reliable Messaging (WS-ReliableMessaging), WS-SecureConversation, and WS-Trust. The scenario describes how to use WS-SecureConversation with WS-ReliableMessaging, the scenario is described from the WS-SecureConversation perspective.

The flow of this Web Services Reliable Messaging (WS-ReliableMessaging) scenario is very similar to the flow of the WS-SecureConversation scenario, and the exchange of the application messages is very similar to the Secure Conversation scenarios. The main difference in the two example scenarios is that the WS-ReliableMessaging sequence is secured with the security context token and scopes the WS-ReliableMessaging sequence to the security context token.

The following figure describes a summary of the message flows that are required to establish a security context token to secure reliable messaging.

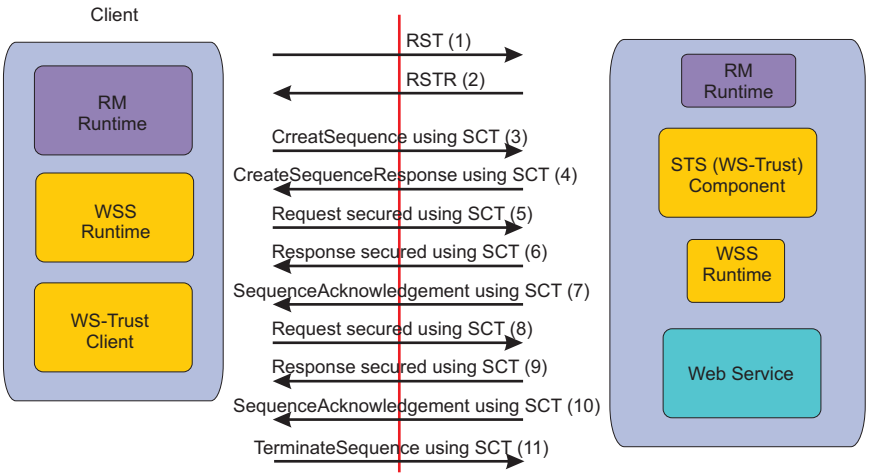


Figure 6. Messages exchange for the SCT and reliable messaging

Scenario

The WS-ReliableMessaging sequence is secured with the security context token and is scoping the WS-ReliableMessaging sequence to the security context token. This scenario focuses on the message exchanges that are using the security context token in the overall flow.

Note: The exact detail of how WS-ReliableMessaging is validating the WS-ReliableMessaging sequence, with respect to the security context token scoping, is not described.

Typically, to use secure conversation and a security context token to secure reliable messaging, the following steps are involved;

- The WS-ReliableMessaging run time calls APIs from the Web Services Security run time to get the UUID of the security context token for the session and also the API for serializing and deserializing the security context token for managed persistent for reliable recovery.

  Because of the security nature of the security context token, the WS-ReliableMessaging protocol makes sure that the serialized security context token in persistent store is protected.

- If there is already a security context token established the UUID of the existing security context token is returned to WS-ReliableMessaging. If there is no security context token already established, the Web Services Security run time initiates a call to the recipient to establish the security context token.

  The latter case is similar to the Secure Conversation scenario.
- After the WS-ReliableMessaging run time acquires the UUID of the security context token, the WS-ReliableMessaging run time scopes the CreateSequence message to the security context token by using the SecurityTokenReference (STR) argument in the CreateSequence message and responds with the CreateSequenceResponse message.

  The exchange of the application messages is very similar to the WS-SecureConversation scenario.
- The WS-ReliableMessaging run time responds with the CreateSequenceResponse message.

  The exchange of the messages is very similar to the exchange in the WS-SecureConversation scenario.
- The WS-ReliableMessaging run time sends a SequenceAcknowledgement message to acknowledge that the message is properly delivered and secured by the security context token.
- Finally, the WS-ReliableMessaging run time sends a TerminateSequence message to terminate the sequence and is secured by the security context token.

Enabling the distributed cache using synchronous update and token recovery:

To support Secure Conversation in a cluster environment, the distributed cache stores the shared state information. Version 7.0 of WebSphere Application Server uses MBeans to improve synchronous update of the cache across the cluster. In addition, persistent token support is provided by storing the token data in a database.

About this task

Synchronous update of shared information in the distributed cache is implemented in version 7.0 of WebSphere Application Server using an MBean solution. When update of the shared state information across cluster members is required, a synchronous blocking call is issued to replicate the token state changes to all the servers in the cluster. This solution removes the limitations of using session affinity for secure conversation in a cluster environment.

Perform the following high-level steps to enable distributed cache and session affinity when using secure conversation for message-level protection in a cluster environment. To enable the distributed cache for the Security Context Token:

1. In the administrative console for WebSphere Application Server, click **Services** → **Security cache**.
2. Click the check box to select the **Distributed cache** option.
3. **Synchronous update of cluster members** is automatically selected. This enables the runtime to update all the cluster members with the updated token information synchronously. If this is selected, then session affinity does not have to be enabled.
4. If **Asynchronous update** is selected, then you must enable session affinity.
   a. Click the radio button to select **Token recovery support**.
   b. Select a data source (database) from the **Cell level data sources** menu list. If you need to add a database, click the **Manage data sources** link.
   c. Click **Apply** to enable token recovery support.
5. Click **OK**, then **Save**, to save the modified configuration.

Results

Token recovery support uses a JDBC database to store the token state. This provides failover support for high availability of the token.

Related tasks

“Enable distributed cache and session affinity when using Secure Conversation” on page 259
WebSphere Application Server provides message-level protection in a cluster environment. You can use Web Services Secure Conversation (WS-SecureConversation) for message-level protection of Java API for XML Web Services 2.0 (JAX-WS) Web services in a cluster environment.

Related reference

WSSCacheManagement command group for the AdminTask object
Use this topic as a reference for the commands for the WSSCacheManagement group of the AdminTask object. Use these commands with your administrative scripts to query, update, and remove distributed cache configuration data.

Secure conversation client cache:

For both distributed and local clients, the WebSphere Application Server secure conversation client cache stores tokens on the client.

WebSphere Application Server supports caching of the security context token for both the distributed client and local client. If the security context token is distributed, a client in the same replication domain uses the same security context token. Distributed caching also supports disk offload to save the security context token to disk for recovery.

To use the administrative console to modify the cache settings, click **Services** → **Security Cache**.

You can configure the cache settings, such as the following.

- Set the time that the token remains in the cache after timeout. The default value is 10 minutes. This value is a time window to renew an expired token.
- Set the renewal interval before the token expires. The default value is 10 minutes, and the minimum value is 3 minutes. Entering a number less than 3 minutes causes an error.

  Note: This setting is critical. This setting represents the maximum roundtrip time for a client to make a request, the transport request to go to the server, the server to process the request, and the transport response (if applicable) back to the client. If the time specified is too small and there is not enough time specified, then the token might expire during the roundtrip, and the client receives a failure response. If the time specified is too large, then performance diminishes.

  If the security context token is renewed too often, it might cause Web Services Secure Conversation (WS-SecureConversation) to fail or even cause an out-of-memory error to occur. It is required that you set the renewal interval before the token expires value for the Secure conversation client cache to a value less than the token timeout value for the security context token. It is also suggested that the token timeout value be at least two times the renewal interval before the token expires value.
- Select the distributed caching check box to support distributed clients. You must ensure that the WebSphere Application Server dynamic cache service, and cache replication, are enabled. For more information on enabling the dynamic cache service, refer to the topic Enable distributed cache and session affinity when using Secure Conversation.
- Define a custom property, edit, or remove existing custom properties.

The WS-SecureConversation client rejects a security context token that is issued at a future time. If you cannot synchronize the clock between the client machine and service machine, the clock skew could be configured to prevent the rejection of a valid token. The default clock skew is 3 minutes. To modify the default clock skew setting, add the following custom property to the desired minutes:

```
clockSkewToleranceInMinutes
```

Alternatively, use the wsadmin commands to manage secure conversation client cache configurations.

Thin client

For a Web Service application client running outside WebSphere Application Server, the security context token is cached only in the local Java process. The following system properties can be used to override the default cache setting on the thin client:

com.ibm.wsspi.wssecurity.SC.cache.cushion
Specifies the time in minutes to renew a security context token to be used with WS-SecureConversation on the client side so that the security context token has enough time to complete the downstream call. The default value is 10 minutes, and the minimum value is 3 minutes.

com.ibm.wsspi.wssecurity.SC.token.clockSkewTolerance
Specifies the tolerant clock skew time for a token between two machines. The default value is 3 minutes.

Web Services Secure Conversation:

Web Services Secure Conversation (WS-SecureConversation) provides a secured session for long running message exchanges and leveraging of the symmetric cryptographic algorithm.

WS-SecureConversation provides session-based security. Session-based security optimizes long message exchanges, as symmetric cryptography can be used to sign and encrypt the message. Typically, symmetric cryptographic algorithm is less CPU intensive than the asymmetric cryptography. Symmetric cryptographic algorithms should provide better performance and throughput when compared to the asymmetric cryptographic algorithms.

The symmetric cryptographic algorithm also provides a means to secure other session-based protocol and exchange patterns, such as Web Services Reliable Messaging (WS-ReliableMessaging).

Security context token for secure conversation

The Web Services Security specification defines the basic mechanisms for providing secure messaging. The Web Services Trust (WS-Trust) specification defines extensions to Web Services Security that provide ways to establish and broker trust relationships between two parties. The WS-Trust protocol defines the syntax of the request that can be sent to a security token service and the corresponding or subsequent response of the security token service. The security token service provided with WebSphere Application Server is called the *trust service*.

Using the WS-Trust protocol, a party can request the trust service issue a security context token (SCT). Then, this token can be used to establish a secure conversation (WS-SecureConversation). The request for a security token is sent to an application endpoint. The request is intercepted by the WebSphere Application Server and routed to the trust service.

A policy can be defined as the default for all trust issue operations, renew operations, validate operations, or cancel operations. Additionally, a policy can be attached to a specific URL and operation pair.

WS-SecureConversation defines extensions to allow security context establishment and sharing, and session key derivation, which allows contexts to be established and, potentially, more efficient keys, or new key material, to be exchanged. The WebSphere Application Server support for WS-Trust and WS-SecureConversation focuses on the issuing, renewing, validating, and cancelling of the security context token for secure conversation.

Policy set and bootstrap policy

In addition to describing these functions, the OASIS WS-SecureConversation draft submission describes multiple methods of establishing a secure session between the initiator and the recipient of the SOAP messages.

The bootstrap security policy is the security policy for the initiating party to acquire the security token for secure conversation from the trust service by using a token-issuing WS-Trust or WS-SecureConversation protocol message. The policy set configuration consists of the security policy for communication with the application service, and the bootstrap policy for communication with the trust service.

If sharing of a policy configuration (using WS-Policy) containing the secure conversation bootstrap policy fails, it may be because the bootstrap request and response policies differ. The message part protection for the bootstrap policy must be the same for both request and response bootstrap messages, because a single policy is published for both request and response.

What is supported for Web Services Secure Conversation

The following list highlights some of the key functions that are supported in WebSphere Application Server. The list is not exhaustive.

- A security context token (SCT) established between the initiating party and the recipient party.
- The WS-SecureConversation operations that are supported on the security context token (SCT), such as Issue token, Renew token, and Cancel token. Validate token is supported using WS-Trust protocol.
- A derived key (explicit and implied)

What is not supported for Web Services Secure Conversation

The following list highlights some of the key functions that are not supported in WebSphere Application Server. The list is not exhaustive.

- WS-SecureConversation does not support establishing a security context through the security context token that is created by an external security token service (trust component). However, WebSphere Application Server supports an internal security token service.
- WebSphere Application Server does not support establishing a security context through the security context token that is created by one of the communicating parties and propagated with a message.
- WebSphere Application Server does not support amending a security context token.
- WebSphere Application Server does not support a client creating the security context token.
- WebSphere Application Server provides no support for exchange and negotiation.

Secure conversation scenarios

The following scenarios describe the WS-SecureConversation functions that WebSphere Application Server supports:

- WS-SecureConversation

  This scenario is based on establishing a security context token with the recipient and using the derived key to sign and encrypt the message. It describes how to establish a security context by using session-based security. Session-based security is where the flow of the initiator establishes the security context token by using the WS-SecureConversation protocol with the recipient.
- WS-SecureConversation with WS-ReliableMessaging

  This scenario is a composite scenario that includes functions that are required for the composition scenario of Web Services Reliable Messaging (WS-ReliableMessaging), WS-SecureConversation, and WS-Trust. This scenario describes how to use WS-SecureConversation with WS-ReliableMessaging where the flow is similar to the previous scenario, but which is from the secure conversation prospective. However, the main difference is that the WS-ReliableMessaging sequence is secured with

the security context token and scopes the WS-ReliableMessaging sequence to the security context token. This description focuses on the message exchanges that are using the security context token in the overall flow.

Scoping of Web Services Secure Conversation:

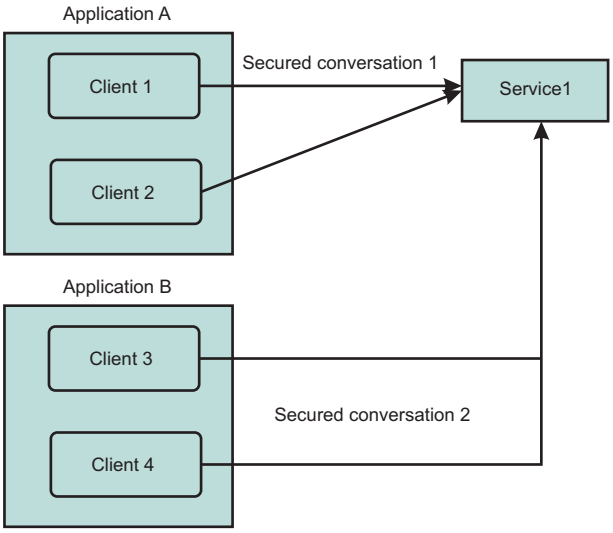
Web Services Secure Conversation supports two scoping mechanisms: the default and the Java API for XML Web Services (JAX-WS) client service level.

Review the following information about the two scoping mechanisms to ensure the proper scoping of secure conversation and policy set for WebSphere Application Server.

Default

The default scope is based on a cluster, an application, a module, and a target service endpoint. For a client running in a thin client environment, it is considered to be a single application, cluster, and module.

In this scoping mode, all the instances of the JAX-WS client within a particular application, cluster, and module to the same target service endpoint share the same secure conversation. For example, in the following figure, the two client instances (Client 1 and Client 2) are in the same module. Client 1 and Client 2 share the same secured conversation with Service 1. The other two client instances (Client 3 and Client 4), which are in a different module than Clients 1 and 2 and which share a secured conversation with each other but not with Clients 1 and 2.



JAX-WS client service level

Scope at the JAX-WS client service level is enabled by specifying a property in the token generator binding configuration of the Secure Conversation Token (SCT)in the client application request (application outbound). The binding is located in the META-INF of the deployed application.

For example, if the application is `WSSampleClientBeta.ear` and the binding directory is `SecureConversation123binding`, the binding file would be located at:

```
$PROFILE_DIR/config/cells/<cellname>/WSSampleClientBeta.ear/deployments/WSSampleClientBeta
/META-INF/SecureConversation123binding/PolicyTypes/WSSecurity/bindings.xml.
```

An example of the configuration follows:

```
<tokenGenerator name="gen_enctgen"
      classname="com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenGenerator">
   <valueType localName="http://schemas.xmlsoap.org/ws/2005/02/sc/sct" uri="" />
      <callbackHandler classname="com.ibm.ws.wssecurity.impl.auth.callback.WSTrustCallbackHandler">
         <properties name="com.ibm.ws.wssecurity.sc.SCTScope" value="SERVICE_SCOPE"/>
      </callbackHandler>
         <properties name="com.ibm.ws.wssecurity.sc.dkt.ServiceLabel" value="WSC"/>
         <properties name="com.ibm.ws.wssecurity.sc.dkt.ClientLabel" value="WSC"/>
            <jAASConfig configName="system.wss.generate.sct"/>
</tokenGenerator>
```

The following code example demonstrate the behavior after the property in the token generator binding configuration of the SCT in the client application request (application outbound) is enabled. In this mode, Web Services Secure Conversation is scoped at the JAX-WS client service instance.

```
QName serviceQname = new QName("http://ws.apache.org/axis2", "EchoService");
QName portQname = new QName("http://ws.apache.org/axis2", "EchoServicePort");
String endpointUrl = "http://myhost/......";
Service svc1 = Service.create(serviceQname);
svc1.addPort(portQname, null, endpointUrl);
Dispatch<Source> dispatch = svc1.createDispatch(portQname, Source.class, null);
......
......
Service svc2 = Service.create(serviceQname);
svc2.addPort(portQname, null, endpointUrl);
Dispatch<Source> dispatch = svc2.createDispatch(portQname, Source.class, null);
```

where `svc1` and `svc2` are in two different secure conversations with the target service endpoint.

You can change the scope by using either the administrative console or by using scripting to add a property.

Derived key token:

After establishing the security context and after the secret have been established (authenticated), derived keys can be used to sign and encrypt the SOAP message to provide message level protection. You can then use derived keys for each key that is used in the security context.

You can enable Web Services Secure Conversation (WS-SecureConversation) by using symmetric keys that are derived from the security token for signing and encrypting the application messages.

Using WS-SecureConversation, the initiator can establish a security context token using the Web Services Trust (WS-Trust) protocol with the recipient. A security context token implies or contains a shared secret. Using a common secret, different key derivations can be defined. Then, using the security context token, the <wsc:DerivedKeyToken> token can be used to derive keys from any security token that has a shared secret, key, or key material. This secret can be used for signing or encrypting messages, but it is recommended that derived keys be used for signing and encrypting messages that are associated only with the security context.

Syntax for the <wsc:DerivedKeyToken> element

The <wsc:DerivedKeyToken> element is used to indicate that the key for a specific reference is generated from the function so that explicit security tokens, secrets, or key material need not be exchanged as often. The derived key token does not support references to it using key identifiers or key names. All references must use an ID to a *wsu:Id* attribute or use a URI reference, <wsse:Reference>, to the <wsc:Identifier> element in the security context token.

The syntax for <wsc:DerivedKeyToken> element is as follows:

```
<wsc:DerivedKeyToken wsu:Id="...">
    <wsse:SecurityTokenReference>...</wsse:SecurityTokenReference>
    <wsc:Label>...</wsc:Label>
    <wsc:Nonce>...</wsc:Nonce>
  </wsc:DerivedKeyToken>
```

Derived keys are expressed as security tokens and use different algorithms for deriving keys. The following URI is used to represent the derived key token type:

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk
```

The nonce is processed as a binary octet sequence (the value prior to base64 encoding). The nonce seed is required, and must be generated by one or more of the communicating parties. Use separate nonces and have independently generated keys for signing and encrypting for request and response. New keys should be derived for each message, meaning that a previous nonce should not be reused.

Implied derived key generation

Implied derived keys define a shortcut mechanism for referencing certain types of derived keys. Specifically, an @wsc:Nonce attribute can be added to the security token reference (STR) that is defined in the WS-Security specification. When present, an implied derived key indicates that the key is not in the referenced token but, instead, is a key that is derived from the key or secret of the referenced token. It is recommended that you do not use implied derived Keys in the <wsc:DerivedKeyToken> element.

The following example illustrates a message that is sent using two derived keys, one for signing and one for encrypting:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
    xmlns:xenc="..." xmlns:wsc="..." xmlns:ds="...">
  <S11:Header>
    <wsse:Security>
      <wsc:SecurityContextToken wsu:Id="ctx2">
        <wsc:Identifier>uuid:...UUID2...</wsc:Identifier>
      </wsc:SecurityContextToken>
      <wsc:DerivedKeyToken wsu:Id="dk2">
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#ctx2"/>
        </wsse:SecurityTokenReference>
        <wsc:Nonce>KJHFRE...</wsc:Nonce>
      </wsc:DerivedKeyToken>
      <xenc:ReferenceList>
        ...
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#dk2"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
          ...
      </xenc:ReferenceList>
      <wsc:SecurityContextToken wsu:Id="ctx1">
        <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
      </wsc:SecurityContextToken>
      <wsc:DerivedKeyToken wsu:Id="dk1">
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#ctx1"/>
        </wsse:SecurityTokenReference>
        <wsc:Nonce>KJHFRE...</wsc:Nonce>
      </wsc:DerivedKeyToken>
      <xenc:ReferenceList>
        ...
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#dk1"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        ...
      </xenc:ReferenceList>
```

```
    </wsse:Security>
  ...
  </S11:Header>
  <S11:Body>
    ...
  </S11:Body>
</S11:Envelope>
```

Web Services Secure Conversation standard:

Web Services Secure Conversation (WS-SecureConversation) is a proposed Organization for the Advancement of Structured Information Standards (OASIS) standard that defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation.

The base Web Services Security (WS-Security) standard from OASIS defines how to digitally sign and encrypt the SOAP message to provide message level protection. The standard also defines how to attach and reference a security token for digital signature and encryption. However, it does not provide session-based protection when a long series of related messages were exchanged. The WS-Security specification focuses on the *message authentication model*. This approach, while useful in many situations, could be subject to several forms of attack.

The WS-SecureConversation specification introduces the concept of a security context and its usage. The security context token is a new WS-Security token type that represents the security context abstract concept. The token is identified by a URI and consists of negotiated keys as well as other security related properties. The *context authentication model* authenticates a series of messages and, therefore, addresses these concerns. The context authentication model increases the overall performance and security of the subsequent exchanges, but it requires additional communications when authentication happens prior to normal application exchanges.

Version 1.0 of the OASIS WS-SecureConversation specification defines extensions that build on the Web Services Security (WS-Security) and Web Services Trust (WS-Trust) standards to provide secure communication across one or more messages.

IBM, Microsoft, and other vendors have been working on the WS-SecureConversation specification since 2004. A draft of this document was jointly published in February, 2005. The WS-SecureConversation draft was submitted to the OASIS Web Service Secure Exchange Technical Committee (WS-SX TC), which was formed in December 2005, along with Web Services Trust (WS-Trust) and Web Services Security Policy (WS-SecurityPolicy) drafts in order to begin the standardization process.

A revised Version 1.1 draft version of the WS-SecureConversation specification standard was submitted to OASIS in February 2005 and further defines the extensions in Version 1.0. This specification defines extensions to allow security context establishment and sharing, and session key derivation. These extensions allow contexts to be established and potentially more efficient keys or new key material to be exchanged.

The most recent version of the specification standard is version 1.3, which was approved by the WS-SX TC on March 1, 2007. Key requirements in this level of the specification include derived keys and per-message keys, and extensible security contexts. Version 7.0 of WebSphere Application Server adds support for version 1.3 of WS-SecureConversation, providing improved error handling using the standard fault codes as defined in the specification.

The Web Services Secure Conversation (WS-SecureConversation) standard is a building block that is used in conjunction with the other Web service and application-specific protocols such as Web Services Security and Web Services Trust to accommodate a wide variety of security models and technologies. WS-SecureConversation is built on top of the WS-Security and WS-Trust models to provide secure

communication between services. The WS-SecureConversation draft specification describes how to establish a security context token between two parties, and the WS-Trust specification describes how to issue and exchange security tokens.

This WS-SecureConversation draft specification includes extensions to Web services security and:

- Describes the security context token.
- Defines how security contexts are established.
- Describes how security contexts are amended, renewed, and cancelled. Amending context is not supported by WebSphere Application Server.
- Specifies how derived keys are computed.
- Specifies how to associate a specific security context with an action, if multiple security contexts exist.

WebSphere Application Server supports the client establishing a secured conversation with the target service endpoint.

WebSphere Application Server supports the OASIS Version 1.1 submission draft, which became available in February 2005. The WebSphere Application Server does not support all of the functions in the submission draft. WebSphere Application Server support of WS-SecureConversation focuses on:

- A security context token that is established between the initiating party and the recipient party.
- The operations that are supported on security context token, such as Issue token, Renew token, and Cancel token.
- The derived key (both explicit and implied)

Secure conversation provided with WebSphere Application Server does not provide support for a security context token (SCT) that is acquired from a third-party trust server, and does not provide support for a security context token that is created by the client.

For information about WS-SecureConversation:

- See the IBM DeveloperWorks Web site.
- See the schema for this specification: WS-SecureConversation schema
- Refer to the following namespace prefixes that are used for WS-SecureConversation: http://schemas.xmlsoap.org/ws/2005/02/sc and http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512

Configuring the token generator and token consumer to use a specific level of WS-SecureConversation:

Use the administrative console to configure the token generator or token consumer to use a specific level of the WS-SecureConversation OASIS specification standard. Select one of the two levels of token types supported: Secure Conversation Token v200502, or Secure Conversation Token v1.3.

About this task

WebSphere Application Server supports two levels of the OASIS standard for WS-SecureConversation including both the submission draft version (February 2005 draft specification) and version 1.3 of the standard, which was approved on March 1, 2007. Using the administrative console, configure the token generator so that the appropriate token type for a specific level of the standard is issued when a security token is requested.

1. Log on to the administrative console and navigate to the panel where the token generator is configured by clicking **Services** → **Policy sets** → **General provider policy set bindings** or **General client policy set bindings**.
2. Click on the name of the binding you want to edit.
3. Click the **WS-Security** policy in the Policies table.

4. Click the **Authentication and protection** link in the Main message security policy bindings section.
5. Click **New token** to create a new token generator or consumer, or click an existing token link from the Protection Tokens table.
6. Enter a token name, then use the Token type drop-down menu to select a secure conversation token type.
   - To specify a submission draft token type, select **Secure Conversation Token v200502**.
   - To specify a version 1.3 token type, select **Secure Conversation Token v1.3**.
7. The local name is populated according to the token type you selected, as follows:
   - Local name for the submission draft token type: http://schemas.xmlsoap.org/ws/2005/02/sc/sct
   - Local name for the version 1.3 token type: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512

   The URI field is also filled in based on the token type.
8. Click to deselect the option **Tolerate Secure Conversation Token v200502** if you want to enforce use of only the version 1.3 tokens. This option specifies whether the provider should handle both Secure Conversation Token version 1.3 and Secure Conversation Token v200502. By default, the provider handles both versions.
9. Click **Apply** to create a secure conversation token of the selected type.

Related reference

Protection token settings (generator or consumer)
Use this page to configure protection tokens. Protection tokens sign messages to protect integrity or encrypt messages to provide confidentiality.

Trust service:

The security token service that is provided by WebSphere Application Server is called the trust service. The WebSphere Application Server trust service uses the secure messaging mechanisms of Web Services Trust (WS-Trust) to define additional extensions for the issuance, exchange, and validation of security tokens.

Web Services Trust (WS-Trust) is an OASIS standard that enables security token interoperability by defining a request/response protocol. This protocol allows SOAP actors, such as a Web services client, to request of some trusted authority that a particular security token be exchanged for another.

WebSphere Application Server is not providing a full security token service that implements all the contents of the WS-Trust draft specification. The WebSphere Application Server support of WS-Trust focuses on establishing a security context token for secure conversation. Version 7.0 of WebSphere Application Server supports many of the security features described in version 1.3 of the WS-Trust OASIS standard, dated March 19, 2007.

Third party WS-Trust client

WebSphere Application Server does not provide a WS-Trust client implementation. You can choose to use a third-party WS-Trust-enabled client but, if you do, WebSphere Application Server does not support a third-party trust-enabled client. A trust client can facilitate the generation of these soap messages and the processing of the response, but the client is not required.

WebSphere Application Server focuses on the issuing, renewing, and canceling of the security context token for Web Services Secure Conversation (WS-SecureConversation).

The WS-Trust specification must be followed to make requests of the trust service. This specification includes the use of Web Services Addressing (WS-Addressing) headers. The WS-Addressing headers are specified in both the August 2004 or the August 2005 specifications. Per the specification, the SOAP body

must consist of a single RequestSecurityToken (RST) element. This element can contain sub-elements as defined in the WS-Trust and WS-SecureConversation specifications.

You can secure the WS-Trust SOAP messages by using the bootstrap policy that is defined in the policy set. The bootstrap security policy is invoked in the process of an initiator establishing communication with an application service. Initial requests to services other than the application service are secured by using the bootstrap policy. These initial requests typically involve one or more requests to a security token service (STS), such as the WebSphere Application Server trust service. An example of a request might be acquiring the security context token necessary for WS-SecureConversation. An *initiator* is the role that initiates the original request and, in most cases, it is the client. The client bootstrap policy set must correspond to the trust service issue and renew attached policy sets for the endpoint. The trust service cancel and validate attached policy sets for the endpoint must correspond to the client's application policy set.

Websphere Application Server provides two ways to secure SOAP messages that are destined for the trust service. One way is to use the bootstrap policy that is defined in the policy set. A second way is to use the Web Services Security API (WSS API). Your application might use the WSS API to acquire the security context token for the programmatic, API-based WS-SecureConversation.

For Secure Conversation, a request from the client to an endpoint service is suspended while a new (second) request is generated and processed by the trust service. The security context token returned with the second request is used to derive keys that secure communications with the service.

High-level trust service functions

The following list includes WS-Trust-related functions that are currently supported in WebSphere Application Server. The list is not exhaustive and it focuses only on the high-level functions.

- The trust service component is embedded into and available on each WebSphere Application Server that processes the WS-Trust protocol messages.
- Communication is accomplished through the RequestSecurityToken (RST), RequestSecurityTokenCollection (RSTC), RequestSecurityTokenResponse (RSTR), and RequestSecurityTokenResponseCollection (RSTRC).

  Note: An RST request can be made to an external security token service (trust service). However, the restriction is that security context token, which is needed for WS-SecureConversation, must be provided by the WebSphere Application Server trust service.
- A security policy for each of the WS-Trust operations (issue, cancel, validate, and renew).
- Pre-configured Security Context Token provider which issues tokens for specific URL.
- Specification of a token provider's token-specific parameters (for example, expiration time).
- A security context token for WS-SecureConversation.
- Caching support for the security context token in both cluster and non-cluster environments. WebSphere Application Server issues security context tokens when requested if the request meets the security requirements. However, WS-SecureConversation provided by WebSphere Application Server only processes security context tokens that are issued by WebSphere Application Server.
- Note that WebSphere Application Server trust service only supports the security context token.
- Trust service supports both the Submission specification (2004/08) and the final specification (2005/08) versions of WS-Addressing.
- Trust service uses a default policy set called TrustServiceSecurityDefault, which includes WS-Security and WS-Addressing and provides default security for the issue and renew operations.
- Trust service uses a second default policy set called TrustServiceSymmetricDefault, which includes WS-Security and WS-Addressing and provides default security for the cancel and validate operations.

Trust service functions that are not supported

The following high-level WS-Trust functions that are not supported in WebSphere Application Server. The list is not exhaustive, and the list focuses only on key functions:

- No negotiation and exchange protocols are supported.
- No other token types are currently supported out of the box; only the security context token is supported.
- No Trust10 specifications from WS-SecurityPolicySet are supported.
- No unsolicited RequestSecurityTokenResponse (RSTR) is supported.
- A Request Security Token (RST) request cannot be issued to an external security token service (STS) to establish a secure conversation; only the embedded trust service is currently supported.
- Policy requests that are contained in the RST are not honored.
- The ability to amend a token (the amend operation) is not supported.
- A dedicated external endpoint for access to the token service is not supported; only the embedded trust service is currently supported.
- The trust services does not support the entropy element that contains an EncryptedKey.
- Delegation and forwarding are not supported.
- The OnBehalfOf element is not supported.
- The Key Exchange Token (KET) binding is not supported.

Trust service operations

WebSphere Application Server specifically supports the ability of the trust service, on behalf of the endpoint, to issue a security context token for WS-SecureConversation. The token-issuing support is currently limited only to the security context token. There is also trust policy management for defining a policy for the trust service to issue, cancel, validate, or renew tokens.

The token service supports the WS-Trust schema namespace. Within this namespace the following actions are supported:

- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Cancel
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Validate
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Renew

The token service also supports the WS-SecureConversation schema namespace. Within this namespace the following actions are supported:

- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew

An inbound RST for the security context token issue operation must contain an Entropy element. The Entropy element must contain a BinarySecret. The trust services does not support the Entropy element that contains an EncryptedKey.

Note that the trust service does not support unsolicited RSTR actions. In addition, the ability to amend a token is not supported by WebSphere Application Server. Also, see the section titled Trust service functions that are not supported.

Trust policy set-related files

The default trust service policy set for issue and renew is TrustServiceSecurityDefault. You can set up the corresponding policy set and binding for each service endpoint URL.

Security context token:

Web Services Trust (WS-Trust) and Web Services Secure Conversation (WS-SecureConversation) support in the application server provides the ability to issue a security context token (SCT). Requests for a security context token are processed by the security token service.

The security token service for WebSphere Application Server is called the trust service. However, the application server does not provide a full security token service that implements all the contents of the WS-Trust specification.

The secure session is referred to as *secure conversation* because the message protocols that are used are defined by WS-SecureConversation and WS-Trust. WebSphere Application Server supports secure conversation.

To request a security context token, a RequestSecurityToken (RST), which is defined by WS-Trust and WS-SecureConversation protocols, is sent to the service endpoint to which you are setting up a secure conversation. These requests are transparently rerouted to the trust service. The trust service processes the RST and responds with a RequestSecurityTokenResponse (RSTR). This response is returned to the requestor as if it was generated by the endpoint service.

The WebSphere Application Server token provider support is limited to the Security Context Token provider. WS-SecureConversation in the application server focuses on the establishing of the security context token between the initiating party and the recipient party for secure conversation.

WebSphere Application Server includes caching support for the Security Context Token in both cluster and non-cluster environments as well as on both the client and server. WebSphere Application Server also provides trust policy set management for each of the trust service operations: issue, cancel, validate, and renew. Trust system policy sets can be managed for each of these trust operations relative to an explicit service endpoint or the trust service default. The default trust service policy set for a trust operation is enforced when there is not an explicit attachment.

See the information about Web Services Trust for the WS-Trust functions that are supported.

For the security context token, you can:

- Configure the security context token provider for WS-SecureConversation, providing issue, renew and cancel operations.
- Configure the trust service to issue a security context token for access to a specific endpoint service (target).
- Configure the security requirements for access to the trust service and applications. WebSphere Application Server provides pre-configured application policy sets and trust service policy sets to assist with this configuration.
- Define a system policy for each of the four trust service operations: issue, cancel, validate, and renew. These policies are configured for the default or a specific endpoint service. Note that the amend operation is not supported.

The Security Context Token provider does not support the following operations:

- WS-SecureConversation amend
- Negotation to establish Secure Conversation
- WS-Trust key exchange requests

- Client-initiated RequestSecurityTokenResponse (RSTR) and RequestSecurityTokenResponseCollection (RSTRC) requests
- WS-SecurityPolicy trust assertions

Definitions

To better understand security tokens, the following terms are defined:

security token
A security token represents a collection of claims.

security context
A security context is an abstract concept that refers to an established authentication state and negotiated key or keys that can have additional security-related properties. A security context needs to be created and shared by the communicating parties before being used. A security context is shared among the communicating parties for the lifetime of a communications session and a security context token is the wire representation of this abstract security context.

WebSphere Application Server does not support a security context token created by one of the communicating parties and propagated with a messageWebSphere Application Server does not support creating a security context token through negotiation and exchanges.

security context token
A security context token is a wire representation of that security context abstract concept, which allows a context to be named by a URI and to be used with Web services security. A secured communication with a security context token between two parties is realized with WS-Trust and WS-SecureConversation.

security token service
A security token service (STS) is a Web service that issues security tokens, meaning it makes assertions that are based on evidence that it trusts, to whoever trusts it (or to specific recipients).

Trust service
The trust service is the security token service and supporting code that is provided by Websphere Application Server.

RequestSecurityToken (RST)
A RST is a message sent to a security token service to request a security token.

RequestSecurityToken Response (RSTR)
A RSTR is a response to a request for a security token from a security token service to a requestor after receiving an RST message.

To communicate trust, a service requires proof, such as a signature, to prove knowledge of a security token or set of security tokens. A service itself can generate tokens or it can rely on a separate security token service to issue a security token with its own trust statement. Note that, for some security token formats, communicating trust can just be a re-issuance or a co-signature that forms the basis of trust brokering.

Syntax for the <wsc:SecurityContextToken> element

A security context is shared among the communicating parties for the lifetime of a communications session and a security context token is the wire representation of this abstract security context.

In the WS-SecureConversation specification, a security context is represented by the <wsc:SecurityContextToken> security token. The following URI represents the security context token type that is required to establish a secure conversation.

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
```

The syntax for <wsc:SecurityContextToken> element is as follows:

```
<wsc:SecurityContextToken wsu:Id="..." ...>
    <wsc:Identifier>...</wsc:Identifier>
    <wsc:Instance>...</wsc:Instance>
    ...
</wsc:SecurityContextToken>
```

The security context token does not support references to it by using key identifiers or key names. All references must use an ID (to a *wsu:Id* attribute) or use a URI reference, <wsse:Reference>, to the <wsc:Identifier> element in the security context token.

RST and RSTR examples to issue a security token

This example shows a RST request to issue a security token. The URI `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct`, which is used in this example, represents the token type:

```
<wsc:SecurityContextToken>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
 xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">

   <soapenv:Header>
   <wsse:Security
        xmlns:wsse="http://docs.oasis-open.org/wss/
                    2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                    soapenv:mustUnderstand="1">
        <wsse:UsernameToken><wsse:Username>user1</wsse:Username>
             <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/
                  oasis-200401-wss-username-token-profile-1.0#PasswordText">security
             </wsse:Password>
        </wsse:UsernameToken>
   </wsse:Security>
        <wsa:To>http://localhost:8080/WSSample/services/EchoService
        </wsa:To>
        <wsa:ReplyTo>
            <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous
            </wsa:Address>
        </wsa:ReplyTo>
        <wsa:MessageID>urn:uuid:646268CB30A01B89D811537688997954
        </wsa:MessageID>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT
        </wsa:Action>
   </soapenv:Header>
   <soapenv:Body>
    <wst:RequestSecurityToken Context="http://www.ibm.com/login/">
      <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</wst:RequestType>
   <wst:TokenType>http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct</wst:TokenType>
      <wst:Entropy>
         <wst:BinarySecret>swYVsjsi75fB+RksmDdWKQ==</wst:BinarySecret>
      </wst:Entropy>
          <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
             <wsa:EndpointReference>
                <wsa:Address>WSSample/services/EchoService</wsa:Address>
             </wsa:EndpointReference>
          </wsp:AppliesTo>
       </wst:RequestSecurityToken>
    </soapenv:Body>
</soapenv:Envelope>
```

This example shows a RSTR request to issue a security token:

```
<soapenv:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
        http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT
    </wsa:Action>
    <wsa:RelatesTo>
        97fd1ce790c257f0:1ea9f29c:1129642ebe1:-7fff
    </wsa:RelatesTo>
 </soapenv:Header>
  <soapenv:Body>
      <wst:RequestSecurityTokenResponse
           xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
           Context="http://www.ibm.com/login/">
      <wst:RequestedSecurityToken
           xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
```

```
        <wsc:SecurityContextToken
             xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                        oasis-200401-wss-wssecurity-utility-1.0.xsd"
             wsu:Id="uuid:617A2281DAD3C3EC211179342073467">
             <wsc:Identifier xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc">
                 uuid:617A2281DAD3C3EC211179342073466
             </wsc:Identifier>
             <wsc:Instance xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc">
              uuid:617A2281DAD3C3EC211179342073465
             </wsc:Instance>
        </wsc:SecurityContextToken>
      </wst:RequestedSecurityToken>

  <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
   <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
                 http://localhost:9080/WSSampleSei/EchoService
        </wsa:Address>
        </wsa:EndpointReference>
   </wsp:AppliesTo>
       <wst:RequestedProofToken xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
          <wst:ComputedKey xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
                http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
          </wst:ComputedKey>
        </wst:RequestedProofToken>
        <wst:Entropy xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
             <wst:BinarySecret xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
                        Type="http://schemas.xmlsoap.org/ws/2005/02/trust/Nonce">
                 0oK29up5fifaCkPiSX3GZg==
             </wst:BinarySecret>
        </wst:Entropy>
       <wst:Lifetime xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
           <wsu:Created
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-utility-1.0.xsd">
                 2007-05-16T19:01:12.625Z
          </wsu:Created>
          <wsu:Expires
           xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-utility-1.0.xsd">
                 2007-05- 16T21:01:12.625Z
          </wsu:Expires>
        </wst:Lifetime>
        <wst:RequestedAttachedReference
           xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
          <wsse:SecurityTokenReference
           xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-secext-1.0.xsd">
            <wsse:Reference
                  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-secext-1.0.xsd"
                  URI="#uuid:617A2281DAD3C3EC211179342073467"
                  ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct">
            </wsse:Reference>
          </wsse:SecurityTokenReference>
        </wst:RequestedAttachedReference>
        <wst:RequestedUnattachedReference
              xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
          <wsse:SecurityTokenReference
               xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                           oasis-200401-wss-wssecurity-secext-1.0.xsd"
              <wsse:Reference
                    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                                oasis-200401-wss-wssecurity-secext-1.0.xsd"
                    URI="uuid:617A2281DAD3C3EC211179342073466"
                    ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct">
              </wsse:Reference>
           </wsse:SecurityTokenReference>
        </wst:RequestedUnattachedReference>
        <wst:Renewing
             xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
             Allow="true" OK="false">
        </wst:Renewing>
        <wst:KeySize
             xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        128
        </wst:KeySize>
     </wst:RequestSecurityTokenResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

RST and RSTR examples to cancel a security token

This example shows a RST request to cancel a security token.

```
<soapenv:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsa:To>
   http://newchina.austin.ibm.com:9080/WSSecConvApis03/FVTVersionSecConvApis03Service
  </wsa:To>
  <wsa:MessageID>
   f20b218a24bf43df:-57ea847:112b47ead6d:-7ffc
  </wsa:MessageID>
  <wsa:Action>
   http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Cancel
  </wsa:Action>
 </soapenv:Header>
 <soapenv:Body>
      <wst:RequestSecurityToken
         xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
         Context="http://www.ibm.com/login/">
      <wst:RequestType
         xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
         http://schemas.xmlsoap.org/ws/2005/02/trust/Cancel
      </wst:RequestType>
      <wst:CancelTarget
          xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
          <wsc:SecurityContextToken
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-utility-1.0.xsd"
            xmlns:Id="http://docs.oasis-open.org/wss/2004/01/
                       oasis-200401-wss-wssecurity-utility-1.0.xsd"
            xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc"
            Id:Id="uuid:3FF175272DA6F83A291179849257996">
            <wsc:Identifier
                xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc">
                uuid:3FF175272DA6F83A291179849257985
            </wsc:Identifier>
            <wsc:Instance
                xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc">
                uuid:3FF175272DA6F83A291179849257984
            </wsc:Instance>
           </wsc:SecurityContextToken>
      </wst:CancelTarget>
   <wst:TokenType
       xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
   </wst:TokenType>
   <wsp:AppliesTo
       xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
       <wsa:EndpointReference
           xmlns:wsa="http://www.w3.org/2005/08/addressing">
           <wsa:Address
               xmlns:wsa="http://www.w3.org/2005/08/addressing">
               http://newchina.austin.ibm.com:9080/WSSecConvApis03
                           /FVTVersionSecConvApis03Service
           </wsa:Address>
       </wsa:EndpointReference>
   </wsp:AppliesTo>
   <wst:Entropy
       xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
       <wst:BinarySecret
           xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
           Type="http://schemas.xmlsoap.org/ws/2005/02/trust/Nonce">
           Tv6pDe6Or3grjd7t+GGCZg==
       </wst:BinarySecret>
   </wst:Entropy>
        <wst:KeySize
            xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
            128
            </wst:KeySize>
       </wst:RequestSecurityToken>
    </soapenv:Body>
</soapenv:Envelope>
```

This example shows a RSTR request to cancel a security token:

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <soapenv:Header>
        <wsa:Action>
```

```
          http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Cancel
       </wsa:Action>
       <wsa:RelatesTo>
           f20b218a24bf43df:-57ea847:112b47ead6d:-7ffc
       </wsa:RelatesTo>
    </soapenv:Header>
     <soapenv:Body>
       <RequestSecurityTokenResponse
            Context="http://www.ibm.com/login/">
          <wst:RequestedTokenCancelled
              xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
          </wst:RequestedTokenCancelled>
       </RequestSecurityTokenResponse>
     </soapenv:Body>
</soapenv:Envelope>
```

RST and RSTR examples to renew a security token

This example shows a RST request to renew a security token.

```
<soapenv:Envelope
          xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <soapenv:Header>
       <wsa:To>
           http://synctest.austin.ibm.com:9080/WSTrust03/FVTVersionTrust03Service
       </wsa:To>
       <wsa:MessageID>
           urn:uuid:85f87aad1772f485:-5f8ede69:112bbe15ec7:-7ffd
       </wsa:MessageID>
       <wsa:Action>
           http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Renew
       </wsa:Action>
    </soapenv:Header>
    <soapenv:Body>
       <wst:RequestSecurityToken Context="http://www.ibm.com/login/"
           xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
           <wst:RequestType>
               http://schemas.xmlsoap.org/ws/2005/02/trust/Renew
           </wst:RequestType>
           <wst:RenewTarget>
               <wsc:SecurityContextToken
                   Id:Id="uuid:C4E1EB7F485526962E1179973151233"
                         xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc"
                         xmlns:Id="http://docs.oasis-open.org/wss/2004/01/
                              oasis-200401-wss-wssecurity-utility-1.0.xsd"
                   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                              oasis-200401-wss-wssecurity-utility-1.0.xsd">
                  <wsc:Identifier>
                       uuid:C4E1EB7F485526962E1179973151216
                  </wsc:Identifier>
                  <wsc:Instance>
                       uuid:C4E1EB7F485526962E1179973151215
                  </wsc:Instance>
               </wsc:SecurityContextToken>
           </wst:RenewTarget>
           <wst:TokenType>
               http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
           </wst:TokenType>
         </wst:RequestSecurityToken>
    </soapenv:Body>
</soapenv:Envelope>
```

This example shows a RSTR request to renew a security token:

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <soapenv:Header>
        <wsa:Action>
            http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Renew
        </wsa:Action>
        <wsa:RelatesTo>
            urn:uuid:85f87aad1772f485:-5f8ede69:112bbe15ec7:-7ffd
        </wsa:RelatesTo>
 </soapenv:Header>
 <soapenv:Body>
        <wst:RequestSecurityTokenResponse
            Context="http://www.ibm.com/login/"
            xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
       <wst:RequestedSecurityToken>
           <wsc:SecurityContextToken
```

```
            wsu:Id="uuid:C4E1EB7F485526962E1179974951825"
            xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                           oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsc:Identifier>
                uuid:C4E1EB7F485526962E1179973151216
            </wsc:Identifier>
            <wsc:Instance>
                uuid:C4E1EB7F485526962E1179974951824
            </wsc:Instance>
         </wsc:SecurityContextToken>
      </wst:RequestedSecurityToken>
      <wst:Entropy>
          <wst:BinarySecret>
              zGIWpvaUZ55+W11GroEWHA==
          </wst:BinarySecret>
      </wst:Entropy>
      <wst:Lifetime>
          <wsu:Created
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                             oasis-200401-wss-wssecurity-utility-1.0.xsd">
              2007-05-24T02:49:10.187Z
          </wsu:Created>
          <wsu:Expires
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                             oasis-200401-wss-wssecurity-utility-1.0.xsd">
              2007-05-24T02:59:10.187Z
          </wsu:Expires>
       </wst:Lifetime>
  <wst:RequestedAttachedReference>
      <wsse:SecurityTokenReference
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                         oasis-200401-wss-wssecurity-secext-1.0.xsd">
          <wsse:Reference
               URI="#uuid:C4E1EB7F485526962E1179974951825"
               ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct">
          </wsse:Reference>
      </wsse:SecurityTokenReference>
  </wst:RequestedAttachedReference>
      <wst:Renewing Allow="true" OK="true"></wst:Renewing>
      </wst:RequestSecurityTokenResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

RST and RSTR examples to validate a security token

This example shows a RST request to validate a security token.

```
<soapenv:Envelope
       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <soapenv:Header>
     <wsa:To>
        http://synctest.austin.ibm.com:9080/WSTrust03/FVTVersionTrust03Service
     </wsa:To>
     <wsa:MessageID>
         urn:uuid:85f87aad1772f485:-5f8ede69:112bbe15ec7:-7fff
     </wsa:MessageID>
     <wsa:Action>
        http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Validate
     </wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
      <wst:RequestSecurityToken Context="http://www.ibm.com/login/"
          xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        <wst:RequestType>
            http://schemas.xmlsoap.org/ws/2005/02/trust/Validate
        </wst:RequestType>
        <wst:ValidateTarget>
           <wsc:SecurityContextToken
             Id:Id="uuid:C4E1EB7F485526962E1179973151233"
             xmlns:wsc="http://schemas.xmlsoap.org/ws/2005/02/sc"
             xmlns:Id="http://docs.oasis-open.org/wss/2004/01/
                     oasis-200401-wss-wssecurity-utility-1.0.xsd"
             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                     oasis-200401-wss-wssecurity-utility-1.0.xsd">
             <wsc:Identifier>
                 uuid:C4E1EB7F485526962E1179973151216
              </wsc:Identifier>
              <wsc:Instance>
                 uuid:C4E1EB7F485526962E1179973151215
              </wsc:Instance>
```

```
          </wsc:SecurityContextToken>
        </wst:ValidateTarget>
        <wst:TokenType>
            http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
        </wst:TokenType>
      </wst:RequestSecurityToken>
    </soapenv:Body>
</soapenv:Envelope>
```

This example shows a RSTR request to validate a security token:

```
<soapenv:Envelope
       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <soapenv:Header>
      <wsa:Action>
         http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Validate
      </wsa:Action>
      <wsa:RelatesTo>
         urn:uuid:85f87aad1772f485:-5f8ede69:112bbe15ec7:-7fff
      </wsa:RelatesTo>
    </soapenv:Header>
    <soapenv:Body>
        <RequestSecurityTokenResponse
              Context="http://www.ibm.com/login/">
         <wst:Status
              xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
           <wst:Code>
            http://schemas.xmlsoap.org/ws/2005/02/trust/status/valid
           </wst:Code>
         </wst:Status>
      </RequestSecurityTokenResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Review the two example scenarios that discuss establishing the security context token.

System policy sets:

A policy set is a named collection of Quality of Service (QoS) policies. You can use either the administrative console or the wsadmin commands to manage system policy sets. Policy sets can be created, deleted, copied, imported or exported.

A policy set can be shared by multiple resources, such as applications, services, inbound or outbound service endpoints, and operations. Default policy sets are installed using profile augmentation. A policy set can also be imported. A policy set does not have its own bindings. You must attach a policy set to a resource, and then assign a binding to the attachment.

Note: When attempting to connect to a Web service from a thin client, verify that the resources that you are specifying are valid before running the updatePolicySetAttachment command. No configuration changes are made if the requested resource does not match a resource in the attachment file for the application.

A client application can dynamically select a policy suite (reference by name from an application-level policy suites list). Options shown in the administrative console list are based on the type of template that is selected to create the policy set. For example, the SecureConversation policy type is made up of policies for both WSSecurity and WSAddressing.

There are two types of policy sets:

- Application policy sets
- System/trust policy sets

WebSphere Application Server provides predefined system policy sets. For example, WebSphere Application Server provides the following system policy sets by default for the security trust service:

- TrustServiceSecurityDefault

This trust policy set specifies the asymmetric algorithm as well as the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA. Message confidentiality is provided by encrypting the body and signature using RSA. This policy set follows the WS-Security specifications for the issue and renew trust operation requests.

- TrustServiceSymmetricDefault

  This policy set specifies the symmetric algorithm as well as the derived keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using HMAC-SHA1. Message confidentiality is provided by encrypting the body and signature using AES. This policy set follows the WS-Security and Secure Conversation specifications for validate and cancel trust operation requests.

- SystemWSSecurityDefault

  This policy set specifies the asymmetric algorithm and both the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA encryption. Message confidentiality is provided by encrypting the body and signature using RSA encryption

You cannot edit default system policy sets. However, you can create your own custom system policy set, which can be edited later. Copy or export a default or existing custom system policy set to create the new custom policy set. System policy sets can also be imported from a predefined location, or from the default repository. Add one or more policies to each policy set. For example, add any of the following existing policies:

- HTTP transport
- WS-Addressing
- WS-Security
- SSL transport
- JMS Transport

The HTTP transport policy can be used for HTTPS, basic authorization, compression, and binary encoding transport methods.

Web Services Trust standard:

Web Services Trust (WS-Trust) is a proposed Organization for the Advancement of Structured Information Standards (OASIS) standard that enables security token interoperability by defining a request/response protocol. This protocol allows SOAP actors, such as a Web services client, to request of some trusted authority that a particular security token be exchanged for another. The trust service, which is provided with WebSphere Application Service, uses the secure messaging mechanisms of WS-Trust to define additional extensions for the issuance, exchange, and validation of security tokens.

WS-Trust defines a request and response protocol for security token exchange. A client sends a RequestSecurityToken (RST) to a security token service. The request includes the security token that the client is asking to be exchanged. The security token service responds back with a RequestSecurityTokenResponse (RSTR) that contains the new token.

In addition to the token exchange, the WS-Trust request/response protocol is general enough to support token *issuance*, where the client presents a claim to the trust service for the service to authorize through the issuance of a corresponding security token. Token *validation* is where the client presents a token to the trust service and asks that its validity be determined.

Also, WS-Trust enables the issuance and dissemination of credentials within different trust domains. To secure a communication between two parties, the two parties must exchange security credentials (either directly or indirectly). Each party must first determine if they can trust the asserted credentials of the other party.

The OASIS WS-Trust specification defines extensions to Web Services Security (WS-Security) for issuing and exchanging security tokens and for providing ways to establish and access the presence of trust relationships. Using these extensions, applications can engage in secure communication, and these extensions are designed to work with the general Web Services framework. The general Web Services framework includes the WSDL service descriptions, UDDI businessServices and bindingTemplates, and SOAP messages.

The WebSphere Application Server support of WS-Trust focuses on establishing a security context token for Web Services Secure Conversation (WS-SecureConversation). The WS-Trust support focuses on the four actions for the security context token: issue, renew, validate, and cancel. Also supported for WS-Trust Version 1.3 are collection requests for the same actions: issue, renew, validate and cancel. The major component for WS-Trust that WebSphere Application Server supports is the security token service, which is referred to as the trust service.

Support for submission draft and approved levels of the WS-Trust standard

Version 6.1 and later of WebSphere Application Server supports the WS-Trust 2005 Submission Draft specification (Version 1.1). However, WebSphere Application Server does not provide a full security token service that implements all the contents of the WS-Trust draft specification.

Support for the approved version 1.3 specification, which is dated March 2007, is provided for WebSphere Application Server version 7.0. The Security Context Token (SCT) provider supports the OASIS version 1.3 specifications for WS-Trust and WS-SecureConversation. There is a configuration option that allows support for the two different levels of the WS-Trust standard to co-exist on the same server. This provides interoperability between systems and products that support different specification levels. See the topic Configuring the security context token provider for the trust service using the administrative console for details.

A setting is also provided to specifically disable support for the WS-Trust 2005 Submission Draft specification (Version 1.1) for the Security Context Token provider. For more information about this property, refer to the topic Disabling the draft standard level for the Security Context Token.

Processing a trust service request depends on the specifications referenced in the request. Also, the trust service response is determined by the level of the specification used in the request.

For more information about WS-Trust:

- See the IBM DeveloperWorks Web site.
- See the schema for the specification: http://docs.oasis-open.org/ws-sx/ws-trust/200512
- Refer to the `wst` namespace prefix that is used for WS-Trust in the Web Services Trust Language (WS-Trust) specification dated March 2007.

Configuring system policy sets using the administrative console:

By defining a custom policy set or defining assertions about how services are defined, you can configure Web services security. You can use the administrative console to manage custom policy sets.

Before you begin

A policy set specifies a set of common message policy assertions that can be specified within a policy. For example, a policy set can define general security policy assertions that apply to other protocols, such as Web Services Security (WS-Security), SOAP messages, Web Services Secure Conversation (WS-Secure Conversation) and Web Services Trust (WS-Trust).

There are two main types of policy sets; application policy sets and system policy sets. Application policy sets are used for business-related assertions. These assertions are related to the business operations that

are defined in the Web Services Description Language (WSDL) file. System policy sets, on the other hand, are used for non-business-related system messages. These messages are defined in other specifications which apply qualities of service (QoS). Examples of QoS are the request security token (RST) messages that are defined in WS-Trust, the create sequence messages that are defined in WS-Reliable Messaging, and the metadata exchange messages defined by WS-MetadataExchange.

Note: Use system policy sets with the trust service, or Web Services MetadataExhange (WS-MEX). The requestor (client) must utilize Java API for XML-Based Web Services (JAX-WS) only. Requestors which use Java API for XML-based remote procedure calls (JAX-RPC) are incompatible with the policy set QOS.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

About this task

Only custom policy sets can be modified. Default system policy sets are read only and cannot be changed.

1. To define system policy sets, click **Services > Policy sets > System policy sets**.
2. Click one of the following actions to work with the system policy set configurations:

   New To create a system policy set configuration. Enter a unique name for the system policy set configuration in the Name field. For example, you might specify `EcommerceTrustServiceSecurity`.

   Delete To delete an existing configuration. Select the check box next to an existing policy set name, and click **Delete**.

   Copy To copy an existing configuration. Select the check box next to an existing policy set name, and click **Copy**.

   Import
   To import an existing configuration. Select the check box next to an existing policy set name, and click **Import**. See Importing policy sets using the administrative console.

   Export
   To export an existing configuration. Select the check box next to an existing policy set name, and click **Export**. See Exporting policy sets using the administrative console.
3. To edit the settings of an existing policy set configuration, click the link for the existing custom system policy set that you want to change. Use the administrative console to modify existing custom policy sets that have been created.
4. Optional: If creating a policy set, enter a short description for the new policy set. Default policy sets can only be viewed. For a custom policy set, edit the brief description of the policy set in the Description field. This description displays in the list on the System policy sets panel. The description should be meaningful to you and other potential users of this policy set.
5. If creating a new policy set, click **Apply**. The policy set name must be applied before you can add policy types to the new policy set.
6. Optional: If needed, add the policy type information, or change the policy types for an existing system policy set. You can add, delete, enable, or disable policy types for the selected policy set. You can add any valid policy types to the policy set collection. The following are available policy types for system policy sets:
   - HTTP transport - for HTTP transport policies
   - SSL transport - for HTTPS transport policies
   - WS-Addressing - for endpoint addressing policies
   - WS-Security - for secure SOAP messages policies
7. Click **OK** and then click **Save** to save the information directly to the master configuration.

Results

You have provided the basic information to create a system policy set. You can also create a new or update an existing system policy set for the WebSphere Application Server trust service, or Web Services MetadataExhange (WS-MEX), using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

After creating a system policy set and adding the policy types, attach the system policy set to a trust service operation for an endpoint, or attach it to one of the trust service default operations.

Defining a new system policy set using the administrative console:

Use policy sets, or assertions, to define system service operations, for your Web services security configuration. Whenever you create a new policy set, you must add policy types to the policy set. You can add HTTP Transport, WS-Addressing, WS-Security, and SSL Transport policy types to the system policy set collection.

Before you begin

A policy set specifies a set of common message policy assertions that can be specified within a policy. For example, a policy set can define general security policy assertions that apply to other protocols such as Web Services Security (WS-Security), SOAP messages, Web Services Trust (WS-Trust), and Web Services Secure Conversation (WS-SecureConversation).

Note: Use system policy sets with the trust service only. The requestor (client) must utilize Java API for XML-Based Web Services (JAX-WS) only. Requestors which use Java API for XML-based remote procedure calls (JAX-RPC) are incompatible with the policy set QOS.

About this task

Use the system policy sets to configure access to the WebSphere Application Server trust service. You can create and define a custom system policy set.

1. Using the administrative console, click **Services > Policy sets > System policy sets** .
2. To create a system policy set and add a policy type, click **New**.
3. Enter a name for the policy set in the **Name** field. The name must be unique for the new system policy set. For example: EcommerceTrustServiceSecurity
4. Enter a brief description of the policy set in the **Description** field. This description displays in the System Policy Sets collection. The description should be descriptive enough for you and other potential users to identify the policy set.
5. Click **Apply** to apply the name and description information.
6. Click **Add** to add a trust policy by selecting one from the policies listed. The following policies are available to use for system policy sets:
   - HTTP transport - for HTTP transport policies
   - SSL transport - for HTTPS transport policies
   - WS-Addressing - for endpoint addressing policies
   - WS-Security - for secure SOAP messages policies
7. Click **Save** to save directly to the master configuration.

Results

You have provided the basic information to create or modify a policy set. You can also create a new or update an existing policy set for the WebSphere Application Server trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

After creating or modifying a system policy set and adding the policy types, attach the policy set to an endpoint operation or attach it to one of the trust service default operations.

System policy set collection:

Use this panel to create and manage policy sets. A policy set is a named collection of policies. System policy sets, or assertions about how services are defined, are used to configure access to the trust service.

There are two main types of policy sets; application policy sets and system policy sets. Application policy sets are used for business-related assertions. These assertions are related to the business operations that are defined in the Web Services Description Language (WSDL) file. System policy sets, on the other hand, are used for non-business-related system messages. These messages are defined in other specifications which apply qualities of service (QoS). Examples of QoS are the request security token (RST) messages that are defined in WS-Trust, the create sequence messages that are defined in WS-Reliable Messaging, and the metadata exchange messages defined by WS-MetadataExchange.

To view this administrative console page, click **Services > Policy sets > System policy sets**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Select:

Provides a check box next to the name of an existing system policy set that you want to select for further actions.

To manage existing system policy sets, select the check box for a system policy set and then select one of the following actions:

Actions	Description
Delete	Removes one or more selected system policy sets.
Copy	Opens a new panel where you can create a copy of the selected existing policy set. Provide a unique name and, optionally, a description for the copied policy set. You must also specify whether to transfer the attachment and binding from the original version to the copy. You can select only one policy set to be copied at one time.
Import	Imports a policy set. This is a menu item with the option of importing a policy set from a default repository or a selected location. You can select and import the default policy sets from the default repository. The default repository for the import function in the administrative console is the directory which contains the default policy sets. The administrative console also displays the default policy sets in a list which includes descriptions, to allow you to select the desired policy set that you want to import.

Actions	Description
Export	Opens a new panel where you can export the selected policy set. You can select only one policy set to be exported at one time.

New:

Specifies to create and define a custom system policy set.

Name:

Provides a list of available system policy sets.

This column displays a list of default and custom system policy set names. WebSphere Application Server provides several default system policy sets:

- **TrustServiceSecurityDefault** is a default trust policy set. This trust policy set specifies the asymmetric algorithm as well as the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA. Message confidentiality is provided by encrypting the body and signature using RSA. This policy set follows the WS-Security specifications for the issue and renew trust operation requests.
- **TrustServiceSymmetricDefault** is a default trust policy set. This trust policy set specifies the symmetric algorithm as well as the derived key algorithms to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using HMAC-SHA1. Message confidentiality is provided by encrypting the body and signature using AES. This policy set follows the WS-Security and WS-SecureConversation specifications for the validate and cancel trust operation requests.
- **SystemWSSecurityDefault** is a default system policy set that specifies the asymmetric algorithm and both the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA encryption. Message confidentiality is provided by encrypting the body and signature using RSA encryption.

All custom system policy sets (for example, EcommerceTrustServiceSecurity) are also displayed in the list. Click the system policy set name to view additional details about the selected policy set.

Data type:	String
Defaults:	TrustServiceSecurityDefault, TrustServiceSymmetricDefault or SystemWSSecurityDefault

Editable:

Provides information as to whether the system policy set can be edited.

This column shows whether the policy set is a user-defined, custom policy set that can be edited or whether the policy set is a default policy set that is not editable. Values displayed in this field are: `Editable` or `Not editable`. You can change the properties for a default, not editable policy set by copying it, and then modifying the properties of the copy. For more information on modifying a default policy set, see the topic Copy of default policy set and bindings settings.

Note: Even though a policy set is identified as not editable, it is deletable. For example, you cannot edit information for the default system policy set, but you can delete the policy set.

Data type:	String
Default:	Not editable

Description:

Provides brief descriptions of the system policy sets that currently exist.

This column provides a brief description of the policy sets that are available. You cannot edit information for the default system policy sets. For custom policy sets that you create, you can create the description when you create the policy set. Or, you can edit any custom policy set and modify the description on the details panel at any time. The description field is optional.

Related reference

Copy of default policy set and bindings settings
Use this page to copy a policy set that you select from a list of available policy sets.

System policy set settings:

Use this panel to create a new system policy set or to edit information about an existing custom system policy set. System policy sets, or assertions about how services are defined, are used to configure access to the trust service.

To view this administrative console page, complete one of the following procedures:

- **Services > Policy sets > System policy sets > *policyset_name***
- **Services > Policy sets > System policy sets > New**

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Note: You can edit the fields on this page only if the policy set is a custom trust policy set. You cannot edit default trust policy sets.

Name:

Specifies the name of the trust policy set.

This field displays the name of the existing custom policy set that you selected. If a new policy set is being created, this field is blank. Enter a policy set name.

Data type: String

Description:

Specifies a brief description of the new or existing custom policy set.

This field provides a brief description for the existing policy set that is displayed in the Name field. If a new policy set is being created, this field is blank. Enter a brief policy set description to help distinguish it from other policy sets. You cannot change the descriptions of the default policy sets.

Data type: String

Policies:

Specifies a collection of trust-related policies.

The Policies section displays a list of pre-configured system-level trust policies. If the system policy set is a default trust policy set, policies cannot be added, deleted, enabled, or disabled. If the system policy set

is an existing custom trust policy set, you can change the policies. If you are creating a new custom trust policy set, you can add new policies. You can also delete, enable, or disable any policies that are added.

Select:

Specifies that you want to select an existing policy for further actions.

Click **Add** to display a list of valid policies that you can add to the named trust policy set.

To manage existing system policies, select the check box for a policy and select one of the following actions:

Actions	Description
Delete	Removes one or more policies from the named custom policy set.
Enable	Specifies that the policy is enabled for the policy set and displays `Enabled` in the State column.
Disable	Specifies that the policy is disabled for the policy set. The policy remains in the list but displays `Disabled` in the State column.

Policy:

Specifies the name of the policy.

This column displays one or more of these pre-configured trust policies:

- **HTTP transport** – for HTTP transport policies
- **SSL transport** – for HTTPS transport policies
- **WS-Addressing** – for endpoint addressing policies
- **WS-Security** – for secure SOAP messages policies

State:

Specifies an enabled or disabled state for each policy.

This column displays whether the policy is enabled or disabled for each of the policies that are listed in the Policy column. For example, to change the state of the policy from enabled to disabled, select the check box for the policy, and click **Disable**.

Description:

Displays a brief description of the policy.

You can view these descriptions only.

Configuring attachments for the trust service using the administrative console:

You can attach the trust service operations for a service endpoint to a system policy set and binding. Each new endpoint that is specified initially has the following four operations: issue, renew, cancel, and validate. By default, all endpoints inherit the policy set and binding that are attached to the respective trust service operation under Trust Service Defaults. However, you can explicitly attach a different policy set.

Before you begin

First you must define your policy sets and bindings. *Policies* describe the protection or quality of service that is provided (such as message security, transport and so forth). *Bindings* specify some details about how to implement the policy, such as: the path for the keystore file, the class name of the token generator, or the JAAS configuration name.

Note: Use system policy sets with the trust service only. The requestor (client) must utilize Java API for XML-Based Web Services (JAX-WS) only. Requestors which use Java API for XML-based remote procedure calls (JAX-RPC) are incompatible with the policy set QOS.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

About this task

You can attach the trust service operations for a new endpoint to an existing policy set and binding. For each new service endpoint that is specified, four trust service operations (cancel, renew, validate and issue) change from having inherited attachments to being explicitly attached. The four operations are attached to the respective policy set and binding as specified in Trust Service Defaults. Then you can change the attachment to the desired existing policy set and binding.

An endpoint policy set consists of two sections: a bootstrap section and an application section. The system policy set attached to the Issue and renew trust service operations for a specific endpoint must correspond to the bootstrap section of the policy set for that endpoint. The system policy set attached to the Cancel and Validate trust service operations for a specific endpoint must correspond to the application section of the policy set for that endpoint.

This task describes how to manage trust service operations for service endpoint URLs that you want to attach to a system policy set and binding. To complete the configuration of the WebSphere Application Server trust service, you must also complete the following task:

- Create or manage targets. You can create explicit assignments for new service endpoints (targets) or manage endpoints that have a security token explicitly assigned or that inherit the Trust Service Default token.

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding. To specify your global security (cell) default bindings, use the administrative console and click **Services > Policy sets > Default policy set bindings**. For environments with multiple security domains, you can optionally choose the general provider and general client bindings that you want to use as the default bindings for a domain. For more information about default bindings see the topic Setting default policy set bindings.

1. To manage system policy set attachments for trust service operations, click **Services > Trust service > Trust service attachments**. The list displays all endpoints that have at least one operation with a policy set attached as well as Trust Service Defaults. The list also displays the system policy set and the binding for each operation.
2. Select one or more of the following actions to configure the trust service attachments:

   New Attachment

   Opens a new panel where you can specify the service endpoint URL. For each new service

endpoint that is specified, four trust service operations (cancel, renew, validate and issue) change from having inherited attachments to being explicitly attached. The four operations are attached to the respective policy set and binding as specified in Trust Service Defaults. These initial attachments can be changed.

Attach

Displays a list of existing system policy sets, including the default trust-related system policy sets, to which each of the four trust service operations for a service endpoint can be attached. First, select the operation (for example, Cancel token) and then click **Attach** to display the list of available system policy sets. Select a default or custom system policy set to attach. When you change the policy set attachment, the binding automatically changes to **Default**. Select the operation and click **Assign Binding** to change the binding.

The pre-configured system policy sets that you can select include:

- **TrustServiceSecurityDefault**

  This trust policy set specifies the asymmetric algorithm as well as the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA. Message confidentiality is provided by encrypting the body and signature using RSA. This policy set follows the WS-Security specification for the issue and renew trust operation requests.

- **TrustServiceSymmetricDefault**

  This trust policy set specifies the symmetric algorithm as well as the derived key algorithms to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using HMAC-SHA1. Message confidentiality is provided by encrypting the body and signature using AES. This policy set follows the WS-Security and WS-SecureConversation specifications for the validate and cancel trust operation requests.

- **SystemWSSecurityDefault**

  This system policy set specifies the asymmetric algorithm and both the public and private keys to provide message security. Message integrity is provided by digitally signing the body, time stamp, and WS-Addressing headers using RSA encryption. Message confidentiality is provided by encrypting the body and signature using RSA encryption.

Inherit Operation Defaults

Sets the operation to inherit the respective trust service default trust service policy set attachment and binding. If you select the attachments to modify and then click **Inherit Operation Defaults**, the explicit attachment for both the policy set and the binding is removed. Thereafter, the operation inherits any change to the default trust service policy set and binding.

Assign Binding

Changes the existing binding. You can create and assign a new binding, assign the Default binding, or assign an existing trust service specific binding to each of the selected trust service attachments.

Update Runtime

Updates the trust service runtime with any configuration changes that are made to the trust service attachments, token providers, and targets.

3. Optional: Modify the custom policy set by clicking the name of a custom policy set from the list. Edit the settings for custom policy sets, as needed. Default trust service policy set information can only be viewed.

   You cannot edit the default policy sets: TrustServiceSecurityDefault and TrustServiceSymmetricDefault, or SystemWSSecurityDefault. TrustServiceSecurityDefault is the default for the issue and renew operations. TrustServiceSymmetricDefault is the default for the cancel and validate operations.

At least one trust service operation for the endpoint service URL must be explicitly attached for the endpoint service URL to be displayed. If an operation is explicitly attached, the system policy set name appears. If no policy set is explicitly attached, the respective default trust service policy set appears, followed by the text (inherited).

4. Optional: Modify the trust service specific binding by clicking the name of a binding from the list, as needed. Edit the settings for the trust service specific binding, as needed. Any modifications to a trust service binding affect all trust service attachments that reference the binding.

   If the resource has a policy set directly attached, either the bindings name appears or Default appears.
5. Save your changes before applying the changes to the trust service runtime configuration.
6. Click **Update Runtime** to update the trust service runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window appears depends on whether you select the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
7. Optional: Confirm or cancel if the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

You have provided the basic information to create or update a trust service attachment. You have configured trust service operation attachments to system policy sets and bindings.

What to do next

You can also create a new attachment for the WebSphere Application Server trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

Creating a service endpoint attachment using the administrative console:

You can attach the trust service operations for a new service endpoint URL to system policy sets and bindings. The operations for each new endpoint are attached to the Trust Service Default policy sets and bindings. Each new endpoint initially has the following four operations: issue, renew, cancel, and validate.

Before you begin

First you must define your policy sets and their bindings. *Policy sets* describe the protection or quality of service that is provided (such as message security, transport and so forth). *Bindings* specify some details about how to implement the policy set, such as: the path for the keystore file, the class name of the token generator, or the JAAS configuration name.

Note: Only use system policy sets with the trust service. The requestor (client) must utilize only Java API for XML-Based Web Services (JAX-WS). Requestors that use Java API for XML-based remote procedure calls (JAX-RPC) are incompatible with the policy set QOS.

About this task

Attaching the trust service operations for a new endpoint to existing policy sets and bindings requires two steps. After initially attaching the endpoint, the following four operations are configured: issue, renew, cancel, and validate. These four operations explicitly attach to Trust Service Defaults. You can then modify these attachments to existing policy sets and bindings.

This task describes how to create or manage service endpoint URLs that you want to attach to the policy set and binding. To complete the configuration for the WebSphere Application Server trust service, you must also create or manage targets.

If no explicit bindings are attached, WebSphere Application Server uses the cell-level default binding, referred to as Default.

1. To view existing trust service attachments, click **Services > Trust service > Trust service attachments**. Until you create the first attachment, only the default attachments for each operation are displayed.
2. To create an attachment, click **New Attachment**.
3. Enter the service endpoint URL in a valid format. Note that when the URL in the trust service attachment does not match the URL, including matching the case, to which the trust service request is sent, the policy set that is defined in the attachment is not applied. Instead, IBM WebSphere Application Server uses the policy set that is attached to the default for the trust operation.

   For example, where `demo` is the endpoint, you might enter: http://localhost:9080/wssamplebeta/demo
4. Click **Attach** to attach the URL and to return to the Trust service attachments panel. After you click **Attach**, the Trust service attachments panel displays the new service endpoint URL and the initial four operations. The service endpoint URL that you specified is listed in the Trust service attachments collection. These four token operations (cancel, renew, validate and issue) for the specified endpoint are initially attached to Trust Service Defaults.
5. On the Trust service attachments panel, change the policy set or binding attachment, as needed. You can return any operation to its initial state by inheriting Trust Service Defaults.

   Note: Changing the policy set forces the binding to change to Default.
6. Save your changes before applying the changes to the Web services security runtime configuration.
7. Click **Update Runtime** to update the Web services security runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window appears depends on whether you selected the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
8. Optional: Confirm or cancel if the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

You have provided the basic information to create a trust service attachment and to configure a policy set, a binding, and the operation information.

What to do next

You can also create a new attachment for the trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

Next, configure the security context token provider or configure targets to complete the trust service configuration.

Trust service attachments collection:

Use this page to view information about or manage system policy set attachments and bindings. Endpoints with at least one operation directly attached to a policy set are displayed.

This page displays each endpoint that has at least one operation that is directly attached to a system policy set. The operations for other endpoints inherit the trust service default policy set and binding data. You can click **New Attachment** to create explicit attachments for endpoints not displayed, or click **Attach** to change the policy set for an operation. Changing the system policy set for an operation removes the binding data for that operation, and resets that data to the system default binding settings. You can also click **Assign Binding** to create a new binding configuration or change the existing binding configuration for the selected operation.

To view this administrative console page, click **Services > Trust service > Trust service attachments**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Show confirmation for update runtime command:

Specifies whether to enable or disable the display of the confirmation window before the Web services security runtime configuration is updated for supported tokens, targets, and trust service attachments.

Click **Preferences** to expand the information. You can select or clear the **Show confirmation for update runtime command** check box. If you do not select this check box, updates to the security runtime configuration are made without first displaying a confirmation window. If you select the check box, the confirmation window is displayed before updates to the security runtime configuration are made.

Data type: Check box
Default: Enabled (check box is selected)

Retain filter criteria:

Specifies whether to retain the filter criteria.

Click **Preferences** to expand the information. You can select or deselect the **Retain filter criteria** check box. This check box determines whether **Endpoint URL** is used as the filter criteria to reduce the displayed list of endpoints.

Data type: String
Default: All (check box is not selected)

Search terms:

Specifies the search criteria to use to reduce the displayed list of endpoints.

Click **Preferences** to expand the information. Type the search term you want to use in the **Search terms** field. Use the asterisk (*) as a wildcard character for all terms. You can also search for multiple unknown or partial characters within the term. For example, typing the search term `par*` returns `partly`, `participate`, `partial`, and all other terms beginning with the letters `par`.

Data type: String
Default: * (search for all)

Select:

Specifies that you want to select an existing resource, such as an endpoint or an operation, for further actions.

For existing endpoints, select the check box next to an operation, and then select one of the following actions:

Actions	Description
Attach	Displays a list of policy sets that are available to be attached to an endpoint operation (cancel, reset, validate, or issue) or to one of the trust service default operations. Highlight and click the policy set to attach the policy set to the selected operation. You cannot attach a policy set to an endpoint.
Inherit operation defaults	Detaches the currently attached policy set and binding for each selected operation and sets the operation to inherit the trust service default policy set and binding for each operation.
Assign binding	Lists the bindings that are available to select for the policy set to which you want to attach the binding. You can also create a new binding. • Select **Default** to create and assign the system default binding to the selected policy set attachment. When you select this binding the runtime uses the default binding for the server, cell or in the multiple security domain environment to which the service resource is deployed. • Select **New Trust Service Specific Binding** to create a binding that is specific to the policy set and shares the characteristics of the policy set. This type of binding is reusable only for trust service attachments. • Select an existing general binding to assign the binding to the selected policy set attachment. Multiple selection is valid only when all the resources have the same policy set attached.

New attachment:

Specifies that you want to create an explicit policy set attachment.

Click **New Attachment** to access a new panel where you can enter an endpoint URL to create attachments for each of the four endpoint operations of the provided URL. Initially, the attachment consists of the policy set and binding that are listed as the Trust Service Default for that operation.

Data type: Button

Update runtime:

Updates the trust service configuration for any changed attachments, targets, and token information.

If the **Show confirmation for update runtime command** preference is enabled, then a panel is displayed where you can confirm that you want to update the trust service configuration. If the preference is disabled, the trust service configuration is updated immediately without any confirmation.

Data type: Button

Service endpoint URL / Operation:

Displays a list of the trust service default operation attachments and every service endpoint URL that has at least one operation with a policy set attached.

Each endpoint has four operations: issue, cancel, renew, and validate. Each of the operations for all other endpoints inherits the trust service default policy set and binding.

When the URL in the trust service attachment does not match the URL to which the trust service request is sent, the policy set that is defined in the attachment is not applied. Instead, IBM WebSphere Application Server uses the policy set that is attached to the default for the trust operation.

Data type: String
Default: Trust Service Default

Policy set:

Displays the attached or inherited policy set for each operation of all endpoint URLs. Any endpoint URL that is not displayed inherits the trust service default policy set for each operation. Provides a list of default and custom system policy sets that are attached to the service endpoint URL.

The policy set names are displayed in this column for each operation. If the policy set is inherited from the trust service default, rather than being explicitly attached, **inherited** is displayed in parentheses following the policy set name. Because only operations can have a policy set attachment, the Policy Set column for each endpoint URL row displays **Not applicable**.

Click the system policy set name to view or edit the policy set details information. Note that you can view, but not edit, the default policy sets. Default policy sets cannot be changed.

Data type: String
Defaults: TrustServiceSecurityDefault, TrustServiceSymmetricDefault or SystemWSSecurityDefault

Binding:

Displays the binding that is assigned to each policy set attachment for each operation of the listed endpoint URLs. Any endpoint URL that is not displayed inherits the trust service default binding for each of the four operations.

The name of the assigned binding for each policy set attachment is displayed in this column for each operation. If the attachment is inherited from the trust service default, **inherited** is displayed in parentheses following the binding name. If you select **Assign Binding > Default**, the system default binding is applied to the policy set attachment, and the word **Default** is displayed in this column. If the system default binding is inherited, then **inherited** is displayed in parentheses following **Default**.

The system default binding is also assigned when you attach a new policy set to an operation. Because only operations can have policy set attachments, the binding column for each endpoint URL row displays **Not applicable**. Rows that are not directly related to a token and display the trust service default, display the text, **Not applicable**, for the binding. Additionally, rows that are not directly related to a token and display only the service endpoint URL display the text, **Not applicable**, for the binding.

Click the trust service specific binding name to view or edit the binding information. You can view, but not edit, the TrustServiceSecurityDefault, TrustServiceSymmetricDefault or SystemWSSecurityDefault bindings.

Data type: String
Default: TrustServiceSecurityDefault, TrustServiceSymmetricDefault or SystemWSSecurityDefault

Trust service attachments settings:

Use this page to create a new attachment to the current Trust Service Defaults policy set and binding for the four token operations: cancel, issue, renew, and validate.

To view this administrative console page, complete the following procedure:

- Click **Services > Trust service > Trust service attachments > New Attachment** .

Service endpoint URL:

Specifies the service endpoint URL that you want to attach to the policy set and binding for the trust service default operations.

Use this field to specify a service endpoint URL. The URL must be specified in a valid format, such as `http://www.mybusiness.com`.

Note that when the URL in the trust service attachment does not match the URL to which the trust service request is sent, the policy set that is defined in the attachment is not applied. Instead, IBM WebSphere Application Server uses the policy set that is attached to the default for the trust operation.

After you enter the URL and click **Attach**, the custom service endpoint URL is displayed in a list of explicitly attached service endpoint URLs on the Trust service attachments panel. In addition to the new service endpoint URL, the Trust service attachments panel displays a list of the corresponding four operations (cancel, issue, renew and validate).

On the Trust service attachments panel, you can change the Trust Service Default policy set and binding attachments for any of the four operations. These policy sets apply to any URL not displayed, and therefore not explicitly attached to a policy set and binding. Changing the policy set for a URL operation resets a custom binding setting to the default value.

On the Trust service attachments panel, if you want to remove the explicit policy set attachments and binding assignments, select each of the URL operations, and click **Inherit Operation Defaults**. If all four operations are changed to inherit the Trust Service Default policy set and binding, then the URL no longer displays on this panel.

Data type: String (URL format)

New general binding settings:

Use this page to create a provider binding which is reusable across policy sets and applications. Use the **Add** button to select policy bindings and then provide configuration settings. Empty bindings are deleted.

To view this administrative console page, click **Services > Trust service > Trust service attachments > Assign Bindings > New General Binding**.

Bindings configuration name:

Enter a name for the new binding.

Description:

Enter a description for the new binding.

Select policy bindings:

Click **Add**, then select an existing policy set binding. To delete a binding, click the checkbox next to the binding name, then click **Delete**.

Actions	Description
Add	Select a policy set binding. Configure binding settings using the panel that appears after you add the binding.
Delete	Click the checkbox next to the binding name, then click **Delete**.

Configuring the security context token provider for the trust service using the administrative console:

Configure the WebSphere Application Server trust service to issue a specific security token to the requestor for communication with an endpoint. Use the administrative console to configure the security context token provider that the trust service provides.

Before you begin

WebSphere Application Server provides a trust service. The trust service provides both a security token service and additional WebSphere Application Server trust-related functionality. To configure the trust service, in addition to managing the security context token provider, you must first complete the following tasks:

- Create or manage supported targets. You can create explicit assignments for new service endpoints (targets) or manage endpoints that have the security context token provider explicitly assigned or that inherit the token provider designated as the Trust Service default.
- Create or manage the attachment of token operations for service endpoints to policy sets and bindings.

The order in which you complete these tasks is not important.

About this task

This task describes how to manage the security context token provider and how to define or modify the properties of the security context token provider.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. To manage the security context token provider, click **Services > Trust service > Token providers**.
2. To edit the settings of the security context token provider configuration, click the link for the token provider name. You cannot edit the name, class name, or token type schema URI when modifying the token provider information.
   a. The format of the token type schema Uniform Resource Identifier (URI) is in the standard URI format. For example, for a version 1.3 security context token, the URI is: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
   b. Change the amount of time, in minutes, in the **Time in cache after timeout** field that the expired token is kept in cache and where the token can still be renewed. The default value is 120 minutes. This value cannot be less than 10 minutes.
   c. Change the amount of time, in minutes, in the **Token timeout** field that the issued token is valid. The default value is 10 minutes. This value cannot be less than 10 minutes.
   d. Select the **Allow renewal after timeout** check box to enable the renewal of a token after the token has expired. If selected, the amount of time, within which an expired token can still be renewed, is specified in minutes in the **Time in cache after expiration** field.

e. Select the **Allow postdated tokens** check box to enable postdated tokens. Use postdated tokens to specify whether a client can request a token to become valid at a later time.

f. Select the **Support Secure Conversation Token v200502** to enable use of the older draft submission specification level of the security context token. The correct URI for this level of the token type schema appears in the field under the check box: http://schemas.xmlsoap.org/ws/2005/02/sc/sct.

g. Click **New** to define a new custom property or click **Edit** to modify the custom property. Specify these settings using the Custom Properties setting. Custom properties are used to set internal system configuration properties. Custom properties are arbitrary name-value pairs of data, where the name might be a property key or a class implementation, and where the value might be a string or the value might be a true or false value.

h. If you define a custom property, type a name. Refer to the documentation for the token provider for valid custom property names.

i. If you define a custom property, type a value. Refer to the documentation for the token provider for the values for a property name.

j. Repeat defining the name and the value for each custom property that you add.

k. Click **OK**. You are returned to the Token providers panel.

3. Save your changes before applying the changes to the Web services security runtime configuration.
4. Click **Update Runtime** to update the Web services security runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window is displayed depends on whether you select the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
5. Optional: Confirm or click **Cancel** when the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

You have completed the required steps to modify the security context token provider configuration and to update the Web services security runtime configuration. You can also update the security context token provider configuration for the trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

Next, if you have not done so already, you must also configure targets or configure attachments to complete the trust service configuration.

Modifying the security context token provider configuration for the trust service using the administrative console:

WebSphere Application Server provides a pre-configured token, the Security Context Token (SCT). Use the administrative console to modify the configuration of the security context token provider.

Before you begin

WebSphere Application Server provides a trust service. The trust service provides both a security token service and additional WebSphere Application Server trust-related functionality. To configure the trust service, in addition to managing the security context token provider, you must first complete the following tasks:

- Create or manage supported targets. You can create explicit assignments for new service endpoints (targets) or manage endpoints that have a security token provider explicitly assigned or that inherit the token provider designated as the Trust Service default.

- Create or manage the attachment of token operations for service endpoints to policy sets and bindings.

The order in which you complete these tasks is not important.

About this task

This task describes how to configure the security context token provider and how to define the token provider properties.

1. To configure the security context token provider, click **Services > Trust services > Token providers**.
2. To change the configuration of the security context token provider, click the link for the token provider name (Security Context Token). For an existing token, the token name, class name and URI are displayed, but are not editable.
3. Optional: Change the amount of time, in minutes, in the **Time in cache after expiration** field that the expired token is kept in cache and where the token can still be renewed. The default value is 120 minutes, and you cannot type a value that is less than 10 minutes.
4. Optional: Change the amount of time, in minutes, in the **Token timeout** field that the issued token is valid. The default value is 120 minutes, and you cannot type a value that is less than 10 minutes.
5. Optional: Select the **Allow renewal after timeout** check box to enable the renewal of a token, after the timeout time has expired. If selected, the amount of time, within which an expired token can still be renewed, is specified in the **Time in cache after expiration** field.
6. Optional: Select the **Allow postdated tokens** check box to enable postdated tokens. Use postdated tokens to specify whether a client can request a token to become valid at a later time.
7. Optional: Select the **Support Secure Conversation Token v200502** check box to enable use of the older draft submission specification level of the security context token. The correct URI for this level of the token type schema appears in the field under the check box: http://schemas.xmlsoap.org/ws/2005/02/sc/sct.
8. Click **New** if you want to define a new custom property. Specify additional configuration using the **Custom Properties** setting. Custom properties are used to set internal system configuration properties. Custom properties are arbitrary name-value pairs of data, where the name might be a property key or a class implementation, and where the value might be a string or Boolean value.
   a. If defining a new custom property, type a name. For example, for a custom property, type: com.ibm.wsspi.wssecurity.trust.keySize
   b. If defining a new custom property, type a value. For example, the following value: 128
   c. Repeat the name and value steps for each new custom property.
9. Click **OK**. You are returned to the Token provider panel.
10. Save your changes before applying the changes to the Web services security runtime configuration.
11. On the Token provider panel, click **Update Runtime** to update the Web services security runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window is displayed depends on whether you select the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
12. Optional: Confirm or click **Cancel** when the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

You have completed the required steps to modify the configuration of the security context token provider and to update the Web services security runtime configuration. You can also modify the configuration of the security context token provider for the trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

If you have not done so already, you must also configure targets or configure attachments to complete the trust service configuration.

Trust service token custom properties:

WebSphere Application Server trust service provides several custom properties by default to define the default security context token (SCT).

Custom properties are name-value pairs of data that are passed to the token provider during configuration.

The Property name column displays the name of the custom property. The name must match the name of a configuration property or setting that the provider understands and expects. The Property value column displays the configuration setting that is passed to the provider during configuration.

These custom properties are provided by default by WebSphere Application Server, for you to configure when using the **Services > Trust service > Token providers > Security Context Token** page.

algorithm:

The value is AES.

keySize:

The value is 128.

Provider:

The value is IBMJCE.

Disabling the submission draft level for the security context token provider:

Use the administrative console to configure the security context token provider that the trust service provides. Two levels of the token are supported on WebSphere Application Server: the token defined by the WS-Trust February 2005 Submission Draft specification, and the token defined by the OASIS WS-Trust Standard version 1.3. You can disable a setting so that the server will not accept a trust request that specifies the submission draft level of the token.

About this task

Disable the Security Context Token provider support for the submission draft specification using the administrative console.

1. Log on to the administrative console and navigate to the Token providers panel by clicking **Services** → **Trust service** → **Token providers**.
2. Click on **Security Context Token**.
3. Click to clear the **Support Secure Conversation Token v200502** check box.
4. Click **Apply** to save the changed setting.

Results

For more information , see the topic Modifying the security context token provider configuration for the trust service using the administrative console.

Related tasks

"Modifying the security context token provider configuration for the trust service using the administrative console" on page 302

WebSphere Application Server provides a pre-configured token, the Security Context Token (SCT). Use the administrative console to modify the configuration of the security context token provider.

Trust service token provider settings:

Use this page to modify information for an existing token provider.

To view this administrative console page, complete the following actions:

- **Services > Trust service > Token providers > *token_provider_name***

Name:

Specifies the name of the token provider.

This field displays the unique name of the token provider (for example, `Security Context Token`). You cannot change the name for any existing token provider.

Data type: String

Class name:

Specifies the package and class name of the trust service's Security Context Token provider.

This field displays the configuration class name, including the package information (for example, `com.ibm.ws.wssecurity.trust.server.sts.ext.sct.SCTHandlerFactory`).

You cannot change the class name for any existing token provider.

Data type: String

Token type schema URI:

Specifies the Uniform Resource Identifier (URI) for the token type schema.

This field displays the unique token type schema URI. Use a valid URI format, such as: `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct`.

You cannot change the schema URI for any existing token provider.

Data type: String

Time in cache after expiration:

Specifies the number of minutes that a token remains in the token cache after the token expires.

This field displays the time, in minutes, that the expired token is kept cached and can still be renewed.

Data type: Integer
Default: 120
Minimum: 10
Maximum: 2147483647

Token timeout:

Specifies the amount of time, in minutes, that the issued token is valid.

This field displays the maximum timeout, in minutes, for a token to be considered valid.

Data type:	Integer
Default:	120
Minimum:	10
Maximum:	2147483647

Allow renewal after timeout:

Specifies to enable or disable the renewal of a token.

This check box specifies whether to allow a client to renew an expired token. Note the **Time in cache after expiration** field specifies the amount of time within which an expired token can still be renewed.

Data type:	Check box
Default:	Do not allow (unchecked)

Allow postdated tokens:

Specifies to enable or disable the use of postdated tokens.

This check box specifies whether a client can request a token to become valid at some point in the future.

Data type:	Check box
Default:	Do not allow (unchecked)

Support Secure Conversation Token v200502: This check box specifies whether support for the WS-Trust and WS-Secure Conversation Feb 2005 Submission Draft OASIS specification is enabled. The default URI for the token type schema is provided in the non-editable field below the check box.

Data type:	Check box
Default:	Enabled (checked)

Custom Properties:

Specifies additional configuration settings that the token provider might require.

This table lists custom properties. Use custom properties to set internal system configuration properties.

The Secure Context Token default configuration settings are :

Property Name	Property Value
com.ibm.wsspi.wssecurity.trust.algorithm	AES
com.ibm.wsspi.wssecurity.trust.keySize	128
com.ibm.wsspi.wssecurity.trust.provider	IBMJCE

Select:

Specifies custom properties that you can add to, edit, or delete from the token provider.

Click **New** to add and define a new custom property.

For existing custom properties, first select the check box for the name of the custom property, and click one of the following actions:

Actions	Description
Edit	Specifies whether to modify existing custom properties. This action requires one or more custom properties to be selected.
Delete	Removes the selected existing property from the listing in the Name column. This action requires one or more custom properties to be selected.

Name:

Displays the names of the custom properties that have been defined for the token provider.

This column displays the name of the custom property (for example, `com.ibm.wsspi.wssecurity.trust.keySize`). Custom properties are name-value pairs of data that are passed to the token provider during configuration. The name that you specify must match the name of a configuration property or setting that the provider understands and expects.

Data type: String

Value:

Specifies the value for the custom property.

This column displays the value for the custom property (for example, `true`). Custom properties are name-value pairs of data. The value, which is represented as a string, is a configuration setting that is passed to the provider during configuration.

Data type: String or Boolean

Trust service token providers collection:

Use this page to view information about or manage token providers for the trust service.

To view this administrative console page, click **Services > Trust service > Token providers**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Show confirmation for update runtime command:

Specifies to enable or disable the display of the confirmation window before the trust service configuration is updated when you click **Update Runtime**.

Click **Preferences** and then select the **Show confirmation for update runtime command** check box. If you select this check box, the confirmation window is displayed before updates to the security trust service configuration are made. If you do not select this check box, clicking **Update Runtime** updates the security trust service configuration without first displaying a confirmation window.

Data type: Check box
Default: Enabled (checked)

Update Runtime:

Updates the trust service configuration for any changed attachments, targets, and token information.

If the **Show confirmation for update runtime command** preference is enabled, then a panel is displayed where you can confirm that you want to update the trust service configuration. If the preference is disabled, updates to the trust service configuration are applied immediately without any confirmation.

Data type: Button

Token Provider Name:

Lists available token providers.

This column displays the names of the pre-configured token providers. The pre-configured token provider is the Security Context Token (SCT). Click a token provider name link to view additional details.

Data type: String
Default: Security Context Token

Token Type Schema URI:

Provides the Uniform Resource Identifier (URI) for the token type schema.

This column displays the URIs of all pre-configured token providers.

Data type: String

Configuring trust service endpoint targets using the administrative console:

The Trust Service manages tokens on behalf of service endpoints. A token provider is either explicitly or implicitly associated with each service endpoint. A specific token can be explicitly assigned to be issued when access to an endpoint is requested. Otherwise, the Trust Service Default token is issued.

Before you begin

The Web Services Secure Conversation specification defines the protocol for a client to establish a secure session with a target service. The security token service that WebSphere Application Server provides, referred to as the trust service, issues only the Security Context Token (SCT). The security context token is used for Web Services Secure Conversation (WS-SecureConversation).

About this task

This task describes how to create new or manage existing assignments of tokens to be issued for endpoint targets. You can create explicit assignments for new service endpoints (targets) or manage existing token assignments.

To complete the configuration for the trust service, you must have performed the following tasks:

- Manage the security context token provider.
- Create or manage service endpoint URLs that you want to attach to the policy set and binding.

The order in which you complete these tasks is not important.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. To configure new and existing trust service endpoint targets, click **Services > Trust service > Targets**. A list of all service endpoints that have a security token provider explicitly defined is displayed. The token provider assigned to the Trust Service Default by default handles requests to issue tokens to access an endpoint.
2. Click one of the following actions to manage a new or existing endpoint target configuration:

   New Assignment

   Opens a new panel where you can specify a custom service endpoint URL and explicitly assign the token provider, which is specified as the Trust Service Default, to be issued for access to the endpoint.

   Change Token

   Changes an explicitly assigned token to be issued for the service endpoint to the security context token. Select an endpoint and then click **Change Token**. Select the Security Context Token.

   Also, removes the explicit assignment of a token to be issued; therefore, the token that is issued is inherited from the Trust Service Default. Select an endpoint and then click **Change Token**. Click **Inherit Default** to remove a token provider assignment for the selected endpoint and to return the issued token to be the token that is specified as the Trust Service Default. If the token that is issued is inherited, the endpoint is no longer displayed in the list because the token provider is no longer explicitly assigned to the endpoint.
3. Click the token name link for an existing endpoint target to modify the token provider configuration information. You can modify the token type schema URI, or change custom properties.
4. Save your changes before applying the changes to the Web services security runtime configuration.
5. Click **Update Runtime** to update the Web services security runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window is displayed depends on whether you select the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
6. Optional: Confirm or click **Cancel** when the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

When you complete these steps, the service endpoint URL displays in the Targets collection, unless you changed the token to inherit the default value. You can also configure the trust service to issue tokens for individual endpoint targets using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

You have completed the required steps to create or manage existing trust service targets, to assign the security token provider to an endpoint target, and to update the Web services security runtime configuration. Next, if you have not competed these tasks already, configure the security context token provider or configure attachments to the policy set and binding to complete the trust service configuration.

Assigning a new target for the trust service using the administrative console:

You can associate a security token provider with a service endpoint using the administrative console. After entering the service endpoint URL, the token provider configured as the Trust Service Default is explicitly associated with the service endpoint.

Before you begin

The Web Services Secure Conversation specification defines the protocol for a client to establish a secure session with a target service. The security token service that WebSphere Application Server provides, referred to as the trust service, issues the Security Context Token (SCT). The security context token is required for Web Services Secure Conversation (WS-SecureConversation).

About this task

This task describes how to register a service endpoint (target) with the trust service. Registration of an service endpoint with the trust service initially associates the token provider configured as the Trust Service Default with that service endpoint.

To complete the configuration for the trust service, you must have completed the following tasks:

- Manage the Security Context Token.
- Create or manage service endpoint URLs that you want to attach to the policy set and binding.

The order in which you complete these tasks is not important.

1. To configure a custom endpoint target, click **Services > Trust service > Targets > New Assignment**.
2. At the New assignment panel, enter the Universal Resource Locator (URL) for the service endpoint, and click **Assign**. You are returned to the Targets panel where the custom service endpoint URL is displayed in the list. Initially, the token that is explicitly assigned to the custom endpoint is the token that is assigned as the Trust Service Default.
3. At the Targets panel, select the check box for a service endpoint, click **Change Token**, and select one of the following:
   a. Security Context Token (SCT). A security context token is defined by the WS-SecureConversation specification.
   b. **Inherit Default** if you want the token that is issued to be the token assigned as the Trust Service Default. The endpoint is not displayed in the list when the assignment is inherited because the token is no longer explicitly assigned to the endpoint.
4. At the targets panel, click the token name link for an existing endpoint target to modify the token provider configuration information.
5. Save your changes before applying the changes to the Web services security runtime configuration.
6. Click **Update Runtime** to update the Web services security runtime configuration with any data changes for token providers, trust service attachments, and targets. Whether the confirmation window is displayed depends on whether you select the **Show confirmation for update runtime command** check box. Expand **Preferences** to view the check box.
7. Optional: Confirm or click **Cancel** when the confirmation window appears. If you deselected the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.

Results

When you complete these steps, service endpoints explicitly associated with a token provider are displayed in the Targets collection. Service endpoints that have been changed to inherit the token provider configured as the Trust Service Default are not displayed. You can also configure the security token service to issue a specific token for access to a target using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

What to do next

You have completed the required steps to create a service endpoint URL, to assign the token to be issued for access to the target, and to update the Web services security runtime configuration. Next, if you have not completed these tasks already, configure the Security Context Token provider or configure attachments to the policy set and binding to complete the trust service configuration.

Trust service targets collection:

Use this page to view a list of targets, which are application server service endpoints. You can manage tokens by specifying which token is to be issued when access to a specific endpoint is requested.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

To view this administrative console page, click **Services > Trust service > Targets**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Show confirmation for update runtime command:

Specifies to enable or disable the display of the confirmation window before the WebSphere Application Server trust service configuration is updated when you click **Update Runtime**.

Click **Preferences** and then select the **Show confirmation for update runtime command** check box. If you select this check box, the confirmation window is displayed before updates to the trust service configuration are made. If you do not select this check box, clicking Update Runtime updates the trust service configuration without first displaying a confirmation window.

Data type:	Check box
Default:	Enabled (checked)

Select:

Specifies a check box for the service endpoint Universal Resource Locator (URL) that you want to select for further actions.

For existing endpoints, select the checkbox for the service endpoint and select one of the following actions:

Actions	Description
Change Token	Changes the token that is issued when access to an endpoint is requested. Selecting **Inherit Default** in the Change Token menu causes the following actions to occur: • The security token assignment is removed for the endpoint. • The token assigned as the Trust Service Default is issued for access to the endpoint. • The endpoint is no longer displayed in the list of endpoints that have tokens explicitly assigned. Only endpoints that are explicitly assigned a security token are displayed in the list. Endpoints that inherit the default do not display in the list.

New Assignment:

Defines a new service endpoint.

Initially, each endpoint is explicitly assigned the Trust Service Default token. By default, the pre-configured Security Context Token (SCT) is assigned, but that can be changed.

Data type: Button

Update Runtime:

Updates the trust service configuration for any changed attachments, targets, and token information.

If the **Show confirmation for update runtime command** preference is enabled, then a panel is displayed where you can confirm that you want to update the trust service configuration. If the preference is disabled, updates the trust service configuration immediately without any confirmation.

Data type: Button

Service Endpoint URL:

Specifies the Universal Resource Locator (URL) of the service endpoint for the explicitly assigned token.

This column lists the default service endpoint, Trust Service Default, and any custom service endpoints that have a token that is explicitly assigned to the endpoint, such as: `http://localhost:9080/EcommerceSTS`.

Data type: String
Default: Trust Service Default

Token Name:

Displays the name of the token to be issued when access to the endpoint is requested.

To inherit the default token, select the check box for a custom service endpoint URL, click **Change Token > Inherit Default**.

You can change the token type that is explicitly assigned as the Trust Service Default, but the token type cannot be left unassigned. If the token is not explicitly assigned, then the endpoint inherits the token that is assigned as the Trust Service Default token.

Click a token name link to access detailed information about the token. You can modify the token information, except for the token name. It is recommended that you do not modify the class name or the token type schema URI for the default token type, Security Context Token.

Changes to token properties apply to all tokens of this type that are issued for any endpoint.

Data type: String
Default: Security Context Token

Token Type Schema URI:

Specifies the schema Uniform Resource Identifier (URI) for the token type.

This column displays the schema URI for the explicitly assigned token type (for example, Security Context Token) in a valid URI format. The token type schema URI is a property of the token name and describes the version of the specification that is implemented for the security token.

Data type: String
Default value: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct

Trust service targets settings:

Use this page to specify a custom service endpoint Universal Resource Locator (URL) and to assign a custom token type to the endpoint URL.

To view this administrative console page, click **Services > Trust service > Targets > New Assignment**.

Service endpoint URL:

Specifies the URL for the service endpoint.

Use this field to specify a custom service endpoint URL. The URL must be specified in a valid format, such as `http://localhost:9080/EcommerceSTS`. After you enter the URL and click **Assign**, the endpoint URL is explicitly assigned to the security token that is assigned the Trust Service Default.

The service endpoint URL is added to the list that displays on the Targets panel. Only endpoints that are explicitly assigned a security token are displayed in the list. Endpoints that inherit the default do not display in the list.

By default, the Trust Service Default token is the Security Context Token (SCT).

After clicking **Assign** and returning to the Targets panel, if you want to remove the explicit token assignment (and thereby change the token to be issued back to the default value), select the custom endpoint URL, and click **Inherit Default**. Then the following actions occur:

- The security token assignment is removed for the endpoint.
- The token assigned as the Trust Service Default is issued for access to the endpoint.
- The endpoint is no longer displayed in the list of endpoints that have tokens explicitly assigned.

Data type: String (URL format)

Updating the Web services security runtime configuration:

Update Web services security runtime configuration with any data changes that you make and save for token providers, trust service attachments, and targets.

Before you begin

Before you update the Web services security runtime configuration, make your required data changes for token providers, trust service attachments, and targets. Save your changes before applying the changes to the Web services security runtime configuration.

About this task

Whether the confirmation window appears depends on whether the **Show confirmation for update runtime command** check box is selected. Expand **Preferences** from the Token providers panel, the Trust service attachments panel, or the Targets panel to see the **Show confirmation for update runtime command** check box. Preferences are collapsed by default.

- If you select the check box, then the confirmation window is displayed before updates to the security runtime configuration are made. The check box is selected by default.
- If you do not select the check box, then updates to the security runtime configuration are made immediately, without first displaying the confirmation window. The confirmation window does not appear again until you re-select the checkbox located under Preferences.

  Or, instead of deselecting the check box under Preferences, you can select the **Do not show this message again** check box on the Update runtime confirmation panel.

1. Perform one of the following tasks:
   - **Services > Trust service > Trust service attachments**
   - **Services > Trust service > Token providers**
   - **Services > Trust service > Targets**
2. Click **Update Runtime**. If you did not select the **Show confirmation for update runtime command** check box, all changes are made immediately without displaying the confirmation window.
3. Optional: Confirm or cancel when the confirmation window appears. If you selected the **Show confirmation for update runtime command** check box, all changes require confirmation and the confirmation window is displayed.

Results

After clicking **Update Runtime**, the configuration changes you made on the Token providers, the Trust service attachments, or the Targets panel are updated for the Web services security runtime configuration.

What to do next

You can also manage and administer the trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

After configuring the trust service targets, attachments, and tokens, next configure the secure conversation client cache, if you have not already completed this step.

Web services update runtime settings:

Use this page to confirm that the Web services trust service runtime should be updated with the most recent configuration changes.

To view this administrative console page, complete one of the following procedures:

- **Services > Trust service > Trust service attachments**

- **Services > Trust service > Supported tokens**
- **Services > Trust service > Targets**

Make changes to attachments, tokens, or targets, save the changes, and click .**Update Runtime**.

Do not show this message again:

Specifies to enable or disable the confirmation panel before the Web services security runtime configuration is updated.

Click **OK** to confirm that you want to make the configuration changes effective immediately for the Web services trust service runtime.

Data type:	Check box
Default	Enabled (checked)

Configuring the Web services security distributed cache using the administrative console:

You can configure the Web services security runtime to use the security distributed cache to store security tokens.

About this task

Web services security functions such as secure conversation, trust, and nonce use the distributed cache to store security tokens when the distributed cache is enabled. If the distributed cache option is not selected, then a local cache is used to store the tokens. WebSphere Application Server Version supports distributed caching for the tokens in both cluster and non-cluster environments. In a cluster environment, you can configure the security cache to be distributed. If the cache is distributed, then all servers in the cluster share information about issued tokens.

1. To configure the secure conversation client cache, click **Services > Security cache**.
2. Change the time in minutes in the **Time token is in cache after timeout** field. The default value is 120 minutes. The minimum allowable time is 10 minutes, meaning you cannot enter a value that is less than 10 minutes. This field specifies the number of minutes that the token is in cache after the token expiration time expires (cache persist period).
3. Change the time in minutes in the **Renewal interval before token timeout** field. The default value and minimum allowable time is 10 minutes. You cannot enter a value that is less than 10 minutes. This field specifies the time period before the token expires when the client attempts to renew the token. This window of time is just before token expires where, if the token is accessed, then the client attempts to renew the token so that a downstream call can complete.

   It is important that this setting be set to a length of time that is longer than the longest possible transaction. This value must include the time it takes to transport to and from the server, the time that is needed by the server to process the request, and the time that is cached by reliable messaging, if appropriate. Setting this value to a length of time that is too small might result in the token expiring in the middle of a transaction and might prevent the transaction from completing.

   If the Security Context Token is renewed too often, it might cause Web Services Secure Conversation (WS-SecureConversation) to fail or even cause an out-of-memory error to occur. It is required that you set the renewal interval before the token expires value for the Secure conversation client cache to a value less than the token timeout value for the Security Context Token. It is also suggested that the token timeout value be at least two times the renewal interval before the token expires value.
4. Select the **Enable distributed caching** check box, if you want to share the tokens across the cluster. When the checkbox is selected to enable distributed caching, choose one of the following settings for updating the caches.
   - Synchronous update of cluster members: performs synchronous update of cache objects on cluster members (default).

- Asynchronous update of cluster members: performs a non-synchronous update of the cache on cluster members. This setting allows interoperability with cluster members that use the older style of updating as implemented in versions of WebSphere Application Server prior to version 7.0.
- Token recovery support: assigns a shared data source as the distributed cache.

If token recovery support is selected as the update method, then you must select a cell level data source using the drop-down list. Token state data is saved in the database defined as the data source. If there are no available data sources in the list, click on Manage data sources to add one or more new data source objects. The data source object supplies an application with connections for accessing the database.

5. To create a new custom property, click **New**. For example, you might add the cancelActionRST custom property with a value of `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel`.
6. To edit an existing custom property, select the check box for the name of the existing custom property, and then click **Edit**. For example, you might change the name or the value of the cancelActionRST custom property.
7. Click **Apply** to save and apply the changes.

Results

You have provided the basic information to configure the Web services security distributed cache. Use either the administrative console or the wsadmin tool to modify the security cache configuration.

What to do next

You can also add or delete custom properties for the trust service using the wsadmin tool. The wsadmin tool examples are written in the Jython scripting language.

Security cache settings:

Use this page to configure the Web Services Secure Conversation (WS-SecureConversation) security local and distributed cache settings using the administrative console.

To view this administrative console page, click **Services > Security cache**.

Time token is in cache after timeout:

Sets the time that the token remains in cache after the token times out.

This field specifies the number of minutes for the time the token is in cache after the token expiration time expires (cache persist period). For example, if you specify 30 minutes, the token is kept in cache for this time period after the token expiration time. The default value is 10 minutes, which is the minimum number of minutes that is allowed.

Data type:	Integer
Default:	10 (minutes)

Renewal interval before token timeout:

Sets the time period before expiration that the client attempts to renew the token.

This field specifies the period of time, in minutes, before expiration that the client attempts to renew the token. This setting must specify a period of time that is longer than the time for the longest transaction or else the token might expire during the transaction. This time must include time for transport to and from

the server, processing by the server, and any time delay that is because of time used for reliable messaging, when applicable. The default value is 10 minutes, which is the minimum number of minutes that is allowed.

If the Security Context Token is renewed too often, it might cause Web Services Secure Conversation (WS-SecureConversation) to fail or even cause an out-of-memory error to occur. It is required that you set the renewal interval before the token expires value for the security cache to a value less than the token timeout value for the Security Context Token. It is also suggested that the token timeout value be at least two times the renewal interval before the token expires value.

Data type: Integer
Default: 10 (minutes)

Enable distributed caching:

Specifies whether distributed caching is enabled or disabled. If distributed caching is enabled, select distributed cache settings.

Use this check box to specify whether to use distributed caching when the server is in a clustered environment and when the tokens are shared across the cluster.

Data type: Check box
Default: No distributed caching (unchecked)

When the checkbox is selected to enable distributed caching, choose one of the following settings for updating the caches.

Button	Resulting Action
Synchronous update of cluster members	Performs synchronous update of cache objects on cluster members (default).
Asynchronous update of cluster members	Performs a non-synchronous update of the cache on cluster members. This setting allows interoperability with cluster members that use the older style of updating as implemented in versions of IBM WebSphere Application Server prior to version 7.0.
Token recovery support	Assigns a shared data source as the distributed cache.

If token recovery support is selected as the update method, then you must select a cell level data source using the drop-down list. Token state data is saved in the database defined as the data source. If there are no available data sources in the list, click on **Manage data sources** to add one or more new data source objects. The data source object supplies an application with connections for accessing the database.

Custom Properties:

Specifies additional configuration settings that the secure conversation client might require.

This table lists custom properties. Use custom properties to set internal system configuration properties. This collection is empty until the first custom property is defined.

Data type: String

Select:

Specifies that you want to select further actions.

Use this check box to select custom properties for further actions. To manage existing custom properties, select the check box beside the name, and then select one of the following actions:

Table 52. Actions for custom properties

Actions	Description
Edit	Select to modify an existing custom property. This action is not displayed until you have added at least one custom property.
Delete	Select to remove an existing custom property.

Data type: Check box

New:

Specifies that you want to add and define a new custom property.

Click **New** to define a new custom property.

Data type: Button

Name:

Lists available custom properties.

This column displays the names of the custom properties that you can use with the secure conversation client (for example, `exampleProperty`). Custom properties are name-value pairs of data, where the name is a string representation of a property that is expected by the secure conversation client.

Data type: String

Value:

Lists the values of the custom properties.

This column displays the values of the custom properties (for example, `true`). Custom properties are name-value pairs of data, where the value is a string representation of the property setting.

Data type: String

Configuring the Kerberos token for Web services security

Use this topic to configure the Kerberos token for message-level Web services security.

Before you begin

Before you can use Kerberos with Web service security, you must configure Kerberos in the IBM WebSphere Application Server. You do not need to enable Kerberos as the authentication mechanism. However, the Kerberos configuration file, `krb5.conf` or `krb5.ini`, and the Kerberos keytab file, `krb5.keytab`, are required.

The initial setup and configuration processes to use Kerberos with Web services security are identical to the configuration processes for using Kerberos with the security function. Therefore, you must set up and configure Kerberos before continuing with the steps in this topic.

The Kerberos (KRB5) authentication mechanism support for security topic provides an overview of the Kerberos functionality and provides the initial steps for setting up and configuring Kerberos for authentication purposes. Within this topic, you must complete the steps in Setting up Kerberos as the authentication mechanism for WebSphere Application Server. Use that topic to configure Kerberos, the service principal, and the keytab files. In addition, that topic provides references to the process for configuring Kerberos as the authentication mechanism using the administrative console or commands. You can also find information on how to setup up Kerberos when the Key Distribution Center (KDC) and the Application Server do not use the same user registry.

About this task

Note: The support for Kerberos with Web services security in WebSphere Application Server Version 7.0 is based on the OASIS Web Service Security Kerberos Token Profile 1.1 specification.

The Kerberos token for JAX-WS applications is configured using policy sets and bindings. The JAX-WS application is attached with a custom policy and the Kerberos token is configured as a message protection token or an authentication token.

The implemented Kerberos functionality for Web services security also leverages existing tools and frameworks for the Kerberos token profile configuration for authentication and message protection.

To configure Kerberos with Web service security, complete the following steps:

1. Enable the Kerberos token profile for JAX-WS applications.

   The JAX-WS application is attached with a custom policy that has a Kerberos token, which is configured with a message protection token or an authentication token. For more information, see "Configuring the Kerberos token policy set for JAX-WS applications."
2. Select the customized Kerberos token type. You can define key bindings for request message protection and response message protection. You can use the key type, such as the key identifier or security token reference, for the outbound key information. If you use a derived key, use a security token reference in both the outbound and inbound key information. If you use a Kerberos session key, you can use a security token reference in the outbound key information and a key identifier in the inbound key information for the client bindings. Then, use a key identifier in the outbound key information and a security token reference in the inbound key information for the provider bindings.
3. Select the customized Kerberos token types for the token generator or token consumer.
4. Configure the bindings for Kerberos message protection for JAX-WS applications. For more information, see the "Configuring the bindings for message protection for Kerberos" on page 322.

What to do next

Using this task, you have configured the Kerberos token for WebSphere Application Server.

Configuring the Kerberos token policy set for JAX-WS applications:

Use this topic to enable the Kerberos token policy set for JAX-WS applications.

Before you begin

Prior to beginning this task, you must specify the Kerberos configuration information for IBM WebSphere Application Server. For more information, see Kerberos (KRB5) authentication mechanism support for security.

The configuration model for the Kerberos token enables you to choose from the following existing WebSphere Application Server frameworks:

- For JAX-RPC applications, the deployment descriptor and bindings are used in the configuration. JAX-RPC application includes the deployment descriptor for a Kerberos custom token, which is configured with authentication tokens.
- For JAX-WS applications, the configuration uses a policy set and bindings. The JAX-WS application is attached by a custom policy with the Kerberos token configured with authentication tokens, message protection tokens, or both.

About this task

Complete the following steps to configure the Kerberos token policy set for JAX-WS applications using the administrative console for WebSphere Application Server. In these steps, the Main policy configuation panel references the administrative console panel that is available after you complete the first five steps.

1. Expand **Services** → **Policy sets** and click **Application policy sets** → **New** to create a new policy set.
2. Specify a name and a short description for the new policy set and click **Apply**.
3. From the Policies heading, click **Add** and then select the **WS-Security** security policy type.
4. Click **OK** and click **Save** to save the new configuration directly to the master configuration.
5. In the **Policies** field, click **WS-Security** and click **Main policy** on the WS-Security panel to configure the main policy for the Kerberos token policy set.
6. From the Key Symmetry heading, select **Use symmetric tokens** for message protection.
7. Click **Symmetric signature and encryption policies** to configure the Kerberos custom token type or clear the **Message level protection** check box if you are configuring an authentication token only.

   Note: You do not need to configure the request token policy if you are using the Kerberos token for message protection. If you are configuring the authentication token only, proceed to the next step. If you are not configuring the request token policy for the authentication token, skip the next step.
8. On the Main policy configuration panel, configure the policy for the request token if you are configuring the authentication token.
   a. From the Policy Details heading, click **Request token policies**.
   b. Click **Add token type** and select **Custom**.
   c. Specify the name of the custom token in the **Custom token name** field.
   d. Specify the local part value in the **Local part** field. For interoperability with other Web services technologies, specify the following local part: `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. If you are not concerned with interoperability issues, you can specify one of the following local name values:
      - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ`
      - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510`
      - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510`
      - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120`
      - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120`

      These alternative values depend on the specification level for the Kerberos AP-REQ token that is generated by the Key Distribution Center (KDC). For more information about when to use these values, see Token type settings.
   e. Do not specify a value for the **Namespace URI** field if you are generating a Kerberos token.
   f. Click **OK** and **Save** to save the configuration directly to the master configuration.

This step completes the configuration process for configuring the request token policy for the authentication token. You do not need to complete the next two steps. Complete the next steps to configure encryption and symmetric signature policies.

9. Return to the main policy configuration panel for the application policy set and click **Symmetric signature and encryption policies** to configure the encryption and symmetric signature policies.
   a. From the Message Integrity heading, click the **Action** menu list beside the **Token type for signing and validating messages** field and select **Custom**.
   b. From the Message Confidentiality heading, select the **Use same token for confidentiality that is used for integrity** option.
   c. Click **OK** and **Save** to save the configuration changes.
   d. From the Message Integrity heading, click the **Action** menu list beside the **Token type for signing and validating messages** field and select **Edit Selected Type Policy**.
   e. Edit the custom token type for the signature and encryption by specifying the local part for the Kerberos custom token.

      For example, specify `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ` for the local part value. Do not specify a Namespace URI value.
   f. Click **OK** and then click the **Save** link to save the configuration changes.
10. Return to the main policy configuration panel for the application policy set and click **Algorithms for symmetric tokens** to configure the symmetric token algorithm.
    a. Select the algorithm suite to use for the symmetric tokens from the **Algorithm suite** menu list. Select the Advanced Encryption Standard (AES) algorithms for a Kerberos token that is compliant with RFC-4120.

       The symmetric key wrap, or private key cryptography, algorithms include:
       - Triple DES key wrap: http://www.w3.org/2001/04/xmlenc#kw-tripledes
       - AES key wrap (aes128): http://www.w3.org/2001/04/xmlenc#kw-aes128
       - AES key wrap (aes256): http://www.w3.org/2001/04/xmlenc#kw-aes256

       Note: To use the 256–bit AES encryption algorithm, you must apply the unlimited jurisdiction policy files. To remain in compliance, see Basic Security Profile compliance tips.

       Before downloading these policy files, back up the existing policy files prior to overwriting them, in case you want to restore the original files later. The existing policy files, which are the `local_policy.jar` and `US_export_policy.jar` files, are located in the *app_server_root*`/java/jre/lib/security/` directory.

       To download the policy files, complete one of the following sets of steps:
       - For application server platforms using IBM Developer Kit, Java Technology Edition Version 5, you can obtain unlimited jurisdiction policy files by completing the following steps:
         1) Visit the IBM developerWorks: Security Information Web site.
         2) Click **Java 5**.
         3) Click **IBM SDK Policy files**.

            The Unrestricted JCE Policy files for SDK 5 Web site is displayed.
         4) Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your workstation.

       For more information on the algorithm suite components, see Algorithms settings.
    b. Select either the **Exclusive cannonicalization** or **Inclusive cannonicalization** value for the **Cannonicalization algorithm** menu list. For more information, see XML digital signature.
    c. Specify the **XPath 1.0** or **XPathfilter 2.0** version to use from the **XPath version** menu list.

What to do next

Configure the bindings for message protection for Kerberos for JAX-WS applications. For more information, see “Configuring the bindings for message protection for Kerberos.”

Configuring the bindings for message protection for Kerberos:

To set up bindings for message protection with JAX-WS applications, you must create a custom binding. Complete this task to set the bindings for a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile Version 1.1.

Before you begin

You must configure Kerberos for IBM WebSphere Application Server. For more information, see Kerberos (KRB5) authentication mechanism support for security. In addition, you must configure the Kerberos token policy set for JAX-WS applications. For more information, see Configuring the Kerberos token policy set for JAX-WS applications.

About this task

You can leverage existing frameworks including the policy set and bindings for JAX-WS applications.

You can configure a symmetric protection token or an authentication token. Both symmetric protection token and authentication token configurations use similar configuration data. However, you do not need to configure the authentication token if you intend to use a Kerberos symmetric protection token. For whichever token type you use, configure the token generator and the token consumer as indicated in the following list:

- Symmetric protection token
  - Token generator
  - Token consumer
- Authentication token
  - Token generator
  - Token consumer

Use the administrative console to configure the application-specific bindings to use a Kerberos token in Web services message protection.

1. Expand **Applications** → **Application Types**.
2. Click **WebSphere enterprise applications** → ***application name*** .
3. From the Web Services Properties heading, click **Service provider policy sets and bindings** to configure the service bindings or click **Service client policy sets and bindings** to configure the client bindings.
4. Select the resource to attach to the Kerberos token policy set and select **Attach Policy Set** → ***policy set name***. To configure the Kerberos token policy set, see “Configuring the Kerberos token policy set for JAX-WS applications” on page 319.
5. Click **Assign bindings** and select the application-specific binding or select **New Application Specific Binding** to create a new binding. To create a new binding, complete the following actions.
   a. Enter a name for the new binding in the **Binding configuration name** field and optionally enter a description for the binding in the **Description** field.
   b. Click **Add** and select **WS-Security** to specify a new policy set.
   c. Click **Authentication and protection** → **New**.
   d. Optional: Define a symmetric protection token for the token generator.

Note: If you configure a symmetric protection token for the token generator, you must define a complimentary symmetric protection token for the token consumer.

1) From the Protection tokens heading, click **New** and select **Token Generator**.
2) Specify the name of the protection token in the **Name** field.
3) Select **Custom** from the values in the **Token type** menu list.
4) Specify the local name value in the **Local name** field.

   For interoperability with other Web services technologies, specify the following local name: `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. If you are not concerned with interoperability issues, you can specify one of the following local name values:

   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120`

   These alternative values depend on the specification level for the Kerberos token that is generated by the Key Distribution Center (KDC). For more information about when to use these values, see Protection token settings (generator or consumer).
5) Do not specify a value for the **Namespace URI** field.
6) Select the **wss.generate.KRB5BST** value from the JAAS login menu list.

   If you have previously defined your own Java Authentication and Authorization Service (JAAS) login module, you can select your login module to handle the Kerberos custom token. To define a custom JAAS login module, click **New Application Login** → **New**, specify an alias for the new module, and click **Apply**. For more information, see Login module settings for Java Authentication and Authorization Service.

   Note: Although the information in the ″Login module settings for Java Authentication and Authorization Service″ topic refers to security and not Web services security, the configuration for a login module for Web services security is identical to security.
7) Specify the token generator custom properties for the target service name, host, and realm.

   The combination of the target service name and host values forms the Service Principal Name (SPN), which represents the target Kerberos service principal name. The Kerberos client requests the initial Kerberos AP_REQ token for the SPN. Specify the following custom properties.

Table 53. Target service custom properties

Name	Value	Type
`com.ibm.wsspi.wssecurity.krbtoken.targetServiceName`	Specify the name of the target service.	Required
`com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost`	Specify the host name that is associated with the target service in the following format: `myhost.mycompany.com`	Required
`com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm`	Specify the name of the realm that is associated with the target service.	Optional

   To specify multiple custom property name and value pairs, click **New**.
8) Click **Apply**.
9) From the Additional bindings heading, click **Callback handler**.
10) From the Class Name heading, select the **Use custom** option and specify `com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler` in the associated field.

11) From the Basic Authentication heading, specify the appropriate values for the **User name**, **Password**, and **Confirm password** fields.

The user name specifies the default user ID that is passed to the constructor of the callback handler; for example, `kerberosuser`.

12) Specify the token generator custom properties for Kerberos client principal name and password to initiate the Kerberos login.

These custom properties control the prompt and establish the token based on the credential cache. Specify the following custom properties.

Table 54. Kerberos login custom properties

Name	Value	Type
`com.ibm.wsspi.wssecurity.krbtoken.loginPrompt`	Enables the Kerberos login when the value is `True`. The default value is `False`.	Optional

To specify multiple custom property name and value pairs, click **New**.

13) Click **Apply** and **OK**.

When you return to the Authentication and protection panel in the next step, a new protection token is defined for the token generator. To edit the configuration for this new token, click its name on the panel.

e. Optional: Return to the Authentication and protection panel to define a symmetric protection token for the token consumer. To return to the Authentication and protection panel, click the **Authentication and protection** link after the messages section of the panel.

Note: If you configure a symmetric protection token for the token consumer, ensure that you have previously defined a complimentary symmetric protection token for the token consumer.

1) From the Protection tokens heading, click **New** and select **Token Consumer**.
2) Specify the name of the protection token in the **Name** field.
3) Select **Custom** from the values in the **Token type** menu list.
4) Specify the local name value in the **Local name** field.

For interoperability with other Web services technologies, specify the following local name: `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. If you are not concerned with interoperability issues, you can specify one of the following local name values:

- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ`
- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510`
- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510`
- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120`
- `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120`

These alternative values depend on the specification level for the Kerberos token that is generated by the Key Distribution Center (KDC). For more information about when to use these values, see Protection token settings (generator or consumer).

5) Do not specify a value for the **Namespace URI** field.
6) Select the **wss.consume.KRB5BST** value from the JAAS login drop-down menu.

If you have previously defined your own Java Authentication and Authorization Service (JAAS) login module, you can select this login module to handle the Kerberos custom token. To

define a custom JAAS login module, click **New Application Login** → **New**, specify an alias for the new module, and click **Apply**. For more information, see Login module settings for Java Authentication and Authorization Service.

Note: Although the information in the Login module settings for Java Authentication and Authorization Service topic refers to security and not Web services security, the configuration for a login module for Web services security is identical to security.

7) Click **Apply**.
8) From the Additional bindings heading, click **Callback handler**.
9) From the Class Name heading, select the **Use custom** option and specify `com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler` in the associated field.
10) Click **Apply** and **OK**.

When you return to the Authentication and protection panel in the next step, you will see a new protection token defined for the token consumer. To edit the configuration for this new token, click its name on the panel.

f. Optional: Return to the Authentication and protection panel to define an authentication token configuration for the token generator. To return to the Authentication and protection panel, click the **Authentication and protection** link after the messages section of the panel.

Authentication tokens are sent in messages to prove or assert an identity.

Note: If you configure an authentication token for the token generator, you must define a complimentary authentication token for the token consumer.

1) From the Authentication tokens heading, click **New** and select **Token Generator**.
2) Specify the name of the authentication token in the **Name** field.
3) Select **Custom** from the values in the **Token type** menu list.
4) Specify the local name value in the **Local name** field.

   For interoperability with other Web services technologies, specify the following local name: `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. If you are not concerned with interoperability issues, you can specify one of the following local name values:

   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120`

   These alternative values depend on the specification level for the Kerberos token that is generated by the Key Distribution Center (KDC). For more information about when to use these values, see Authentication generator or consumer token settings.
5) Do not specify a value for the **Namespace URI** field.
6) Select the **wss.generate.KRB5BST** value from the JAAS login menu list.

   If you have previously defined your own Java Authentication and Authorization Service (JAAS) login module, you can select this login module to handle the Kerberos custom token. To define

a custom JAAS login module, click **New Application Login** → **New**, specify an alias for the new module, and click **Apply**. For more information, see Login module settings for Java Authentication and Authorization Service.

Note: Although the information in the Login module settings for Java Authentication and Authorization Service topic refers to security and not Web services security, the configuration for a login module for Web services security is identical to security.

7) Specify the token generator custom properties for the target service name, host, and realm.

The combination of the target service name and host values forms the Service Principal Name (SPN), which represents the target Kerberos service principal name. The Kerberos client requests the initial Kerberos AP_REQ token for the SPN. Specify the following custom properties.

Table 55. Target service custom properties

Name	Value	Type
com.ibm.wsspi.wssecurity.krbtoken.targetServiceName	Specify the name of the target service.	Required
com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost	Specify the host name that is associated with the target service in the following format: myhost.mycompany.com	Required
com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm	Specify the name of the realm that is associated with the target service.	Optional

To specify multiple custom property name and value pairs, click **New**.

8) Click **Apply**.
9) From the Additional bindings heading, click **Callback handler**.
10) From the Class Name heading, select the **Use custom** option and specify com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler in the associated field.
11) From the Basic Authentication heading, specify the appropriate values for the **User name**, **Password**, and **Confirm password** fields.

The user name specifies the default user ID that is passed to the constructor of the callback handler. For example: kerberosuser

12) Specify the token generator custom properties for Kerberos client principal name and password to initiate the Kerberos login.

These custom properties control the prompt and establish the token based on the credential cache. Specify the following custom properties name and value pairs.

Table 56. Kerberos login custom properties

Name	Value	Type
com.ibm.wsspi.wssecurity.krbtoken.loginPrompt	Enables the Kerberos login when the value is True. The default value is False.	Optional

To specify multiple custom property name and value pairs, click **New**.

13) Click **Apply** and **OK**.

When you return to the Authentication and protection panel in the next step, you will see a new authentication token is defined for the token generator. To edit the configuration for this new token, click its name on the panel.

g. Optional: Return to the Authentication and protection panel to define an authentication token configuration for the token consumer. To return to the Authentication and protection panel, click the **Authentication and protection** link after the messages section of the panel.

Note: If you configure an authentication token for the token consumer, ensure that you have previously defined an authentication token for the token consumer.

1) From the Protection tokens heading, click **New** and select **Token Consumer**.
2) Specify the name of the protection token in the **Name** field.

3) Select **Custom** from the values in the **Token type** menu list.
4) Specify the local name value in the **Local name** field.

   For interoperability with other Web services technologies, specify the following local name: `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. If you are not concerned with interoperability issues, you can specify one of the following local name values:

   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120`
   - `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120`

   These alternative values depend on the specification level for the Kerberos token that is generated by the Key Distribution Center (KDC). For more information conditions under which to use these values, see the related link for the ″Authentication generator or consumer token settings″ topic.
5) Do not specify a value for the **Namespace URI** field.
6) Select the **wss.consume.KRB5BST** value from the JAAS login drop-down menu.

   If you have previously defined your own Java Authentication and Authorization Service (JAAS) login module, you can select this login module to handle the Kerberos custom token. To define a custom JAAS login module, click **New Application Login** → **New**, specify an alias for the new module, and click **Apply**. For more information, see Login module settings for Java Authentication and Authorization Service.

   Note: Although the information in the Login module settings for Java Authentication and Authorization Service topic refers to security and not Web service security, the configuration for a login module for Web services security is identical to security.
7) Click **Apply**.
8) From the Additional bindings heading, click **Callback handler**.
9) From the Class Name heading, select the **Use custom** option and specify `com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler` in the associated field.
10) Click **Apply** and **OK**.

When you return to the Authentication and protection panel in the next step, you will see a new authentication token is defined for the token consumer. To edit the configuration for this new token, click its name on the panel.

What to do next

You can optionally define key bindings for the request message protection and response message protection. If you choose to derive a key from the Kerberos token, configure the derived key information when you configure the key information for signature and encryption.

Return to the steps in the Configuring the Kerberos token for Web services security topic to ensure you have completed the steps for configuring the Kerberos token.

Securing JAX-RPC Web services using message level security

Version 6 and later applications

Standards and profiles address how to provide protection for messages that are exchanged in a Web service environment.

Before you begin

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

About this task

To secure Web services with WebSphere Application Server, you must specify several different configurations. Although there is not a specific sequence in which you must specify these different configurations, some configurations reference other configurations. See “Web services security configuration considerations” on page 81.

Web service security is supported in the managed Web service container. To establish a managed environment and to enforce constraints for Web services security, you must perform a Java Naming and Directory Interface (JNDI) lookup on the client to resolve the service reference.

Because of the relationship between the different Web services security configurations, it is recommended that you specify the configurations on each level of the configuration in the following order. You can choose to configure Web services security for the application level or the server level as it depends upon your environment and security needs.

- Assemble your Web services security-enabled application by using an assembly tool. Prior to modifying a Web services security-enabled application in the WebSphere Application Server administrative console, you must assemble your application using an assembly tool. Although you can modify some of the application settings using the administrative console, you must configure the generator and the consumer security constraints using an assembly tool. Return to this article after you have assembled your application and imported it into the administrative console. For more information on assembly tools, see the topic Assembly tools.
- Modify the application-level configurations in the administrative console.
  1. Configure the trust anchors for the generator binding.
  2. Configure the collection certificate store for the generator binding.
  3. Configure the token for the generator binding.
  4. Configure the key locators for the generator binding.
  5. Configure the key information for the generator binding.
  6. Configure the signing information for the generator binding.
  7. Configure the encryption information for the generator binding.
  8. Configure the trust anchors for the consumer binding.
  9. Configure the collection certificate store for the consumer binding.
  10. Configure the token for the consumer binding.
  11. Configure the key locators for the consumer binding.

12. Configure the key information for the consumer binding.
13. Configure the signing information for the consumer binding.
14. Configure the encryption information for the consumer binding.

- Specify the server-level configurations.
  1. Configure the trust anchors for the server level.
  2. Configure the collection certificate store for the server level.
  3. Configure a token generator for the server level.
  4. Configure a nonce for the server level.
  5. Configure the key locators for the generator binding.
  6. Configure the key information for the generator binding.
  7. Configure the signing information for the generator binding.
  8. Configure the encryption information for the generator binding.
  9. Configure the trusted ID evaluators for the server level.
  10. Configure a token consumer for the server level.
  11. Configure the key information for the consumer binding.
  12. Configure the signing information for the consumer binding.
  13. Configure the encryption information for the consumer binding.

Results

After completing these steps for WebSphere Application Server, you have secured Web services.

Migrating JAX-RPC Web services security applications to Version 7.0 applications

Version 6 and later applications

Migration of a Java Platform, Enterprise Edition (Java EE) Version 1.3 application that uses Web services security to a Java EE Version 1.4 application is possible.

Before you begin

You can install Java Platform, Enterprise Edition (Java EE) Version 1.3 applications that use Web services security on a WebSphere Application Server Version 7.0 server. However, if you want Java EE Version 1.3 applications to use the Web services security (WSS) Version 1.0 or 1.1 specifications and the other new features added in Version 7.0, you must migrate the Java EE Version 1.3 applications to Java EE Version 1.4.

About this task

Complete the following steps to migrate a Version 1.3 application, along with the Web services security configuration information, to a Version 7.0 application:

1. Save the original Java EE Version 1.3 application. You need the Web services security configuration files of the Java EE Version 1.3 application to recreate the configuration in the new format for the Java EE Version 1.4 application.
2. Use the Java Platform, Enterprise Edition (Java EE) Migration Wizard in an assembly tool to migrate the Java EE Version 1.3 application to Java EE Version 1.4.

   Note: After you migrate to Java EE Version 1.4 using the Java EE Migration Wizard, you cannot view the Java EE Version 1.3 extension and binding information within an assembly tool. You can view the Java EE Version 1.3 Web services security extension and binding information using a text editor. However, do not edit the extension and binding information using a text editor. The

Java EE Migration Wizard does not migrate the Web services security configuration files to the new format in the Java EE Version 1.4 application. Rather the wizard is used to migrate your files from Java EE Version 1.3 to Version 1.4.

To access the Java EE Migration Wizard, complete the following steps:

a. Right-click the name of your application.

b. Click **Migrate > Java EE Migration Wizard**.

3. Manually delete all of the Web services security configuration information from the binding and extension files of the application that is migrated to Java EE Version 1.4.

   a. Delete the `<securityRequestReceiverServiceConfig>` and `<securityResponseSenderServiceConfig>` sections from the server-side `ibm-webservices-ext.xmi` extension file.

   b. Delete the `<securityRequestReceiverBindingConfig>` and `<securityResponseSenderBindingConfig>` sections from the server-side `ibm-webservices-bnd.xmi` binding file.

   c. Delete the `<securityRequestSenderServiceConfig>` and `<securityResponseReceiverServiceConfig>` sections from the client-side `ibm-webservicesclient-ext.xmi` extension file.

   d. Delete the `<securityRequestSenderBindingConfig>` and `<securityResponseReceiverBindingConfig>` sections from client-side `ibm-webservicesclient-bnd.xmi` binding file.

4. Recreate the Web services security configuration information in the new Java EE Version 1.4 format. At this stage, because the application is already migrated to the Java EE Version 1.4, use an assembly tool to configure the original Web services security information in the new Version 7.0 format. For more information on assembly tools, see the related information.

Results

This task provides general information about how to migrate Java EE Version 1.3 applications to Java EE Version 1.4.

What to do next

The following articles contain some general scenarios that map some of the basic Web services security information specified in a Java EE Version 1.3 application to a Java EE Version 1.4 application and specify this information using an assembly tool. The Web services security configuration information is contained in four configuration files: two server-side configuration files and two client-side configuration files. The migration of all of the configuration information is divided into four sections; one for each configuration file. When you recreate the Web services security information in the new Java EE Version 1.4 format, it is recommended that you configure the extensions and binding files in the following order:

1. Configure the `ibm-webservices-ext.xmi` server-side extensions file. For more information, see “Migrating the JAX-RPC server-side extensions configuration.”
2. Configure the `ibm-webservicesclient-ext.xmi` client-side extensions file. For more information, see “Migrating the client-side extensions configuration” on page 333.
3. Configure the `ibm-webservices-bnd.xmi` server-side bindings file. For more information, see “Migrating the server-side bindings file” on page 334.
4. Configure the `ibm-webservicesclient-bnd.xmi` client-side bindings file. For more information, see “Migrating the client-side bindings file” on page 336.

Migrating the JAX-RPC server-side extensions configuration:

Version 6 and later applications

You can migrate the Web services security server-side extensions configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application for the JAX-RPC programming model.

About this task

The following table lists the mappings for the top-level sections under the server-side **Security Extensions** tab within an assembly tool from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 57. The mapping of the configuration sections

Java EE Version 1.3 extensions configuration	Java EE Version 1.4 extensions configuration
Request Receiver Service Configuration Details	Request Consumer Service Configuration Details
Response Sender Service Configuration Details	Response Generator Service Configuration Details

For information about the assembly tools that are available for WebSphere Application Server Version 6.0.*x*, see Assembly tools.

Consider the following steps to migrate the server-side extensions from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration.

- Import the Java EE Version 1.3 application into an assembly tool and identify all the message parts that are required to be signed and encrypted. The message parts are listed in the Required Integrity and Required Confidentiality sections under the Request Receiver Service Configuration Details section. In a Java EE Version 1.4 application, these message parts map to the Message parts field of the **Required integrity** and **Required confidentiality** dialogs windows within the assembly tool.

  To specify these message parts within an assembly tool, complete the following steps in the Web Services editor. The steps are based on typical scenarios, but the steps are not all-inclusive.

  1. Click the **Extensions** tab.
  2. Navigate to the Required integrity subsection within the Request Consumer Service Configuration Details section.
  3. Specify each message part to be signed in the Message Parts field.

  For example, if the message part in the Java EE Version 1.3 application is `body`, you need to specify **body** in the Message parts keyword field. Similarly, on the **Extensions** tab, configure the message parts to be encrypted using the Required Confidentiality dialog. Also, for all the message parts that are migrated from a Java EE Version 1.3 application, you must select **http://www.ibm.com/websphere/webservices/wssecurity/dialect-was** in the Message parts dialect field and **Required** in the Usage type field.
- Optional: Configure the Required Security Token and Caller Part sections on the **Extensions** tab if the authentication method of BasicAuth is configured under the Login Config section of the Java EE Version 1.3 application. When you configure the Required Security Token section, select **Username** in the name field and **Required** in the Usage type field within the Required Security Token Dialog window. The following table shows how the authentication method values for a Java EE Version 1.3 application map to the token type values within the Java EE Version 1.4 application.

Table 58. Authentication method to token type mappings

Login Config Authentication method values in the Java EE Version 1.3 extensions configuration	Token type values in the Java EE Version 1.4 extensions configuration
`BasicAuth`	`UsernameToken`
`Signature`	`X509 certificate`
`LTPA`	`LTPAToken`

If the authentication method value is IDAssertion within the Login Config section, the token type that you

must specify in the Java EE Version 1.4 application depends upon the IDType value within the IDAssertion section. The following table shows how the IDType values for Java EE Version 1.3 application map to the token type values in the Java EE Version 1.4 application.

Table 59. IDType values to token type mappings

IDType values in the Java EE Version 1.3 application extensions configuration	Token type values in the Java EE Version 1.4 application extensions configuration
X509Certificate	X509 certificate
Username	Username

- Select the appropriate token type in the Name field of the Call Part Dialog window based on the previous two tables. Select the **Username** token type when you are configuring the caller part for the basic authentication method. Configuring the other token types in the Caller part dialog is similar to configuring token types in the Required Security Token dialog. If you need to map the IDAssertion authentication method from a Java EE Version 1.3 application to a Java EE Version 1.4 application, select the **Use IDAssertion** option and configure the ID assertion section of the Caller Part Dialog window. The Trust Mode field under the IDAssertion section maps to the Trust method name field of the Trust method property section in the Caller Part Dialog window. If Signature is selected for the Trust method, specify the Required Integrity part that specifies the signature of the trusted intermediary certificate.
- Configure a nonce in the Version 7.0 Binding Configurations section if nonce is specified in the Add Authentication Method dialog under Login Config within the Java EE Version 1.3 application extensions configuration.

  Note: Nonce is configured in the bindings for a Java EE Version 1.4 application and not in the extensions.

  To configure a nonce on the Binding Configurations tab, set the com.ibm.wsspi.wssecurity.token.Username.verifyNonce property in the Token Consumer configuration for the Username token.
- Configure the Add Timestamp section to migrate the time stamp information if the <addReceivedTimestamp> element is configured in the Java EE Version 1.3 extensions. To migrate the Response Sender Service Configuration Details section in the Java EE Version 1.3 extensions, identify all of the message parts listed within the Integrity and Confidentiality sections. Configure these message parts using the Integrity and Confidentiality dialogs under the Response Generator Service Configuration details section. This configuration is similar to the configuration for Required Integrity and Required Confidentiality, with the exception of the Order field in the Integrity Dialog. The value of this Order field specifies the order in which the message parts specified in the Message Parts field are digitally signed or encrypted in the SOAP message. For example, the extensions contain the following information:
  - One integrity entry called `int_part1` with a value of `1` in the Order field
  - One confidentiality entry called `conf_part1` with a value of `2` in the Order field

  In this example, the message parts that are specified by the `int_part1` integrity entry are signed before the message parts specified by the `conf_part1` confidentiality entry are encrypted. The same rule for the order attribute applies for multiple integrity or confidentiality elements.

Results

These steps describe the types of information that you need to migrate the Web services security server-side extensions for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the client-side extensions for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the client-side extensions configuration” on page 333.

Migrating the client-side extensions configuration: Version 6 and later applications

You can migrate the Web services security client-side extensions configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mappings of the top-level sections under the client-side **Security Extensions** tab for Web services security from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 60. The mapping of the configuration sections

Java EE Version 1.3 security extensions for Web services security	Java EE Version 1.4 extensions for Web services security
Request Sender Configuration	Request Generator Configuration
Response Receiver Configuration	Response Consumer Configuration

Consider the following steps to migrate the client-side extensions configuration from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

- Migrate the message parts that you need to sign or encrypt from the Integrity and Confidentiality sections in the Java EE Version 1.3 application to the Integrity and Confidentiality sections on the **WS Extensions** tab in an assembly tool for a Java EE Version 1.4 application.
- Configure the Security Token section under the Request Generator Configuration on the **WS Extensions** tab if Login Config section is configured in the Java EE Version 1.3 extensions configuration. When you configure the security token, select the token type in the Token type field that matches the authentication method value of the Login Config in the Java EE Version 1.3 application. For example, if the authentication method in the Java EE Version 1.3 extensions configuration is BasicAuth, then select **Username** in the Token type field within the assembly tool. For more information on how the authentication methods for Web services security map from a Java EE Version 1.3 application to a Java EE Version 1.4 application, see Table 58 on page 331. If the authentication method is IDAssertion, there is no action required because in a Java EE Version 1.4 application the identity assertion configuration is not required in the client-side extensions configuration. In a Java EE Version 1.4 application, the identity assertion configuration is specified in the server-side extensions configuration and in the client-side bindings configuration.
- Migrate the Required Integrity and Required Confidentiality sections by configuring the Required Integrity and Required Confidentiality sections in an assembly tool. Migrating the Response Receiver Configuration section is similar to migrating the Request Receiver Service Configuration Details section of the server-side extensions configuration. For more information, see “Migrating the JAX-RPC server-side extensions configuration” on page 330.
- Migrate the nonce configuration in the Login Config section in a Java EE Version 1.3 extensions configuration for Web services security to a Java EE Version 1.4 application.

  Note: Nonce is not configured in a Java EE Version 1.4 extension file for Web services security. Rather, it is configured in the binding file for Web services security.

  To configure a nonce in the binding file, define the com.ibm.wsspi.wssecurity.token.username.addNonce property in the token generator of the username token.
- Configure the Add Timestamp section under the Request Generator Configuration in the assembly tool if the **Add Created Time Stamp** option is configured in the Java EE Version 1.3 extensions.

Results

This set of steps describe the types of information that you need to migrate the client-side extensions configuration for Web services security for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the server-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the server-side bindings file.”

Migrating the server-side bindings file: Version 6 and later applications

You can migrate the server-side bindings configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mappings of the top-level sections under the server-side **Binding Configurations** tab from a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 61. The mapping of the configuration sections

Java EE Version 1.3 Binding Configurations	Java EE Version 1.4 Binding Configurations
Request Receiver Binding Configuration Details	Request Consumer Service Binding Configuration Details
Response Sender Binding Configuration Details	Response Generator Binding Configuration Details

Consider the following steps to migrate the server-side bindings from Java EE Version 1.3 to Java EE Version 1.4. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

- Migrate the configuration information under the Request Receiver Binding Configuration Details section of a Java EE Version 1.3 application.
  1. Migrate any trust anchor information that is specified in the Java EE Version 1.3 application to Java EE Version 1.4 using the Trust Anchor dialog.
  2. Migrate the information under the certificate store list that is specified in the Java EE Version 1.3 application to Java EE Version 1.4 by configuring the Certificate Store List section in the Java EE Version 1.4 application.
  3. Configure the key locator and token consumer information that is referenced from the Key Information dialog window. The configuration of the key locator and the token consumer depends upon the key information type. For example, if an X.509 certificate that is embedded in the `<wsse:Security>` security header is used for digital signature, complete the following steps:
     a. For configuring the key locator, specify the com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator class as the key locator class and do not specify a key store.
     b. For configuring the token consumer, select the com.ibm.wsspi.wssecurity.token.509TokenConsumer class, specify `X509 certificate token` for the value type Uniform Resource Identifier (URI), and specify `system.wssecurity.X509BST` in the jaas.config.name field. Also, you must specify the certificate path settings (the trust anchor reference and the certificate store reference) as part of the token consumer configuration.
  4. Explicitly specify the key information type in the Key Information Dialog window. In a Java EE Version 1.3 application, the key information type, such as the security token reference and the key identifier, is not explicitly specified. The key information type is implied by the configuration. In a Java EE Version 1.4 application, you must specify the key information type explicitly using the Key

Information Dialog when you have digital signature or encryption information in the binding file. Before you configure the key information, make sure that you have configured the key locator and token consumer information that is referenced from the Key Information dialog.

When you configure the key information for either digital signature or encryption, you need to specify the correct key information type. The value of the key information type depends upon the type of mechanism that is used to reference the security token that is used for digitally signing or encrypting. The following information describes the Security token reference (or Direct reference) and the Key identifier, which are the most common, recommended key information types that are used for digitally signing and encrypting:

Security token reference (or Direct reference)

The security token is directly referenced using the Uniform Resource Identifiers (URIs). The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Reference URI="#mytoken" />
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

Key identifier

The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the `KeyIdentifier` value depends upon the token type. For example, a hash of the important elements of the security token is used for generating the `KeyIdentifier` value. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

In the Key Information Dialog window, specify the names of the key locator and the token consumer that you configured previously. The Key name field is optional for the consumer side.

5. Migrate the information in the Signing Information section by configuring the Signing Information, Part References, and Transforms sections.
   - Specify the Signature method and Canonicalization method algorithms in the Signing Information Dialog window.
   - Specify the Digest method algorithm in the Part Reference Dialog window.
6. Migrate the information under the Encryption Information section. In the Encryption Information Dialog window, select the name of the Key Information element that is configured for encryption, and specify the `RequiredConfidentiality` part. Verify that the value for the selected `RequiredConfidentiality` part is the same name as the Required Confidentiality part that is configured in the extension file.

   The Login Mapping section in the Java EE Version 1.3 application maps to the Token Consumer configuration for the type of token that is specified by the authentication method. For example, to migrate a Login Mappings configuration that uses the `BasicAuth` authentication method, configure a token consumer for the username token. To configure a token consumer for a username token, complete the following steps:
   a. Select the com.ibm.wsspi.wssecurity.UsernameTokenConsumer token consumer class.
   b. Specify the name of the Required Security Token configuration from the Extensions within in the Security Token field.
   c. Select **Username Token** for value type.
   d. Specify the system.wssecurity.UsernameToken value in the jaas.config.name field.

- Migrate the configuration information in the Response Sender Binding Configuration Details section of the Java EE Version 1.3 bindings file to the Response Generator Binding Configuration Details section of the Java EE Version 1.4 application. Configuring the Response Generator section is very similar to configuring the Request Consumer section.
  1. Migrate the information from the Key Locators section by using the Key Locator Dialog window in an assembly tool.
  2. Configure a token generator, which is referenced in the Key Information Dialog window. You must configure a token generator for every security token that is generated in the SOAP message. If the token generator is for an X.509 certificate that is used for digital signature or encryption, complete the following steps:
     a. For configuring the key locator, specify the com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator class as the key locator class and do not specify a key store.
     b. For configuring the token generator, select the com.ibm.wsspi.wssecurity.X509TokenGenerator class and specify `X509 certificate token` for the value type Uniform Resource Identifier (URI). The key store information that is specified for the token generator is the same information that is used for configuring the key locator. Therefore, the keystore information from the Key Locators configuration in a Java EE Version 1.3 application is used to configure the key locator and the token generator in a Java EE Version 1.4 application.
     c. In the Token Generator Dialog window, specify the key store information that is required by the callback handler to obtain the key information that is required for generating the token.
     d. For the callback handler, select the com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler class.
  3. Specify the names of the key locator and the token generator in the Key Information Dialog window that you configured previously. The Key name is required for the generator side. The key that is specified in the Key Information Dialog window must exist in the list of keys that is specified in the key locator configuration. Also, migrating the Signing Information and the Encryption Information configurations is similar to migrating the Signing Information and the Encryption Information configurations for the Request Receiver Binding Configuration section. Configuring the key information for the response generator section is similar to configuring the key information for the request consumer section.

Results

This set of steps describes the types of information that you need to migrate the server-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Migrate the client-side binding configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application. For more information, see “Migrating the client-side bindings file.”

Migrating the client-side bindings file: Version 6 and later applications

You can migrate the Web services security client-side binding configuration for a Java Platform, Enterprise Edition (Java EE) Version 1.3 application to a Java EE Version 1.4 application.

About this task

The following table lists the mapping of the top-level sections under the client-side **Port Bindings** tab within a Java EE Version 1.3 application to a Java EE Version 1.4 application.

Table 62. The mapping of the configuration sections

Java EE Version 1.3 binding configuration for Web services security	Java EE Version 1.4 binding configuration for Web services security
Security Request Sender Binding Configuration	Security Request Generator Binding Configuration
Security Response Receiver Binding Configuration	Security Response Consumer Binding Configuration

Consider the following steps to migrate the client-side binding configuration from a Java EE Version 1.3 application to a Java EE Version 1.4 application. These steps are dependent upon your specific configuration. The steps are based on typical scenarios, but the steps are not all-inclusive.

- Migrate the information in the Security Request Sender Binding Configuration section in a Java EE Version 1.3 application to a Java EE Version 1.4 application. The migrations process for the Security Request Sender Binding Configuration section is similar to the process for the Response Sender Binding Configuration Details section in the server-side binding configuration. For more information, see "Migrating the server-side bindings file" on page 334.
- Migrate the information in the Key Locators, Signing Information, and the Encryption Information sections of the Java EE Version 1.3 application to a Java EE Version 1.4 application. The migration process for these elements on the client side is similar to migration process on the server side. For more information, see "Migrating the server-side bindings file" on page 334.
- Migrate the information in the Login Bindings section in a Java EE Version 1.3 application to a Java EE Version 1.4 application. The migration of the Login Bindings section depends upon the value of the authentication method. If the authentication method is `BasicAuth` or `IDAssertion`, configure a token generator for the username token. If the authentication method is LTPA, select the com.ibm.wsspi.wssecurity.token.LTPATokenGenerator class as the token generator class. If the client-side bindings for the Web service uses IDAssertion, complete the following steps:
  1. Configure a token generator for the authentication token of the original client.
  2. Define the com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed property and set its value to `true` in the Token Generator Dialog window within an assembly tool. If the original client is using a username token for authentication and if the target Web service is using `BasicAuth` for authentication, configure the following token generators in the client-side binding file:
     - The username token of the original client. You must set the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property in the token generator of the original client.
     - The username token of the intermediary Web service.
- Migrate the Security Response Receiver Binding Configuration section from a Java EE Version 1.3 application to a Java EE Version 1.4 application. Migrating the Security Response Receiver Binding Configuration section is similar to migrating the Request Receiver Binding Configuration Details section of the server-side bindings configuration. Migrate this information under the Security Response Consumer Binding Configuration section. For more information, see "Migrating the server-side bindings file" on page 334.

  To configure a nonce in the binding file, define the com.ibm.wsspi.wssecurity.token.username.addNonce property in the token generator of the username token.

Results

This set of steps describe the types of information that you need to migrate the Web services security client-side bindings configuration for a Java EE Version 1.3 application to a Java EE Version 1.4 application.

What to do next

Verify that you have migrated both the server-side and the client-side extension and binding configurations for a Java EE Version 1.3 application to a Java EE Version 1.3 application. For more information, see

“Migrating JAX-RPC Web services security applications to Version 7.0 applications” on page 329.

View Web services client deployment descriptor: **Version 6 and later applications**

Use this page to view your client deployment descriptor.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Before you begin this task, the Web services application must be installed.

By completing this task, you can gather information that enables your to maintain or configure binding information. After the Web services application is installed, you can view the Web services deployment descriptors.

To view this administrative console page, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name*** .
3. Under Web Services Properties, click **View Web services client deployment descriptor extension**.

The information in the following implementation indicates how to configure your application-level bindings. If the Web server is acting as a client, the default bindings are used. To configure the server-level bindings, which are the defaults, complete the following steps:
1. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

If you are using any of the following configurations, verify that the deployment descriptor is configured properly:
- Request signing
- Request encryption
- BasicAuth authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response digital signature verification
- Response decryption

Request signing

If the integrity constraints (digital signature) are specified, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web services security propertiess, click **Web Services: Client security bindings**.
4. In the Response receiver binding column, click **Edit** > **Signing information** > **New**.

To configure the key locators, complete the following steps:
1. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

3. Under Additional properties, click **Key locators**.

Request encryption

If the confidentiality constraints (encryption) are specified, verify that you configured the encryption information in the binding files.

To configure the encryption parameters, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Encryption Information > New**.

To configure the key locators, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Additional properties, click **Web Services: Default bindings for Web services security > Key locators**.

BasicAuth authentication

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains the CallBack support implementations:

com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler
This implementation prompts for basic authentication information, the user name and password, in an interface.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This implementation reads the basic authentication information from the binding file.

com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler
This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login binding information, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Identity (ID) Assertion authentication with BasicAuth TrustMode

Configure a login binding in the bindings file with a com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation. Specify a BasicAuth user name and password that a trusted ID evaluator on a downstream server trusts.

To configure the login binding information, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Identity (ID) Assertion authentication with the Signature TrustMode

Configure the signing information in the bindings file with a signing key pointing to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure ID assertion, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Additional properties, click **JAX-WS and JAX-RPC security runtime > Login mappings > IDAssertion**.

To configure the login binding information, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Client security bindings**.
4. Under Request sender bindings, click **Edit > Login binding**.

Response digital signature verification

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Signing information > New**.

To configure the trust anchors, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors > New**.

To configure the collection certificate store, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store > New**.

Response decryption

If the confidentiality constraints (encryption) are specified, verify that you defined the encryption information.

To configure the encryption information, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.

3. Under Web services security properties, click **Web services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Encryption information > New**.

To configure the key locators, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

View Web services server deployment descriptor: **Version 6 and later applications**

Use this page to view your server deployment descriptor settings.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Before you begin this task, the Web services application must be installed.

By completing this task, you can gather information that enables you to maintain or configure binding information. After the Web services application is installed, you can view the Web services deployment descriptors.

To view this administrative console page, complete the following steps:
1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Properties, click **View Web services server deployment descriptor**.

WebSphere Application Server has two levels of bindings: application-level and server-level. The information in the following implementation descriptions indicate how to configure your application-level bindings. To configure the server-level bindings, which are the defaults, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

- Request digital signature verification
- Request decryption
- Basic authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response signing
- Response encryption

Request digital signature verification

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit > Signing information**.

To configure the trust anchor, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.

To configure the collection certificate store, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.

To configure the key locators, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

Request decryption

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To configure the encryption information parameters, complete the following steps:
1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web services security properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit > Encryption information**.

To configure the key locators, complete the following steps:
1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

Basic authentication

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains callback support implementations:

com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler
The implementation prompts for BasicAuth information (user name and password) in an interface panel.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This implementation reads the BasicAuth information from the binding file.

com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler
This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login mapping information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

Identity (ID) assertion authentication with the BasicAuth TrustMode

Configure a login binding in the bindings file with a com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation. Specify a user name and password for basic authentication that a TrustedIDEvaluator on a downstream server trusts.

To configure the login mapping information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

Identity (ID) assertion authentication with the signature TrustMode

Configure the signing information in the bindings file with a signing key that points to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure the login mapping information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

The Java Authentication and Authorization Service (JAAS) uses `WSLogin` as the name of the login configuration. To configure JAAS, complete the following steps:

1. Click **Security** → **Global security**.
2. Under Authentication, click **Java Authentication and Authorization Service** → **Application logins**.

The value of the <TrustedIDEvaluatorRef> tag in the binding must match the value of the <TrustedIDEvaluator> name.

To configure the trusted ID evaluators, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.

Response signing

If the integrity constraints (digital signature) are defined, verify that you have the signing information configured in the binding files.

To specify the signing information, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Server security bindings**.
4. In the Request receiver binding column, click **Edit > Signing information**.

To configure the key locators, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

Response encryption

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To specify the encryption information, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web services security properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit > Encryption information**.

To configure the key locators, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

Securing messages using JAX-RPC at the request and response generators

You can secure messages with tokens and encryption to protect message integrity, authenticity, and confidentiality.

About this task

To secure messages, you can:

- Configure generator signing to protect message integrity
- Configure encryption to protect message confidentiality at the server level and at the application level
- Configure tokens to protect message authenticity at the server level and at the application level
- To configure generator signing to protect message integrity, see the steps outlined in "Configuring generator signing using JAX-RPC to protect message integrity."
- To configure encryption to protect message confidentiality at the application level, see the steps outlined in "Configuring encryption using JAX-RPC to protect message confidentiality at the application level" on page 379.
- To configure encryption to protect message confidentiality at the server level, see the steps outlined in "Configuring encryption using JAX-RPC to protect message confidentiality at the server or cell level" on page 392.
- To configure tokens to protect message authenticity at the application level, see the steps outlined in "Configuring token generators using JAX-RPC to protect message authenticity at the application level" on page 394.
- To configure tokens to protect message authenticity at the server level, see the steps outlined in "Configuring token generators using JAX-RPC to protect message authenticity at the server level" on page 412.

Results

By completing the steps in the previous tasks, you have secured messages using tokens and encryption to protect message integrity, authenticity, and confidentiality.

Configuring generator signing using JAX-RPC to protect message integrity:

You can configure the generator key and signing information at the server and application level to protect message integrity.

About this task

To protect message integrity, you can:

- Configure the signing information for the client-side request generator and the server-side response generator bindings at the server level and at the application level
- Configure the key information for the request generator (client side) and the response generator (server side) bindings on the server level and at the application level
- To configure the signing information for the client-side request generator and the server-side response generator bindings at the server , see the steps outlined in "Configuring the signing information using JAX-RPC for the generator binding on the server level" on page 346.
- To configure the signing information for the client-side request generator and the server-side response generator bindings at the application level, see the steps outlined in "Configuring the signing information using JAX-RPC for the generator binding on the application level" on page 349
- To configure the key information for the request generator (client side) and the response generator (server side) bindings at the application level, see the steps outlined in "Configuring the key information using JAX-RPC for the generator binding on the application level" on page 366.

- To configure the key information for the generator binding on the server level, see the steps outlined in "Configuring the key information for the generator binding using JAX-RPC on the server level" on page 364.

Results

By completing the steps in these tasks, you have configured generator signing to protect the integrity of messages.

Configuring the signing information using JAX-RPC for the generator binding on the server level:

Version 6 and later applications

You can configure the signing information for the client-side request generator and the server-side response generator bindings at the server level.

Before you begin

Note: For WebSphere Application Server version 6.x or earlier only, in the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the Signing information panel within the administrative console.

About this task

This task explains the steps that are needed for you to configure the signing information for the client-side request generator and the server-side response generator bindings at the server level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message that include the body, time stamp, and user name token if these bindings are not defined at the application level. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the signing information for the generator sections of the bindings files on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Signing information**.
3. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the signing configuration in the Signing information name field. For example, you might specify `gen_signinfo`.
4. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the default generator must match the algorithm that is specified for the default consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a SIGNATURE based on a symmetric key must have a value of http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#hmac-sha1.

5. Select a canonicalization method from the Canonicalization method field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured canonical XML and exclusive XML canonicalization algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
6. Select a key information signature type from the **Key information signature type** field. The key information signature type determines how to digitally sign the key. WebSphere Application server supports the following signature types:

   None Specifies that the <KeyInfo> element is not signed.

   Keyinfo
   Specifies that the entire <KeyInfo> element is signed.

   Keyinfochildelements
   Specifies that the child elements of the <KeyInfo> element are signed.

   The key information signature type for the generator must match the signature type for the consumer. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses `keyinfo`, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
7. Select a signing key information reference from the Signing key information field. This selection is a reference to the signing key that the Application Server uses to generate digital signatures. In the binding files, this information is specified within the <signingKeyInfo> tag. The key that is used for signing is specified by the key information element, which is defined at the same level as the signing information. For more information, see “Configuring the key information for the generator binding using JAX-RPC on the server level” on page 364.
8. Click **OK** to save the configuration.
9. Click the name of the new signing information configuration. This configuration is the one that you specified in the previous steps.
10. Specify the part reference, digest algorithm, and transform algorithm. The part reference specifies which parts of the message to digitally sign.
    a. Under Additional Properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference.
    b. Specify a unique part name for the message part that needs signing. This message part is specified on both the server side and the client side. You must specify an identical part name for both the server side and the client side. For example, you might specify `reqint` for both the generator and the consumer.

       Note: You do not need to specify a value for the Part reference in the default bindings like you specify on the application level because the part reference on the application level points to a particular part of the message that is signed. Because the default bindings for the server level is applicable to all of the services that are defined on a particular server, you cannot specify this value.

c. Select a digest method algorithm in the **Digest method algorithm** field. The digest method algorithm that is specified in the binding files within the <DigestMethod> element is used in the <SigningInfo> element.

   WebSphere Application Server supports the following algorithms:

   - http://www.w3.org/2000/09/xmldsig#sha1
   - http://www.w3.org/2001/04/xmlenc#sha256
   - http://www.w3.org/2001/04/xmlenc#sha512

d. Click **OK** and **Save** to save the configuration.

e. Click the name of the new part reference configuration. This configuration is the one that you specified in the previous steps.

f. Under Additional properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.

g. Select a transform algorithm from the menu. The transform algorithm is specified within the <Transform> element. This algorithm element specifies the transform algorithm for the digital signature. WebSphere Application Server supports the following algorithms:

   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/TR/1999/REC-xpath-19991116

     Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.

   - http://www.w3.org/2002/06/xmldsig-filter2
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - http://www.w3.org/2002/07/decrypt#XML
   - http://www.w3.org/2000/09/xmldsig#enveloped-signature

   The transform algorithm that you select for the generator must match the transform algorithm that you select for the consumer.

   Note: If both of the following conditions are true, WebSphere Application Server signs the referenced token:

   - You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
   - You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

11. Click **Apply**.
12. Click **Save** at the top of the panel to save your configuration.

Results

After completing these steps, you have configured the signing information for the generator on the server level.

What to do next

You must specify a similar signing information configuration for the consumer.

Configuring the signing information using JAX-RPC for the generator binding on the application level:

Version 6 and later applications

You can configure the signing information for the client-side request generator and the server-side response generator bindings at the application level.

Before you begin

Note: For WebSphere Application Server version 6.x or earlier only, in the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you must configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

About this task

This task explains the required steps to configure the signing information for the client-side request generator and the server-side response generator bindings at the application level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message including the body, time stamp, and user name token. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment. Complete the following steps to configure the signing information for the generator sections of the bindings files on the application level:

1. Locate the signing information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties, you can access the signing information for the request generator and the response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Required properties, click **Signing information**.
   e. Click **New** to create a signing information configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Signing information name field. For example, you might specify `gen_signinfo`.
2. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the generator, which is either the request generator or the response generator configuration, must match the algorithm that is specified for the consumer, which is either the request consumer or response consumer configuration. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1

   Note: Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP).

Any ds:SignatureMethod/@Algorithm element in a SIGNATURE based on a symmetric key must have a value of http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#hmac-sha1.

3. Select a canonicalization method from the **Canonicalization method** field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
4. Select a key information signature type from the Key information signature type field. WebSphere Application Server supports the following signature types:

   None Specifies that the <KeyInfo> element is not signed.

   Keyinfo
   Specifies that the entire <KeyInfo> element is signed.

   Keyinfochildelements
   Specifies that the child elements of the <KeyInfo> element are signed.

   The key information signature type for the generator must match the signature type for the consumer. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses `keyinfo`, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
5. Select a signing key information reference from the Signing key information field. This selection is a reference to the signing key that the Application Server uses to generate digital signatures.
6. Click **OK** and **Save** to save the configuration.
7. Click the name of the new signing information configuration. This configuration is the one that you specified in a previous step.
8. Specify the part reference, digest algorithm, and transform algorithm. The part reference specifies which parts of the message to digitally sign.
   a. Under Additional properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference.
   b. Specify a unique part name for this part reference. For example, you might specify `reqint`.
   c. Select a part reference from the Part reference field.

      The part reference refers to the message part that is digitally signed. The part attribute refers to the name of the <Integrity> element in the deployment descriptor when the <PartReference> element is specified for the signature. You can specify multiple <PartReference> elements within the <SigningInfo> element. The <PartReference> element has two child elements when it is specified for the signature: <DigestTransform> and <Transform>.
   d. Select a digest method algorithm from the menu. The digest method algorithm specified within the <DigestMethod> element is used in the <SigningInfo> element.

      WebSphere Application Server supports the following algorithms:
      - http://www.w3.org/2000/09/xmldsig#sha1
      - http://www.w3.org/2001/04/xmlenc#sha256
      - http://www.w3.org/2001/04/xmlenc#sha512
   e. Click **OK** to save the configuration.

f. Click the name of the new part reference configuration. This configuration is the one that you specified in a previous step.

g. Under Additional Properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.

h. Select a transform algorithm from the menu. The transform algorithm is that is specified within the <Transform> element and specifies the transform algorithm for the signature. WebSphere Application Server supports the following algorithms:

- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/TR/1999/REC-xpath-19991116

  Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.
- http://www.w3.org/2002/06/xmldsig-filter2
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
- http://www.w3.org/2002/07/decrypt#XML
- http://www.w3.org/2000/09/xmldsig#enveloped-signature

The transform algorithm that you select for the generator must match the transform algorithm that you select for the consumer.

Note: If both of the following conditions are true, WebSphere Application Server signs the referenced token:

- You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
- You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

9. Click **Apply**.
10. Click **Save** at the top of the panel to save your configuration.

Results

After completing these steps, the signing information is configured for the generator on the application level.

What to do next

You must specify a similar signing information configuration for the consumer.

Signing information collection:

Use this page to view a list of signing parameters. Signing information is used to sign and validate parts of a message including the body, time stamp, and user name token. You can also use these parameters for X.509 validation when the authentication method is IDAssertion and the ID type is X509Certificate in the server-level configuration. In such cases, you must fill in the certificate path fields only.

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application Servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information**.
4. Click **New** to create a signing parameter. Click **Delete** to delete a signing parameter.

To view this administrative console page on the application level for signing information, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. **Version 6 and later applications** Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. **Version 6 and later applications** Under Required properties, click **Signing information**.
5. **Version 5.x application** Under Additional properties, you can use this panel to configure the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
6. **Version 5.x application** Under Additional properties, click **Signing information**.
7. Click **New** to create a signing parameter. Click **Delete** to delete a signing parameter.

Signing information name: **Version 6 and later applications**

Specifies the unique name that is assigned to the signing configuration.

Signature method: **Version 5.x or 6 application**

Specifies the signature method algorithm that is chosen for the signing configuration.

Canonicalization method: **Version 6 and later applications**

Specifies the canonicalization method algorithm that is chosen for the signing configuration.

Signing information configuration settings:

Use this page to configure new signing parameters.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Signature Syntax and Specification: W3C Recommendation 12 Feb 2002*.

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information**.
4. Click **New** to create a signing parameter or click the name of an existing configuration to modify its settings.

To view this administrative console page on the application level for signing information, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. **Version 6 and later applications** Under Required properties, click **Signing information**.
5. **Version 5.x application** Under Additional properties, you can access the signing information for the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
6. **Version 5.x application** Under Additional properties, click **Signing information**.
7. Click **New** to create a signing parameter or click the name of an existing configuration to modify its settings.

Signing information name: **Version 6 and later applications**

Specifies the name that is assigned to the signing configuration.

Signature method: **Version 5.x or 6 application**

Specifies the algorithm Uniform Resource Identifiers (URI) of the signature method.

The following pre-configured algorithms are supported:

- **Version 5.x or 6 application** http://www.w3.org/2000/09/xmldsig#rsa-sha1
- **Version 5.x or 6 application** http://www.w3.org/2000/09/xmldsig#dsa-sha1

  Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a signature based on a symmetric key must have a value of http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#hmac-sha1.
- **Version 6 and later applications** http://www.w3.org/2000/09/xmldsig#hmac-sha1

For Version 6.0.*x* applications, you can specify additional signature methods on the Algorithm URI panel. To access the Algorithm URI panel, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings >** ***algorithm_factory_engine_class_name*** **> Algorithm URI > New**.

When you specify the Algorithm URI, you also must specify an algorithm type. To have the algorithm display as a selection in the Signature method field on the Signing information panel, you must select **Signature** as the algorithm type.

This field is available for Version 6.0.*x* applications and for the request receiver and response receiver bindings for Version 5.*x* applications.

Digest method: **Version 5.x application**

Specifies the algorithm URI of the digest method.

The http://www.w3.org/2000/09/xmldsig#sha1 algorithm is supported.

This field is available for the request receiver and response receiver bindings for Version 5.*x* applications.

Canonicalization method: **Version 5.x or 6 application**

Specifies the algorithm URI of the canonicalization method.

The following pre-configured algorithms are supported:
- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/2001/10/xml-exc-c14n#WithComments
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

This field is for Version 6.0.*x* applications and for the request receiver and response receiver bindings for Version 5.*x* applications.

Key information signature type: **Version 6 and later applications**

Specifies how to sign a KeyInfo element if `dsigkey` or `enckey` is specified for the signing part in the deployment descriptor.

This product supports the following keywords:

keyinfo (default)
Specifies that the entire KeyInfo element is signed.

keyinfochildelements
Specifies that the child elements of the KeyInfo element is signed.

If you do not specify a keyword, the application server uses the KeyInfo value, by default.

The Key information signature type field is available for the token consumer binding.

For Version 6.0.*x* applications, this field is also available for the default consumer, request consumer, and response consumer bindings.

Signing key information: **Version 6 and later applications**

Specifies a reference to the key information that the application server uses to generate the digital signature.

You can specify one signing key only for the default generator binding on the server level. However, you can specify multiple signing keys for the default consumer bindings. The signing keys for the default consumer bindings are specified using the Key Information references link under Additional properties on the Signing information panel.

On the application level, you can specify only one signing key for the request generator and the response generator. You can specify multiple signing keys for the request consumer and response generator. The signing keys for the request consumer and the response consumer are specified using the Key information references link under Additional properties.

You can specify a signing key configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default Generator Bindings, click **Key information**.

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default Consumer Bindings, click **Key information**.

Certificate path: **Version 5.x application**

Specifies the settings for the certificate path validation. When you select **Trust any**, this validation is skipped and all incoming certificates are trusted.

The certificate path options are available on the application level.

Trust anchor

The application server searches for trust anchor configurations on the application and server levels and lists the configurations in this menu.

Version 5.x application You can specify trust anchors as an additional property for the response receiver binding and the request receiver binding.

You can specify a trust anchor configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Trust anchors > New**.

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Trust anchors > New**.
Response receiver	Application level for Version 5.*x* applications	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Click **Web services: Client security bindings**. 4. Under the Response receiver binding, click **Edit**. 5. Under Additional properties, click **Trust anchors > New**.
Request receiver	Application level for Version 5.*x* applications	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Click **Manage modules >** ***URI_name***. 3. Click **Web services: Server security bindings**. 4. Under the Request receiver binding, click **Edit**. 5. Under Additional properties, click **Trust anchors > New**.

For an explanation of the fields on the trust anchor panel, see the help topic Trust anchor configuration settings.

Certificate store

The application server searches for certificate store configurations on the application and server levels and lists the configurations in this menu.

You can specify a certificate store configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Collection certificate store > New**.
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Collection certificate store > New**.
Response receiver	Application level for Version 5.*x* applications	1. Click **Applications → Application Types → WebSphere enterprise applications →** ***application_name***. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Click **Web services: Client security bindings**. 4. Under the Response receiver binding, click **Edit**. 5. Under Additional properties, click **Collection certificate store > New**.
Request receiver	Application level for Version 5.*x* applications	1. Click **Applications → Application Types → WebSphere enterprise applications →** ***application_name***. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Click **Web services: Server security bindings**. 4. Under the Request receiver binding, click **Edit**. 5. Under Additional properties, click **Collection certificate store > New**.

For an explanation of the fields on the collection certificate store panel, see the help topic Collection certificate store configuration settings.

Part reference collection:

Use this page to view the message part references for signature and encryption that are defined in the deployment descriptors.

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information > *signing_information_name***.
4. Under Additional properties, click **Part references**.

To view this administrative console page on the application level for signing information, complete the following steps. Part references are available through the administrative console using Version 6.x applications only.

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sending) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. Under Required properties, click **Signing information > *signing_information_name***.
5. Under Additional properties, click **Part references**.

Part name: **Version 6 and later applications**

Specifies the name that is assigned to the part reference configuration.

Part reference: **Version 6 and later applications**

Specifies the name of the signed part that is defined in the deployment descriptor.

The Part reference field is specified in the application binding configuration only.

Digest method algorithm: **Version 6 and later applications**

Specifies the algorithm Uniform Resource Identifier (URI) of the digest method that is used for the signed part that is specified by the part reference.

Part reference configuration settings:

Use this page to specify a reference to the message parts for signature and encryption that are defined in the deployment descriptors.

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information >** ***signing_information_name***.
4. Under Additional properties, click **Part references**.
5. Click **New** to create a part reference or click the name of an existing configuration to modify its settings.

To view this administrative console page on the application level for signing information, complete the following steps.

Note: Part references are available through the administrative console using Version 6.0.*x* applications only.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** .
2. Click **Manage modules >** ***URI_name*** .
3. Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sending) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. Under Required properties, click **Signing information >** ***signing_information_name***.
5. Under Additional properties, click **Part references**.
6. Click **New** to create a part reference or click the name of an existing configuration to modify its settings.

You must specify a part name and select a part reference before specifying additional properties. Before specifying the digest method properties that are accessible under Additional properties, specify a digest method algorithm on this panel. If you specify `none` and click **Digest method**, an error message is displayed.

Part name: **Version 6 and later applications**

Specifies the name that is assigned to the part reference configuration.

Part reference: **Version 6 and later applications**

Specifies the name of the <integrity> or <requiredIntegrity> element for the signed part of the message or it specifies the name of the <confidentiality> or <requiredConfidentiality> element for the encrypted part of the message in the deployment descriptor.

The part names that are defined in the deployment descriptor are listed as options in this field. This field is displayed for the binding configuration on the application level only.

Digest method algorithm: **Version 6 and later applications**

Specifies the algorithm Uniform Resource Identifier (URI) of the digest method that is used for the signed part that is specified by the part reference.

This product provides the following predefined algorithm URIs:

- `http://www.w3.org/2000/09/xmldsig#sha1`
- `http://www.w3.org/2001/04/xmlenc#sha256`
- `http://www.w3.org/2001/04/xmlenc#sha512`

If you want to specify a custom algorithm, you must configure the custom algorithm in the Algorithm URI panel before setting the digest method algorithm.

To access the Algorithm URI panel, complete the following steps for the server level:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings > *algorithm_factory_engine_class_name* > Algorithm URI > New**.

The specified algorithms are listed as options for this field.

When you specify the Algorithm URI, you also must specify an algorithm type. To have the algorithm display as a selection in the Digest method algorithm field on the Part reference panel, you must select **Digest value calculation (Message digest)** as the algorithm type.

Transforms collection:

Use this page to view the transform algorithm that is used for processing the Web services security message.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Applications → Application Types → WebSphere enterprise applications → *application_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information > *signing_information_name***.
4. Under Additional properties, click **Part references > *part_name***.
5. Under Additional properties, click **Transforms**.

Version 6 and later applications

To view this administrative console page for the application level, complete the following steps.

Note: This option is available for Version 6 and later applications only.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage Modules** > ***URI_name***.
3. Under Web Services Security Properties, you can access the transforms information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Signing information** > ***signing_information_name***.
5. Under Additional properties, click **Part references** > ***part_name*** > **Transforms**.

Transform name:

Version 6 and later applications

Specifies the name that is assigned to the transform algorithm.

Transform algorithm:

Version 6 and later applications

Specifies the algorithm Uniform Resource Identifier (URI) of the transform algorithm.

Transforms configuration settings:

Use this page to specify the transform algorithm that is used for processing the Web services security message.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > ***application_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Signing information** > ***signing_information_name***.
4. Under Additional properties, click **Part references** > ***part_name***.
5. Under Additional properties, click **Transforms**.
6. Click **New** to create a transform configuration or click the name of an existing configuration to modify its settings.

Version 6 and later applications To view this administrative console page for the application level, complete the following steps. This option is available for Version 6.*x* applications only.

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access the transforms information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Signing information > *signing_information_name***.
5. Under Additional properties, click **Part references > *part_name* > Transforms**.
6. Click **New** to create a transform configuration or click the name of an existing configuration to modify its settings.

You must specify a transform name and select a transform algorithm before specifying additional properties.

Transform name: **Version 6 and later applications**

Specifies the name that is assigned to the transform algorithm.

Transform algorithm: **Version 6 and later applications**

Specifies the algorithm Uniform Resource Identifier (URI) of the transform algorithm.

This product supports the following algorithms:

http://www.w3.org/2001/10/xml-exc-c14n#
This algorithm specifies the World Wide Web Consortium (W3C) Exclusive Canonicalization recommendation.

http://www.w3.org/TR/1999/REC-xpath-19991116
This algorithm specifies the W3C XML path language recommendation. If you specify this algorithm, you must specify the property name and value by clicking **Properties**, which is displayed under Additional properties. For example, you might specify the following information:

Property
`com.ibm.wsspi.wssecurity.dsig.XPathExpression`

Value `not(ancestor-or-self::*[namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature'])`

Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.

http://www.w3.org/2002/06/xmldsig-filter2

This algorithm specifies the XML-Signature XPath Filter Version 2.0 proposed recommendation.

When you use this algorithm, you must specify a set of properties. You can use multiple property sets for the XPath Filter Version 2. Therefore, it is recommended that your property names end with the number of the property set, which is denoted by an asterisk in the following examples:

- To specify an XPath expression for the XPath filter2, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Expression_*
  ```

- To specify a filter type for each XPath, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Filter_*
  ```

  Following this expression, you can have a value, `[intersect]`, `[subtract]`, or `[union]`.

- To specify the processing order for each XPath, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Order_*
  ```

  Following this expression, indicate the processing order of the XPath.

The following is a list of complete examples:

```
com.ibm.wsspi.wssecurity.dsig.XPath2Expression_2 = [XPath expression#1]
com.ibm.wsspi.wssecurity.dsig.XPath2Filter_1 = [intersect]
com.ibm.wsspi.wssecurity.dsig.XPath2Order_1 = [1]
com.ibm.wsspi.wssecurity.dsig.XPath2Expression_2 = [XPath expression#2]
com.ibm.wsspi.wssecurity.dsig.XPath2Filter_2 = [subtract]
com.ibm.wsspi.wssecurity.dsig.XPath2Order_2 = [2]
```

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform

This algorithm specifies the enhancements to SOAP messaging that provide message integrity and confidentiality.

http://www.w3.org/2002/07/decrypt#XML

This algorithm specifies the W3C decryption transform for XML Signature recommendation.

http://www.w3.org/2000/09/xmldsig#enveloped-signature

This algorithm specifies the W3C recommendation for XML digital signatures.

Configuring the key information for the generator binding using JAX-RPC on the server level:

Use the key information for the default generator to specify the key that is used by the signing or the encryption information configurations if these bindings are not defined at the application level.

About this task

The signing and encryption information configurations can share the same key information, which is why they are both defined on the same level. WebSphere Application Server provides default values for these bindings. However, an administrator must modify these values for a production environment.

Complete the following steps to configure the key information for the generator binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Key information**.

3. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key configuration in the Key information name field. For example, you might specify `sig_keyinfo`.
4. Select a key information type from the Key information type field. WebSphere Application Server supports the following types of key information:

   Key identifier
   This key information type is used when two parties agree on how to create a key identifier. For example, a field of X.509 certificates can be used for the key identifier according to the X.509 profile.

   Key name
   This key information type is used when the sender and receiver agree on the name of the key.

   Security token reference
   This key information type is typically used when an X.509 certificate is used for digital signature.

   Embedded token
   This key information type is used to embed a security token in an embedded element.

   X509 issuer name and issuer serial
   This key information type specifies an X.509 certificate with its issuer name and serial number.

   Select **Security token reference** if you are using an X.509 certificate for the digital signature. In these steps, it is assumed that **Security token reference** is selected for this field.

   Note: This key information type must match the key information type that is specified for the consumer.
5. Select a key locator reference from the Key locator reference menu. In these steps, assume that the key locator reference is called `sig_klocator`. The key locator reference is the name of the key locator that is used to generate the key for digital signature. You must configure a key locator before you can select it in this field. For more information on configuring the key locator, see “Configuring the key locator using JAX-RPC on the server level” on page 472.
6. Click **Get keys** to view a list of key name references. After you click **Get keys**, the key names that are defined in the <sig_klocator> element are shown in the key name reference menu. If you change the key locator reference, you must click **Get keys** again to display the list of key names that are associated with the new key locator.
7. Select a key name reference from the Key name reference menu. The key name reference specifies the name of the key that is used for generating the digital signature or for encryption. The Key name reference menu displays a list of key names that are defined for the selected key locator in the Key locator reference field. For example, select **signerkey**. It is assumed that signer key is a key name that is defined for the `sig_klocator` key locator.
8. Select a token reference from the Token reference field. The token reference refers to the name of a configured token generator. When a security token is required in the deployment descriptor, the token reference attribute is required. If you select **Security token reference** in the Key information type field, the token reference is required and you can specify an X.509 token generator. To specify an X.509 token generator, you must have an X.509 token generator configured. To configure an X.509 token generator, see “Configuring token generators using JAX-RPC to protect message authenticity at the server level” on page 412. For the remaining steps, it is assumed that an X.509 token generator that is named gen_tcon is already configured.
9. Optional: Select an encoding method from the Encoding method field This field specifies the encoding format for the key identifier. The encoding method attribute is valid when you select **Key identifer** as the key information type. WebSphere Application Server supports the following encoding methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

10. Optional: Select a calculation method from the Calculation method field. The calculation method specifies the calculation algorithm that is used for the key identifier. This attribute is valid when you select **Key identifier** as the key information type. WebSphere Application Server supports the following calculation methods:
    - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
    - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1
11. Optional: Specify a Uniform Resource Identifier (URI) of the value type for a security token from the Namespace URI field. The namespace URI is referenced by the key identifier. This attribute is valid when you select **Key identifier** as the key information type. When you specify the X.509 certificate token, you do not need to specify the namespace URI. If another token is specified, you must specify the namespace URI. For example, you can specify `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the Lightweight Third Party Authentication (LTPA) token and `http://www.ibm.com/websphere/appserver/tokentype` for the LTPA_PROPAGATION token.
12. Optional: Specify the local name of the value type for a security token in the **Local name** field. The local name is referenced by the key identifier. This attribute is valid when you select **Key identifier** as the key information type. WebSphere Application Server supports the following local names:

    For an X.509 certificate token
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

    For X.509 certificates in a PKIPath
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

    For a list of X.509 certificates and CRLs in a PKCS#7
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

    For LTPA
    LTPA

    For LTPA_PROPAGATION
    LTPA_PROPAGATION
13. Click **OK** and **Save** to save the configuration.

Results

You have configured the key information for the generator binding at the server level.

What to do next

You must specify a similar key information configuration for the consumer.

Configuring the key information using JAX-RPC for the generator binding on the application level:

The key information is used to specify the configuration needed to generate the key for digital signature and encryption. The signing information and the encryption information configurations can share the key information, so they are both defined at the same level.

Before you begin

Before you begin this task, configure the key locators and the token consumers that are referenced by the Key locator reference and Token reference fields within the key information panel.

About this task

This task provides the steps needed for configuring the key information for the request generator (client side) and the response generator (server side) bindings at the application level.

Complete the following information to configure the key information for the generator binding on the application level:

1. Locate the key information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Required properties, click **Key information**.
   e. Click **New** to create a key information configuration, select the box next to an existing configuration and click **Delete** to delete the configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Key information name field. For example, you might specify `gen_signkeyinfo`.
2. Select a key information type from the Key information type field. The key information type specifies how to reference the security tokens. WebSphere Application Server supports the following key information types:

   Key identifier

   The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the <KeyIdentifier> element value depends upon the token type. For example, a hash of the important elements of the security token is used for generating the <KeyIdentifier> element value. The following <KeyInfo> element is generated in the SOAP message for this key information type:

   ```
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference>
         <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...
         </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   ```

   Key name

   The security token is referenced using a name that matches an identity assertion within the token. It is recommended that you do not use this key type as it might result in multiple security tokens that match the specified name. The following <KeyInfo> element is generated in the SOAP message for this key information type:

   ```
   <ds:KeyInfo>
      <ds:KeyName>CN=Group1</ds:KeyName>
   </ds:KeyInfo>
   ```

   Security token reference

   The security token is directly referenced using Universal Resource Identifiers (URIs). The following <KeyInfo> element is generated in the SOAP message for this key information type:

   ```
   <ds:KeyInfo>
      <wsse:SecurityTokenReference>
         <wsse:Reference URI="#mytoken" />
      </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   ```

Embedded token

The security token is directly embedded within the <SecurityTokenReference> element. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
         ...
      </wsse:Embedded>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

X509 issuer name and issuer serial

The security token is referenced by an issuer name and an issuer serial number of an X.509 certificate. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
     <ds:X509Data>
        <ds:X509IssuerSerial>
           <ds:X509IssuerName>CN=Jones, O=IBM, C=US
           </ds:X509IssuerName>
           <ds:X509SerialNumber>1040152879
            </ds:X509SerialNumber>
        </ds:X509IssuerSerial>
     </ds:X509Data>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

Each type of key information is described in the Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS standard, which is located at: http://www.oasis-open.org/home/index.php under Web services security.

3. Select a key locator reference from the Key locator reference field. This reference specifies a key locator that WebSphere Application Server uses to locate the keys that are used for digital signature and encryption. Before you can select a key locator, you must have configured a key locator. For more information on configuring a key locator, see the following articles:
   - “Configuring the key locator using JAX-RPC for the generator binding on the application level” on page 463
   - “Configuring the key locator using JAX-RPC for the consumer binding on the application level” on page 470
4. Click **Get keys** to view a list of key name references. After you click **Get keys**, the key names that are defined in the <sig_klocator> element are shown in the key name reference menu. If you change the key locator reference, you must click **Get keys** again to display the list of key names associated with the new key locator.
5. Select a key name reference from the Key name reference field. This reference specifies the name of a key that is used for generating a digital signature and for encryption. The list of key names provided comes from the key locator specified with the key locator reference.
6. Select a token reference from the Token reference field. This token reference specifies the name of token generator that is used for processing the security token. However, WebSphere Application Server requires this field only when you select Security token reference or Embedded token in the Key information type field. Before specifying a token reference, you must configure a token generator. For more information on configuring a token generator, see “Configuring token generators using JAX-RPC to protect message authenticity at the application level” on page 394.
7. Optional: If you select Key identifier as the key information type on this panel, you must specify an encoding method, calculation method, value type namespace URI, and a value type local name.
   a. Select an encoding method from the Encoding method field. The encoding method specifies the encoding format for the key identifier. WebSphere Application Server supports the following encoding methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

b. Select a calculation method from the Calculation method field. WebSphere Application Server supports the following calculation methods:
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1

c. Specify a value type namespace Uniform Resource Identifier (URI) in the Namespace URI field. In this field, specify the namespace URI of the value type for a security token that is referenced by the key identifier. When you specify the X.509 certificate token, you do not need to specify this option. If you want to specify another token, you must specify the URI of the qualified name (QName) for value type.

d. Specify a value type local name. This name is the local name of the value type for a security token that is referenced by the key identifier. When this local name is used in conjunction with the corresponding namespace URI, the information is called the value type qualified name or QName.

   When you specify the X.509 certificate token, it is recommended that you use the predefined local names. When you specify the predefined local names, you do not need to specify the namespace URI of the value type. However, if you do not use one of the predefined local names, you must specify both the uniform resource identifier (URI) and the local name. WebSphere Application Server provides the following predefined local names:

   X.509 certificate token
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

   X.509 certificates in a PKIPath
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

   A list of X509 certificates and CRLs in a PKCS#7
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

   LTPA Lightweight Third-Party Authentication token. When you specify a value type local name of LTPA, you must also specify a namespace URI of http://www.ibm.com/websphere/appserver/tokentype/5.0.2.

   LTPA_PROPAGATION
   Lightweight Third-Party Authentication propagation token. When you specify a value type local name of LTPA_PROPAGATION, you must also specify a namespace URI of http://www.ibm.com/websphere/appserver/tokentype.

8. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the key information for the generator binding at the application level

What to do next

You must specify a similar key information configuration for the consumer.

Key information collection:

Use this page to view the configurations that are currently available for generating or consuming the key for XML digital signatures and XML encryption.

To view this administrative console page on the server level for the key information references, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings or the JAX-RPC Default Consumer Bindings, click **Key information**.

To view this administrative console page on the application level for the key information references, complete the following steps.

Note: This option is available on the application level for Version 6 and later applications.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Key information**.

Key information name: **Version 6 and later applications**

Specifies the name that is given for the key configuration.

Key information class name: **Version 6 and later applications**

Specifies the class name that is used for the key information type.

Key information type: **Version 6 and later applications**

Specifies the type of mechanism used to reference the security token. The type corresponds to the class name that is specified in the Key information class name field.

Key information configuration settings:

Use this page to specify the related configuration need to specify the key for XML digital signature or XML encryption.

To view this administrative console page on the server level for the key information references, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

3. Under JAX-RPC Default Generator Bindings or the JAX-RPC Default Consumer Bindings, click **Key information**.
4. Click **New** to create a new configuration or click the configuration name to modify its contents.

To view this administrative console page on the application level for the key information references, complete the following steps.

Note: This option is available on the application level for Version 6.0.*x* applications.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Key information**.
5. Click **New** to create a new configuration or click the configuration name to modify its contents.

Before clicking **Properties** under Additional properties, you must enter a value in the **Key information name** field and select an option for the Key information type and Key locator reference options.

Key information name: **Version 6 and later applications**

Specifies a name for the key information configuration.

Key information type: **Version 6 and later applications**

Specifies the type of key information. The key information type specifies how to reference security tokens.

This product supports the following types of key information. Each type of key information is described in Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

Type	Description
Key identifier	The security token is referenced using an opaque value that uniquely identifies the token.
Key name	The security token is referenced using a name that matches an identity assertion within the token.
Security token reference	With this type, the security token is directly referenced.
Embedded token	With this type, the security token reference is embedded.
X509 issuer name and issuer serial	With this type, the security token is referenced by an issuer and serial number of an X.509 certificate

The X.509 issuer name and issuer serial is described in Web Services Security: X.509 Certificate Token Profile Version 1.0. The other types are described in Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).

If you select **Key identifier** for the key information type, you can specify values in the following fields on this panel:

- Encoding method
- Calculation method
- Value type namespace URI
- Value type local name

Key locator reference: **Version 6 and later applications**

Specifies the reference that is used to retrieve the key for digital signature and encryption.

Before specifying a key locator reference, you must configure a key locator. You can specify a signing key configuration for the following bindings:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**. 4. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.

Binding name	Server level or application level	Path
Version 5.x application Request sender binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 5.x application Response receiver binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 5.x application Request receiver binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 5.x application Response sender binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.

Binding name	Server level or application level	Path
Version 6 and later applications Request generator (sender) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 6 and later applications Response consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 6 and later applications Request consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Version 6 and later applications Response generator (sender) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.

Key name reference: **Version 6 and later applications**

Specifies the name of the key that is used for generating digital signature and encryption.

This field is displayed for the default generator and is also displayed for the request generator and response generator.

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**. 4. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Request generator (sender) binding **Version 6 and later applications**	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules > *URI_name***. 3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.
Response generator (sender) binding **Version 6 and later applications**	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules > *URI_name***. 3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. 4. Under Additional properties, click **Key locators**. 5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration.

Token reference: **Version 6 and later applications**

Specifies the name of a token generator or token consumer that is used for processing a security token.

The application server requires this field only when you specify Security token reference or Embedded token in the **Key information type** field. The **Token reference** field is also required when you specify a key identifier type for the consumer. Before specifying a token reference, you must configure a token generator or token consumer. You can specify a token configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default Generator Bindings, click **Token generator**. 4. Click **New** to create a new token generator or click the name of a configured token generator to modify its configuration.
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default Consumer Bindings, click **Token consumer**. 4. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration.
Request generator (sender) binding **Version 6 and later applications**	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. 4. Under Additional properties, click **Token generators**. 5. Click **New** to create a new token generator or click the name of a configured token generator to modify its configuration.

Binding name	Server level or application level	Path
Version 6 and later applications Response consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. 4. Under Required properties, click **Token consumers**. 5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration.
Version 6 and later applications Request consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. 4. Under Required properties, click **Token consumers**. 5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration.
Version 6 and later applications Response generator (sender) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. 4. Under Additional properties, click **Token generators**. 5. Click **New** to create a new token generator or click the name of a configured token generator to modify its configuration.

Encoding method: **Version 6 and later applications**

Specifies the encoding method that indicates the encoding format for the key identifier.

This field is valid when you specify Key identifier in the Key information type field. This product supports the following encoding methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

This field is available for the default generator binding only.

Calculation method: **Version 6 and later applications**

This field is valid when you specify Key identifier in the **Key information type** field. This product supports the following calculation methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1

This field is available for the generator binding only.

Value type namespace URI: **Version 6 and later applications**

Specifies the namespace Uniform Resource Identifier (URI) of the value type for a security token that is referenced by the key identifier.

This field is valid when you specify Key identifier in the **Key information type** field. When you specify the X.509 certificate token, you do not need to specify this option. If you want to specify another token, specify the URI of `QName` for value type.

This product provides the following predefined value type URIs for the Lightweight Third Party Authentication (LTPA) token:

- http://www.ibm.com/websphere/appserver/tokentype
- http://www.ibm.com/websphere/appserver/tokentype/5.0.2

This field is available for the generator binding only.

Value type local name: **Version 6 and later applications**

Specifies the local name of the value type for a security token that is referenced by the key identifier.

When this local name is used with the corresponding namespace URI, the information is called the *value type qualified name* or *QName*.

This field is valid when you specify Key identifier in the **Key information type** field. When you specify the X.509 certificate token, it is recommended that you use the predefined local names. When you specify the predefined local names, you do not need to specify the URI of the value type. This product provides the following predefined local names:

X.509 certificate token
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

X.509 certificates in a PKIPath
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

A list of X509 certificates and CRLs in a PKCS#7
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

Note: For LTPA, the value type local name is `LTPA`. If you enter `LTPA` for the local name, you must specify the `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` URI value in the **Value type URI**

field as well. For LTPA token propagation, the value type local name is `LTPA_PROPAGATION`. If you enter `LTPA_PROPAGATION` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype URI value in the **Value type URI** field as well. For the other predefined value types (User name token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the **Value type local name** field begins with `http://`. For example, if you are specifying the user name token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the **Value type local name** field and then you do not need to enter a value in the value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

This field is also available for the generator binding only.

Configuring encryption using JAX-RPC to protect message confidentiality at the application level:

Version 6 and later applications

You can configure encryption information, used to specify how the generators (senders) encrypt outgoing messages, for the request generator (client side) and the response generator (server side) bindings at the application level.

Before you begin

Configure the key information that is referenced by the key information references in the encryption information panel.

About this task

This task provides the steps that are needed for configuring encryption information for the request generator (client side) and the response generator (server side) bindings at the application level. This encryption information is used to specify how the generators (senders) encrypt outgoing messages.

Complete the following steps to configure the encryption information for the request generator or response generator section of the bindings file on the application level:

1. Locate the encryption information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties, you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Required properties, click **Encryption information**.
   e. Click **New** to create an encryption information configuration. Click **Delete** to delete an existing configuration or click the name of an existing encryption information configuration to edit its settings. If you are creating a new configuration, enter a name in the **Encryption information name** field. For example, you might specify `gen_encinfo`.

2. Select a data encryption algorithm from the **Data encryption algorithm** field. The selection specifies the algorithm that is used to encrypt parts of the message. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

   The data encryption algorithm that you select for the generator side must match the data encryption method that you select for the consumer side.
3. Select a key encryption algorithm from the **Key encryption algorithm** field. This selection specifies the algorithm that is used to encrypt keys. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

     When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

     Note: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

     By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:
     - `http://www.w3.org/2001/04/xmlenc#sha256`
     - `http://www.w3.org/2001/04/xmlenc#sha512`

     By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

     Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

The key encryption algorithm that you select for the generator side must match the key encryption method that you select for the consumer side.

4. Select an encryption key information reference from the Encryption key information menu. This selection is a reference to the encryption key that is used to encrypt parts of the message. To configure the key information, see "Configuring the key information using JAX-RPC for the generator binding on the application level" on page 366.
5. Select a part reference from the **Part reference** field. This field specifies the name of the part reference for the generator binding element in the deployment descriptor.
6. Click **OK** and then click **Save** to save the configuration.

Results

The encryption information is configured for the generator binding at the application level.

What to do next

You must specify a similar encryption information configuration for the consumer.

Encryption information collection:

Use this page to specify the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message, including the body and user name token.

To view the administrative console panel for the encryption information on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under either JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Encryption information**.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access encryption information for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.

- For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
- For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.

4. **Version 5.x application** Under Additional properties, you can access encryption information for the following bindings:
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.

Encryption information name: **Version 5.x or 6 application**

Specifies the name of the encryption information.

Key locator reference: **Version 5.x application**

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

Key encryption algorithm: **Version 5.x or 6 application** Specifies the algorithm that is used to encrypt and decrypt keys.

Data encryption algorithm: **Version 5.x or 6 application** Specifies the algorithm that is used to encrypt and decrypt data.

Encryption information configuration settings: Message parts:

Use this page to configure the encryption and decryption parameters. You can use these parameters to encrypt and decrypt various parts of the message, including the body and the token.

To view the administrative console panel for the encryption information on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under either JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Encryption information**.
4. Click **New** to create a new encryption configuration or click the name of an existing encryption configuration.

To view this administrative console page for the encryption information on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.

2. Under Modules, click **Module update** > ***module_name***.
3. Under Web Services Security Properties, you can access encryption information for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
4. Click either **New** to create a new encryption configuration or click the name of an existing encryption configuration.

Encryption information name: **Version 5.x or 6 application**

Specifies the name for the encryption information.

Data type String

Data encryption algorithm: **Version 5.x or 6 application**

Specifies the algorithm Uniform Resource Identifier (URI) of the data encryption method.

The following algorithms are supported:
- http://www.w3.org/2001/04/xmlenc#tripledes-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc. To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html. For more information, see “Encryption information configuration settings: Methods” on page 389.
- http://www.w3.org/2001/04/xmlenc#aes192-cbc. To use this algorithm, you must download the unrestricted JCE policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html. For more information, see the help topic Encryption information configuration settings: Methods.

  Note: Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files. For more information, see the **Key encryption algorithm** field description.

Key locator reference: **Version 5.x application**

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

The Key locator reference field is displayed for the request receiver and response receiver bindings, which are used by Version 5.*x* applications.

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

You can specify an encryption key configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**.
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**.
Version 5.x application Request sender	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***. 2. Under Modules, click **Manage modules > *URI_name***. 3. Click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. 4. Under Additional properties, click **Key locators**.

Binding name	Server level or application level	Path
Version 5.x application Request receiver	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***. 2. Under Modules, click **Manage modules> *URI_name***. 3. Click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. 4. Under Additional properties, click **Key locators**.
Version 5.x application Response sender	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***. 2. Under Modules, click **Manage modules> *URI_name***. 3. Click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. 4. Under Additional properties, click **Key locators**.
Version 5.x application Response receiver	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***. 2. Under Modules, click **Manage modules> *URI_name***. 3. Click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. 4. Under Additional properties, click **Key locators**.

Key encryption algorithm: **Version 5.x or 6 application**

Specifies the algorithm Uniform Resource Identifier (URI) of the key encryption method.

The following algorithms are provided by the application server:

- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

  When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with Software Development Kit (SDK) Version 1.5 or later.

  By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

  – `http://www.w3.org/2001/04/xmlenc#sha256`

  – `http://www.w3.org/2001/04/xmlenc#sha512`

  By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption

algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

- http://www.w3.org/2001/04/xmlenc#rsa-1_5
- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes192

Note: Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

- http://www.w3.org/2001/04/xmlenc#kw-aes256

IBM Software Development Kit Version 1.4:

For the IBM i operating system and IBM Software Development Kit Version 1.4, the tuning of Web services security is not required. The unrestricted jurisdiction policy files for IBM Software Development Kit Version 1.4 are automatically configured when the prerequisite software is installed.

- For IBM i (formerly known as i5/OS® V5R3) and IBM Software Development Kit Version 1.4, install product 5722AC3, Crypto Access Provider 128-bit.
- For IBM i 5.4 and IBM Software Development Kit Version 1.4, install product 5722SS1 Option 3, Extended Base Directory Support.

IBM Software Development Kit Version 1.5:

For IBM i 5.4 and IBM i (formerly known as i5/OS V5R3) and IBM Software Development Kit 1.5, the restricted JCE jurisdiction policy files are configured, by default. You can download the unrestricted JCE jurisdiction policy files from the following Web site: IBM developer works: Security Information, Version 5

Note: If Java Platform, Standard Edition 6 (Java SE 6) 32-bit for IBM i is the enabled Java virtual machine (JVM) for your profile, substitute `/QOpenSys/QIBM/ProdData/JavaVM/jdk50/32bit/jre` for `/QIBM/ProdData/Java400/jdk15` as the path name in the following steps.

To configure the unrestricted jurisdiction policy files for IBM i and the IBM Software Development Kit Version 1.5:

1. Make backup copies of these files:

   ```
   /QIBM/ProdData/Java400/jdk15/lib/security/local_policy.jar
   /QIBM/ProdData/Java400/jdk15/lib/security/US_export_policy.jar
   ```

2. Download the unrestricted policy files from IBM developer works: Security Information to the `/QIBM/ProdData/Java400/jdk15/lib/security` directory.
   a. Go to this Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
   b. Click **J2SE 5.0**.
   c. Scroll down and click **IBM SDK Policy files**. The Unrestricted JCE Policy files for the SDK Web site is displayed.
   d. Click **Sign in** and provide your IBM intranet ID and password.
   e. Select the appropriate unrestricted JCE policy files, and then click **Continue**.
   f. View the license agreement, and then click **I Agree**.
   g. Click **Download Now**.

3. Use the DSPAUT command to ensure *PUBLIC is granted*RX data authority but also ensure that no object authority is provided to both the `local_policy.jar` and the `US_export_policy.jar` files in the `/QIBM/ProdData/Java400/jdk15/lib/security` directory. For example:

   ```
   DSPAUT OBJ('/qibm/proddata/java400/jdk15/lib/security/local_policy.jar')
   ```

4. Use the CHGAUT command to change authorization, if needed. For example:

   ```
   CHGAUT OBJ('/qibm/proddata/java400/jdk15/lib/security/local_policy.jar')
   USER(*PUBLIC) DTAAUT(*RX) OBJAUT(*NONE)
   ```

Custom algorithms on the server level

To specify custom algorithms on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.
4. Click **New** to specify a new algorithm mapping or click the name of an existing configuration to modify its settings.
5. Under Additional properties, click **Algorithm URI**.
6. Click **New** to create a new algorithm URI. You must specify **Key encryption** in the **Algorithm type** field to have the configuration display in the **Key encryption algorithm** field on the Encryption information configuration settings panel.

Encryption key information: **Version 5.x or 6 application**

Specifies the name of the key information reference that is used for encryption. This reference is resolved to the actual key by the specified key locator and defined in the key information.

Version 6 and later applications You must specify either one or no encryption key configurations for the request generator and response generator bindings.

Version 6 and later applications For the response consumer and the request consumer bindings, you can configure multiple encryption key references. To create a new encryption key reference, under Additional properties, click **Key information references**.

You can specify an encryption key configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default generator binding, click **Key information**.
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers >** *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default consumer binding, click **Key information**.
Version 5.x application Request generator (sender) binding	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Under Web Services Security Properties, click **Web services: Client security bindings**. 4. Under Request generator (sender) binding, click **Edit custom**. 5. Under Required properties, click **Key information**.
Version 5.x application Response generator (sender) binding	Application level	1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*. 2. Under Modules, click **Manage modules >** ***URI_name***. 3. Under Web Services Security Properties, click **Web services: Server security bindings**. 4. Under Response generator (sender) binding, click **Edit custom**. 5. Under Required properties, click **Key information**.

Part Reference: Version 6 and later applications

Specifies the name of the <confidentiality> element for the generator binding or the <requiredConfidentiality> element for the consumer binding element in the deployment descriptor.

This field is available on the application level only.

Encryption information configuration settings: Methods:

Use this page to configure the encryption and decryption parameters for the signature method, digest method, and canonicalization method.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Encryption Syntax and Processing: W3C Recommendation 10 Dec 2002*.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name* and complete one of the following steps:
   - Click **Manage modules > *URI_file_name* > Web Services: Client Security Bindings**. Under Request sender binding, click **Edit**. Under Web Service Security Properties, click **Encryption Information**.
   - Under Modules, click **Manage modules > *URI_file_name* > Web Services: Server Security Bindings**. Under Response sender binding, click **Edit**. Under Web Service Security Properties, click **Encryption Information**.
2. Select **None** or **Dedicated encryption information**. The application server can have either one or no encryption configurations for the request sender and the response sender bindings. If you are not using encryption, select **None**. To configure encryption for either of these two bindings, select **Dedicated encryption information** and specify the configuration settings using the fields that are described in this topic.

Encryption information name: **Version 5.x application**

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

Key locator reference: **Version 5.x application**

Specifies the name that is used to reference the key locator.

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

To configure the key locators on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

To configure the key locators on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.

2. Under Modules, click **Manage modules** > ***URI_name***.
3. Under Web Service Security Properties, you can access the key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. Under Additional properties, click **Key locators**.

Encryption key name: **Version 5.x application**

Specifies the name of the encryption key that is resolved to the actual key by the specified key locator.

Data type String

Key encryption algorithm: **Version 5.x application**

Specifies the algorithm uniform resource identifier (URI) of the key encryption method.

The following algorithms are supported:
- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

  When running with IBM Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with JDK 1.5 or later.

  By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod. The property value is one of the following URIs of the digest method:
  - `http://www.w3.org/2001/04/xmlenc#sha256`
  - `http://www.w3.org/2001/04/xmlenc#sha512`

  By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

  Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.
- http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- http://www.w3.org/2001/04/xmlenc#kw-tripledes.
- http://www.w3.org/2001/04/xmlenc#kw-aes128.
- http://www.w3.org/2001/04/xmlenc#kw-aes192. To use the 192-bit key encryption algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file.

  Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
- http://www.w3.org/2001/04/xmlenc#kw-aes256. To use the 256-bit key encryption algorithm, you must download the unrestricted JCE policy file.

Note: If an InvalidKeyException error occurs and you are using the 129xxx or 256xxx encryption algorithm, the unrestricted policy files might not exist in your configuration.

Java Cryptography Extension

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256-bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files.

Note: Before downloading these policy files, back up the existing policy files (`local_policy.jar` and `US_export_policy.jar` in the *WAS_HOME*`/jre/lib/security/` directory) prior to overwriting them in case you want to restore the original files later.

To download the policy files, complete one of the following sets of steps:

After following either of these sets of steps, two Java archive (JAR) files are placed in the Java virtual machine (JVM) `jre/lib/security/` directory.

IBM i operating system and IBM Software Development Kit 1.4

For the IBM i operating system and IBM Software Development Kit Version 1.4, the tuning of Web services security is not required. The unrestricted jurisdiction policy files for the IBM Software Development Kit Version 1.4 are automatically configured when the prerequisite software is installed.

For the IBM i 5.4 operating system and IBM Software Development Kit Version 1.4, the unrestricted jurisdiction policy files for the IBM Java Developer Kit 1.4 are automatically configured by installing product 5722SS1 Option 3, Extended Base Directory Support.

For the IBM i operating system (formerly known as i5/OS V5R3) and IBM Software Development Kit Version 1.4, the unrestricted jurisdiction policy files for the IBM Software Development Kit Version 1.4 are automatically configured by installing product 5722AC3, Crypto Access Provider 128-bit.

IBM i operating system and IBM Software Development Kit 1.5

For IBM i 5.4 and IBM i (formerly known as i5/OS V5R3) and IBM Software Development Kit 1.5, the restricted JCE jurisdiction policy files are configured, by default. You can download the unrestricted JCE jurisdiction policy files from the following Web site: Security information: IBM J2SE 5 SDKs

To configure the unrestricted jurisdiction policy files for the IBM i operating system and the IBM Software Development Kit Version 1.5:

1. Make backup copies of these files:
   ```
   /QIBM/ProdData/Java400/jdk15/lib/security/local_policy.jar
   /QIBM/ProdData/Java400/jdk15/lib/security/US_export_policy.jar
   ```
2. Download the unrestricted policy files from IBM developer kit: Security information to the `/QIBM/ProdData/Java400/jdk15/lib/security` directory.
   a. Go to this Web site: IBM developer kit: Security information
   b. Click **J2SE 5.0**.
   c. Scroll down and click **IBM SDK Policy files**. The Unrestricted JCE Policy files for the SDK Web site is displayed.
   d. Click **Sign in** and provide your IBM intranet ID and password.
   e. Select the appropriate unrestricted JCE policy files, and then click **Continue**.
   f. View the license agreement, and then click **I Agree**.
   g. Click **Download Now**.

3. Use the DSPAUT command to ensure *PUBLIC is granted*RX data authority but also ensure that no object authority is provided to both the `local_policy.jar` and the `US_export_policy.jar` files in the `/QIBM/ProdData/Java400/jdk15/lib/security` directory. For example:

   ```
   DSPAUT OBJ('/qibm/proddata/java400/jdk15/lib/security/local_policy.jar')
   ```

4. Use the CHGAUT command to change authorization, if needed. For example:

   ```
   CHGAUT OBJ('/qibm/proddata/java400/jdk15/lib/security/local_policy.jar')
   USER(*PUBLIC) DTAAUT(*RX) OBJAUT(*NONE)
   ```

Data encryption algorithm: **Version 5.x application**

Specifies the algorithm Uniform Resource Identifiers (URI) of the data encryption method.

The following algorithms are supported:

- http://www.w3.org/2001/04/xmlenc#tripledes-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes192-cbc

  Note: Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).
- http://www.w3.org/2001/04/xmlenc#aes256-cbc

By default, the JCE ships with restricted or limited strength ciphers. To use 192-bit and 256- bit AES encryption algorithms, you must apply unlimited jurisdiction policy files. For more information, see the Key encryption algorithm field description.

Configuring encryption using JAX-RPC to protect message confidentiality at the server or cell level: **Version 6 and later applications**

You can configure the encryption information for the generator binding on the server or cell level.

About this task

The encryption information for the default generator specifies how to encrypt the information on the sender side if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the encryption information for the generator binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Encryption information**.
3. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the encryption configuration in the Encryption information name field. For example, you might specify `gen_encinfo`.

4. Select a data encryption algorithm from the Data encryption algorithm field. This algorithm is used to encrypt the data. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Note: Do not use this algorithm, the 192-bit key encryption algorithm, if you want your configured application to be in compliance with the Basic Security Profile (BSP).

   The data encryption algorithm that you select for the generator side must match the data encryption algorithm that you select for the consumer side.
5. Select a key encryption algorithm from the Key encryption algorithm field. This algorithm is used to encrypt the key. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

     When running with JDK 1.4, the list of supported key transport algorithms will not include this one. This algorithm will appear in the list of supported key transport algorithms when running with JDK 1.5.

     Note: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

     By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

     – `http://www.w3.org/2001/04/xmlenc#sha256`

     – `http://www.w3.org/2001/04/xmlenc#sha512`

     By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

     Note: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

Note: Do not use this algorithm, the 192-bit key encryption algorithm, if you want your configured application to be in compliance with the Basic Security Profile (BSP).

If you select **None**, the key is not encrypted.

The key encryption algorithm that you select for the generator side must match the key encryption algorithm that you select for the consumer side.

6. Select a encryption key configuration from the Encryption key information field. This attribute specifies the name of the key that is used to encrypt the message. To configure the key information, see "Configuring the key information for the generator binding using JAX-RPC on the server level" on page 364.
7. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the encryption information for the generator binding at the server or cell level.

What to do next

You must specify a similar encryption information configuration for the consumer.

Configuring token generators using JAX-RPC to protect message authenticity at the application level: **Version 6 and later applications**

When you specify the token generators at the application level, the information is used on the generator side to generate the security token.

About this task

Complete the following steps to configure the token generator on the application level:

1. Locate the token generator panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the token generators for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Token generators**.
   e. Click **New** to create a token generator configuration, select an existing configuration. Click **Delete** to delete an existing configuration, or click the name of an existing token generator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Token generator name** field. For example, you might specify `gen_signtgen`.
2. Specify a class name in the **Token generator class name** field. The token generator class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. The token generator class name for the request generator and the response generator must be similar to the token consumer class name for the request consumer and the response consumer. For example, if your application requires a username token consumer, you can specify the

com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer class name on the token consumer panel for the application level and the com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator class name in this field.

3. Optional: Select a part reference in the **Part reference** field. The part reference indicates the name of the security token that is defined in the deployment descriptor.

   Note: On the application level, if you do not specify a security token in your deployment descriptor, the **Part reference** field is not displayed. If you define a security token called `user_tgen` in your deployment descriptor, `user_tgen` is displayed as an option in the **Part reference** field. You can specify a security token in the deployment descriptor when you assemble your application using an assembly tool.

4. Select either **None** or **Dedicated signing information** for the certificate path. Select **None** when the token generator does not use the PKCS#7 token type. When the token generator uses the PKCS#7 token type and you want to package certificate revocation lists (CRLs) in the security token, select **Dedicated signing information** and select a certificate store. To configure a collection certificate store and certificate revocation lists for the generator bindings on the application level, complete the following steps:
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Related Items, click **EJB Modules** or **Web Modules > *URI_name***.
   c. Under Additional Properties you can access the collection certificate store configuration for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.

   d. Under Additional properties, click **Collection certificate store**.

   also see the information about configuring a collection certificate store.

5. Optional: Select the **Add nonce** option. This option indicates whether a nonce is included in the user name token for the token generator. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Add nonce** option is valid only when the generated token type is a user name token and is available only for the request generator binding.

   If you select the **Add nonce** option, you can specify the following properties under Additional properties. These properties are used by the request consumer.

Table 63. Additional nonce properties

Property name	Default value	Explanation
`com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.cacheTimeout`	600 seconds	Specifies the timeout value, in seconds, for the nonce value that is cached on the server.
`com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.clockSkew`	0 seconds	Specifies the time, in seconds, before the nonce time stamp expires.
`com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.maxAge`	300 seconds	Specifies the clock skew value, in seconds, to consider when WebSphere Application Server checks the timeliness of the message.

On the server level, you can specify these additional properties for a nonce on the Default bindings for Web services security panel within the administrative console. To access the panel, click **Servers > Server Types > WebSphere application servers > *server_name***. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

6. Optional: Select the **Add timestamp** option. This option indicates whether to insert a time stamp into the user name token. The **Add timestamp** option is valid only when the generated token type is a user name token and is available only for the request generator binding.
7. Specify the value type local name in the **Local name** field. For a user name token and an X.509 certificate security token, WebSphere Application Server provides predefined local names for the value type. When you specify any of the following local names, you do not need to specify a value type URI:

   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken
   This local name specifies a user name token.

   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509
   This local name specifies an X.509 certificate token.

   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1
   This local name specifies X.509 certificates in a public key infrastructure (PKI) path.

   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7
   This local name specifies a list of X.509 certificates and certificate revocation lists in a PKCS#7 format.

   For an LTPA token, you can use `LTPA` for the value type local name and `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the value type Uniform Resource Identifier (URI). For LTPA token propagation, you can use `LTPA_PROPAGATION` for the value type local name and `http://www.ibm.com/websphere/appserver/tokentype` for the value type URI.
8. Optional: Specify the value type URI in the **URI** field. This entry specifies the namespace URI of the value type for the generated token.
9. Click **OK** and **Save** to save the configuration.
10. Click the name of your token generator configuration.
11. Under Additional properties, click **Callback handler**.
12. Specify the settings for the callback handler.
    a. Specify a class name in the **Callback handler class name** field. This class name is the name of the callback handler implementation class that is used to plug-in a security token framework. The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler interface and must provide a constructor using the following syntax:

       `MyCallbackHandler(String username, char[] password, java.util.Map properties)`

       Where:

       username
       Specifies the user name that is passed into the configuration.

       password
       Specifies the password that is passed into the configuration.

       properties
       Specifies the other configuration properties that are passed into the configuration.

       This constructor is required if the callback handler needs a user name and a password. However, if the callback handler does not need a user name and a password, such as X509CallbackHandler, use a constructor with the following syntax:

       `MyCallbackHandler(java.util.Map properties)`

       WebSphere Application Server provides the following default callback handler implementations:

       com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
       This callback handler uses a login prompt to gather the user name and password

information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This callback handler does not issue a prompt and returns the user name and password if it is specified on this panel. You can use this callback handler when the Web service is acting as a client. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified on this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a Java EE application client. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore and a key definition is required for this callback handler. If you use this implementation, you must provide a key store password, path, and type on this panel.

com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You can specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. If you use this implementation, you must provide a key store password, path, and type on this panel.

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used. If you use this implementation, you must provide a key store password, path, and type on this panel.

The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is a plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

b. Optional: Select the **Use identity assertion** option. Select this option if you have identity assertion defined in the IBM extended deployment descriptor. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server sends only the user name of the original caller for a username token generator. For an X.509 token generator, the application server sends the original signer certification only.

c. Optional: Select the **Use RunAs identity** option. Select this option if you have identity assertion defined in the IBM extended deployment descriptor and you want to use the Run As identity instead of the initial caller identity for identity assertion in a downstream call. This option is valid only if you have configured Username TokenGenerator as a token generator.

d. Optional: Specify the basic authentication user ID in the **Basic authentication user ID** field. This entry specifies the user name that is passed to the constructors of the callback handler implementation. The basic authentication user name and password are used if you specified one of the following default callback handler implementations in the **Callback handler class name** field:

   - com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler

e. Optional: Specify the basic authentication password in the **Basic authentication password** field. This entry specifies the password that is passed to the constructors of the callback handler implementation.

f. Optional: Specify the key store password in the **Key store password** field. This entry specifies the password used to access the key store file. The key store and its configuration are used if you select on of the following default callback handler implementations that are provided by WebSphere Application Server:

   com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
   The keystore is used to build the X.509 certificate with the certificate path.

   com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
   The keystore is used to build the X.509 certificate with the certificate path.

   com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
   The keystore is used to retrieve the X.509 certificate.

g. Optional: Specify the key store path in the **Path** field. It is recommended that you use the `${USER_INSTALL_ROOT}` in the path name as this variable expands to the WebSphere Application Server path on your machine. To change the path used by this variable, click **Environment > WebSphere variables**, and click **USER_INSTALL_ROOT**. This field is required when you use the com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler, com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler, or com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler callback handler implementations.

h. Optional: Select the key store type in the **Type** field. This selection indicates the format used by the keystore file. You can select one of the following values for this field:

   JKS Use this option if the keystore uses the Java Keystore (JKS) format.

   JCEKS
   Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption.

   JCERACFKS
   Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

PKCS11KS (PKCS11)
Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

PKCS12KS (PKCS12)
Use this option if your keystore uses the PKCS#12 file format.

13. Click **OK** and then click **Save** to save the configuration.
14. Click the name of your token generator configuration.
15. Under Additional properties, click **Callback handler > Keys**.
16. Specify the key name, key alias, and the key password.
    a. Click **New** to create a key configuration, click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Key name** field. For digital signatures, the key name is used by the request generator or response generator signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption. The key name must be a fully qualified, distinguished name. For example, `CN=Bob,O=IBM,C=US`.
    b. Specify the key alias in the **Key alias** field. The key alias is used by the key locator to find the key within the keystore file.
    c. Specify the key password in the **Key password** field. This password is needed to access the key object within the keystore file.
17. Click **OK** and **Save** to save the configuration.

Results

You have configured the token generator for the application level.

What to do next

You must specify a similar token consumer configuration for the application level.

Request generator (sender) binding configuration settings:

Use this page to specify the binding configuration for the request generator.

To view this administrative console page, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** .
2. Under Modules, click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security Properties, click **Web services: Client security bindings**.
5. Under Request generator (sender) binding, click **Edit custom**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

The security constraints or bindings are defined using the application assembly process before the application is installed.

This product provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the Use defaults option on this panel and use the default binding information for the server level. The default binding provided by this product is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

Digital signature security constraint (integrity)

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

Information type	Required or optional
Signing information	Required
Key information	Required
Key locators	Optional
Collection certificate store	Optional
Token generator	Optional
Properties	Optional

You can use the key locators and the collection certificate store that are defined at the server-level.

Encryption constraint (confidentiality)

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

Information type	Required or optional
Encryption information	Required
Key information	Required
Key locators	Optional
Collection certificate store	Optional
Token generator	Optional
Properties	Optional

You can use the key locators and the collection certificate store that are defined at the server-level.

Security token constraint

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

Information type	Required or optional
Token generator	Required
Collection certificate store	Optional
Properties	Optional

You can use the collection certificate store that is defined at the server-level.

Use defaults: **Version 6 and later applications**

Select this option if you want to use the default binding information from the server-level.

Component: **Version 6 and later applications**

Specifies the enterprise bean in an assembled EJB module.

Port: **Version 6 and later applications**

Specifies the port in the Web service that is defined during application assembly.

Web service: **Version 6 and later applications**

Specifies the name of the Web service that is defined during application assembly.

Response generator (sender) binding configuration settings:

Use this page to specify the binding configuration for the response generator or response sender.

To view this administrative console page, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security Properties, click **Web services: Server security bindings**.
5. Under Response generator (sender) binding, click **Edit custom**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

The security constraints or bindings are defined using the application assembly process before the application is installed.

This product provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the **Use defaults** option on this panel and use the default binding information for the server-level. The default binding that is provided by this product is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

Digital signature security constraint (integrity)

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

Information type	Required or optional
Signing information	Required
Key information	Required
Key locators	Optional
Collection certificate store	Optional

Information type	Required or optional
Token generator	Optional
Properties	Optional

You can use the key locators and the collection certificate store that are defined at the server-level.

Encryption constraint (confidentiality)

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

Information type	Required or optional
Encryption information	Required
Key information	Required
Key locators	Optional
Collection certificate store	Optional
Token generator	Optional
Properties	Optional

You can use the key locators and the collection certificate store that are defined at the server-level.

Security token constraint

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

Information type	Required or optional
Token generator	Required
Collection certificate store	Optional
Properties	Optional

You can use the collection certificate store that is defined at the server-level.

Use defaults: **Version 6 and later applications**

Select this option if you want to use the default binding information from the server level.

If you select this option, the application server checks for binding information on the server level.

Port: **Version 6 and later applications**

Specifies the port number in the Web service that is defined during application assembly.

Web service: **Version 6 and later applications**

Specifies the name of the Web service that is defined during application assembly.

Callback handler configuration settings:

Use this page to specify how to acquire the security token that is inserted in the Web services security header within the SOAP message. The token acquisition is a pluggable framework that leverages the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface for acquiring the security token.

To view this administrative console page for the callback handler on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name*** .
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default generator bindings, click **Token generators > *token_generator_name***.
4. Under Additional properties, click **Callback handler**.

To view this administrative console page for the callback handler on the application level , complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage Modules > *URI_name***.
3. Under Web services security properties, you can access the callback handler information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Additional properties, click **Token generator**. Click **New** to create a new token generator configuration or click the name of an existing configuration to modify its settings. Under Additional properties, click **Callback handler**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Additional properties, click **Token generator**. Click **New** to create a new token generator configuration or click the name of an existing configuration to modify its settings. Under Additional properties, click **Callback handler**.

Callback handler class name: **Version 6 and later applications**

Specifies the name of the callback handler implementation class that is used to plug in a security token framework.

The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler class. The implementation of the JAAS javax.security.auth.callback.CallbackHandler interface must provide a constructor using the following syntax:

```
MyCallbackHandler(String username, char[] password,
    java.util.Map properties)
```

Where:

username
: Specifies the user name that is passed into the configuration.

password
: Specifies the password that is passed into the configuration.

properties
: Specifies the other configuration properties that are passed into the configuration.

The application server provides the following default callback handler implementations:

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**

This callback handler uses a login prompt to gather user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and the application server returns the user name and password to the token generator if it is specified on this panel. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**

This callback handler does not issue a prompt and returns the user name and password if it is specified on this panel. You can use this callback handler when the Web service is acting as a client.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**

This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, the application server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**

This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, the application server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**

This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the RunAs invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified on this panel, the application server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the RunAs Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a Java EE application client.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**

This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore and a key definition is required for this callback handler.

Version 6 and later applications **com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**

This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You must specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format.

Version 6 and later applications

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler

This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used.

The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is the plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the com.ibm.websphere.wssecurity.wssapi.token.SecurityToken interface. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side, respectively.

Use identity assertion: Version 6 and later applications

Select this option if you have identity assertion defined in the IBM extended deployment descriptor.

This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, the application server sends only the user name of the original caller for a Username TokenGenerator. For an X.509 token generator, the application server sends the original signer certification only.

Use RunAs identity: Version 6 and later applications

Select this option if you have identity assertion defined in the IBM extended deployment descriptor and you want to use the Run As identity instead of the initial caller identity for identity assertion for a downstream call.

This option is valid only if you have Username TokenGenerator configured as a token generator.

Basic authentication user ID: Version 6 and later applications

Specifies the user name that is passed to the constructors of the callback handler implementation.

The basic authentication user name and password are used if you select one of the following default callback handler implementations provided by this product:

- com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler

These implementations are described in detail under the **Callback handler class name** field description in this article.

Basic authentication password: Version 6 and later applications

Specifies the password that is passed to the constructor of the callback handler.

The keystore and its related configuration are used if you select one of the following default callback handler implementations provided by this product:

com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
The keystore is used to build the X.509 certificate with the certificate path.

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
The keystore is used to build the X.509 certificate with the certificate path.

com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
The keystore is used to retrieve the X.509 certificate.

Key store configuration name: **Version 6 and later applications**

Specifies the name of the key store configuration defined in the keystore settings in secure communications.

Key store password: **Version 6 and later applications**

Specifies the password that is used to access the keystore file.

Key store path: **Version 6 and later applications**

Specifies the location of the keystore file.

Use ${*USER_INSTALL_ROOT*} in the path name because this variable expands to the product path on your machine. To change the path used by this variable, click **Environment > WebSphere variables** and click **USER_INSTALL_ROOT**.

Key store type: **Version 6 and later applications**

Specifies the type of keystore file format

Choose one of the following values for this field:

JKS Use this option if the keystore uses the Java Keystore (JKS) format.

JCEKS
Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in the application server. This option provides stronger protection for stored private keys by using Triple DES encryption.

PKCS11KS (PKCS11)
Use this option if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

PKCS12KS (PKCS12)
Use this option if your keystore file uses the PKCS#12 file format.

Key collection:

Use this page to view a list of logical names that is mapped to a key alias in the keystore file.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name*** .
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default generator bindings, click **Token Generators > *token_generator_name*** .
4. Under Additional properties, click **Callback handler > Keys**.

Keys are also available by clicking **Key locators > *key_locator_name***. Under Additional properties, click **Keys**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
4. **Version 5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.

Key name: **Version 5.x or 6 application**

Specifies the name of the key object that is found in the keystore file.

Key alias: **Version 5.x or 6 application**

Specifies an alias for the key object.

The alias is used when the key locator searches for the key objects in the keystore file.

Key configuration settings:

Use this page to define the mapping of a logical name to a key alias in a keystore file.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default generator bindings, click **Token Generators > *token_generator_name***.
4. Under Additional properties, click **Callback handler > Keys**.
5. Specify a new key configuration by clicking **New** or by clicking the key configuration name to modify the settings.

Keys are also available by clicking **Key locators > *key_locator_name***. Under Additional properties, click **Keys > New**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. **Version 6 and later applications** Under Additional properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
4. Under Web Services Security Properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
5. Specify a new key configuration by clicking **New** or by clicking the key configuration name to modify the settings.

Key name: **Version 5.x or 6 application**

Specifies the name of the key object. For digital signatures, the key name is used by the request sender or request generator signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption.

The key name must be a fully qualified, distinguished name. For example, `CN:Bob,O=IBM,C=US`.

Note: If you enter the distinguished name with spaces before or after commas and equal symbols, the application server normalizes the distinguished names automatically during run time by removing these extra spaces.

Key alias: **Version 5.x or 6 application**

Specifies the alias for the key object, which is used by the key locator to find the key within the keystore file.

Key password: **Version 5.x or 6 application**

Specifies the password that is needed to access the key object within the keystore file.

Web services: Client security bindings collection:

Use this page to view a list of application-level, client-side binding configurations for Web services security. These bindings are used when a Web service is a client to another Web service.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Security Properties, click **Web services: Client security bindings**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Component: **Version 5.x or 6 application**

Specifies the enterprise bean in an assembled Enterprise JavaBeans (EJB) module.

Port: **Version 5.x or 6 application**

Specifies the port that is used to send messages to a server and receive messages from a server.

Web service: **Version 5.x or 6 application**

Specifies the name of the Web service that is defined during application assembly.

Request generator (sender) binding: **Version 6 and later applications**

Specifies the binding configuration that is used to send request messages to the request consumer.

Click **Edit custom** to configure the required and additional properties such as signing information, key information, token generators, key locators, and collection certificate stores.

The binding information for the request generator that is specified for the client must match the binding information for the request consumer that is specified for the server.

Response consumer (receiver) binding: **Version 6 and later applications**

Specifies the binding configuration that is used to receive response messages from the response generator.

Click **Edit custom** to configure the required and additional properties such as signing information, key information, token consumers, key locators, collection certificate stores, and trust anchors.

The binding information for the response consumer that is specified for the client must match the binding information for the response generator that is specified for the server.

Request sender binding: **Version 5.x application**

Specifies the binding configuration that is used to send request messages to the request receiver.

Click **Edit** to configure the additional properties for the request sender such as signing information, key information, encryption information, key locators, and the login binding.

The binding information for the request sender that is specified for the client must match the binding information for the request receiver that is specified for the server.

Response receiver binding: **Version 5.x application**

Specifies the binding configuration that is used to receive response messages from the response sender.

Click **Edit** to configure the additional properties for the response receiver such as signing information, encryption information, trust anchors, collection certificate stores, and key locators.

The binding information for the response receiver that is specified for the client must match the binding information for the response sender that is specified for the server.

HTTP basic authentication: **Version 5.x or 6 application**

Specifies the user name and password to use for this port with HTTP transport-level basic authentication. You can enable transport-level authentication security independently of message-level security.

Click **Edit** to configure the basic authentication ID and password for transport-level authentication.

HTTP SSL configuration: **Version 5.x or 6 application**

Enables and configures transport-level Secure Sockets Layer (SSL) security for this port. You can enable transport-level SSL security independently of message-level security.

Click **Edit** to specify the settings for transport-level HTTP SSL configuration for this port.

Web services: Server security bindings collection:

Use this page to view a list of server-side binding configurations for Web services security.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page, complete the following steps:
1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Port: **Version 5.x or 6 application**

Specifies the port in which messages are received from the request generator or the request sender. The request generator is the term that is used for Version 6.*x* applications and the request sender is the term that is used for Version 5.*x* applications.

Web service:

Specifies the name of the Web service that is defined during application assembly.

Request consumer (receiver) binding: **Version 6 and later applications**

Specifies the binding configuration that is used to receive request messages from the request generator (sender) binding.

Click **Edit custom** to configure the required and additional information such as signing information, key information, token consumers, key locators, intermediate certificates in the collection certificate store, and trust anchors.

The binding information for the request consumer that is specified for the server must match the binding information for the request generator that is specified for the client.

Response generator (sender) binding: **Version 6 and later applications**

Specifies the binding configuration that is used to send request messages to the response consumer.

Click **Edit custom** to configure the required and additional information such as signing information, key information, token generators, key locators, and intermediate certificates in the collection certificate store.

The binding information for the response generator that is specified for the server must match the binding information for the response consumer that is specified for the client.

Request receiver binding: **Version 5.x application**

Specifies the binding configuration that is used to receive request messages from the request sender binding.

Click **Edit** to configure additional properties for the request receiver such as signing information, encryption information, trust anchors, collection certificate stores, key locators, trusted ID evaluators, and login mappings.

The binding information for the request receiver that is specified for the server must match the binding information for the request sender that is specified for the client.

Response sender binding: **Version 5.x application**

Specifies the binding configuration that is used to send request messages to the response receiver.

Click **Edit** to configure additional properties for the response sender such as signing information, encryption information, and key locators.

The binding information for the response sender that is specified for the server must match the binding information for the response receiver that is specified for the client.

Configuring token generators using JAX-RPC to protect message authenticity at the server level:

Version 6 and later applications

The token generator on the server level is used to specify the information for the token generator if these bindings are not defined at the application level. The signing information and the encryption information can share the token generator information, which is why they are all defined at the same level.

About this task

WebSphere Application Server provides default values for bindings. You must modify the defaults for a production environment.

Complete the following steps to configure the token generators on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Token generators**.
3. Click **New** to create a token generator configuration, click **Delete** to delete an existing configuration, or click the name of an existing token generator configuration to edit its settings. If you are creating a new configuration, enter a unique name for the token generator configuration in the **Token generator name** field. For example, you might specify `sig_tgen`. This field specifies the name of the token generator element.
4. Specify a class name in the **Token generator class name** field. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side.

Note: The com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface is not used with JAX-WS Web services. If you are using JAX-RPC Web services, this interface is still valid.

The token generator class name must be similar to the token consumer class name. For example, if your application requires an X.509 certificate token consumer, you can specify the com.ibm.wsspi.wssecurity.token.X509TokenConsumer class name on the Token consumer panel and the com.ibm.wsspi.wssecurity.token.X509TokenGenerator class name in this field. WebSphere Application Server provides the following default token generator class implementations:

com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator
This implementation generates a username token.

com.ibm.wsspi.wssecurity.token.X509TokenGenerator
This implementation generates an X.509 certificate token.

com.ibm.wsspi.wssecurity.token.LTPATokenGenerator
This implementation generates a Lightweight Third Party Authentication (LTPA) token.

5. Select a certificate path option. The certificate path specifies the certificate revocation list (CRL), which is used for generating a security token that is wrapped in a PKCS#7 with a CRL. WebSphere Application Server provides the following certificate path options:

   None Select this option in case the CRL is not used for generating a security token. You must select this option when the token generator does not use the PKCS#7 token type.

   Dedicated signing information
   If the CRL is wrapped in a security token, select **Dedicated signing information** and select a collection certificate store name from the **Certificate store** field. The **Certificate store** field shows the names of collection certificate stores already defined.

6. Select the **Add nonce** option to include a nonce in the user name token for the token generator. Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Add nonce** option is available if you specify a user name token for the token generator.
7. Select the **Add timestamp** option to include a time stamp in the user name token for the token generator.
8. Specify a value type local name in the **Local name** field. This entry specifies the local name of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as Key information type. To specify the Key information type, see “Configuring the key information for the generator binding using JAX-RPC on the server level” on page 364. WebSphere Application Server provides the following predefined X.509 certificate token configurations:

   X.509 certificate token
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

   X.509 certificates in a PKIPath
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

   A list of X.509 certificates and CRLs in a PKCS#7
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

   LTPA For LTPA, the value type local name is LTPA. If you enter `LTPA` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 uniform resource identifier (URI) value in the **Value type URI** field as well.

   LTPA version 2
   For LTPA version 2, the value type local name is LTPAv2. If you enter LTPAv2 for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype uniform resource identifier (URI) value in the **Value type URI** field as well.

LTPA_PROPAGATION
For LTPA token propagation, the value type local name is `LTPA_PROPAGATION`. If you enter `LTPA_PROPAGATION` for the local name, you must specify the `http://www.ibm.com/websphere/appserver/tokentype` URI value in the **Value type URI** field as well.

For example, when an X.509 certificate token is specified, you can use `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` for the local name.

9. Specify the value type URI in the **URI** field. This entry specifies the namespace URI of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as Key information type on the Key information panel for the default generator. When the X.509 certificate token is specified, you do not need to specify the namespace URI. If another token is specified, you must specify the namespace URI of the value type.
10. Click **OK** and then **Save** to save the configuration.
11. Click the name of your token generator configuration.
12. Under Additional properties, click **Callback handler** to configure the callback handler properties. The callback handler specifies how to acquire the security token that is inserted in the Web services security header within the SOAP message. The token acquisition is a pluggable framework that leverages the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface for acquiring the security token.
    a. Specify a callback handler class implementation in the **Callback handler class name** field. This attribute specifies the name of the Callback handler class implementation that is used to plug in a security token framework. The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler class. WebSphere Application Server provides the following default callback handler implementations:

    com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
    This callback handler uses a login prompt to gather the user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

    com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
    This callback handler does not issue a prompt and returns the user name and password if it is specified in the basic authentication section of this panel. You can use this callback handler when the Web service is acting as a client.

    com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
    This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified in the basic authentication section of this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java Platform, Enterprise Edition (Java EE) application client only.

    com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
    This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified in the basic authentication section of this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token. It obtains the security token this way rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a Java EE application client.

    com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler
    This callback handler is used to create the X.509 certificate that is inserted in the Web

services security header within the SOAP message as a binary security token. A keystore file and a key definition are required for this callback handler.

com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
This callback handler is used to create X.509 certificates that are encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore file is required for this callback handler. You must specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. For more information on configuring the collection certificate store, see "Configuring the collection certificate on the server level" on page 492.

com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
This callback handler is used to create X.509 certificates that are encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore file is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used.

For an X.509 certificate token, you might specify the com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler implementation.

b. Optional: Select the **Use identity assertion** option. Select this option if you have identity assertion that is defined in the IBM extended deployment descriptor. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server sends only the user name of the original caller for a user name token generator. For an X.509 token generator, the application server sends the original signer certification only.

c. Optional: Select the **Use RunAs identity** option. Select this option if the following conditions are true:
   - You have identity assertion defined in the IBM extended deployment descriptor.
   - You want to use the Run As identity instead of the initial caller identity for identity assertion for a downstream call.

d. Optional: Specify a basic authentication user ID and password in the **User ID** and **Password** fields. This entry specifies the user name and password that is passed to the constructors of the callback handler implementation. The basic authentication user ID and password are used if you specify one of the following default callback handler implementations that are provided by WebSphere Application Server:
   - com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler

e. Optional: Specify a keystore password and path. The keystore and its related information are necessary when the key or certificate is used for generating a token. For example, the keystore information is required if you select one of the following default callback handler implementations that are provided by WebSphere Application Server:
   - com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler

   The keystore files contain public and private keys, root certificate authority (CA) certificates, intermediate CA certificates, and so on. Keys that are retrieved from the keystore file are used to sign and validate or encrypt and decrypt messages or message parts. To retrieve a key from a keystore file, you must specify the keystore password, the keystore path, and the keystore type.

13. Select a keystore type from the **Type** field. WebSphere Application Server provides the following options:

JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

JCEKS
Use this option if you are using Java Cryptography Extensions.

PKCS11KS (PKCS11)
Use this format if your keystore file uses the PKCS#11 file format. Key store files using this format might contain RSA keys on cryptographic hardware or might encrypt the keys that use cryptographic hardware to ensure protection.

PKCS12KS (PKCS12)
Use this option if your keystore file uses the PKCS#12 file format.

14. Click **OK** and then **Save** to save the configuration.
15. Click the name of your token generator configuration.
16. Under Additional properties, click **Callback handler > Keys**.
17. Click **New** to create a key configuration, click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. If you are creating a new configuration, enter a unique name for the key configuration in the **Key name** field. This name refers to the name of the key object that is stored within the keystore file.
18. Specify an alias for the key object in the **Key alias** field. Use the alias when the key locator searches for the key objects in the keystore.
19. Specify the password that is associated with the key in the **Key password** field.
20. Click **OK** and **Save** to save the configuration.

Results

You have configured the token generators at the server level.

What to do next

You must specify a similar token consumer configuration.

Token generator collection:

Use this page to view the token generators. The information is used on the generator side only to generate the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings, click **Token generators**.

Token generator name: **Version 6 and later applications**

Specifies the name of the token generator configuration.

For example, the default X509 token generator names are either `gen_enctgen` for encrypting or `gen_signtgen` for signing. Or a custom token generator name might be `sig_tgen` for signing.

Token generator class name: **Version 6 and later applications**

Specifies the name of the token generator implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

Token generator class name: **Version 6 and later applications**

Specifies the name of the token generator implementation class.

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side.

Token generator configuration settings:

Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Generator Bindings, click **Token generators >** ***token_generator_name*** or click **New** to create a new token generator.

1. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name***.
2. Under Modules, click **Manage modules>** ***URI_name***.
3. Under Additional properties, you can access the token generator information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. Click **New** to create a new token generator or click the name of an existing token generator name to specify its settings.

To view this administrative console page for the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name***.
2. Under Modules, click **Manage modules >** ***URI_name***.
3. Under Web Services Security Properties, click **Web services: Client security bindings**.
4. Under Request generator (sender) binding, click **Edit custom**.
5. Under Additional properties, click **Token generators > New**.

Before specifying additional properties, specify a value in the **Token generator name** and the **Token generator class name** fields.

Token generator name: **Version 6 and later applications**

Specifies the name of the token generator configuration.

For example, the default X509 token generator names are either `gen_enctgen` for encrypting or `gen_signtgen` for signing. Or, a custom token generator name might be `sig_tgen` for signing.

Token generator class name: **Version 6 and later applications**

Specifies the name of the token generator implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

Token generator class name: **Version 6 and later applications**

Specifies the name of the token generator implementation class.

Certificate path: **Version 6 and later applications**

Specifies the certificate revocation list (CRL) that is used for generating a security token wrapped in a PKCS#7 token type with CRL.

When the token generator is not for a PKCS#7 token type, you must select **None**. When the token generator is for the PKCS#7 token type and you want to package CRL in the security token, select **Dedicated signing information** and specify the CRL for the collection certificate store.

You can specify a certificate store configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default generator bindings	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Collection certificate store**.

Using the collection certificate store, you can configure a related certificate revocation list by clicking **Certificate revocation list** under Additional properties.

Add nonce: **Version 6 and later applications**

Indicates whether nonce is included in the user name token for the token generator. *Nonce* is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens.

On the application level, if you select the **Add nonce** option, you can specify the following properties under Additional properties:

Table 64. Additional nonce properties

Property name	Default value	Explanation
com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.cacheTimeout	600 seconds	Specifies the timeout value, in seconds, for the nonce value that is cached on the server.
com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.clockSkew	0 seconds	Specifies the time, in seconds, before the nonce time stamp expires.
com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.maxAge	300 seconds	Specifies the clock skew value, in seconds, to consider when the application server checks the timeliness of the message.

These properties are available on the administrative console at the cell and server level. However, on the application level, you can configure the properties under Additional properties.

This option is displayed on the cell, server, and application levels. This option is valid only when the generated token type is a user name token.

Add timestamp: **Version 6 and later applications**

Specifies whether to insert the time stamp into the user name token.

This option is displayed on the cell, server, and application levels. This option is valid only when the generated token type is a user name token.

Value type local name: **Version 6 and later applications**

Specifies the local name of the value type for the generated token.

For a user name token and an X.509 certificate security token, this product provides predefined value types. When you specify the following local names, you do not need to specify the Uniform Resource Identifier (URI) of value type.

Username token
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

X509 certificate token
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

X509 certificates in a PKIPath
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

A list of X509 certificates and CRLs in a PKCS#7
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

Note: For LTPA, the value type local name is `LTPA`. If you enter `LTPA` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well. For LTPA token propagation, the value type local name is `LTPA_PROPAGATION`. If you enter `LTPA_PROPAGATION` for the local name, you must specify the http://www.ibm.com/websphere/

appserver/tokentype URI value in the Value type URI field as well. For the other predefined value types (Username token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the local name field begins with `http://`. For example, if you are specifying the user name token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the Value type local name field and then you do not need to enter a value in the Value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

Value type URI: **Version 6 and later applications**

Specifies the namespace URI of the value type for the generated token.

When you specify the token generator for the user name token or the X.509 certificate security token, you do not need to specify this option. If you want to specify another token, specify the URI of the QName of the value type.

The application server provides the following predefined value type URIs:

- For the LTPA token: http://www.ibm.com/websphere/appserver/tokentype/5.0.2
- For the LTPA token propagation: http://www.ibm.com/websphere/appserver/tokentype

Algorithm URI collection:

Use this page to view a list of uniform resource identifier (URI) algorithms for XML digital signature or XML encryption that are mapped to an algorithm factory engine class. With algorithm mappings, service providers can use other cryptographic algorithms for digest value calculation, digital signature signing and verification, data encryption and decryption, and key encryption and decryption.

To view administrative console page on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.
4. Click on an algorithm mapping name.
5. Under Additional properties, click **Algorithm URI**.

Algorithm URI: **Version 6 and later applications**

Specifies the algorithm uniform resource identifier (URI) for the specified algorithm type.

Algorithm type: **Version 6 and later applications**

Specifies the algorithm type.

Algorithm URI configuration settings:

Use this page to specify the algorithm uniform resource identifier (URI) and its usage type.

This product supports the following algorithm URI types:

Message digest
Specifies the algorithm URI that is used for digest value calculation.

Signature
Specifies the algorithm URI that is used for digital signature, including both signature and signing verification.

Data encryption
Specifies the algorithm URI that is used for both encrypting and decrypting data.

Key encryption
Specifies the algorithm URI that is used for encrypting and decrypting the encryption key.

If the URI is used for multiple usage types, then you must define a mapping of the URI to each usage type.

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.

   Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS)** option has been selected on the SSL certificate and key management panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.
4. Click **New**.
5. Under Additional properties, click **Algorithm URI >** ***algorithm_URI_name***.

To view the administrative console page on the cell level:

1. Click **Security > JAX-WS and JAX-RPC security runtime**, or **Services > JAX-WS and JAX-RPC security runtime**.
2. Under Additional properties, click **Algorithm mappings**.
3. Click **New**.
4. Under Additional properties, click **Algorithm URI >** ***algorithm_URI_name***.

Algorithm URI: **Version 6 and later applications**

Specifies the algorithm uniform resource identifier (URI) for the specified algorithm type.

The algorithm URI that is defined on this page is available to the various binding configurations. For example, if you specify an algorithm URI and select **Signature** from the Algorithm type field, the URI displays in the Signature method field on the signing information panel.

Algorithm type: **Version 6 and later applications**

Specifies the type of algorithm that is specified in the Algorithm URI field.

The following types of algorithms are supported by this product. The following list shows where configurations that are specified on this panel are displayed for a binding configuration:

Algorithm type	Explanation	Location of the configuration
Signature	This algorithm type is used for digital signatures.	This configuration displays in the Signature method field on the Signing information panel. For information on how to access the Signing information panel, see the help topic Signing information configuration settings.
Digest value calculation (message digest)	This algorithm type is used for calculating the digest value.	This configuration displays in the Digest method algorithm field on the Part references panel. For information on how to access the Part references panel, see the help topic Part reference configuration settings.
Data encryption	This algorithm type is used for encrypting data.	This configuration displays in the Data encryption algorithm field on the Encryption information panel. For information on how to access the Encryption information panel, see the help topic Encryption information configuration settings: Message parts.
Key encryption	This algorithm type is used for encrypting the key that is used for data encryption.	This configuration displays in the Key encryption algorithm field on the Encryption information panel. For information on how to access the Encryption information panel, see the help topic Encryption information configuration settings: Message parts.

The actual implementation of the algorithm is done in the implementation class for the engine factory.

Algorithm mapping collection:

You can view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. The application server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

To view this administrative console page on the cell level, complete the following steps:

1. Click **Security > JAX-WS and JAX-RPC security runtime**.
2. Under Additional properties, click **Algorithm mappings**.

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.

Algorithm factory engine class: **Version 6 and later applications**

Specifies the custom class that implements the engine factory implementation class for the algorithm factory engine.

The implementation class for the engine factory implements the cryptographic functions of the defined uniform resource identifier (URI).

Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS) algorithms** option has been selected on the Global security panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.

Algorithm mapping configuration settings:

Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. The application server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings > *algorithm_factory_engine_class_name***.

   Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS)** option has been selected on the SSL certificate and key management panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.
4. Click **New**.

To view this administrative console page on the cell level:

1. Click **Security > JAX-WS and JAX-RPC security runtime**, or **Services > JAX-WS and JAX-RPC security runtime**.
2. Under Additional properties, click **Algorithm mappings > *algorithm_factory_engine_class_name***.
3. Click **New**.

Algorithm factory engine class: **Version 6 and later applications**

Specifies the custom class that implements the engine factory interface.

To use this algorithm mapping feature, you must specify a custom algorithm class in the Algorithm factory engine class field for digital signature, data encryption, digest value calculation, and key encryption. The algorithm factory engine provides a plug-in point for service providers to provide their implementation for digest value calculation, digital signature, key encryption, and data encryption that is based on a specified algorithm uniform resource identifier (URI). By clicking **Algorithm URI** under Additional properties, you can specify the algorithm URI and its usage type. This product supports the following algorithm types:

Message digest
Specifies the algorithm URI that is used for digest value calculation.

Signature
Specifies the algorithm URI that is used for digital signatures including both signing and signature verification.

Data encryption
Specifies the algorithm URI that is used for both encrypting and decrypting data.

Key encryption
Specifies the algorithm URI that is used for both encrypting and decrypting the encryption key.

If the URI is used for multiple usage types, then you must define a mapping of the URI to each usage type. The actual implementation of the algorithm is provided by the custom class that implements the engine factory interface. For more information, refer to the information center documentation on how to implement a factory class.

Enabling or disabling single sign-on interoperability mode for the LTPA token:

You can set an interoperability flag on the token generator to determine whether an LTPA Version 1 token or an LTPA Version 2 token is retrieved when a request message is received.

About this task

In WebSphere Application Server Version 7.0, a flag is set in the global security settings to enable single sign-on interoperability mode for the LTPA token. This option determines whether an LTPA Version 1 token or an LTPA Version 2 token is sent when a message request is received. When the interoperability flag is set to true, then the AuthenticationToken is an LTPA Version 1 token, and the SingleSignonToken is an LTPA Version 2 token. When the interoperability flag is set to false, then both the AuthenticationToken and the SingleSignonToken are LTPA Version 2 tokens.

When the interoperability mode is enabled (the flag is set to true), and the Web Services security binding configuration specifies LTPA Version 1 as the token, the AuthenticationToken is used to retrieve the token that is sent with the message. If interoperability mode is not enabled (the flag is set to false), and the Web Services security binding configuration specifies LTPA Version 1 as the token, an exception error is logged.

You can disable the interoperability checking function by setting the custom property, com.ibm.wsspi.wssecurity.tokenGenerator.ltpav1.pre.v7, on the token generator. This setting determines the LTPA token without checking the state of the interoperability flag, providing compatibility with servers running WebSphere Application Server Version 6.1 and earlier.

To enforce use of the LTPA Version 2 token, edit the token settings, and set the **Enforce token version** option for the token.

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned an application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.
7. Click a consumer or generator token link from the Protection Tokens table.
8. Select the **Enforce token version** check box after the **Token type** field.

Related tasks

“Configuring token generators using JAX-RPC to protect message authenticity at the server level” on page 412

The token generator on the server level is used to specify the information for the token generator if these bindings are not defined at the application level. The signing information and the encryption information can share the token generator information, which is why they are all defined at the same level.

Related reference

Authentication generator or consumer token settings

Authentication tokens are used to prove or assert an identity. Use the administrative console to add authentication token settings for message parts when you are editing a general binding.

Securing messages using JAX-RPC at the request and response consumers

You can secure messages at the request and response consumer level to protect message confidentiality and security.

About this task

To secure messages, you can:

- Configure signing to protect message confidentiality
- Configure encryption to protect message confidentiality at the server level and at the application level
- Configure tokens to protect message authenticity at the server level and at the application level
- To configure consumer signing to protect message confidentiality, see the steps outlined in “Configuring consumer signing using JAX-RPC to protect message integrity”
- To configure encryption to protect message confidentiality at the application level, see the steps outlined in “Configuring encryption to protect message confidentiality at the application level” on page 437.
- To configure encryption to protect message confidentiality at the server level, see the steps outlined in “Configuring encryption to protect message confidentiality at the server level” on page 440.
- To configure tokens at the application level to protect message authenticity, see the steps outlined in “Configuring token consumers using JAX-RPC to protect message authenticity at the application level” on page 441.
- To configure tokens at the server level to protect message authenticity, see the steps outlined in “Configuring token consumers using JAX-RPC to protect message authenticity at the server level” on page 450.

Results

By completing the steps in the previous tasks, you have secured messages at the request and response consumer level.

Configuring consumer signing using JAX-RPC to protect message integrity:

You can configure protect message integrity by configuring signing and key information at the server and application level.

Before you begin

About this task

To protect message integrity, you can:

- Configure the signing information for the consumer binding on the application level or at the server level
- Configure the key information for the consumer binding on the application level or at the server level

- To configure the signing information for the consumer binding on the application level, see the steps outlined in “Configuring the signing information using JAX-RPC for the consumer binding on the application level”
- To configure the signing information for the consumer binding on the server level, see the steps outlined in “Configuring the signing information using JAX-RPC for the consumer binding on the server level” on page 431
- To configure the key information for the consumer binding on the application level, see the steps outlined in “Configuring the key information for the consumer binding on the application level” on page 434
- To configure the key information for the consumer binding on the server level, see the steps outlined in “Configuring the key information for the consumer binding using JAX-RPC on the server level” on page 436

Results

By completing the steps in these tasks, you have configured the consumer signing to protect the integrity of messages.

Configuring the signing information using JAX-RPC for the consumer binding on the application level:

You can configure the signing information for the server-side request consumer and the client-side response consumer bindings at the application level.

Before you begin

Note: For WebSphere Application Server version 6.x or earlier only, in the server-side extensions file and the client-side deployment descriptor extensions file, you must specify which parts of the message are signed.

About this task

Configure the key information that is referenced by the key information references on the signing information panel within the administrative console. WebSphere Application Server uses the signing information on the consumer side to verify the integrity of the received SOAP message by validating that the message parts are signed. Complete the following steps to configure the signing information for the server-side request consumer and client-side response consumer sections of the bindings files on the application level.

1. Access the administrative console.

   To access the administrative console, enter `http://`*`server_name`*`:`*`port_number`*`/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Application Types > WebSphere enterprise applications >** ***application_name***.
3. Under Manage modules, click ***URI_name***.
4. Under Web Services Security Properties you can access the signing information for the request generator and response generator bindings.
   - To configure the request consumer signing information, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - To configure the response consumer signing information, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
5. Under Required properties, click **Signing information**.
6. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Signing information name field.

7. Select a signature method algorithm from the Signature method field. The signature method is the algorithm that is used to convert the canonicalized <SignedInfo> element in the binding file into the <SignatureValue> element. The algorithm that is specified for the consumer, which is either the request consumer or the response consumer configuration, must match the algorithm specified for the generator, which is either the request generator or response generator configuration. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1

     Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a signature based on a symmetric key must have a value of http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#hmac-sha1.
8. Select a canonicalization method from the Canonicalization method field. The canonicalization method algorithm is used to canonicalize the <SignedInfo> element before it is incorporated as part of the digital signature operation. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
9. Select a key information signature type from the Key information signature type field. The key information signature type specifies how the <KeyInfo> element in the SOAP message is digitally signed. WebSphere Application Server supports the following signature types:

   None Specifies that the key is not signed.

   Keyinfo
   Specifies that the entire KeyInfo element is signed.

   Keyinfochildelements
   Specifies that the child elements of the KeyInfo element are signed.

   If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default. The key information signature type for the consumer must match the signature type for the generator.
10. Under Additional properties, click **Key information references**.
    a. Click **New** to create a key information reference or click the name of an existing entry to edit its configuration. The Key information references panel is displayed.
    b. Enter a name in the Name field.
    c. Select a key information reference in the Key information reference field. This reference is the key information configuration name that specifies the key information that is used by this signing information configuration.
11. Return to the Signing information panel. Under Additional properties, click **Part references**. On the Part references panel, you can specify references to the message parts that are defined in the deployment descriptor extensions file.
    a. Click **New** to create a new Part reference or click the name of an existing part reference to edit its configuration. The Part reference panel is displayed.
    b. Enter a name in the Part name field. This name is the name of the required integrity configuration in the deployment descriptor extensions file and specifies the message parts that must be digitally signed.
    c. Select a digest method algorithm from the Digest method algorithm field.

WebSphere Application Server supports the following pre-configured algorithms:

- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmlenc#sha512

If you want to specify a custom algorithm, you must configure the custom algorithm in the Algorithm URI panel before setting the digest method algorithm.

12. Under Additional properties, click **Transforms**.
    a. Click **New** to create a new transform or click the name of an existing transform to edit its configuration.
    b. Enter a name in the Transform name field.
    c. Select a transform algorithm from the Transform algorithm field. WebSphere Application Server supports the following pre-configured algorithms:
       - http://www.w3.org/2001/10/xml-exc-c14n#
       - http://www.w3.org/TR/1999/REC-xpath-19991116

         Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.
       - http://www.w3.org/2002/06/xmldsig-filter2
       - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
       - http://www.w3.org/2002/07/decrypt#XML
       - http://www.w3.org/2000/09/xmldsig#enveloped-signature

       The transform algorithm that you select for the consumer must match the transform algorithm that you select for the generator. For each part reference in the signing information, specify both a digest method algorithm and a transform algorithm.
13. Click **OK**.
14. Click **Save** at the top of the panel to save your configuration.

Results

After completing these steps, you have configured the signing information for the consumer.

What to do next

You must specify a similar signing information configuration for the generator.

Key information references collection:

Use this page to view the key information references that are needed for encryption or signing.

To view this administrative console page on the server level, complete the following steps. On the server level, you can configure the key information references for the default consumer bindings only.

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Consumer Bindings, click either of the following links:
   - Click **Encryption information >** ***encryption_information_name***.
   - Click **Signing information >** ***signing_information_name***.

4. Under Additional properties, click **Key information reference**.

To view this administrative console page on the application level, complete the following steps. On the application level, you can configure the key information reference for the consumer bindings only.

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security properties, you can access the signing information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**. Click **New** to create a new encryption configuration or click the name of a configuration to modify its settings.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**. Click **New** to create a new encryption configuration or click the name of a configuration to modify its settings.

Name: **Version 6 and later applications**

Specifies the name of the Key information reference.

Key information reference: **Version 6 and later applications**

Specifies a reference to the message parts that are signed or encrypted.

The value of this field is the name of the <requiredIntegrity> or the <requiredConfidentiality> element in the deployment descriptor.

Key information reference configuration settings:

Use this page to specify a reference to the message parts for signature and encryption that is defined in the deployment descriptors.

To view this administrative console page on the server level for the key information references, complete the following steps. On the server level, you can configure the key information references for the default consumer bindings only.

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Consumer Bindings, click either of the following links:
   - Click **Encryption information > *encryption_information_name***.
   - Click **Signing information > *signing_information_name***.
4. Under Additional properties, click **Key information references**.

To view this administrative console page on the application level, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules > *URI_name***.

3. Under Web Services Security Properties, you can access the key information references for the following bindings:
   - For the Response consumer (sender) binding, click **Web services: Client security bindings**. Under Response consumer (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information** > ***encryption_information_name***. Under Additional properties, click **Key information references**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information** > ***encryption_information_name***. Under Additional properties, click **Key information references**.

Name: **Version 6 and later applications**

Specifies the name of the key information reference.

Key information reference: **Version 6 and later applications**

Specifies a reference to the message parts that are signed or encrypted.

The value of this field is the name of the <requiredIntegrity> or the <requiredConfidentiality> element in the deployment descriptor. You can specify a signing key configuration for the following bindings:

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers** > **Server Types** > **WebSphere application servers** > *server_name*. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under JAX-RPC Default Consumer Bindings, click **Key information**.
Response consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** . 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Under Web Services Security Properties, click **Web services: Client security bindings**. 4. Under Response consumer (receiver) binding, click **Edit custom**. 5. Under Required properties, click **Key information**.

Binding name	Server level or application level	Path
Request consumer (receiver) binding	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** > ***URI_name***. 3. Under Web Services Security Properties, click **Web services: Server security bindings**. 4. Under Request consumer (receiver) binding, click **Edit custom**. 5. Under Required properties, click **Key information**.

Configuring the signing information using JAX-RPC for the consumer binding on the server level:

Version 6 and later applications

You can configure the signing information for the client-side request generator and server-side response generator bindings at the server level.

Before you begin

Note: For WebSphere Application Server version 6.x or earlier only, in the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

About this task

This task explains the steps that are needed for you to configure the signing information for the client-side request generator and server-side response generator bindings at the server level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message including the body, time stamp, and user name token, if these bindings are not defined at the application level. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the signing information for the consumer sections of the bindings files on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Signing information**.
3. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the signing configuration in the Signing information name field. For example, you might specify `gen_signinfo`.

4. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the default consumer must match the algorithm that is specified for the default generator. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1

     Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a SIGNATURE based on a symmetric key must have a value of http://www.w3.org/2000/09/xmldsig#rsa-sha1 or http://www.w3.org/2000/09/xmldsig#hmac-sha1.
5. Select a canonicalization method from the Canonicalization method field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured canonical XML and exclusive XML canonicalization algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
6. Select a key information signature type from the Key information signature type field. The key information signature type determines how to digitally sign the key. WebSphere Application Server supports the following signature types:

   None Specifies that the KeyInfo element is not signed.

   Keyinfo
   Specifies that the entire KeyInfo element is signed.

   Keyinfochildelements
   Specifies that the child elements of the KeyInfo element are signed.

   The key information signature type for the consumer must match the signature type for the generator. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
7. Click **OK** to save the configuration.
8. Click the name of the new signing information configuration. This configuration is the one that you specified in the previous steps.
9. Specify the key information reference, part reference, digest algorithm, and transform algorithm.
   a. Under Additional properties, click **Key information references > New** to create a new reference, click **Key information references > Delete** to delete an existing reference, or click a reference name to edit an existing key information reference.
   b. Enter a name for the configuration in the Name field. For example, enter `con_skeyinfo`.
   c. Select a key information reference from the Key information reference field. The key Information reference points to the key that WebSphere Application Server uses for digital signing. In the binding files, the reference is specified within the <signingKeyInfo> element. The key that is used for signing is specified by the Key information element, which is defined at the same level as the signing information. For more information, see “Configuring the key information for the consumer binding on the application level” on page 434.
   d. Click **OK** and **Save** to save the configuration.

e. Under Additional Properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference. The part reference specifies which parts of the message to digitally sign. The part attribute refers to the name of the <RequiredIntegrity> element in the deployment descriptor when <PartReference> is specified for the digital signature. WebSphere Application Server enables you to specify multiple <PartReference> elements for the <SigningInfo> element. The `<PartReference>` element has two child elements: <DigestMethod> and <Transform>

f. Specify a unique part name for this part reference. For example, you might specify `reqint`.

Note: You do not need to specify a value for the Part reference field like you specify on the application level because the part reference on the application level points to a particular part of the message that is signed. Because the default bindings for the server level is applicable to all of the services that are defined on a particular server, you cannot specify this value.

g. Select a digest method algorithm in the **Digest method algorithm** field. The digest method algorithm specified within the <DigestMethod> element that is used in the <SigningInfo> element.

WebSphere Application Server supports the following algorithms:

- http://www.w3.org/2000/09/xmldsig#sha1
- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmlenc#sha512

h. Click **OK** and **Save** to save the configuration.

i. Click the name of the new part reference configuration. This configuration is the one that you specified in the previous steps.

j. Under Additional properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.

k. Select a transform algorithm from the menu. The transform algorithm is specified within the <Transform> element. It specifies the transform algorithm for the signature. WebSphere Application Server supports the following algorithms:

- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/TR/1999/REC-xpath-19991116

Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use http://www.w3.org/2002/06/xmldsig-filter2 to ensure compliance.

- http://www.w3.org/2002/06/xmldsig-filter2
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
- http://www.w3.org/2002/07/decrypt#XML
- http://www.w3.org/2000/09/xmldsig#enveloped-signature

The transform algorithm that you select for the consumer must match the transform algorithm that you select for the generator.

Note: If both of the following conditions are true, WebSphere Application Server signs the referenced token:

- You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
- You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

10. Click **OK**.

11. Click **Save** at the top of the panel to save your configuration.

Results

After completing these steps, you have configured the signing information for the consumer on the server level.

What to do next

You must specify a similar signing information configuration for the generator.

Configuring the key information for the consumer binding on the application level:

Version 6 and later applications

You can configure the key information for the request consumer (server side) and the response consumer (client side) bindings at the application level.

Before you begin

Configure the key locators and the token consumers that are referenced by the Key locator reference and the Token reference fields within the key information panel.

About this task

This task provides the steps that are needed for configuring the key information for the request consumer (server side) and the response consumer (client side) bindings at the application level. The key information on the consumer side is used for specifying the information about the key, which is used for validating the digital signature in the received message or for decrypting the encrypted parts of the message. Complete the following steps to configure the key information for consumer binding on the application level.

1. Locate the key information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties, you can access the key information for the request consumer and response consumer bindings.
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under response consumer (receiver) binding, click **Edit custom**.
   d. Under Required properties, click **Key information**.
   e. Click one of the following to work with key information configuration:

      New To create a key information configuration. Enter a name in the Key information name field. For example, you might specify `con_signkeyinfo`.

      Delete To delete a configuration (selected in the box next to that configuration).

      an existing key information configuration
      To edit the settings of a key information configuration.
2. Select a key information type from the Key information type field. The key information types specify different mechanisms for referencing security tokens using the <wsse:SecurityTokenReference> element within the <ds:KeyInfo> element. WebSphere Application Server supports the following key information types:

Key identifier

The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the <KeyIdentifier> element value depends upon the token type. For example, you can use the identifier for the public keys that are defined in the Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01
      /oasis-200401-wss-x509-token-profile-1.0#X509v3SubjectKeyIdentifier">
      /62wXO...
      </wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

Key name

The security token is referenced using a name that matches an identity assertion within the token. It is recommended that you do not use this key type as it might result in multiple security tokens that match the specified name. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <ds:KeyName>CN=Group1</ds:KeyName>
</ds:KeyInfo>
```

In general, use a key name when you use a Key-Hashing Message Authentication Code (HMAC) digital signature algorithm, such as http://www.w3.org/2000/09/xmldsig#hmac-sha1.

Security token reference

The security token is directly referenced using Universal Resource Identifiers (URIs). The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Reference URI='#SomeCert'
       ValueType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-x509-token-profile-1.0#X509v3" />
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

Note: As stated in the Web services Interoperability Organization (WS-I) Basic Security Profile Version 1 draft and shown in the previous example, the wsse:Reference element in a SECURE_ENVELOPE must have a ValueType attribute.

Embedded token

The security token is directly embedded within the <SecurityTokenReference> element. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
         ...
      </wsse:Embedded>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

X509 issuer name and issuer serial

The security token is referenced by an issuer name and an issuer serial number of an X.509 certificate. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
     <ds:X509Data>
        <ds:X509IssuerSerial>
           <ds:X509IssuerName>CN=Jones, O=IBM, C=US</ds:X509IssuerName>
           <ds:X509SerialNumber>1040152879</ds:X509SerialNumber>
```

```
            </ds:X509IssuerSerial>
          </ds:X509Data>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
```

Each type of key information is described in the Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS standard, which is located at: http://www.oasis-open.org/home/index.php under Web services security.

3. Select a key locator reference from the Key locator reference field. The value of this field is a reference to a key locator that WebSphere Application Server uses to locate the keys that are used for digital signature and encryption. Before you can select a key locator, you must configure a key locator. For more information on configuring a key locator, see “Configuring the key locator using JAX-RPC for the consumer binding on the application level” on page 470.
4. Select a token reference from the Token reference field. The token reference specifies a reference to a token consumer that is used for processing the security token in the message. However, WebSphere Application Server requires this field only when you select Security token reference or Embedded token in the Key information type field. Before specifying a token reference, you must configure a token consumer. For more information on configuring a token consumer, see “Configuring token consumers using JAX-RPC to protect message authenticity at the application level” on page 441.

   Select **(none)** if a token consumer is not required for this key information configuration.
5. Click **OK** and **Save** to save this configuration.

Results

You have configured the key information for the request or response (or both) consumer binding at the application level.

What to do next

If you have not configured the key information for the generator binding, you must specify a similar key information configuration for the generator. After you configure the key information for both the consumer and the generator, configure the signing information or encryption information, which references the key information that is specified in this key information task.

Configuring the key information for the consumer binding using JAX-RPC on the server level:

Version 6 and later applications

The key information for the default consumer is used to specify the key for the signing or the encryption information configurations if these bindings are not defined at the application level.

About this task

The signing and encryption information configurations can share the same key information, which is why they are both defined on the same level. WebSphere Application Server provides default values for these bindings. However, an administrator must modify these values for a production environment.

Complete the following steps to configure the key information for the consumer binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

2. Under Default consumer bindings, click **Key information**.
3. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key configuration in the Key information name field. For example, you might specify `con_signkeyinfo`.
4. Select a key information type from the Key information type field. WebSphere Application Server supports the following types of key information:

   Key identifier
   This key information type is used when two parties agree on how to create a key identifier. For example, a field of X.509 certificates can be used for the key identifier according to the X.509 profile.

   Key name
   This key information type is used when the sender and receiver agree on the name of the key.

   Security token reference
   This key information type is typically used when an X.509 certificate is used for digital signature.

   Embedded token
   This key information type is used to embed a security token in an embedded element.

   X509 issuer name and issuer serial
   This key information type specifies an X.509 certificate with its issuer name and serial number.

   Select **Security token reference** if you are using an X.509 certificate for the digital signature. In these steps, it is assumed that **Security token reference** is selected for this field.

   Note: This key information type must match the key information type that is specified for the generator.
5. Select a key locator reference from the Key locator reference menu. In these steps, assume that the key locator reference is called `sig_klocator`. You must configure a key locator before you can select it in this field. For more information on configuring the key locator, see “Configuring the key locator using JAX-RPC on the server level” on page 472.
6. Select a token reference from the Token reference field. The token reference refers to the name of a configured token consumer. When a security token is required in the deployment descriptor, the token reference attribute is required. If you select **Security token reference** in the Key information type field, the token reference is required and you can specify an X.509 token consumer. To specify an X.509 token consumer, you must have an X.509 token consumer configured. To configure an X.509 token consumer, see “Configuring token consumers using JAX-RPC to protect message authenticity at the server level” on page 450.
7. Click **OK** and **Save** to save the configuration.

Results

You have configured the key information for the consumer binding at the server level.

What to do next

You must specify a similar key information configuration for the generator.

Configuring encryption to protect message confidentiality at the application level:

Version 6 and later applications

You can configure the encryption information for the request consumer (server side) and response consumer (client side) bindings at the application level.

Before you begin

Configure the key information that is referenced in the encryption information panel. For more information, see “Configuring the key information for the consumer binding on the application level” on page 434.

About this task

This task provides the steps that are needed for configuring the encryption information for the request consumer (server side) and response consumer (client side) bindings at the application level. The encryption information on the consumer side is used for decrypting the encrypted message parts in the incoming SOAP message.

Complete the following steps to configure the encryption information for the request consumer or response consumer section of the bindings file on the application level:

1. Locate the Encryption information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the encryption information for the request consumer and response consumer bindings.
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Required properties, click **Encryption information**.
   e. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit its settings. If you are creating a new configuration, enter a name in the **Encryption information name** field. For example, you might specify `cons_enc info`.
2. Select a data encryption algorithm from the **Data encryption algorithm** field. The data encryption algorithm is used for encrypting or decrypting parts of a SOAP message such as the SOAP body or the username token. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

   The data encryption algorithm that you select for the consumer side must match the data encryption method that you select for the generator side.
3. Select a key encryption algorithm from the **Key encryption algorithm** field. The key encryption algorithm is used for encrypting the key that is used for encrypting the message parts within the SOAP message. Select **(none)** if the data encryption key, which is the key that is used for encrypting the message parts, is not encrypted. WebSphere Application Server supports the following pre-configured algorithms:

- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

  When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

  Note: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.
- http://www.w3.org/2001/04/xmlenc#rsa-1_5
- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes256

  To use the http://www.w3.org/2001/04/xmlenc#aes256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- http://www.w3.org/2001/04/xmlenc#kw-aes192

  To use the http://www.w3.org/2001/04/xmlenc#kw-aes192 algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

  Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

The key encryption algorithm that you select for the consumer side must match the key encryption method that you select for the generator side.

4. Optional: Select a part reference in the **Part reference** field. The part reference specifies the name of the message part that is encrypted and is defined in the deployment descriptor. For example, you can encrypt the bodycontent message part in the deployment descriptor. The name of this Required Confidentiality part is `conf_con`. This message part is shown as an option in the **Part reference** field.
5. Under Additional properties, click **Key information references**.
6. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit its settings. If you are creating a new configuration, enter a name in the **Name** field. For example, you might specify `con_ekeyinfo`. This entry is the name of the <encryptionKeyInfo> element in the binding file.
7. Select a key information reference from the **Key information reference** field. This reference is the value of the keyinfoRef attribute of the <encryptionKeyInfo> element and it is the name of the <keyInfo> element that is referenced by this key information reference. Each key information reference entry generates an <encryptionKeyInfo> element under the <encryptionInfo> element in the binding configuration file. For example, if you enter `con_ekeyinfo` in the **Name** field and `dec_keyinfo` in the **Key information reference** field, the following <encryptionKeyInfo> element is generated in the binding file:

   ```
   <encryptionKeyInfo xmi:id="EncryptionKeyInfo_1085092248843"
   keyinfoRef="dec_keyinfo" name="con_ekeyinfo"/>
   ```
8. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the encryption information for the consumer binding at the application level

What to do next

You must specify a similar encryption information configuration for the generator.

Configuring encryption to protect message confidentiality at the server level:

Version 6 and later applications

The encryption information for the default consumer specifies how to process the encryption information on the receiver side if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, an administrator must modify the defaults for a production environment.

About this task

Complete the following steps to configure the encryption information for the consumer binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Encryption information**.
3. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the encryption configuration in the Encryption information name field. For example, you might specify `con_encinfo`.
4. Select a data encryption algorithm from the Data encryption algorithm field. This algorithm is used to encrypt the data. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

     Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

   The data encryption algorithm that you select for the consumer side must match the data encryption algorithm that you select for the generator side.
5. Select a key encryption algorithm from the Key encryption algorithm field. This algorithm is used to encrypt the key. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.

     When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

     Note: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5

- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes256

  To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
- http://www.w3.org/2001/04/xmlenc#kw-aes192

  To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

  Note: Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

If you select **None**, the key is not encrypted.

The key encryption algorithm that you select for the consumer side must match the key encryption algorithm that you select for the generator side.

6. Under Additional properties, click **Key information references**.
7. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key information configuration in the name field. For example, you might specify `con_enckeyinfo`.
8. Select a key information reference from the Key information reference field. This selection refers to the name of the key information that is used for encryption. For more information, see "Configuring the key information for the consumer binding using JAX-RPC on the server level" on page 436.
9. Click **OK** and **Save** to save the configuration.

Results

You have configured the encryption information for the consumer binding at the server level.

What to do next

You must specify a similar encryption information configuration for the generator.

Configuring token consumers using JAX-RPC to protect message authenticity at the application level: Version 6 and later applications

You can specify the token consumer on the application level. The token consumer information is used on the consumer side to incorporate the security token.

About this task

Complete the following steps to configure the token consumer on the application level.

1. Locate the token consumer panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Modules, click **Manage modules > *URI_name***.
   c. Under Web Services Security Properties you can access the token consumer for the following bindings:
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.

- For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.

d. Under Required properties, click **Token consumer**.

e. Click **New** to create a token consumer configuration, click **Delete** to delete an existing configuration, or click the name of an existing token consumer configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Token consumer name** field. For example, you might specify `con_signtcon`.

2. Specify a class name in the **Token consumer class name** field. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side.

   The token consumer class name for the request consumer and the response consumer must be similar to the token generator class name for the request generator and the response generator. For example, if your application requires a user name token consumer, you can specify the com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator class name on the Token generator panel for application level and the com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer class name in this field.

3. Optional: Select a part reference in the **Part reference** field. The part reference indicates the name of the security token that is defined in the deployment descriptor. For example, if you receive a username token in your request message, you might want to reference the token in the username token consumer.

   Note: On the application level, if you do not specify a security token in your deployment descriptor, the **Part reference** field is not displayed. If you define a security token called `user_tcon` in your deployment descriptor, `user_tcon` is displayed as an option in the **Part reference** field.

4. Optional: In the certificate path section of the panel, select a certificate store type and indicate the trust anchor and certificate store name, if necessary. These options and fields are necessary when you specify com.ibm.wsspi.wssecurity.token.X509TokenConsumer as the token consumer class name. The names of the trust anchor and the collection certificate store are created in the certificate path under your token consumer.

   Note: The com.ibm.wsspi.wssecurity.token.TokenConsumingComponent interface is not used with JAX-WS Web services. If you are using JAX-RPC Web services, this interface is still valid.

   You can select one of the following options:

   None If you select this option, the certificate path is not specified.

   Trust any
   If you select this option, any certificate is trusted. When the received token is consumed, the Application Server does not validate the certificate path.

   Dedicated signing information
   If you select this option, you can select a trust anchor and a certificate store configuration. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the trust anchor and the certificate store before setting the certificate path.

   Trust anchor
   A trust anchor specifies a list of key store configurations that contain trusted root certificates. These configurations are used to validate the certificate path of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trustworthiness of a certificate chain. You must create the keystore file using the keytool utility. The keytool utility is available using the QShell Interpreter.

   You can configure trust anchors for the application level by completing the following steps:

a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
b. Under Related Items, click **EJB Modules** or **Web Modules > *URI_name***.
c. Access the token consumer from the following bindings:
   - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
d. Under Additional properties, click **Trust anchors**.

Collection certificate store
A collection certificate store includes a list of untrusted, intermediary certificates and certificate revocation lists (CRLs). The collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens. You can configure the collection certificate store for the application level by completing the following steps:
a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
b. Under Related Items, click **EJB Modules** or **Web Modules > *URI_name***.
c. Access the token consumer from the following bindings:
   - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
d. Under Additional properties, click **Collection certificate store**.

5. Optional: Specify a trusted ID evaluator. The trusted ID evaluator is used to determine whether to trust the received ID. You can select one of the following options:

None If you select this option, the trusted ID evaluator is not specified.

Existing evaluator definition
If you select this option, you can select one of the configured trusted ID evaluators. For example, you can select the SampleTrustedIDEvaluator, which is provided by WebSphere Application Server as an example.

Binding evaluator definition
If you select this option, you can configure a new trusted ID evaluator by specifying a trusted ID evaluator name and class name.

Trusted ID evaluator name
Specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default bindings.

Trusted ID evaluator class name
Species the class name of the trusted ID evaluator. The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. The default TrustedIDEvaluator class is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. When you use this default TrustedIDEvaluator class, you must specify the name and value properties for the default trusted ID evaluator to create the trusted ID list for evaluation. To specify the name and value properties, complete the following steps:
a. Under Additional properties, click **Properties > New**.

b. Specify the trusted ID evaluator name in the **Property** field. You must specify the name in the form, `trustedId_n` where `_n` is an integer from 0 to n.

c. Specify the trusted ID in the **Value** field.

For example:

```
property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
property name="trustedId_1, value="user1"
```

If the distinguished name (DN) is used, the space is removed for comparison. See the programming model information in the documentation for an explanation of how to implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. For more information, see "Default implementations of the Web services security service provider programming interfaces" on page 114.

Note: Define the trusted ID evaluator on the server level instead of the application level. To define the trusted ID evaluator on the server level, complete the following steps:

a. Click **Servers > Server Types > WebSphere application servers > *server_name***.

b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

c. Under Additional properties, click **Trusted ID evaluators**.

d. Click **New** to define a new trusted ID evaluator.

The trusted ID evaluator configuration is available only for the token consumer on the server-side application level.

6. Optional: Select the **Verify nonce** option. This option indicates whether to verify a nonce in the user name token if it is specified for the token consumer. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Verify nonce** option is valid only when the incorporated token type is a user name token.
7. Optional: Select the **Verify timestamp** option. This option indicates whether to verify a time stamp in the user name token. The **Verify nonce** option is valid only when the incorporated token type is a user name token.
8. Specify the value type local name in the **Local name** field. This field specifies the local name of the value type for the consumed token. For a user name token and an X.509 certificate security token, WebSphere Application Server provides predefined local names for the value type.

Table 65. Uniform Resource Identifier (URI) and Local name combinations

URI	Local name	Description
A namespace URI is not applicable.	Specify `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` as the local name value.	Specifies the name of an X.509 certificate token
A namespace URI is not applicable.	Specify `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1` as the local name value.	Specifies the name of the X.509 certificates in a PKI path

Table 65. Uniform Resource Identifier (URI) and Local name combinations (continued)

URI	Local name	Description
A namespace URI is not applicable.	Specify `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7` as the local name value.	Specifies a list of X509 certificates and certificate revocation lists (CRL) in a PKCS#7
Specify `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` as the URI value.	Specify `LTPA` as the local name value.	Specifies a binary security token that contains an embedded Lightweight Third Party Authentication (LTPA) token.

9. Optional: Specify the value type URI in the **URI** field. This entry specifies the namespace URI of the value type for the consumed token.

   Note: If you specify the token consumer for a username token or an X.509 certificate security token, you do not need to specify a value type URI.

   If you want to specify another token, you must specify both the local name and the URI. For example, if you have an implementation of your own custom token, you can specify `CustomToken` in the **Local name** field and `http://www.ibm.com/custom`
10. Click **OK** and **Save** to save the configuration.
11. Click the name of your token consumer configuration.
12. Under Additional properties, click **JAAS configuration**. The Java Authentication and Authorization Service (JAAS) configuration specifies the name of the JAAS configuration that is defined in the JAAS login panel. The JAAS configuration specifies how the token logs in on the consumer side.
13. Select a JAAS configuration from the **JAAS configuration name** field. The field specifies the name of the JAAS system of application login configuration. You can specify additional JAAS system and application configurations by clicking **Global security**. Under Authentication, click **Java Authentication and Authorization Service** and click either **Application logins > New** or **System logins > New**. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module. WebSphere Application Server provides the following predefined JAAS configurations:

    ClientContainer
    This selection specifies the login configuration that is used by the client container applications. The configuration uses the CallbackHandler application programming interface (API) that is defined in the deployment descriptor for the client container. To modify this configuration, see the JAAS configuration panel for application logins.

    WSLogin
    This selection specifies whether all of the applications can use the WSLogin configuration to perform authentication for the security run time. To modify this configuration, see the JAAS configuration panel for application logins.

    DefaultPrincipalMapping
    This selection specifies the login configuration that is used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries. To modify this configuration, see the JAAS configuration panel for application logins.

    system.LTPA_WEB
    This selection processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSPs) files. To modify this configuration, see the JAAS configuration panel for system logins.

system.RMI_OUTBOUND
This selection processes RMI requests that are sent outbound to another server when the com.ibm.CSIOutboundPropagationEnabled property is `true`. This property is set in the CSIv2 authentication panel.

To access the panel, click **Security > Global security**. Under Authentication, click **RMI/IIOP security > CSIv2 Outbound authentication**. To set the com.ibm.CSIOutboundPropagationEnabled property, select **Security attribute propagation**. To modify this JAAS login configuration, see the JAAS - System logins panel.

system.wssecurity.X509BST
This selection verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.PKCS7
This selection verifies an X.509 certificate within a PKCS7 object that might include a certificate chain, a certificate revocation list, or both. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.PkiPath
This section verifies an X.509 certificate with a public key infrastructure (PKI) path. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.UsernameToken
This selection verifies the basic authentication (user name and password) data. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.IDAssertionUsernameToken
This selection supports the use of identity assertion in Versions 6 and later applications to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_INBOUND
This selection specifies the login configuration for inbound or consumer requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_OUTBOUND
This selection specifies the login configuration for outbound or generator requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

None With this selection, you do not specify a JAAS login configuration.

14. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the token consumer for the application level.

What to do next

You must specify a similar token generator configuration for the application level.

Request consumer (receiver) binding configuration settings:

Use this page to specify the binding configuration for the request consumer.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.

2. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security Properties, click **Web services: Server security bindings**.
5. Under Request consumer (receiver) binding, click **Edit custom**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

The security constraints or bindings are defined using the application assembly process before the application is installed.

This product provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the **Use defaults** option on this panel and use the default binding information for the server level. The default binding that is provided by this product is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

Digital signature security constraint (integrity)

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

Information type	Required or optional
Signing information	Required
Key information	Required
Token consumer	Required
Key locators	Optional
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the key locators, collection certificate stores, and trust anchors that are defined at the server level.

Encryption constraint (confidentiality)

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

Information type	Required or optional
Encryption information	Required
Key information	Required
Token consumer	Required
Key locators	Optional
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the key locators, collection certificate store, and trust anchors that are defined at the server level.

Security token constraint

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

Information type	Required or optional
Token consumer	Required
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the collection certificate store and trust anchors that are defined at the server level.

Use defaults: **Version 6 and later applications**

Select this option if you want to use the default binding information from the server level.

If you select this option, the application server checks for binding information on the server level.

Port: **Version 6 and later applications**

Specifies the port in the Web service that is defined during application assembly.

Web service: **Version 6 and later applications**

Specifies the name of the Web service that is defined during application assembly.

Response consumer (receiver) binding configuration settings:

Use this page to specify the binding configuration for the response consumer.

To view this administrative console page, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security Properties, click **Web services: Client security bindings**.
5. Under Response consumer (receiver) binding, click **Edit custom**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

The security constraints or bindings are defined using the application assembly process before the application is installed.

This product provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the **Use defaults** option on this panel and use the default binding information for the server level. The default binding that is provided by this product is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

Digital signature security constraint (integrity)

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

Information type	Required or optional
Signing information	Required
Key information	Required
Token consumer	Optional
Key locators	Optional
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the key locators, collection certificate stores, and trust anchors that are defined at the server level.

Encryption constraint (confidentiality)

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

Information type	Required or optional
Encryption information	Required
Key information	Required
Token consumer	Optional
Key locators	Optional
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the key locators, collection certificate store, and trust anchors that are defined at the application or server level.

Security token constraint

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

Information type	Required or optional
Token consumer	Required
Collection certificate store	Optional
Trust anchors	Optional
Properties	Optional

You can use the collection certificate store and trust anchors that are defined at the application or server level.

Use defaults: **Version 6 and later applications**

Select this option if you want to use the default binding information from the server level.

If you select this option, the application server checks for binding information on the server level.

Component: **Version 6 and later applications**

Specifies the enterprise bean in an assembled Enterprise JavaBeans (EJB) module.

Port: **Version 6 and later applications**

Specifies the port in the Web service that is defined during application assembly.

Web service: **Version 6 and later applications**

Specifies the name of the Web service that is defined during application assembly.

Configuring token consumers using JAX-RPC to protect message authenticity at the server level:

Version 6 and later applications

The token consumer on the server level is used to specify the information that is needed to process the security token if it is not defined at the application level.

About this task

WebSphere Application Server provides default values for bindings. You must modify the defaults for a production environment.

Complete the following steps to configure the token consumers on the server level.

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Token consumers**.
3. Click **New** to create a token consumer configuration, click **Delete** to delete an existing configuration, or click the name of an existing token consumer configuration to edit its settings. If you are creating a new configuration, enter a unique name for the token consumer configuration in the **Token consumer name** field. For example, you might specify `sig_tcon`. This field specifies the name of the token consumer element.
4. Specify a class name in the Token consumer class name field. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side.

Note: The com.ibm.wsspi.wssecurity.token.TokenConsumingComponent interface is not used with JAX-WS Web services. If you are using JAX-RPC Web services, this interface is still valid.

The token consumer class name must be similar to the token generator class name.

For example, if your application requires an X.509 certificate token consumer, you can specify the com.ibm.wsspi.wssecurity.token.X509TokenGenerator class name on the Token generator panel and the com.ibm.wsspi.wssecurity.token.X509TokenConsumer class name in this field. WebSphere Application Server provides the following default token consumer class implementations:

com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer
This implementation integrates a user name token.

com.ibm.wsspi.wssecurity.token.X509TokenConsumer
This implementation integrates an X.509 certificate token.

com.ibm.wsspi.wssecurity.token.LTPATokenConsumer
This implementation integrates a Lightweight Third Party Authentication (LTPA) token.

com.ibm.wsspi.wssecurity.token.IDAssertionUsernameTokenConsumer
This implementation integrates an IDAssertionUsername token.

A corresponding token generator class does not exist for this implementation.

5. Select a certificate path option. The certificate path specifies the certificate revocation list (CRL) that is used for generating a security token wrapped in a PKCS#7 with a CRL. WebSphere Application Server provides the following certificate path options:

   None If you select this option, the certificate path is not specified.

   Trust any
   If you select this option, any certificate is trusted. When the received token is consumed, the certificate path validation is not processed.

   Dedicated signing information
   If you select this option, you can specify a trust anchor and a certificate store. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the collection certificate store before setting the certificate path. To define a collection certificate store on the server level, see “Configuring the collection certificate on the server level” on page 492.

   a. Select a trust anchor in the Trust anchor field. WebSphere Application Server provides two sample trust anchors. However, it is recommended that you configure your own trust anchors for a production environment. For information on configuring a trust anchor, see “Configuring trust anchors on the server level” on page 480.
   b. Select a collection certificate store in the Certificate store field. WebSphere Application Server provides a sample collection certificate store. If you select **None**, the collection certificate store is not specified. For information on specifying a list of certificate stores that contain untrusted, intermediary certificate files awaiting validation, see “Configuring trusted ID evaluators on the server level” on page 494.

6. Select a trusted ID evaluator from the Trusted ID evaluation reference field. This field specifies a reference to the Trusted ID evaluator class name that is defined in Trusted ID evaluators panel. The trusted ID evaluator is used for evaluating whether the received ID is trusted. If you select **None**, the trusted ID evaluator is not referenced in this token consumer configuration. To configure a trusted ID evaluator, see “Configuring trusted ID evaluators on the server level” on page 494.
7. Select the **Verify nonce** option if a nonce is included in a user name token on the generator side. Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Verify nonce** option is available if you specify a user name token for the token consumer and nonce is added to the user name token on the generator side.

8. Select the **Verify timestamp** option if a time stamp is included in the user name token on the generator side. The **Verify Timestamp** option is available if you specify a user name token for the token consumer and a time stamp is added to the user name token on the generator side.
9. Specify the local name of the value type for the integrated token. This entry specifies the local name of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as the key information type. To specify the key information type, see "Configuring the key information for the consumer binding using JAX-RPC on the server level" on page 436. WebSphere Application Server has predefined value type local names for the user name token and the X.509 certificate security token. Enter one of the following local names for the user name token and the X.509 certificate security token. When you specify the following local names, you do not need to specify the URI of the value type:

   Username token
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

   X.509 certificate token
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

   X.509 certificates in a PKIPath
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

   A list of X.509 certificates and CRLs in a PKCS#7
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

   Note: To specify Lightweight Third Party Authentication (LTPA) or token propagation (LTPA_PROPAGATION), you must specify both the value type local name and the Uniform Resource Identifier (URI). For LTPA, specify `LTPA` for the local name and `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the URI. For LTPA token propagation, specify `LTPA_PROPAGATION` for the local name and `http://www.ibm.com/websphere/appserver/tokentype` for the URI.

   For example, when an X.509 certificate token is specified, you can use `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` for the local name. When you specify the local name of another token, you must specify a value type Qname. For example: uri=`http://www.ibm.com/custom`, localName=`CustomToken`
10. Specify the value type uniform resource identifier (URI) in the URI field. This entry specifies the namespace URI of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as the key information type on the Key information panel for the default generator. When you specify the token consumer for the user name token or an X.509 certificate security token, you do not need to specify this option. If you specify another token, you need to specify the URI of the QName for the value type.
11. Click **OK** and then **Save** to save the configuration. After saving the token generator configuration, you can specify a JAAS configuration for your token consumer.
12. Click the name of your token generator configuration.
13. Under Additional properties, click **JAAS configuration**.
14. Select a JAAS configuration from the JAAS configuration name field.

    The field specifies the name of the JAAS system for application login configuration. You can specify additional JAAS system and application configurations by clicking **Security > Global security**. Expand Java Authentication and Authorization Service, then click **Application logins > New** or **System logins > New**. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module. WebSphere Application Server provides the following predefined JAAS configurations:

    ClientContainer
    This selection specifies the login configuration that is used by the client container

applications. The configuration uses the CallbackHandler application programming interface (API) that is defined in the deployment descriptor for the client container. To modify this configuration, see the JAAS configuration panel for application logins.

WSLogin
This selection specifies whether all of the applications can use the WSLogin configuration to perform authentication for the security run time. To modify this configuration, see the JAAS configuration panel for application logins.

DefaultPrincipalMapping
This selection specifies the login configuration that is used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries. To modify this configuration, see the JAAS configuration panel for application logins.

system.wssecurity.IDAssertion
This selection enables a Version 5.*x* application to use identity assertion to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.Signature
This selection enables a Version 5.*x* application to map a distinguished name (DN) in a signed certificate to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

system.LTPA_WEB
This selection processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSP) files. To modify this configuration, see the JAAS configuration panel for system logins.

system.WEB_INBOUND
This selection handles login requests for Web applications, which include servlets and JavaServer Pages (JSP) files. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

system.RMI_INBOUND
This selection handles logins for inbound Remote Method Invocation (RMI) requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

system.DEFAULT
This selection handles the logins for inbound requests that are made by internal authentications and most of the other protocols except Web applications and RMI requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

system.RMI_OUTBOUND
This selection processes RMI requests that are sent outbound to another server when the com.ibm.CSIOutboundPropagationEnabled property is `true`. This property is set in the CSIv2 authentication panel. To access the panel, click **Security > Global security**. Under Authentication, expand **RMI/IIOP security** and click **CSIv2 outbound authentication**. To set the com.ibm.CSIOutboundPropagationEnabled property, select **Security attribute propagation**. To modify this JAAS login configuration, see the JAAS - System logins panel.

system.wssecurity.X509BST
This section verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.PKCS7
This selection verifies an X.509 certificate with a certificate revocation list in a PKCS7 object. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.PkiPath
This section verifies an X.509 certificate with a public key infrastructure (PKI) path. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.UsernameToken
This selection verifies the basic authentication (user name and password) data. To modify this configuration, see the JAAS configuration panel for system logins.

system.wssecurity.IDAssertionUsernameToken
This selection enables Versions 6 and later applications to use identity assertion to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_INBOUND
This selection specifies the login configuration for inbound or consumer requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_OUTBOUND
This selection specifies the login configuration for outbound or generator requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

None With this selection, you do not specify a JAAS login configuration.

15. Click **OK** and then **Save** to save the configuration.

Results

You have configured the token consumer at the server level.

What to do next

You must specify a similar token generator configuration for the server level.

Token consumer collection:

Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Default Generator Bindings, click **Token consumers**.

To view this administrative console page for Version 6 and later applications on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name*** .

3. Under Web Services Security Properties, you can access the signing information for the following bindings:
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Token consumers**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Token consumers**.

Token consumer name: **Version 6 and later applications**

Specifies the name of the token consumer configuration.

For example, the default X509 token consumer names can be either `con_enctcon` for encrypting or `con_signtcon` for signing. Or a custom token consumer name might be `sig_tcon` for signing.

Token consumer class name: **Version 6 and later applications**

Specifies the name of the token consumer implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface.

Token consumer class name: **Version 6 and later applications**

Specifies the name of the token consumer implementation class.

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side.

Token consumer configuration settings:

Use this page to specify the information for the token consumer. The information is used at the consumer side only to process the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name*** .
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under JAX-RPC Default Consumer Bindings, click **Token consumers > *token_consumer_name*** or click **New** to create a new token consumer.

To view this administrative console page for Version 6 and later applications on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access the signing information for the following bindings:

- For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Token consumers**.
- For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Token consumers**.

4. Click **New** to specify a new configuration or click the name of an existing configuration to modify its settings.

Before specifying additional properties, specify a value in the Token consumer name, the Token consumer class name, and the Value type local name fields.

Token consumer name: **Version 6 and later applications**

Specifies the name of the token consumer configuration.

For example, the default X509 token consumer names are either `con_enctcon` for encrypting or `con_signtcon` for signing. Or a custom, the token consumer name might be `sig_tcon` for signing.

Token consumer class name: **Version 6 and later applications**

Specifies the name of the token consumer implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface.

Token consumer class name: **Version 6 and later applications**

Specifies the name of the token consumer implementation class.

The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side.

Part reference: **Version 6 and later applications**

Specifies a reference to the name of the security token that is defined in the deployment descriptor.

On the application level, when the security token is not specified in the deployment descriptor, the Part reference field is not displayed.

Certificate path: **Version 6 and later applications**

Specifies the trust anchor and the certificate store.

You can select the following options:

None If you select this option, the certificate path is not specified.

Trust any
If you select this option, any certificate is trusted. When the received token is incorporated, the certificate path validation is not processed.

Dedicated signing information
If you select this option, you can specify the trust anchor and the certificate store. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the collection certificate store before setting the certificate path.

Trust anchor

You can specify a trust anchor for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Trust anchors**.

Certificate store

You can specify a certificate path configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Collection certificate store**.

Trusted ID evaluator reference: **Version 6 and later applications**

Specifies the reference to the Trusted ID evaluator class name that is defined in the Trusted ID evaluators panel. The trusted ID evaluator is used for determining whether the received ID is trusted.

You can select the following options:

None If you select this option, the trusted ID evaluator is not specified.

Existing evaluator definition
If you select this option, you can select one of the configured trusted ID evaluators.

You can specify a certificate path configuration for the following bindings on the following levels:

Binding name	Server level or application level	Path
Default consumer binding	Server level	1. Click **Servers > Server Types > WebSphere application servers > *server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Trusted ID evaluators**.

Binding evaluator definition

If you select this option, you can specify a new trusted ID evaluator and its class name.

When you select a trusted ID evaluator reference, you must configure the trusted ID evaluators before setting the token consumer.

The Trusted ID evaluator field is displayed in the default binding configuration and the application server binding configuration.

Verify nonce: **Version 6 and later applications**

Specifies whether the nonce of the user name token is verified.

This option is displayed on the cell, server, and application levels. This option is valid only when the type of incorporated token is the user name token.

Verify timestamp: **Version 6 and later applications**

Specifies whether the time stamp of user name token is verified.

This option is displayed on the cell, server, and application levels. This option is valid only when the type of incorporated token is the user name token.

Value type local name: **Version 6 and later applications**

Specifies the local name of value type for the consumed token.

This product has predefined value type local names for the user name token and the X.509 certificate security token. Use the following local names for the user name token and the X.509 certificate security token. When you specify the following local names, you do not need to specify the Uniform Resource Identifier (URI) of the value type:

Username token

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

X509 certificate token
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

 # X509 certificates in a PKIPath
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

A list of X509 certificates and CRLs in a PKCS#7
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

Note: For Lightweight Third Party Authentication (LTPA), the value type local name is `LTPA`. If you enter LTPA for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well. For LTPA token propagation, the value type local name is `LTPA_PROPAGATION`. If you enter `LTPA_PROPAGATION` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype URI value in the Value type URI field as well.For the other predefined value types (Username token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the local name field begins with `http://`. For example, if you are specifying the username token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the value type local name field and then you do not need to enter a value in the value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the Quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

Value type URI: **Version 6 and later applications**

Specifies the namespace URI of the value type for the integrated token.

When you specify the token consumer for the user name token or the X.509 certificate security token, you do not need to specify this option. If you want to specify another token, specify the URI of the QName for the value type.

The application server provides the following predefined value type URIs:

- For the LTPA token: http://www.ibm.com/websphere/appserver/tokentype/5.0.2
- For the LTPA token propagation: http://www.ibm.com/websphere/appserver/tokentype

Configuring Web services security using JAX-RPC at the platform level

In the platform configuration, general properties and additional properties can be specified, and the default binding is included. You can configure security for Web services at a platform level with a variety of tasks including configuring key locators, trust anchors, and the collection certificate at the generator, consumer binding, and sever levels.

Before you begin

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based

annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

Besides the application-level constraints, there is a server-level Web services security (WSS) configuration called a *platform-level configuration*:

- These configurations are global for all applications and include some configurations only for WebSphere Application Server Version 5.*x* applications and some only for version 6.0.*x* applications.
- You can use the default binding as an application-level binding configuration so that applications do not have to define the binding in the application. There is only one set of default bindings that can be shared by multiple applications. This set is only available for WebSphere Application Server Version 6.*x* applications.

Therefore, binding configuration files can be specified at these levels: application and server. Each binding configuration overrides the next higher one. For any deployed application, the nearest configuration binding is applied. The visibility scope of the binding depends on where the file is located. If the binding is defined in an application, its visibility is scoped to that particular application. If it is located at the server level, the visibility scope is all applications that are deployed on that server.

About this task

To ensure Web services security at the platform level, you can configure:

- A nonce on the server level
- The key locator for the generator or consumer binding on the application level or at the server level
- Trust anchors for the generator or consumer binding on the application level or at the server level
- The collection certificate store for the generator or consumer binding on the application level or server level
- Trusted ID evaluators on the server level
- Hardware cryptographic devices for Web services security
- The `rrdSecurity.props` property file
- To configure a nonce on the server level, see the steps in “Configuring a nonce on the server level” on page 461
- To configure the key locator for the generator binding on the application level, see the steps in “Configuring the key locator using JAX-RPC for the generator binding on the application level” on page 463
- To configure the key locator for the consumer binding on the application level, see the steps in “Configuring the key locator using JAX-RPC for the consumer binding on the application level” on page 470
- To configure the key locator on the server level, see the steps in “Configuring the key locator using JAX-RPC on the server level” on page 472
- To configure trust anchors for the generator binding on the application level, see the steps in “Configuring trust anchors for the generator binding on the application level” on page 474
- To configure trust anchors for the consumer binding on the application level, see the steps in “Configuring trust anchors for the consumer binding on the application level” on page 478
- To configure trust anchors on the server level, see the steps in “Configuring trust anchors on the server level” on page 480
- To configure the collection certificate store for the generator binding on the application level, see the steps in “Configuring the collection certificate store for the generator binding on the application level” on page 481

- To configure the collection certificate store for the consumer binding on the application level, see the steps in “Configuring the collection certificate store for the consumer binding on the application level” on page 490
- To configure the collection certificate on the server level, see the steps in “Configuring the collection certificate on the server level” on page 492
- To configure trusted ID evaluators on the server level, see the steps in “Configuring trusted ID evaluators on the server level” on page 494
- To enable hardware cryptographic devices for Web services security, see the steps in “Enabling hardware cryptographic devices for Web Services Security” on page 499
- To work with the `rrdSecurity.props` file, see “rrdSecurity.props file” on page 497

Results

By completing these steps, you have configured Web services security at the platform level.

Configuring a nonce on the server level: Version 6 and later applications

You can configure nonce for the server by using the WebSphere Application Server administrative console.

About this task

Nonce is a randomly generated, cryptographic token that is used to prevent replay attacks of user name tokens that are used with SOAP messages. Typically, nonce is used with the user name token.

You can configure nonce at the application level and the server level. However, you must consider the order of precedence.

The following list shows the order of precedence:

1. Application level

   The application level settings for the nonce maximum age and nonce clock skew fields are specified through the additional properties.
2. Server level

If you configure nonce on the application level and the server level, the values that are specified for the application level take precedence over the values that are specified for the server level. Likewise, the values that are specified for the application level take precedence over the values specified for the server level. Complete the following steps to configure nonce on the server level:

Complete the following steps to configure a nonce on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Specify a value, in seconds, for the **Nonce cache timeout** field. The value that is specified for the **Nonce cache timeout** field indicates how long the nonce remains cached before it is discarded. You must specify a minimum of 300 seconds. However, if you do not specify a value, the default is 600 seconds. This field is optional on the server level.
3. Specify a value, in seconds, for the **Nonce maximum age** field. The value that is specified for the **Nonce maximum age** field indicates how long the nonce is valid. You must specify a minimum of 300

seconds, but the value cannot exceed the number of seconds that is specified for the **Nonce cache timeout** field. If you do not specify a value, the default is 300 seconds. This field is optional on the server level.

4. Specify a value, in seconds, for the **Nonce clock skew** field. The value that is specified for the **Nonce clock skew** field specifies the amount of time, in seconds, to consider when the message receiver checks the freshness of the value. Consider the following information when you set this value:
   - Difference in time between the message sender and the message receiver, if the clocks are not synchronized.
   - Time that is needed to encrypt and transmit the message.
   - Time that is needed to get through network congestion.

   At a minimum, you must specify 0 seconds in this field. However, the maximum value cannot exceed the number of seconds indicated in the Nonce maximum age field. If you do not specify a value, the default is 0 seconds. This field is optional on the server level.
5. Optional: For Network Deployment only, select **Distribute nonce caching**. This option enables you to distribute the caching for a nonce using a Data Replication Service (DRS). In previous releases of WebSphere Application Server, the nonce was cached locally. By selecting this option, the nonce is propagated to other servers in your environment. However, the nonce might be subject to a one-second delay in propagation and subject to any network congestion.
6. Restart the server. If you change the nonce cache timeout value and do not restart the server, the change is not recognized by the server.

Distributing nonce caching to servers in a cluster: Version 6 and later applications

Distributed nonce caching enables you to distribute the cache for a nonce to different servers in a cluster.

Before you begin

Before configuring distributed nonce caching, configure cache replication.

About this task

In previous releases of WebSphere Application Server, the nonce was cached locally. To use this feature, you must complete the following actions:

1. Verify that you created an appropriate domain setting when you form a cluster.
2. Verify that replication domain is properly secured. The nonce cache is crucial to the integrity of the nonce validation process. If the nonce cache is compromised, then you cannot trust the result of the validation process.
3. In the administrative console for the server level, select the **Distribute nonce caching** option. You can enable the option by completing the following steps:
   a. Click **Security > Web services**.
   b. Select the **Distribute nonce caching** option.
4. Restart the servers within your cluster.

Results

When you select the **Distribute nonce caching** option in the administrative console, the nonce is propagated to other servers in your environment. However, the nonce might be subject to a one-second delay in propagation and subject to any network congestion.

What to do next

For more information on distributed nonce caching, see “Web services security enhancements” on page 55.

Configuring the key locator using JAX-RPC for the generator binding on the application level:

Version 6 and later applications

The key locator information for the default generator specifies which key locator implementation is used to locate the key to be used for signature and encryption information. The key locator information for the generator specifies which key locator implementation is used to locate the key to be used for signature validation or encryption.

About this task

WebSphere Application Server provides default values for the bindings. However, you must modify the defaults for a production environment.

Complete the following steps to configure the key locator for the generator binding on the application level:

1. Locate the encryption information configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Key locators**.
   e. Click **New** to create a key locator configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing key locator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Key locator name** field. For example, you might specify `gen_keyloc`.
2. Specify a class name for the key locator class implementation in the **Key locator class name** field. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side. Specify a class name according to the requirements of the application. For example, if the application requires that the key is read from a keystore file, specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation. WebSphere Application Server supports the following default key locator class implementations for Versions 6.0.*x* and later applications that are available to use with the request generator or response generator:

   com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator
   This implementation locates and obtains the key from the specified keystore file.

   com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator
   This implementation uses the public key from the signer certificate and is used by the response generator.
3. Specify the keystore password, the keystore location, and the keystore type. Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys retrieved from the keystore are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the

com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a keystore password, location, and type.

a. Specify a password in the keystore **Password** field. This password is used to access the keystore file.
b. Specify the location of the keystore file in the keystore **Path** field.
c. Select a keystore type from the **Type** field. The Java Cryptography Extension (JCE) that is used by IBM supports the following keystore types:

   JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

   JCEKS
   Use this option if you are using Java Cryptography Extensions.

   JCERACFKS
   Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

   PKCS11KS (PKCS11)
   Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

   PKCS12KS (PKCS12)
   Use this option if your keystore uses the PKCS#12 file format.

   WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `Storepass` and the type is `JCEKS`.

   Note: Do not use the sample keystore files in a production environment. These samples are provided for testing purposes only.

4. Click **OK** and then click **Save** to save the configuration.
5. Under Additional properties, click **Keys**.
6. Click **New** to create a key configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. This entry specifies the name of the key object within the keystore file. If you are creating a new configuration, enter a unique name in the **Key name** field. For digital signatures, the key name is used by the request generator or the response generator signing information to determine which key is used to digitally sign the message.

   You must use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.

   Note: Do not use the sample key files in a production environment. These samples are provided for testing purposes only.
7. Specify an alias in the **Key alias** field. The key alias is used by the key locator to search for key objects in the keystore.
8. Specify a password in the **Key password** field. The password is used to access the key object within the keystore file.
9. Click **OK** and **Save** to save the configuration.

Results

You have configured the key locator for the generator binding at the application level.

What to do next

You must specify a similar key information configuration for the consumer.

Key locator collection:

Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.
4. **Version 5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**.

Note: The bindings for a version 5.x application has a link that says **Edit** and the bindings for a Version 6.0.x. or later application has a link that says **Edit custom**. This is quick reference to determine which application version that you are configuring.

Using this **Key locator collection** panel, complete the following steps:

1. Specify a key locator name and a key locator class name on the panel.

2. Save your changes by clicking **Save** in the messages section at the top of the administrative console. The administrative console home panel is displayed.
3. After saving your changes, update the Web services security run time with the default binding information by clicking **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.
4. After you define key locators, click the key locator name to specify additional properties and keys under **Additional Properties**.

Key locator name: **Version 5.x or 6 application**

Specifies the unique name of the key locator.

Key locator class name: **Version 5.x or 6 application**

Specifies the class name of the key locator, which retrieves the key that is used for digital signing and encryption.

Key locator configuration settings:

Use this page to specify the settings for a key locator configuration. The key locators retrieve keys from the keystore file for digital signature and encryption. This product enables you to plug in a custom key locator configuration.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.
4. Click **New** to create a new configuration or click the name of a configuration to modify its settings.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.

4. **Version 5.x application** Under Additional properties, you can access key locators for the following bindings:

- For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**.
- For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**.
- For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**.
- For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**.

5. Click **New** to create a new configuration or click the name of a configuration to modify its settings.

Key locator name: **Version 5.x or 6 application**

Specifies the name of the key locator.

Data type String

Key locator class name: **Version 5.x or 6 application**

Specifies the name for the key locator class implementation.

Key locators that are associated with Versions 6 and later applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. This product provides the following default key locator class implementations for Versions 6 and later applications:

com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator
This implementation locates and obtains the key from the specified keystore file.

com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator
This implementation uses the public key from the certificate of the signer. This class implementation is used by the response generator.

This property is for the JAX-RPC programming model only. To implement signer certificate encryption for the JAX-WS programming model, set a custom property on the callback handler for the encryption token consumer. For more information, read the topic Callback handler settings.

com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator
This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

Version 5.x application Key locators that are associated with Version 5.*x* applications must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface. This product provides the following default key locator class implementations for Version 5.*x* applications.

com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator
This implementation maps an authenticated identity to a key and is used by the response sender. If encryption is used, this class is used to locate a key to encrypt the response message. The com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator class can map an authenticated identity from the invocation credential of the current thread to a key that is used to encrypt the message. If an authenticated identity is present on the current thread, the class maps the ID to the mapped name. For example, user1 is mapped to mappedName_1. Otherwise, `name="default"`. When a matching key is not found, the authenticated identity is mapped to the default key that is specified in the binding file. This implementation supports the following formats: JKS, JCEKS, and PKCS12.

com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator
This implementation maps a name to an alias and is used by the response receiver, request sender, and request receiver. The encryption process uses this class to obtain a key to encrypt a message, and the digital signature process uses this class to obtain a key to sign a message. The com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator class maps a logical name to a key alias in the keystore file. For example, key #105115176771 is mapped to `CN=Alice, O=IBM, c=US`.

com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator
This implementation uses the signer certificate to encrypt the response. This class implementation is used by the response sender and response receiver.

Data type String

Key store password: **Version 5.x or 6 application**

Specifies the password that is used to access the keystore file.

Key store configuration name: **Version 6 and later applications**

Specifies the name of the key store configuration that is defined in the keystore settings in secure communications.

Key store path: **Version 5.x or 6 application**

Specifies the location of the keystore file.

Key store type: **Version 5.x or 6 application**

Specifies the type of keystore file.

JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

JCEKS
Use this option if you are using Java Cryptography Extensions.

PKCS11KS (PKCS11)
Use this format if your keystore file uses the PKCS#11 file format. Keystores files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

PKCS12KS (PKCS12)
Use this option if your keystore file uses the PKCS#12 file format.

Default	JKS
Range	JKS, JCEKS, PKCS11KS (PKCS11), PKCS12KS (PKCS12)

Web services security property collection:

Use this page to a view a list of additional properties for the configuration.

You can view a Web services security property collection panel in several ways. Complete the following steps to view one of these administrative console pages:

1. Click **Security > JAX-WS and JAX-RPC security runtime**.
2. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Properties**.
3. Click **New** to create a new property.
4. Click **Delete** to a delete a property that you specified previously.

Property name: **Version 5.x or 6 application**

Specifies the name of the property.

Property value: **Version 5.x or 6 application**

Specifies the value for the property.

Web services security property configuration settings:

Use this page to configure additional security properties.

You can view a Web services security property configuration settings panel in several ways. Complete the following steps to view one of these administrative console pages:

1. Click **Security > JAX-WS and JAX-RPC security runtime**.
2. Under JAX-RPC Default Generator Bindings or JAX-RPC Default Consumer Bindings, click **Properties > New**.

Property Name: **Version 5.x or 6 application**

Specifies the name of the property.

Data type: String

Property Value: **Version 5.x or 6 application**

Specifies the value for the property.

Data type: String

The following table lists the properties that you can configure by using the Web services security property panels.

Configuration panel name	Property name	Property value	Description
JAAS configuration	com.ibm.wsspi.wssecurity.token.X509.issuerName	Specify the SubjectDN or the IssuerDN of the issuer for the X.509 certificate.	This property is used to specify the issuer of the certificate in the token consumer component.
JAAS configuration	com.ibm.wsspi.wssecurity.token.X509.issuerSerial	Specify the serial number of the X.509 certificate.	This property is used to specify the serial number of the certificate in the token consumer component.
Key information	com.ibm.wsspi.wssecurity.keyinfo.EncodingNS	Specify the namespace Uniform Resource Identifier (URI) for the qualified name (QName).	This property is used to specify the namespace URI part of the QName that represents the encoding method.

Configuration panel name	Property name	Property value	Description
Properties	com.ibm.ws.wssecurity.handler.hardwareCacheEntryRefreshHours	Specify a numeric value from 1 to 24 that represents the number of hours that a temporary key is valid.	This property is used to specify the amount of time before a key is retranslated. Temporary keys outside the keystore typically expire in a short period of time, measured in days or hours. If the server is configured to use a hardware acceleration card, but not the hardware keystore, you can configure it to translate the temporary keys periodically before they expire. If this property is not set , a key will be retranslated after 8 hours. Setting this value to 0 disables retranslation.
Request generator and Response generator	com.ibm.wsspi.wssecurity.timestamp.SOAPHeaderElement	Specify `1` or `true`.	This property is used with the **Add nonce** option to set the mustUnderstand flag in the deployment descriptor.
Request generator and Response generator	com.ibm.wsspi.wssecurity.timestamp.dialect		
Signing information	com.ibm.wsspi.wssecurity.dsig.dumpPath	Specify the path used to locate the output file.	This property is used to specify an output file for dumping the target UTF-8 binary data before signing and verifying messages.
Token generator	com.ibm.wsspi.wssecurity.token.username.timestampExpires	Specify `1` or `true`.	This property is used to specify an expiration date for the user name token.
Transform algorithms	com.ibm.wsspi.wssecurity.dsig.XPathExpression	`not(ancestor-or-self::* [namespace-uri()='http://www.w3.org/2000 /09/xmldsig#' and local`	This property is used with this algorithm: `http://www.w3.org/TR/1999 /REC-xpath-19991116`

Configuring the key locator using JAX-RPC for the consumer binding on the application level:

Version 6 and later applications

The key locator information for the consumer at the application level specifies which key locator implementation is used. The key locator implementation locates the key to be used to validate the digital signature or the encryption information by the application.

About this task

Complete the following steps to configure the key locator for the consumer binding on the application level:

1. Locate the key locator configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties, you can access the key information for the request consumer and response consumer bindings.
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Additional properties, click **Key locators**.
   e. Click **New** to create a key locator configuration, click **Delete** and select the box next to the configuration to delete an existing configuration, or click the name of an existing key locator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Key locator name** field. For example, you might specify `klocator`.

2. Specify a name for the key locator class implementation. The Java Authentication and Authorization Service (JAAS) Login Module implementation is used to validate (authenticate) the security token on the consumer side. Specify a class name according to the requirements of the application. For example, if the application requires that the key is read from a keystore file, specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation. WebSphere Application Server provides the following default key locator class implementations for Version 6.0.*x* applications that are available to use with the request consumer or response consumer:

   com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator
   This implementation locates and obtains the key from the specified keystore file.

   com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator
   This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

3. Specify the keystore password, the keystore location, and the keystore type. Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys that are retrieved from the keystore files are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a keystore password, location, and type.
   a. Specify a password in the keystore **Password** field. This password is used to access the keystore file.
   b. Specify the location of the keystore file in the keystore **Path** field.
   c. Select a keystore type from the keystore **Type** field. The Java Cryptography Extension (JCE) that is used by IBM supports the following keystore types:

      JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      JCEKS
      Use this option if you are using Java Cryptography Extensions.

      PKCS11KS (PKCS11)
      Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

      PKCS12KS (PKCS12)
      Use this option if your keystore uses the PKCS#12 file format.

      WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `Storepass` and the type is `JCEKS`.

      Note: Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

4. Click **OK** and **Save** to save the configuration.
5. Under Additional properties, click **Keys**.
6. Click **New** to create a key configuration, click **Delete** and select the box next to the configuration to delete an existing configuration, or click the name of an existing key configuration to edit its settings. This entry specifies the name of the key object within the keystore file. If you are creating a new configuration, enter a unique name in the **Key name** field.

   It is recommended that you use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.

7. Specify an alias in the **Key alias** field. The key alias is used by the key locator to search for key objects in the keystore file.

8. Specify a password in the **Key password** field. The password is used to access the key object within the keystore file.
9. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the key locator for the consumer binding at the application level.

What to do next

You must specify a similar key information configuration for the generator.

Configuring the key locator using JAX-RPC on the server level:

Version 6 and later applications

The key locator information for the default generator bindings specifies which key locator implementation is used to locate the key for signature and encryption information if these bindings are not defined at the application level.

About this task

The key locator information for the default consumer bindings specifies which key locator implementation is used to locate the key that is used for signature validation or decryption if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, you must modify the defaults for a production environment.

Complete the following steps to configure the key locator on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Key locator**. You can configure the key locator configurations for both the default generator and the default consumer in this location.
3. Click one of the following to work with the key locator configurations:

   New To create a key locator configuration. Enter a unique name for the key locator configuration in the **Key locator name** field. For example, you might specify `sig_klocator`.

   Delete To delete an existing configuration

   an existing key locator configuration
   To edit the settings of an existing configuration.
4. Specify a name for the key locator class implementation in the **Key locator class name** field. The key locators that are associated with Version 6.0.*x* applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface.

   Note: This interface is valid only for JAX-RPC applications. For JAX-WS applications, the Java Authentication and Authorization Service (JAAS) Login Module implementation is used to create the security token on the generator side and to validate (authenticate) the security token on the consumer side.

   WebSphere Application Server provides the following default key locator class implementations for Version 6.0.*x* applications:

com.ibm.wsspi.wssecurity.keyinfo.KeyStoreLeyLocator
This implementation locates and obtains the key from a specified keystore file.

com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator
This implementation uses the public key from the certificate of the signer. This class implementation is used by the response generator.

com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator
This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

For example, you might specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreLeyLocator implementation if you need the configuration to be the key locator for signing information.

5. Specify the keystore password, the keystore location, and the keystore type. Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys that are retrieved from the keystore file are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a key store password, location, and type.
   a. Specify a password in the **Key store password** field. This password is used to access the keystore file.
   b. Specify the location of the keystore file in the **Key store path** field.
   c. Select a keystore type from the **Key store type** field. The Java Cryptography Extension (JCE) that is used supports the following key store types:

      JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      JCEKS
      Use this option if you are using Java Cryptography Extensions.

      PKCS11
      Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

      PKCS12
      Use this option if your keystore file uses the PKCS#12 file format.

      WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `storepass` and the type is JCEKS.

      Note: Do not use these keystore files in a production environment. These samples are provided for testing purposes only.
6. Click **OK** and **Save** to save the configuration.
7. Under Additional properties, click **Keys**.
8. Click one of the following to work with the key configurations:

   New To create a key configuration. Enter a unique name in the **Key name** field. You must use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.

   Delete To delete an existing configuration.

   an existing key configuration
   To edit the settings of the existing configuration.

   This entry specifies the name of the key object within the keystore file.

9. Specify an alias in the **Key alias** field. The key alias is used by the key locator to search for key objects in the keystore file.
10. Specify a password in the **Key password** field. The password is used to access the key object within the keystore file.
11. Click **OK** and then click **Save** to save the configuration.

Results

You have configured the key locator for the server level.

What to do next

Configure the key information for the default generator and the default consumer bindings that reference this key locator.

Configuring trust anchors for the generator binding on the application level:

Version 6 and later applications

A *trust anchor* specifies key stores that contain trusted root certificates, which validate the signer certificate. These key stores are used by the request generator and the response generator (when Web services is acting as client) to generate the signer certificate for the digital signature. You can configure trust anchors for the generator binding at the application level by using the administrative console.

Before you begin

You can configure a trust anchor using an assembly tool or the administrative console. This task describes how to configure the application-level trust anchor using the administrative console. For more information on assembly tools, see the related information.

About this task

The keystores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration that is specified for the request generator must match the binding configuration for the response generator.

The trust anchor configuration for the request generator on the client must match the configuration for the request consumer on the server. Also, the trust anchor configuration for the response generator on the server must match the configuration for the response consumer on the client.

Complete the following steps to configure trust anchors for the generator binding on the application level:

1. Locate the trust anchor panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the trust anchor configuration for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Trust anchors**.

e. Click **New** to create a trust anchor configuration, click **Delete** to delete an existing configuration, or click the name of an existing trust anchor configuration to edit its settings. If you are creating a new configuration, enter a unique name in the **Trust anchor name** field.

2. Specify the keystore password, the keystore location, and the keystore type. Key store files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys retrieved from the keystore are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a key store password, location, and type.
   a. Specify a password in the **Key store password** field. This password is used to access the keystore file.
   b. Specify the location of the key store file in the **Key store path** field.
   c. Select a keystore type from the **Key store type** field. The Java Cryptography Extension (JCE) used by IBM supports the following key store types:

      JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      JCEKS
      Use this option if you are using Java Cryptography Extensions.

      PKCS11KS (PKCS11)
      Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

      PKCS12KS (PKCS12)
      Use this option if your keystore uses the PKCS#12 file format.

      WebSphere Application Server provides some sample keystore files in the following directory, using the *USER_INSTALL_ROOT* variable:

      For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `Storepass` and the type is JCEKS.

      Note: Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

Results

This task configures trust anchors for the generator binding at the application level.

What to do next

You must specify a similar trust anchor configuration for the consumer.

Trust anchor collection:

Use this page to view a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trust of a certificate chain.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To create the keystore file, use the keytool utility. The keytool utility is available using the QShell Interpreter.

To view this administrative console page for trust anchors on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.

To view this administrative console page for trust anchors on the application level,

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. **Version 6 and later applications** Under Web Services Security Properties, you can access trust anchors information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. **Version 5.x application** Under Additional properties, you can access the trust anchors information for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
5. Under Additional properties, click **Trust anchors**.

If you click **Update runtime**, the Web services security run time is updated with the default binding information, which is contained in the `ws-security.xml` file that was previously saved. If you make changes on this panel, you must complete the following steps:

1. Save your changes by clicking **Save** at the top of the administrative console. When you click **Save**, you are returned to the administrative console home panel.
2. Return to the Trust anchors collection panel and click **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.

Trust anchor name: **Version 5.x or 6 application**

Specifies the unique name that is used to identify the trust anchor.

Key store path: **Version 5.x or 6 application**

Specifies the location of the keystore file that contains the trust anchors.

Key store type: **Version 5.x or 6 application**

Specifies the type of keystore file.

The value for this field is **JKS**, **JCEKS**, **JCERACFKS** (z/OS only), **JCE4758RACFKS** (z/OS only), **PKCS11KS (PKCS11)**, or **PKCS12KS (PKCS12)**.

Trust anchor configuration settings:

Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page for trust anchors on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.
4. Click **New** to create a trust anchor or click the name of an existing configuration to modify its settings.

To view this administrative console page for trust anchors on the application level,

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access trust anchors information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. **Version 5.x application** Under Additional properties, you can access the trust anchors information for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
5. Under Additional properties, click **Trust anchors**.
6. Click **New** to create a trust anchor or click the name of an existing configuration to modify its settings.

Trust anchor name: **Version 5.x or 6 application**

Specifies the unique name that is used by the application binding to reference a predefined trust anchor definition in the default binding.

Key store configuration name:

Specifies the name of the key store configuration defined in the keystore settings in secure communications.

Key store password: **Version 5.x application**

Specifies the password that is needed to access the key store file.

Key store path: **Version 5.x or 6 application**

Specifies the location of the keystore file.

Use ${*USER_INSTALL_ROOT*} as this path expands to the WebSphere Application Server path on your machine.

Key store type: **Version 5.x or 6 application**

Specifies the type of keystore file.

Choose from the following options:

Version 5.x or 6 application **JKS**

Use this option if you are not using Java Cryptography Extensions (JCE).

Version 5.x or 6 application **JCEKS**

Use this option if you are using Java Cryptography Extensions.

Version 6 and later applications **PKCS11KS (PKCS11)**

Use this format if your keystore uses the PKCS#11 file format. Keystores that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

Version 6 and later applications **PKCS12KS (PKCS12)**

Use this option if your keystore uses the PKCS#12 file format.

Default	JKS
Range	JKS, JCEKS, PKCS11KS (PKCS11), PKCS12KS (PKCS12)

Configuring trust anchors for the consumer binding on the application level:

Version 6 and later applications

You can configure trust anchors for the consumer binding at the application level.

About this task

This article does not describe how to configure trust anchors at the server or cell level. Trust anchors that are defined at the application level have a higher precedence over trust anchors that are defined at the server or cell level. For more information on creating and configuring trust anchors on the server or cell level, see “Configuring trust anchors on the server level” on page 480.

You can configure a trust anchor at the application level using an assembly tool or the administrative console. This article describes how to configure the application-level trust anchor using the administrative console.

A trust anchor specifies key stores that contain trusted root Certificate Authority (CA) certificates, which validate the signer certificate. These keystores are used by the request consumer (as defined in the `ibm-webservices-bnd.xmi` file) and the response consumer (as defined in the `ibm-webservicesclient-bnd.xmi` file when a Web service is acting as a client) to validate the X.509 certificate in the SOAP message. The keystores are critical to the integrity of the digital signature validation. If the keystores are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request consumer in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response consumer in the `ibm-webservicesclient-bnd.xmi` file. The trust anchor configuration for the request consumer on the server side must match the request generator configuration on the client side. Also, the trust anchor configuration for the response consumer on the client side must match the response generator configuration on the server side.

Complete the following steps to configure trust anchors for the consumer binding on the application level:

1. Locate the trust anchor panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties you can access the trust anchor configuration for the following bindings:
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Additional properties, click **Trust anchors**.
   e. Click one of the following to work with trust anchor configuration:

      New To create a trust anchor configuration. Enter a unique name in the Trust anchor name field.

      Delete To delete the existing configuration selected in the box next to the configuration.

      an existing trust anchor configuration
      To edit the settings of an existing trust anchor configuration.
2. Specify the keystore password, the keystore location, and the keystore type. A trust anchor keystore file contains the trusted root Certificate Authority (CA) certificates that are used for validating the X.509 certificate that is used in digital signature or XML encryption.
   a. Specify a password in the Key store password field. This password is used to access the keystore file.
   b. Specify the location of the keystore file in the Key store path field.
   c. Select a keystore type from the Key store type field. The Java Cryptography Extension (JCE) that is used by IBM supports the following keystore types:

      JKS Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      JCEKS
      Use this option if you are using Java Cryptography Extensions.

      PKCS11KS (PKCS11)
      Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

PKCS12KS (PKCS12)
Use this option if your keystore file uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the following directory, using the *USER_INSTALL_ROOT* variable:

For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `storepass` and the type is JCEKS.

Note: Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

Results

You have configured trust anchors for the consumer binding at the application level.

What to do next

You must specify a similar trust anchor information for the generator.

Configuring trust anchors on the server level: **Version 6 and later applications**

You can configure a list of keystore objects that contain trusted root certificates to be used for certificate path validation of incoming X.509-formatted security tokens.

Before you begin

Prior to completing the steps to configure trust anchors, you must create the keystore file using the keytool utility. The keytool utility is available using the QShell Interpreter.

About this task

This task provides the steps that are needed to configure a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath application programming interface (API) to determine whether to trust a certificate chain.

Complete the following steps to configure the trust anchors on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Trust anchors**.
3. Click one of the following to work with trust anchor configuration:

   New To create a trust anchor configuration. Enter a unique name for the trust anchor in the Trust anchor name field.

   Delete To delete an existing configuration.

   an existing trust anchor configuration
   To edit the settings for an existing trust anchor.
4. Specify a password in the Key store password field that is used to access the keystore file.

5. Specify the absolute location of the keystore file in the **Key store path** field. It is recommended that you use the *USER_INSTALL_ROOT* variable as a portion of the keystore path. To change this predefined variable, click **Environment > WebSphere variables**. The *USER_INSTALL_ROOT* variable might display on the second page of variables.
6. Specify the type of keystore file in the key store type field. WebSphere Application Server supports the following keystore types:

   JKS Use this option if you are not using Java Cryptography Extensions (JCE) and your keystore file uses the Java Key Store (JKS) format.

   JCEKS
   Use this option if you are using Java Cryptography Extensions.

   PKCS11KS (PKCS11)
   Use this option if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

   PKCS12KS (PKCS12)
   Use this option if your keystore file uses the PKCS#12 file format.
7. Click **OK** and **Save** to save your configuration.

Results

You have configured trust anchors at the server level.

Configuring the collection certificate store for the generator binding on the application level:

Version 6 and later applications

You can configure a collection certificate for the generator bindings on the application level.

About this task

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check for a valid signature in a digitally signed SOAP message.

Complete the following steps to configure a collection certificate for the generator bindings on the application level:

1. Locate the collection certificate store configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security Properties, you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Collection certificate store**.
2. Specify the Certificate store name. Click **New** to create a collection certificate store configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, .or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.

The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the application level, the store name must be unique to the application level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.

3. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the *profile_root*`/properties/java.security` file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.
4. Click **OK** and **Save** to save the configuration.
5. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.
6. Under Additional properties, click **Certificate revocation lists**.
7. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists (CRL). This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/mycertstore/mycrl1. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendation for using certificate revocation lists:
   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
8. Click **OK** and **Save** to save the configuration.
9. Return to the collection certificate store configuration panel. To access the panel, complete the following steps:
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Manage modules, click ***URI_name***.
   c. Under Web Services Security properties, you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Collection certificate store > *certificate_store_name***.
10. Under Additional properties, click **X.509 certificates**.

11. Click **New** to create a X.509 certificate configuration, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
12. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificate. The collection certificate store is used to validate the certificate path of incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of path name. For example, you might type: *USER_INSTALL_ROOT*/etc/ws-security/samples/intca2.cer. Do not use this certificate path for production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

    Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.
13. Click **OK** and then **Save** to save your configuration.

Results

You have configured the collection certificate store for the generator binding.

What to do next

You must specify a similar collection certificate store configuration for the consumer.

Collection certificate store collection:

Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

The following list provides recommendations for using CRLs:

- If CRLs are added to the collection certificate store collection, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_name***.

3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **Version 5.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.

Complete the following steps:

1. Click **New** to specify a new certificate store name and certificate store provider.
2. Click **OK** and messages display at the top of the administrative console panel.
3. Within the messages at the top of the administrative console panel, click **Save**.
4. Return to the collection certificate store collection panel and click **Update runtime** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file. When you click **Update runtime**, the configuration changes made to the other Web services are also updated in the Web services security run time.

Certificate store name: **Version 5.x or 6 application**

Specifies the name of the certificate store.

Certificate store provider: **Version 5.x or 6 application**

Specifies the provider of the certificate store.

Collection certificate store configuration settings:

Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.

4. Specify a new collection certificate store by clicking **New** or by clicking the collection certificate store name to modify its settings.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **Version 5.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request receiver binding click **Edit > Collection certificate store**.
   - For the Response receiver binding, click **Edit > Collection certificate store**.
5. Specify a new collection certificate store by clicking **New** or by clicking the collection certificate store name to modify its settings.

After configuring a collection certificate store, you can select the new configuration under Certificate store on the token generator and token consumer panels. To access these panels, complete the following steps:

1. Click **Security > JAX-WS and JAX-RPC security runtime**.
2. Under JAX-RPC Default Generator Bindings, click **Token generators** or under JAX-RPC Default Consumer Bindings, click **Token consumers**.
3. Click **New** to create a new token generator or token consumer, or click the name of an existing configuration to make modifications.

After you configure your collection certificate store on this panel, you must click **Apply** before configuring either the certificate revocation list or an X.509 certificate. The certificate revocation list configuration is not available for version 5.*x* applications through the administrative console. After you configure your certificate revocation list or X.509 certificate, complete the following steps:

1. Click **Save**, at the top of the administrative console panel, which returns you to the list of the configured collection certificate stores.
2. Click **Update runtime** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file.

Certificate store name: **Version 5.x or 6 application**

Specifies the name for the certificate store.

The name of the collection certificate store must be unique in the scope. For example, the name must be unique at the server level. The name specified in **Certificate store name** field is used by other configurations to refer to a pre-defined collection certificate store. For example, the application binding refers to a collection certificate store that is defined on the server level. The application server looks up the collection certificate store based on proximity. For example, if *cert1* is defined as the name of the

certificate store on the cell and server levels and *cert1* is referenced in the application binding, the application server uses the server-level collection certificate store.

Certificate Store Provider: **Version 5.x or 6 application**

Specifies the provider for the certificate store implementation.

This product supports the IBMCertPath certificate path provider. If you need to use another certificate path provider, define the provider implementation in the provider list within the `java.security` file in the Software Development Kit (SDK).

Data type	String
Default	IBMCertPath

X.509 certificates collection:

Use this page to view a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **X.509 certificates**.

To view this administrative console page for an X.509 certificate on the application level, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **Version 5.x application** Under Additional properties, you can access the collection certificate stores for the following bindings.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.

- For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Collection certificate store**.

5. Click the name of a configured collection certificate store or create a new collection certificate store first.
6. Under Additional properties, click **X.509 certificates**.

X.509 certificate path: **Version 5.x or 6 application**

Specifies the location of the X.509 certificate.

X.509 certificate configuration settings:

Use this page to specify a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **X.509 certificates**.
6. Specify a new X.509 certificate path by clicking **New** or by clicking the X.509 certificate path to modify its settings.

To view this administrative console page for an X.509 certificate on the application level, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **Version 5.x application** Under Additional properties, you can access the collection certificate stores for the following bindings.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Collection certificate store**.

5. Click the name of a configured collection certificate store or create a new collection certificate store first.
6. Under Additional properties, click **X.509 certificates**.
7. Specify a new X.509 certificate path by clicking **New** or click the X.509 certificate path to modify its settings.

X.509 Certificate Path: **Version 5.x or 6 application**

Specifies the absolute path to the location of the X.509 certificate.

As shown in the following example, you can use the USER_INSTALL_ROOT variable as part of the path name: {*USER_INSTALL_ROOT*}/etc/ws-security/samples/intca2.cer. This X.509 certificate path is not for production use. Obtain your own X.509 from a certificate authority before putting your application server environment into production.

You can configure the USER_INSTALL_ROOT variable in the administrative console by clicking **Environment** → **WebSphere Variables**.

Certificate revocation list collection:

Use this page to determine the location of the certificate revocation list (CRL) known to the application server. The Application Server checks the CRL to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

View the administrative console panel for the collection certificate store on the server level.

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation list**.

Version 6 and later applications View the administrative console page for the collection certificate store on the application level.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.

- For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.

4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation list**.
6. Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
7. Under Additional properties, click **Collection certificate store > *certificate_store_name***.
8. Under Additional properties, click **X.509 certificates**.
9. Click **New** and specify the path to the certificate revocation list.

Version 5.x application

Add a certificate revocation list for a version 5.*x* application.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Related items, click **Manage modules > *URI_name***.

Certificate revocation list path:

Version 6 and later applications

Specifies the location where you can find the list of certificates that are not valid.

Certificate revocation list configuration settings:

Use this page to specify a list of certificate revocations that check the validity of a certificate. The application server checks the certificate revocation lists (CRL) to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists > New** to specify the path to a new list or click the name of a certificate revocation list to modify its path.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.

- For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
- For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
- For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.

4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists > New** to specify the path to a new list or click the name of a certificate revocation list to modify its path.

Certificate revocation list path: **Version 6 and later applications**

Specifies a fully qualified path to the location where you can find the list of certificates that are not valid.

For portability reasons, it is recommended that you use application server variables to specify a relative path to the certificate revocation list. This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as *$USER_INSTALL_ROOT/mycertstore/mycrl* where *mycertstore* represents the name of your certificate store and *mycrl* represents the certificate revocation list. For a list of the supported variables, click **Environment > WebSphere variables** in the administrative console.

The following list provides recommendations for using CRLs:

- If CRLs are added to the collection certificate store collection, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

Configuring the collection certificate store for the consumer binding on the application level:

Version 6 and later applications

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check for a valid signature in a digitally signed SOAP message.

About this task

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs) that can be used to check for a valid signature in a digitally signed SOAP message. Complete the following steps to configure a collection certificate for the consumer bindings on the application level:

1. Locate the collection certificate store configuration panel in the administrative console.
   a. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
   b. Under Modules, click **Manage modules > *URI_name***.

c. Under Web services security properties, you can access the collection certificate store information for the response consumer and request consumer bindings.
   - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.

d. Under Additional properties, click **Collection certificate store**.

2. Click **New** to create a collection certificate store configuration, click **Delete** to delete an existing configuration, or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.

   The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the application level, the store name must be unique to the application level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

   For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.

3. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the *profile_root*`/properties/java.security` file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.
4. Click **OK** and **Save** to save the configuration.
5. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.
6. Under Additional properties, click **Certificate revocation lists**.
7. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists (CRL). This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/mycertstore/mycrl1. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendation for using certificate revocation lists:
   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
8. Click **OK** and **Save** to save the configuration.
9. Return to the Collection certificate store configuration panel. See the first few steps of this article to locate the collection certificate store panel.
10. Under Additional properties, click **X.509 certificates**.

11. Click **New** to create a new configuration for X.509 certificates, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
12. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificates. The collection certificate store is used to validate the certificate path of incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of the path name. For example, you might type: *USER_INSTALL_ROOT*`/etc/ws-security/samples/intca2.cer`. Do not use this certificate path for production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

    Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.
13. Click **OK** and then **Save** to save your configuration.

Results

You have configured the collection certificate store for the consumer binding.

What to do next

You must configure a token consumer configuration that references this certificate store configuration.

Configuring the collection certificate on the server level:

Version 6 and later applications

Collection certificate stores contain untrusted, intermediary certificate files awaiting validation. You can configure the collection certificate store on the server level and the cell level.

About this task

Validation might consist of checking for a valid signature in a digitally signed SOAP message to see if the certificate is on a certificate revocation list (CRLs), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

Complete the following steps to configure a collection certificate store on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Collection certificate store**.
3. Click **New** to create a collection certificate store configuration, click **Delete** to delete an existing configuration, or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field. For example, you might name your certificate store `sig_certstore`.

   The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the server level, the store name must be unique to the server level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.

4. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the *profile_root*`/properties/java.security` file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.
5. Click **OK** and **Save** to save the configuration.
6. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.
7. Under Additional properties, click **Certificate revocation lists**. For the generator binding, a certificate revocation list (CRL) is used when it is included in a generated security token. For example, a security token might be wrapped in a PKCS#7 format with a CRL. For more information on certificate revocation lists, see "Certificate revocation list" on page 124.
8. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. WebSphere Application Server uses the certificate revocation list to check the validity of the sender certificate.

   For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists. This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment.

   For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/*mycertstore*/*mycrl1* where *mycertstore* represents the name of your certificate store and *mycrl1* represents the certificate revocation list. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendations for using certificate revocation lists:

   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
9. Click **OK** and then **Save** to save the configuration.
10. Return to the Collection certificate store configuration panel.
11. Under Additional properties, click **X.509 certificates**. The X.509 certificate configuration specifies intermediate certificate files that are used for certificate path validation of incoming X.509-formatted security tokens.
12. Click **New** to create an X.509 certificate configuration, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
13. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificate. The collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of path name. For example, you might type: $*USER_INSTALL_ROOT*`/etc/ws-security/samples/intca2.cer`. Do not use this certificate path for

production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.

14. Click **OK** and then **Save** to save your configuration.
15. Return to the Collection certificate store collection panel and click **Update run time** to update the Web services security run time with the default binding information, which is located in the `ws-security.xml` file. When you click **Update run time**, the configuration changes made to other Web services are also updated in the run time for Web services security. Policy sets can only be used with JAX-WS applications. Policy sets cannot be used for JAX-RPC applications.

Results

You have configured the collection certificate store for the server level.

Configuring trusted ID evaluators on the server level: **Version 6 and later applications**

You can configure trusted identity (ID) evaluators. The trusted ID evaluator determines whether or not to trust the identity-asserting authority.

About this task

This task provides the steps that are needed to configure trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, the WebSphere Application Server issues the proper credentials based on the identity, which are used in a downstream call to another server for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

Complete the following steps to configure the trusted ID evaluators on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Trusted ID evaluators**.
3. Click **New** to create a trusted ID evaluator configuration, click **Delete** to delete an existing configuration, or click the name of an existing configuration to edit the settings. If you are creating a new configuration, enter a unique name for the trusted ID evaluator configuration in the Trusted ID evaluator name field. This field specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default binding.
4. Specify a class name in the Trusted ID evaluator class name field. The default class name is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator class. When you use the default TrustedIDEvaluator class, you must specify the name and value properties for the default trusted ID evaluator to create the trusted ID list for evaluation.
5. Under Additional properties, click **Properties > New**.
6. Specify the trusted ID evaluator name as a property name. You must specify the trusted ID evaluator name in the form, `trustedId_n`, where *_n* is an integer from zero (0) to n.
7. Specify the trusted ID as a property value.

```
property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
property name="trustedId_1, value="user1"
```

If a distinguished name (DN) is used, the space is removed for comparison.

8. Click **OK** and then **Save**.

Results

You have configured the trusted ID evaluators at the server level.

Trusted ID evaluator collection:

Use this page to view a list of trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, the application server issues the proper credentials based on the identity, which are used in a downstream call for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page for trusted ID evaluators on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.
4. Click **New** to create a trusted ID evaluator or click **Delete** to a delete a trusted ID evaluator.

To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage Modules > *URI_name***.
3. Under Web Service Security Properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit**.
5. Click **Trusted ID evaluators**.
6. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

Note: Trusted ID evaluators are only required for the request receiver (Version 5.x applications) and the request consumer (Version 6.0.x applications), if identity assertion is configured.

Using this trusted ID evaluator collection panel, complete the following steps:

1. Specify a trusted ID evaluator name and a trusted ID evaluator class name.
2. Save your changes by clicking **Save** in the messages section at the top of the administrative console.
3. Click **Update run time** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file. The configuration changes made to the other Web services also are updated in the Web services security run time.

Trusted ID evaluator name: **Version 6 and later applications**

Specifies the unique name of the trusted ID evaluator.

Trusted ID evaluator class name: **Version 6 and later applications**

Specifies the class name of the trusted ID evaluator.

Trusted ID evaluator configuration settings:

Use this information to configure trust identity (ID) evaluators.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

To view this administrative console page for trusted ID evaluators on the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.
4. Click **New** to create a trusted ID evaluator or click the name of an existing configuration to modify the settings.

Version 6 and later applications To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit**.
5. Click **Trusted ID evaluators**.
6. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

Note: Trusted ID evaluators are only required for the request receiver (Version 5.x applications) and the request consumer (Version 6.x applications), if identity assertion is configured.

You can specify one of the following options:

None Choose this option if you are not specifying a trusted ID evaluator.

Existing evaluator definition
Choose this option to specify a currently defined trusted ID evaluator.

Binding evaluator definition
Choose this option to specify a new trusted ID evaluator. A description of the required fields follows.

Trusted ID evaluator name: **Version 5.x or 6 application**

Specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default binding.

Trusted ID evaluator class name: **Version 5.x or 6 application**

Specifies the class name of the trusted ID evaluator.

The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. The default TrustedIDEvaluator class is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. When you use this default TrustedIDEvaluator class, you must specify the name and the value properties for the default trusted ID evaluator to create the trusted ID list for evaluation.

To specify the name and value properties, complete the following steps:

1. Under Additional properties, click **Properties > New**.
2. Specify the trusted ID evaluator name as a property name. You must specify the trusted ID evaluator name in the form, `trustedId_n`, where `_n` is an integer from zero (0) to n.
3. Specify the trusted ID as a property value.

For example:

```
property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
property name="trustedId_1", value="user1"
```

If a distinguished name (DN) is used, the space is removed for comparison.

Default com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl

See the programming model information in the documentation for an explanation of how to implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

rrdSecurity.props file:

Remote request dispatcher (RRD) supports LTPA and security attribute propagation for Web services security (WS-Security). You can enable token propagation in the *was_install*/profiles/*profileName*/properties/rrdSecurity.props file.

The `rrdSecurity.props` file contains comments to describe the security attributes.

The following is the format of the `rrdSecurity.props` file. The default values are in bold face type.

- `LTPAPropagation= ( True | False)`
- `SecurityAttributePropagation= ( True | False)`
- `SSLRequired= ( True | False)`

The WS-Security run time inspects the run as (invocation) subject and propagates the security tokens in the subject. The default setting is to only propagate the LTPA tokens.

Custom security tokens can be passed as attributes of the LTPA tokens. The security attribute propagation support uses the same pluggable JAAS login module as the CSIv2 support. The security attribute is not signed or encrypted, therefore, you should not send the attribute in clear text form. You must require SSL to ensure integrity and confidentiality. If SSL is not required, RRD uses the same scheme, such as HTTP or HTTPS, to make the Web services call that the original request used.

You must also configure the target Web service to validate the LTPA tokens and security attributes.

Developing Web services clients that retrieve tokens from the JAAS Subject in an application

Version 6 and later applications

The security handlers are responsible for propagating security tokens. These security tokens are embedded in the SOAP security header and passed to downstream servers.

About this task

This information applies only to Java API for XML-based RPC (JAX-RPC) Web services.

The security tokens are encapsulated in the implementation classes for the com.ibm.wsspi.wssecurity.auth.token.Token interface. You can retrieve the security token data from either a server application or a client application.

With a client application, the application serves as the request generator and the response consumer and runs as the Java Platform, Enterprise Edition (Java EE) client application. The consumer component for Web services security stores the security tokens that it receives in one of the properties of the MessageContext object for the current Web services call. You can retrieve a set of token objects through the javax.xml.rpc.Stub interface of that Web Services call. You must know which security tokens to retrieve and their token IDs in case multiple security tokens are included in the SOAP security header. Complete the following steps to retrieve the security token data from a client application:

1. Use the com.ibm.wsspi.wssecurity.token.tokenPropergation key string to obtain the Hashtable for the tokens through a property value in the javax.xml.rpc.Stub interface. The following example shows how to obtain the Hashtable:

   ```
   java.util.Hashtable t;
   javax.xml.rpc.Service serv = ...;
   MyWSPortType pt = (MyWSPortType)serv.getPort(MyWSPortType.class);
   t = (Hashtable)((javax.xml.rpc.Stub)pt)._getProperty(
   com.ibm.wsspi.wssecurity.Constants.WSSECURITY_TOKEN_PROPERGATION);
   ```

2. Search the targeting token objects in the Hashtable. Each token object in the Hashtable is set with its token ID as a key. You must have prior knowledge of the security token IDs to retrieve the security tokens. The following example shows how to retrieve a username token from the security header with a certain token ID value:

   ```
   com.ibm.wsspi.wssecurity.auth.token.UsernameToken unt;
   if (t != null) {
     unt = (com.ibm.wsspi.wssecuty.auth.token.UsernameToken)t.get("...");
   }
   ```

Results

After completing these steps, you have retrieved the security tokens from the JAAS Subject in a client application

Developing Web services applications that retrieve tokens from the JAAS Subject in a server application

Version 6 and later applications

With a server application, the application acts as the request consumer, and the response generator is deployed and runs in the Java Platform, Enterprise Edition (Java EE) container. The consumer component for Web services security stores the security tokens that it receives in the Java Authentication and Authorization Service (JAAS) Subject of the current thread. You can retrieve the security tokens from the JAAS Subject that is maintained as a local thread in the container.

About this task

This information applies only to Java API for XML-based RPC (JAX-RPC) Web services.

The security handlers are responsible for propagating security tokens. These security tokens are embedded in the SOAP security header and passed to downstream servers. The security tokens are encapsulated in the implementation classes for the com.ibm.wsspi.wssecurity.auth.token.Token interface. You can retrieve the security token data from either a server application or a client application.

Complete the following steps to retrieve the security token data from a server application:

1. Obtain the JAAS Subject of the current thread using the WSSubject utility class. If you enable Java 2 Security on the Global security panel in the administrative console, access to the JAAS Subject is denied if the application code is not granted the javax.security.auth.AuthPermission(″wssecurity.getCallerAsSubject″) permission. The following code sample shows how to obtain the JAAS subject:

   ```
   javax.security.auth.Subject subj;
   try {
   subj = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
   } catch (com.ibm.websphere.security.WSSecurityException e) {
     ...
   }
   ```

2. Obtain a set of private credentials from the Subject. For more information, see the application programming interface (API) com.ibm.websphere.security.auth.WSSubject class through the information center . To access this information within the information center, click **Reference > Developer > API Documentation > Application Programming Interfaces**. In the Application Programming Interfaces article, click **com.ibm.websphere.security.auth > WSSubject**.

   Note: When Java 2 Security is enabled, you might need to use the AccessController class to avoid a security violation that is caused by operating the security objects in the Java EE container.

   The following code sample shows how to set the AccessController class and obtain the private credentials:

   ```
   Set s = (Set) AccessController.doPrivileged(new PrivilegedAction() {
      public Object run() {
      return subj.getPrivateCredentials();
      }
         });
   ```

3. Search the targeting token class in the private credentials. You can search the targeting token class by using the java.util.Iterator interface. The following example shows how to retrieve a username token with a certain token ID value in the security header. You can also use other method calls to retrieve security tokens. For more information, see the application programming interface (API) documents for the com.ibm.wsspi.wssecurity.auth.token.Token interface or custom token classes.

   ```
   com.ibm.wsspi.wssecurity.auth.token.UsernameToken unt;
   Iterator it = s.iterator();
   while (it.hasNext()) {
     Object obj = it.next();
     if (obj != null &&
   obj instanceOf com.ibm.wsspi.wssecurity.auth.token.UsernameToken) {
       unt =(com.ibm.wsspi.wssecurity.auth.token.UsernameToken) obj;
   if (unt.getId().equals("...")) break;
   else continue;
     }
   }
   ```

Results

After completing these steps, you have retrieved the security tokens from the JAAS Subject in a server application

Enabling hardware cryptographic devices for Web Services Security

You can enable Web Services Security by using cryptographic hardware devices for both Web service clients and Web service providers that are running in the WebSphere Application Server environment. A

cryptographic token is a hardware or software device with a built-in keystore implementation. Cryptographic devices are used to manage certificates stored on the cryptographic tokens. These devices are also called *smartcards*. You enable hardware cryptographic devices for Web service security by either using keys that are stored in hardware devices or by using keys stored in a Java keystore file.

About this task

Web services security using cryptographic hardware devices is supported for both Web (JavaServer Pages (JSP) or servlet) and Enterprise JavaBeans (EJB) Web service clients. You can enable Web Services Security by using cryptographic hardware devices for both Web service clients and Web service providers that are running in the WebSphere Application Server environment.

There are two ways to enable hardware cryptographic devices for Web service security: use keys that are stored in hardware devices or use keys stored in a Java keystore file.

1. Determine whether to use keys that are stored in hardware devices or in a Java keystore file for the individual application.
2. Enable hardware cryptographic devices for Web service security by using one of the following two methods:
   - Enable cryptographic operations on hardware devices. See “Configuring hardware cryptographic devices for Web Services Security” for more details.
   - Enable cryptographic keys that are stored in hardware devices. See “Enabling cryptographic keys stored in hardware devices in Web Services Security” on page 501

   Note: Hardware cryptographic devices for Web Services Security are not supported on the Java Platform, Enterprise Edition (Java EE) Application Client on distributed platform.

Related concepts

“Hardware cryptographic device support for Web Services Security” on page 98
In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

Configuring hardware cryptographic devices for Web Services Security

Before you can use a hardware cryptographic device, you must configure and enable it. You must first configure a hardware cryptographic device using the Secure Sockets Layer (SSL) certificate and key management panels in the administrative console. The key for the cryptographic operation can be stored in an ordinary Java keystore file and need not be stored on the hardware devices.

Before you begin

You must first configure a hardware cryptographic device using the Secure Sockets Layer (SSL) certificate and key management panels in the administrative console.

1. In the administrative console, click **Servers** > **Server types** > **WebSphere application servers** and then select the server name.
2. Under **Security**, select **JAX-WS and JAX-RPC security runtime**.
3. Under Cryptographic Hardware, select **Enable cryptographic operations on hardware device** and then specify the name of the hardware cryptographic device configuration name. For more information, see Configuring a hardware cryptographic keystore.
4. Click **OK**.

Results

This procedure configures a hardware cryptographic device for all Web services security applications running on the WebSphere Application Server.

Related concepts

“Hardware cryptographic device support for Web Services Security” on page 98
In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

Related tasks

“Enabling cryptographic keys stored in hardware devices in Web Services Security”
You can enable individual Web service applications to use cryptographic keys stored in hardware devices in Web Services Security.

Configuring a hardware cryptographic keystore
You can create a hardware cryptographic keystore that WebSphere Application Server can use to provide cryptographic token support in the server configuration.

Enabling cryptographic keys stored in hardware devices in Web Services Security

You can enable individual Web service applications to use cryptographic keys stored in hardware devices in Web Services Security.

Before you begin

You must first configure the hardware acceleration device using the key management panels in the administrative console. See “Configuring hardware cryptographic devices for Web Services Security” on page 500

1. In the administrative console, click **Servers** > **Server types** > **WebSphere application servers** and then select the server name.
2. Under **Security**, click **JAX-WS and JAX-RPC security runtime**.
3. Under **Additional properties**, click **key locators**.
4. Select the key locator name.
5. Under **Key store**, specify the name of the keystore configuration.

   If the keystore reference is specified to a hardware device configuration, the Web Services Security runtime first attempts to obtain the cryptographic algorithm from the hardware device. If the hardware device is not supported or if it fails, the runtime for Web services security obtains the cryptographic algorithm from the security providers list. See Creating a keystore configuration for a preexisting keystore file for more information about how to create the name of a keystore configuration.
6. Click **OK**.

Results

If the name of the keystore reference is a Java keystore file, a hardware acceleration device that is configured at the application server level (`ws-security.xml`) will be used for cryptographic operations.

Related concepts

“Hardware cryptographic device support for Web Services Security” on page 98
In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

Related tasks

“Configuring hardware cryptographic devices for Web Services Security” on page 500
Before you can use a hardware cryptographic device, you must configure and enable it. You must first configure a hardware cryptographic device using the Secure Sockets Layer (SSL) certificate and key management panels in the administrative console. The key for the cryptographic operation can be stored in an ordinary Java keystore file and need not be stored on the hardware devices.

Creating a keystore configuration for a preexisting keystore file
A Secure Sockets Layer (SSL) configuration references keystore configurations during WebSphere Application Server runtime. Whether a keystore file was created by another keystore tool or saved from a previous configuration, the file must be referenced by a keystore configuration object to be used by the server. A keystore configuration object can be created to reference a pre-existing keystore object.

Securing Web services for Version 5.x applications based on WS-Security

Version 5.x application

Web services security for WebSphere Application Server is based on standards included in the Web Services Security (WS-Security) specification. These standards address how to provide protection for messages exchanged in a Web service environment.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

The specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Web services security is a message-level standard based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens.

About this task

Note: Use the deprecated ″Securing Apache SOAP Web services″ topics in the WebSphere Application Server, Version 5 documentation if you are still using Apache SOAP Version 2.3.

Note: WebSphere Application Server Version 5.*x* implements the JAX-RPC 1.1 standard. You can use policy sets only with Java API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

To secure Web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today’s business environment is the ability to inter-operate between formerly incompatible security technologies, such as public key infrastructure and Kerberos in heterogeneous environments like Microsoft .NET and environments that are based on the Java Platform,

Enterprise Edition (Java EE) standards. The complete Web services security protocol stack and technology roadmap is described in Security in a Web Services World: A Proposed Architecture and Roadmap.

Specification: Web Services Security (WS-Security) proposes a standard set of SOAP extensions that you can use to build secure Web services. These standards confirm integrity and confidentiality, which are generally provided with digital signature and encryption technologies. In addition, Web services security provides a general purpose mechanism for associating security tokens with messages. A typical example of the security token is a user name and password token, in which a user name and password are included as text. Web services security defines how to encode binary security tokens using methods such as X.509 certificates and Kerberos tickets.

To establish a secured environment and to enforce constraints for Web services security, you must perform a Java Naming and Directory Interface (JNDI) lookup on the client to resolve the service reference.

An administrator can use any of the following methods to integrate message-level security into a WebSphere Application Server environment:

- "Securing Web services for Version 5.x applications using XML digital signature" on page 540
- "Securing Web services for Version 5.x applications using XML encryption" on page 600
- "Securing Web services for Version 5.x applications using basic authentication" on page 621
- "Securing Web services for Version 5.x applications using identity assertion authentication" on page 630
- "Securing Web services for version 5.x applications using signature authentication" on page 636
- "Securing Web services for version 5.x applications using a pluggable token" on page 643

Example: Sample configuration for Web services security for a version 5.x application

WebSphere Application Server provides the following sample keystores for sample configurations. These sample keystores are for testing and sample purposes only. Do not use them a in production environment.

- `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks`
  - The keystore password is `client`
  - Trusted certificate with alias name, `soapca`
  - Personal certificate with alias name, `soaprequester` and key password `client` issued by intermediary certificate authority `Int CA2`, which is, in turn, issued by `soapca`
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks`
  - The keystore password is `server`
  - Trusted certificate with alias name, `soapca`
  - Personal certificate with alias name, `soapprovider` and key password `server`, issued by intermediary certificate authority `Int CA2`, which is, in turn, issued by `soapca`
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks`
  - The keystore password is `storepass`
  - Secret key `CN=Group1`, alias name `Group1`, and key password `keypass`
  - Public key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`
  - Private key `CN=Alice, O=IBM, C=US`, alias name `alice`, and key password `keypass`
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks`
  - The keystore password is `storepass`
  - Secret key `CN=Group1`, alias name `Group1`, and key password `keypass`
  - Private key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`
  - Public key `CN=Alice, O=IBM, C=US`, alias name `alice`, and key password `keypass`

- ${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
  - The intermediary certificate authority is Int CA2.

Default binding (cell and server level)

WebSphere Application Server provides the following default binding information:

Trust anchors
Used to validate the trust of the signer certificate.
- SampleClientTrustAnchor is used by the response receiver to validate the signer certificate.
- SampleServerTrustAnchor is used by the request receiver to validate the signer certificate.

Collection Certificate Store
Used to validate the certificate path.
- SampleCollectionCertStore is used by the response receiver and the request receiver to validate the signer certificate path.

Key Locators
Used to locate the key for signature, encryption, and decryption.
- SampleClientSignerKey is used by the requesting sender to sign the SOAP message. The signing key name is clientsignerkey, which can be referenced in the signing information as the signing key name.
- SampleServerSignerKey is used by the responding sender to sign the SOAP message. The signing key name is serversignerkey, which can be referenced in the signing information as the signing key name.
- SampleSenderEncryptionKeyLocator is used by the sender to encrypt the SOAP message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator.
- SampleReceiverEncryptionKeyLocator is used by the receiver to decrypt the encrypted SOAP message. The implementation is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). However, to use it for asymmetric encryption (RSA), you must add the private key CN=Bob, O=IBM, C=US, alias name bob, and key password keypass.
- SampleResponseSenderEncryptionKeyLocator is used by the response sender to encrypt the SOAP response message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks keystore and the com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator key locator. This key locator maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key alice, and you must change WebSphere Application Server to the appropriate user. The SampleResponseSenderEncryptionKeyLocator key locator also can set a default key for encryption. By default, this key locator is configured to use public key alice.

Trusted ID Evaluator
Used to establish trust before asserting to the identity in identity assertion.

SampleTrustedIDEvaluator is configured to use the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl implementation. The default implementation of com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator contains a list of trusted identities. The list is defined as properties with trustedId_* as the key and the value as the trusted identity. Define this information for the server level in the administration console by completing the following steps:

1. Click **Servers > Application Servers > *server1***.
2. Under Additional Properties, click **Web Services: Default bindings for Web Services Security > Trusted ID Evaluators > *SampleTrustedIDEvaluator***.

Login Mapping

Used to authenticate the incoming security token in the Web services security SOAP header of a SOAP message.

- The BasicAuth authentication method is used to authenticate user name security token (user name and password).
- The Signature authentication method is used to map a distinguished name (DN) into a WebSphere Application Server Java Authentication and Authorization Server (JAAS) Subject.
- The IDAssertion authentication method is used to map a trusted identity into a WebSphere Application Server JAAS Subject for identity assertion.
- The Lightweight Third Party Authentication (LTPA) authentication method is used to validate a LTPA security token.

The previous default bindings for trust anchors, collection certificate stores, and key locators are for testing or sample purpose only. Do not use them for production.

A sample configuration

The following examples demonstrate what IBM deployment descriptor extensions and bindings can do. The unnecessary information was removed from the examples to improve clarity. Do not copy and paste these examples into your application deployment descriptors or bindings. These examples serve as reference only and are not representative of the recommended configuration.

Use the following tools to create or edit IBM deployment descriptor extensions and bindings:

- Use an assembly tool to create or edit the IBM deployment descriptor extensions.
- Use an assembly tool or the administrative console to create or edit the bindings file.

The following example illustrates a scenario that:

- Signs the SOAP body, time stamp, and security token.
- Encrypts the body content and user name token.
- Sends the user name token (basic authentication data).
- Generates the time stamp for the request.

For the response, the SOAP body and time stamp are signed, the body content is encrypted, and the SOAP message freshness is checked using the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints.

The request sender and the request receiver are a pair. Similarly, the response sender and the response receiver are a pair.

Note: It is recommended that you use the WebSphere Application Server variables for specifying the path to the key stores. In the administrative console, click **Environment > Manage WebSphere Variables**. These variables often help with platform differences such as file system naming conventions. In the following examples, `$${USER_INSTALL_ROOT}` is used for specifying the path to the key stores.

Client-side IBM deployment descriptor extension

The client-side IBM deployment descriptor extension describes the following constraints:

Request Sender

- Signs the SOAP body, time stamp and security token
- Encrypts the body content and user name token
- Sends the basic authentication token (user name and password)

- Generates the time stamp to expire in three minutes

Response Receiver
- Verifies that the SOAP body and time stamp are signed
- Verifies that the SOAP body content is encrypted
- Verifies that the time stamp is present (also check for message freshness)

Example 1: Sample client IBM deployment descriptor extension

The `xmi:id` statements are removed for readability. These statements must be added for this example to work.

Note: In the following code sample, lines 2 through 4 were split into three lines due to the width of the printed page.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscext:WsClientExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:com.ibm.etools.webservice.wscext=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscext.xmi">
  <serviceRefs serviceRefLink="service/myServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <clientServiceConfig actorURI="myActorURI">
        <securityRequestSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </confidentiality>
          <loginConfig authMethod="BasicAuth"/>
          <addCreatedTimeStamp flag="true" expires="PT3M"/>
        </securityRequestSenderServiceConfig>
        <securityResponseReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
          </requiredIntegrity>
          <requiredConfidentiality>
            <confidentialParts part="bodycontent"/>
          </requiredConfidentiality>
          <addReceivedTimeStamp flag="true"/>
        </securityResponseReceiverServiceConfig>
      </clientServiceConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscext:WsClientExtension>
```

Client-side IBM extension bindings

Example 2 shows the client-side IBM extension binding for the security constraints described previously in the discussion on client-side IBM deployment descriptor extensions.

The signer key and encryption (decryption) key for the message can be obtained from the keystore key locator implementation (com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator). The signer key is used for encrypting the response. The sample is configured to use the Java Certification Path API to validate the certificate path of the signer of the digital signature. The user name token (basic authentication) data is collected from the standard in (stdin) prompts using one of the default JAAS implementations :javax.security.auth.callback.CallbackHandler implementation (com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler).

Example 2: Sample client IBM extension binding

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscbnd:ClientBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wscbnd=
  "http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscbnd.xmi">
  <serviceRefs serviceRefLink="service/MyServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <securityRequestSenderBindingConfig>
        <signingInfo>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <signingKey name="clientsignerkey" locatorRef="SampleClientSignerKey"/>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfo>
        <keyLocators name="SampleClientSignerKey" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "$${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
        </keyLocators>
        <encryptionInfo name="EncInfo1">
          <encryptionKey name="CN=Bob, O=IBM, C=US" locatorRef=
          "SampleSenderEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfo>
        <keyLocators name="SampleSenderEncryptionKeyLocator" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}LCswLTovPiws" path=
          "$${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks" type="JCEKS"/>
          <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
        </keyLocators>
        <loginBinding authMethod="BasicAuth" callbackHandler=
        "com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
      </securityRequestSenderBindingConfig>
      <securityResponseReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
            <trustAnchorRef ref="SampleClientTrustAnchor"/>
            <certStoreRef ref="SampleCollectionCertStore"/>
          </certPathSettings>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfos>
        <trustAnchors name="SampleClientTrustAnchor">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "$${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
        </trustAnchors>
        <certStoreList>
          <collectionCertStores provider="IBMCertPath" name="SampleCollectionCertStore">
            <x509Certificates path="$${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer"/>
          </collectionCertStores>
        </certStoreList>
        <encryptionInfos name="EncInfo2">
          <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfos>
        <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "$${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
        </keyLocators>
```

```
      </securityResponseReceiverBindingConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscbnd:ClientBinding>
```

Server-side IBM deployment descriptor extension

The client-side IBM deployment descriptor extension describes the following constraints:

Request Receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Verifies that the SOAP body, time stamp, and security token are signed.
- Verifies that the SOAP body content and user name token are encrypted.
- Verifies that the basic authentication token (user name and password) is in the Web services security SOAP header.
- Verifies that the time stamp is present (also check for message freshness). The freshness of the message indicates whether the message complies with predefined time constraints.

Response Sender (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Signs the SOAP body and time stamp
- Encrypts the SOAP body content
- Generates the time stamp to expire in 3 minutes

Example 3: Sample server IBM deployment descriptor extension

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsext:WsExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsext=
http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsext.xmi">
  <wsDescExt wsDescNameLink="MyServ">
    <pcBinding pcNameLink="Port1">
      <serverServiceConfig actorURI="myActorURI">
        <securityRequestReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </requiredIntegrity>
          <requiredConfidentiality">
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </requiredConfidentiality>
          <loginConfig>
            <authMethods text="BasicAuth"/>
          </loginConfig>
          <addReceivedTimestamp flag="true"/>
        </securityRequestReceiverServiceConfig>
        <securityResponseSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
          </confidentiality>
          <addCreatedTimestamp flag="true" expires="PT3M"/>
        </securityResponseSenderServiceConfig>
      </serverServiceConfig>
    </pcBinding>
  </wsDescExt>
</com.ibm.etools.webservice.wsext:WsExtension>
```

Server-side IBM extension bindings

The following binding information reuses some of the default binding information defined either at the server level or the cell level, which depends upon the installation. For example, request receiver is referencing the `SampleCollectionCertStore` certification store and the `SampleServerTrustAnchor` trust store is defined in the default binding. However, the encryption information in the request receiver is referencing a `SampleReceiverEncryptionKeyLocator` key locator defined in the application-level binding (the same `ibm-webservices-bnd.xmi` file). The response sender is configured to use the signer key of the digital signature of the request to encrypt the response using one of the default key locator (com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator) implementations.

Example 4: Sample server IBM extension binding

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsbnd:WSBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsbnd=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsbnd.xmi">
  <wsdescBindings wsDescNameLink="MyServ">
    <pcBindings pcNameLink="Port1" scope="Session">
      <securityRequestReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
            <trustAnchorRef ref="SampleServerTrustAnchor"/>
            <certStoreRef ref="SampleCollectionCertStore"/>
          </certPathSettings>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfos>
        <encryptionInfos name="EncInfo1">
          <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfos>
        <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
         "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}LCswLTovPiws" path="$${USER_INSTALL_ROOT}/
           etc/ws-security/samples/enc-receiver.jceks" type="JCEKS"/>
          <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
          <keys alias="bob" keypass="{xor}NDomLz4sLA==" name="CN=Bob, O=IBM, C=US"/>
        </keyLocators>
      </securityRequestReceiverBindingConfig>
      <securityResponseSenderBindingConfig>
        <signingInfo>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <signingKey name="serversignerkey" locatorRef="SampleServerSignerKey"/>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfo>
        <encryptionInfo  name="EncInfo2">
          <encryptionKey locatorRef="SignerKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfo>
        <keyLocators name="SignerKeyLocator" classname=
         "com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator"/>
      </securityResponseSenderBindingConfig>
    </pcBindings>
  </wsdescBindings>
  <routerModules transport="http" name="StockQuote.war"/>
</com.ibm.etools.webservice.wsbnd:WSBinding>
```

Web services security specification—a chronology

Version 5.x application

This chronology describes the process that has been used to develop the Web services security specifications. The chronology includes both the Organization for the Advancement of Structured Information Standards (OASIS) and non-OASIS activities.

Non-OASIS activities

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

In April 2002, IBM, Microsoft, and VeriSign proposed the *Web Services Security (WS-Security) specification* on their Web sites. This specification included the basic ideas of security token, XML signature, and XML encryption. The specification also defined the format for user name tokens and encoded binary security tokens. After some discussion and an inter-operability test that was based on the specification, the following issues were noted:

- The specification requires that the Web services security processors understand the schema correctly so that the processor distinguishes between the ID attribute for XML signature and XML encryption.
- The freshness of the message, which indicates whether the message complies with predefined time constraints, cannot be determined.
- Digested password strings do not strengthen security.

In August 2002, IBM, Microsoft, and VeriSign published the *Web Services Security Addendum*, which attempted to address the previously listed issues. The following solutions were put in the addendum:

- Require a global ID attribute for XML signature and XML encryption.
- Use time stamp header elements that indicate the time of the creation, receipt, or expiration of the message.
- Use password strings that are digested with a timestamp and nonce (randomly generated token).

OASIS activities

In June 2002, OASIS received a proposed Web services security specification from IBM, Microsoft, and Verisign. The Web Services Security Technical Committee (WSS TC) was organized at OASIS soon after the submission. The technical committee included many companies including IBM, Microsoft, VeriSign, Sun Microsystems, and BEA Systems.

In September 2002, WSS TC published its first specification, *Web Services Security Core Specification, Working Draft 01*. This specification included the contents of both the original Web services security specification and its addendum.

The coverage of the technical committee became larger as the discussion proceeded. Since the Web Services Security Core Specification allows arbitrary types of security tokens, proposals were published as profiles. The profiles described the method for embedding tokens, including Security Assertion Markup Language (SAML) tokens and Kerberos tokens imbedded into the Web services security messages. Subsequently, the definitions of the usage for user name tokens and X.509 binary security tokens, which were defined in the original Web Services Security Specification, were divided into the profiles.

WebSphere Application Server supports the following specifications:

- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)
- Web Services Security: Username Token Profile Draft 2

The following figure shows the various Web services security-related specifications. As indicated in the figure, the current support level for Web services security: SOAP message security is based on Draft 13 from May 2003. The current support level for Web services security user name token profiles, is based on Draft 2 from February 2003.

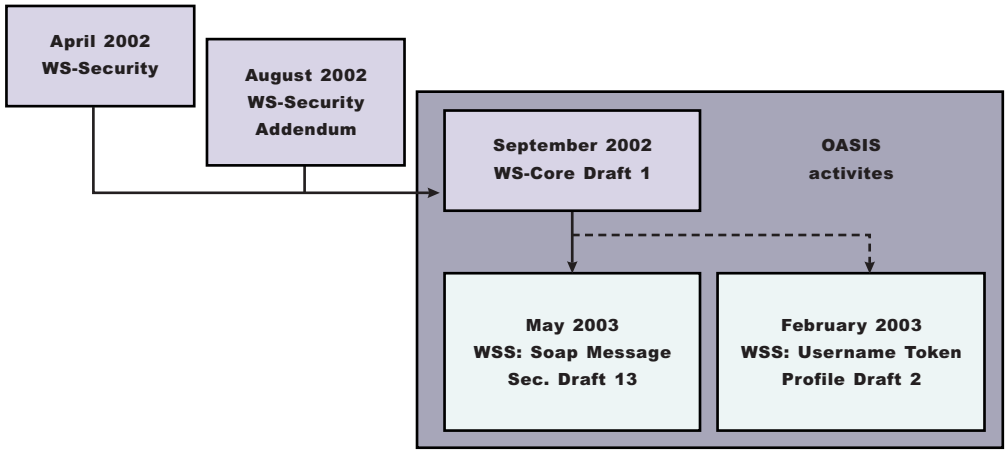


Figure 7. Web services security specification support

Web services security support

Version 5.x application

IBM supports Web services security, which is an extension of the IBM Web services engine, to provide a quality of service. The WebSphere Application Server security infrastructure fully integrates Web services security with the Java Platform, Enterprise Edition (Java EE) security specification.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server, Versions 4.*x*, 5, and 5.0.1 support digital signature for Apache SOAP Version 2.*x*. Beginning with WebSphere Application Server, Version 5.0.2, IBM supports Web services security. The IBM implementation is based on the Web services security specification, Web Services Security (WS-Security), originally proposed by IBM, Microsoft, and VeriSign in April 2002. Early versions of the proposed draft specification can be found in Web Services Security (WS-Security) Version 1.0 05 April 2002 and Web Services Security Addendum 18 August 2002. The WebSphere Application Server implementation is based on the Organization for the Advancement of Structured Information Standards (OASIS) working Draft 13 specification. (See the OASIS Web Services Security TC Web site for the latest working specification.) However, not all the features in the OASIS working Draft 13 specification are implemented.

Web services security is not supported in a pure Java client or a nonmanaged client. When a user ID and password are embedded in a request message, authentication is performed with the user ID and password. If authentication is successful, a user identity is established and further resource access is authorized based on that identity. After the user ID and password are authenticated by the Web services security run time, a Java EE container performs authorization.

WebSphere Application Server provides an implementation of the key features of Web services security based on the following specifications:

- Specification: Web Services Security (WS-Security) Version 1.0 05 April 2002
- Web Services Security Addendum 18 August 2002
- Web Services Security: SOAP Message Security Working 13 May 2003
- Web Services Security: Username Token Profile Draft

The following table provides a summary of Web services security elements supported by WebSphere Application Server:

Table 66. Web services security elements

Element	Notes
UsernameToken	Both the user name and password for the BasicAuth authentication method and the user name for the identity assertion authentication method are supported. WebSphere Application Server supports nonce, a randomly generated value.
BinarySecurityToken	X.509 certificates and Lightweight Third Party Authentication (LTPA) can be embedded, but there is no implementation to embed Kerberos tickets. However, the binary token generation and validation are pluggable and are based on the Java Authentication and Authorization Service (JAAS) Application Programming Interfaces (APIs). You can extend this implementation to generate and validate other types of binary security tokens.
Signature	The X.509 certificate is embedded as a binary security token and can be referenced by the SecurityTokenReference. WebSphere Application Server does not support shared, key-based signature.
Encryption	Both the EncryptedKey and ReferenceList XML tags are supported. KeyIdentifier specifies public keys and KeyName identifies the secret keys. WebSphere Application Server has the capability to map an authenticated identity to a key for encryption or use the signer certificate to encrypt the response message.
Time stamp	WebSphere Application Server supports the Created and Expires attributes. The freshness of the message, which indicates whether the message complies with predefined time constraints, is checked only if the Expires attribute is present in the message. WebSphere Application Server does not support the Received attribute, which is defined in the addendum. Instead, WebSphere Application Server uses the TimestampTraceReceived attribute, which is defined in the OASIS specification.
XML-based token	You can insert and validate an arbitrary format of XML tokens into a message. This format mechanism is based on the JAAS APIs.

Signing and encrypting attachments is not supported by WebSphere Application Server. However, WebSphere Application Server signs and encrypts the following elements for the request message.

Method	Element
XML digital signature	• Body • Securitytoken • Timestamp
XML encryption	• Bodycontent • Usernametoken

Method	Element
AuthMethod	• BasicAuth • IDAssertion (from WebSphere Application Server to another WebSphere Application Server • Signature • Lightweight Third Party Authentication (LTPA) on the server side • Other customer tokens

WebSphere Application Server signs and encrypts the following elements for the response message:

Method	Element
XML digital signature	• Body • Timestamp
XML encryption	• Bodycontent

The namespaces that are used for sending a message were published by OASIS in draft 13 (http://schemas.xmlsoap.org/ws/2003/06/secext). However, the Web services security run time in WebSphere Application Server can accept any of the following namespaces:

April 2002 specification
http://schemas.xmlsoap.org/ws/2002/04/secext

August 2002 addendum
http://schemas.xmlsoap.org/ws/2002/07/secext

http://schemas.xmlsoap.org/ws/2002/07/utility

OASIS draft published on draft 13 May 2003
http://schemas.xmlsoap.org/ws/2003/06/secext

http://schemas.xmlsoap.org/ws/2003/06/utility

Note: WebSphere Application Server only uses the previously mentioned two namespaces for sending out requests and responses. However, the product can process all other mentioned namespaces for incoming requests and responses.

WebSphere Application Server provides the following capabilities for Web services security:
- Integrity of the message
- Authenticity of the message
- Confidentiality of the message
- Privacy of the message
- Transport level security: provided by Secure Sockets Layer (SSL)
- Security token propagation (pluggable)
- Identity assertion

Refer to the Web services security elements table for a description of capabilities that are not supported.

Web services security and Java Platform, Enterprise Edition security relationship

Version 5.x application

This article describes the relationship between Web services security (message level security) model and the Java Platform, Enterprise Edition (Java EE) security model. It also includes information on Java EE role-based authorization checks.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server supports Java Specification Requests (JSR) 101 and JSR 109. JSRs 101 and 109 define Web services for the Java EE architecture so that you can develop and run Web services on the Java EE component architecture.

Note: *Web services security* refers to the Web services security: SOAP Message Security specification. For more information, see “Web services security support” on page 511.

Securing Web services with WebSphere Application Server security (Java EE role-based security)

You can secure Web services using the existing security infrastructure of WebSphere Application Server, Java EE role-based security, and Secure Sockets Layer (SSL) transport level security.

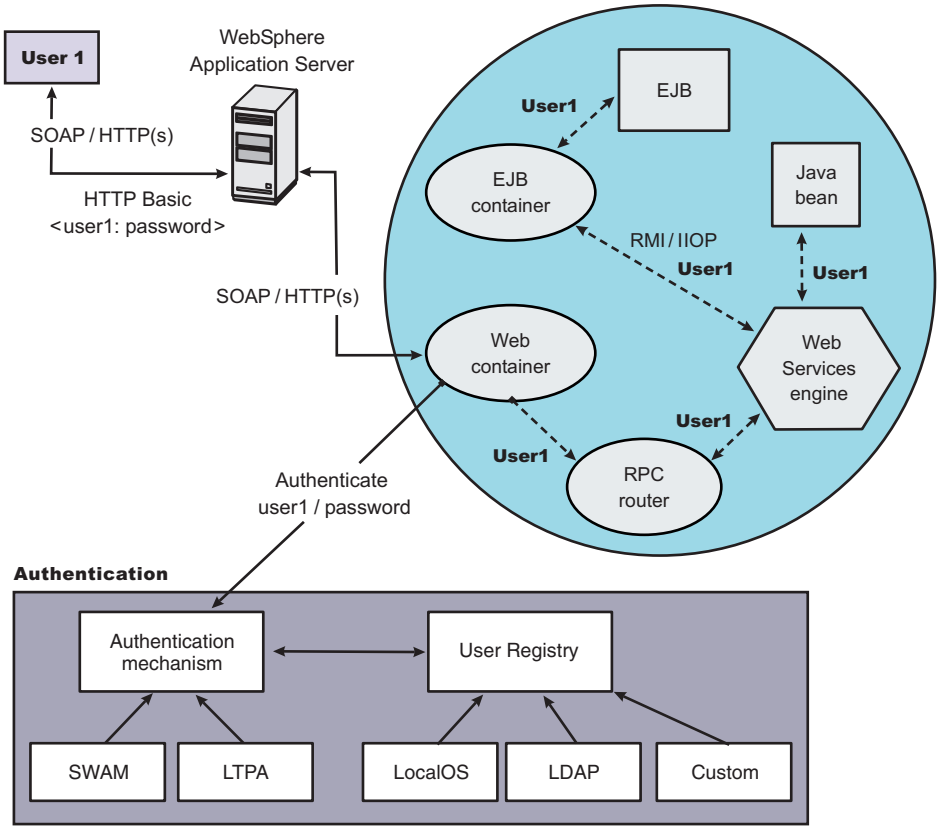


Figure 8. SOAP message flow using existing security infrastructure of WebSphere Application Server

The Web services port can be secured using Java EE role-based security. The Web services sender sends the basic authentication data using the HTTP header. SSL (HTTPS) can be used to secure the transport. When the WebSphere Application Server receives the SOAP message, the Web container authenticates the user (in this example, `user1`) and sets the security context for the call. After the security

context is set, the SOAP router servlet sends the request to the implementation of the Web services (the implementation can be JavaBeans or enterprise bean files). For enterprise bean implementations, the EJB container performs an authorization check against the identity of `user1`.

The Web services port also can be secured using the Java EE role. Then, authorization is performed by the Web container before the SOAP request is dispatched to the Web services implementation.

Securing Web services with Web services security at the message level

You can also secure Web services using Web services security at the message level. In this case, you can digitally sign or encrypt a certain part of the message. Web services security also supports security token propagation within the SOAP message. The following scenario assumes that the Web services port is not secured with Java EE role-based security and the enterprise bean is secured with Java EE role-based security.

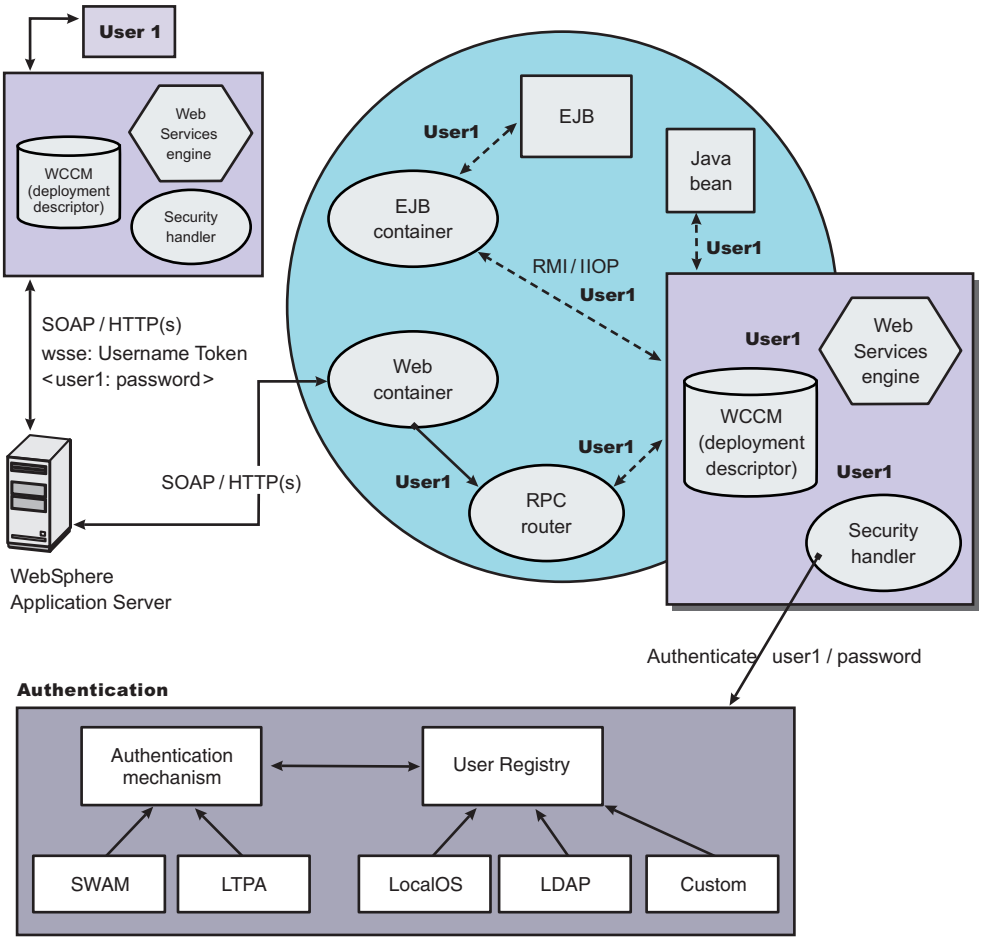


Figure 9. SOAP message flow using Web services security

In this case, the Web services port is not secured with Java EE role-based security. The Web services engine processes the SOAP message before the client sends the message to the Web services port. The Web services security run time acts on the security constraints, such as digitally signing, encrypting, or generating (and inserting) a security token in the SOAP header. In this case `<wsse:UsernameToken>` is generated using `user1` and the password. On the server-side (receiving), the Web services process the incoming message and Web services security enforces security constraints. This enforcement includes

making sure that messages are properly signed, properly encrypted, and decrypted, authenticating the security token, and setting up the security context with the authenticated identity. (In this case, `user1` is the authenticated identity.) Finally, the SOAP message is dispatched to the Web services implementation (if the implementation is an enterprise beans file, the Enterprise JavaBeans (EJB) container performs an authorization check against `user1`). SSL also might be used in this scenario.

Mixing the two

The second scenario shows that Web services security can complement Java EE role-based security. For example, SSL can be enabled at the transport level to provide a secure channel. Furthermore, if the Web services implementation is an enterprise beans file, you can leverage the EJB role-based authorization by performing authorization checks. Web services security run time leverages the security infrastructure to set the authenticated identity in the security context. The authenticated identity can be used in the downstream call to Java EE resources (or other resource types).

There are subtle consequences of combining the two scenarios. For example, if the HTTP transport is sending basic authentication data with `user1` and password in the HTTP header, but `<wsse:UsernameToken>` with `user99` and `letmein` also is inserted into the SOAP header. In the previous scenarios, there are two authentications performed. One authentication is performed by the Web container for authenticating `user1`, and the other is performed by Web services security for authenticating `user99`. The Web services security run time runs after the Web container runs and `user99` is the authenticated identity that is set in the security context.

Web services security can also propagate security tokens from the sender to the receiver for SOAP over a Java Message Service (JMS) transport.

Java EE role-based authorization checks

A standard for authorization does not exist for Web services. However, IBM, in conjunction with Microsoft Corporation, jointly published a security white paper road map for Web services called Security in a Web Services World: A Proposed Architecture and Roadmap in which a proposal exists for the WS-Authorization specification. However, the WS-Authorization specification has not been published.

The existing implementation of Web services security is based upon the Web Services for Java EE specification or the Java Specification Requirements (JSR) 109 specification. The implementation of Web services security leverages the Java EE role-based authorization checks. For conceptual information on role-based authorization, see Role-based authorization. If you develop a Web service that requires method-level authorization checks, then you must use stateless session beans to implement your Web service. For more information on using stateless session beans to implement a Web service, see Developing Web services applications with JAX-WS or Developing Web services applications with JAX-RPC depending on your programming model. Also see “Securing enterprise bean applications” on page 23. If you develop a Web service that is implemented as a servlet, you can use coarse-grained or URL-based authorization in the Web container. However, in this situation, you cannot use the identity from Web services security for authorization checks. Instead, you can use the identity from the transport. If you use SOAP over HTTP, then the identity is in the HTTP transport.

Web services security model in WebSphere Application Server

Version 5.x application

The Web services security model used by WebSphere Application Server is the declarative model. WebSphere Application Server does not include any application programming interfaces (APIs) for programmatically interacting with Web services security. However, a few Server Provider Interfaces (SPIs) are available for extending some security-related behaviors.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

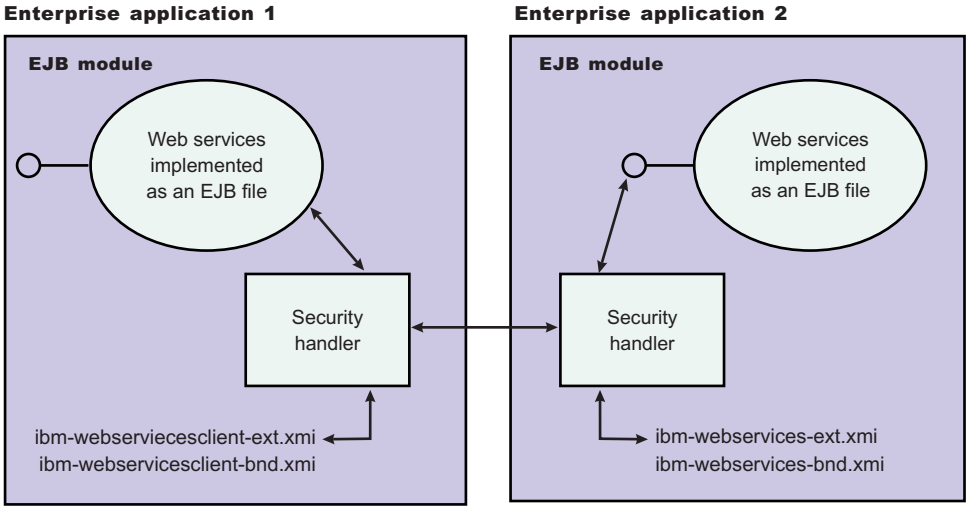


Figure 10. Web services security model

The security constraints for Web services security are specified in IBM deployment descriptor extensions for Web services. The Web services security run time acts on the constraints to enforce Web services security for the SOAP message. The scope of the IBM deployment descriptor extension is at the enterprise bean (EJB) or Web module level. Bindings are associated with each of the following IBM deployment descriptor extensions:

Client (Might be either a Java Platform, Enterprise Edition (Java EE) client (application client container) or Web services acting as a client)

`ibm-webservicesclient-ext.xmi`

`ibm-webservicesclient-bnd.xmi`

Server

`ibm-webservices-ext.xmi`

`ibm-webservices-bnd.xmi`

It is recommended that you use the assembly tools provided by IBM to create the IBM deployment descriptor extension and bindings. After the bindings are created, you can use the administrative console or an assembly tool to specify the bindings.

Note: The binding information is collected after application deployment rather than during application deployment. The alternative is to specify the required binding information before deploying your application.

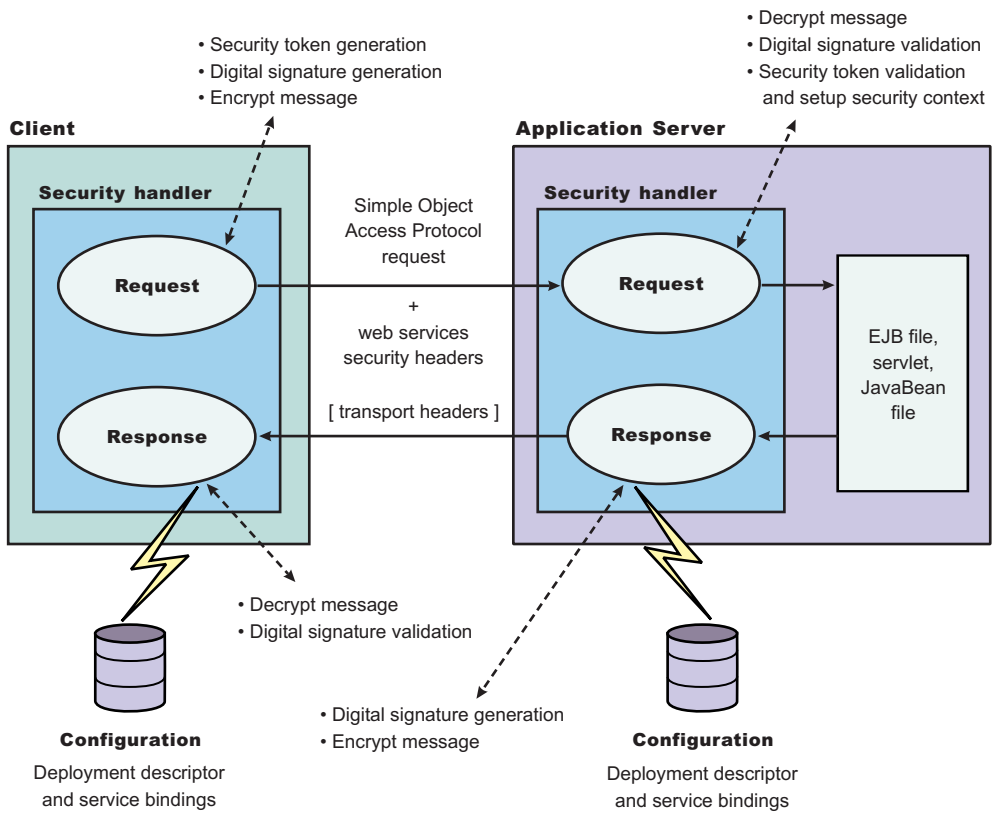


Figure 11. Web services security message interpretation

The Web services security run time enforces Web services security based on the defined security constraints in the deployment descriptor and binding files. Web services security has the following four points where it intercepts the message and acts on the security constraints defined:

Message points	Description
Request sender (defined in the `ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` files)	• Applies the appropriate security constraints to the SOAP message (such as signing or encryption) before the message is sent, generating the time stamp or the required security token.
Request receiver (defined in the `ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` files)	• Verifies that the Web services security constraints are met. • Verifies the freshness of the message based on the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints. • Verifies the required signature. • Verifies that the message is encrypted and decrypts the message if encrypted. • Validates the security tokens and sets up the security context for the downstream call.
Response sender (defined in the `ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` files)	• Applies the appropriate security constraints to the SOAP message response, like signing the message, encrypting the message, or generating the time stamp.

Message points	Description
Response receiver (defined in the `ibm-webservicesclient-ext.xmi` or `ibm-webservicesclient-bnd.xmi` files)	• Verifies that the Web services security constraints are met. • Verifies the freshness of the message based on the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints. • Verifies the required signature. • Verifies that the message is encrypted and decrypts the message, if encrypted.

Propagating security tokens

Version 5.x application

In this example, security tokens are propagated using Web services security, the security infrastructure of the WebSphere Application Server, and Java Platform, Enterprise Edition (Java EE) security.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

An example scenario

In this example, Client 1 invokes Web services 1. Then Web services 1 calls the Enterprise JavaBeans (EJB) file 2. The EJB file 2 calls Web services 3 and Web services 3 calls Web services 4.

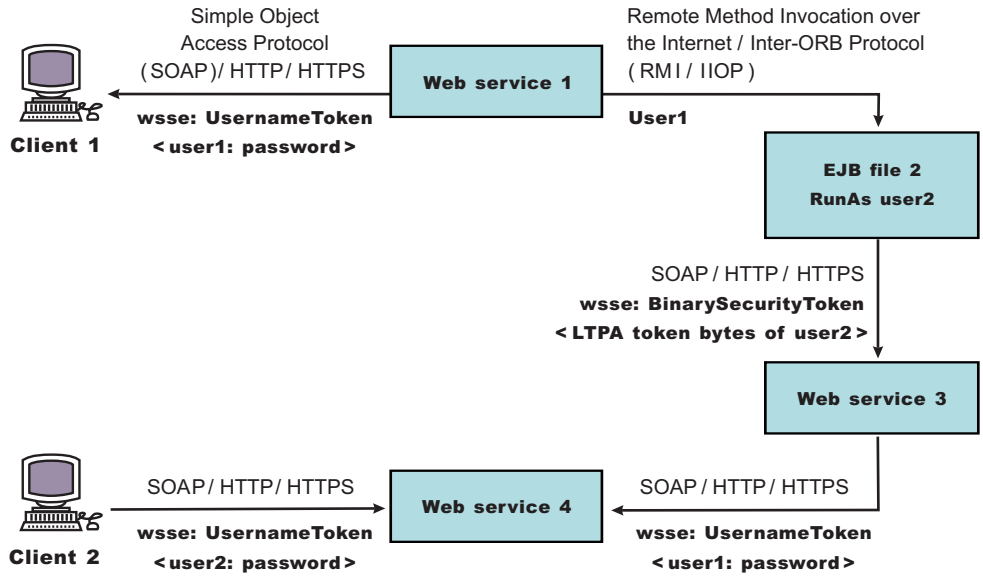


Figure 12. Propagating security tokens

The previous figure shows security tokens propagated using Web services security, the security infrastructure of the WebSphere Application Server, and Java Platform, Enterprise Edition (Java EE)

security. Web services 1 is configured to accept <wsse:UsernameToken> only and use the BasicAuth authentication method. However, Web services 4 is configured to accept either <wsse:UsernameToken> using the BasicAuth authentication method or Lightweight Third Party Authentication (LTPA) as <wsse:BinarySecurityToken>. The following steps describe the scenario shown in the previous figure:

1. Client 1 sends a SOAP message to Web services 1 with `user1` and `password` in the <wsse:UsernameToken> element.
2. The `user1` and `password` values are authenticated by the Web services security run time and set in the current security context as the Java Authentication and Authorization Service (JAAS) Subject.
3. Web services 1 invokes EJB file 2 using the Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) protocol.
4. The `user1` identity is propagated to the downstream call.
5. The EJB container of EJB file 2 performs an authorization check against `user1`.
6. EJB file 2 calls Web services 3 and Web services 3 is configured to accept LTPA tokens.
7. The RunAs role of EJB file 2 is set to `user2`.
8. The LTPA CallbackHandler implementation extracts the LTPA token from the current JAAS Subject in the security context and Web services security run time inserts the token as <wsse: BinarySecurityToken> in the SOAP header.
9. The Web services security run time in Web services 3 calls the JAAS login configuration to validate the LTPA token and set it in the current security context as the JAAS Subject.
10. Web services 3 is configured to send LTPA security to Web services 4. In this case, assume that the RunAs role is not configured for Web services 3. The LTPA token of `user2` is propagated to Web services 4.
11. Client 2 uses the <wsse:UsernameToken> element to propagate the basic authentication data to Web services 4.

Web services security complements the WebSphere Application Server security run time and the Java EE role-based security. This example demonstrates how to propagate security tokens across multiple resources such as Web services and EJB files.

Web services security constraints

Version 5.x application

The Web services security model that is used by WebSphere Application Server is the declarative model. A version 5.*x* application must be secured with Web services security by defining the security constraints in the IBM extension deployment descriptors and in IBM extension bindings.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

No Application Programming Interfaces (APIs) exist in WebSphere Application Server for programmatically interacting with Web services security. However, Service Provider Programming Interfaces (SPIs) are available for extending some security runtime behaviors. You can secure an application with Web services security by defining security constraints in the IBM extension deployment descriptors and in IBM extension bindings.

The development life cycle of a Web services security-enabled application is similar to the Java Platform, Enterprise Edition (Java EE) programming model. See the following figure for more details.

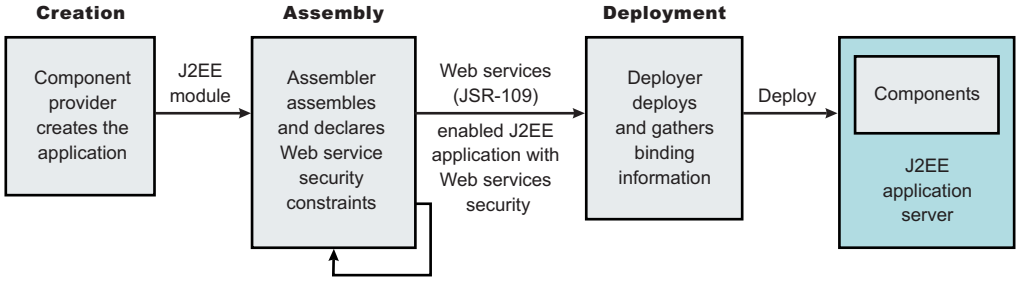


Figure 13. Application development life cycle

The Web services security constraints are defined by the assembler during the application assembly phase if the Java EE application is Web services-enabled. Create, define, and edit the Web services security constraints with an assembly tool. For more information, see Assembly tools.

Web services security constraints

The security constraints for Web services security are specified in the IBM deployment descriptor extension for Web services. The assembler defines these constraints during the application assembly phase, if the Java EE application is Web services enabled. Define the Web services security constraints using an assembly tool. See more information about assembling applications.

The Web services security run time acts on the constraints to enforce Web services security for the SOAP message. The scope of the IBM deployment descriptor extension is at the Enterprise JavaBeans (EJB) module or Web module level. There also are bindings associated with each of the following IBM deployment descriptor extensions:

Client (might be either a Java EE client (application client container) or Web services acting as a client)
- `ibm-webservicesclient-ext.xmi`
- `ibm-webservicesclient-bnd.xmi`

Server
- `ibm-webservices-ext.xmi`
- `ibm-webservices-bnd.xmi`

The IBM extension deployment descriptor and bindings are associated with each EJB module or Web module. See Figure 2 for more information. If Web services is acting as a client, then it contains the client IBM extension deployment descriptors and bindings in the EJB module or Web module.

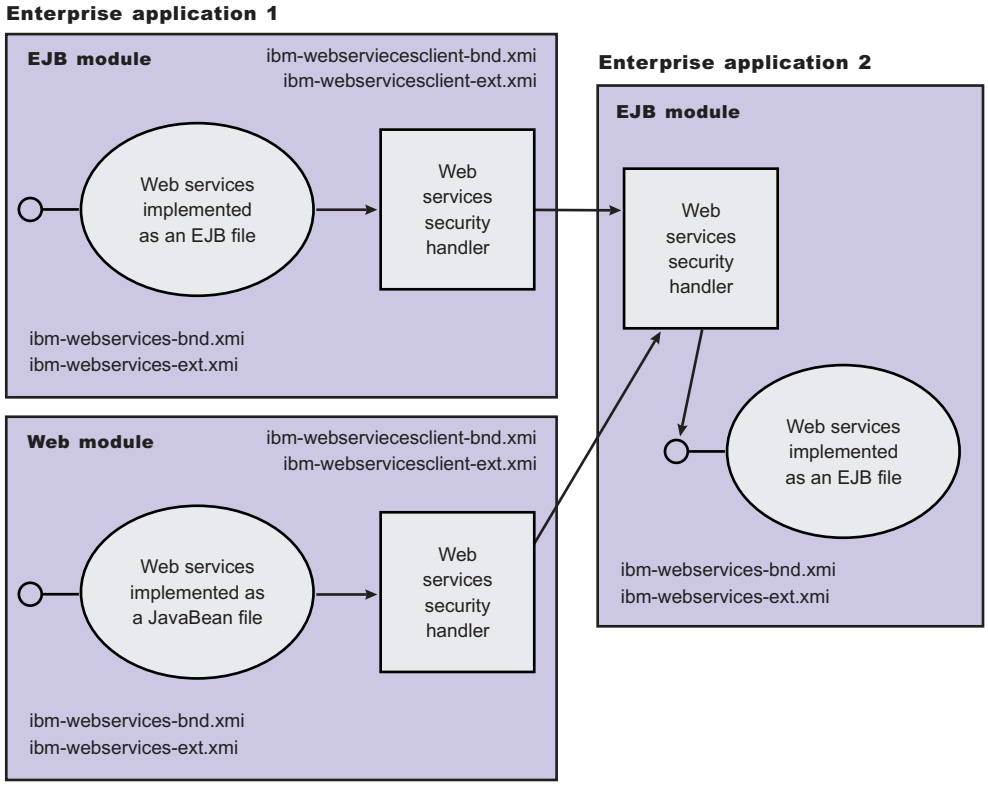


Figure 14. IBM extension deployment descriptors and bindings

The Web services security handler acts on the security constraints defined in the IBM extension deployment descriptor and enforces the security constraints accordingly. There are outbound and inbound configurations in both the client and server security constraints.

In a SOAP request, the following message points exist:

- Sender outbound
- Receiver inbound
- Receiver outbound
- Sender inbound

These message points correspond to the following four security constraints:

- Request sender (sender outbound)
- Request receiver (receiver inbound)
- Response sender (receiver outbound)
- Response receiver (sender inbound)

The security constraints of request sender and request receiver must match. Also, the security constraints of the response sender and response receiver must match. For example, if you specify integrity as a constraint in the request receiver, then you must configure the request sender to have integrity applied to the SOAP message. Otherwise, the request is denied because the SOAP message does not include the integrity specified in the request constraint.

The four security constraints are shown in the following figure of Web services security constraints.

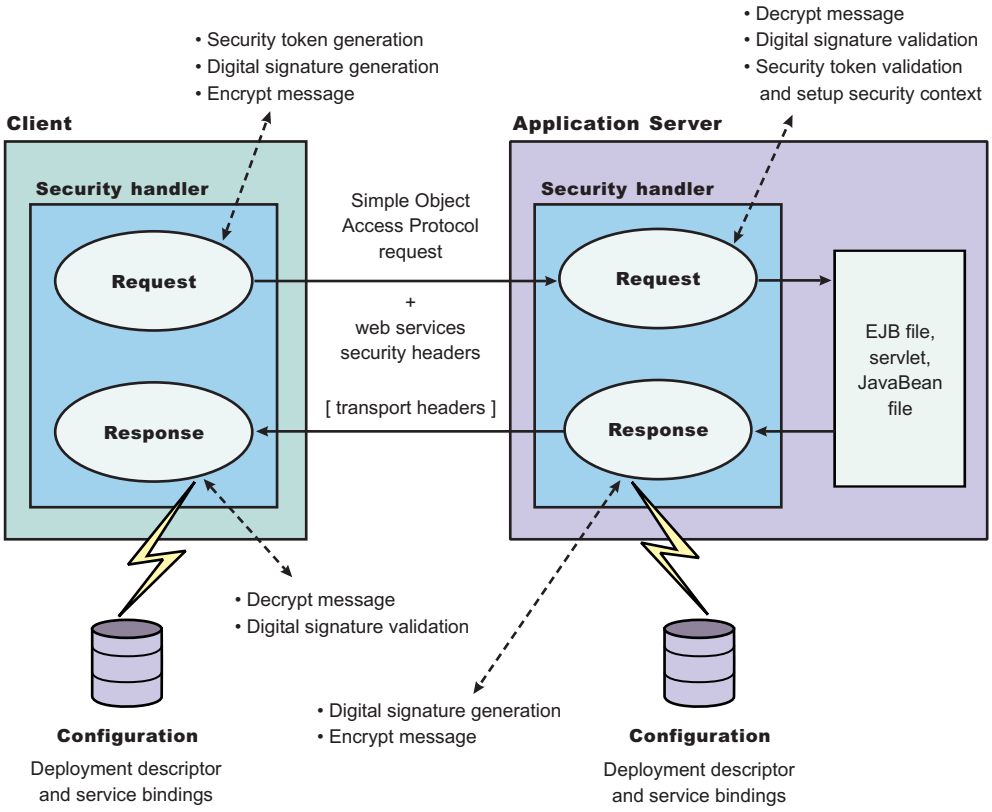


Figure 15. Web services security constraints

Example: Sample configuration for Web services security for a version 5.x application:

Version 5.x application

To secure a version 5.*x* application with Web services security, you must define the security constraints in the IBM extension deployment descriptors and in IBM extension bindings. Sample keystore files and default binding information are provided for a sample configuration to demonstrate what IBM deployment descriptor extensions and bindings can do.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides the following sample keystores for sample configurations. These sample keystores are for testing and sample purposes only. Do not use them a in production environment.

- `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks`
  - The keystore password is `client`
  - Trusted certificate with alias name, `soapca`
  - Personal certificate with alias name, `soaprequester` and key password `client` issued by intermediary certificate authority `Int CA2`, which is, in turn, issued by `soapca`
- `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks`
  - The keystore password is `server`

  - Trusted certificate with alias name, soapca
  - Personal certificate with alias name, soapprovider and key password server, issued by intermediary certificate authority Int CA2, which is, in turn, issued by soapca
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks
  - The keystore password is storepass
  - Secret key CN=Group1, alias name Group1, and key password keypass
  - Public key CN=Bob, O=IBM, C=US, alias name bob, and key password keypass
  - Private key CN=Alice, O=IBM, C=US, alias name alice, and key password keypass
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks
  - The keystore password is storepass
  - Secret key CN=Group1, alias name Group1, and key password keypass
  - Private key CN=Bob, O=IBM, C=US, alias name bob, and key password keypass
  - Public key CN=Alice, O=IBM, C=US, alias name alice, and key password keypass
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
  - The intermediary certificate authority is Int CA2.

Default binding (cell and server level)

WebSphere Application Server provides the following default binding information:

Trust anchors

Used to validate the trust of the signer certificate.

- SampleClientTrustAnchor is used by the response receiver to validate the signer certificate.
- SampleServerTrustAnchor is used by the request receiver to validate the signer certificate.

Collection Certificate Store

Used to validate the certificate path.

- SampleCollectionCertStore is used by the response receiver and the request receiver to validate the signer certificate path.

Key Locators

Used to locate the key for signature, encryption, and decryption.

- SampleClientSignerKey is used by the requesting sender to sign the SOAP message. The signing key name is clientsignerkey, which can be referenced in the signing information as the signing key name.
- SampleServerSignerKey is used by the responding sender to sign the SOAP message. The signing key name is serversignerkey, which can be referenced in the signing information as the signing key name.
- SampleSenderEncryptionKeyLocator is used by the sender to encrypt the SOAP message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator.
- SampleReceiverEncryptionKeyLocator is used by the receiver to decrypt the encrypted SOAP message. The implementation is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). However, to use it for asymmetric encryption (RSA), you must add the private key CN=Bob, O=IBM, C=US, alias name bob, and key password keypass.
- SampleResponseSenderEncryptionKeyLocator is used by the response sender to encrypt the SOAP response message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks keystore and the com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator key locator. This key locator

maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key `alice`, and you must change WebSphere Application Server to the appropriate user. The `SampleResponseSenderEncryptionKeyLocator` key locator also can set a default key for encryption. By default, this key locator is configured to use public key `alice`.

Trusted ID Evaluator

Used to establish trust before asserting to the identity in identity assertion.

`SampleTrustedIDEvaluator` is configured to use the `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl` implementation. The default implementation of `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator` contains a list of trusted identities. The list is defined as properties with `trustedId_*` as the key and the value as the trusted identity. Define this information for the server level in the administration console by completing the following steps:

1. Click **Servers > Server Types > WebSphere application servers***server1*.
2. Under Additional Properties, click **JAX-WS and JAX-RPC security runtime > Trusted ID Evaluators >** *SampleTrustedIDEvaluator*

Login Mapping

Used to authenticate the incoming security token in the Web services security SOAP header of a SOAP message.

- The BasicAuth authentication method is used to authenticate user name security token (user name and password).
- The signature authentication method is used to map a distinguished name (DN) into a WebSphere Application Server Java Authentication and Authorization Server (JAAS) Subject.
- The IDAssertion authentication method is used to map a trusted identity into a WebSphere Application Server JAAS Subject for identity assertion.
- The Lightweight Third Party Authentication (LTPA) authentication method is used to validate a LTPA security token.

The previous default bindings for trust anchors, collection certificate stores, and key locators are for testing or sample purpose only. Do not use them for production.

A sample configuration

The following examples demonstrate what IBM deployment descriptor extensions and bindings can do. The unnecessary information was removed from the examples to improve clarity. Do not copy and paste these examples into your application deployment descriptors or bindings. These examples serve as reference only and are not representative of the recommended configuration.

Use the following tools to create or edit IBM deployment descriptor extensions and bindings:

- Use an assembly tool to create or edit the IBM deployment descriptor extensions.
- Use an assembly tool or the administrative console to create or edit the bindings file.

The following example illustrates a scenario that:

- Signs the SOAP body, time stamp, and security token.
- Encrypts the body content and user name token.
- Sends the user name token (basic authentication data).
- Generates the time stamp for the request.

For the response, the SOAP body and time stamp are signed, the body content is encrypted, and the SOAP message freshness is checked using the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints.

The request sender and the request receiver are a pair. Similarly, the response sender and the response receiver are a pair.

Note: It is recommended that you use the WebSphere Application Server variables for specifying the path to the key stores. In the administrative console, click **Environment > Manage WebSphere Variables**. These variables often help with platform differences such as file system naming conventions. In the following examples, `$${USER_INSTALL_ROOT}` is used for specifying the path to the key stores.

Client-side IBM deployment descriptor extension

The client-side IBM deployment descriptor extension describes the following constraints:

Request Sender

- Signs the SOAP body, time stamp and security token
- Encrypts the body content and user name token
- Sends the basic authentication token (user name and password)
- Generates the time stamp to expire in three minutes

Response Receiver

- Verifies that the SOAP body and time stamp are signed
- Verifies that the SOAP body content is encrypted
- Verifies that the time stamp is present (also check for message freshness)

Example 1: Sample client IBM deployment descriptor extension

The `xmi:id` statements are removed for readability. These statements must be added for this example to work.

Note: In the following code sample, lines 2 through 4 were split into three lines due to the width of the printed page.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscext:WsClientExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:com.ibm.etools.webservice.wscext=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscext.xmi">
  <serviceRefs serviceRefLink="service/myServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <clientServiceConfig actorURI="myActorURI">
        <securityRequestSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </confidentiality>
          <loginConfig authMethod="BasicAuth"/>
          <addCreatedTimeStamp flag="true" expires="PT3M"/>
        </securityRequestSenderServiceConfig>
        <securityResponseReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
          </requiredIntegrity>
          <requiredConfidentiality>
            <confidentialParts part="bodycontent"/>
          </requiredConfidentiality>
          <addReceivedTimeStamp flag="true"/>
```

```
        </securityResponseReceiverServiceConfig>
      </clientServiceConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscext:WsClientExtension>
```

Client-side IBM extension bindings

Example 2 shows the client-side IBM extension binding for the security constraints described previously in the discussion on client-side IBM deployment descriptor extensions.

The signer key and encryption (decryption) key for the message can be obtained from the keystore key locator implementation (`com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator`). The signer key is used for encrypting the response. The sample is configured to use the Java Certification Path API to validate the certificate path of the signer of the digital signature. The user name token (basic authentication) data is collected from the standard in (stdin) prompts using one of the default JAAS implementations `:javax.security.auth.callback.CallbackHandler` implementation (`com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler`).

Example 2: Sample client IBM extension binding

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscbnd:ClientBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wscbnd=
  "http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscbnd.xmi">
  <serviceRefs serviceRefLink="service/MyServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <securityRequestSenderBindingConfig>
        <signingInfo>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <signingKey name="clientsignerkey" locatorRef="SampleClientSignerKey"/>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfo>
        <keyLocators name="SampleClientSignerKey" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
        </keyLocators>
        <encryptionInfo name="EncInfo1">
          <encryptionKey name="CN=Bob, O=IBM, C=US" locatorRef=
          "SampleSenderEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfo>
        <keyLocators name="SampleSenderEncryptionKeyLocator" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}LCswLTovPiws" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks" type="JCEKS"/>
          <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
        </keyLocators>
        <loginBinding authMethod="BasicAuth" callbackHandler=
        "com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
      </securityRequestSenderBindingConfig>
      <securityResponseReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
            <trustAnchorRef ref="SampleClientTrustAnchor"/>
            <certStoreRef ref="SampleCollectionCertStore"/>
          </certPathSettings>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

```
        <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      </signingInfos>
      <trustAnchors name="SampleClientTrustAnchor">
        <keyStore storepass="{xor}PDM2OjEr" path=
        "$${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
      </trustAnchors>
      <certStoreList>
        <collectionCertStores provider="IBMCertPath" name="SampleCollectionCertStore">
          <x509Certificates path="$${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer"/>
        </collectionCertStores>
      </certStoreList>
      <encryptionInfos name="EncInfo2">
        <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
        <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
        <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      </encryptionInfos>
      <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
      "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
        <keyStore storepass="{xor}PDM2OjEr" path=
        "$${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
        <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
      </keyLocators>
    </securityResponseReceiverBindingConfig>
  </portQnameBindings>
 </serviceRefs>
</com.ibm.etools.webservice.wscbnd:ClientBinding>
```

Server-side IBM deployment descriptor extension

The client-side IBM deployment descriptor extension describes the following constraints:

Request Receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Verifies that the SOAP body, time stamp, and security token are signed.
- Verifies that the SOAP body content and user name token are encrypted.
- Verifies that the basic authentication token (user name and password) is in the Web services security SOAP header.
- Verifies that the time stamp is present (also check for message freshness). The freshness of the message indicates whether the message complies with predefined time constraints.

Response Sender (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Signs the SOAP body and time stamp
- Encrypts the SOAP body content
- Generates the time stamp to expire in 3 minutes

Example 3: Sample server IBM deployment descriptor extension

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsext:WsExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsext=
http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsext.xmi">
  <wsDescExt wsDescNameLink="MyServ">
    <pcBinding pcNameLink="Port1">
      <serverServiceConfig actorURI="myActorURI">
        <securityRequestReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </requiredIntegrity>
          <requiredConfidentiality">
```

```
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </requiredConfidentiality>
          <loginConfig>
            <authMethods text="BasicAuth"/>
          </loginConfig>
          <addReceivedTimestamp flag="true"/>
        </securityRequestReceiverServiceConfig>
        <securityResponseSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
          </confidentiality>
          <addCreatedTimestamp flag="true" expires="PT3M"/>
        </securityResponseSenderServiceConfig>
      </serverServiceConfig>
    </pcBinding>
  </wsDescExt>
</com.ibm.etools.webservice.wsext:WsExtension>
```

Server-side IBM extension bindings

The following binding information reuses some of the default binding information defined either at the server level or the cell level, which depends upon the installation. For example, request receiver is referencing the `SampleCollectionCertStore` certification store and the `SampleServerTrustAnchor` trust store is defined in the default binding. However, the encryption information in the request receiver is referencing a `SampleReceiverEncryptionKeyLocator` key locator defined in the application-level binding (the same `ibm-webservices-bnd.xmi` file). The response sender is configured to use the signer key of the digital signature of the request to encrypt the response using one of the default key locator (`com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator`) implementations.

Example 4: Sample server IBM extension binding

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsbnd:WSBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsbnd=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsbnd.xmi">
  <wsdescBindings wsDescNameLink="MyServ">
    <pcBindings pcNameLink="Port1" scope="Session">
      <securityRequestReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
            <trustAnchorRef ref="SampleServerTrustAnchor"/>
            <certStoreRef ref="SampleCollectionCertStore"/>
          </certPathSettings>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfos>
        <encryptionInfos name="EncInfo1">
          <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfos>
        <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
         "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}LCswLTovPiws" path="$${USER_INSTALL_ROOT}/
           etc/ws-security/samples/enc-receiver.jceks" type="JCEKS"/>
          <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
          <keys alias="bob" keypass="{xor}NDomLz4sLA==" name="CN=Bob, O=IBM, C=US"/>
        </keyLocators>
      </securityRequestReceiverBindingConfig>
      <securityResponseSenderBindingConfig>
        <signingInfo>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
```

```
          <signingKey name="serversignerkey" locatorRef="SampleServerSignerKey"/>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfo>
        <encryptionInfo  name="EncInfo2">
          <encryptionKey locatorRef="SignerKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfo>
        <keyLocators name="SignerKeyLocator" classname=
         "com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator"/>
      </securityResponseSenderBindingConfig>
    </pcBindings>
  </wsdescBindings>
  <routerModules transport="http" name="StockQuote.war"/>
</com.ibm.etools.webservice.wsbnd:WSBinding>
```

Overview of authentication methods

Version 5.x application

The Web services security implementation for WebSphere Application Server supports the following authentication methods: BasicAuth, Lightweight Third Party Authentication (LTPA), digital signature, and identity assertion.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

When the WebSphere Application Server is configured to use the BasicAuth authentication method, the sender attaches the Lightweight Third Party Authentication (LTPA) token as a BinarySecurityToken from the current security context or from basic authentication data configuration in the binding file in the SOAP message header. The Web services security message receiver authenticates the sender by validating the user name and password against the configured user registry. With the LTPA method, the sender attaches the LTPA BinarySecurityToken it previously received in the SOAP message header. The receiver authenticates the sender by validating the LTPA token and the token expiration time. With the Digital Signature authentication method, the sender attaches a BinarySecurityToken from a X509 certificate to the Web services security message header along with a digital signature of the message body, time stamp, security token, or any combination of the three. The receiver authenticates the sender by verifying the validity of the X.509 certificate and the digital signature using the public key from the verified certificate.

The identity assertion authentication method is different from the other three authentication methods. This method establishes the security credential of the sender based on the trust relationship. You can use the identity assertion authentication method, for example, when an intermediary server must invoke a service from a downstream server on behalf of the client, but does not have the client authentication information. The intermediary server might establish a trust relationship with the downstream server and then assert the client identity to the same downstream server.

Web Services Security supports the following trust modes:

- BasicAuth
- Digital signature
- Presumed trust

When you use the BasicAuth and digital signature trust modes, the intermediary server passes its own authentication information to the downstream server for authentication. The presumed trust mode establishes a trust relationship using some external mechanism. For example, the intermediary server

might pass SOAP messages through a Secure Socket Layers (SSL) connection with the downstream server and transport layer client certificate authentication.

The Web services security implementation for WebSphere Application Server validates the trust relationship by following this procedure:

1. The downstream server validates the authentication information of the intermediary server.
2. The downstream server verifies whether the authenticated intermediary server is authorized for identity assertion. For example, the intermediary server must be in the trust list for the downstream server.

The client identity might be represented by a name string, a distinguished name (DN), or an X.509 certificate. The client identity is attached in the Web services security message in a UsernameToken with just a user name, DN, or in a BinarySecurityToken of a certificate. The following table summarizes the type of security token that is required for each authentication method.

Table 67. Authentication methods and their security tokens

Authentication method	Security token
BasicAuth	BasicAuth requires <wsse:UsernameToken> with <wsse:Username> and <wsse:Password>.
Signature	Signature requires <ds:Signature> and <wsse:BinarySecurityToken>.
IDAssertion	IDAssertion requires <wsse:UsernameToken> with <wsse:Username> or <wsse:BinarySecurityToken> with a X.509 certificate for client identity depending on <idType>. This method also requires other security tokens according to the <trustMode>: • If the <trustMode> is BasicAuth, IDAssertion requires <wsse:UsernameToken> with <wsse:Username> and <wsse:Password>. • If the <trustMode> is Signature, IDAssertion requires <wsse:BinarySecurityToken>.
LTPA	LTPA requires <wsse:BinarySecurityToken> with an LTPA token.

A Web service can support multiple authentication methods simultaneously. The receiver side of the Web services deployment descriptor can specify all the authentication methods that are supported in the ibm-webservices-ext.xmi XML file. The Web services receiver-side, as shown in the following example, is configured to accept all the authentication methods described previously:

```
<loginConfig xmi:id="LoginConfig_1052760331326">
      <authMethods xmi:id="AuthMethod_1052760331326" text="BasicAuth"/>
      <authMethods xmi:id="AuthMethod_1052760331327" text="IDAssertion"/>
      <authMethods xmi:id="AuthMethod_1052760331336" text="Signature"/>
      <authMethods xmi:id="AuthMethod_1052760331337" text="LTPA"/>
</loginConfig>
<idAssertion xmi:id="IDAssertion_1052760331336" idType="Username" trustMode="Signature"/>
```

You can define only one authentication method in the sender-side Web services deployment descriptor. A Web service client can use any of the authentication methods that are supported by the particular Web services application. The following example illustrates an identity assertion authentication method configuration in the ibm-webservicesclient-ext.xmi deployment descriptor extension of the Web service client:

```
<loginConfig xmi:id="LoginConfig_1051555852697">
      <authMethods xmi:id="AuthMethod_1051555852698" text="IDAssertion"/>
</loginConfig>
<idAssertion xmi:id="IDAssertion_1051555852697" idType="Username" trustMode="Signature"/>
```

As shown in the previous example, the client identity type is `Username` and the trust mode is digital signature.

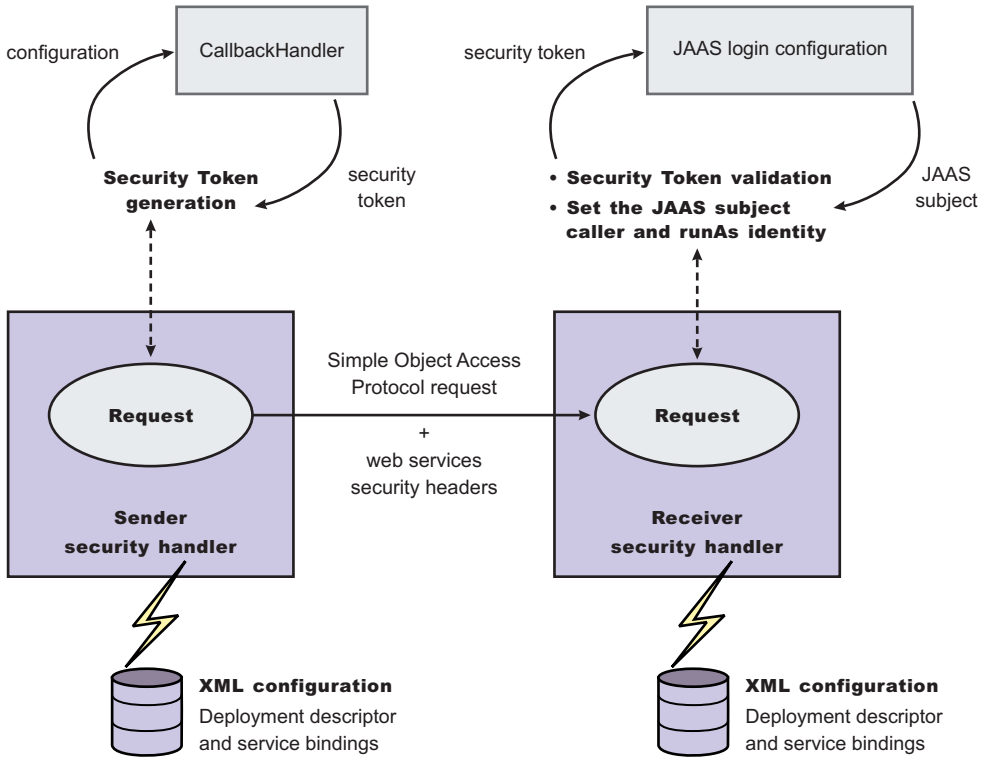


Figure 16. Security token generation and validation

The sender security handler invokes the `handle()` method of an implementation of the `javax.security.auth.callback.CallbackHandler` interface. The `javax.security.auth.callback.CallbackHandler` interface creates the security token and passes it back to the sender security handler. The sender security handler constructs the security token based on the authentication information in the callback array and inserts the security token into the Web services security message header.

The receiver security handler compares the token type in the message header with the expected token types configured in the deployment descriptor. If none of the expected token types are found in the Web services security header of the SOAP message, the request is rejected with a SOAP fault exception. Otherwise, the token type is used to map to a Java Authentication and Authorization Service (JAAS) login configuration for validating the token. If the authentication is successful, a JAAS Subject is created and associated with the running thread. Otherwise, the request is rejected with a SOAP fault exception.

Overview of token types

Version 5.x application

Web services security defines the types of security tokens. The deployment descriptor extension file defines the types of tokens that the message can accept.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.1*x* and later applications.

The types of security tokens that are defined by Web services security are:

- User name token
- Binary security token

A user name token consists of a user name and, optionally, password information. You can include a user name token directly in the <Security> header within the message. Binary tokens, such as X.509 certificates, Kerberos tickets, Lightweight Third Party Authentication (LTPA) tokens, or other non-XML formats, require a special encoding for inclusion. The Web services security specification describes how to encode binary security tokens such as X.509 certificates and Kerberos tickets, and it also describes how to include opaque encrypted keys. The specification also includes extensibility mechanisms that you can use to further describe the characteristics of the credentials that are included with a message.

WebSphere Application Server Version 5.0.2 supports user name tokens, which include both user name and password for basic authentication and user name, which is used for identity assertion. The WebSphere Application Server Version 5.0.2 binary security token implementation supports both X.509 certificates and LTPA binary security. You extend the implementation to generate other types of tokens. However, Kerberos tickets are not supported in WebSphere Application Server Version 5.0.2. Each type of token is processed by a corresponding token generation and validation module. The binary token generation and validation modules are pluggable that is based on the Java Authentication and Authorization Service (JAAS) framework. For example, an arbitrary XML-based token format is supported using the JAAS pluggable framework. WebSphere Application Server Version 5.0.2 does not support an XML-based token that is used in the SecurityTokenReference.

You can define the types of tokens that the message can accept in the deployment descriptor extension file, `ibm.webservices-ext.xmi`. A message receiver might support one or more types of security tokens. The following example shows that the receiver supports four types of security tokens:

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsext:WsExtension xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.etools.webservice.wsext=
"http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsext.xmi"
xmi:id="WsExtension_1052760331306" routerModuleName="StockQuote.war">
  <wsDescExt xmi:id="WsDescExt_1052760331306" wsDescNameLink="StockQuoteFetcher">
    <pcBinding xmi:id="PcBinding_1052760331326" pcNameLink="urn:xmltoday-delayed-quotes"
      scope="Session">
       <serverServiceConfig
         xmi:id="ServerServiceConfig_1052760331326"actorURI="myActorURI">
          <securityRequestReceiverServiceConfig
           xmi:id="SecurityRequestReceiverServiceConfig_1052760331326">
            <loginConfig xmi:id="LoginConfig_1052760331326">
              <authMethods xmi:id="AuthMethod_1052760331326" text="BasicAuth"/>
              <authMethods xmi:id="AuthMethod_1052760331327" text="IDAssertion"/>
              <authMethods xmi:id="AuthMethod_1052760331336" text="Signature"/>
              <authMethods xmi:id="AuthMethod_1052760331337" text="LTPA"/>
           </loginConfig>
<idAssertion xmi:id="IDAssertion_1052760331336" idType="Username" trustMode="Signature"/>
```

The message sender might choose one of the token types that are supported by the receiver when sending a message. You can define the type of token to be used by the sending side in the client descriptor extension file, `ibm-webservicesclient-ext.xmi`. The following example shows that the sender chooses to send a UsernameToken to the receiver:

Note: In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscext:WsClientExtension xmi:version="2.0"
mlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.etools.webservice.wscext=
"http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscext.xmi"
xmi:id="WsClientExtension_1052760331496">
<ServiceRefs xmi:id="ServiceRef_1052760331506" serviceRefLink="service/StockQuoteService">
    <portQnameBindings xmi:id="PortQnameBinding_1052760331506"
portQnameLocalNameLink="StockQuote">
      <clientServiceConfig xmi:id="ClientServiceConfig_1052760331506"
actorURI="myActorURI">
          <securityRequestSenderServiceConfig
xmi:id="SecurityRequestSenderServiceConfig_1052760331506" actor="myActorURI">
          <loginConfig xmi:id="LoginConfig_1052760331506" authMethod="BasicAuth"/>
```

Username token: **Version 5.x application**

The <UsernameToken> element propagates a user name and optionally propagates the password information. Use this token type to carry basic authentication information.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Both a user name and a password are used to authenticate the message. A <UsernameToken> element that contains the user name is used in identity assertion. Identity assertion establishes the identity of the user, based on the trust relationship.

The following example shows the syntax of the <UsernameToken> element:

```
<UsernameToken Id="...">
    <Username>...</Username>
    <Password Type="...">...</Password>
</UsernameToken>
```

The Web services security specification defines the following password types:

wsse:PasswordText (default)
This type is the actual password for the user name.

wsse:PasswordDigest
The type is the digest of the password for the user name. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

WebSphere Application Server supports the default PasswordText type. However, it does not support password digest because most user registry security policies do not expose the password to the application software.

The following example illustrates the use of the <UsernameToken> element:

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Joe</wsse:Username>
                <wsse:Password>ILoveJava</wsse:Password>
```

```
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
</S:Envelope>
```

Nonce, a randomly generated token: **Version 5.x application**

Nonce is a randomly generated, cryptographic token used to prevent the theft of user name tokens used with SOAP messages. Nonce is used with the basic authentication (BasicAuth) method.

Without nonce, when a UsernameToken is passed from one machine to another machine using a nonsecure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same key might be reused when the username token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption.

To help eliminate these replay attacks, the <wsse:Nonce> and <wsu:Created> elements are generated within the <wsse: usernameToken> element and used to validate the message. The request receiver or response receiver checks the freshness of the message to verify the difference between when the message is created and the current time falls within a specified time period. Also, WebSphere Application Server verifies that the receiver has not processed the token within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

Binary security token: **Version 5.x application**

The <ValueType> attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType indicates how the security token is encoded, for example, Base64Binary. The <BinarySecurityToken> element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 5.0.2 supports both LTPA and X.509 certificate binary security tokens.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

A binary security token has the following attributes that are used for interpretation:

- Value type
- Encoding type

The following example depicts an LTPA binary security token in a Web services security message header:

```
<wsse:BinarySecurityToken xmlns:ns7902342339871340177=
                      "http://www.ibm.com/websphere/appserver/tokentype/5.0.2"
     EncodingType="wsse:Base64Binary"
     ValueType="ns7902342339871340177:LTPA">
            MIZ6LGPt2CzXBQfio9wZTo1VotWov0NW3Za6lU5K7Li78DSnIK6iHj3hxXgrUn6p4wZI
            8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbFQJOqh5VcRvI+AKEaNmnEgEV65jUYAC9
            C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
            msu0sewsOKfl/WPsjW0bR/2g3NaVvBy18VlTFBpUbGFVGgzHRjBKAGo+ctkl80nlVLIk
            TUjt/XdYvEpOr6QoddGi4okjDGPyyoDxcvKZnReXww5UsoqlpfXwN4KG9as=
</wsse:BinarySecurityToken></wsse:Security></soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

XML token: **Version 5.x application**

XML tokens are offered in two formats, Security Assertion Markup Language (SAML) and Extensible rights Markup Language (XrML).

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

XML-based security tokens are growing in popularity. Two well-known formats are:

- Security Assertion Markup Language (SAML)
- Extensible rights Markup Language (XrML)

Using extensibility of the <wsse:Security> header in XML-based security tokens, you can directly insert these security tokens into the header.

SAML assertions are attached to Web services security messages using Web services by placing assertion elements inside the <wsse:Security> header. The following example illustrates a Web services security message with a SAML assertion token.

```
<S:Envelope xmlns:S="...">&
      <wsse:Security xmlns:wsse="...">
          <saml:Assertion
                    MajorVersion="1"
                    MinorVersion="0"
                    AssertionID="SecurityToken-ef375268"
                       Issuer="elliotw1"
                       IssueInstant="2002-07-23T11:32:05.6228146-07:00"
                     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                     ...
           </saml:Assertion>
      </wsse:Security>
 </S:Header>
 <S:Body>
 ...
 </S:Body>
</S:Envelope>
```

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

XML digital signature

Version 5.x application

XML-Signature Syntax and Processing (XML signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

XML signature does not introduce new cryptographic algorithms. WebSphere Application Server uses XML signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method.

XML canonicalization (c14n) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. For example, although their octet representations are different, the following examples are identical:

- `<person first="John" last="Smith"/>`
- `<person last="Smith" first="John"></person>`

C14n is a process used to canonicalize XML information. Select an appropriate c14n algorithm because the information that is canonicalized is dependent upon this algorithm. One of the major c14n algorithms, Exclusive XML Canonicalization, canonicalizes the character encoding scheme, attribute order, namespace declarations, and so on. The algorithm does not canonicalize white space outside tags, namespace prefixes, or data type representation.

XML signature in the Web Services Security-Core specification

The Web Services Security-Core (WSS-Core) specification defines a standard way for SOAP messages to incorporate an XML signature. You can use almost all of the XML signature features in WSS-Core except enveloped signature and enveloping signature. However, WSS-Core has some recommendations such as exclusive canonicalization for the c14n algorithm and some additional features such as `SecurityTokenReference` and `KeyIdentifier`. The `KeyIdentifier` is the value of the SubjectKeyIdentifier field within the X.509 certificate. For more information on the `KeyIdentifier`, see “Reference to a Subject Key Identifier” within the OASIS Web Services Security X.509 Certificate Token Profile documentation.

By including XML signature in SOAP messages, the following are realized:

Message integrity
: A message receiver can confirm that attackers or accidents have not altered parts of the message after these parts are signed by a key.

Authentication
: You can assume that a valid signature is *proof of possession*. A message with a digital certificate issued by a certificate authority and a signature in the message that is validated successfully by a public key in the certificate, is proof that the signer has the corresponding private key. The receiver can authenticate the signer by checking the trustworthiness of the certificate.

XML signature in the current implementation

XML signature is supported in Web services security, however, an application programming interface (API) is not available. The current implementation has many hardcoded behaviors and has some user-operable configuration items. To configure the client for digital signature, see Configuring the client for response digital signature verification: Verifying the message parts. To configure the server for digital signature, see Configuring the server for request digital signature verification: Verifying the message parts.

Security considerations

In a replay attack, an attacker taps the lines, receives a signed message, and then returns the message to the receiver. In this case, the receiver receives the same message twice and might process both of them if the signatures are valid. Processing both messages can cause damage to the receiver if the message is a claim for money. If you have the signed generation time stamp and the signed expiration time in a message replay, attacks might be reduced. However, this is not a complete solution. A message must have a nonce value to prevent these attacks and the receiver must reject a message that contains a processed nonce. The current implementation does not provide a standard way to generate and check nonces in messages. In WebSphere Application Server, Version 5.1, nonce is supported in username tokens only. The username token profile contains concrete nonce usage scenarios for username tokens. Applications handle nonces (such as serial numbers) and they need to be signed.

Signing parameter configuration settings: **Version 5.x application**

Use this page to configure new signing parameters.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Signature Syntax and Specification: W3C Recommendation 12 Feb 2002*.

To view this administrative console page, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Under Modules, click **Manage modules** → ***URI_name***.
3. **Version 5.x application** Under Additional properties, you can access the signing information for the following bindings:
   a. For the Request sender binding, click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. Under Additional properties, click **Signing information**.
   b. For the Response sender binding, click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. Under Additional properties, click **Signing information**.
4. In the Request Sender Binding column, click **Edit** → **Signing Information**.

If the signing information is not available, select **None**.

If the signing information is available, select **Dedicated Signing Information** and specify the configuration in the following fields:

Signature method: **Version 5.x application**

Specifies the algorithm Uniform Resource Identifiers (URI) of the signature method.

The following algorithms are supported:
- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2000/09/xmldsig#dsa-sha1
- http://www.w3.org/2000/09/xmldsig#hmac-sha1

You can also add custom algorithms.

Digest method: **Version 5.x application**

Specifies the algorithm URI of the digest method.

WebSphere Application Server supports the http://www.w3.org/2000/09/xmldsig#sha1 algorithm.

Canonicalization method: **Version 5.x or 6 application**

Specifies the algorithm URI of the canonicalization method.

The following algorithms are supported:
- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/2001/10/xml-exc-c14n#WithComments
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Key name: **Version 5.x application**

Specifies the name of the key object found in the keystore file.

Key locator reference: **Version 5.x or 6 application**

Specifies the name used to reference the key locator

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

You can specify a key locator configuration for the following bindings on the following levels:

Binding name	Cell level, server level, or application level	Path
N/A	Server level	1. Click **Servers** → **Server Typ** → **WebSphere application servers** → ***server_name***. 2. Under Security, click **JAX-WS and JAX-RPC security runtime**. **Note:** In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**.
Request sender	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** → ***URI_name***. 3. Click **Web services: Client security bindings**. 4. Under Request sender binding, click **Edit**. 5. Under Additional properties, click **Key locators**.

Binding name	Cell level, server level, or application level	Path
Request receiver	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage module** → ***URI_name***. 3. Click **Web services: Server security bindings**. 4. Under Request receiver binding, click **Edit**. 5. Under Additional properties, click **Key locators**.
Response sender	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage module** → ***URI_name***. 3. Click **Web services: Server security bindings**. 4. Under Response sender binding, click **Edit**. 5. Under Additional properties, click **Key locators**.
Response receiver	Application level	1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***. 2. Under Modules, click **Manage modules** → ***URI_name***. 3. Click **Web services: Client security bindings**. 4. Under Response receiver binding, click **Edit**. 5. Under Additional properties, click **Key locators**.

Securing Web services for Version 5.x applications using XML digital signature

Version 5.x application

XML digital signature is one of the methods WebSphere Application Server provides to secure your Web services. It provides message integrity and authentication capabilities when used with SOAP messages.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides several different methods to secure your Web services; XML digital signature is one of these methods. You can secure your Web services by using any of the following methods:

- XML digital signature

- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

About this task

XML digital signature provides both message integrity and authentication capabilities when it is used with SOAP messages. A message receiver can verify that attackers or accidents have not altered parts of the message after the message was signed by a key. If a message has a digital certificate issued by a certificate authority (CA) and a signature in the message is validated successfully by a public key in the certificate, it is proof that the signer has the corresponding private key. To use XML digital signature to secure Web services, complete the following steps:

1. Define the security constraints or extensions. To configure the security constraints, you must use an assembly tool. For more information, see the related information on Assembly Tools.
   a. Configure the client to digitally sign a message request. To configure the client, complete the following steps to specify which parts of the SOAP message to digitally sign and define the method used to digitally sign the message. The client in these steps is the request sender.
      1) Specify the message parts by following the steps found in “Configuring the client for request signing: digitally signing message parts” on page 572.
      2) Select the method used to digitally sign the request message. You can select the digital signature method by following the steps in “Configuring the client for request signing: choosing the digital signature method” on page 574.
   b. Configure the server to verify the digital signature that is used in the message request. To configure the server, you must specify which parts of the SOAP message, sent by the request sender, contain digitally signed information and which method was used to digitally sign the message. The settings chosen for the request receiver, or the server in this step, must match the settings chosen for the request sender in the previous step.
      1) Define the message parts by following the steps found in “Configuring the server for request digital signature verification: Verifying the message parts” on page 576.
      2) Select the same method used by the request sender to digitally sign the message. You can select the digital signature method by following the steps in “Configuring the server for request digital signature verification: choosing the verification method” on page 578
   c. Configure the server to digitally sign a message response. To configure the server, complete the following steps to specify which parts of the SOAP message to digitally sign and define the method used to digitally sign the message. The sender in these steps is the response sender.
      1) Specify which message parts to digitally sign by following the steps found in “Configuring the server for response signing: digitally signing message parts” on page 581.
      2) Select the method used to digitally sign the response message. You can select the digital signature method by following the steps in “Configuring the server for response signing: choosing the digital signature method” on page 583
   d. Configure the client to verify the digital signature that is used in the message response. To configure the client, you must specify which parts of the SOAP message sent by the response sender contain digitally signed information and which method was used to digitally sign the message. The settings chosen for the response receiver, or client in this step, must match the settings chosen for the response sender in the previous step.
      1) Define the message parts by following the steps found in “Configuring the client for response digital signature verification: verifying the message parts” on page 584
      2) Select the same method used by the response sender to digitally sign the message. You can select the digital signature method by following the steps in “Configuring the client for response digital signature verification: choosing the verification method” on page 586

2. Define the client security bindings. To configure the client security bindings, complete the steps in either of the following topics:
   - “Configuring the client security bindings using an assembly tool” on page 588
   - “Configuring the security bindings on a server acting as a client using the administrative console” on page 591
3. Define the server security bindings. To configure the server security bindings, complete the steps in either of the following topics:
   - “Configuring the server security bindings using an assembly tool” on page 593
   - “Configuring the server security bindings using the administrative console” on page 595

Results

After completing these steps, you have secured your Web services using XML digital signature.

Configuring nonce using Web services security tokens: Version 5.x application

Nonce is a randomly generated, cryptographic token that is used to thwart the highjacking of user name tokens, which are used with SOAP messages. Use nonce in conjunction with the BasicAuth authentication method.

About this task

Note: The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure nonce at the application level and server level.

If you configure nonce on the application level and the server level, the values specified for the application level take precedence over the values specified for the server level.

You must consider the order of precedence:

1. Application level
2. Server level

Complete these high-level tasks in the order listed:

1. Configure nonce for the application level.
2. Configure nonce for the server level.

What to do next

After completing these steps, restart the server if it has not already been restarted.

Configuring nonce for the server level: Version 5.x application

Nonce is a randomly generated, cryptographic token that is used to prevent the theft of username tokens, which are used with SOAP messages. Nonce is used in conjunction with the basic authentication (BasicAuth) method. You can configure nonce for the server level by using the WebSphere Application Server administrative console.

About this task

Note: The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure nonce at the application level and the cell level.

However, you must consider the order of precedence:

1. Application level
2. Server level

If you configure nonce on the application level and the server level, the values specified for the application level take precedence over the values specified for the server level.

In a WebSphere Application Server or WebSphere Application Server Express environment, you must specify values for the Nonce cache timeout, Nonce maximum age, and Nonce clock skew fields on the server level to use nonce effectively.

Complete the following steps to configure nonce on the server level:

1. Connect to the administrative console.

   Type `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Servers > Server Types > WebSphere application servers > *server_level***.
3. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
4. Specify a value, in seconds, for the Nonce cache timeout field. The value specified for the Nonce cache timeout field indicates how long the nonce remains cached before it is expunged. You must specify a minimum of 300 seconds. However, if you do not specify a value, the default is 600 seconds. This field is required for the server level.
5. Specify (optional) a value, in seconds, for the Nonce maximum age field.

   The value specified for the Nonce Maximum Age field indicates how long the nonce is valid. You must specify a minimum of 300 seconds, but the value cannot exceed the number of seconds specified for the Nonce cache timeout field on the server level.

   This field is required for the server level.
6. Specify a value, in seconds, for the Nonce clock skew field. The value specified for the Nonce clock skew field specifies the amount of time, in seconds, to consider when the message receiver checks the timeliness of the value. Consider the following information when you set this value:
   - Difference in time between the message sender and the message receiver if the clocks are not synchronized.
   - Time needed to encrypt and transmit the message.
   - Time needed to get through network congestion.

   You must specify at least 0 seconds for the Nonce clock skew field. However, the maximum value cannot exceed the number of seconds specified in the Nonce maximum age field on the server level. If you do not specify a value, the default is 0 seconds.
7. Restart the server. If you change the Nonce cache timeout value and do not restart the server, the change is not recognized by the server.

Configuring nonce for the application level: **Version 5.x application**

Nonce is a randomly generated, cryptographic token that is used to thwart the highjacking of Username tokens, which are used with SOAP messages. Use nonce in conjunction with the basic authentication (BasicAuth) method. You can configure nonce for the application level by using the WebSphere Application Server administrative console.

About this task

Note: The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure nonce at the application level and server level.

However, you must consider the order of precedence:

1. Application level
2. Server level

If you configure nonce on the application level and the server level, the values specified for the application level take precedence over the values specified for the server level.

1. Connect to the administrative console.

   Type `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Manage modules, click ***URI_name***.
4. Under Web Services Security Properties, click **Web services: Server security bindings**.
5. Click **Edit** under Request receiver binding
6. Under Additional properties, click **Login mappings > New**.
7. Specify (optional) a value, in seconds, for the **Nonce maximum age** field. This panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value:

   ```
   Nonce is not supported for authentication methods other than
   BasicAuth.
   ```

   If you specify BasicAuth, but do not specify values for the **Nonce maximum age** field, the Web services security runtime searches for a nonce maximum age value on the server level.

   The value specified for the **Nonce maximum age** field indicates how long the nonce is valid. You must specify a minimum of 300 seconds; however, the value cannot exceed the number of seconds that is specified for the **Nonce cache timeout** field for the server level.

   You can specify the nonce cache timeout value for the server level by completing the following steps:

   a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
   b. Under Security, click **JAX-WS and JAX-RPC security runtime**.

      Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
8. Specify (optional) a value, in seconds, for the **Nonce clock skew** field. The value specified for the **Nonce clock skew** field specifies the amount of time, in seconds, to consider when the message receiver checks the timeliness of the value. This panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value:

```
Nonce is not supported for authentication methods other than
BasicAuth.
```

If you specify BasicAuth, but do not specify values for the **Nonce clock skew** field, the Web services security runtime searches for a Nonce clock skew value on the server level.

Consider the following information when you set this value:

- Difference in time between the message sender and the message receiver if the clocks are not synchronized.
- Time needed to encrypt and transmit the message.
- Time needed to get through network congestion.

9. Restart the server.

Default binding:

The default binding information is defined in the `ws-security.xml` file and can be administered by either the administrative console or by scripting. Only default bindings for JAX-RPC applications are supported. Default bindings for JAX-WS applications are not supported.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications. Also, policy sets can only be used with JAX-WS applications. Policy sets cannot be used for JAX-RPC applications.

Certain applications can share certain binding information. This information includes truststores, keystores, and authentication methods (token validation). WebSphere Application Server provides support for default binding information. Administrators can define binding information at:

- The server level

Applications can refer to this binding information.

You can define the following binding information in the `ws-security.xml` file:

Trust anchors (truststore)

- *Trust anchors* contain key store configuration information that has the root-trusted certificates. Trust anchors are used for certificate path validation of the incoming X.509-formatted security tokens.
- The Trust Anchor Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the trust anchor defined in the default binding information. The trust anchor name must be unique in the trust anchor collection.

Collection certificate store

- The *collection certificate store* specifies a list of untrusted, intermediate certificates and is used for certificate path validation of incoming X.509-formatted security tokens. The default provider is IBMCertPath.
- The Certificate Store Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the certificate store defined in the default binding information. The Certificate Store Name must be unique to the collection certificate store collection.

Key locators

- *Key locators* specify implementation of the com.ibm.wsspi.wssecurity.config.KeyLocator interface. This interface is used to retrieve keys for signature or encryption. Customer implementations can extend the key locator interface to retrieve keys using other methods.

WebSphere Application Server provides implementations to retrieve a key from the key store, map an authenticated identity to a key in the key store, or retrieve a key from the signer certificate (mapping and retrieving actions are used for encrypting the response).

- The Key Locator Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the key locator defined in the default binding information. The Key Locator Name must be unique to the key locators collection in the default binding information.

Trusted ID evaluators

- *Trusted ID evaluators* are an implementation of the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. This interface is used to make sure the identity (ID)-asserting authority is trusted. Additionally, you can extend the trusted identity evaluator to validate the trust. WebSphere Application Server provides a default implementation for validating trust based on a predefined list of identities.
- The Trusted ID Evaluator Name is used in the binding file (`ibm-webservices-bnd.xmi`) to refer to the trusted identity evaluator defined in the default binding information. The Trusted ID Evaluator Name must be unique to the Trusted ID Evaluator collection.

Login mappings

- *Login mappings* define the mapping of the authentication method to the Java Authentication and Authorization Service (JAAS) login configuration. The mappings are used to authenticate the incoming security token embedded in the Web services security SOAP message header. The JAAS login configuration is defined in the administrative console under **Security > Global security > Java Authentication and Authorization Service > Application logins**.
- WebSphere Application Server defines the following authentication methods:

  BasicAuth
  Authenticates user name and password.

  Signature
  Maps the subject distinguished name (DN) in the certificate to a WebSphere Application Server credential.

  IDAssertion
  Maps the identity to a WebSphere Application Server credential.

  LTPA Authenticates a Lightweight Third Party Authentication (LTPA) token.

  After identity authentication, the associated credential is used in the downstream call.
- This method can be extended to authenticate custom security tokens by providing a custom JAAS login configuration and by using the com.ibm.wsspi.wssecurity.auth.module.WSSecurityMappingModule to create the principal and credential required by WebSphere Application Server.
- If LoginConfig (`AuthMethod`) is defined in the IBM extension deployment descriptor (`ibm-webservices-ext.xmi`), but there are no login mapping bindings (`ibm-webservices-bnd.xmi`) defined for the `AuthMethod`, Web services security run time uses the login mapping defined in the default binding information.

Default configuration for WebSphere Application Server: **Version 5.x application**

For JAX-RPC applications, each application server, in WebSphere Application Server, uses a copy of the `ws-security.xml` file to define the default binding information for Web services security.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

In the WebSphere Application Server, each application server has a copy of the `ws-security.xml` file, which defines the default binding information for Web services security. The following list contains the defaults defined in the `ws-security.xml` file:

Trust anchors
Identifies the trusted root certificates for signature verification.

Collection certificate stores
Contains certificate revocation lists (CRLs) and non-trusted certificates for verification.

Key locators
Locates the keys for digital signature and encryption.

Trusted ID evaluators
Evaluates the trust of the received identity before identity assertion.

Login mappings
Contains the Java Authentication and Authorization Service (JAAS) configurations for AuthMethod token validation.

If the Web services security constraints specified in the deployment descriptors and the required bindings are not defined in the bindings file, the default constraints in the `ws-security.xml` file are used.

When you use the **addNode** command, the `ws-security.xml` file is added with the server configuration to the new cell. The following figure shows the activity when you use the **addNode** command.

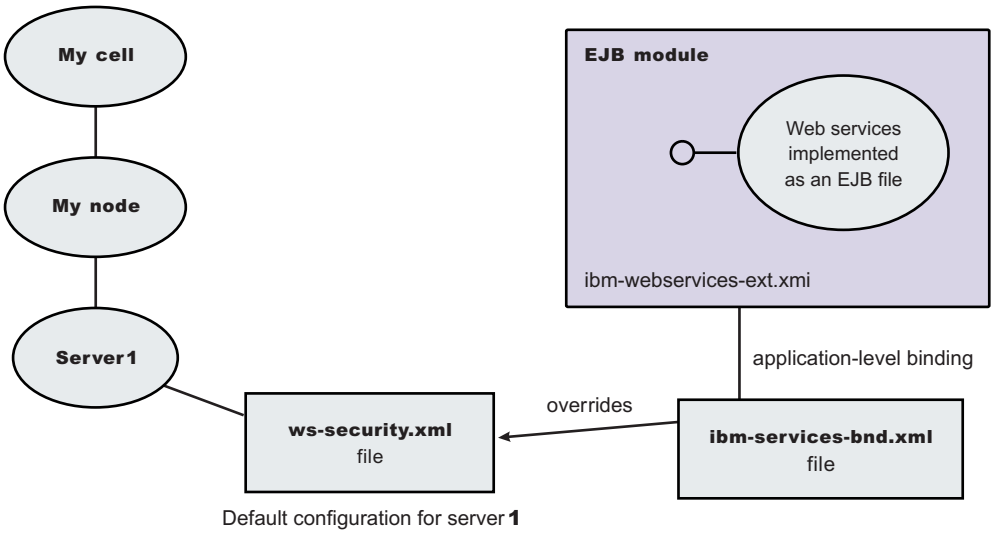


Figure 17. Configuration when using the **addNode** *command*

Trust anchors: **Version 5.x application**

A trust anchor specifies keystores that contain trusted root certificates that validate the signer certificate. The request receiver and the response receiver use these keystores to validate the signer certificate of the digital signature.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `ibm-webservicesclient-bnd.xmi` file when Web services is acting as client) use these keystores to validate the signer certificate of the digital signature. The keystores are critical to the integrity of the digital signature validation. If the keysores are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request receiver in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response receiver in the `ibm-webservicesclient-bnd.xmi` file.

The trust anchor is defined as javax.security.cert.TrustAnchor in the Java CertPath application programming interface (API). The Java CertPath API uses the trust anchor and the certificate store to validate the incoming X.509 certificate that is embedded in the SOAP message.

The Web services security implementation in WebSphere Application Server supports this trust anchor. In WebSphere Application Server, the trust anchor is represented as a Java keystore object. The type, path, and password of the keystore are passed to the implementation through the administrative console or by scripting.

Configuring trust anchors using an assembly tool: Version 5.x application

Use an assembly tool to configure trust anchors (that specify keystores which contain trusted root certificates to validate the signer certificate) or trust stores at the application level.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This document describes how to configure trust anchors or trust stores at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors defined at the application level have a higher precedence over trust anchors defined at the server or cell level. You can configure an application-level trust anchor using an assembly tool or the administrative console. This document describes how to configure the application-level trust anchor using an assembly tool. For more information on creating and configuring trust anchors at the server or cell level, see either “Configuring the server security bindings using an assembly tool” on page 593 or “Configuring the server security bindings using the administrative console” on page 595.

About this task

A trust anchor specifies keystores that contain trusted root certificates, which validate the signer certificate. These keystores are used by the request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `application-client.xml` file when Web services is acting as client) to validate the signer certificate of the digital signature. The keystores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration

specified for the request receiver in the ibm-webservices-bnd.xmi file must match the binding configuration for the response receiver in the application-client.xml file.

Complete the following steps to configure trust anchors using an assembly tool.

1. Configure an assembly tool to work with a Java Platform, Enterprise Edition (Java EE) enterprise application. For more information, see the related information on Assembly Tools.
2. Create a Web services-enabled Java EE enterprise application. See either "Configuring the server security bindings using an assembly tool" on page 593 or "Configuring the server security bindings using the administrative console" on page 595 for an introduction on how to manage Web services security binding information on the server.
3. Configure the client-side response receiver, which is defined in the ibm-webservicesclient-bnd.xmi bindings extensions file.
   a. Use an assembly tool to import your Java EE application.
   b. Click **Window > Open Perspective > Other > J2EE**.
   c. Click **Application Client projects** > ***application_name*** > **appClientModule > META-INF**
   d. Right-click the application-client.xml file, select **Open with > Deployment Descriptor Editor**, and click the **WS Binding** tab. The Client Deployment Descriptor is displayed.
   e. Locate the Port qualified name binding section and either select an existing entry or click **Add**, to add a new port binding. The Web services client port binding editor displays for the selected port.
   f. Locate the Trust anchor section and click **Add**. The Trust anchor window is displayed.
      1) Enter a unique name within the port binding for the **Trust anchor name**.
         The name is used to reference the trust anchor that is defined.
      2) Enter the keystore password, path, and keystore type.
         The supported keystore types are the Java Cryptography Extension (JCE) and Java Cryptography Extension Keystores (JCEKS) types.

      Click **Edit** to edit the selected trust anchor.

      Click **Remove** to remove the selected trust anchor.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
   g. Save the changes.
4. Configure the server-side request receiver, which is defined in the ibm-webservices-bnd.xmi bindings extensions file.
   a. Click **Window > Open perspective > J2EE**.
   b. Select the Web services enabled Enterprise JavaBeans (EJB) or Web module.
   c. In the Package Explorer window, click the META-INF directory for an EJB module or the WEB-INF directory for a Web module.
   d. Right-click the webservices.xml file, select **Open with > Web services editor**, and click the bindings tab. The Web services bindings editor is displayed.
   e. Locate the Web service description bindings section and either select an existing entry or click **Add** to add a new Web services descriptor.
   f. Click **Binding configurations**. The Web services binding configurations editor is displayed for the selected Web services descriptor.
   g. Locate the Trust anchor section and click **Add**. The Trust anchor dialog box is displayed.
      1) Enter a unique name within the binding for the **Trust anchor name**.
         This unique name is used to reference the trust anchor defined.
      2) Enter the keystore password, path, and keystore type. The supported keystore types are JCE and JCEKS.

      Click **Edit** to edit the selected trust anchor.

Click **Remove** to remove the selected trust anchor.

When you start the application, the configuration is validated in the run time while the binding information is loading.

h. Save the changes.

Results

This procedure defines trust anchors that can be used by the request receiver or the response receiver (if the Web services is acting as client) to verify the signer certificate.

Example

The request receiver or the response receiver (if the Web service is acting as a client) uses the defined trust anchor to verify the signer certificate. The trust anchor is referenced using the trust anchor name.

What to do next

To complete the signing information configuration process for request receiver, complete the following tasks:

1. “Configuring the server for request digital signature verification: Verifying the message parts” on page 576
2. “Configuring the server for request digital signature verification: choosing the verification method” on page 578

To complete the process for the response receiver, if the Web services is acting as a client, complete the following tasks:

1. “Configuring the client for response digital signature verification: verifying the message parts” on page 584
2. “Configuring the client for response digital signature verification: choosing the verification method” on page 586

Configuring trust anchors using the administrative console: **Version 5.x application**

Use the WebSphere Application Server administrative console to configure trust anchors that specify key stores which contain trusted root certificates to validate the signer certificate.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This document describes how to configure trust anchors or trust stores at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors defined at the application level have a higher precedence over trust anchors defined at the server or cell level. For more information on creating and configuring trust anchors at the server or cell level, see either “Configuring the server security bindings using an assembly tool” on page 593 or “Configuring the server security bindings using the administrative console” on page 595.

You can configure an application-level trust anchor using an assembly tool or the administrative console. This document describes how to configure the application-level trust anchor using the administrative console.

About this task

A trust anchor specifies key stores that contain trusted root certificates, which validate the signer certificate. These key stores are used by the request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `ibm-webservicesclient-bnd.xmi` file when Web services is acting as client) to validate the signer certificate of the digital signature. The keystores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request receiver in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response receiver in the `ibm-webservicesclient-bnd.xmi` file.

The following steps are for the client-side response receiver, which is defined in the `ibm-webservicesclient-bnd.xmi` file and the server-side request receiver, which is defined in the `ibm-webservices-bnd.xmi` file.

1. Configure an assembly tool to work with a Java Platform, Enterprise Edition (Java EE) enterprise application. For more information, see the related information on Assembly Tools.
2. Create a Web services-enabled Java EE enterprise application. See either “Configuring the server security bindings using an assembly tool” on page 593 or “Configuring the server security bindings using the administrative console” on page 595 for an introduction on how to manage Web services security binding information on the server.
3. Click **Applications > Application Types > WebSphere enterprise applications > *enterprise_application***.
4. Under Manage modules, click ***URI_name***.
5. Under Web Services Security Properties, click **Web services: client security bindings** to edit the response receiver binding information, if Web services is acting as a client.
   a. Under Response receiver binding, click **Edit**.
   b. Under Additional properties, click **Trust anchors**.
   c. Click **New** to create a new trust anchor.
   d. Enter a unique name within the request receiver binding for the Trust anchor name field. The name is used to reference the trust anchor that is defined.
   e. Enter the key store password, path, and key store type.
   f. Click the trust anchor name link to edit the selected trust anchor.
   g. Click **Remove** to remove the selected trust anchor or anchors.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
6. Return to the Web services-enabled module panel accessed in step 2.
7. Under Web Services Security Properties, click **Web services: server security bindings** to edit the request receiver binding information.
   a. Under Request receiver binding, click **Edit**.
   b. Under Additional properties, click **Trust anchors**.
   c. Click **New** to create a new trust anchor

      Enter a unique name within the request receiver binding for the Trust anchor name field. The name is used to reference the trust anchor that is defined.

      Enter the key store password, path, and key store type.

      Click the trust anchor name link to edit the selected trust anchor.

      Click **Remove** to remove the selected trust anchor or anchors.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
8. Save the changes.

Results

This procedure defines trust anchors that can be used by the request receiver or the response receiver (if the Web services is acting as client) to verify the signer certificate.

Example

The request receiver or the response receiver (if the Web service is acting as a client) uses the defined trust anchor to verify the signer certificate. The trust anchor is referenced using the trust anchor name.

What to do next

To complete the signing information configuration process for request receiver, complete the following tasks:

1. “Configuring the server for request digital signature verification: Verifying the message parts” on page 576
2. “Configuring the server for request digital signature verification: choosing the verification method” on page 578

To complete the process for the response receiver, if the Web services is acting as client, complete the following tasks:

1. “Configuring the client for response digital signature verification: verifying the message parts” on page 584
2. “Configuring the client for response digital signature verification: choosing the verification method” on page 586

Collection certificate store: **Version 5.x application**

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

The collection certificate stores are used when processing a received SOAP message. This collection is configured in the securityRequestReceiverBindingConfig section of the binding file for servers and in the securityResponseReceiverBindingConfig section of the binding file for clients.

A collection certificate store is one kind of certificate store. A certificate store is defined as javax.security.cert.CertStore in the Java CertPath application programming interface (API). The Java CertPath API defines the following types of certificate stores:

Collection certificate store
A collection certificate store accepts the certificates and CRLs as Java collection objects.

Lightweight Directory Access Protocol certificate store
The Lightweight Directory Access Protocol (LDAP) certificate store accepts certificates and CRLs as LDAP entries.

The CertPath API uses the certificate store and the trust anchor to validate the incoming X.509 certificate that is embedded in the SOAP message.

The Web services security implementation in the WebSphere Application Server supports the collection certificate store. Each certificate and CRL is passed as an encoded file. This configuration is done using either the administrative console or by scripting.

Configuring the client-side collection certificate store using an assembly tool:

Version 5.x application

You can configure the client-side collection certificate store using the assembly tool.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message.

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console. Complete the following steps to configure the client-side collection certificate store using the assembly tool.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the **WS Binding** tab, which is located at the bottom of deployment descriptor editor within the assembly tool. The Client Deployment Descriptor is displayed.
5. Click the **Port binding** tab in deployment descriptor editor within the assembly tool. The Web services client port binding window is displayed.
6. Select one of the port-qualified name binding entries.
7. Expand the **Security response receiver binding configuration > certificate store list > Collection certificate store** section.
8. Click **Add** to create a new collection certificate store, click **Edit** to edit an existing certificate store, or click **Remove** to delete an existing certificate store.
9. Enter a name in the **Name** field. This name is referenced in the Certificate store reference field in the Signing info dialog box.
10. Leave the Provider field as `IBMCertPath`.
11. Click **Add** to enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have additional certificate store paths, click **Add** to add the paths.
12. Click **OK** when you finish adding paths.

Configuring the client-side collection certificate store using the administrative console:

Version 5.x application

You can configure the client-side collection certificate store by using the administrative console.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message.

You can configure the collection certificate either by using the assembly tools or the WebSphere Application Server administrative console. Complete the following steps to configure the client-side collection certificate store using the administrative console.

1. Connect to the WebSphere Application Server administrative console.

   You can connect to the administrative console by typing `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Manage modules, click *URI_name*.
4. Under Web Services Security Properties, click **Web services: Client security bindings** to add the collection certificate store to the client security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions for either the client or the server.

   To configure the security extensions for the client, see the following topics:
   - "Configuring the client for response digital signature verification: verifying the message parts" on page 584
   - "Configuring the client for response digital signature verification: choosing the verification method" on page 586
5. Under Response receiver binding, click **Edit** to edit the client security bindings.
6. Click **Collection certificate store**.
7. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
8. Enter a name in the Certificate store name field. The name entered in this field is a name that is referenced in the Certificate store field on the Signing information configuration page.
9. Leave the Certificate store provider field value as `IBMCertPath`.
10. Click **Apply**.
11. Under Additional properties, click **X.509 certificates > New**.
12. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have any additional certificate store paths to enter, click **New** and add the path names.
13. Click **OK**.

Configuring the server-side collection certificate store using an assembly tool:

Version 5.x application

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collections of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message. You can configure the server-side collection certificate store by using an assembly tool.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure the collection certificate either by using an assembly tool or by using the WebSphere Application Server administrative console. Complete the following steps to configure the server-side collection certificate store using an assembly tool.

1. Start an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB projects > *application_name* > ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, select **Open with > Web Services Editor**.
5. Click the Binding configurations tab in the Web services editor within the assembly tool. The Web Service Binding Configuration window is displayed.
6. Select one of the Web service description binding entries under the Port Component Binding section.
7. Expand the **Request receiver binding configuration details > Certificate store list > Collection certificate store** section.
8. Click **Add** to create a new collection certificate store, click **Edit** to edit an existing certificate store, or click **Remove** to delete an existing certification store.
9. Enter a name in the **Name** field. This name is referenced in the **Certificate store reference** field in the Signing info dialog.
10. Leave the **Provider** field as `IBMCertPath`.
11. Click **Add** to enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT]/etc/ws-security/samples/intca2.cer`. If you have additional certificate store paths, click **Add** to add the paths.
12. Click **OK** when you finish adding paths.

Configuring the server-side collection certificate store using the administrative console:

Version 5.x application

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

Complete the following steps to configure the server-side collection certificate store using the administrative console.

1. Connect to the WebSphere Application Server administrative console.

You can connect to the administrative console by typing `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.

2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Manage modules, click ***URI_name***
4. Under Web Services Security Properties, click **Web services: server security bindings** to add the collection certificate store to the server security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions for the server.

   To configure the security extensions for the server, see the following topics:

   - "Configuring the server for request digital signature verification: Verifying the message parts" on page 576
   - "Configuring the server for request digital signature verification: choosing the verification method" on page 578
5. Click **Edit** under Request Receiver Binding to edit the server security bindings.
6. Click **Collection certificate store**.
7. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
8. Enter a name in the Certificate store name field. The name entered in this field is a name that is referenced in the Certificate store field on the Signing information configuration page.
9. Leave the Certificate store provider field as `IBMCertPath`.
10. Click **Apply**.
11. Under Additional Properties, click **X.509 Certificates > New**.
12. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have any additional certificate store paths to enter, click **New** and add the path names.
13. Click **OK**.

Configuring default collection certificate stores at the server level in the WebSphere Application

Server administrative console: **Version 5.x application**

You can define a single collection certificate store for all of the applications that need to use the same certificates. Use the WebSphere Application Server administrative console to configure the default collection certificate store at the server level.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message. A certificate store typically refers to a certificate store located in the file system. The location of the certificate store can vary from machine to machine, so you might configure a default collection certificate store for a specific machine and reference it from within the signing information. The signing information is found within the binding configurations of any application installed on the machine. This suggestion enables you to define a single collection certificate store for all of the applications that need to use the same certificates. You also can specify the default binding information at the cell level.

Complete the following steps to configure the default collection certificate store at the server level using the WebSphere Application Server administrative console:

1. Connect to the administrative console.

   You can access the administrative console by typing `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Servers > Server Types > WebSphere application servers > *server_name***.
3. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
4. Under Additional properties, click **Collection certificate store**.
5. Enter a name in the **Certificate store name** field. This name is referenced in the **Certificate store** field on the Signing information configuration page.
6. Leave the **Certificate store provider** field value as `IBMCertPath`.
7. Click **Apply**.
8. Under Additional properties, click **X.509 certificates > New**.
9. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`.

   If you have any additional certificate store paths to enter, click **New** and add the path names.
10. Click **OK**.

Key locator: **Version 5.x application**

A *key locator* (com.ibm.wsspi.wssecurity.config.KeyLocator) is an abstraction of the mechanism that retrieves the key for digital signature and encryption.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

You can use any of the following infrastructure from which to retrieve the keys depending upon the implementation:

- Java keystore file
- Database
- Lightweight Third Party Authentication (LTPA) server

Key locators search the key using some type of a clue. The following types of clues are supported:

- A string label of the key, which is explicitly passed through the application programming interface (API). The relationships between each key and its name (string label) is maintained inside the key locator.
- The execution context of the key locator; explicit information is not passed to the key locator. A key locator determines the appropriate key according to the execution context.

Current versions of key locators do not support the retrieval of verification keys because current Web services security implementations do not support the secret key-based signature. Because the key locators support the public key-based signature only, the key for verification is embedded in the X.509 certificate as a `<BinarySecurityToken>` element in the incoming message.

For example, key locators can obtain the identity of the caller from the context and can retrieve the public key of the caller for response encryption.

Usage scenarios

This section describes the usage scenarios for key locators.

Signing

The name of the signing key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. The corresponding X.509 certificate also can be returned.

Verification

As described previously, key locators are not used in signature verification.

Encryption

The name of the encryption key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned.

Decryption

The Web services security specification recommends using the key identifier instead of the key name. However, while the algorithm for computing the identifier for the public keys is defined in Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280, there is no agreed upon algorithm for the secret keys. Therefore, the current implementation of Web services security uses the identifier only when public key-based encryption is performed. Otherwise, the ordinal key name is used.

When you use public key-based encryption, the value of the key identifier is embedded in the incoming encrypted message. Then, the Web services security implementation searches for all the keys managed by the key locator and decrypts the message using the key whose identifier value matches the one in the message.

When you use secret key-based encryption, the value of the key name is embedded in the incoming encrypted message. The Web services security implementation asks the key locator for the key with the name that matches the name in the message and decrypts the message using the key.

Keys: Version 5.x application

Keys are used for XML signature and encryption.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

There are two predominant kinds of keys used in the current Web services security implementation:

- Public key - such as Rivest Shamir Adleman (RSA) encryption and Digital Signature Algorithm (DSA) encryption
- Secret key - such as Data Encryption Standard (DES) encryption

In public key-based signature, a message is signed using the sender private key and is verified using the sender public key. In public key-based encryption, a message is encrypted using the receiver public key and is decrypted using the receiver private key. In secret key-based signature and encryption, the same key is used by both parties.

While the current implementation of Web services security can support both kinds of keys, there are a few items to note:

- Secret key-based signature is not supported.
- The format of the message differs slightly between public key-based encryption and secret key-based encryption.

Web services security service provider programming interfaces:

Version 5.x application

Several Service Provider Interfaces (SPIs) are provided to extend the capability of the Web services security runtime.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The following list contains the SPIs that are available for WebSphere Application Server:

- com.ibm.wsspi.wssecurity.config.KeyLocator is an abstract for obtaining the keys for digital signature and encryption. The following list contains the default implementations:
  1. com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator implements the Java key store.
  2. com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator povides a mapping of the authenticated identity to a key for encryption or, the implementation uses the default key that is specified.
  3. com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator Provides the capability of using the signer key for encryption in the response message. This implementation is typically used in the response sender configuration.
- com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator is an interface that is used to evaluate the trust for identity assertion. The default implementation is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, which enables you to define a list of trusted identities.
- The Java Authentication and Authorization Service (JAAS) CallbackHandler application programming interfaces (APIs) are used for token generation by the request sender. This interface can be extended to generate a custom token that can be inserted in the Web services security header. The following list contains the default implementations that are provided by WebSphere Application Server:
  1. com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler presents a login prompt to gather the basic authentication data. Use this implementation in the client environment only.
  2. com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler cllects the basic authentication data in the standard in (stdin) prompt. Use this implementation in the client environment only.
  3. com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler reads the basic authentication data from the application binding file. This implementation might be used on the server side to generate a user name token.
  4. com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler Generates a Lightweight Third Party Authentication (LTPA) token in the Web services security header as a binary security token. If basic authentication data is defined in the application binding file, it is used to perform a login, to extract the LTPA token from the WebSphere credentials, and to insert the token in the Web services security header. Otherwise, it will extract the LTPA security token from the invocation credentials (RunAs identity) and insert the token in the Web services security header.

What to do next

The JAAS LoginModule API is used for token validation on the request receiver side of the message. You can implement a custom LoginModule API to perform validation of the custom token on the request receiver of the message. After the token is verified and validated, the token is set as the caller and then run as the identity in the WebSphere Application Server runtime. The identity is used for authorization checks by the containers before a Java Platform, Enterprise Edition (Java EE) resource is invoked. The following list presents the default AuthMethod configurations provided by WebSphere Application Server:

BasicAuth
Validates a user name token.

Signature
Maps the distinguished name (DN) of a verified certificate to a Java Authentication and Authorization Service (JAAS) subject.

IDAssertion
Maps a trusted identity to a JAAS subject.

LTPA Validates an LTPA token that is received in the message and creates a JAAS subject.

Configuring key locators using an assembly tool: Version 5.x application

The following information provides instructions on how to configure key locators using an assembly tool.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure key locators in various locations within the assembly tool. The following procedure provides instructions on how to configure key locators at any of these locations because the concept is the same.

1. Start an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the **WS Binding** tab. The Client Deployment Descriptor is displayed.
5. Click the **WS Binding** tab in deployment descriptor editor within the assembly tool or the **Binding configurations** tab in the Web services editor within the assembly tool.
6. Expand one of the **Binding configuration** sections.
7. Expand the **Key locators** section.
8. Click **Add** to create a new key locator, click **Edit** to edit an existing key locator, or click **Remove** to delete an existing key locator.
9. Enter a key locator name. The name entered for the **Key locator name** is used to refer to the key locator from the Encryption information and Signing Information sections.
10. Enter a key locator class. The key locator class is the implementation of the `KeyLocator` interface. When using default implementations, select a class from the menu.
11. Determine whether to click **Use key store**. Select this option when you use the default implementations as they use key stores. If you click **Use key store**, complete the following steps:

a. Enter a value in the key store storepass field. The key store storepass is the password used to access the key store.
b. Enter a path name in the key store path field. The key store path is the location on the file system where the key store resides. Make sure that the location can be found wherever you deploy the application.
c. Enter a type value in the key store type field. The valid types to enter are JKS and JCEKS. JKS is used when you are not using the Java Cryptography Extensions (JCE) policy. JCEKS is used when you are using JCE. Although the JCEKS type is more secure, it might decrease performance.
d. Click **Add** to create an entry for a key in the key store.
   1) Enter a value in the Alias field.

      The key alias is a reference to this particular key from the Signing Information section.
   2) Enter a value in the Key pass field.

      The key pass is the password associated with the certificate which is created using the keytool utility. The keytool utility is available using the QShell Interpreter.
   3) Enter a value in the Key name field.

      The key name refers to the alias of the certificate as found in the key store.

12. Click **Add** to create a custom property. The property can be used by custom key locator implementations. For example, you can use properties with the `WSIdKeyStoreMapKeyLocator` default implementation. The key locator implementation has the following property names:
    - *id_*, which maps to a credential user ID.
    - *mappedName_* , which maps to the key alias to use for this user name.
    - *default*, which maps to a key alias to use when a credential does not have an associated *id_* entry.

    A typical set of properties for this key locator might be: `id_1=user1`, `mappedName_1=key1`, `id_2=user2`, `mappedName_2=key2`, `default=key3`. If `user1` or `user2` authenticates, then the associated `key1` or `key2` is used, respectively. However, if none of the user properties authenticate or the user is not `user1` or `user2`, then `key3` is used.
    a. Enter a name in the Name field. The name entered is the property name.
    b. Enter a value in the Value field. This value entered is the property value.

Configuring key locators using the administrative console: Version 5.x application

You can configure binding information and key locators using the WebSphere Application Server administrative console.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task provides instructions on how to configure key locators using the WebSphere Application Server administrative console. You can configure binding information in the administrative console. You must use an assembly tool to configure extensions. The following steps are used to configure a key locator in the administrative console for a specific application:

1. Open the administrative console.

   Type `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.

2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Related Items, click either **Web Modules** or **EJB Modules**, depending on the type of module you are securing.
4. Click the name of the module you are securing.
5. Under Additional Properties, click either **Web services: Client security bindings** or **Web services: Server security bindings**, depending on whether you are adding the key locator to the client security bindings or to the server security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions.
6. Edit the Request Sender Binding, Response Receiver Binding, Request Receiver Binding, or Response Sender Binding.
   - If you are editing your client security bindings, click **Edit** for either the Request Sender Binding or the Response Receiver Binding.
   - If you are editing your server security bindings, click **Edit** for either the Request Receiver Binding or the Response Sender Binding.
7. Click **Key Locators**.
8. Click **New** to configure a new key locator, select the box next to a key locator name and click **Delete** to delete a key locator, or click the name of a key locator to edit its configuration. If you are configuring a new key locator or editing an existing one, complete the following steps:
   a. Specify a name for the key locator in the **Key Locator Name** field.
   b. Specify a name for the key locator class implementation in the **Key Locator Classname** field. WebSphere Application Server has the following default key locator class implementations:

      com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator
      This class is used by the response sender to map an authenticated identity to a key. If encryption is used, this class is used to locate a key to encrypt the response message. The com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator class has the capability to map an authenticated identity from the invocation credential of the current thread to a key that is used to encrypt the message. If an authenticated identity is present on the current thread, the class maps the ID to the mapped name. For example, user1 is mapped to mappedName_1. Otherwise, `name="default"`. When a matching key is not found, the authenticated identity is mapped to the default key specified in the binding file.

      com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator
      This class is used by the response receiver, the request sender, and the request receiver to map a name to an alias. Encryption uses this class to obtain a key to encrypt a message and digital signature uses this class to obtain a key to sign a message. The com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator class maps a logical name to a key alias in the key store file. For example, key #105115176771 maps to `CN=Alice, O=IBM, C=US`.

   c. Specify the password used to access the key store password in the **Key Store Password** field. This field is optional because the key locator does not use a key store.
   d. Specify the path name used to access the key store in the **Key Store Path** field. This field is optional because the key locator does not use a key store. Use `${USER_INSTALL_ROOT}` because this path expands to the WebSphere Application Server path on your machine.
   e. Select a keystore type from the **Key Store Type** field. This field is optional because the key locator does not use a key store. Use the **JKS** option if you are not using the Java Cryptography Extensions (JCE) policy and use **JCEKS** if you are using the JCE policy.

Trusted ID evaluator: **Version 5.x application**

The trusted ID evaluator is an abstraction of the mechanism that evaluates whether the given ID name is to be trusted. The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

Depending upon the implementation, you can use various types of infrastructure to store a list of the trusted IDs, such as:

- Plain text file
- Database
- Lightweight Directory Access Protocol (LDAP) server

The Web services security implementation (com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator) invokes the trusted ID evaluator and passes the identity name of the intermediary as a parameter. If the identity is evaluated and deemed trustworthy, the procedure continues. Otherwise, an exception is created and the procedure is stopped.

Login mappings: **Version 5.x application**

Login mappings, found in the `ibm-webservices-bnd.xmi` Extended Markup Language (XML) file, contains a mapping configuration. This mapping configuration defines how the Web services security handler maps the token <ValueType> element that is contained within the security token extracted from the message header, to the corresponding authentication method. The token <ValueType> element is contained within the security token extracted from a SOAP message header.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

The sender-side Web services security handler generates and attaches security tokens based on the <AuthMethods> element that is specified in the deployment descriptor. For example, if the authentication method is BasicAuth, the sender-side security handler generates and attaches UsernameToken (with both user name and password) to the SOAP message header. The Web services security run time uses the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface as a security provider to generate security tokens on the client side (or when Web services is acting as client).

The sender security handler invokes the `handle()` method of a javax.security.auth.callback.CallbackHandler interface implementation. This implementation creates the security token and passes the token back to the sender security handler. The senders security handler constructs the security token based on the authentication information in the callback array. The security handler then inserts the security token into the Web Services Security message header.

The CallbackHandler interface implementation that you use to generate the required security token is defined in the <loginBinding> element in the `ibm-webservicesclient-bnd.xmi` Web services security binding file. For example,

```
<loginBinding xmi:id="LoginBinding_1052760331526" authMethod="BasicAuth"
      callbackHandler="com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
```

The <loginBinding> element associates the com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler interface with the BasicAuth authentication method. WebSphere Application Server provides the following set of CallbackHandler interface implementations you can use to create various security token types:

com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
If there is no basic authentication data defined in the login binding information (this information is not the same as the HTTP basic authentication information), the previous token type prompts for user name and password through a login panel. The implementation uses the basic authentication data defined in the login binding. Use this CallbackHandler with the BasicAuth authentication method. Do not use this CallbackHandler implementation on the server because it prompts you for login binding information.

com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
If basic authentication data is not defined in the login binding (this information is not the same as the HTTP basic authentication information), the implementation prompts for the user name and password using standard in (stdin). The implementation uses the basic authentication data defined in the login binding. Use this CallbackHandler implementation with the BasicAuth authentication method. Do not use this CallbackHandler implementation on the server because it prompts you for login binding information.

com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
This CallbackHandler implementation does not prompt. Rather, it uses the basic authentication data defined in the login binding (this information is not the same as the HTTP basic authentication information). This CallbackHandler implementation is meant for use with the BasicAuth authentication method. You must define the basic authentication data in the login binding information for this CallbackHandler implementation. You can use this implementation when Web services is running as a client and needs to send basic authentication (<wsse:UsernameToken>) to the downstream call.

com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
The CallbackHandler generates Lightweight Third Party Authentication (LTPA) tokens from the run as JAAS Subject (invocation Subject) of the current WebSphere Application Server security context. However, if basic authentication data is defined in the login binding information (not the HTTP basic authentication information), the implementation uses the basic authentication data and LTPA token generated. The **Token Type URI** and **Token Type Local Name** values must be defined in the login binding information for this CallbackHandler implementation. The token value type is used to validate the token to the request sender and request receiver binding configuration. The Web services security run time inserts the LTPA token as a binary security token (<wsse:BinarySecurityToken>) into the message SOAP header. The value type is mandatory. (See LTPA for more information). Use this CallbackHandler implementation with the LTPA authentication method.

Figure 1 shows the sender security handler in the request sender message process.

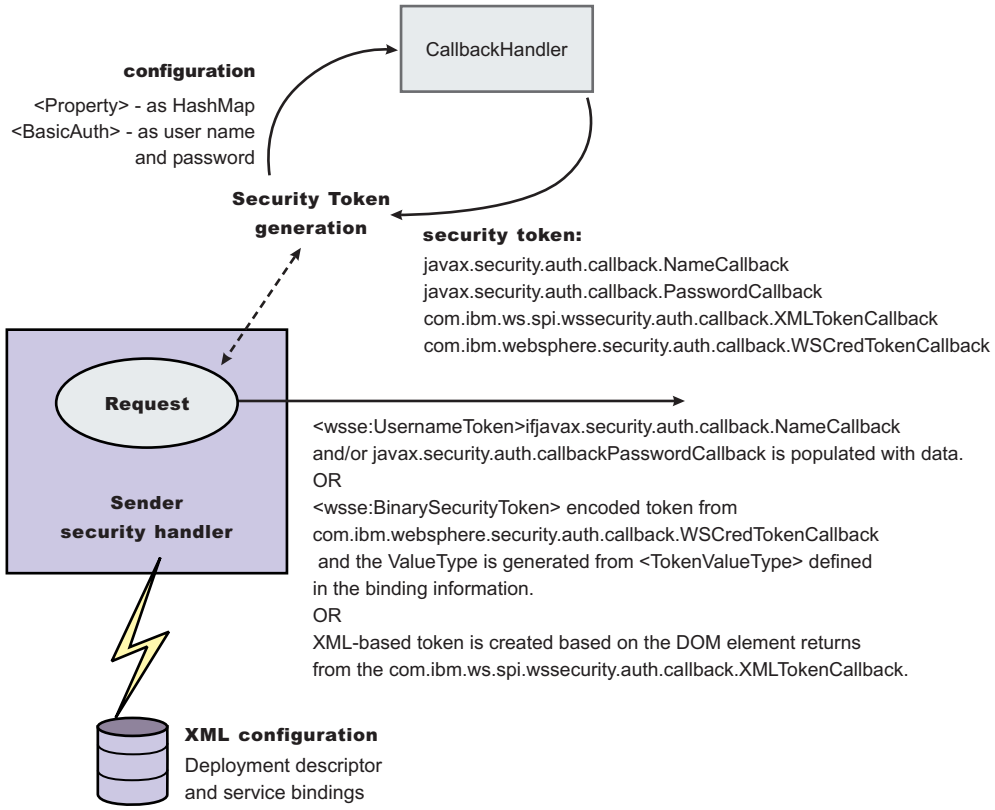


Figure 18. Request sender SOAP message process

You can configure the receiver-side security server to support multiple authentication methods and multiple types of security tokens. The following steps describe the request sender SOAP message process:

1. After receiving a message, the receiver Web services security handler compares the token type (in the message header) with the expected token types configured in the deployment descriptor.
2. The Web services security handler extracts the security token form the message header and maps the token <ValueType> element to the corresponding authentication method. The mapping configuration is defined in the <loginMappings> element in the `ibm-webservices-bnd.xmi` XML file. For example:

   ```
   <loginMappings xmi:id="LoginMapping_1051977980074" authMethod="LTPA"
         configName="WSLogin">
        <callbackHandlerFactory xmi:id="CallbackHandlerFactory_1051977980081"
        classname="com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl"/>
         <tokenValueType xmi:id="TokenValueType_1051977980081"
         uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2" localName="LTPA"/>
   </loginMappings>
   ```

   The com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface is a factory for javax.security.auth.callback.CallbackHandler.
3. The Web services security run time initiates the factory implementation class and passes the authentication information from Web services security header to the factory class through the set methods.
4. The Web services security run time invokes the newCallbackHandler() method to obtain the javax.security.auth.CallbackHandler object, which generates the required security token.
5. When the security handler receives an LTPA BinarySecurityToken, it uses the WSLogin JAAS login configuration and the newCallbackHandler() method to validate the security token. If none of the expected token types are found in the SOAP message Web services security header, the request is rejected with an SOAP fault. Otherwise, the token type is used to map to a JAAS login configuration

for token validation. If authentication is successful, a JAAS Subject is created and associated with the running thread. Otherwise, the request is rejected with a SOAP fault.

The following table shows the authentication methods and the JAAS login configurations.

Authentication method	JAAS login configuration
BasicAuth	WSLogin
Signature	system.wssecurity.Signature
LTPA	WSLogin
IDAssertion	system.wssecurity.IDAssertion

Figure 2 shows the receiver security handler in the request receiver message process.

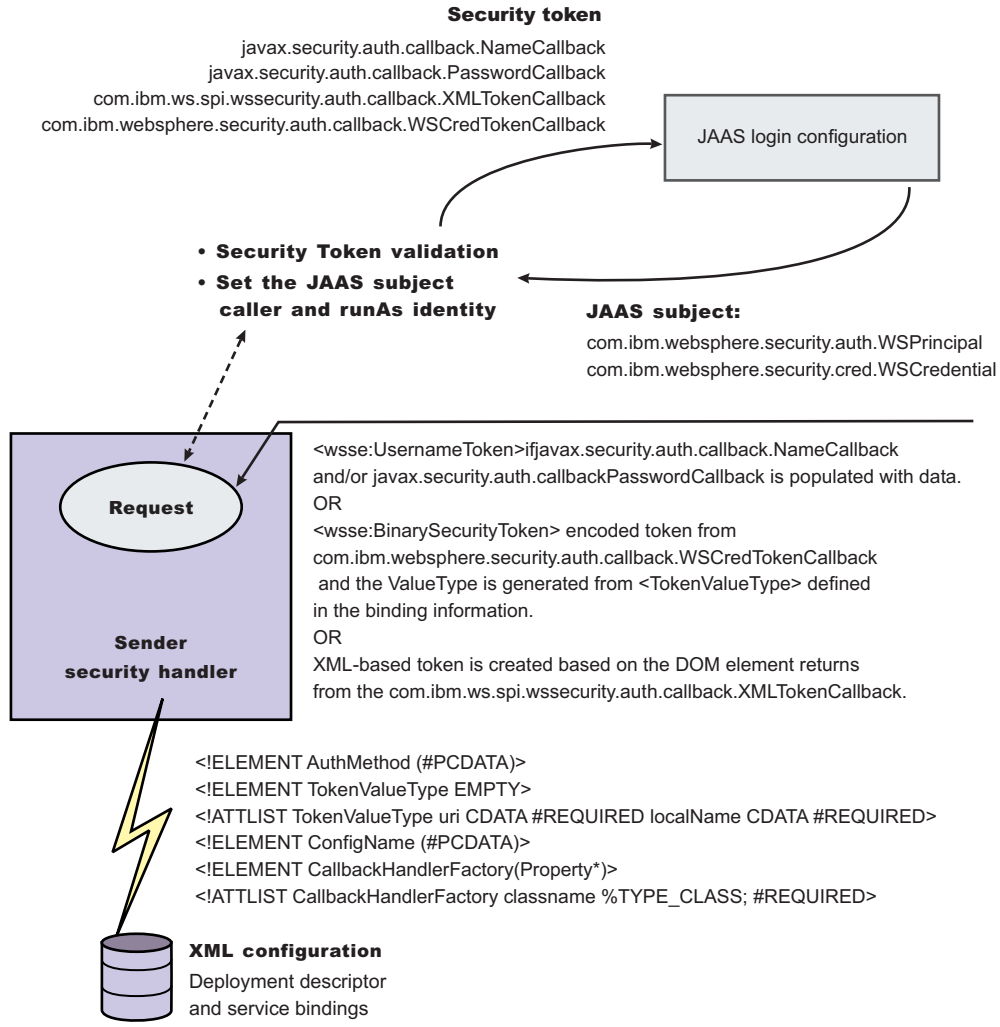


Figure 19. Request receiver SOAP message process

The default <LoginMapping> is defined in the following files:

- Server-level `ws-security.xml` file

If nothing is defined in the binding file information, the `ws-security.xml` default is used. However, an administrator can override the default by defining a new <LoginMapping> element in the binding file.

6. The client reads the default binding information in the ${*install_dir*}/properties/ws-security.xml file.
7. The server runtime component loads the following files if they exist:
   - Server-level ws-security.xml file

   On a base application server, the server run time component only loads the server-level ws-security.xml file. The server-side ws-security.xml file and the application Web services security binding information are managed using the administrative console. You can specify the binding information during application deployment using the administrative console. The Web services security policy is defined in the deployment descriptor extension (ibm-webservicesclient-ext.xmi) and the bindings are stored in the IBM binding extension (ibm-webservicesclient-bnd.xmi). However, the ${*install_dir*}/properties/ws-security.xml file defines the default binding value for the client. If the binding information is not specified in the binding file, the run time reads the binding information from the default ${*install_dir*}/properties/ws-security.xml file.

Login mappings collection: **Version 5.x application**

Use this page to view a list of configurations for validating security tokens within incoming messages. Login mappings map an authentication method to a Java Authentication and Authorization Service (JAAS) login configuration to validate the security token. Four authentication methods are predefined in the WebSphere Application Server: BasicAuth, Signature, IDAssertion, and Lightweight Third Party Authentication (LTPA).

Note: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.
4. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

To view this administrative console page for the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Login mappings**.

If you click **Update runtime**, the Web services security run time is updated with the default binding information, which is contained in the ws-security.xml file that was previously saved. After you specify the authentication method, the JAAS configuration name, and the Callback Handler Factory class name on this panel, you must complete the following steps:

1. Click **Save** in the messages section at the top of the administrative console.

2. Click **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.

Note: If the login mapping configuration is not found on the application level, the Web services run time searches for the login mapping configuration on the server level.

Authentication method: **Version 5.x application**

Specifies the authentication method used for validating the security tokens.

The following authentication methods are available:

BasicAuth
The basic authentication method includes both a user name and a password in the security token. The information in the token is authenticated by the receiving server and is used to create a credential.

Signature
The signature authentication method sends an X.509 certificate as a security token. For Lightweight Directory Access Protocol (LDAP) registries, the distinguished name (DN) is mapped to a credential, which is based on the LDAP certificate filter settings. For local OS registries, the first attribute of the certificate, usually the common name (CN) is mapped directly to a user name in the registry.

IDAssertion
The identity assertion method maps a trusted identity (ID) to a WebSphere Application Server credential. This authentication method only includes a user name in the security token. An additional token is included in the message for trust purposes. When the additional token is trusted, the IDAssertion token user name is mapped to a credential.

LTPA Lightweight Third Party Authentication (LTPA) validates an LTPA token.

JAAS configuration name: **Version 5.x application**

Specifies the name of the Java Authentication and Authorization Service (JAAS) configuration.

Callback handler factory class name: **Version 5.x application**

Specifies the name of the factory for the CallbackHandler class.

Login mapping configuration settings: **Version 5.x application**

Use this page to specify the Java Authentication and Authorization Service (JAAS) login configuration settings that are used to validate security tokens within incoming messages.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

3. Under Additional properties, click **Login mappings**.
4. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

To use this administrative console page for the application level, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
2. Under Modules, click **Manage modules > *URI_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Login mappings**.
6. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

Note: If the login mapping configuration is not found on the application level, the Web services run time searches for the login mapping configuration on the server level.

Authentication method: **Version 5.x application**

Specifies the method of authentication.

You can use any string, but the string must match the element in the service-level configuration. The following words are reserved and have special meanings:

BasicAuth
Uses both a user name and a password.

IDAssertion
Uses only a user name, but requires that additional trust is established on the receiving server using a TrustedIDEvaluator mechanism.

Signature
Uses the distinguished name (DN) of the signer.

LTPA Validates a token.

JAAS configuration name: **Version 5.x application**

Specifies the name of the Java Authentication and Authorization Service (JAAS) configuration.

Among the predefined system login configurations that you can use are the following:

system.wssecurity.IDAssertion
Enables a version 5.*x* application to use identity assertion to map a user name to a WebSphere Application Server credential principal.

system.wssecurity.Signature
Enables a version 5.*x* application to map a distinguished name (DN) in a signed certificate to a WebSphere Application Server credential principal.

system.LTPA_WEB
Processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSP) files.

system.WEB_INBOUND
Handles logins for Web application requests, which include servlets and JavaServer Pages. This login configuration is used by WebSphere Application Server Version 5.1.1.

system.RMI_INBOUND
Handles logins for inbound Remote Method Invocation (RMI) requests. This login configuration is used by WebSphere Application Server Version 5.1.1.

system.DEFAULT
Handles the logins for inbound requests made by internal authentications and most of the other protocols except Web applications and RMI requests. This login configuration is used by WebSphere Application Server Version 5.1.1.

system.RMI_OUTBOUND
Processes RMI requests that are sent outbound to another server when the com.ibm.CSIOutboundPropagationEnabled property is `true`. This property is set in the CSIv2 authentication panel. To access the panel, click **Security > Global security**. Expand RMI/IIOP security, then click on **CSIv2 Outbound authentication**. To set the com.ibm.CSIOutboundPropagationEnabled property, select **Security attribute propagation**.

Version 6 and later applications **system.wssecurity.X509BST**
Verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path.

Version 6 and later applications **system.wssecurity.PKCS7**
Verifies an X.509 certificate with a certificate revocation list in a PKCS7 object.

Version 6 and later applications **system.wssecurity.PkiPath**
Verifies an X.509 certificate with a public key infrastructure (PKI) path.

Version 6 and later applications **system.wssecurity.UsernameToken**
Verifies basic authentication (user name and password).

These system login configurations are defined on the System logins panel, which is accessible by completing the following steps:

1. Click **Security > Global security**.
2. Expand Java Authentication and Authorization Service, then click **System logins**.

Note: The predefined system login configurations are listed on the System logins configuration panel without the system prefix. For example, the system.wssecurity.UsernameToken configuration listed in the Java Authentication and Authorization Service (JAAS) configuration name option corresponds to the wssecurity.UsernameToken configuration that is on the System logins configuration panel.

You can use the following predefined application login configurations:

ClientContainer
Specifies the login configuration that is used by the client container application, which uses the CallbackHandler API that is defined in the deployment descriptor of the client container.

WSLogin
Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

DefaultPrincipalMapping
Specifies the login configuration used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries.

These application login configurations are defined on the Application logins panel, which is accessible by completing the following steps:

1. Click **Security > Global security**.
2. Expand Java Authentication and Authorization Service, then click **Application logins**.

Do not remove these predefined system or application login configurations. Within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module.

Callback handler factory class name: **Version 5.x application**

Specifies the name of the factory for the CallbackHandler class.

You must implement the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory class in this field.

Token type URI: **Version 5.x application**

Specifies the namespace Uniform Resource Identifiers (URI), which denotes the type of security token that is accepted.

If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace. If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.

If the reserved words are specified previously in the Authentication method field, this field is ignored.

Data type: Unicode characters except for non-ASCII characters, but including the number sign (#), the percent sign (%), and the square brackets ([ ]).

Token type local name: **Version 5.x application**

Specifies the local name of the security token type, for example, X509v3.

If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType attribute identifies the type of security token and its namespace. If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.

If the reserved words are specified previously in the Authentication method field, this field is ignored.

Nonce maximum age: **Version 5.x application**

Specifies the time, in seconds, before the nonce timestamp expires. Nonce is a randomly generated value.

You must specify a minimum of 300 seconds for the Nonce maximum age field. However, the maximum value cannot exceed the number of seconds specified in the Nonce cache timeout field for the server level.

You can specify the Nonce maximum age value for the server level by completing the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

Note: The Nonce maximum age field on this panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value: `Nonce is not supported for authentication methods other than BasicAuth.`

If you specify the BasicAuth method, but do not specify values for the Nonce maximum age field, the Web services security run time searches for a Nonce maximum age value on the server level.

Default	300 seconds
Range	300 to Nonce cache timeout seconds

Nonce clock skew: **Version 5.x application**

Specifies the clock skew value, in seconds, to consider when WebSphere Application Server checks the freshness of the message. Nonce is a randomly generated value.

You can specify the **Nonce clock skew** value for the server level by completing the following steps:

1. Click **Servers > Server Types > WebSphere application servers > *server_name***.
2. Under Security, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

You must specify a minimum of zero (0) seconds for the Nonce Clock Skew field. However, the maximum value cannot exceed the number of seconds that is specified in the Nonce maximum age field on this Login mappings panel.

Note: The Nonce clock skew field on this panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value: `Nonce is not supported for authentication methods other than BasicAuth.`

Note: If you specify BasicAuth, but do not specify values for the Nonce clock skew field, WebSphere Application Server searches for a Nonce clock skew value on the server level.

Default	0 seconds
Range	0 to Nonce Maximum Age seconds

Configuring the client for request signing: digitally signing message parts:

Version 5.x application

To configure the client for request signing, specify which message parts to digitally sign when configuring the client.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **Security Extensions** tab and the **Port Binding** tab in the Web Services Client Editor within an assembly tool.

- "Configuring the client security bindings using an assembly tool" on page 588
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 591

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to specify which message parts to digitally sign when configuring the client for request signing:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Click **Window > Open perspective > Other > J2EE**.
3. Click **Application Client projects > *application_name* > appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the **WS Extension** tab. The Client Deployment Descriptor is displayed.
5. Expand **Request sender configuration > Integrity**. *Integrity* refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet. For more information on digitally signing SOAP messages, see "XML digital signature" on page 536.
6. Indicate which parts of the message to sign by clicking **Add** and selecting **body**, **timestamp**, or **SecurityToken**. The following list contains descriptions of the message parts

   body The body is the user data portion of the message.

   timestamp
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If **timestamp** is selected, proceed to the next step and select **Add created time stamp** to add a time stamp to a message.

   SecurityToken
   The security token authenticates the client. If this option is selected, the message is signed.

   You can choose to digitally sign the message using a time stamp if **Add created time stamp** is selected and configured. You can digitally sign the message using a security token if a login configuration authentication method is selected.
7. Optional: Expand the **Add created time stamp** section and select this option if you want a time stamp added to the message. You can specify an expiration time for the time stamp, which helps defend against replay attacks. The lexical representation for duration is the [ISO 8601] extended format *PnYnMnDTnHnMnS*, where:
   - *nY* represents the number of years
   - *nM* represents the number of months
   - *nD* represents the number of days

- *T* is the date and time separator
- *nH* represents the number of hours
- *nM* represents the number of minutes
- *nS* represents the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is: `P1Y2M3DT10H30M`. Typically, you configure a message time stamp for about 10 to 30 minutes, for example, 10 minutes is represented as: `P0Y0M0DT0H10M0S`. The *P* character precedes time and date values.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web Services Client Editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the **Actor** fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **Actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server **Actor** fields are synchronized.

What to do next

After you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message. See “Configuring the client for request signing: choosing the digital signature method” for more information.

Configuring the client for request signing: choosing the digital signature method:

Version 5.x application

To configure the client for request signing, specify which message parts to digitally sign when configuring the client.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **Security extensions** tab and the **Port binding** tab in the Web services client editor within an assembly tool:

- "Configuring the client security bindings using an assembly tool" on page 588
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 591

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively. You must specify which parts of the message sent by the client must be digitally signed. See "Configuring the client for request signing: digitally signing message parts" on page 572 for more information.

About this task

Complete the following steps to specify which message parts to digitally sign when configuring the client for request signing:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **Application Client projects > *application_name* > appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the **WS Binding** tab. The Client Deployment Descriptor is displayed.
5. Expand **Security request sender binding configuration > Signing information**.
6. Select **Edit** to view the signing information and select a digital signature method from the **Signature method algorithm** field. The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web site http://www.w3.org/TR/xmldsig-core.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the `<SignedInfo>` element before the information is digested as part of the signature operation.
Digest method algorithm	Applies to the data after transforms are applied, if specified, to yield the `<DigestValue>` element. Signing the `<DigestValue>` element binds the resource content to the signer key. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.
Signature method algorithm	Converts the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.
Signing key name	Represents the key entry associated with the signing key locator. The key entry refers to an alias of the key, which is found in the key store and is used to sign the request.
Signing key locator	Represents a reference to a key locator implementation class that locates the correct hey store where the alias and the certificate exist.

7. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the **Digest method algorithm** and **Signature method algorithm** drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the WebSphere administrative console.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **Actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which method is used to digitally sign a message when the client sends a message to a server.

What to do next

After you configure the client to digitally sign the message, you must configure the server to verify the digital signature. See “Configuring the server for request digital signature verification: Verifying the message parts” for more information.

Configuring the server for request digital signature verification: Verifying the message parts:

Version 5.x application

Configure the server for request digital signature verification by modifying the extensions to indicate which parts of the request to verify.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **Extensions** tab and the **Binding Configurations** tab in the Web services editor within the assembly tools:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

You can use these two tabs to configure the Web services security extensions and the Web services security bindings, respectively. Also, you must specify which parts of the message sent by the client must be digitally signed. See "Configuring the client for request signing: digitally signing message parts" on page 572 to determine which message parts are digitally signed. The message parts specified for the client request sender must match the message parts specified for the server request receiver.

About this task

Complete the following steps to configure the server for request digital signature verification. The steps describe how to modify the extensions to indicate which parts of the request to verify.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab in the Web services editor.
6. Expand the **Request receiver service configuration details > Required integrity** section. Required integrity refers to the parts of the message that require digital signature verification. The purpose of digital signature verification is to make sure that the message parts have not been modified while transmitting across the Internet.
7. Indicate parts of the message to verify by clicking **Add**, and selecting one of the following three parts: **body**, **Timestamp**, or **SecurityToken**. You can determine which parts of the message to verify by looking at the Web service request sender configuration in the client application. To view the Web service request sender configuration information in the Web services client editor, click the Security extensions tab and expand **Request sender configuration > Integrity**. The following includes a list and description of the message parts:

   Body This is the user data portion of the message.

   Timestamp
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If **Timestamp** is selected, proceed to the next step to `Add Created Time Stamp` to the message.

   SecurityToken
   The security token authenticates the client. If **SecurityToken** is selected, the message is signed.
8. Optional: Expand the **Add received time stamp** section. The Add Received Time Stamp value indicates to validate the Add Created Time Stamp option configured by the client. You must select this option if you selected the Add Created Time Stamp on the client. The time stamp ensures message integrity by indicating the timeliness of the request. This option helps defend against replay attacks.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts are digitally signed and must be verified by the server when the client sends a message to a server.

What to do next

After you specify which message parts contain a digital signature that must be verified by the server, you must configure the server to recognize the digital signature method used to digitally sign the message. See “Configuring the server for request digital signature verification: choosing the verification method” for more information.

Configuring the server for request digital signature verification: choosing the verification method:

Version 5.x application

To configure the server for request digital signature verification, use an assembly tool to modify the extensions and indicate which digital signature method the server will use during verification.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding Configurations tab in the Web Services Editor within the IBM assembly tools:

- “Configuring the server security bindings using an assembly tool” on page 593

- “Configuring the server security bindings using the administrative console” on page 595

You can use these two tabs to configure the Web services security extensions and Web services security bindings, respectively. You must specify which message parts contain digital signature information that must be verified by the server. See “Configuring the server for request digital signature verification: Verifying the message parts” on page 576. The message parts specified for the client request sender must match the message parts specified for the server request receiver. Likewise, the digital signature method chosen for the client must match the digital signature method used by the server.

About this task

Complete the following steps to configure the server for request digital signature verification. The steps describe how to modify the extensions to indicate which digital signature method the server will use during verification.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Binding Configurations** tab.
6. Expand the **Security request receiver binding configuration details > Signing information** section.
7. Click **Edit** to edit the signing information. The signing information dialog is displayed, select or enter the following information:
   - Canonicalization method algorithm
   - Digest method algorithm
   - Signature method algorithm
   - Use certificate path reference
   - Trust anchor reference
   - Certificate store reference
   - Trust any certificate

   For more conceptual information on digitally signing SOAP messages, see XML digital signature. The following table describes the purpose for each of these selections. Some of the following definitions are based on the XML-Signature specification, which is located at the following Web address: http://www.w3.org/TR/xmldsig-core.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the `<SignedInfo>` element before it is digested as part of the signature operation. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration.
Digest method algorithm	Applies to the data after transforms are applied, if specified, to yield the `<DigestValue>` element. The signing of the `<DigestValue>` element binds resource content to the signer key. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration.
Signature method algorithm	Converts the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration.

Name	Purpose
Use certificate path reference or **Trust any certificate**	Validates a certificate or signature sent with a message. When a message is signed, the public key used to sign it is sent with the message. This public key or certificate might not be validated at the receiving end. By selecting **User certificate path reference**, you must configure a trust anchor reference and a certificate store reference to validate the certificate sent with the message. By selecting **Trust any certificate**, the signature is validated by the certificate sent with the message without the certificate itself being validated.
Use certificate path reference: Trust anchor reference	Refers to a key store that contains trusted, self-signed certificates and certificate authority (CA) certificates. These certificates are trusted certificates that you can use with any applications in your deployment.
Use certificate path reference: Certificate store reference	Contains a collection of X.509 certificates. These certificates are not trusted for all applications in your deployment, but might be used as an intermediary to validate certificates for an application.

8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm dropdown lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations on the client:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the Actor field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified the method that the server uses to verify the digital signature in the message parts.

What to do next

After you configure the client for request signing and the server for request digital signature verification, you must configure the server and the client to handle the response. Next, specify the response signing for the server. See “Configuring the server for response signing: digitally signing message parts” for more information.

Configuring the server for response signing: digitally signing message parts:

Version 5.x application

Use an assembly tool to specify which message parts to digitally sign when configuring the server for response signing.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **Extensions** tab and the **Binding** configurations tab in the Web services editor within the IBM assembly tools:

- “Configuring the server security bindings using an assembly tool” on page 593
- “Configuring the server security bindings using the administrative console” on page 595

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to specify which message parts to digitally sign when configuring the server for response signing:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >** ***application _name*** **>ejbModule > META-INF**.
4. Right-click the `webservices.xml` file and click **Open with > Web services editor**.
5. Click the **Extension**s tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender service configuration details > Integrity**. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet. For more information on digitally signing SOAP messages, see XML digital signature.
7. Indicate the parts of the message to sign by clicking **Add**, and selecting **Body**, **Timestamp**, or **Securitytoken**.

   The following list contains descriptions of the message parts:

   Body The body is the user data portion of the message.

   Timestamp
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If this option if selected, proceed to the next step and click **Add Created Time Stamp**, which indicates that the time stamp is added to the message.

Securitytoken
The security token is used for authentication. If this option is selected, the authentication information is added to the message.

8. Optional: Expand the **Add created time stamp** section. Select this option if you want a time stamp added to the message. You can specify an expiration time for the time stamp, which helps defend against replay attacks. The lexical representation for duration is the ISO 8601 extended format, *PnYnMnDTnHnMnS*, where:
   - *nY* represents the number of years.
   - *nM* represents the number of months.
   - *nD* represents the number of days.
   - *T* is the date and time separator.
   - *nH* represents the number of hours.
   - *nM* represents the number of minutes.
   - *nS* represents the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

   For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is: `P1Y2M3DT10H30M`. Typically, you configure a message time stamp for about 10 to 30 minutes. 10 minutes is represented as: `P0Y0M0DT0H10M0S`. The *P* character precedes time and date values.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations:
- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations:
- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts to digitally sign when the server sends a response to the client.

What to do next

After you specifying which message parts to digitally sign, you must specify which method is used to digitally sign the message. See “Configuring the server for response signing: choosing the digital signature method” on page 583 for more information.

Configuring the server for response signing: choosing the digital signature method:

Version 5.x application

Use an assembly tool to specify which digital signature method to use when configuring the server for response signing.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the **Binding configurations** tab in the Web services editor within the IBM assembly tools:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to specify which digital signature method to use when configuring the server for response signing:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects > *application _name* >ejbModule > META-INF**.
4. Right-click the `webservices.xml` file and click **Open with > Web services editor**.
5. Click the **Binding Configurations** tab.
6. Expand **Response sender binding configuration details > Signing information**.
7. Click **Edit** to choose a signing method. The signing info dialog is displayed and either select or enter the following information:
   - **Canonicalization method algorithm**
   - **Digest method algorithm**
   - **Signature method algorithm**
   - **Signing key name**
   - **Signing key locator**

   The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following address: http://www.w3.org/TR/xmldsig-core.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the <SignedInfo> element before the information is digested as part of the signature operation. Use the same algorithm on the client response receiver. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

Name	Purpose
Digest method algorithm	Applies to the data after transforms are applied, if specified, to yield the <DigestValue> element. Signing the <DigestValue> element binds resource content to the signer key. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.
Signature method algorithm	Converts the canonicalized <SignedInfo> element into the <SignatureValue> element. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.
Signing key name	Represents the key entry associated with the signing key locator. The key entry refers to an alias of the key, which is found in the key store and is used to sign the request.
Signing key locator	Represents a reference to a key locator implementation class that locates the correct key store where the alias and certificate exists. For more information on configuring key locators, see any of the following files: • "Configuring key locators using an assembly tool" on page 560 • "Configuring key locators using the administrative console" on page 561

8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm dropdown lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Results

You have specified which method is used to digitally sign a message when the server sends a message to a client.

What to do next

After you configure the server to digitally sign the response message, you must configure the client to verify the digital signature contained in the response message. See "Configuring the client for response digital signature verification: verifying the message parts" for more information.

Configuring the client for response digital signature verification: verifying the message parts:

Version 5.x application

To configure the Web services security extensions and the Web services security bindings, use the **WS Extension** tab and the **WS Binding** tab in the Client Deployment Descriptor within an assembly tool.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to becomes familiar with the **WS Extension** tab and the **WS Binding** tab in the Client Deployment Descriptor within the assembly tool:

- "Configuring the client security bindings using an assembly tool" on page 588
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 591

You can use these two tabs to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to configure the client for response digital signature verification. The steps describe how to modify the extensions to indicate which parts of the response to verify.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **Application Client projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file and click **Open With > Deployment descriptor editor**.
5. Click the **WS extension** tab.
6. Expand the **Response receiver configuration > Required integrity** section. Required integrity refers to parts that require digital signature verification. Digital signature verification decreases the risk that the message parts have been modified while the message is transmitted across the Internet.
7. Indicate the parts of the message that must be verified. You can determine which parts of the message to verify by looking at the Web service response sender configuration. Click **Add** and select one of the following parts:

   Body The body is the user data portion of the message.

   Timestamp
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If the timestamp option is selected, proceed to the next step to add a received time stamp to the message.

   Securitytoken
   The security token authenticates the client. If the Securitytoken option is selected, the message is signed.
8. Optional: Expand the **Add received time stamp** section. Select **Add received time stamp** to add the received time stamp to the message.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.

- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the Actor field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts are digitally signed and must be verified by the client when the server sends a response message to the client.

What to do next

After you specify which message parts contain a digital signature that must be verified by the client, you must configure the client to recognize the digital signature method used to digitally sign the message. See "Configuring the client for response digital signature verification: choosing the verification method" for more information.

Configuring the client for response digital signature verification: choosing the verification method: **Version 5.x application**

You can configure the Web services security extensions and Web services security bindings using the **WS extension** tab and the **WS binding** tab in the Web services editor within an assembly tool.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **WS extension** tab and the **WS binding** tab in the Web services editor within the IBM assembly tools:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

You can use these two tabs to configure the Web services security extensions and Web services security bindings, respectively. Also, you must specify which message parts contain digital signature information that must be verified by the client. See "Configuring the client for response digital signature verification: verifying the message parts" on page 584 to specify which message parts are digitally signed by the server and must be verified by the client. The message parts specified for the server response sender must match the message parts specified for the client response receiver. Likewise, the digital signature method chosen for the server must match the digital signature method used by the client.

About this task

Complete the following steps to configure the client for response digital signature verification. The steps describe how to modify the extensions to indicate which digital signature method the client will use during verification.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open perspective > Other > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Binding** tab.
6. Expand the **Security response receiver binding configuration > Signing information** section.
7. Click **Edit** to select a digital signature method. The signing info dialog displays and either select or enter the following information:
   - **Canonicalization method algorithm**
   - **Digest method algorithm**
   - **Signature method algorithm**
   - **Signing key name**
   - **Signing key locator**

   For more conceptual information on digitally signing SOAP messages, see XML digital signature. The following table describes the purpose for each of these selections. Some of the following definitions are based on the XML-Signature specification, which can be found at: http://www.w3.org/TR/xmldsig-core.

Name	Purpose
Canonicalization method algorithm	The canonicalization method algorithm is used to canonicalize the `<SignedInfo>` element before it is digested as part of the signature operation.
Digest method algorithm	The digest method algorithm is the algorithm applied to the data after transforms are applied, if specified, to yield the `<DigestValue>`. The signing of the `<DigestValue>` binds resource content to the signer key. The algorithm selected for the client response receiver configuration must match the algorithm selected in the server response sender configuration.
Signature method algorithm	The signature method is the algorithm that is used to convert the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the client response receiver configuration must match the algorithm selected in the server response sender configuration.
Use certificate path reference or **Trust any certificate**	When a message is signed, the public key used to sign it is transmitted with the message. To validate this public key at the receiving end, configure a certificate path reference. By selecting **User certificate path reference,** you must configure a trust anchor reference and certificate store reference to validate the certificate sent with the message. By selecting **trust any certificate**, the signature is validated by the certificate sent with the message without the certificate itself being validated.
Use certificate path reference: Trust anchor reference	A trust anchor is a configuration that refers to a keystore that contains trusted, self-signed certificates and certificate authority (CA) certificates. These certificates are trusted certificates that you can use with any applications in your deployment.
Use certificate path reference: Certificate store reference	A certificate store is a configuration that has a collection of X.509 certificates. These certificates are not trusted for all applications in your deployment, but might be used as an intermediary to validate certificates for an application.

8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm dropdown lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set

the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Results

Note: If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when running the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which method the client uses to verify the digital signature in the message parts.

What to do next

After you configure the server for response signing and the client for request digital signature verification, verify that you have configured the client and the server to handle the message request.

Configuring the client security bindings using an assembly tool:

Version 5.x application

Use the Web services client editor within an assembly tool to include the binding information, that describes how to run the security specifications found in the extensions, in the client enterprise archive (EAR) file.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

When configuring a client for Web services security, the bindings describe how to run the security specifications found in the extensions. Use the Web services client editor within an assembly tool to include the binding information in the client enterprise archive (EAR) file.

You can configure the client-side bindings from a pure client accessing a Web service or from a Web service accessing a downstream Web service. This document focuses on the pure client situation. However, the concepts, and in most cases the steps, also apply when a Web service is configured to communicate downstream to another Web service that has client bindings. Complete the following steps to edit the security bindings on a pure client (or server acting as a client) using an assembly tool:

1. Import the Web services client EAR file into an assembly tool. When you edit the client bindings on a server acting as a client, the same basic steps apply. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects > *application_name* > appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**. The Client Deployment Descriptor is displayed.
5. Click the **WS Extension** tab and select the port QName bindings that you want to configure. The Web services security extensions are configured for outbound requests and inbound responses. You need to configure the following information for Web services security extensions. These topics are discussed in more detail in other sections of the documentation.

   Request sender configuration details

   Details
   "Configuring the client for request signing: digitally signing message parts" on page 572

   Integrity
   "Configuring the client for request signing: digitally signing message parts" on page 572

   Confidentiality
   "Configuring the client for request encryption: Encrypting the message parts" on page 605

   Login Config

   BasicAuth
   "Configuring the client for basic authentication: specifying the method" on page 622

   IDAssertion
   "Configuring the client for identity assertion: specifying the method" on page 630

   Signature
   "Configuring the client for signature authentication: specifying the method" on page 637

   LTPA "Configuring the client for LTPA token authentication: specifying LTPA token authentication" on page 650

   ID assertion
   "Configuring the client for identity assertion: specifying the method" on page 630

   Add created time stamp
   "Configuring the client for request signing: digitally signing message parts" on page 572

   Response receiver configuration details

   Required integrity
   "Configuring the client for response digital signature verification: verifying the message parts" on page 584

   Required confidentiality
   "Configuring the client for response decryption: decrypting the message parts" on page 618

Add received time stamp
"Configuring the client for response digital signature verification: verifying the message parts" on page 584

6. Click the **WS binding** tab and select the port qualified name binding that you want to configure. The Web services security bindings are configured for outbound requests and inbound responses. You need to configure the following information for Web services security bindings. These topics are discussed in more details in other sections of the documentation.

   Security request sender binding configuration

   Signing information
   "Configuring the client for request signing: choosing the digital signature method" on page 574

   Encryption information
   "Configuring the client for request encryption: choosing the encryption method" on page 606

   Key locators
   "Configuring key locators using an assembly tool" on page 560

   Login binding

   BasicAuth
   "Configuring the client for basic authentication: collecting the authentication information" on page 623

   ID assertion
   "Configuring the client for identity assertion: collecting the authentication method" on page 632

   Signature
   "Configuring the client for signature authentication: collecting the authentication information" on page 639

   LTPA "Configuring the client for LTPA token authentication: collecting the authentication method information" on page 651

   Security response receiver binding configuration

   Signing information
   "Configuring the client for response digital signature verification: choosing the verification method" on page 586

   Encryption information
   "Configuring the client for response decryption: choosing a decryption method" on page 619

   Trust anchor
   "Configuring trust anchors using an assembly tool" on page 548

   Certificate store list
   "Configuring the client-side collection certificate store using an assembly tool" on page 553

   Key locators
   "Configuring key locators using an assembly tool" on page 560

What to do next

Note: When configuring the security request sender binding configuration, you must synchronize the information used to perform the specified security with the security request receiver binding configuration, which is configured in the server EAR file. These two configurations must be synchronized in all respects because there is no negotiation during run time to determine the requirements of the server.

For example, when configuring the encryption information in the security request sender binding Configuration, you must use the public key from the server for encryption. Therefore, the key locator that

you choose must contain the public key from the server configuration. The server must contain the private key to decrypt the message. This example illustrates the important relationship between the client and server configuration. Additionally, when configuring the security response receiver binding configuration, the server must send the response using security information known by this client security response receiver binding configuration.

The following table shows the related configurations between the client and the server. The client request sender and the server request receiver are relative configurations that must be synchronized with each other. The server response sender and the client response receiver are related configurations that must be synchronized with each other. Note that the related configurations are end points for any request or response. One end point must communicate its actions with the other end point because run time requirements are not negotiated.

Table 68. Related configurations

Client configuration	Server configuration
Request sender	Request receiver
Response receiver	Response sender

Configuring the security bindings on a server acting as a client using the administrative console:

Version 5.x application

Use the Web services client editor within an assembly tool to include the binding information, that describes how to run the security specifications found in the extensions, in the client enterprise archive (EAR) file.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

When configuring a client for Web services security, the bindings describe how to run the security specifications found in the extensions. Use the Web services client editor within an assembly tool to include the binding information in the client enterprise archive (EAR) file.

You can configure the client-side bindings from a pure client accessing a Web service or from a Web service accessing a downstream Web service. Complete the following steps to find the location in which to edit the client bindings from a Web service that is running on the server. When a Web service communicates with another Web service, you must configure client bindings to access the downstream Web service.

1. Deploy the Web service using the WebSphere Application Server administrative console. Click **Applications > Install New Application**.

   You can access the administrative console by typing `http://server_name:port_number/ibm/console` in your Web browser unless you have changed the port number.

   See also Installing a new application.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Manage modules, click ***URI_name***.
4. Under Web Services Security Properties, click **Web Services: Client security bindings**. A table displays with the following columns:
   - Component Name

- Port
- Web Service
- Request Sender Binding
- Request Receiver Binding
- HTTP Basic Authentication
- HTTP SSL Configuration

For Web services security, you must edit the request sender binding and response receiver binding configurations. You can use the defaults for some of the information at the server level. Default bindings are convenient because you can configure commonly reused elements such as key locators once and then reference their aliases in the application bindings.

5. View the default bindings for the server using the administrative console by clicking **Servers > Server Types > WebSphere application servers >** ***server_name*** . Under Additional Properties, click **JAX-WS and JAX-RPC security runtime**.

   Note: In a mixed node cell with a server using Websphere Application Server version 6.1 or earlier, click **Web services: Default bindings for Web services security**.

   You can configure the following sections. These topics are discussed in more detail in other sections of the documentation.

   - Request sender binding
     - "Signing parameter configuration settings" on page 538
     - "Encryption information configuration settings: Methods" on page 389
     - "Key locator configuration settings" on page 466
     - "Login bindings configuration settings" on page 601
   - Response receiver binding
     - "Signing information configuration settings" on page 352
     - "Encryption information configuration settings: Message parts" on page 382
     - "Trust anchor configuration settings" on page 477
     - "Collection certificate store configuration settings" on page 484
     - "Key locator configuration settings" on page 466

What to do next

Note: When configuring the security request sender binding configuration, you must synchronize the information used to perform the specified security with the security request receiver binding configuration, which is configured in the server EAR file. These two configurations must be synchronized in all respects because there is no negotiation during run time to determine the requirements of the server. For example, when configuring the encryption information in the security request sender binding configuration, you must use the public key from the server for encryption. Therefore, the key locator that you choose must contain the public key from the server configuration. The server must contain the private key to decrypt the message. This example illustrates the important relationship between the client and server configuration. Additionally, when configuring the security response receiver binding configuration, the server must send the response using security information known by this client security response receiver binding configuration.

The following table shows the related configurations between the client and the server. The client request sender and the server request receiver are relative configurations that must be synchronized with each other. The server response sender and the client response receiver are related configurations that must be synchronized with each other. Note that related configurations are end points for any request or response. One end point must communicate its actions with the other end point because run time requirements are not required.

Table 69. Related configurations

Client configuration	Server configuration
Request sender	Request receiver
Response receiver	Response sender

Configuring the server security bindings using an assembly tool:

Version 5.x application

Use an assembly tool to edit bindings for a Web service after these bindings are deployed on a server.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to importing the Web services enterprise archive (EAR) file into the assembly tool, make sure that you have already run the `wsdl2java` command on your Web service to enable your Java Platform, Enterprise Edition (Java EE) application. You must import the Web services EAR file into the assembly tool.

About this task

Create an Enterprise JavaBeans (EJB) file Java archive (JAR) file or a Web archive (WAR) file containing the security binding file (`ibm-webservices-bnd.xmi`) and the security extension file (`ibm-webservices-ext.xmi`). If this archive is acting as a client to a downstream service, you also need the client-side binding file (`ibm-webservicesclient-bnd.xmi`) and the client-side extension file (`ibm-webservicesclient-ext.xmi`). These files are generated using the WSDL2Java command for JAX-RPC applications. You can edit these files using the Web services editor in the assembly tool.

When configuring server-side security for Web services security, the security extensions configuration specifies what security is performed, the security bindings configuration indicates how to perform what is specified in the security extensions configuration. You can use the defaults for some elements at the cell and server levels in the bindings configuration, including key locators, trust anchors, the collection certificate store, trusted ID evaluators, and login mappings and reference these elements from the WAR and JAR binding configurations.

Open the Web services editor in an assembly tool to begin editing the server security extensions and bindings. The following steps can locate the server security extensions and bindings. Other tasks specify how to configure each section of the extensions and bindings in more detail.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java EE perspective. Click **Window > Open Perspective > J2EE**.
3. Configure the server for inbound requests and outbound responses security configuration. To configure the server for inbound requests and outbound responses, complete the following steps:
   a. Click **EJB Projects > *application_name* > ejbModule > META-INF**.
   b. Right-click the `webservices.xml` file and click **Open with > Web services editor**. The `webservices.xml` file represents the server-side (inbound) Web services configuration. The `webservicesclient.xml` file represents the client-side (outbound) Web services configuration.
4. In the Web services editor (for the `webservices.xml` file and inbound requests and outbound responses Web services configuration), there are several tabs at the bottom of the editor including Web Services,

Port Components, Handlers, Security Extensions, Bindings, and Binding Configurations. The security extensions are edited using the **Security Extensions** tab. The security bindings are edited using the Security Bindings tab.

a. Click the **WS Extensions** tab and select the port component binding to edit. The Web services security extensions are configured for inbound requests and outbound responses. You need to configure the following information for Web services security extensions. These topics are discussed in more detail in other topics in the documentation.

   Request receiver service configuration details

   Required integrity
   "Configuring the server for request digital signature verification: Verifying the message parts" on page 576

   Required confidentiality
   "Configuring the server for request decryption: decrypting the message parts" on page 610

   Login config

   BasicAuth
   "Configuring the server to handle basic authentication information" on page 626

   ID assertion
   "Configuring the server to handle identity assertion authentication" on page 633

   Signature
   "Configuring the server to support signature authentication" on page 640

   LTPA "Configuring the server to handle LTPA token authentication information" on page 652

   Add received time stamp
   "Configuring the server for request digital signature verification: Verifying the message parts" on page 576

   Response sender service configuration details

   Details
   "Configuring the server for response signing: digitally signing message parts" on page 581

   Integrity
   "Configuring the server for response signing: digitally signing message parts" on page 581

   Confidentiality
   "Configuring the server for response encryption: encrypting the message parts" on page 614

   Add created time stamp
   "Configuring the server for response signing: digitally signing message parts" on page 581

b. Click the **Binding Configurations** tab and select the port component binding to edit. The Web services security bindings are configured for inbound requests and outbound responses. You need to configure the following information for Web services security bindings. These topics are discussed in more details in other topics in the documentation.

   Response receiver binding configuration details

   Signing Information
   "Configuring the server for request digital signature verification: choosing the verification method" on page 578

   Encryption Information
   "Configuring the server for request decryption: choosing the decryption method" on page 611

Trust Anchor
"Configuring trust anchors using an assembly tool" on page 548

Certificate Store List
"Configuring the server-side collection certificate store using an assembly tool" on page 554

Key Locators
"Configuring key locators using an assembly tool" on page 560

Login Mapping

Basic auth
"Configuring the server to validate basic authentication information" on page 627

ID assertion
"Configuring the server to validate identity assertion authentication information" on page 634

Signature
"Configuring the server to validate signature authentication information" on page 641

LTPA "Configuring the server to validate LTPA token authentication information" on page 653

Trusted ID evaluator

Trusted ID evaluator reference

Response sender binding configuration details

Signing information
"Configuring the server for response signing: choosing the digital signature method" on page 583

Encryption information
"Configuring the server for response encryption: choosing the encryption method" on page 615

Key locators
"Configuring key locators using an assembly tool" on page 560

What to do next

Configure the client for outbound requests and inbound responses security configuration by right-clicking the `webservicesclient.xml` file and clicking **Open With > Deployment descriptor editor**. For more information, see "Configuring the client security bindings using an assembly tool" on page 588.

Configuring the server security bindings using the administrative console:

Version 5.x application

Use the WebSphere Application Server administrative console to edit bindings for a Web service after these bindings are deployed on a server.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Create an Enterprise JavaBeans (EJB) file Java archive (JAR) file or Web archive (WAR) file containing the security binding file (`ibm-webservices-bnd.xmi`) and the security extension file (`ibm-webservices-ext.xmi`). If this archive is acting as a client to a downstream service, you also need the client-side binding file (`ibm-webservicesclient-bnd.xmi`) and the client-side extension file (`ibm-webservicesclient-ext.xmi`). These files are generated using the WSDL2Java command for JAX-RPC applications command. You can edit these files using the Web Services Editor in the Assembly tools.

When configuring server-side security for Web services security, the security extensions configuration specifies what security is to be performed while the security bindings configuration indicates how to perform what is specified in the security extensions configuration. You can use the defaults for some elements at the cell and server levels in the bindings configuration, including key locators, trust anchors, the collection certificate store, trusted ID evaluators, and login mappings and reference them from the WAR and JAR binding configurations.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you also must configure the client bindings to access the downstream Web service.

1. Deploy the Web service using the WebSphere Application Server administrative console.

   Type `http://`*`server_name`*`:`*`port_number`*`/ibm/console` in your Web browser unless you have changed the port number.

   After you log into the administration console, click **Applications > Install new application** to deploy the Web service. For more information, see Installing enterprise application files with the console.
2. After you deploy the Web service, click **Applications > Enterprise applications >** ***application_name*** .
3. Under Manage modules, click ***URI_name***.
4. Under Web Services Security Properties, click **Web services: client security bindings** for outbound requests and inbound responses. Click **Web services: server security bindings** for inbound requests and outbound responses.
5. If you click **Web services: server security bindings**, the following sections can be configured. These topics are discussed in more detail in other sections of the documentation.
   - Request receiver binding
     - Signing information
     - Encryption information
     - Trust anchors
     - Collection certificate store
     - Key locator
     - Trusted ID evaluator
     - Login mappings
   - Response sender binding
     - Signing parameters
     - Encryption information
     - Key locator

XML encryption

Version 5.x application

Extensible Markup Language (XML) encryption is a specification developed by World Wide Web (WWW) Consortium (W3C) in 2002 that contains the steps to encrypt data, the steps to decrypt encrypted data,

the syntax to represent XML encrypted data, the information used to decrypt the data, and a list of encryption algorithms such as triple Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Rivest-Shamir-Adleman algorithm (RSA).

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

You can apply XML encryption to an XML element, XML element content, and arbitrary data, including an XML document. For example, suppose that you need to encrypt the <CreditCard> element shown in the example 1.

Example 1: Sample XML document

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Example 2: XML document with a common secret key

Example 2 shows the XML document after encryption. The <EncryptedData> element represents the encrypted <CreditCard> element. The <EncryptionMethod> element describes the applied encryption algorithm, which is triple DES in this example. The <KeyInfo> element contains the information to retrieve a decryption key, which is a <KeyName> element in this example. The <CipherValue> element contains the ciphertext obtained by serializing and encrypting the <CreditCard> element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
       Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
       <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
         <KeyName>John Smith</KeyName>
       </KeyInfo>
       <CipherData>
         <CipherValue>ydUNqHkMrD...</CipherValue>
       </CipherData>
  </EncryptedData>
</PaymentInfo>
```

Example 3: XML document encrypted with the public key of the recipient

In example 2, it is assumed that both the sender and recipient have a common secret key. If the recipient has a public and private key pair, which is most likely the case, the <CreditCard> element can be encrypted as shown in example 3. The <EncryptedData> element is the same as the <EncryptedData> element found in Example 2. However, the <KeyInfo> element contains an EncryptedKey.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
```

```
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

XML Encryption in the WSS-Core

WSS-Core specification is under development by Organization for the Advancement of Structured Information Standards (OASIS). The specification describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The message confidentiality is realized by encryption based on XML Encryption.

The WSS-Core specification supports encryption of any combination of body blocks, header blocks, their sub-structures, and attachments of a SOAP message. The specification also requires that when you encrypt parts of a SOAP message, you prepend a reference from the security header block to the encrypted parts of the message. The reference can be a clue for a recipient to identify which encrypted parts of the message to decrypt.

The XML syntax of the reference varies according to what information is encrypted and how it is encrypted. For example, suppose that the <CreditCard> element in example 4 is encrypted with either a common secret key or the public key of the recipient.

Example 4: Sample SOAP message

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The resulting SOAP messages are shown in Examples 5 and 6. In these example, the <ReferenceList> and <EncryptedKey> elements are used as references, respectively.

Example 5: SOAP message encrypted with a common secret key

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
```

```
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 6: SOAP message encrypted with the public key of the recipient

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Relationship to digital signature

The WSS-Core specification also provides message integrity, which is realized by a digital signature based on the XML-Signature specification.

A combination of encryption and digital signature over common data introduces cryptographic vulnerabilities.

Securing Web services for Version 5.x applications using XML encryption

Version 5.x application

XML encryption is one method that WebSphere Application Server provides to secure your Web services. It enables you to encrypt an XML element, the content of an XML element, or arbitrary data such as an XML document.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides several different methods to secure your Web services. XML encryption is one of these methods. You can secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

About this task

XML encryption enables you to encrypt an XML element, the content of an XML element, or arbitrary data such as an XML document. Like XML digital signature, a message is sent by the client as the request sender to the server as the request receiver. The response is sent by the server as the response sender to the client as the request receiver. Unlike XML digital signature, which verifies the authenticity of the sender, XML encryption scrambles the message content using a key, which can be unscrambled by a receiver that possesses the same key. You can use XML encryption in conjunction with XML digital signature to scramble the content while verifying the authenticity of the message sender.

To use XML encryption to secure Web services, you must use an assembly tool. For more information, see the related information on Assembly Tools.

To securing Web services for Version 5.*x* applications using XML encryption, complete the following steps:

1. Specify the encryption settings for the request sender. The message parts and the encryption method settings chosen for the request sender on the client must match the message parts and the method settings chosen for the request receiver on the server. To specify the encryption settings for the request sender:
   a. “Configuring the client for request encryption: Encrypting the message parts” on page 605.
   b. “Configuring the client for request encryption: choosing the encryption method” on page 606.
2. Specify the encryption settings for the request receiver. The decryption settings chosen for the request receiver must match the encryption settings chosen for the request sender.
   To specify the decryption settings for the request receiver:
   a. “Configuring the server for request decryption: decrypting the message parts” on page 610.
   b. “Configuring the server for request decryption: choosing the decryption method” on page 611.

3. Specify the encryption settings for the response sender. The message parts and the encryption method settings chosen for the response sender on the server must match the message parts and the method settings chosen for the response receiver on the client. To specify the encryption settings for the response sender:
   a. "Configuring the server for response encryption: encrypting the message parts" on page 614.
   b. "Configuring the server for response encryption: choosing the encryption method" on page 615.
4. Specify the encryption settings for the response receiver.

   Note: The decryption settings chosen for the response receiver must match the encryption settings chosen for the response sender.

   To specify the decryption settings for the response receiver, complete the following steps:
   a. "Configuring the client for response decryption: decrypting the message parts" on page 618.
   b. "Configuring the client for response decryption: choosing a decryption method" on page 619.

Results

After completing these steps, you have secured your Web services using XML encryption.

Login bindings configuration settings: **Version 5.x application**

Use this page to configure the encryption and decryption parameters.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The pluggable token uses the Java Authentication and Authorization Service (JAAS) CallBackHandler (javax.security.auth.callback.CallBackHandler) interface to generate the token that is inserted into the message. The following list describes the CallBack support implementations:

com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
This implementation is used for generating binary tokens inserted as `<wsse:BinarySecurityToken/@ValueType>` in the message.

javax.security.auth.callback.NameCallback and javax.security.auth.callback.PasswordCallback
This implementation is used for generating user name tokens inserted as `<wsse:UsernameToken>` in the message.

com.ibm.wsspi.wssecurity.auth.callback.XMLTokenSenderCallback
This implementation is used to generate Extensible Markup Language (XML) tokens and is inserted as the `<SAML: Assertion>` element in the message.

com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback
This implementation is used to obtain properties that are specified in the binding file.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_file_name***. Under Web Services Security Properties, click **Web Services: Client security bindings**.
3. Under Request Sender Bindings, click **Edit**.
4. Under Additional properties, click **Login binding**.

If the encryption information is not available, select **None**.

If the encryption information is available, select **Dedicated login binding** and specify the configuration in the following fields:

Authentication method:

Specifies the unique name for the authentication method.

You can uses any string to name the authentication method. However, the string must match the element in the server-level configuration. The following words are reserved by WebSphere Application Server:

BasicAuth
This method uses both a user name and a password.

IDAssertion
This method uses a user name, but it requires that additional trust is established by the receiving server using a trusted ID evaluator mechanism.

Signature
This method uses the distinguished name (DN) of the signer.

LTPA This method validates the token.

Callback handler:

Specifies the name of the callback handler. The callback handler must implement the javax.security.auth.callback.CallbackHandler interface.

Basic authentication user ID:

Specifies the user name for basic authentication. With the basic authentication method, you can define a user name and a password in the binding file.

Basic authentication password:

Specifies the password for basic authentication.

Token type URI:

Specifies the namespace Uniform Resource Identifiers (URI), which denotes the type of security token that is accepted.

The value of this field if is impacted by the following conditions:

- If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace.
- If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.
- The Token type URI field is ignored if the reserved words, which are listed in the description of the Authentication method field, are specified.

This information is inserted as `<wsse:BinarySecurityToken>/ValueType` for the <SAML: Assertion> XML token.

Token type local name:

Specifies the local name of the security token type. For example, X509v3.

The value of this field if is impacted by the following conditions:

- If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace.
- If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.
- The Token type URI field is ignored if the reserved words, which are listed in the description of the Authentication method field, are specified.

This information is inserted as `<wsse:BinarySecurityToken>/ValueType` for the <SAML: Assertion> XML token.

Request sender: **Version 5.x application**

The security handler on the request sender side of the SOAP message enforces the security constraints, located in the `ibm-webservicesclient-ext.xmi` file, and bindings, located in the `ibm-webservicesclient-bnd.xmi file`. These constraints and bindings apply both to Java Platform, Enterprise Edition (Java EE) application clients or when Web services is acting as a client. The security handler acts on the security constraints before sending the SOAP message. For example, the security handler might digitally sign the message, encrypt the message, create a time stamp, or insert a security token.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

The security handler on the request sender side of the SOAP message enforces the security constraints, located in the `ibm-webservicesclient-ext.xmi` file, and the bindings, located in the `ibm-webservicesclient-bnd.xmi` file. These constraints and bindings apply both to Java Platform, Enterprise Edition (Java EE) application clients or when Web services is acting as a client. The security handler acts on the security constraints before sending the SOAP message. Request sender security constraints must match the security constraint requirements defined in the request receiver. For example, the security handler might digitally sign the message, encrypt the message, create a time stamp, or insert a security token. You can specify the following security requirements for the request sender and apply them to the SOAP message:

Integrity (digital signature)

You can select multiple parts of a message to sign digitally. The following list contains the integrity options:

- Body
- Time stamp
- Security token

Confidentiality (encryption)

You can select multiple parts of a message to encrypt. The following list contains the confidentiality options:

- Body content
- Username token

Security token

You can insert only one token into the message. The following list contains the security token options:

- Basic authentication, which requires both a user name and a password
- Identity assertion, which requires a user name only
- X.509 binary security token
- Lightweight Third Party Authentication (LTPA) binary security token

- Custom token , which is pluggable and supports custom-defined tokens in the SOAP message

Timestamp

You can have a time stamp to indicate the timeliness of the message.

- Timestamp

Request sender binding collection: **Version 5.x application**

Use this page to specify the binding configuration to send request messages for Web services security.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_file_name***.
3. Under Web Services Security Properties, click **Web services: Client security bindings**.
4. Under Request sender binding, click **Edit**.

Web services security namespace: **Version 5.x application** Specifies the namespace that is used by Web services security to send a request. However, this field configures the namespace value only and does not enforce the semantics of the specification related to the namespace. Web services security uses the processing semantic only in draft 13 of the OASIS specification. The following schemas are available:

- http://schemas.xmlsoap.org/ws/2003/06/secext
- http://schemas.xmlsoap.org/ws/2002/07/secext
- http://schemas.xmlsoap.org/ws/2002/04/secext
- None

The namespace used by the response sender is based on the namespace of the incoming message in the request receiver.

Signing information: **Version 5.x application**

Specifies the configuration for the signing parameters. Signing information is used to sign and validate parts of the message including the body and time stamp.

You can also use these parameters for X.509 validation when the Authentication method is `IDAssertion` and the ID Type is `X509Certificate`, in the server-level configuration. In such cases, you must fill in the Certificate Path fields only.

Encryption information: **Version 5.x application**

Specifies the configuration for the encrypting and decrypting parameters. Encryption information is used for encrypting and decrypting various parts of a message, including the body and user name token.

Key locators: **Version 5.x application**

Specifies a list of key locator objects that retrieve the keys for digital signature and encryption from a keystore file or a repository. The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

Login mappings: **Version 5.x application**

Specifies a list of configurations for validating tokens within incoming messages.

Login mappings map the authentication method to the Java Authentication and Authorization Service (JAAS) configuration.

To configure JAAS, complete the following steps:

1. Click **Security > Global security**.
2. Under the Java Authentication and Authorization Service field, select **Application logins** or **System logins**.

Configuring the client for request encryption: Encrypting the message parts:

Version 5.x application

To configure the client for request encryption, specify which message parts to encrypt when configuring the client.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to familiarize yourself with the **WS Extensions** tab and the **WS Binding** tab in the Client Deployment Descriptor Editor within an assembly tool:

- “Configuring the client security bindings using an assembly tool” on page 588
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 591

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete the following steps to specify which message parts to encrypt when configuring the client for request encryption:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects > *application_name* > appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS extensions** tab, which is located at the bottom of Client Deployment Descriptor Editor within the assembly tool.
6. Expand **Request sender configuration > Confidentiality**. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the

message flowing across the Internet. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the correct target. For more information on encrypting , see XML encryption.

7. Select the parts of the message that you want to encrypt by clicking **Add**. You can select one of the following parts:

   Bodycontent
   User data portion of the message

   Usernametoken
   Basic authentication information, if selected

What to do next

After you specify which message parts to encrypt, you must specify which method to use to encrypt the request message. See “Configuring the client for request encryption: choosing the encryption method” for more information.

Configuring the client for request encryption: choosing the encryption method:

Version 5.x application

To configure the client for request encryption, specify which encryption method to use when configuring the client.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to familiarize yourself with the **WS Extensions** tab and the **WS Binding** tab in the Client Deployment Descriptor editor within an assembly tool:

- “Configuring the client security bindings using an assembly tool” on page 588
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 591

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete the following steps to specify which encryption method to use when configuring the client for request encryption:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > *application_name* > **appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS binding** tab, which is located at the bottom of the Client Deployment Descriptor editor within the assembly tool.
6. Expand **Security request sender binding configuration > Encryption information**.

7. Select an encryption option and click **Edit** to view the encryption information or click **Add** to add another option. The following table describes the purpose of this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   Encryption name
   Refers to the name of the encryption information entry.

   Data encryption method algorithm
   Encrypts and decrypts data in fixed size, multiple octet blocks.

   Key encryption method algorithm
   Represents public key encryption algorithms that are specified for encrypting and decrypting keys.

   Encryption key name
   Represents a Subject (**Owner** field of the certificate) from a public key certificate found by the encryption key locator, which is used by the key encryption method algorithm to encrypt the private key. The private key is used to encrypt the data.

   The key chosen must be a public key of the target. Encryption must be done using the public key and decryption must be done by the target using the private key (the personal certificate of the target).

   Encryption key locator
   Represents a reference to a key locator implementation class that locates the correct key store where the alias and the certificate exist. For more information on configuring key locators, see "Configuring key locators using an assembly tool" on page 560 and "Configuring key locators using the administrative console" on page 561.
8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the **Data Encryption method algorithm** and **Key Encryption method algorithm** dropdown lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the WebSphere administrative console.

Results

For more information, see "Configuring key locators using an assembly tool" on page 560 and "Configuring key locators using the administrative console" on page 561.

What to do next

You must specify which parts of the request message to encrypt. See "Configuring the client for request encryption: Encrypting the message parts" on page 605 if you have not previously specified this information.

Request receiver: **Version 5.x application**

The request receiver defines the security requirement of the SOAP message. The security handler on the request receiver side of the SOAP message enforces the security specifications that are defined in the IBM extension deployment descriptor (`ibm-webservices-ext.xmi`) and bindings (`ibm-webservices-bnd.xmi`).

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

The security constraint for request sender must match the security requirement of the request receiver for the server to accept the request. If the incoming SOAP message does not meet all the security requirements defined, then the request is rejected with the appropriate fault code returned to the sender. For security tokens, the token is validated using Java Authentication and Authorization Service (JAAS) login configuration and authenticated identity is set as the identity for the downstream invocation.

For example, if there is a security requirement to have the SOAP body digitally signed by Joe Smith and if the SOAP body of the incoming SOAP message is not signed by Joe Smith, then the request is rejected.

You can define the following security requirements for the request receiver:

Required integrity (digital signature)
You can select multiple parts of a message to sign digitally. The following list contains the integrity options:

- Body
- Time stamp
- Security token

Required confidentiality (encryption)
You can select multiple parts of a message to encrypt. The following list contains the confidentiality options:

- Body content
- Token

You can have multiple security tokens. The following list contains the security token options:

- Basic authentication, which requires both a user name and a password
- Identity assertion, which requires a user name only
- X.509 binary security token
- Lightweight Third Party Authentication (LTPA) binary security token
- Custom token, which is pluggable and supports custom-defined tokens validated by the JAAS login configuration

Received time stamp
You can have a time stamp for checking the timeliness of the message.

- Time stamp

Request receiver binding collection: **Version 5.x application**

Use this page to specify the binding configuration to receive request messages for Web services security.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_file_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit**.

Signing information: **Version 5.x application**

Specifies the configuration for the signing parameters. Signing information is used to sign and validate parts of a message including the body, the timestamp, and the user name token.

You also can use these parameters for X.509 certificate validation when the authentication method is `IDAssertion` and the ID Type is `X509Certificate` in the server-level configuration. In such cases, you must fill in the Certificate Path fields only.

Encryption information: **Version 5.x application**

Specifies the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message that include the body and the user name token.

Trust anchors: **Version 5.x application**

Specifies a list of keystore objects that contain the trusted root certificates that are issued by a certificate authority (CA).

The certificate authority authenticates a user and issues a certificate. The CertPath API uses the certificate to validate the certificate chain of incoming, X.509-formatted security tokens or trusted, self-signed certificates.

Collection certificate store: **Version 5.x application**

Specifies a list of the untrusted, intermediate certificate files.

The collection certificate store contains a chain of untrusted, intermediate certificates. The CertPath API attempts to validate these certificates, which are based on the trust anchor.

Key locators: **Version 5.x application**

Specifies a list of key locator objects that retrieve the keys for digital signature and encryption from a keystore file or a repository. The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

Trusted ID evaluators: **Version 5.x application**

Specifies a list of trusted ID evaluators that determine whether to trust the identity-asserting authority or message sender.

The trusted ID evaluators are used to authenticate additional identities from one server to another server. For example, a client sends the identity of user A to server 1 for authentication. Server 1 calls downstream to server 2, asserts the identity of user A, and includes the user name and password of server 1. Server 2 attempts to establish trust with server 1 by authenticating its user name and password and checking the trust based on the TrustedIDEvaluator implementation. If the authentication process and the trust check are successful, server 2 trusts that server 1 authenticated user A and a credential is created for user A on server 2 to invoke the request.

Login mappings: **Version 5.x application**

Specifies a list of configurations for validating tokens within incoming messages.

Login mappings map the authentication method to the Java Authentication and Authorization Service (JAAS) configuration.

To configure JAAS, complete the following steps:

1. Click **Security > Global security**.
2. Under the Java Authentication and Authorization Service, click **Application logins** or **System logins**.

Configuring the server for request decryption: decrypting the message parts:

Version 5.x application

Use the **WS Extensions** tab and the **WS Binding** configurations tab to specify which parts of the request message must be decrypted by the server.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.x and later applications.

Complete this task to specify which parts of the request message must be decrypted by the server. You must know which parts of the request message the client encrypts because the server must decrypt the same message parts.

Prior to completing these steps, read either of the following topics to become familiar with the **WS Extensions** tab and the **WS Binding** configurations tab:

- “Configuring the server security bindings using an assembly tool” on page 593
- “Configuring the server security bindings using the administrative console” on page 595

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete the following steps to configure the request receiver extensions:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects > *application_name* > ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Extensions** tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Required confidentiality** section.
7. Select the parts of the message to decrypt. The message parts selected for the request decryption on the server must match the message parts selected for the message encryption on the client. Click **Add** and select either of the following message parts:

   bodycontent
   The user data section of the message.

usernametoken
This token is the basic authentication information.

What to do next

After you specify which parts of the request message to decrypt, you must specify the method to use decrypt the message. See “Configuring the server for request decryption: choosing the decryption method” for more information.

Configuring the server for request decryption: choosing the decryption method:

Version 5.x application

You can use an assembly tool and the administrative console to configure the Web services security extensions and Web services security bindings.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **WS Extensions** tab and the **WS Bindings** tab:

- “Configuring the server security bindings using an assembly tool” on page 593
- “Configuring the server security bindings using the administrative console” on page 595

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete this task to specify which decryption method is used by the server to decrypt the request message. You must know which decryption method the client uses because the server must use the same method.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, select **Open with > Web services editor**.
5. Click the **Binding Configurations** tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Encryption information** section.
7. Click **Edit** to view the encryption information. The following table describes the purpose for each of these selections. Some definitions are taken from the XML-Encryption specification , which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   Encryption name
   Represents the name of this encryption information entry; an alias for the entry.

   Data encryption method algorithm
   Encrypts and decrypts data in fixed size, multiple octet blocks. This algorithm must be the same as the algorithm selected in the client request sender configuration.

Key encryption method algorithm
Represents algorithms specified for encrypting and decrypting keys. This algorithm must be the same as the algorithm selected in the client request sender configuration.

Encryption key name
Represents a Subject from a personal certificate, which is typically a distinguished name (DN) that is found by the encryption key locator. The subject is used by the key encryption method algorithm to decrypt the secret key, and the secret key is used to decrypt the data.

The key chosen must be a private key in the key store configured by the key locator. The key requires the same Subject used by the client to encrypt the data. Encryption must be done using the public key and decryption by using the private key (personal certificate). To ensure that the client encrypts the data with the correct public or private key, you must extract the public key from the server key store and add it to the key store specified in the encryption configuration information for the client request sender.

For example, the personal certificate of a server is `CN=Bob, O=IBM, C=US`. Therefore the server contains the public and private key pair. The client sending the request should encrypt the data using the public key for `CN=Bob, O=IBM, C=US`. The server decrypts the data using the private key for `CN=Bob, O=IBM, C=US`.

Encryption key locator
Represents a reference to a key locator implementation class that finds the correct keystore where the alias and the certificate exist. For more information on configuring key locators, go to the following sections: “Configuring key locators using an assembly tool” on page 560 and “Configuring key locators using the administrative console” on page 561.

8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm dropdown lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Results

It is important to note that for decryption, the encryption key name chosen must refer to a personal certificate that can be located by the key locator of the server referenced in the encryption information. Enter the Subject of the personal certificate here, which is typically a Distinguished Name (DN). The Subject uses the default key locator to find the key. If a custom key locator is written, the encryption key name can be anything used by the key locator to find the correct encryption key. The encryption key locator references the implementation class that finds the correct key store where this alias and certificate exist. Refer to “Configuring key locators using an assembly tool” on page 560 and “Configuring key locators using the administrative console” on page 561 for more information.

What to do next

You must specify which parts of the request message to decrypt. See “Configuring the server for request decryption: decrypting the message parts” on page 610 if you have not previously specified this information.

Response sender: **Version 5.x application**

The response sender defines the security requirements of the SOAP response message. The security handler acts on the security constraints that are defined for the response in the IBM extension deployment descriptors.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The IBM extension deployment descriptors are located in the `ibm-webservices-ext.xmi` file and the bindings, located in the `ibm-webservices-bnd.xmi` file. The security handler signs, encrypts, or generates the time stamp for the SOAP response message before the response is send to the caller.

Integrity constraints (digital signature)
You can select which parts of the message are digitally signed.

- Body
- Time stamp

Confidentiality (encryption)
You can encrypt the body content of the message.

Time stamp
You can have a time stamp for checking the timeliness of the message.

The security constraints that apply to the SOAP response message must match the security requirements defined in the response receiver. Otherwise, the response is rejected by the response receiver (caller).

Response sender binding collection: **Version 5.x application**

Use this page to specify the binding configuration for sender response messages for Web services security.

Note: There is an important distinction between Version 5.0.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** .
2. Under Modules, click **Manage modules** > ***URI_file_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.
4. Under Response sender binding, click **Edit**.

Signing information: **Version 5.x application**

Specifies the configuration for the signing parameters.

You also can use these parameters for X.509 certificate validation when the authentication method is IDAssertion and the ID Type is X509Certificate in the server-level configuration. In such cases, you must fill-in the Certificate Path fields only.

Encryption information: **Version 5.x application**

Specifies the configuration for the encryption and decryption parameters.

Key locators: **Version 5.x application**

Specifies a list of key locator objects that retrieve the keys for a digital signature and encryption from a keystore file or a repository. The key locator maps a name or logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

Configuring the server for response encryption: encrypting the message parts:

Version 5.x application

You can specify which parts of the response message to encrypt when configuring the server for response encryption.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **WS Extensions** tab and the **WS Bindings** tab in the Web services editor within an assembly tool:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to specify which parts of the response message to encrypt when configuring the server for response encryption:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, select **Open with > Web services editor**.
5. Click the **Extensions** tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender service configuration details > Confidentiality**. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the message flowing across the Internet. With confidentiality specifications, the response is encrypted before it is sent and decrypted when it is received at the correct target.
7. Select the parts of the response that you want to encrypt by clicking **Add** and selecting **Bodytoken** or **Usernametoken**. The following information describes the message parts:

   Bodycontent
   User data portion of the message.

   Usernametoken
   Basic authentication information, if selected.

A user name token does not appear in the response so you do not need to select this option for the response. If you select this option, make sure that you also select it for the client response receiver. If you do not select this option, make sure that you do not select it for the client response receiver.

What to do next

After you specify which message parts to encrypt, you must specify which method to use message encryption. See the task for choosing the encryption method when configuring the server for response encryption.

Configuring the server for response encryption: choosing the encryption method:

Version 5.x application

You can specify which method the server uses to encrypt the response message.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **Extensions** tab and the **Binding configurations** tab in the Web services editor within an assembly tool:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete the following steps to specify which method the server uses to encrypt the response message:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Binding Configuration**s tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender binding configuration details > Encryption information**.
7. Click **Edit** to view the encryption information. The following table describes the purpose of this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   Encryption name
   Refers to the name of the encryption information entry.

   Data encryption method algorithm
   Encrypts and decrypts data in fixed size, multiple octet blocks. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

Key encryption method algorithm
Represents public key encryption algorithms that are specified for encrypting and decrypting keys. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

Encryption key name
Represents a Subject from a public key certificate typically distinguished name (DN) that is found by the encryption key locator and used by the key encryption method algorithm to encrypt the private key. The private key is used to encrypt the data.

The key name chosen in the server response sender encryption information must be the public key of the key configured in the client response receiver encryption information. Encryption by the response sender must be done using the public key and decryption must be done by the response receiver using the associated private key (the personal certificate of the response receiver).

Encryption key locator
The encryption key locator represents a reference to a key locator implementation class that finds the correct key store where the alias and the certificate exist. For more information, see the tasks for configuring key locators.

8. Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Results

The encryption key name chosen must refer to a public key of the response receiver. For the encryption key name, use the Subject of the public key certificate, typically a Distinguished Name (DN). The name chosen is used by the default key locator to find the key. If you write a custom key locator, the encryption key name might be anything that is used by the key locator to find the correct encryption key (a public key). The encryption key locator references the implementation class that finds the correct key store where the alias and certificate exist.

What to do next

You must specify which parts of the response message to encrypt. See the task for configuring the server for response encryption if you have not previously specified this information.

Response receiver: **Version 5.x application**

The response receiver defines the security requirements of the response received from a request to a Web service. The security constraints for response sender must match the security requirements of the response receiver. If the constraints do not match, the response is not accepted by the caller or the sender.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

The security handler enforces the security constraints based on the security requirements defined in the IBM extension deployment descriptor, located in the `ibm-webservicesclient-ext.xmi` file and in the bindings, located in the `ibm-webservicessclient-bnd.xmi` file.

For example, the security requirement might have the response SOAP body encrypted. If the SOAP body of the SOAP message is not encrypted, the response is rejected and the appropriate fault code is communicated back to the caller of the Web services.

You can specify the following security requirements for a response receiver:

Required integrity (digital signature)
You can select which parts of a message are digitally signed. The following list contains the integrity options:
- Body
- Time stamp

Required confidentiality (encryption)
You can encrypt the body content of the message.

Received time stamp
You can have a time stamp for checking the timeliness of the message.

Response receiver binding collection: **Version 5.x application**

Use this page to specify the binding configuration for receiver response messages for Web services security.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

To view this administrative console page, complete the following steps:
1. Click **Applications > Application Types > WebSphere enterprise applications***application_name*.
2. Under Modules, click **Manage modules > *URI_file_name* > Web Services: Client security bindings**.
3. Under Response receiver binding, click **Edit**.

Signing information: **Version 5.x application**

Specifies the configuration for the signing parameters. Signing information is used to sign and to validate parts of the message including the body and the timestamp.

You can also use these parameters for X.509 validation when the authentication method is IDAssertion and the ID type is X509Certificate, in the server-level configuration. In such cases, you must fill in the certificate path fields only.

Encryption information: **Version 5.x application**

Specifies the configuration for the encryption and decryption parameters.

Encryption information is used for encrypting and decrypting various parts of a message, including the body and the user name token.

Trust anchors: **Version 5.x application**

Specifies a list of keystore objects that contain the trusted root certificates that are self-signed or issued by a certificate authority.

The certificate authority authenticates a user and issues a certificate. After the certificate is issued, the keystore objects, which contain these certificates, use the certificate for certificate path or certificate chain validation of incoming X.509-formatted security tokens.

Collection certificate store: **Version 5.x application**

Specifies a list of the untrusted, intermediate certificate files.

The collection certificate store contains a chain of untrusted, intermediate certificates. The CertPath API attempts to validate these certificates, which are based on the trust anchor.

Key locators: **Version 5.x application**

Specifies a list of key locator objects that retrieve the keys for a digital signature and encryption from a keystore file or a repository.

The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

Configuring the client for response decryption: decrypting the message parts:

Version 5.x application

To configure the client for response decryption, specify which response message parts to decrypt when configuring the client. The server response encryption and client response decryption configurations must match.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **WS Extensions** tab and the **WS Binding** tab in the Client Deployment Descriptor Editor within an assembly tool:

- “Configuring the client security bindings using an assembly tool” on page 588
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 591

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

About this task

Complete the following steps to specify which response message parts to decrypt when configuring the client for response decryption. The server response encryption and client response decryption configurations must match.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.

2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Extensions** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Response receiver configuration > Required confidentiality** section.
7. Select the parts of the message that you must decrypt by clicking **Add** and selecting either **Bodycontent** or **Usernametoken**. The following information describes these message parts:

   Bodycontent
   The user data portion of the message.

   Usernametoken
   The basic authentication information, if selected.

   The information selected in this step is encrypted by the server in the response sender.

   Note: A Username Token is typically not sent in the response. Thus, you usually do not need to select username token.

What to do next

After you specify which message parts to decrypt, you must specify which method to use when decrypting the response message. See "Configuring the client for response decryption: choosing a decryption method" for more information.

Configuring the client for response decryption: choosing a decryption method:

Version 5.x application

To configure the client for response decryption, specify which decryption method to use when the client decrypts the response message. The server response encryption and client response decryption configurations must match.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, read either of the following topics to become familiar with the **WS Extensions** tab and the **WS Bindings** tab in the Client Deployment Descriptor Editor within an assembly tool:

- "Configuring the client security bindings using an assembly tool" on page 588
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 591

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

About this task

Complete the following steps to specify which decryption method to use when the client decrypts the response message. The server response encryption and client response decryption configurations must match.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Binding** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Security response receiver binding configuration > Encryption information** section. For more information on encrypting and decrypting SOAP messages, see "XML encryption" on page 596.
7. Click **Edit** to view the encryption information. The following table describes the purpose for this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   Encryption name
   Refers to the alias that is used for the encryption information entry.

   Data encryption method algorithm
   Encrypts and decrypts data in fixed size, multiple octet blocks.

   Key encryption method algorithm
   Represents public key encryption algorithms specified for encrypting and decrypting keys.

   Encryption key name
   Represents a Subject from a personal certificate, which is typically a distinguished name (DN) that is found by the encryption key locator. The Subject is used by the key encryption method algorithm to decrypt the secret key. The secret key is used to decrypt the data.

   Note: The key chosen must be a private key of the client. Encryption must be done using the public key and decryption must be done by the private key (personal certificate). For example, the personal certificate of the client is: `CN=Alice, O=IBM, C=US`. Therefore, the client contains the public and private key pair. The target server that sends the response encrypts the secret key by using the public key for `CN=Alice, O=IBM, C=US`. The client decrypts the secret key by using the private key for `CN=Alice, O=IBM, C=US`.

   Encryption key locator
   Represents a reference to a key locator implementation class that finds the correct key store where the alias and the certificate exist. For more information on configuring key locators, see "Configuring key locators using an assembly tool" on page 560 and "Configuring key locators using the administrative console" on page 561.
8. Optional: Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Results

For decryption, the encryption key name chosen must refer to a personal certificate that can be located by the client key locator. The Subject (**owner** field of the certificate) of the personal certificate should be

entered in the Encryption key name, this is typically a Distinguished Name (DN). The default key locator uses the Encryption key name to find the key within the keystore. If you write a custom key locator, the encryption key name can be anything used by the key locator to find the correct encryption key. The encryption key locator references the implementation class that locates the correct key store where this alias and certificate exists. For more information, see “Configuring key locators using an assembly tool” on page 560 and “Configuring key locators using the administrative console” on page 561.

What to do next

You must specify which parts of the request message to decrypt. See the topic“Configuring the client for response decryption: decrypting the message parts” on page 618 if you have not previously specified this information.

Securing Web services for Version 5.x applications using basic authentication

Version 5.x application

With the basic authentication (BasicAuth) authentication method, the request sender generates a BasicAuth security token using a callback handler. The request receiver retrieves the BasicAuth security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. Trust is established by using user name and password validation.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides several different methods to secure your Web services. BasicAuth authentication is one of these methods. You might also secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- BasicAuth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

To use BasicAuth authentication to secure Web services, complete the following tasks:

1. Secure the client for BasicAuth authentication.
   a. Configure the client for basic authentication: specifying the method
   b. Configure the client for basic authentication: collecting the authentication information
2. Secure the server for BasicAuth authentication.
   a. Configure the server to handle basic authentication
   b. Configure the server to validate basic authentication information

Results

After completing these steps, you have secured your Web services using BasicAuth authentication.

Configuring the client for basic authentication: specifying the method:

Version 5.x application

Basic authentication (BasicAuth) refers to the user ID and password of a valid user in the registry of the target server. BasicAuth information can be collected in many ways, including through an administrative console prompt, a standard in (Stdin) prompt, or specified in the bindings that prevents user interaction.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

For more information on BasicAuth authentication, see: "BasicAuth authentication method."

About this task

Note: WebSphere Application Server supports nonce (randomly generated token) with BasicAuth authentication. For more information, see Nonce.

Complete the following steps to specify BasicAuth as the authentication method:

1. Launch an assembly tool. See more information on the assembly tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Extensions** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section. The only valid login configuration choices for a pure client are **BasicAuth** and **Signature**.
7. Select **BasicAuth** to authenticate the client using a user ID and a password. This user ID and password must be specified in the target user registry. The other choice, Signature, attempts to authenticate the client using the certificate used to digitally sign the message.

What to do next

For more information on getting started with the Web services client editor within the assembly tool, see either of the following topics:

- "Configuring the client security bindings using an assembly tool" on page 588
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 591

After you specify the BasicAuth authentication method, you must specify how to collect the authentication information. See "Configuring the client for basic authentication: collecting the authentication information" on page 623.

BasicAuth authentication method: **Version 5.x application**

When you use the BasicAuth authentication method, the security token that is generated is a <wsse:UsernameToken> element with <wsse:Username> and <wsse:Password> elements.

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

WebSphere Application Server supports text passwords but not password digest because passwords are not stored and cannot be retrieved from the server. On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, a Java Authentication and Authorization Service (JAAS) login module is used to validate the security token. These two operations, token generation and token validation, are described in the following sections.

BasicAuth token generation

The request sender generates a BasicAuth security token using a callback handler. The security token returned by the callback handler is inserted in the SOAP message. The callback handler that is used is specified in the <LoginBinding> element of the bindings file, `ibm-webservicesclient-bnd.xmi` . The following callback handler implementations are provided with WebSphere Application Server and can be used with the BasicAuth authentication method:

- com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

You can add your own callback handlers that implement the `javax.security.auth.callback.CallbackHandler` method.

BasicAuth token validation

The request receiver retrieves the BasicAuth security token from the SOAP message and validates it using a JAAS login module. The <wsse:Username> and <wsse:Password> elements in the security token are used to perform the validation. If the validation is successful, the login module returns a JAAS Subject. This Subject is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the <LoginMapping> element of the bindings file. Default bindings are specified in the `ws-security.xml` file. However, you can override these bindings using the application-specific `ibm-webservices-bnd.xmi` file. The configuration information consists of a CallbackHandlerFactory and a ConfigName value. The CallbackHandlerFactory option specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl CallbackHandlerFactory implementation. The ConfigName value specifies a JAAS configuration name entry. WebSphere Application Server searches the `security.xml` file for a matching configuration name entry. If a match is not found, it searches the `wsjaas.conf` file for a match. WebSphere Application Server provides the WSLogin default configuration entry, which is suitable for the BasicAuth authentication method.

Configuring the client for basic authentication: collecting the authentication information:

Version 5.x application

The basic authentication (BasicAuth) method refers to the user ID and the password of a valid user in the registry of the target server. Collection of BasicAuth information can occur in many ways including through a user interface prompt, a standard in (Stdin) prompt, or specified in the bindings, which prevents user interaction.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

For more information on BasicAuth authentication, see “BasicAuth authentication method” on page 622.

Complete this task to specify the authentication information needed for BasicAuth authentication:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Binding** tab, which is located at the bottom of deployment descriptor editor within the assembly tool.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** or **Enable** to view the login binding information. The login binding information displays and enter the following information:

   Authentication method
   Specifies the type of authentication. Select **BasicAuth** to use basic authentication.

   Token value type URI and Token value type local name
   When you select **BasicAuth**, you cannot edit the token value type URI and the local name values. Specifies values for custom authentication types. For BasicAuth authentication, leave these values blank.

   Callback handler
   Specifies the Java Authentication and Authorization Server (JAAS) callback handler implementation for collecting the BasicAuth information. You can use the following default implementations for the callback handler:

   com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
   This implementation is used for non-user interface console prompts.

   com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
   This implementation is used for user interface panel prompts.

   com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
   This implementation is used when you plan to always enter the user ID and password in the BasicAuth user ID and password section that follows.

   Basic Authentication user ID and Basic Authentication password
   Specifies values for the BasicAuth user ID and password, regardless of the default callback handler indicated previously, these user ID and password values are used to authenticate to the server for the Web services security authentication. If you leave these values blank, use either the GUIPromptCallbackHandler or the StdinPromptCallbackHandler implementation, but only on a pure client. Always fill-in these values for any Web service that acts as a client to another Web service that you want to specify for BasicAuth for authentication downstream. If you want the client identity of the originator to flow downstream, configure the Web service client to use either ID assertion or Lightweight Third Party Authentication (LTPA).

   Property
   Specifies properties with name and value pairs for custom callback handlers to use. For BasicAuth authentication, you do not need to enter any information. To enter a new property, click **Add** and enter the new property and value.

Results

Other basic authentication entries: There is a basic authentication entry in the Port Qualified Name Binding Details section. This entry is used for HTTP transport authentication, which might be required if the router servlet is protected.

Information specified in the Web services security basic authentication section overrides the basic authentication information specified in the Port Qualified Name Binding Details section for authorizing the Web service.

For a server that acts as a client, do not specify a user interface or non-user interface prompt callback handler. To configure BasicAuth authentication from one Web service to a downstream Web service, select the com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHander implementation and explicitly specify the BasicAuth user ID and password. If you want the client identity of the originator to flow downstream, configure the Web service client to use ID assertion.

What to do next

To use the BasicAuth authentication method, you must specify the method in the Login configuration section of the assembly tool . See “Configuring the client for basic authentication: specifying the method” on page 622 if you have not previously specified this information.

Identity assertion authentication method: **Version 5.x application**

When using the identity assertion (IDAssertion) authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, the security token is validated. These two operations, token generation and token validation operations, are described in the following sections.

Identity assertion token validation:

The request receiver retrieves the IDAssertion security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. With identity assertion, special processing is required to establish trust before asserting the identity as the established identity of the running thread. This special processing is defined by the <IDAssertion> element in the deployment descriptor file, `ibm-webservices-ext.xmi`. If all the validation checks are successful, the asserted identity is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the<LoginMapping> element of the bindings file. Default bindings are specified in the `ws-security.xml` file. However, you can override these bindings using the application specific `ibm-webservices-bnd.xmi` file. The configuration information consists of CallbackHandlerFactory and a ConfigName. CallbackHandlerFactory specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the `com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl` CallbackHandlerFactory implementation. ConfigName specifies a JAAS configuration name entry.

WebSphere Application Server searches the `security.xml` file for a matching configuration name entry. If a match is not found it searches the `wsjaas.conf` file. WebSphere Application Server provides the system.wssecurity.IDAssertion default configuration entry, which is suitable for the identity assertion authentication method.

The <IDAssertion> element in the `ibm-webservices-ext.xmi` deployment descriptor file specifies the special processing required when using the identity assertion authentication method. The <IDAssertion> element is composed of two sub-elements: <IDType> and <TrustMode>.

The <IDType> element specifies the method for asserting the identity. The supported values for asserting the identity are:

- Username
- Distinguished name (DN)
- X.509 certificate

When <IDType> is *username*, a username token (for example, Bob) is provided. This user name is mapped to a user in the user registry and is the asserted identity after successful trust validation. When the <IDType> value is *DN*, a user name token containing a distinguished name is provided (for example, `cn=Bob Smith, o=ibm, c=us`). This DN is mapped to a user in the user registry and this user is the asserted identity after successful trust validation. When the <IDType> is *X509Certificate*, a binary security token containing an X509 certificate is provided and the SubjectDN value from the certificate (for example, `cn=Bob Smith, o=ibm, c=us`) is extracted. This SubjectDN value is mapped to a user in the user registry and this user is the asserted identity after successful trust validation.

The `<TrustMode>` element specifies how the trust authority, or asserting authority, provides trust information. The supported values are:

- Signature
- BasicAuth
- No value specified

When the <TrustMode> value is `Signature` the signature is validated. Then, the signer (for example, `cn=IBM Authority, o=ibm, c=us`) is mapped to an identity in the user registry (for example, IBMAuthority). To ensure that the asserting authority is trusted, the mapped identity (for example, IBMAuthority) is validated against a list of trusted identities. When the <TrustMode> element is `BasicAuth`, there is a user name token with a user name and password, which is the user name and password of the asserting authority.

The user name and password are validated. If they are successfully validated, that user name (for example, IBMAuthority) is validated against a list of trusted identities. If a value is not specified for `<TrustMode>`, trust is presumed and additional trust validation is not performed. This type of identity assertion is called *presumed trust mode*. Use the presumed trust mode only in an environment where the trust is established using some other mechanism.

If all the validations described previously succeed, the asserted identity (for example, Bob) is set as the identity of the running thread. If any of the validations fail, the request is rejected with a SOAP fault exception.

Configuring the server to handle basic authentication information:

Version 5.x application

Basic authentication (BasicAuth) refers to the user ID and the password of a valid user in the registry of the target server. After a request is received that contains basic authentication information, the server needs to log in to form a credential. The credential is used for authorization.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

About this task

Complete the following steps to configure the server to handle BasicAuth authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects > *application_name* > ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Extensions** tab, which is located at the bottom of the Web services editor within an assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select the following options:
   - **BasicAuth**
   - **Signature**
   - **ID assertion**
   - **Lightweight Third Party Authentication (LTPA)**
7. Select **BasicAuth** to authenticate the client with a user ID and a password. The client must specify a valid user ID and password in the server user registry. If the user ID and the password supplied are not valid, an exception is provided, and the request ends without invoking the resource.

   You can select multiple login configurations, which means that different types of security information might be received at the server. The order in which the login configurations are added decides the order in which they are processed when a request is received. Problems can occur if you have multiple login configurations added that have security tokens in common. For example, ID assertion contains a BasicAuth token. For ID assertion to work properly, list ID assertion ahead of BasicAuth in the processing list or the BasicAuth processing overrides the IDAssertion processing.

What to do next

After you specify how the server handles BasicAuth authentication information, you must specify how the server validates the authentication information. See the task for configuring the server to validate BasicAuth authentication if you have not previously specified this information.

Configuring the server to validate basic authentication information:

Version 5.x application

Basic authentication (BasicAuth) refers to the user ID and the password of a valid user in the registry of the target server. You can specify how the server validates the BasicAuth information.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

After a request is received that contains basic authentication information, the server needs to log in to form a credential. The credential is used for authorization. If the user ID and the password supplied is invalid, an exception is thrown and the request ends without invoking the resource.

About this task

Complete the following steps to specify how the server validates the BasicAuth authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects >** ***application_name*** **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Binding Configurations** tab, which is located at the bottom of the Web services editor within an assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information or click **Add** to add new login mapping information. The login mapping dialog is displayed. Select or enter the following information:

   Authentication method
   Specifies the type of authentication that occurs. Select **BasicAuth** to use basic authentication.

   Configuration name
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the BasicAuth authentication method, enter `WSLogin` for the JAAS login Configuration name.

   Use token valid type
   Determines if you want to specify a custom token type. For the default authentication method selections, you do not need to specify this option.

   Token value type URI and Token value type URI local name
   When you select BasicAuth, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For BasicAuth authentication leave these fields blank.

   Callback handler factory class name
   Creates a JAAS CallbackHandler implementation that understands the following callbacks:
   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback
   - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
   - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
   - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

   Callback handler factory property name and Callback handler factory property value
   Specifies callback handler properties for custom callback handler factory implementations. You do not need to specify any properties for the default callback handler factory implementation. For BasicAuth, you do not need to enter any property values.

   Login mapping property name and Login mapping property value
   Specifies properties for a custom login mapping. For the default implementations including BasicAuth, leave these fields blank.

What to do next

You must specify how the server handles the BasicAuth authentication method. See the task for configuring the server to handle basic authentication if you have not previously specified this information.

Identity assertion in a SOAP message

Version 5.x application

Identity assertion is a method for expressing the identity of the sender (for example, user name) in a SOAP message. When identity assertion is used as an authentication method, the authentication decision is performed based only on the name of the identity and not on other information, such as passwords and certificates.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Identity assertion involves:

ID type

The Web Services Security implementation in WebSphere Application Server can handle these identity types:

User name
Denotes the user name, such as the one in the local operating system (for example, `alice`). This name is embedded in the <Username> element within the <UsernameToken> element.

DN Denotes the distinguished name (DN) for the user, such as `"CN=alice, O=IBM, C=US"`. This name is embedded in the <Username> element within the <UsernameToken> element.

X.509 certificate
Represents the identity of the user as an X.509 certificate instead of a string name. This certificate is embedded in the <BinarySecurityToken> element.

Managing trust

The intermediary host in the SOAP message itinerary can assert claimed identity of the initial sender. Two methods (called *trust mode*) are supported for this assertion:

Basic authentication
The intermediary adds its user name and password pair to the message.

Signature
The intermediary digitally signs the <UsernameToken> element of the initial sender.

Note: This trust mode does not support the X.509 certificate ID type.

Typical scenario

ID assertion is typically used in the multi-hop environment where the SOAP message passes through one or more intermediary hosts. The intermediary host authenticates the initial sender. The following scenario describes the process:

1. The initial sender sends a SOAP message to the intermediary host with some embedded authentication information. This authentication information might be a user name and a password pair with an Lightweight Third Party Authentication (LTPA) token.
2. The intermediary host authenticates the initial sender according to the embedded authentication information.
3. The intermediary host removes the authentication information from the SOAP message and replaces it with the <UsernameToken> element, which contains a user name.
4. The intermediary host asserts the trust according to the trust mode.
5. The intermediary host sends the updated SOAP message to the ultimate receiver.

6. The ultimate receiver checks the trust against the intermediary host information according to the configured trust mode. Also, the trusted ID evaluator is invoked.
7. If trust is established by the final receiver, the receiver invokes the Web service under the authorization of the user name (that is, the initial sender) in the SOAP message.

Securing Web services for Version 5.x applications using identity assertion authentication

Version 5.x application

With the identity assertion authentication method, the security token generates a <wsse:UsernameToken> element that contains a <wsse:Username> element. On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, the security token is validated. Unlike BasicAuth authentication, trust is established through the use of a security token rather than through user name and password validation.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides several different methods to secure your Web services. Identity assertion authentication is one of these methods. You might also secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- BasicAuth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

About this task

To use identity assertion authentication to secure Web services, complete the following tasks:

1. Secure the client for identity assertion authentication.
   a. “Configuring the client for identity assertion: specifying the method”
   b. “Configuring the client for identity assertion: collecting the authentication method” on page 632
2. Secure the server for identity assertion authentication.
   a. “Configuring the server to handle identity assertion authentication” on page 633
   b. “Configuring the server to validate identity assertion authentication information” on page 634

Results

After completing these steps, you have secured your Web services by using identity assertion authentication.

Configuring the client for identity assertion: specifying the method:

Version 5.x application

You can configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task is used to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client. Identity assertion works only when you configure on the client-side of a Web service acting as a client to a downstream Web service.

In order for the downstream Web service to accept the identity of the originating client (just the user name), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see “Trusted ID evaluator” on page 562.

Complete the following steps to specify identity assertion as the authentication method:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Extension** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section.
7. Select **IDAssertion** as the authentication method. For more conceptual information on identity assertion authentication, see “Identity assertion in a SOAP message” on page 629.
8. Expand the IDAssertion section.
9. For the ID type, select **Username**. This value works with all registry types and originating authentication methods.
10. For the trust mode, select either **BasicAuth** or **Signature**.
    - By selecting **BasicAuth**, you must include basic authentication information (user ID and password), which the downstream Web service has specified in the trusted ID evaluator as a trusted user ID. See “Configuring the client for signature authentication: collecting the authentication information” on page 639 to specify the user ID and password information.
    - By selecting **Signature** the certificate configured in the signature information section used to sign the data also is that is used as the trusted subject. The Signature is used to create a credential and user ID, which the certificate mapped to the downstream registry, is used in the trusted ID evaluator as a trusted user ID.

What to do next

See “Configuring the client security bindings using an assembly tool” on page 588 for more information on the Web services client editor within the assembly tool.

After you specify identity assertion as the authentication method used by the client, you must specify how to collect the authentication information. See “Configuring the client for identity assertion: collecting the authentication method” on page 632 for more information.

Configuring the client for identity assertion: collecting the authentication method:

Version 5.x application

You can configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task is used to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client. Identity assertion works only when you configure on the client-side of a Web service acting as a client to a downstream Web service.

In order for the downstream Web service to accept the identity of the originating client (just the user name), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. See the information on trusted ID evaluators.

Complete the following steps to specify how the client collects the authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > *application_name* > **appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Binding** tab, which is located at the bottom of the Deployment Descriptor Editor within an assembly tool.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** to view the login binding information and select **IDAssertion**. The login binding dialog is displayed. Select or enter the following information:

   Authentication method
   The authentication method specifies the type of authentication that occurs. Select **IDAssertion** to use identity assertion.

   Token value type URI and Token value type Local name
   When you select IDAssertion, you cannot edit the token value type Universal Resource Identifier (URI) and the local name. Specifies custom authentication types. For IDAssertion authentication, leave these values blank.

   Callback handler
   Specifies the Java Authentication and Authorization Service (JAAS) callback handler implementation for collecting the BasicAuth information. Specify the com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation for IDAssertion.

   Basic authentication User ID and Basic authentication Password
   In this field, the trust mode entered in the extensions is BasicAuth. Specifies the trusted user ID and password in these fields. The user ID specified must be an ID that is trusted by the downstream Web service. The Web service trusts the user ID if it is entered as a trusted ID in

a trusted ID evaluator in the downstream Web service bindings. If the trust mode entered in the extensions is `Signature`, you do not need to specify any information in this field.

Property name and Property value
Specifies properties with name and value pairs, for use by custom callback handlers. For IDAssertion, you do not need to specify any information in this field.

What to do next

To use the identity assertion authentication method, you must specify the method in the Security extensions section of an assembly tool. See "Configuring the client for identity assertion: specifying the method" on page 630 if you have not previously specified this information.

Configuring the server to handle identity assertion authentication:

Version 5.x application

The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. You can configure identity assertion authentication for the server. Do not attempt to configure identity assertion from a pure client.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

For the downstream Web service to accept the identity of the originating client (user name only), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see "Trusted ID evaluator" on page 562. The server side passes the special BasicAuth credential into the trusted ID evaluator, which returns `true` or `false` that this ID is trusted. After it is trusted, the user name of the client is mapped to the credential, which is used for authorization.

Complete the following steps to configure the server to handle identity assertion authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Extensions** tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. The options you can select are:
   - **BasicAuth**
   - **Signature**
   - **ID assertion**
   - **LTPA** (Lightweight Third Party Authentication)
7. Select **IDAssertion** to authenticate the client using the identity assertion data provided.

   The user ID of the client must be in the target user registry or repository, which is configured on the **Security > Global security** panel in the administrative console for WebSphere Application Server. You

can select multiple login configurations, which means that different types of security information can be received at the server. The order in which the login configurations are added determines the processing order when a request is received. Problems can occur if you have multiple login configurations added that have common security tokens. For example, ID assertion contains a BasicAuth token, which is the trusted token. For ID assertion to work properly, you must list ID assertion ahead of BasicAuth in the list or BasicAuth processing overrides ID assertion processing.

8. Expand the **IDAssertion** section and select both the **ID Type** and the **Trust Mode**.
   a. For ID Type, the options are:
      - **Username**
      - **DN** (distinguished name)
      - **X509certificate**

      These choices are just preferences and are not guaranteed. Most of the time the Username option is used. You must choose the same ID Type as the client.
   b. For Trust Mode, the options are:
      - **BasicAuth**
      - **Signature**

      The Trust Mode refers to the information sent by the client as the trusted ID.
      1) If you select **BasicAuth**, the client sends basic authentication data (user ID and password). This BasicAuth data is authenticated to the configured user registry. When the authentication occurs successfully, the user ID must be part of the trusted ID evaluator trust list.
      2) If you select **Signature**, the client signing certificate is sent. This certificate must be mappable to the configured user registry. For **Local OS**, the common name (CN) of the distinguished name (DN) is mapped to a user ID in the registry. For **Lightweight Directory Access Protocol (LDAP)**, the DN is mapped to the registry for the ExactDN mode. If it is in the CertificateFilter mode, attributes are mapped accordingly. In addition, the user name from the credential generated must be in the Trusted ID Evaluator trust list.

What to do next

For more information on getting started with the Web Services Editor within an assembly tool, see “Configuring the server security bindings using an assembly tool” on page 593.

After you specify how the server handles identity assertion authentication information, you must specify how the server validates the authentication information. See the task for configuring the server to validate identity assertion authentication information.

Configuring the server to validate identity assertion authentication information:

Version 5.x application

The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

Use this task to configure identity assertion authentication. Do not attempt to configure identity assertion from a pure client.

For the downstream Web service to accept the identity of the originating client (user name only), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see “Trusted ID evaluator” on page 562. The server side passes the special BasicAuth credential into the trusted ID evaluator, which returns a `true` or `false` response that this ID is trusted. After it is trusted, the user name of the client is mapped to the credential, which is used for authorization.

Complete the following steps to validate the identity assertion authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects >** ***application_name*** **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information. Click **Add** to add new login mapping information. The login mapping dialog is displayed. Select or enter the following information:

   Authentication method
   Specifies the type of authentication that occurs. Select **IDAssertion** to use basic authentication.

   Configuration name
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the IDAssertion authentication method, enter `system.wssecurity.IDAssertion` for the Java Authentication and Authorization Service (JAAS) login configuration name.

   Use token value type
   Determines if you want to specify a custom token type. For the default authentication method selections, you do not need to specify this option.

   Token value type URI and Token value type local name
   When you select ID assertion, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For the ID assertion authentication method, leave these values blank.

   Callback handler factory class name
   Ccreates a JAAS CallbackHandler implementation that understands the following callbacks:

   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback
   - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
   - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
   - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

   For any of the default authentication methods (BasicAuth, IDAssertion, and Signature), use the callback handler factory default implementation. Enter the following class name for any of the default Authentication methods including IDAssertion:

   ```
   com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl
   ```

   This implementation creates the correct callback handler for the default implementations.

   Callback handler factory property name and Callback handler factory property value
   Specifies callback handler properties for custom callback handler factory implementations. The default callback handler factory implementation does not need any specified properties. For ID assertion, leave these values blank.

Login mapping property name and Login mapping property value
Specifies properties for a custom login mapping. For the default implementations including IDAssertion, leave these values blank.

8. Expand the **Trusted ID evaluator** section.
9. Click **Edit** to see a dialog that displays all the trusted ID evaluator information. The following table describes the purpose of this information.

Class name
Refers to the implementation of the trusted ID evaluator that you want to use. Enter the default implementation as

```
com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl
```

If you want to implement your own trusted ID evaluator, you must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

Property name
Represents the name of this configuration. Enter `BasicIDEvaluator`.

Property value
Defines the name and value pairs that can be used by the trusted ID evaluator implementation. For the default implementation, the trusted list is defined here. When a request comes in and the trusted ID is verified, the user ID, as it appears in the user registry, must be listed in this property. Specify the property as a name and value pair where the name is *trustedId_n*. *n* is an integer starting from 0 and the value is the user ID associated with that name. An example list with the trusted names include two properties.

For example: `trustedId_0 = user1, trustedId_1 = user2`. The previous example means that both user1 and user2 are trusted. `user1` and `user2` must be listed in the configured user registry

10. Expand the **Trusted ID evaluator reference** section.
11. Click **Enable** to add a new entry. The text you enter for the **Trusted ID evaluator reference** must be the same as the name entered previously in the **Trusted ID evaluator**. Make sure that the name matches exactly because the information is case sensitive. If an entry is already specified, you can change it by clicking **Edit**.

What to do next

You must specify how the server handles the identity assertion authentication method. See “Configuring the server to handle identity assertion authentication” on page 633 if you have not previously specified this information.

Securing Web services for version 5.x applications using signature authentication

Version 5.x application

WebSphere Application Server provides several different methods to secure your Web services. XML digital signature is one of these methods.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

You can secure your Web services by using any of the following methods:

- XML digital signature
- XML encryption
- BasicAuth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

About this task

With the signature authentication method, the request sender generates a signature security token using a callback handler. The security token returned by the callback handler is inserted in the SOAP message. The request receiver retrieves the Signature security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. To use signature authentication to secure Web services, complete the following tasks:

1. Secure the client for signature authentication.
   a. "Configuring the client for signature authentication: specifying the method."
   b. "Configuring the client for signature authentication: collecting the authentication information" on page 639.
2. Secure the server for signature authentication.
   a. "Configuring the server to support signature authentication" on page 640.
   b. "Configuring the server to validate signature authentication information" on page 641.

Results

After completing these steps, you have secured your Web services using signature authentication.

Configuring the client for signature authentication: specifying the method:

Version 5.x application

Signature authentication, the use of an X.509 certificate to login on the target server, can be configured.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task is used to configure signature authentication. A signature refers to the use of an X.509 certificate to login on the target server. For more information on signature authentication, see "Signature authentication method" on page 638.

Complete the following steps to specify signature as the authentication method:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Extension** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.

6. Expand the **Request sender configuration > Login configuration** section. The following login configuration options are valid for a managed client and Web services acting as a client are:

   BasicAuth
   Use this option for a managed client.

   Signature
   Use this option for a managed client.

   IDAssertion
   Use this option for Web services acting as a client.
7. Select **Signature** to authenticate the client using the certificate used to digitally sign the request.

Results

For more information on getting started with the Web services client editor within the assembly tool, see "Configuring the client security bindings using an assembly tool" on page 588.

After you specify signature as the authentication method, you must specify how to collect the authentication information. See "Configuring the client for signature authentication: collecting the authentication information" on page 639 for more information.

Signature authentication method: **Version 5.x application**

Signature authentication refers to an X.509 certificate that is sent by the client to the server. The certificate is used to authenticate to the user registry that is configured at the server. When using the signature authentication method, the security token is generated with a ds:Signature and a wsse:BinarySecurityToken element.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, a Java Authentication and Authorization Service (JAAS) login module is used to validate the security token. These two operations, token generation and token validation, are described in the following sections.

Signature token generation
The request sender generates a Signature security token using a callback handler. The security token returned by the callback handler is inserted in the SOAP message. The callback handler is specified in the <LoginBinding> element of the bindings file, `ibm-webservicesclient-bnd.xmi`. WebSphere Application Server provides the following callback handler implementation that can be used with the Signature authentication method:
com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

You can add your own callback handlers that implement the javax.security.auth.callback.CallbackHandler implementation.

Security token validation
The request receiver retrieves the Signature security token from the SOAP message and validates it using a JAAS login module. The <ds:Signature> and <wsse:BinarySecurityToken> elements in the security token are used to perform the validation. If the validation is successful, the login module returns a Java Authentication and Authorization Service (JAAS) Subject. This Subject then is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the <LoginMapping> element of the bindings file. Default bindings are specified in the ws-security.xml file. However, you can override these bindings using the application-specific ibm-webservices-bnd.xmi file. The configuration information consists of a CallbackHandlerFactory and a ConfigName. The CallbackHandlerFactory specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImp CallbackHandlerFactory implementation. The ConfigName specifies a JAAS configuration name entry. WebSphere Application Server searches in the security.xml file for a matching configuration name entry. If a match is not found, it searches the wsjaas.conf file. WebSphere Application Server provides the system.wssecurity.Signature default configuration entry, which is suitable for the signature authentication method.

Configuring the client for signature authentication: collecting the authentication information:

Version 5.x application

Signature authentication refers to an X.509 certificate that is sent by the client to the server. The certificate is used to authenticate to the user registry that is configured at the server. The client collects the authentication information for signature authentication.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

You can configure signature authentication. A signature refers to the use of an X.509 certificate to login on the target server.

Complete the following steps to specify how the client collects the authentication information for signature authentication:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > *application_name* > **appClientModule > META-INF**.
4. Right-click the application-client.xml file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Security request sender binding configuration > Signing information** and click **Edit** to modify the signing key name and signing key locator. To create new signing information, click **Enable**. The certificate that is sent to log in at the server is the one configured in the Signing Information section. Review the key locator information to understand how the signing key name maps to a key within the key locator entry.

   The following list describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web address: http://www.w3.org/TR/xmldsig-core

   Canonicalization method algorithm
   Canonicalizes the <SignedInfo> element before it is digested as part of the signature operation.

   Digest mehod algorithm
   Represents the algorithm that is applied to the data after transforms are applied, if specified, to

yield the <DigestValue> element. The signing of the <DigestValue> element binds the resource content to the signer key. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.

Signature method algorithm
Represents the algorithm that is used to convert the canonicalized <SignedInfo> element value into the <SignatureValue> value. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.

Signing key name
Represents the key entry that is associated with the signing key locator. The key entry refers to an alias of the key, which is used to sign the request.

Signing key locator
Represents a reference to a key locator implementation.

7. Expand the **Security request sender binding configuration > Login binding** section.
8. Click **Edit** to view the login binding information. Select or enter the following information:

Authentication method
Specifies the type of authentication that occurs. Select **Signature** to use signature authentication.

Token value type URI and Token value type URI local name
When you select **Signature**, you cannot edit token value type Uniform Resource Identifier (URI) and local name values. Specifies custom authentication types. For signature authentication, leave these fields blank.

Callback handler
Specifies the Java Authentication and Authorization Server (JAAS) callback handler implementation for collecting signature information. Enter the following callback handler for signature authentication:
`com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler`

This callback handler is used because the signature method does not require user interaction.

Basic authentication user ID and Basic authentication password
Leave the BasicAuth fields blank when signature authentication is used.

Property name and property value
This field enables you to enter properties and name and value pairs for use by custom callback handlers. For signature authentication, do not enter any information.

What to do next

Other customization entries: There is a basic authentication entry in the Port Qualified Name Binding Details section. This entry is used for HTTP transport authentication, which might be required if the router servlet is protected.

Information specified in the Web services security signature authentication section overrides the basic authentication information specified in the Port Qualified Name Binding Details section for authorizing the Web service.

To use the signature authentication method, you must specify the authentication method in the Login configuration section of an assembly tool.

Configuring the server to support signature authentication: **Version 5.x application**

Signature authentication refers to an X.509 certificate sent by the client to the server. The certificate is used to authenticate to the user registry configured at the server. After a request is received by the server

that contains the certificate, the server needs to log in to form a credential. The credential is used for authorization. You can configure signature authentication at the server.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

If the certificate supplied cannot be mapped to an entry in the user registry, an exception is provided and the request ends without invoking the resource.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects > *application_name* > ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Extensions** tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select from the following options:
   - BasicAuth
   - Signature
   - ID assertion
   - Lightweight Third Party Authentication (LTPA)
7. Select **Signature** to authenticate the client using an X509 certificate. The certificate that is sent from the client is the certificate that issued for signing the message. You must be able to map this certificate to the configured user registry. For Local operating system (OS) registries, the common name (cn) of the distinguished name (DN) is mapped to a user ID in the registry. For Lightweight Directory Access Protocol (LDAP), you can configure multiple mapping modes:
   - EXACT_DN is the default mode that directly maps the DN of the certificate to an entry in the LDAP server.
   - CERTIFICATE_FILTER is the mode that provides the LDAP advanced configuration with a place to specify a filter that maps specific attributes of the certificate to specific attributes of the LDAP server.

What to do next

For more information on getting started with the Web services editor within the assembly tool, see “Configuring the server security bindings using an assembly tool” on page 593.

After you specify how the server handles signature authentication information, you must specify how the server validates the authentication information. See the task for configuring the server to validate signature authentication.

Configuring the server to validate signature authentication information:

Version 5.x application

Signature authentication refers to an X.509 certificate sent by the client to the server. The certificate is used to authenticate to the user registry configured at the server. After a request is received by the server that contains the certificate, the server needs to log in to form a credential. The credential is used for authorization. You can validate signature authentication at the server.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

If the certificate supplied cannot be mapped to an entry in the user registry, an exception is thrown and the request ends without invoking the resource.

Complete the following steps to configure the server to validate signature authentication:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule** > **META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the **Binding Configurations** tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information or click **Add** to add new login mapping information. The login mapping dialog is displayed and you select (or enter) the following information:

   Authentication method
   Specifies the type of authentication. Select **Signature** to use signature authentication.

   Configuration name
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the signature authentication method, enter `system.wssecurity.Signature` for the JAAS login configuration name. This specification logs in with the com.ibm.wsspi.wssecurity.auth.module.SignatureLoginModule JAAS login module.

   Use token value type
   Determines if you want to specify a custom token type. For the default authentication method selections, you can leave this field blank.

   URI and local name
   When you select Signature method, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For signature authentication, you can leave this field blank.

   Callback handler factory class name
   Creates a JAAS CallbackHandler implementation that understands the following callback handlers:

   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback
   - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
   - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
   - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

   For any of the default authentication methods (BasicAuth, IDAssertion, and Signature), use the callback handler factory default implementation. Enter the following class name for any of the default authentication methods including signature:

   ```
   com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl
   ```

   This implementation creates the correct callback handler for the default implementations.

Callback handler factory property name and callback handler factory property value
Specifies callback handler properties for custom callback handler factory implementations. You do not need to specify any properties for the default callback handler factory implementation. For signature, you can leave this field blank.

Login mapping property name and login mapping property value
Specifies properties for a custom login mapping to use. For the default implementations including signature, you can leave this field blank.

What to do next

Specify how the server handles the signature authentication method. See "Configuring the server to support signature authentication" on page 640 if you have not previously specified this information.

Security token

Version 5.x application

A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A specific type of security token is not required by Web services security. Web services security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

A security token is embedded in the SOAP message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the WebSphere Application Server security handler authenticates the security token and sets up the caller identity on the running thread.

Securing Web services for version 5.x applications using a pluggable token

Version 5.x application

To use pluggable tokens to secure your Web services, you must configure both the client request sender and the server request receiver. You can configure your pluggable tokens using the WebSphere Application Server administrative console.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

WebSphere Application Server provides several different methods to secure your Web services. A pluggable token is one of these methods. You might secure your Web services by using any of the following methods:

- XML digital signature

- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

Complete the following steps to secure your Web services using a pluggable token:

1. Generate a security token using the Java Authentication and Authorization Service (JAAS) CallbackHandler interface. The Web services security runtime uses the JAAS CallbackHandler interface as a plug-in to generate security tokens on the client side or when Web services is acting as a client.
2. Configure your pluggable token. For more information, see the following tasks:
   - "Configuring pluggable tokens using an assembly tool"
   - "Configuring pluggable tokens using the administrative console" on page 647

Configuring pluggable tokens using an assembly tool: Version 5.x application

The following information describes how to configure a pluggable token using an assembly tool.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This document describes how to configure a pluggable token in the request sender (`ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` file) and request receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` file).

The pluggable token is required for the request sender and request receiver because they are a pair. The request sender and the request receiver must match for the receiver to accept a request.

Prior to completing these steps, it is assumed that you have already created a Web service that is based on the Java Platform, Enterprise Edition (Java EE) specification. See either of the following topics for an introduction of how to manage Web services security binding information for the server:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

About this task

You must specify the security constraints in the `ibm-webservicesclient-ext.xmi` and the `ibm-webservices-ext.xmi` files for the required tokens using an IBM assembly tool.

Complete the following steps to configure the request sender using the `ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` files:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Switch to the Java EE perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Extension** tab. The Web service client security extensions editor is displayed.

a. Under Service References, select an existing service reference or click **Add** to create a new reference.
b. Under Port Qname Bindings, select an existing port qualified name for the selected service reference or click **Add** to create a new port name binding.
c. Under Request Sender Configuration: Login Configuration, select an exiting authentication method or type in a new one in the editable list box (Lightweight Third Party Authorization (LTPA) is a supported token generation when Web services is acting as client).
d. Click **File > Save** to save the changes.

6. Click the **Web services client binding** tab. The Web services client binding editor is displayed.
   a. Under Port qualified name binding, select an existing entry or click **Add** to add a new port name binding. The Web services client binding editor displays for the selected port.
   b. Under Login binding, click **Edit** or **Enable**. The Login Binding dialog box is displayed.
      1) In the Authentication Method field, enter the authentication method. The authentication method that you enter in this field must match the authentication method defined on the **Security Extension** tab for the same Web service port. This field is mandatory.
      2) (Optional) Enter the token value type information in the URI and Local name fields. These fields are ignored for the BasicAuth, Signature, and IDAssertion authentication methods, but required for other authentication methods. The token value type information is inserted into the <wsse:BinarySecurityToken>@ValueType element for binary security token and is used as the namespace for the XML-based token.
      3) Enter an implementation of the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface. This field is mandatory.
      4) Enter the basic authentication information in the **User ID** and **Password** fields. The basic authentication information is passed to the construct of the CallbackHandler implementation. The use of the basic authentication information depends on the implementation of CallbackHandler.
      5) In the Property field, add name and value pairs. These pairs are passed to the construct of the CallbackHandler implementation as `java.util.Map` values.
      6) Click **OK**.

      Click **Disable** under Login binding on the **Web services client port binding** tab to remove the authentication method login binding.
   c. Click **File > Save** to save the changes.
7. In the Package Explorer window, right-click the `webservices.xml` file and click **Open with > Web services editor**. The Web Services window displays.
   a. Click the **Security extensions** tab. The Web service security extensions editor is displayed.
      1) Under Web Service Description Extension, select an existing service reference or click **Add** to create a new extension.
      2) Under Port Component Binding, select an existing port qualified name for the selected service reference or click **Add** to create a new one.
      3) Under Request Receiver Service Configuration Details: Login Configuration, select an exiting authentication method or click **Add** and enter a new method in the Add AuthMethod field that displays. You can select multiple authentication methods for the request receiver. The security token of the incoming message is authenticated against the authentication methods in the order that they are specified in the list. Click **Remove** to remove the selected authentication method or methods.
   b. Click **File > Save** to save the changes.
   c. Click the **Bindings** tab. The Web services bindings editor is displayed.
      1) Under Web service description bindings, select an existing entry or click **Add** to add a new Web services descriptor.

2) Click the **Binding configurations** tab. The Web services binding configurations editor is displayed for the selected Web services descriptor.
3) Under Request receiver binding configuration details: login mapping, click **Add** to create a new login mapping or click **Edit** to edit the selected login mapping. The Login mapping dialog is displayed.
   a) In the Authentication method field, enter the authentication method. The information entered in this field must match the authentication method defined on the **Security Extensions** tab for the same Web service port. This field is mandatory.
   b) In the **Configuration name** field, enter a JAAS login configuration name. This field is mandatory. You must define the JAAS login configuration name in the WebSphere Application Server administrative console under **Security > Global security**. Under Authentication, click **Java Authentication and Authorization Service > Application logins**. For more information, see Configuring programmatic logins for Java Authentication and Authorization Service.
   c) (Optional) Select **Use Token value type** and enter the token value type information in the URI and Local name fields. This information is optional for BasicAuth, Signature and IDAssertion authentication methods, but required for any other authentication method. The token value type is used to validate the <wsse:BinarySecurityToken>@ValueType element for binary security tokens and to validate the namespace of the XML-based token.
   d) Under Callback Handler Factory, enter an implementation of the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface in the Class name field. This field is mandatory.
   e) Under Callback Handler Factory property, click **Add** and enter the name and value pairs for the Callback Handler Factory Property. These name and value pairs are passed as java.util.Map to the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory.init() method. The use of these name and value pairs is determined by the CallbackHandlerFactory implementation.
   f) Under Login Mapping Property, click **Add** and enter the name and value pairs for the Login mapping property. These name and value pairs are available to the JAAS Login Modules through thecom.ibm.wsspi.wssecurity.auth.callback.PropertyCallback JAAS Callback interface. Click **Remove** to delete the selected login mapping.
   g) Click **OK**.

d. Click **File > Save** to save the changes.

Results

The previous steps define how to configure the request sender to create security tokens in the SOAP message and to configure the request receiver to validate the security tokens found in the incoming SOAP message. WebSphere Application Server supports pluggable security tokens.

You can use the authentication method defined in the login bindings and login mappings to generate security tokens in the request sender and validate security tokens in the request receiver.

What to do next

After you configure pluggable tokens, you must configure both the client and the server to support pluggable tokens. See the following topics to configure the client and the server:

- Configuring the client for LTPA token authentication: Specifying LTPA token authentication
- Configuring the client for LTPA token authentication: Collecting the authentication information
- Configuring the server to handle LTPA token authentication
- Configuring the server to validate LTPA token authentication information

Configuring pluggable tokens using the administrative console: **Version 5.x application**

You can configure the client-side request sender (`ibm-webservicesclient-bnd.xmi` file) or server-side request receiver (`ibm-webservices-bnd.xmi` file) by using the WebSphere Application Server administrative console.

Before you begin

Note: There is an important distinction between Version 5.*x* and Version 6 and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Prior to completing these steps, it is assumed that you have already created a Web service that is based on the Java Platform, Enterprise Edition (Java EE) specification. See either of the following topics for an introduction of how to manage Web services security binding information for the server:

- "Configuring the server security bindings using an assembly tool" on page 593
- "Configuring the server security bindings using the administrative console" on page 595

About this task

This document describes how to configure a pluggable token in the request sender (`ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` file) and request receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` file).

Note: The pluggable token is required for the request sender and request receiver as they are a pair. The request sender and the request receiver must match for a request to be accepted by the receiver.

Prior to completing these steps, it is assumed that you deployed a Web services-enabled enterprise application to the WebSphere Application Server.

Use the following steps to configure the client-side request sender (`ibm-webservicesclient-bnd.xmi` file) or server-side request receiver (`ibm-webservices-bnd.xmi` file) using the WebSphere Application Server administrative console.

1. Click **Applications > Application Types > WebSphere enterprise applications > *enterprise_application***.
2. Under Modules, click **Manage modules > URI_name**. The *URI* is the Web services-enabled module.
   a. Under Web Services Security Properties, click **Web services: client security bindings** to edit the response sender binding information, if Web services is acting as client.
      1) Under Response sender binding, click **Edit**.
      2) Under Additional Properties, click **Login binding**.
      3) Select **Dedicated login binding** to define a new login binding.
         a) Enter the authentication method, this must match the authentication method defined in IBM extension deployment descriptor. The authentication method must be unique in the binding file.
         b) Enter an implementation of the JAAS javax.security.auth.callback.CallbackHandler interface.
         c) Enter the basic authentication information (User ID and Password) and the basic authentication information is passed to the construct of the CallbackHandler implementation. The usage of the basic authentication information is up to the implementation of the CallbackHandler.
         d) Enter the token value type, it is optional for BasicAuth, Signature and IDAssertion authentication methods but required for any other authentication method. The token value

type is inserted into the <wsse:BinarySecurityToken>@ValueType for binary security token and used as the namespace of the XML based token.

e) Click **Properties**. Define the property with name and value pairs. These pairs are passed to the construct of the CallbackHandler implementation as `java.util.Map`.

Select **None** to deselect the login binding.

b. Under Web Services Security Properties, click **Web services: server security bindings** to edit the request receiver binding information.

1) Under Request Receiver Binding, click **Edit**.

2) Under Additional Properties, click **Login mappings**.

3) Click **New** to create new login mapping.

a) Enter the authentication method, this must match the authentication method defined in the IBM extension deployment descriptor. The authentication method must be unique in the login mapping collection of the binding file.

b) Enter a JAAS Login Configuration name. The JAAS Login Configuration must be defined under **Security > Global security**. Under Authentication, click **Java Authentication and Authorization Service > Application logins**. For more information, see Configuring programmatic logins for Java Authentication and Authorization Service.

c) Enter an implementation of the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface. This is a mandatory field.

d) Enter the token value type, it is optional for BasicAuth, Signature and IDAssertion authentication methods but required for any other authentication method. The token value type is used to validate against the <wsse:BinarySecurityToken>@ValueType for binary security token and against the namespace of the XML based token.

e) Enter the name and value pairs for the ″Login Mapping Property″ by clicking **Properties** . These name and value pairs are available to the JAAS Login Module or Modules by com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback JAAS Callback. **Note**: This is true when editing existing login mappings but not when creating new login mappings.

f) Enter the name and value pairs for the ″Callback Handler Factory Property″, these name and value pairs is passed as `java.util.Map` to the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory.init() method. The usage of these name and value pairs is up to the CallbackHandlerFactory implementation.

c. Click authentication method link to edit the selected login mapping.

d. Click **Remove** to remove the selected login mapping or mappings.

3. Click **Save**.

Results

The previous steps define how to configure the request sender to create security tokens in the SOAP message and the request receiver to validate the security tokens found in the incoming SOAP message. WebSphere Application Server supports pluggable security tokens.

You can use the authentication method defined in the login bindings and login mappings to generate security tokens in the request sender and validate security tokens in the request receiver.

What to do next

After you have configured pluggable tokens, you must configure both the client and the server to support pluggable tokens. See the following topics to configure the client and the server:

- “Configuring the client for LTPA token authentication: specifying LTPA token authentication” on page 650

- “Configuring the client for LTPA token authentication: collecting the authentication method information” on page 651
- “Configuring the server to handle LTPA token authentication information” on page 652
- “Configuring the server to validate LTPA token authentication information” on page 653

Pluggable token support: **Version 5.x application**

Pluggable security token support provides plug-in points to support customer security token types, including token generation, token validation, and client identity mapping to a WebSphere Application Server identity that is used by the Java Platform, Enterprise Edition (Java EE) authorization engine. Moreover, the pluggable token generation and validation framework supports XML-based tokens to be inserted into the Web service message header and validated on the receiver-side validation.

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* applications.

You can extend the WebSphere Application Server login mapping mechanism to handle new types of authentication tokens. WebSphere Application Server provides a pluggable framework to generate security tokens on the sender-side of the message and to validate the security token on the receiver-side of the message. The framework is based on the Java Authentication and Authorization Service (JAAS) Application Programming Interfaces (APIs).

Use the javax.security.auth.callback.CallbackHandler implementation to create a new type of security token following these guidelines:

- Use a constructor that takes a user name (a string or null, if not defined), a password (a char[] or null, if not defined) and java.util.Map (empty, if properties are not defined).
- Use handle() methods that can process the following implementations:
  - javax.security.auth.callback.NameCallback
  - javax.security.auth.callback.PasswordCallback
  - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenCallback
  - com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl

  If:
  1. Either the javax.security.auth.callback.NameCallback or the javax.security.auth.callback.PasswordCallback implementation is populated with data, then a <wsse:UsernameToken> element is created.
  2. com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl is populated, the <wsse:BinarySecurityToken> element is created from the com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl implementation.
  3. com.ibm.wsspi.wssecurity.auth.callback.XMLTokenCallback is populated, a XML-based token is created based on the Document Object Model (DOM) element that is returned from the XMLTokenCallback.

  Encode the token byte by using the security handler and not by using the javax.security.auth.callback.CallbackHandler implementation.

You can implement the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface, which is a factory for instantiating the javax.security.auth.callback.CallbackHandler implementation. For your own implementation, you must provide the javax.security.auth.callback.CallbackHandler interface. The Web service security run time instantiates the factory implementation class and passes the authentication information from the Web services message header to the factory class through the setter methods. The

Web services security run time then invokes the newCallbackHandler() method of the factory implementation class to obtain an instance of the javax.security.auth.CallbackHandler object. The object is passed to the JAAS login configuration.

The following is an example the definition of the CallbackHandlerFactory interface:

```
public interface com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory {
       public void setUsername(String username);
       public void setRealm(String realm);
       public void setPassword(String password);
       public void setHashMap(Map properties);
       public void setTokenByte(byte[] token);
       public void setXMLToken(Element xmlToken);
       public CallbackHandler newCallbackHandler();
```

Configuring the client for LTPA token authentication: specifying LTPA token authentication:

Version 5.x application

To configure Lightweight Third-Party Authentication (LTPA) token authentication, specify LTPA token authentication. Only configure the client for LTPA token authentication if the authentication mechanism configured in WebSphere Application Server is LTPA.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Use this task to configure Lightweight Third-Party Authentication (LTPA) token authentication. Only configure the client for LTPA token authentication if the authentication mechanism configured in WebSphere Application Server is LTPA. When a client authenticates to a WebSphere Application Server, the credential created contains an LTPA token. When a Web service calls a downstream Web service, you can configure the first Web service to send the LTPA token from the originating client. Do not attempt to configure LTPA from a pure client. LTPA works only when you configure the client-side of a Web service acting as a client to a downstream Web service. For the downstream Web service to validate the LTPA token, the LTPA keys on both servers must be the same.

Complete the following steps to specify LTPA token as the authentication method:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > *application_name* > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **Extensions** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section.
7. Select **LTPA** as the authentication method. For more conceptual information on LTPA authentication, see “Lightweight Third Party Authentication” on page 654.

What to do next

After you specify LTPA token as the authentication method, you must specify how to collect the LTPA token information. See “Configuring the client for LTPA token authentication: collecting the authentication method information” on page 651 for more information.

Configuring the client for LTPA token authentication: collecting the authentication method information: **Version 5.x application**

To configure Lightweight Third-Party Authentication (LTPA) token authentication, collect the LTPA token authentication information. Do not configure the client for LTPA token authentication unless the authentication mechanism configured in WebSphere Application Server is LTPA.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

Use this task to configure Lightweight Third-Party Authentication (LTPA) token authentication. Do not configure the client for LTPA token authentication unless the authentication mechanism configured in WebSphere Application Server is LTPA. When a client authenticates to a WebSphere Application Server, the credential created contains an LTPA token. When a Web service calls a downstream Web service, you can configure the first Web service to send the LTPA token from the originating client. Do not attempt to configure LTPA from a pure client. LTPA works only when you configure the client-side of a Web service acting as a client to a downstream Web service. In order for the downstream Web service to validate the LTPA token, the LTPA keys on both servers must be the same.

Complete the following steps to specify how to collect the LTPA token authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **Application Client Projects** > ***application_name*** > **appClientModule** > **META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the **WS Bindings** tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** to view the login binding information and select **LTPA**. If LTPA is not already there, enter it as an option. The login binding dialog is displayed. Select or enter the following information:

   Authentication method
   Specifies the type of authentication that occurs. Select **LTPA** to use identity assertion.

   Token value type URI and token value type local name
   When you select **LTPA**, you must edit the token value type **URI** (Uniform Resource Identifier) and the **local name** fields. Specifies values for custom authentication types, which are authentication methods not mentioned in the specification. For the token value type **URI** field, enter the following string: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`. For the **local name** field, enter the following string: `LTPA`.

   Callback handler
   Specifies the Java Authentication and Authorization Service (JAAS) callback handler implementation for collecting the LTPA information. Specify the com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler implementation for LTPA.

   Basic authentication user ID and basic authentication password
   For LTPA, you can leave these fields empty. However, when you omit this information, the LTPA CallbackHandler implementation attempts to obtain the LTPA token from the invocation (RunAs) credential. If an invocation (RunAs) credential does not exist, then the LTPA token is not propagated.

Property name and property value
For LTPA, you can leave these fields blank.

What to do next

See “Configuring the client for LTPA token authentication: specifying LTPA token authentication” on page 650 if you have not previously specified this information.

Configuring the server to handle LTPA token authentication information:

Version 5.x application

Lightweight Third-Party Authentication (LTPA) is a type of authentication mechanism in WebSphere Application Server security that defines a particular token format. The purpose of the LTPA token authentication is to flow the LTPA token from the first Web service, which authenticated the originating client, to the downstream Web service. You can configure the server for LTPA token authentication.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task is used to configure LTPA. Do not attempt to configure LTPA from a pure client. After the downstream Web service receives the LTPA token, it validates the token to verify that the token has not been modified and has not expired. For validation to be successful, the LTPA keys that are used by both the sending and receiving servers must be the same.

Complete the following steps to specify that LTPA is the authentication method. The authentication method indicated in these steps must match the authentication method that is specified for the client.

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java Platform, Enterprise Edition (Java EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule** > **META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select from the following options:
   - **BasicAuth**
   - **Signature**
   - **ID assertion**
   - **LTPA**
7. Select **LTPA** to authenticate the client using the LTPA token received from the request.

What to do next

After you specify the authentication method, you must specify the information that the server must validate. See “Configuring the server to validate LTPA token authentication information” on page 653 for more information.

Configuring the server to validate LTPA token authentication information:

Version 5.x application

Lightweight Third-Party Authentication (LTPA) is a type of authentication mechanism in WebSphere Application Server security that defines a particular token format. The purpose of the LTPA token authentication is to flow the LTPA token from the first Web service, which authenticated the originating client, to the downstream Web service. You can configure the server to validate LTPA token authentication.

About this task

Note: There is an important distinction between Version 5.*x* and Version 6.0.*x* and later applications. The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6.0.*x* and later applications.

This task is used to configure LTPA. Do not attempt to configure LTPA from a pure client. After the downstream Web service receives the LTPA token, it validates the token to verify that the token has not been modified and has not expired. For validation to be successful, the LTPA keys used by both the sending and receiving servers must be the same.

Complete the following steps to specify how the server must validate the LTPA token authentication information:

1. Launch an assembly tool. For more information, see the related information on Assembly Tools.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Click **EJB Projects** > ***application_name*** > **ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information. The login mapping information is displayed. Select or enter the following information:

   Authentication method
   Specifies the type of authentication that occurs. Select **LTPA** to use LTPA token authentication.

   Configuration name
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the LTPA authentication method, enter `WSLogin` for the JAAS login configuration name. This configuration understands how to validate an LTPA token.

   Use token value type
   Determines if you want to specify a custom token type. For LTPA authentication, you must select this option because LTPA is considered a custom type. LTPA is not in the Web Services Security Specification.

   Token value type URI and local name
   Specifies custom authentication types. If you select **Use Token value type** you must enter data into the Token value Type URI (Uniform Resource Identifier) and local name fields. For the token value type URI field, enter the following string: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`. For the local name, enter the following string: `LTPA`

   Callback handler factory class name
   Creates a JAAS CallbackHandler implementation that understands the following callback handlers:

- javax.security.auth.callback.NameCallback
- javax.security.auth.callback.PasswordCallback
- com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
- com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
- com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

For any of the default authentication methods (BasicAuth, IDAssertion, Signature, and LTPA), use the callback handler factory default implementation. Enter the following class name for any of the default authentication methods including LTPA:

`com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl`

This implementation creates the correct callback handler for the default implementations.

Callback handler factory property
Specifies callback handler properties for custom callback handler factory implementations. Default callback handler factory implementation does not any property specifications. For LTPA, leave this field blank.

Login mapping property
Specifies properties for a custom login mapping. For default implementations including LTPA, leave this field blank.

What to do next

See the task for configuring the server to handle LTPA token authentication information if you have not previously specified this information.

Lightweight Third Party Authentication: **Version 5.x application**

When you use the lightweight third party authentication (LTPA) method, the <wsse:BinarySecurityToken> security token is generated. On the request sender side, the security token is generated by invoking a callback handler. On the request receiver side, the security token is validated by a Java Authentication and Authorization Service (JAAS) login module.

Note: The information in this article supports Version 5.*x* applications only that are used with WebSphere Application Server Version 6.0.*x* and later. The information does not apply to Version 6 and later applications.

The following information describes token generation and token validation operations.

LTPA token generation
The request sender uses a callback handler to generate an LTPA security token. The callback handler returns a security token that is inserted in the SOAP message. Specify the appropriate callback handler in the <LoginBinding> element of the bindings file (`ibm-webservicesclient-bnd.xmi`). The following callback handler implementation can be used with the LTPA authentication method:

- com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler

You can add your own callback handlers that implement the `javax.security.auth.callback.CallbackHandler` property.

When using the LTPA authentication method (or any authentication method other than BasicAuth, Signature or IDAssertion), the TokenValueType attribute of the <LoginBinding> element in the bindings file (`ibm-webservicesclient-bnd.xmi`) must be specified. The values to use for the LTPA TokenValueType attribute are:

- `uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2"`

- localName="LTPA"

LTPA token validation

The request receiver retrieves the LTPA security token from the SOAP message and validates the message using a JAAS login module. The <wsse:BinarySecurityToken> security token is used to perform the validation. If the validation is successful, the login module returns a JAAS Subject. Subsequently, this Subject is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault.

The appropriate JAAS login configuration to use is specified in the bindings file <LoginMapping> element. Default bindings specified in the ws-security.xml file, but these can be overridden using the application-specific ibm-webservices-bnd.xmi file. The configuration information consists of a CallbackHandlerFactory, a ConfigName and a TokenValueType attribute. The CallbackHandlerFactory specifies the name of a class to use to create the JAAS CallbackHandler object. A CallbackHandlerFactory implementation is provided (com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl). The ConfigName attribute specifies a JAAS configuration name entry. The Web services security run time first searches the security.xml file for a matching entry and if a matching entry is not found, the run time searches the wsjaas.conf file. A default configuration entry suitable for the LTPA authentication method is provided (WSLogin). An appropriate TokenValueType element is located in the LTPA LoginMapping section of the default ws-security.xml file.

Tuning Web services security for Version 5.x applications

Version 5.x application

The Java Cryptography Extension (JCE) policy is integrated into the IBM Software Development Kit (SDK) Version 1.4.*x* and is no longer an optional package. However, due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only.

About this task

To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services security might improve.

Results

After following either of these sets of steps, two Java Archive (JAR) files are placed in the JVM directory.

Enabling security for WSIF

The steps to be taken to enable the Web Services Invocation Framework (WSIF) to interact with a security manager.

About this task

WSIF interacts with a security manager in the following ways:
- WSIF runs in the Java Platform, Enterprise Edition (Java EE) security context without modification.

- When WSIF is run under a Java EE container, port implementations can use the security context to pass on security tokens or credentials as necessary.
- WSIF implementations can automatically convert Java EE security context into appropriate context for onward services.

For WSIF to interact effectively with the WebSphere Application Server security manager, complete the following step:

To load the WSDL file, enable the **FilePermission** attribute in the `was.policy` file. This permission is required when a WSDL file is referred to using the `file://` protocol.

Configuring UDDI registry security

The UDDI Version 3 registry is designed to exploit the advantages of WebSphere Application Server security. The registry also supports the UDDI Version 1 and Version 2 security features and the UDDI Version 3 Security API.

About this task

For production use, it is recommended that the UDDI Version 3 registry is configured to use WebSphere Application Server security, and that use of UDDI Version 1 and Version 2 security features and the Version 3 Security API is avoided. However, for solutions with a strong preference for UDDI security, the UDDI Version 3 registry can be configured to enable this, both when WebSphere Application Server security is enabled and when it is disabled.

To configure UDDI registry security, complete the following steps:

1. Follow the appropriate link for the type of configuration you wish to set up:
   - "Configuring the UDDI registry to use WebSphere Application Server security"
   - "Configuring the UDDI registry to use UDDI security" on page 658
2. Review the "UDDI registry security additional considerations" on page 660.

Configuring the UDDI registry to use WebSphere Application Server security

You can configure the UDDI registry to determine whether users are allowed access to services, and to determine security of data at the transport level.

Before you begin

Before you start this task, complete the following two steps:

- Enable WebSphere Application Server security (see Enabling security). This allows the UDDI registry to exploit the WebSphere Application Server security features.
- Ensure that WebSphere Application Server is configured to use HTTPS (SSL). This allows the use of secure access with the UDDI registry. By default, WebSphere Application Server is configured to accept SSL requests on port 9443. To make additional SSL configuration changes, see SSL configurations for selected scopes.

About this task

The UDDI registry exploits two aspects of WebSphere Application Server security:

Authorization
Authorization determines whether users are allowed access to services. WebSphere Application Server determines authorization by mapping users, or groups of users, to roles. UDDI uses two

WebSphere Application Server special subjects: *Everyone* (all users are allowed access) and *AllAuthenticatedUsers* (only valid WebSphere Application Server registered users are allowed access).

Data confidentiality

Data confidentiality determines security at the transport level. Data confidentiality for WebSphere Application Server services can be either 'none' (HTTP is used as the transport protocol) or 'confidential' (requiring the use of SSL; HTTPS is used as the transport protocol).

When WebSphere Application Server security is enabled, the default settings in the UDDI Version 3 Application and Web deployment descriptors produce the following results:

- Publish, Custody Transfer and Security services are mapped to the AllAuthenticatedUsers special subject, and data confidentiality is enforced (HTTPS is used). Authentication uses the standard WebSphere Application Server security facilities and there is no separate registration function for the UDDI registry. To use publish functions, users must supply their WebSphere Application Server user name and password (unless you have modified the supplied publish role), and must also be registered UDDI Publishers. By registering users as UDDI Publishers, you control which users in the AllAuthenticatedUsers subject can update the UDDI registry.
- Inquiry services are mapped to the Everyone special subject, and data confidentiality is not enforced (HTTP is used). To use inquiry services, users do not need to supply a user name or password, and do not need to be registered UDDI publishers.

We recommend that you use the default settings, as described previously. However, you can change the defaults by mapping roles to different users or user groups. If you do this, turn on the **Automatically register UDDI publishers** property (see UDDI node settings) so that you do not need to use two mechanisms to give access to a subset of users. You can also have a role that is not mapped to any users or user groups, in which case all access to that role is disabled.

For more information about UDDI role mappings, and a list of UDDI registry services and roles, see Access control for UDDI registry interfaces.

To change the default settings, use the following steps:

1. Optional: To change the role mappings using the administrative console, complete the following steps:
   a. In the navigation pane, click **Applications** → **Application Types** → **WebSphere enterprise applications**.
   b. In the content pane, click the UDDI registry application.
   c. Under **Detail Properties** click **Security role to user/group mapping**.
   d. Make any changes you require and click **OK**.
2. Optional: To change the role mappings using the wsadmin command, complete the following steps:
   a. Use the MapRolesToUsers option of the edit command of the AdminApp object to map the roles defined in the UDDI registry application to special subjects (Everyone or AllAuthenticatedUsers), to users, or to user groups. For example, the following command maps the Version 3 GUI Publish role to Everyone, and the Version 3 SOAP Publish role to user 'user1' and group 'group1'. UDDI_Registry_Application is a variable that represents the name of the UDDI registry application.

      Using Jython:

```
AdminApp.edit(UDDI_Registry_Application, ["-MapRolesToUsers",
[["GUI_Publish_User", "Yes", "No", "", ""],
["V3SOAP_Publish_ User_Role", "No", "No", "user1", "group1"]]] )
```

      Using Jacl:

```
$AdminApp edit $UDDI_Registry_Application {-MapRolesToUsers {
{"GUI_Publish_User" Yes No "" ""}
{"V3SOAP_Publish_User_Role" No No "user1" "group1"} }}
```

      For more information about using the MapRolesToUsers option, see Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands.

3. To change the data confidentiality settings, refer to Configuring SOAP API and GUI services for the UDDI registry.

Configuring the UDDI registry to use UDDI security

The UDDI Version 3 registry can be configured to enable UDDI security when there is a strong preference for UDDI security, compared to WebSphere Application Server security.

About this task

You can use the UDDI registry security features when WebSphere Application Server security is either enabled or disabled. For example, you can use UDDI security with WebSphere Application Server security disabled for test purposes. Each situation requires different configuration, and results in different behavior.

Note: Do not disable WebSphere Application Server security for production configurations.

To continue configuring the UDDI registry to use UDDI security, choose one of the following options:

- "Configuring UDDI Security with WebSphere Application Server security enabled"
- "Configuring UDDI Security with WebSphere Application Server security disabled" on page 659

Configuring UDDI Security with WebSphere Application Server security enabled

To configure UDDI security, you can use the UDDI Version 3 security API or the UDDI Version 1 and Version 2 publish security features. Because WebSphere Application Server security is enabled, WebSphere Application Server data confidentiality management is independent of UDDI security.

About this task

When WebSphere Application Server security is enabled, use the administrative console to complete the following steps:

1. In the navigation pane, click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. In the content pane, click the UDDI registry application. Under **Detail Properties** click **Security role to user/group mapping**.
3. Set the WebSphere Application Server security role mappings to Everyone for the following UDDI services:
   - Versions 1 and 2 SOAP publish service (SOAP_Publish _User)
   - Version 3 publish service (V3SOAP_Publish_User_Role)
   - Version 3 custody transfer service (V3SOAP_CustodyTransfer_User_Role)
   - Version 3 security service (V3SOAP_Security_User_Role)

   Changing the role mappings to Everyone prevents WebSphere Application Server security from overriding UDDI security.
4. Ensure that UDDI Policy is set to require the use of authentication tokens for the UDDI Version 3 Publish and Custody Transfer services (use of authentication tokens is already required for Version 1 and Version 2 Publish services). To do this, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **API policies**. Select the **Authorization for publish** and **Authorization for custody transfer** check boxes. (Select the **Authorization for inquiry** check box if you require authentication for UDDI Inquiry services).
5. Click **OK**.

Results

With this configuration, no Security Role authentication restriction is imposed, but the credentials (user name and password) associated with the authentication token are authenticated by WebSphere Application Server.

Note: When WebSphere Application Server security is enabled, WebSphere Application Server data confidentiality management is independent of UDDI security and is managed as described in "Configuring the UDDI registry to use WebSphere Application Server security" on page 656.

Configuring UDDI Security with WebSphere Application Server security disabled

To configure UDDI security, you can use the UDDI Version 3 security API or the UDDI Version 1 and Version 2 publish security features. Because WebSphere Application Server security is disabled, authentication tokens will be required for publish and custody transfer requests.

About this task

With WebSphere Application Server security disabled, neither WebSphere Application Server security roles nor data confidentiality constraints apply. This mode may be useful for test UDDI registry configurations.

In this mode, UDDI Version 1 and Version 2 security features are active:

- UDDI Version 1 and Version 2 publish requests require UDDI Version 1 and Version 2 authentication tokens respectively. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.
- UDDI Version 1 and Version 2 inquiry requests do not require authentication tokens.

No further configuration is required for UDDI Version 1 and Version 2 security.

For UDDI Version 3, the use of the UDDI Version 3 Security API, and the use of authentication tokens with Version 3 Publish and Custody Transfer APIs, is optional. To make use of these UDDI Version 3 security features, use the administrative console to complete the following steps:

1. Specify that use of authInfo is required. Click **UDDI** → **UDDI Nodes** > *uddi_node_name*.
2. In the **General Properties** section, select the **Use authInfo credentials if provided** check box.
3. Click **OK**.

Results

Authentication tokens will now be required for publish and custody transfer requests, but not for inquiry requests. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.

Access control for UDDI registry interfaces

Access to UDDI registry interfaces is controlled by a combination of Java platform for enterprise editions declarative security using role mappings, and UDDI properties and policies, such as registering users as UDDI publishers.

Each UDDI registry interface is represented by a security role. The interfaces and their corresponding roles are as follows:

UDDI registry interface	Security role
Version 3 SOAP inquiry	V3SOAP_Inquiry_User_Role
Version 3 SOAP publish	V3SOAP_Publish_User_Role
Version 3 SOAP custody transfer	V3SOAP_CustodyTransfer_User_Role
Version 3 SOAP security	V3SOAP_Security_User_Role

UDDI registry interface	Security role
Version 3 GUI inquiry	GUI_Inquiry_User
Version 3 GUI publish	GUI_Publish_User
Versions 1 and 2 SOAP inquiry	SOAP_Inquiry_User
Versions 1 and 2 SOAP publish	SOAP_Publish_User
EJB inquiry	EJB_Inquiry_Role
EJB publish	EJB_Publish_Role

By default, the inquiry roles are mapped to the *Everyone* special subject and the non inquiry roles are mapped to the *AllAuthenticatedUsers* special subject. With these default settings, after you enable WebSphere Application Server security, you do not need access control to use the UDDI registry inquiry interfaces. However, to use the publish roles and the Version 3 custody transfer role, you must be authenticated using a WebSphere Application Server user id and password. The Version 3 security role is a special case, because it uses UDDI registry security instead of WebSphere Application Server security, and it must be specially configured.

Roles that are mapped to the *AllAuthenticatedUsers* special subject are further protected, because the user must also be registered as a UDDI publisher to publish data to the UDDI registry. If the user is not registered, an E_unknownUser error is returned in the disposition report. You can register users as UDDI publishers in one of two ways:

- Create a new UDDI publisher using the administrative console or the Java Management Extensions (JMX) interface.
- Set the **Automatically register UDDI publishers** property so that users are automatically registered as a publisher on their first publish request.

In accordance with the UDDI specification, there is additional access control, in that for an entity that is published to the UDDI registry, only the user who originally published that entity can update or delete it.

The UDDI registry also provides some management interfaces that are protected by the requirement of administrative permissions for certain operations.

UDDI registry security additional considerations

In addition to the configuration of UDDI registry security, a number of other UDDI registry settings can affect the behavior of the UDDI registry. Some of these settings are security specific and others are points to consider when configuring security.

Security specific considerations

UDDI registry interfaces are protected as detailed in Access control for UDDI registry interfaces.

The UDDI registry supports the use of XML Digital Signatures to sign UDDI entities. See Use of digital signatures with the UDDI registry.

Additional policy considerations

A number of the UDDI property and policy settings also determine the behavior of a UDDI registry with respect to security.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** → ***uddi_node_name***. The settings are also detailed in the administrative console help.

Key space requests require digital signature
This setting determines whether all tModel:keyGenerator requests for key space must be digitally signed. To understand key space, see UDDI registry Version 3 Entity Keys.

Use authInfo credentials if provided
This setting applies only when WebSphere Application Server security is disabled. See Configuring UDDI Security with WebSphere Application Server security disabled.

Authentication token expiry period
The authentication token expiry period is the length of idle time (in minutes) allowed before an authentication token is no longer valid.

Default user name
The default user name is used for publish operations when WebSphere Application Server security is disabled and no authentication token data is supplied.

To review or change the following policy settings, click **UDDI** → **UDDI Nodes** → ***uddi_node_name***. Then under **Policy Groups**, click **API policies**. The settings are also detailed in the administrative console help.

Authorization for inquiry
Specifies whether authorization using authentication tokens is required for inquiry API requests.

Authorization for publish
Specifies whether authorization using authentication tokens is required for publish API requests.

Authorization for custody transfer
Specifies whether authorization using authentication tokens is required for custody transfer API requests.

These policy settings apply when UDDI security features are used and WebSphere Application Server security is enabled. If the UDDI service is mapped to the AllAuthenticatedUsers security role, these settings are overridden. See Configuring UDDI Security with WebSphere Application Server security enabled.

Other considerations

The publish related actions that a registered UDDI publisher can perform are defined by their entitlements, as described in UDDI registry user entitlements.

In addition to the property and policy settings already described, some UDDI keying and user policy settings also influence publish behavior. These settings are not specific to security, but you should consider them because they also place restrictions on successful completion of publish requests.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** → ***uddi_node_name***. The settings are also detailed in the administrative console help

Automatically register UDDI publishers
The UDDI registry requires that publisher entitlements are set before allowing any publish requests. This option automatically registers users with default entitlements.

If this option is not selected, users (and their entitlements) can be registered. See UDDI Publisher settings.

Use tier limits
If selected, tier limits are enforced.

If this option is selected, you need one or more tiers configured (see Tier collection and UDDI Tier settings). Also, ensure that registered UDDI Publishers are assigned to a tier (see UDDI Publisher settings).

To review or change the following property setting, click **UDDI** → **UDDI Nodes** → **uddi_node_name**. Then under **Policy Groups** click **Keying policies**. The setting is also detailed in the administrative console help.

Registry key generation
: If this option is selected, publishers can request key space and, if successful, publish with publisher assigned keys.

UDDI registry user entitlements

UDDI registry user entitlements define the set of publishing related actions that registered UDDI users are entitled to perform.

An important entitlement is the number of entities of each type that a UDDI user can publish. This is controlled by assigning the user to a UDDI publisher tier. Any number of tiers can be defined to the UDDI registry, and some predefined tiers are supplied when the UDDI registry is deployed. A UDDI publisher tier specifies the maximum number of each entity (business, service, binding, tModel, and publisher assertion) that a user assigned to that tier can publish. (See UDDI Tier settings for information about defining UDDI publisher tiers.)

Other entitlements relate to the entitlement of a user to allocate key spaces, where they can specify publisher assigned keys when publishing UDDI entities. A key space is allocated by publishing a keyGenerator tModel, and there are a number of entitlements relating to different kinds of key generator. For full details of these entitlements, see UDDI Publisher settings. For more information about key generators and publisher assigned keys, see UDDI registry Version 3 Entity Keys.

The entitlements for a UDDI user can be set using the administrative console, or through Java Management Extensions (JMX) using the UDDI Administrative interface, as described in UDDI node collection and UDDI registry Administrative (JMX) Interface.

Security API for the UDDI Version 3 registry

The UDDI Version 3 registry has an independent security API, unlike UDDI Version 1 and Version 2, where the security API was part of the publish API.

To access all API calls and arguments that are supported by the UDDI Version 3 registry programmatically, use the UDDI Version 3 Client for Java (see UDDI Version 3 Client). To access the API functions graphically, you can use the UDDI user interface, but not all functions are available with this method.

The UDDI Version 3 registry supports the following Security API calls:

discard_authToken
: Informs a node that a previously obtained authentication token is no longer required and is no longer valid if it is used after this message is received. The token is discarded and the session is effectively ended.

get_authToken
: Requests an authentication token in the form of an authInfo element from a UDDI node.

For full details of the query syntax, refer to the UDDI Version 3 API specification at http://www.uddi.org/pubs/uddi_v3.htm.

Chapter 6. Service integration

Security

This topic is an overview of the tasks for administering messaging security for a service integration bus.

Before you begin

Review the security requirements for the bus. For guidance, see Planning your security requirements.

About this task

Messaging security protects the bus from unauthorized access. By default, messaging security is enabled for the bus. Providing administrative security is also enabled, messaging security enforces a security policy that prevents unauthorized client applications from connecting to the bus, and accessing bus resources. There may be circumstances when you do not require messaging security, for example on a development system. In this case, you can disable messaging security.

You can customize the security configuration for the bus using the administrative console, or wsadmin scripting commands. The security configuration controls the following aspects of bus security:

- Authorizing groups of users in the user registry to perform selected operations on bus destinations.
- The transport policies that maintain the integrity of messages in transit on the bus.
- The use of global, and multiple custom security domains.
- The integrity of links between messaging engines, foreign buses and databases.

Use the following tasks to administer messaging security:

- “Securing buses”
- “Enabling client SSL authentication” on page 672
- “Adding unique names to the bus authorization policy” on page 673
- “Administering authorization permissions” on page 674
- “Administering permitted transports for a bus” on page 700
- “Securing messages between messaging buses” on page 703
- “Securing access to a foreign bus” on page 704
- “Securing links between messaging engines” on page 704
- “Controlling which foreign buses can link to your bus” on page 705
- “Securing database access” on page 705
- “Securing mediations” on page 705

Securing buses

Securing a service integration bus provides the bus with an authorization policy to prevent unauthorized users from gaining access. If a bus is configured to use multiple security domains, the bus also has a security domain and user realm to further enforce its authorization policy.

Before you begin

- If administrative security for the cell in which the bus resides is not already enabled, you must enable it. The tasks below use an administrative console wizard that detects if administrative security is not enabled, and takes you through the steps to enable it. You need to supply the type of user repository used by the server, and the administrative security username and password.
- If the bus contains a bus member at WebSphere Application Server Version 6.x, you must provide an authentication alias to establish trust between bus members, and to enable the bus to operate securely. The administrative console wizard detects if an authentication alias is required, and prompts you to supply one. If you want to specify a new authentication alias, you must provide a username and password.

© Copyright IBM Corp. 2008

About this task

When you secure a bus, consider the following points:

- If you are securing a WebSphere Application Server Version 7.0 bus that contains only Version 7.0 bus members, you can use a non-global security domain for the bus. If the bus has a WebSphere Application Server Version 6.x bus member, or might have a Version 6.x bus member in the future, you must assign the bus to the global security domain.
- If you want to assign the bus to a custom domain, you can select an existing security domain, or create a new one.
- If you assign the bus to a custom domain, you must specify a user realm. You can select an existing user realm, or use the global user realm.

What to do next

- The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.
- Use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Disabling bus security

If you do not want to protect the bus from unauthorized access, you can disable messaging security.

Before you begin

- Ensure that there are no indoubt transactions on the messaging engine because incomplete transactions cannot be recovered after the bus security is disabled. For more information, refer to Resolving indoubt transactions.
- Stop all servers on which the SIB Service enabled before you disable bus security. This ensures that the bus security configuration is applied consistently when the servers are restarted. For more information, refer to Stopping an application server.

About this task

Messaging security protects the bus from access by unauthorized client applications. Messaging security is enabled by default, providing administrative security for the cell is enabled. However, there may be circumstances when you do not require messaging security, for example on a development system. In this case, you can use the administrative console to disable messaging security. If you want to re-enable bus security, refer to “Securing buses” on page 663.

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed.
2. Find the bus for which you want to disable security, and click **Enabled** in the security column. The security settings for the selected bus are displayed.
3. Clear the checkbox **Enable bus security**.
4. Click **Apply**.
5. Save your changes to the master configuration.

Results

You have disabled security for the selected bus.

What to do next

You must restart the servers. Starting an application server provides more information.

Adding a secured bus

This task uses an administrative console wizard to add a new service integration bus that uses a security domain. A bus can use the global security settings, inherit the security settings that exist at cell level, or use a set of customized settings that are unique to the bus, or shared with another resource.

Before you begin

You must decide which type of security domain to use for the bus. For more information, refer to Planning your security requirements and Messaging security and multiple security domains.

About this task

By default, administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, the wizard prompts you to enable it. You must specify the type of user repository, and the administrative security username and password.

Security settings are held in a security domain. If the bus might contain a WebSphere Application Server Version 6.x bus member, you must select the global domain. You can select any type of domain for a bus that contains only WebSphere Application Server Version 7.0 bus members.

1. Log into the administrative console.
2. Click **Service integration** → **Buses**. A list of buses is displayed.
3. In the content pane, click **New**.
4. Type a name for the new bus. It is advisable to choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after a bus is created, so if you need to interoperate with WebSphere MQ in the future, it will be much simpler if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.
5. Ensure that the **Bus security** checkbox is checked.
6. Click **Next**. The Bus Security Configuration wizard is launched.
7. Read the Introduction panel, and click **Next**.
8. If administrative security is disabled, follow the instructions in the wizard to select, and configure the appropriate user repository.
9. Click **Next**. A summary of the administrative security settings for the bus is displayed.
10. Review the summary, and click **Finish**. Administrative security for the cell is now enabled.
11. By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** .
12. Select a security domain for the bus. The domain you specify depends on the versions of the bus members. If the bus may contain a Version 6.x bus member, you must select the global domain. If the bus will only contain Version 7.0 bus members, you can select the cell-level or a custom domain. If you select a custom security domain, you must also follow the instructions to specify a user realm.
13. Review the summary of your choices, and click **Finish**.
14. Save your changes to the master configuration.

Results

Administrative security is enabled, and a new bus is created with your chosen security settings.

What to do next

You can now add bus members to the bus.

Securing an existing bus using multiple security domains

You can configure an existing service integration bus so that it uses a security domain other than the default global security domain. Using non-global security domains provides the scope to use multiple security domains. The bus can inherit security settings from the cell, or have a unique security configuration.

Before you begin

- For more information about using security domains, refer to Planning your security requirements and Messaging security and multiple security domains.
- The bus you want to secure must exist in the administrative console. If you want to create a new secured bus, refer to “Adding a secured bus” on page 665.
- All the bus members must be at WebSphere Application Server Version 7.0; use of multiple security domains is not supported for earlier versions of WebSphere Application Server. If the bus you want to secure has a WebSphere Application Server Version 6.x bus member, refer to “Securing an existing bus using the global security domain” on page 667.

About this task

This task uses the Bus Security Configuration wizard to secure a selected bus, and assign it to a security domain. If administrative security for the cell is disabled, the wizard prompts you to enable it. You need to know the type of user repository, and the administrative security username and password. If you specify a custom domain, you must also specify a user realm.

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The general properties for the selected bus are displayed.
2. Click **Configure Bus Security** to launch the Bus Security Configuration wizard.
3. Read the Introduction panel, and click **Next**.
4. If administrative security is disabled, follow the instructions to configure the appropriate user repository, and click **Next**.
5. Review the summary of your choices:
   a. If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
   b. Click **Finish** when you are ready to confirm your choices.

   Administrative security for the cell is now enabled.
6. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** . By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity.
7. Select the cell-level or custom security domain for the bus. You can only select a non-global security domain if all the bus members are at Version 7.0.
8. Optional: To create a new custom security domain:
   a. Use the name suggested for the security domain, or type a new one.
   b. Optional: Provide a description of the security domain.
   c. Select a user realm for the domain. You can use the user realm configured in the global security domain, or follow the steps to configure a new user realm.
9. Click **Next**.
10. Review the summary of your choices:
    a. Optional: If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
    b. Click **Finish** to confirm your choices.
11. Save your changes to the master configuration.

Results

You have specified that the selected bus uses a non-global security domain. The security settings configured for the bus are displayed in the updated Bus Security Settings panel. The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Securing an existing bus using the global security domain

Use this task to secure an existing service integration bus using the global security domain.

Before you begin

- The bus you want to secure must exist in the administrative console. If you want to create a new secured bus, refer to “Adding a secured bus” on page 665.
- If administrative security for the cell in which the bus resides is disabled, the wizard prompts you to enable it. You need to know the type of user repository, and the administrative security username and password.
- If the service bus contains a bus member at WebSphere Application Server Version 6.x, the wizard prompts you to select an existing authentication alias, or specify a new one. If you want to specify a new authentication alias, you must provide a username and password.

About this task

Use this task if you want to secure a bus that exists already in the administrative console, and you want to use the global (default) security domain. For example, you are securing a Version 7.0 bus that has a bus member at WebSphere Application Server Version 6.x; mixed version buses cannot use multiple security domains.

This task uses an administrative console wizard to guide you through the steps to secure a bus. The following steps are conditional, depending on the bus environment:

- If administrative security for the cell in which the bus resides is disabled, the wizard prompts you to enable administrative security.
- If the bus has a bus member at WebSphere Application Server Version 6.x, the wizard prompts you for an authentication alias to establish trust between bus members, and to enable the bus to operate securely.

Use the administrative console to secure a selected bus using the global security domain as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The general properties for the selected bus are displayed.
2. Click **Configure Bus Security** to launch the Bus Security Configuration wizard.
3. Read the Introduction panel, and click **Next**. The next step is conditional, depending on whether administrative security is enabled or disabled:
   - If administrative security is disabled, complete all the following steps.
   - If administrative security is already enabled, continue from step 7.
4. Select the appropriate user repository, and click **Next**.
5. Depending on the type of user registry you selected, do one of the following:
   - For a federated repository, specify a username and password for administrative security, and click **Next**.
   - For all other types of repository, follow the wizard prompts, and click **Next**.

6. Review the summary of your choices:
   a. If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
   b. Click **Finish** when you are ready to confirm your choices.

   Administrative security for the cell is now enabled.
7. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** . By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity.
8. Select the global security domain option, and click **Next**.
9. If at least one bus member is at Version 6.x, you must specify an authentication alias. Specify either an existing authentication alias, or create a new one:
   - Select **Specify existing authentication alias**, and select the alias name from the list box.
   - Select **Create a new authentication alias**, type a unique alias name and password.
10. Review the summary of your choices:
    a. Optional: If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
    b. Click **Finish** to confirm your choices.
11. Save your changes to the master configuration.

Results

You have secured the bus using the global security domain. The new security settings for the bus are displayed in the updated Bus Security Settings panel. The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Migrating an existing secure bus to multiple domain security

Use this task to migrate a secured service integration bus that uses the global security domain to using non-global security domains.

Before you begin

All the bus members must be at WebSphere Application Server Version 7.0 or later; use of multiple domain security is not supported for earlier versions of WebSphere Application Server.

About this task

When you migrate a bus from using the global security domain to using a non-global security domain, you configure the bus to use a either an existing non-global domain, or you create a new security domain. If you create a new security domain, you must also specify a user realm. You can choose to use the existing global security settings, or customize a user realm specifically for your domain.

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The security settings panel for the selected bus are displayed.
2. Click **Use the selected domain**.
3. To use an existing domain, select its name in the list box.
4. To create a new security domain: .
   a. Click **Apply**.

b. Click the link **Configure Security Domain** which is now available. The security domain configuration panel for the selected bus is displayed.
c. Use the name suggested for the security domain, or type a new one.
d. Optional: Type a description of the security domain.
e. Select the type of user realm for the domain. You can either use the global security settings, or configure a new one.

5. Click **Next**.
6. Review the summary of your choices:
   a. Optional: If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
   b. Click **Finish** to confirm your choices.
7. Save your changes to the master configuration.

Results

You have migrated your existing bus from using the global domain to a non-global security domain. The new security settings for the bus are displayed in the updated Bus Security Settings panel.

What to do next

The bus will start using the new security domain after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.

Configuring bus security using an administrative console panel

Use this task to enable or disable messaging security, and configure security properties for an existing service integration bus.

Before you begin

The bus must exist in the administrative console. If you want to create a new bus, refer to Adding buses.

About this task

This task uses the Bus Security administrative console panel. You can launch the Bus security wizard from the panel, or specify individual security properties directly in the panel. The bus security properties are effective only when administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, it prompts you to enable it.

The security properties available to a particular bus depend on the versions of the bus members. If the bus has a WebSphere Application Server Version 6.x bus member, you must specify the global security domain. The bus cannot use cell level, or custom security domains. You must also specify an inter-engine authentication alias to prevent unauthenticated messaging engines from establishing a connection with the bus. If the bus contains Version 7.0 bus members only, you can specify any type of security domain, and you do not need to specify an inter-engine or mediation authentication alias.

If you want to run mediations across multiple security domains, you can specify a single server identity for the bus, rather than specify a mediation authentication alias for each domain. You can also use a server identity to run mediations on the global domain.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value***. *security_value* is either `Enabled` or `Disabled`, depending on the security status of the bus.
3. Click **Launch Bus Security Wizard** to launch the wizard, or specify the following properties directly:

Enable bus security
Bus security is enabled by default. Clear this check box if you do not want a secure bus. The check box is read-only if administrative security is disabled.

Inter-engine authentication alias
The name of the authentication alias used to authorize communication between messaging engines on the bus. Specify an inter-engine authentication alias if the bus has a Version 6.x bus member, bus security is enabled, and you want to prevent unauthorized messaging engines from establishing a connection with the bus.

Permitted transports
Specify one of the following transports for the bus:

- Any messaging transport chain defined to any bus member.
- Only messaging transport chains that are protected by an SSL chain.
- Only the transports specified in the list of permitted transports.

If you want to add and remove permitted transports, click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**.

Use the Server ID when running mediations
Check this option if you want to run mediations using the server identity, instead of using a mediation authentication alias.

Mediations are deployed as applications, and run in the domain used by the application server, not the bus domain. If you want to run a mediation on multiple servers in different domains, you must ensure that the user identity in the mediation authentication alias exists in the configuration for each domain. Alternatively, you can choose to use the server identity option. You can use this option when multiple domains are not in use.

Bus security domain
Specify one of the following security domains for the bus:

Global domain
You must specify the global domain if the bus contains a Version 6.x bus member, or you do not want the bus to use multiple domains.

Cell level domain
Specify the cell-level security domain if the bus has Version 7.0 bus members only, and you want the bus to share security settings with the administrative cell.

Custom domain
Specify a custom security domain if the bus has Version 7.0 bus members only, and you want the bus to use a security domain that is used by another resource, or you want to create a new security configuration for this bus.

4. Save your changes to the master configuration.

Results

You have configured security properties for the selected bus.

What to do next

You can use the administrative console to control access to the bus.

Configuring the bus to access secured mediations

Use this task to ensure that the service integration bus is authorized to access secured mediations.

Before you begin

The mediation is secured using a Java Platform, Enterprise Edition (Java EE) Connector Architecture authentication alias. For information about creating a Java EE authentication alias, refer to Managing J2EE Connector Architecture authentication data entries.

About this task

To configure the bus to access a secured mediation, you must add the mediation authentication alias for the secured mediation to the properties for the bus:

- If the bus has a Version 6.x bus member, you must provide the principal and its associated password.
- If the bus has WebSphere Application Server Version 7.0 bus members only, you need only provide the principal.

1. Log into the navigation pane.
2. Click **Service integration** → **Buses** → ***security_value***. The bus security configuration panel is displayed.
3. In the **Mediations authentication alias** field, select the principal for the mediation, and its associated password if required.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

The selected bus is configured to access secured mediations.

What to do next

You can assign security roles to your mediation handlers to protect them from use by unauthorized users. For more information, see Deploying secured applications.

Configuring a bus to run mediations in a multiple security domain environment

Use this task to configure a secured bus so that it can run mediations successfully on bus members in different security domains.

Before you begin

The secured bus must be configured to use a non-global security domain. For more information about securing buses using multiple security domains, refer to “Securing buses” on page 663.

About this task

If your bus topology has bus members in different security domains, you can configure the bus to allow mediations to run under the server identity. This means that a mediation can run on any server in any domain. You do not have to add a dedicated user ID for each mediation to the user repository, or maintain a mediation authentication alias.

Use the administrative console to configure a secured bus to run mediations successfully as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The security settings for the selected bus are displayed.
2. Check the option **Use the Server ID when running mediations**.
3. Click **Apply**.
4. Save your changes to the master configuration.

Results

You have configured the bus to run mediations successfully across servers in multiple security domains.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Enabling client SSL authentication

You can configure a service integration bus to allow connecting client JMS applications to authenticate using Secure Sockets Layer (SSL) certificates.

About this task

This is the parent task for the steps required to establish client SSL authentication for connections between messaging engines and JMS applications running in a client container. You need to configure the bus to allow client SSL authentication, and configure the JMS client application to perform client SSL authentication. See the following topics:

- "Configuring a bus to allow client SSL authentication."
- "Configuring JMS client applications to perform client SSL authentication" on page 673.

Configuring a bus to allow client SSL authentication

This task describes how to configure a service integration bus to enable connecting client JMS applications to authenticate using Secure Sockets Layer (SSL) certificates.

Before you begin

You must ensure that the following tasks have been completed:

- Administrative security is enabled. For more information, see Enabling security.
- A standalone Lightweight Directory Access Protocol (LDAP) user registry has been configured for storing user and group IDs. To access the user registry, you must know a valid user ID that has the administrative role, and password, the server host and port of the registry server, and the base distinguished name (DN). For more information, see Configuring Lightweight Directory Access Protocol user registries.
- Bus security is enabled. For more information, see "Disabling bus security" on page 664.
- JMS client applications have been configured to authenticate using client SSL certificates.

About this task

If you want to allow connecting JMS application clients to authenticate to the bus using client SSL certificates you need to define an SSL configuration. There are two parts to this task. First you use the administrative console to map SSL certificates to entries in the LDAP user registry. Secondly, you create a unique SSL configuration for each endpoint address for which you want to use client SSL authentication. Do not use the default SSL configuration for the bus.

1. Use the administrative console to define certificate filters to map an SSL certificate to an entry in the LDAP server. For more information, see Creating a Secure Sockets Layer configuration. The client SSL certificate is mapped to a user ID in the user registry.
2. Create a separate SSL configuration file for each endpoint address for server, bus member or cluster on the bus, and select that client authentication is required. For more information, see Creating a Secure Sockets Layer configuration

Results

The bus is configured to allow client SSL authentication.

What to do next

Connecting JMS client applications can now authenticate to the bus using client SSL certificates.

Configuring JMS client applications to perform client SSL authentication

This task describes how to configure JMS client applications to authenticate to the bus using client Secure Sockets Layer (SSL) authentication.

Before you begin

- You have already obtained a Secure Sockets Layer (SSL) certificate for the JMS client application.
- The JMS client application is already configured to use SSL. For more information, see SSL client properties file

About this task

This task has two objectives. First, you install the SSL certificate for the client application in the key store for the application client. Secondly, you modify the sib.client.ssl.properties file to use client SSL authentication. You use the Key Management (iKeyman) utility to work with SSL certificates. The iKeyman user interface is Java-based and uses the Java support that is installed with IBM HTTP Server.

Take the following steps to configure a JMS client application to use client SSL authentication:

1. Start the iKeyman user interface. Refer to the *iKeyman User's Guide* available from IBM developer kits for more information about using iKeyman.
2. When prompted, select the key store for the JMS client application.
3. When prompted for the type off certificate to work with, select the option **Personal certificates**. A list of personal certificates is displayed.
4. Select that you want to import a certificate to the selected key store.
5. When prompted, type the location and name for the certificate. You can provide an alias for the certificate. The certificate is installed into the keystore of the client application.
6. Close the iKeyman user interface.
7. Open a text editor to work with the sib.client.ssl.properties properties file. This file is located in the profile_root/properties directory of the application server installation, where profile_root is the directory in which profile-specific information is stored.
8. Set the value for the property com.ibm.ssl.client.clientAuthentication to *True*.
9. Set the value for the property com.ibm.ssl.client.keyStoreClientAlias to the alias name for the certificate in the client key store.
10. Save the sib.client.ssl.properties properties file.

Results

You have now configured a JMS client application to use client SSL authentication.

Adding unique names to the bus authorization policy

How to update the authorization policy for the service integration bus with unique name entries.

About this task

You should carry out this task if you are migrating from WebSphere Application Server Version 6.x to WebSphere Application Server Version 7.0. In this task, you manually run the populateUniqueNames

command to query the user repository for a selected bus for unique names, and add them to the authorization policy. If you do not manually run this command, the messaging engine performs the query, and adds the missing unique names to the authorizations policy, which adversely affects the start up time.

When you migrate from a Version 6.x node to a Version 7.0 node, the authorization policy only contains the user and group security names; it does not contain the names in the user registry that uniquely define each user and group. If an LDAP user registry is in use, the unique name is the distinguished name (DN). By default, only missing unique names are added to the authorization policy. If you set the **-force** parameter, all unique name entries added to the authorization policy

1. Launch a scripting command. For more information, refer to Starting the wsadmin scripting client.
2. At the wsadmin command prompt, type the populateUniquenames command. The following example syntax queries the user repository for the unique names that match the security names for a bus called Bus 1, and adds the missing unique names to the authorization policy .

   ```
   AdminTask.populateUniquenames('[-bus Bus1]')
   ```
3. Save your changes to the master configuration repository. The following example presents the syntax:

   ```
   AdminConfig.save()
   ```

Results

The authorization policy for the bus is updated with the missing unique names.

Example

The following example updates all the unique name entries in the authorization policy for a bus called Bus 1.

```
AdminTask.populateUniqueNames(AdminTask.populateUniquenames('[-bus Bus1 -force TRUE]')
```

What to do next

Use the administrative console to administer bus security authorizations.

Administering authorization permissions

Service integration messaging security uses role-based authorization. When a user is assigned to a role, the user is granted all of the permissions that the role contains. By administering authorization permissions, you can control user access to a bus and its resources when messaging security is enabled.

Before you begin

For guidance on security authorization for a service integration bus, refer to Planning your security requirements.

About this task

When a bus is created, a set of default authorization roles is created. Default roles provide authenticated users who have the bus connector role with full access to all local destinations on the bus. By default, only members of the Server group have the bus connector role. If a specific user needs to connect to the bus, you must explicitly add that user to the bus connector role.

You can make changes to authorization permissions when messaging security is enabled or disabled. Any changes that you make when security is disabled do not have any effect until security is enabled, as described in “Disabling bus security” on page 664.

Note: When you specify the group authorization permissions, the group distinguished name (DN) must be used. If you specify a common name (CN) for the group name, users in that group do not have the specified authorities. For more details see Standalone Lightweight Directory Access Protocol registries.

When security is enabled, by default users cannot connect to a foreign bus. If a specific user needs to connect to a foreign bus, you must explicitly add that user to the foreign bus access list.

Use the following tasks to administer authorization permissions for a bus to meet your security requirements.
- "Administering the bus connector role"
- "Administering default roles" on page 677
- "Administering destination roles" on page 680
- "Administering foreign bus roles" on page 685
- "Administering topic space root roles" on page 692
- "Administering topic roles" on page 695
- Listing security roles for service integration by using the wsadmin tool
- Removing users and groups by using the wsadmin tool
- Removing authorization data by using the wsadmin tool

Administering the bus connector role

Service integration bus security uses role-based authorization. By administering groups of users in the bus connector role, you can control access to the local service integration bus and its resources when messaging security is enabled.

About this task

Adding a group of users to the bus connector role for a local bus grants the members of the group permission to access local bus destinations. The users and groups you want to work with must exist in the user repository.

Use the following tasks to list, add and remove groups of users in the bus connector roles using the administrative console.
- "Listing users and groups in the bus connector role"
- "Adding users and groups in the bus connector role" on page 676
- "Removing users and groups from the bus connector role" on page 677

Listing users and groups in the bus connector role:

Service integration bus security uses role-based authorization. By listing the users and groups in the bus connector role, you can find out which users and group members are authorized to connect to a selected secured local bus, and its resources.

Before you begin

Ensure that security is enabled for the bus. For more information, refer to "Securing buses" on page 663.

About this task

In this task you use the administrative console to list the users and groups in the bus connector role for a selected local bus. By default, the list is empty for a new bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**.

Results

A list of the users and groups in the bus connector role for the selected bus is displayed. The list is empty for a new bus.

What to do next

You can add and delete users and groups in the bus connector role for the selected bus.

Adding users and groups in the bus connector role:

Service integration bus security uses role-based authorization. By adding users and groups to the bus connector role for a secured bus, you can control which users and group members have access to the bus and its resources.

Before you begin

- Ensure that security is enabled for the bus. For more information, refer to "Securing buses" on page 663.
- The users and groups that you want to add to the bus connector role must exist already in the user repository.

About this task

Adding users and groups to the bus connector role enables them to connect to the bus to carry out messaging operations. You can add a user directly to the bus connector role, or indirectly by adding a group to which the user belongs. You can also add special groups of users. There are three special groups:

Server
The server identity is a WebSphere Application Server . You cannot specify the Server group for a JMS message-driven bean (MDB).

All Authenticated
This group comprises all user identities that authenticate successfully to the bus.

Everyone
The user identities in this group are anonymous, and connect to the bus without security authentication.

Note:

- If the user registry is an LDAP registry, you must use the group distinguished name (DN) when you specify a group name to add to a bus connector role. Using the common name (CN) causes problems in security authorization. For more information, refer to Service integration bus security: Troubleshooting tips and Standalone Lightweight Directory Access Protocol registries.
- If you attempt to add a user or a group that is already a member of the bus connector role, a warning message is displayed.

In this task you use an administrative console wizard to add groups and users to the bus connector role for a selected local bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**. A list of the users and groups already in the bus connector role for the selected bus is displayed. By default, the list is empty for a new bus.
3. Click **New** to launch the Security Resource Wizard.
4. Choose whether you want to add groups or users:

- If you want to add a special group, select **The built-in special groups** option.
- If you want to add other groups or users in the user repository, select the appropriate option, and complete the following mandatory fields:

  Search pattern
  Specify a string to match against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. You can specify wildcard characters.

  Maximum number of search results to display
  Specify the maximum number of user IDs or group names to display.

5. Click **Next** to display a list of groups or users.
6. Select the names of the groups or users you want to add to the bus connector role, and click **Next**.
7. Click **Finish** to confirm you choices.
8. Save your changes to the master configuration.

Results

The selected users and groups are added to the bus connector role for the selected bus.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from the bus connector role:

Service integration bus security uses role-based authorization. By removing selected users and groups from the bus connector role for a selected secured bus you prevent those users and group members from connecting to the bus.

About this task

The users and groups that you remove from the bus connector role for a bus can no longer perform messaging operations on the bus. Note that removing a user from the bus connector role does not prevent that user from connecting to the bus if they are also a member of a group that is in the bus connector role.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**. A list of the users and groups in the bus connector role for the selected bus is displayed.
3. Select the check box next to the name of the user or group that you want to remove, and click **Delete**.
4. Save your changes to the master configuration.

Results

The selected users and groups are removed from the bus connector role.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering default roles

Service integration bus security uses role-based authorization. By adding a user or a group to the default roles for a secured bus, you can control which users and group members have access to the bus and its resources in the default roles when messaging security is enabled.

About this task

The default roles are sender, receiver, browser, and creator. These roles apply to bus destinations that do not have a destination role, or have been configured to inherit the default roles. By default, all destinations inherit the default roles. Access in the default roles exists in addition to any specific access roles that have been configured for a destination.

Use the following tasks to list, add, and remove groups of users in default roles using the administrative console.
- "Adding users and groups to default roles"
- "Removing users and groups from default roles" on page 679
- "Listing users and groups in default roles"

Listing users and groups in default roles:

Service integration bus security uses role-based authorization. By listing the users and groups in the default roles for a selected secured bus, you can find out which users and group members are authorized to perform messaging operations on a local bus destinations that is allowed to inherit default roles.

About this task

In this task you use the administrative console to list users and groups in the default access roles for a selected secured bus. The default role types are sender, receiver, browser and creator.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed. The information for the default access is displayed in a collapsed section.
3. Expand the Default access header.

Results

A list of users and groups in default roles for the selected bus is displayed.

What to do next

You can also add and remove users and groups in default roles.

Adding users and groups to default roles:

Service integration bus security uses role-based authorization. By adding selected users and groups to the default roles for all the local bus destinations on a secured bus, you provide those users and group members with access to the local bus destinations that are allowed to inherit default roles.

Before you begin

If a bus destination is not allowed to inherit the default roles, you must first add the user or group to the role that grants authorization permission for the specific local destination. For more information, see "Adding users and groups to destination roles" on page 681.

About this task

The default roles are sender, receiver, creator and browser. In this task you use an administrative console wizard, the Security wizard, to add selected users or groups to the default roles. The Security wizard requests information to enable it to retrieve selected users or groups from the potentially very large number of users and groups in the user repository.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed.
3. Expand the Default access header to list the users and groups that have been assigned to default access roles.
4. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to default access roles:
   a. Search for the users or groups that you want to add to default access roles:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to the default access roles, and click **Next**. A list of user IDs or group names that you can add to the default access roles is displayed. Note that some users or groups may already be assigned to default access roles.
   d. Select the role types that you want to assign to a user or group. For example, to assign a group to the sender role, click the sender icon for the appropriate group name. The icon changes from to to show that you have added the user or group to the access role for the resource.
   e. Complete the previous step for each user or group that you want to add to access roles, and then click **Next**. A summary of your role type assignments is displayed.
   f. Click **Previous** to review and change your assignments, if required.
   g. Click **Finish** to confirm your assignments. The Default access roles panel is redisplayed and shows the new role type assignments.
5. Save your changes to the master configuration.

Results

The selected users and groups are added to selected default roles for the selected bus.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from default roles:

Service integration bus security uses role-based authorization. By removing selected users and groups from the default roles for a selected secured bus, you prevent those users and group members from accessing the bus using the default roles.

About this task

When you remove users and groups from the default roles for a bus, they can no longer access local bus destinations that inherit default access permissions. Note that removing a user from the default roles does not prevent that user from accessing the bus if they are also a member of a group that has been assigned to the default roles for that bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed.
3. Expand the Default access header to list the users and groups that have been assigned to default access roles for the selected bus.
4. Select the users and groups that you want to remove from the default access roles for this bus, and click **Remove**.
5. Save your changes to the master configuration.

Results

The selected users and groups are removed from the default roles for the selected bus. The Default access roles panel displays the changes to the default access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering destination roles

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups must have authority to perform messaging operations, at a bus destination. By administering destination roles, you can control which users and groups can perform operations at a bus destination, and the type of operation that they can perform.

About this task

You use the administrative console to administer users and groups in access roles for a destination. The access roles available for a destination depend on the type of destination. The table below lists the roles that you can assign for each destination type:

Destination type	Access roles
queue	sender, receiver, browser, creator
port	sender, receiver, browser, creator
webService	sender, receiver, browser, creator
topicSpace	sender, receiver
foreignDestination	sender
alias	sender, receiver, browser

In addition to controlling which users and groups have access to a specific local or foreign destination, you can also control the inheritance of access roles for a specific local destination. In this case, the default access roles that apply to all the destinations in the local bus namespace are added to any access roles that have been added for a specific destination.

Use the following tasks to administer destination roles.

- Listing users and groups in bus destination roles by using the wsadmin tool
- Adding users and groups to bus destination roles by using the wsadmin tool
- Removing users and groups from bus destination roles by using the wsadmin tool

- Defining destination defaults inheritance by using the wsadmin tool
- Determining destination defaults inheritance by using the wsadmin tool

Listing users and groups in destination roles:

Service integration bus security uses role-based authorization. By listing the users and groups in the destination roles for a selected secured bus, you can find out which users and groups are authorized to access the bus, and its resources.

About this task

In this task you use the administrative console to list all the users and groups in destination roles for selected destinations. The list includes users and groups that have references in the service integration role-based configuration; it does not include all the users and groups that exist in the user repository. The permitted destination roles are sender, receiver, browser and creator, depending on the destination type. Icons are used in the administrative console to represent the roles to which users and groups have been assigned. For example, if the role type set icon ( ) is displayed in the sender role for a group called Group 1, it means that Group 1 has been assigned to the sender role for a selected destination. For a complete description of all the icons used to represent role assignments in the administrative console, refer to Access role assignments for bus security resources.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destinations panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination header.

Results

The Destination access roles panel lists the users and groups in access roles for the expanded destination.

What to do next

You can now administer the users and groups in destination roles at this destination.

Adding users and groups to destination roles:

Service integration bus security uses role-based authorization. By adding users and groups to the destination roles for a secured bus, you can control which users and group members can perform messaging operations at a bus destination.

Before you begin

Ensure that the following conditions are met:

- Security is enabled for the bus. For more information, refer to “Securing buses” on page 663.
- The users and groups that you want to add to destination roles must exist already in the user repository.

About this task

By adding users or groups to destination role, you grants the users or groups authority to perform the operation defined by the role at a selected destination. The destination roles are sender, receiver, browser, and creator, depending on the destination type.

In this task you use the administrative console Security wizard to retrieve selected users or groups from the user repository, and add them to destination roles for selected bus destinations.

Note: To add a large number of users to destination roles, it is advisable to create a group in the user repository, and add the group to the destination roles.

1. Start the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destination to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles for this destination. You can verify that the user or group you want to add does not already have a role at this destination.
5. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to access roles for the expanded destination:
   a. Search for the users or groups that you want to add to access roles for the expanded destination:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to access roles for the currently expanded destination, and click **Next**. A list of user IDs or group names that you can add to destination roles is displayed. Note that some users or groups may already be assigned to access roles for this destination.
   d. Select the appropriate access role icon for the user ID or group name that you want to add to the role at this destination. For example, select the **Receiver** icon for a user ID or group name that you want to add to the receiver role. The icon changes from to to show that you have added the user or group to the access role for the resource.
   e. Repeat the previous step to add more users or groups to access roles for the currently expanded destination, and then click **Next**. A summary of your access role assignments is displayed.
   f. Click **Previous** to review and change your assignments, if required.

g. Click **Finish** to confirm your assignments.

6. Repeat steps 4 and 5 for each destination you want to work with.
7. Save your changes to the master configuration.

Results

The selected users and groups are added to the access roles for the currently expanded destination. The new access role assignments are displayed in the Destination access roles panel.

Example

A group called MyGroup receives messages from three queues, Queue 1, Queue 2, and Queue 3. If you want the group MyGroup to produce and consume messages at an additional destination, Queue 4, you add MyGroup to Queue 4, and then add MyGroup to the sender and receiver roles for Queue 4.

What to do next

Use the administrative console to perform other security administrative tasks.

Removing users and groups from destination roles:

Service integration bus security uses role-based authorization. By removing users and groups from the destination roles for a secured bus, you can prevent those users and group members from performing messaging operations on the bus.

About this task

When selected users and groups no longer require access to a destination, you can remove them from all the roles for that destination.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles** A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destinations to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles at this destination, and verify that the user or group that you want to remove has a role at this destination.
5. Select the users and groups that you want to remove from all role types at this destination, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from all role types at the selected destination. The Manage access roles for users and groups panel displays the updated role type assignments.

Example

The members of three groups, Group A, Group B, and Group C, belong to the sender role and the receiver role for two queue destination, Queue 1 and Queue 2. If Group B is no longer required to send

and receive messages on Queue 2, you can use this task to remove Group B from all the role types on Queue 2.

What to do next

You can perform other security administration tasks by using the administrative console.

Restoring default inheritance for a destination:

Service integration bus security uses role-based authorization. By default, all local destinations inherit access roles from the default resource. If default inheritance has been previously overridden, you can restore it for a selected destination.

Before you begin

Default inheritance has been overridden for a selected secured destination. For more information, refer to "Overriding inheritance from the default resource for a destination."

About this task

If default inheritance has been overridden for a particular destination, you can override it. In this task, you use the administrative console to restore the role type assignments from the default resource to a selected destination. A destination can only inherit access roles that are allowed for that particular type of destination. For example, a topic space can inherit the sender and receiver roles, but it cannot inherit the browser role. Inherited access roles are added to any existing access roles for the destination.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destinations panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination to list the users and groups that have been assigned to roles for this destination.
5. Select the **Inherit from default** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The role type assignments for the default resource are inherited by the selected destination. The Destination access roles panel displays the newly inherited default access roles for the destination, and any existing access roles.

What to do next

You can perform other security administration tasks by using the administrative console.

Overriding inheritance from the default resource for a destination:

Service integration bus security uses role-based authorization. By default, local destinations can inherit access roles from the default resource. If you do not want users and groups in the default access role to access a particular destination, you can override default inheritance for a selected destination.

About this task

All the destinations in a local bus namespace can inherit default access roles with the following exceptions:

- A destination for which default inheritance is overridden.
- Foreign destinations.
- Alias destinations that have an alias bus name that is not the local bus name.

In this task, you use the administrative console to override default inheritance for a selected destination. This means that the users or groups that belong to the default access role can no longer access the selected destination.

1. Log into the administrative console
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destination panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination to list the users and groups that have been assigned to roles for this destination.
5. Clear the **Inherit from default** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The inherited role type assignments are removed from the selected destination. The Destination access roles panel displays the updated access roles for the destination.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering foreign bus roles

Service integration bus security uses role-based authorization. When messaging security is enabled, groups of users require authority to send messages from a local bus destination to a foreign bus. By listing, adding and removing users and groups in foreign bus roles, you can control who can send messages to foreign buses.

Before you begin

These tasks assumes that one or more foreign bus connections have been configured. For more information, refer to Configuring foreign bus connections.

About this task

The foreign bus connection represents another service integration bus. The bus is either in the same cell as the local bus, or in a different cell, or it represents a WebSphere® MQ queue manager. From the local bus, every other bus is regarded as a foreign bus, even if it is a bus in the same cell. Messages route to a foreign bus either directly through a link between the local bus and the foreign bus, or indirectly through one or more intermediate buses. A member of a group that belongs to the sender role on the local bus and the foreign bus can send messages directly to the foreign bus. The sender role is the only foreign bus role.

For more information about how to list, add and remove groups of users in foreign bus roles using the administrative console, refer to the following tasks.
- “Adding users and groups to foreign bus roles”
- “Removing users and groups from foreign bus roles” on page 688
- “Listing users and groups in foreign bus roles”

Listing users and groups in foreign bus roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send messages from a secured local bus destination to a secured foreign bus. By listing all the users and groups in foreign bus roles for a selected foreign bus, you can find out who has authority to send messages from the local bus to the selected foreign bus.

About this task

In this task you use the administrative console to list users and groups in the sender role for selected foreign buses. The list includes users and groups that have references in the service integration role-based configuration. It does not include all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of all the foreign buses defined for the selected bus is displayed.
3. Select one or more foreign buses:
   - Click the name of a single foreign bus.
   - Select the check boxes next to multiple foreign buses and click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each foreign bus is displayed in a collapsible section.
4. Expand the section header for a foreign bus.

Results

A list of all the users and groups that have been assigned to the sender role for the currently expanded foreign bus is displayed.

What to do next

You can add and remove users and groups in the sender role for the selected foreign bus.

Adding users and groups to foreign bus roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send messages from a secured local bus destination to a secured foreign bus. By adding selected users and groups to the sender role for a selected foreign bus, you can control who has authority to send messages to the selected foreign bus.

Before you begin

This task assumes that the following conditions have been met:
- One or more foreign bus connections have been configured for the local bus. For more information, refer to Configuring foreign bus connections.
- The users and groups that you want to add to foreign bus roles must exist in the user repository.

About this task

By default, when security is enabled, users and groups cannot send messages to a foreign bus. You need to add them to the sender role for the foreign bus. In this task you uses an administrative console wizard to select one or more foreign buses, retrieve selected users or groups from the potentially very large number of users and groups in the user repository, and add them to the sender role for the selected foreign buses.

1. Start the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of the foreign buses defined for the selected bus is displayed in the Foreign buses panel.
3. Select one or more foreign buses to work with:
   - Click a single foreign bus name.
   - Select the check boxes next to multiple foreign bus names, and then click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each foreign bus you have selected is displayed in a collapsed section.
4. Expand a foreign bus header to list the users and groups that have been assigned to roles for this foreign bus. You can verify that the user or group you want to add does not already have a role for this foreign bus.
5. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to the sender role for the selected foreign bus:
   a. Search for the users or groups that you want to add to the sender role for the expanded foreign bus:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to the sender role for the currently expanded foreign bus, and click **Next**. A list of users IDs or group names that you can add to the sender role is displayed. Note that some users or groups may already be assigned to the sender role for this foreign bus.
   d. Select the **Sender** icon for a user ID or group name that you want to add to the sender role. The icon changes from to to show that you have added the user or group to the access role for the resource.
   e. Repeat the previous step for each user or group you want to add to the sender role, and then click **Next**. A summary of your role assignments is displayed.
   f. Click **Previous** to review and change your assignments, if required.
   g. Click **Finish** to confirm your assignments.
6. Save your changes to the master configuration.

Results

The selected users and groups are added to the sender role for the selected foreign bus. The new access roles are displayed in the Foreign bus access roles panel.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from foreign bus roles:

Service integration bus security uses role-based authorization. By removing users and groups from the foreign bus roles for a selected secured local bus, you prevent those users and groups from sending messages to the foreign bus.

About this task

In this task you use the administrative console to remove selected users or groups from the sender role for a selected foreign bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of all the foreign buses defined for the selected bus is displayed.
3. Select one or more foreign buses to work with:
   - Click a single foreign bus name.
   - Select the check boxes next to multiple foreign bus names, and then click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each selected foreign bus is displayed in a collapsed section.
4. Expand the section header for a foreign bus to display details for all the users and groups that have the sender role for this foreign bus.
5. Select the users and groups that you want to remove from the sender role, and click **Remove**. The selected users and groups are removed from the sender role for this foreign bus.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from the sender role for the selected foreign bus. The Foreign bus access roles panel displays the changed access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering temporary destination prefix roles

Service integration bus security uses role-based authorization. A temporary destination prefix can have two role types: creator and sender. The messaging engine uses the temporary destination prefix at runtime to determine which users and groups have authority to create a temporary destination, and send messages to temporary destinations. By administering temporary destination prefix roles for a bus, you control which users and groups can create and send messages to temporary destinations for a selected bus.

Before you begin

Ensure that security is enabled for the bus. For more information, refer to “Securing buses” on page 663.

About this task

By default, a bus does not contain any temporary destination prefixes. You use the administrative console to add a new temporary destination prefix to the bus, and then assign selected users and groups to the sender role for the new temporary destination prefix. The creator role is assigned by default. All authenticated users can create temporary destinations by default. You can remove selected users and groups from the sender role for a selected temporary destination prefix, and you can remove a selected temporary destination prefix.

Use the following tasks to administer temporary destination prefix roles.
- "Adding users and groups to temporary destination prefix roles"
- "Removing users and groups from temporary destination prefix roles" on page 691
- "Listing users and groups in temporary destination prefix roles"
- "Removing a temporary destination prefix" on page 691

Listing users and groups in temporary destination prefix roles:

Service integration bus security uses role-based authorization. Temporary destination prefix roles are used to authorize access to the temporary destinations for a bus. By listing the users and groups in the temporary destination prefix roles for a selected bus, you can find out which users and groups can create temporary destinations, and send messages to temporary destinations for a selected bus.

About this task

In this task you use the administrative console to list users and groups in temporary destination prefix roles for the selected bus. The list includes users and groups that have references in the role-based security configuration for service integration. It does not include all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles**. The Temporary destination prefixes panel lists all the temporary destination prefixes defined on the bus.
3. Select one or more temporary destination prefixes to work with:
   - Click the name of a single temporary destination prefix.
   - Select the check boxes next to multiple temporary destination prefixes, and click **Manage Access Roles**.

   The Temporary destination prefix access roles panel is displayed. The access roles information for each temporary destination prefix is displayed in a collapsed section.
4. Expand the header for a selected temporary destination prefix to show its access roles.

Results

The expanded section lists the users, groups and group members that are assigned to access roles for the selected temporary destination prefix.

What to do next

You can now administer the users and groups in the sender role for a selected temporary destination prefix.

Adding users and groups to temporary destination prefix roles:

Service integration bus security uses role-based authorization. The messaging engine uses the temporary destination prefix at runtime to determine whether a client application is authorize to create, or send

messages to a particular temporary destination. By adding users and groups to temporary destination prefix roles for a selected bus, you can control which users and groups can create temporary destinations, and send messages to them.

Before you begin

The users and groups that you want to add to temporary destination prefix roles must already exist in the user repository.

About this task

By default, the bus security configuration does not contain any temporary destination prefixes. In this task, you use the administrative console Security wizard to first add a new temporary destination prefix, and then add users and groups to the sender role for the new temporary destination prefix. Note that the creator role is assigned by default to the creator of the temporary destination; you cannot use the administrative console to add users and groups to the creator role. By default, members of the All Authenticated group have authority in the creator role for temporary destination prefixes.

1. Log into the administrative console. The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus. By default, this list is empty.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles**
3. Click **Add** to launch the Security wizard:
   a. Define the name of the temporary destination prefix, and identify the users or groups that you want to add to the sender role for the temporary destination prefix:

      Resource
      This field is mandatory. Specify a name for the new temporary destination prefix.

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user identities or group names in the user repository. Only user identities or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wild card characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user identities or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes for the user identities or group names that you want to assign to the sender role for the temporary destination prefix, and click **Next**. Note that you cannot assign users and groups to the creator role; it is assigned by default.
   d. Select the **Sender** icon for each user identity or group name that you want to add to the sender role. The icon changes from to to show that you have added the user or group to the access role for the resource.
   e. Click **Next**. A summary of your role type assignments is displayed.
   f. Click **Previous** to review and change your role type assignments. Make your changes on the Select role types page, and then click **Next**. Note that you cannot change the name of the temporary destination prefix.
   g. Click **Finish** to confirm your assignments. The role type assignments are saved to the master configuration, and the new assignments are displayed in the Temporary destination prefixes panel.

4. Save your changes to the master configuration.

Results

The selected users, groups, and group members are added to the sender role for the selected temporary destination prefix roles. The Manage access roles panel displays the new access roles.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from temporary destination prefix roles:

Service integration bus security uses role-based authorization. When security is enabled, a temporary destination prefix role is used to authorize access to temporary destinations. The temporary destination prefix is used at runtime to create temporary destinations on the bus. By removing users and groups from temporary destination prefix roles for a selected bus, you can prevent selected users and groups from sending messages to temporary destinations on the bus.

About this task

In this task you use the administrative console to remove users, groups, and group members from the sender role for selected temporary destination prefixes. Note that you cannot use this task to remove users and groups from the creator role. If you want to remove the creator role from a user or group, refer to "Removing a temporary destination prefix."

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles** The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus.
3. Select one or more temporary destination prefixes to work with:
   - Click the name of a single temporary destination prefixes.
   - Select the check boxes next to multiple temporary destination prefixes, and click Manage Access Roles.

   The Temporary destination prefix access roles panel is displayed. The access roles information for each temporary destination prefix is displayed in a collapsed section.
4. Expand the header for a selected resource to show its role type assignments.
5. Select the users and groups that you want to remove from the sender role for the currently selected temporary destination prefix, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users, groups, and group members are removed from the sender role for the selected temporary destination prefix. The Temporary destination prefix access roles panel is updated to show the changes to the access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing a temporary destination prefix:

A temporary destination prefix is used by the service integration bus security model to determine what operations the creator of the temporary destination can perform. By removing a temporary destination prefix, you remove the creator role from the identity of the user that created the temporary destination.

About this task

A temporary destination prefix can have two authorization role types: creator and sender. In this task you use the administrative console to remove a selected temporary destination prefix from a selected bus, which removes the creator role from the user identity that created the temporary destination.

If you want to remove users and groups from the sender role for a temporary destination prefix, refer to "Removing users and groups from temporary destination prefix roles" on page 691.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles** The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus.
3. Select the temporary destination prefix you want to remove.
4. Click **Remove**.
5. Save your changes to the master configuration.

Results

The selected temporary destination prefix is removed, and the creator role is removed from the user identity that created the temporary destination.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering topic space root roles

Service integration bus security uses role-based authorization. When messaging security is enabled, groups of users require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By adding and removing users and groups in topic space root roles, you can control access to the topic space root.

About this task

Topic space root (/) is also called the virtual root, and it is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space, and it is a type of destination. Note that these tasks apply only to the topic space root; they do not apply to topics or a topic space. For information about administering topic access roles, refer to "Administering topic roles" on page 695, and for information about administering topic space access roles, refer to "Administering destination roles" on page 680.

You can add and remove users and groups in the sender and receiver roles for the topic space root. The topic space root can also inherit access in the sender and receiver roles from the topic space, providing the topic space is configured to inherit the default destination roles. For more information about topic inheritance, refer to Topic security.

For the topic space root roles to have an effect, the **Topic Access Check Required** checkbox must be set in the topic space configuration. For more information, refer to Configuring bus destination properties.

Use the following tasks to list, add and remove users and groups in the topic space root roles using the administrative console.

- "Listing users and groups in topic space root roles" on page 693
- "Adding users and groups to topic space root roles" on page 693

- “Removing users and groups from topic space root roles” on page 695

Listing users and groups in topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By listing the users and groups in topic space root roles, you can find out who has access to the topic space root for a selected topic space.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. In this task you use the administrative console wizard to list users and groups in topic space root roles for a selected root topic.

This task applies to the topic space root only; it does not apply to the topics within the topic space, or to the topic space. If you want to list users and groups in topic roles, refer to “Listing users and groups in topic roles” on page 696. If you want to list users and groups in a topic space (which is a type of destination), refer to “Listing users and groups in destination roles” on page 681.

The users and groups listed in this task have references in the service integration bus security configuration. The list does not include all users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**

Results

The Topic space root panel lists all the users and groups in topic space root roles for the selected bus.

What to do next

You can administer the role type assignments for the users and groups displayed.

Adding users and groups to topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By adding users and groups to topic space root roles, you control access to the root topic in a selected topic space.

Before you begin

- The users and groups you want to add to topic space root roles must exist in the user repository.
- Topic space root roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to Configuring bus destination properties.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. Note that this task applies only to the topic space root; it does not apply to adding users and groups to topics or a topic space. For information about adding users and groups to topic access roles, refer to “Adding users and groups to topic roles” on page 696, and for adding users and groups to topic space access roles, refer to “Adding users and groups to destination roles” on page 681.

You can add users and groups to the sender and receiver roles for the topic space root. The topic space root can also inherit access in the sender and receiver roles from the topic space, providing the topic space is configured to inherit the default destination roles. For more information about topic inheritance, refer to Topic security.

By default, a topic space does not contain a root topic. In this task you use an administrative console wizard to add a root topic to an existing topic space, retrieve the users and groups from the user repository that you want to assign to roles on the new root topic, and add them to the root topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**. The Topic spaces panel lists the topic spaces defined on the selected bus.
3. Select the name of the topic space where you want to add a new root topic. The Topics panel displays the selected topic space in a collapsible section.
4. Click **Add** to launch the Security wizard:
   a. Identify the users or groups that you want to add to the sender and receiver roles for the new root topic:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. You can use wildcard characters in the search string.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names that you want the administrative console to display.

   b. Click **Next**. The wizard displays the new root topic, and lists the users IDs or group names in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to assign to roles on the new root topic.
   d. Click **Next**. The wizard displays the topic role types that you can assign for the users or groups you selected in the previous step. Role types might already have been assigned for a specific user or group.
   e. Select the role types for the selected users or groups. For example, to assign a user to the sender role, select the **Sender** icon for the appropriate user ID. The icon changes from to to show that you have added the user or group to the access role for the resource.
   f. Click **Next**. A summary of your role type assignments for the root topic is displayed.
   g. If you want to change your assignments, click **Previous** to return to the Select role types page, change your assignments, and then click **Next**.
   h. Click **Finish** to confirm your assignments. The role type assignments are saved to the master configuration, and the new assignments are displayed in the Topics panel.
5. Save your changes to the master configuration.

Results

The selected users and groups are added to topic space root roles for the new root topic. The **Manage access roles** panel displays the new access role assignments.

What to do next

You can perform additional security administration using the administrative console.

Removing users and groups from topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By removing users and groups from topic space root roles, you prevent them from accessing the root topic in a selected topic space.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. Note that this task applies only to the topic space root; it does not apply to removing users and groups from topics or a topic space. For information about removing users and groups from topic access roles, refer to “Removing users and groups from topic roles” on page 697, and for removing users and groups from topic space roles, refer to “Removing users and groups from destination roles” on page 683.

In this task you use the administrative console to remove selected users and groups from the sender and receiver roles for the selected root topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**. The Topic spaces panel lists the topic spaces defined on the bus.
3. Select the topic space you want to work with. The selected topic space is displayed in the Topics panel. The root topic (/) is displayed by default.
4. Select the topic space root. The Topic access roles panel lists the role type assignments for the topic space root.
5. Select the names of the users, groups and group members that you want to remove from all role types for the selected root topic, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from all roles for the selected root topic. The Topic access roles panel is updated to show the changes to the access roles assignments.

What to do next

You can perform other security administration commands using the administrative console.

Administering topic roles

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to access a topic in a publish/subscribe topic hierarchy. By adding and removing users and groups in topic roles, you can control access to the topic.

About this task

You use the administrative console to list, add and remove users and groups in the sender and receiver roles, and to define topic role inheritance. By default, a child topic inherits its topic roles from its parent topic. You can change the default roles for a particular topic by adding or removing topic roles at the topic level. You can also allow or block inheritance of topic roles at topic level.

You can add access roles for a topic before it exists. Topics are created at runtime only, and exist only for as long as they are active.

For information about how to list, add, and remove users and groups in topic roles using the administrative console, refer to the following tasks.
- "Listing users and groups in topic roles"
- "Adding users and groups to topic roles"
- "Removing users and groups from topic roles" on page 697
- "Enabling topic role inheritance" on page 698
- "Disabling topic role inheritance" on page 699

Listing users and groups in topic roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to access topics in a publish/subscribe topic hierarchy. By listing the users and groups that are members of topic roles for a selected topic, you can find out who has authority to send messages to and from the topic.

About this task

In this task you use the administrative console to list users and groups that have access roles for a selected topic in a selected topic space. The list includes users, groups and group members that have references in the role-based security configuration for service integration. It does not list all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topics panel displays the information for the topic in a collapsed section.
3. Expand the section header to display the users and groups that are assigned to role types for the selected topic.

What to do next

You can add and remove users and groups in the topic roles for the selected topic.

Adding users and groups to topic roles:

Service integration bus security uses role-based authorization. When messaging security, and topic level authorization is enabled, users and groups must be authorized to access topics in a publish/subscribe topic hierarchy. By adding users and groups to topic roles, you control access to a topic in a selected topic space.

Before you begin

- The users and groups you want to add to topic space root roles must exist in the user repository.
- Topic roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to Configuring bus destination properties.

About this task

Topics are organized into one or more hierarchies within a topic space. If the **Topic Access Check Required** setting is enabled for the topic space, a user must have authorization to access the topic itself. You can add access roles to a topic before it is created at runtime. A topic inherits access roles from its parent unless you explicitly block the inheritance. For more information, refer to "Enabling topic role inheritance" on page 698.

In this task you use an administrative console wizard to add users or groups to the sender and receiver roles for a selected topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic space root panel lists the users and groups that are assigned to role types for the selected topic.
3. Click **Add** to launch the Security wizard:
   a. Provide the following information to enable the wizard to identify the users or groups that you want to add to role types for the selected topic:

      Resource
      Specify the name of the topic.

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wild card characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard lists the users IDs or group names that match the information that you provided in the previous step.
   c. Select the check boxes for the user IDs or group names that you want to assign to roles for the selected topic.
   d. Click **Next**. The wizard lists the topic role types that you can assign for the users or groups you selected in the previous step. Role types may already have been assigned for a specific user or group.
   e. Select the role types for each of the selected users or groups. For example, to assign a user ID to the sender role, select the **Sender** icon for that user ID. The icon changes from to to show that you have added the user or group to the access role for the resource.
   f. Click **Next**. A summary of your role type assignments for the selected topic is displayed.
   g. If you want to change your assignments, click **Previous** to return to the Select role types step. Make changes to your assignments, and click **Next** to return to the Confirm step.
   h. Click **Finish** to confirm your assignments and save your changes to the master configuration.

Results

The updated role type assignments for the selected users or groups are displayed in the Topic access roles panel.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from topic roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, and the **Topic access check required** setting is enabled for the topic space, users and groups require

authority to access a topic in the topic space. By removing users and groups from all topic roles for a selected topic, you prevent them from accessing the topic.

Before you begin

Topic roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to Configuring bus destination properties.

About this task

This task uses the administrative console to remove users and groups from both the sender and receiver roles for a selected topic in a selected topic space.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel is displayed. The information for the topic is displayed in a collapsed section.
3. Expand the section header to display the users and groups that are assigned to role types for the selected topic.
4. Select the users and groups that you want to remove from the sender and receiver roles for the selected topic, and click **Remove**.
5. Save your changes to the master configuration.

Results

The selected users and groups are removed from the sender and receiver roles for the selected topic. The Topic access roles panel is updated to show that the selected users and groups have no topic role type assignments.

Enabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, and users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If topic role inheritance has been disabled for a particular topic, you can restore it by using the administrative console.

Before you begin

You must ensure that the following conditions are met:

- Messaging security is enabled. For more information, refer to “Disabling bus security” on page 664.
- Topic level security is enabled for the topic space. Check the setting **Topic Access Check Required?** in the topic space destination configuration. For more information, refer to Configuring bus destination properties.

About this task

In this task you use the administrative console to restore topic role inheritance for selected topics. A topic can only inherit the sender and receiver roles from the parent topic in the topic hierarchy.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.
3. Expand the topic name header to display details of the users and groups that have one or more access roles for this topic.

4. Select the **Inherit sender role from parent topic** check box.
5. Select the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The select topic inherits access roles from the parent topic. The Topic access roles panel displays the inherited access roles for the topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Disabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If you do not want topics to inherit topic roles from the parent topic in the topic hierarchy, you can override topic role inheritance by using the administrative console.

Before you begin

About this task

In this task you use the administrative console to prevent a selected topic from inheriting authorization roles from its parent topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.
3. Expand the topic name header to display details of the users and groups that have access one or more access roles for the selected topic.
4. Clear the **Inherit sender role from parent topic** check box.
5. Clear the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The selected topic cannot inherit access roles from its parent topic. The Topic access roles panel displays the changed access roles for the selected topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing access roles from unknown users and groups

Service integration bus security uses role-based authorization. Users and groups are assigned to access roles for specific bus resources. If a user or a group that has access roles is removed from the user repository, it becomes an unknown user. You can identify the unknown users and groups for a selected bus, and removes their access roles.

About this task

In this task, you use the administrative console to remove access roles from users and groups for unknown users and groups.

1. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage users and groups not known to the user repository** The Unknown users and groups panel lists all the users and groups that have access roles assigned to them, but they are not known in the user registry.
2. Select the check boxes next to multiple user and group names.
3. Click **Remove all roles**. The access role assignments for the selected users and groups are removed.
4. Save your changes to the master configuration.

Results

The access role assignments are removed from the selected users and groups. The Unknown users and groups panel is updated to show the changes to the role type assignments.

Administering permitted transports for a bus

Use these tasks to configure a transport policy for a service integration bus, and to administer the transports chains that remote applications clients can use to connect to a service integration bus.

About this task

- "Administering permitted transports for a bus"
- "Listing permitted transports for a bus" on page 701
- "Adding a permitted transport to a bus" on page 701
- "Removing a permitted transport from a bus" on page 702

Configuring a transport policy for a bus

By configuring a transport policy for the bus you can affect the security of messages in transit.

Before you begin

There are no prerequisites for this task.

About this task

The transport policy for a bus controls which transport mechanism remote application clients can use to connect to the bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value***. The configuration panel for the selected bus is displayed
3. Choose one of the following transport policies for the bus:

   Allow the use of all defined transport channel chains
   Select this option to allow the bus to use unsecured ports.

   Restrict the use of defined transport channel chains to those protected by SSL
   Select this option to prevent the use of the InboundBasicMessaging port.

   Restrict the use of defined transport channel chains to the list of permitted transports
   Select this option if you want connecting client applications to use named transport channel chains. This provides the highest level of control over the use of transport channel chains.
4. Click **Apply**.
5. Save your changes to the master configuration.

Results

The transport policy you have configured for the selected bus controls how application clients connect to the bus.

What to do next

You can use the administrative console to add and remove transport chains in the list of permitted transports for the bus.

Listing permitted transports for a bus

Use this task to display a list of the transport chains that are available for a remote client application to use to connect to a selected service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 700.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. In this task, you use the administrative console to list all the permitted transports for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**.

Results

The Permitted transports panel displays the list of permitted transports for the selected bus.

What to do next

You can use the administrative console to add and remove transport chains in the list of permitted transports to control which transport chains a remote client application can use to connect to the bus.

Adding a permitted transport to a bus

Use this task to make a new transport chain available to a remote client application to use to connect to a service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 700.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. If you want to allow remote client applications to connect to the bus using a new transport chain, you must add the new transport chain to the list of permitted transports for the bus. In this task, you use the administrative console to add a new transport chain to the list of permitted transports for a selected bus.

1. Log onto the administrative console.

2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**. The Permitted transports panel displays the list of permitted transports for the selected bus.
3. Click **New**.
4. Select the name of the transport chain you want to add in the list box.
5. Click **OK**.
6. Save your changes to the master configuration.

Results

The new transport name is displayed in the list of permitted transports, and its is available for use by a remote client application to connect to the bus.

What to do next

You can use the administrative console to remove transport chains from a bus.

Removing a permitted transport from a bus

Use this task to remove a selected transport chain from the list of permitted transport chains for a selected service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 700.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. If you want to prevent a remote client application from using a particular transport chain to connect to the bus, you can remove the transport chain from the list of permitted transports for the bus. In this task, you use the administrative console to remove a transport chain from the list of permitted transports for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**. The Permitted transports panel displays the list of permitted transports for the selected bus.
3. Select the name of the transport chain you want to remove.
4. Click **Delete**. The Permitted transports panel displays an updated list of permitted transports for the selected bus.
5. Save your changes to the master configuration.

Results

The selected transport chain is removed from the list of permitted transports for the selected bus, and a remote client can no longer use it to connect to the bus.

What to do next

You can use the administrative console to manage the transport policy for the bus.

Securing messages between messaging buses

Use these tasks to administer the access control security associated with sending messages between buses.

About this task

- “Protecting messages transmitted between buses”
- “Administering access to foreign destinations”

Protecting messages transmitted between buses

Use this task to protect the integrity of the data that is transmitted between secured linked service integration buses.

Before you begin

- Review the information in the topics Secure transport considerations and Configuring transport chains.
- Configure a transport policy for the bus, as described in the topic “Configuring a transport policy for a bus” on page 700.

About this task

In this task, you configure transport chains for the remote bus to ensure that only secured inbound transport chains can contact the messaging engines on the server. The remote bus may be a foreign bus, or an WebSphere MQ link.

1. Ensure that the linked buses are secured. For further information, see “Securing buses” on page 663.
2. Configure the Target inbound transport chain property for the foreign bus, or WebSphere MQ link, as appropriate:
   - For connections to a foreign bus, refer to Connecting service integration buses to use point-to-point messaging or Connecting service integration buses to use publish-subscribe messaging
   - For connections to WebSphere MQ, refer to Creating a new WebSphere MQ link.

Results

You have configured the Target inbound transport chains for the remote service integration bus.

What to do next

Administering access to foreign destinations

Use these tasks to administer the access control security associated with sending messages to foreign bus destinations.

About this task

To define the authentication information used in the access control checks that are performed when a message is sent to a destination in a foreign bus, complete the following steps:

- Define the required authorization permissions to allow a sender to access a destination on a foreign bus. These can be defined using either a foreign bus definition, as described in “Administering foreign bus roles” on page 685, or a foreign bus destination definition, as described in “Administering destination roles” on page 680. These permissions are used when the message is sent, to check that the sender is allowed to access the foreign bus.
- The administrator of the foreign bus must define the required authorization permissions to allow the messages to access the destination on the foreign bus. These permissions are used when the message enters the foreign bus.
- Define the authorization permissions to allow any messages entering your bus from the foreign bus to access the required destinations.

Securing access to a foreign bus

You can secure the link between a local bus and a foreign bus.

Before you begin

Before you can secure the link between a local bus and a foreign bus, there must be foreign bus connection on the local bus, and therefore a link between the buses.

About this task

This task summarizes the significant tasks to secure the link between a local bus and a foreign bus. For more general information about service integration bus security, see “Security” on page 663.

When you create a foreign bus connection, there are some options to secure the connection during that procedure.

1. Enable security on the service integration bus and the foreign bus. See “Disabling bus security” on page 664.
2. Secure the link between the buses. See “Securing messages between messaging buses” on page 703.
3. Grant access to the local bus for users who will be sending messages to the foreign bus. See “Administering the bus connector role” on page 675.
4. Grant access to the foreign bus for users who will be sending messages to it. See “Administering foreign bus roles” on page 685.
5. Optional: Give users access to foreign or alias destinations that will forward messages to a foreign bus. See “Administering destination roles” on page 680.

Securing links between messaging engines

When security is enabled, messaging engines in a mixed version bus establish trust using an authentication alias.

Before you begin

You must ensure that the user ID that you intend to use for the authentication alias meets the following conditions:

- It exists in the user registry
- It is used only for the purpose of messaging engine to messaging engine authentication.
- It has not been added to the bus connector access role.

About this task

If you have a secure, mixed version bus, you must define an Inter-engine authentication alias for use in preventing unauthorized messaging engines from establishing a connection. The Inter-engine authentication alias is used by messaging engines to establish trust in the following scenarios:

- A WebSphere Application Server Version 6.x messaging engine initiates a link with a Version 7.0 messaging engine.
- A Version 7.0 messaging engine initiates a link with a Version 6.x messaging engine.

Trust between Version 7.0 messaging engines is established using a Lightweight Third Party Authentication (LTPA) token.

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The bus security configuration panel is displayed.
2. In the **Inter-engine authentication alias** field, select an authentication alias.
3. Click **OK**.

4. Save your changes to the master configuration.

Results

You have selected an Inter engine authentication alias for the bus to use in establishing trust between mixed version messaging engines.

What to do next

If you require additional security, you can configure the SSL certificate stores to restrict objects that can make an SSL connection, and thereby connect to the bus. For more information see Configuring secure sockets layer.

Controlling which foreign buses can link to your bus

Use this task to control which foreign buses are allowed to link to your bus.

About this task

When messaging security is enabled, it is important that only authorized foreign buses are allowed to link to your bus.

To control which foreign buses can create a link to your bus, complete the following steps:

1. Set the authentication alias property on the foreign bus connection to be used for authentication of the foreign bus joining your bus, as described in Connecting service integration buses to use point-to-point messaging or Connecting service integration buses to use publish-subscribe messaging.
2. If you require extra security, configure the SSL certificate stores to restrict who can make an SSL connection, and hence link to the bus. For more information, see Configuring secure sockets layer.

Securing database access

How to protect the data store from access by unauthorized users.

About this task

To secure access to the data store, you must configure a user name and password on the data store for the messaging engine. If you want to apply additional levels of security such as encrypting the contents of the database, use the specific security features provided by your database.

Securing mediations

Use the following tasks to secure mediations at an operations level. For example, a mediation inherits its identity from a the messaging engine, but you might want to specify an alternative identity for the mediation to use.

Configuring an alternative mediation identity for a mediation handler

Use this task to configure an alternative mediation identity for a mediation handler

About this task

By default, a mediation inherits the identity used by the messaging engine. In some cases, you might want to specify an alternative identity for a mediation handler to use. For example, for a single mediation that sends messages to a destination. To do this, you specify a ″run-as″ identity for the mediation handler at deployment, and map the mediation handler to an identity other than the default mediation identity using a role name. Follow the steps below to specify an alternative mediation identity:

1. Package your mediation handler as an EAR file.

2. Edit the deployment descriptor file to define the roles. For more information, see Securing enterprise bean applications using the assembly tools.
3. Assign users to the role. For more information, see Mapping users to RunAs roles using an assembly tool and Assembling secured applications.
4. Deploy the mediation handler in WebSphere Application Server, and assign users to the RunAs role. For more information, see Assigning users to RunAs roles. You can confirm the mappings of users to roles, add new users and groups, and modify existing information during this step. For more information, see Deploying secured applications.

Example

What to do next

Next, you are ready to authorize mediations to access destinations. For more information, see "Administering authorization permissions" on page 674.

Configuring the bus to access secured mediations

Use this task to ensure that the service integration bus is authorized to access secured mediations.

Before you begin

The mediation is secured using a Java Platform, Enterprise Edition (Java EE) Connector Architecture authentication alias. For information about creating a Java EE authentication alias, refer to Managing J2EE Connector Architecture authentication data entries.

About this task

To configure the bus to access a secured mediation, you must add the mediation authentication alias for the secured mediation to the properties for the bus:

- If the bus has a Version 6.x bus member, you must provide the principal and its associated password.
- If the bus has WebSphere Application Server Version 7.0 bus members only, you need only provide the principal.

1. Log into the navigation pane.
2. Click **Service integration** → **Buses** → ***security_value***. The bus security configuration panel is displayed.
3. In the **Mediations authentication alias** field, select the principal for the mediation, and its associated password if required.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

The selected bus is configured to access secured mediations.

What to do next

You can assign security roles to your mediation handlers to protect them from use by unauthorized users. For more information, see Deploying secured applications.

Configuring a bus to run mediations in a multiple security domain environment

Use this task to configure a secured bus so that it can run mediations successfully on bus members in different security domains.

Before you begin

The secured bus must be configured to use a non-global security domain. For more information about securing buses using multiple security domains, refer to “Securing buses” on page 663.

About this task

If your bus topology has bus members in different security domains, you can configure the bus to allow mediations to run under the server identity. This means that a mediation can run on any server in any domain. You do not have to add a dedicated user ID for each mediation to the user repository, or maintain a mediation authentication alias.

Use the administrative console to configure a secured bus to run mediations successfully as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The security settings for the selected bus are displayed.
2. Check the option **Use the Server ID when running mediations**.
3. Click **Apply**.
4. Save your changes to the master configuration.

Results

You have configured the bus to run mediations successfully across servers in multiple security domains.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Auditing the service integration security infrastructure

Security auditing is an important part of the security infrastructure. Security auditing provides a mechanism for auditable events to be tracked and archived while ensuring the integrity of the records.

Before you begin

Before enabling the security auditing subsystem for service integration, you must enable global security in your environment.

About this task

The primary responsibility of the security infrastructure is to prevent unauthorized usage of resources. Security auditing has two primary goals:

- Confirming the effectiveness and integrity of the existing security configuration.
- Identifying areas where improvement to the security configuration might be needed.

Security auditing achieves these goals by providing the infrastructure that allows you to implement your code to capture and store supported security auditable events. All code other than the Java enterprise application code is considered to be trusted. Each time an enterprise application accesses a resource, any internal application server process that has audit points that are added within their code can be recorded as an auditable event. The security auditing subsystem captures the following types of auditable events:

- Authentication
- Authorization
- Principal and Credential Mapping
- Key management

- System management
- Security policy management
- Audit policy management
- Administrative configuration management
- Administrative runtime management
- User registry and identity management
- Password changes
- Delegation

These types of events can be recorded into audit log files. The audit log flies can be signed and encrypted to ensure data integrity. These audit log files can be analyzed to discover breaches over the existing security mechanisms and to discover potential weaknesses in the current security infrastructure. Security event audit records are also useful for providing evidence, accountability, and vulnerability analysis. The security auditing configuration provides four default filters, a default audit service provider, and a default event factory. This following steps describe how to customize your security auditing subsystem. Additional information specific to messaging is included in the step description where appropriate.

1. Enabling the security auditing subsystem

   Security auditing is not performed unless the audit security subsystem has been enabled. Global security must be enabled for the security audit subsystem to function, as no security events occur if global security is not also enabled.

   To allow messaging security events to be audited, audit security must be enabled:

   a. For each bus to be audited, click **Service integration** → **Buses** → ***security_value***, and select the **Enable the auditing service for this bus** check box.

   b. For publish/subscribe messaging, also on each topic space on the bus being audited, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***, and select the **Enable the auditing service for this topic space** check box.

2. Auditor role

   A user with the auditor role is required to enable and configure the security auditing subsystem. It is important to require strict access control for security policy management. The auditor role provides granularity to support separation of the auditing role from the authority of the administrator.

3. Creating security auditing event type filters

   You can configure event type filters to only record a specific subset of auditable event types in your audit logs. Filtering the event types that are recorded makes for simpler analysis of your audit records by ensuring the records to only display what you selected as important for your an environment.

   The audit events that can be configured for messaging are:

   SECURITY_AUTHN
   This event is produced when the identity of a messaging client or messaging engine connecting to a messaging bus is authenticated.

   SECURITY_AUTHZ
   This event is produced when the identity of a messaging client is checked for access authority to a bus or a message queue when sending, directly or by publication, or receiving messages, directly or by subscription.

   SECURITY_AUTHN_TERMINATE
   This event is produced when the connection between a messaging client or messaging engine and a messaging bus is terminated.

   SECURITY_MGMT_CONFIG
   This event is produced when a messaging client changes the contents of a service data object (SDO) repository in an import or remove operation.

You can create event filters for each permutation of an event and its possible outcomes such as SUCCESS, DENIED, or error conditions of different levels of severity.

See messaging security events for more information on which messaging security audit events are produced and when they are produced.

4. Configuring audit service providers for security auditing

   The audit service provider is used to format the audit data object that is passed to it before outputting the data to a repository.

5. Configuring audit event factories for security auditing

   The audit event factory gathers the data associated with the auditable events and creates an audit data object. The audit data object is then sent to the audit service provider to be formatted and recorded to the repository.

6. Protecting your security audit data It is important for the recorded audit data to be both secured and with the data integrity ensured. To ensure that access to the data is restricted and secure, you can encrypt and sign your audit data.

7. Configuring security audit subsystem failure notifications

   You can enable notifications to generate alerts when the security auditing subsystem experiences a failure. Notifications can be configured to record an alert in the audit logs or can be configured to send an alert through e-mail to a specified list of recipients.

Results

After successfully completing this task, you audit data is recorded for the selected auditable events that were specified in the configuration.

What to do next

After configuring security auditing, you can analyze your audit data for potential weaknesses in the current security infrastructure and to discover security breaches that might have occurred over the existing security mechanisms.

Securing bus-enabled Web services

Service integration technologies provides a range of facilities for secure communication between the service requester and the service integration bus, and between the bus and the target service.

About this task

By default, bus-enabled Web services are available when WebSphere Application Server security is enabled and your service integration buses are secured. However this level of security does not impose any restrictions on the users of individual Web services. To control how your Web services are used by each group of your colleagues or customers, you can further configure your Web services to work with password-protected components and servers, with WS-Security and with HTTPS. To do this, see the following topics:

- “Configuring bus-enabled Web services to use an authentication alias to access a secure service integration bus” on page 710.
- “Configuring secure transmission of SOAP messages using WS-Security” on page 712.
- “Working with password-protected components” on page 713.
- “Invoking outbound services over HTTPS” on page 720.

Configuring bus-enabled Web services to use an authentication alias to access a secure service integration bus

When you install WebSphere Application Server, security is enabled and every installed service integration bus is secured. By default, the bus-enabled Web services configuration works when security is enabled. However you can override the default method of allowing bus-enabled Web services to access a secure bus, by configuring an authentication alias that the service integration technologies resource adapter uses to access the bus.

About this task

The default configuration that bus-enabled Web services use to access a secure bus is as follows:

- Access to a bus is configured through the *bus connector role*. By default, every bus connector role includes a group called *server*. Members of this group are authorized to connect to the bus.
- The service integration technologies resource adapter uses a J2C activation specification to communicate with the bus. By default, this activation specification has a Boolean custom property *useServerSubject* that is set to "true". This property allows the service integration technologies resource adapter to connect to the bus as a subject (a member) of the server group.

For more information, see "Bus-enabled Web services default configuration for accessing a secure service integration bus" on page 711.

You can override this default method by configuring an authentication alias that the service integration technologies resource adapter uses to access the bus. Using an authentication alias does not make your configuration more secure. However, you might want to use an alias for consistency of approach if you have other application servers running under WebSphere Application Server Version 6 or later, or to support your internal business controls for use of IDs and passwords.

To configure an authentication alias for the resource adapter to use when it communicates with the bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value*** → **[Related Items] JAAS - J2C authentication data**.
2. Create a J2C authentication alias.
3. Configure authentication for the resource adapter by completing the following steps:
   a. In the administrative console navigation pane, click **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification_name***, where ***activation_specification_name*** is **SIBWS_OUTBOUND_MDB**.
   b. In the **Authentication alias** selection list, choose the authentication alias that you created.
   c. Click **Apply**.
4. Optional: Disable the default authentication configuration.

   If you configure an authentication alias you need not also disable the default configuration. If an authentication alias exists, it overrides the default configuration. This means that if you use an authentication alias that is authorized to access the bus then the communication will succeed, and if you use an authentication alias that is not authorized to access the bus then the communication will fail, irrespective of the default settings. However if you subsequently remove the authentication alias from the activation specification, the default configuration will again take control and (if not disabled) will allow the service integration technologies resource adapter to continue to access the bus. For more information, see "Bus-enabled Web services default configuration for accessing a secure service integration bus" on page 711.

   To disable the default authentication configuration, complete the following steps:

a. In the administrative console navigation pane, click **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification_name*** → → **[Additional Properties] J2C activation specification custom properties**, where ***activation_specification_name*** is **SIBWS_OUTBOUND_MDB**.
b. In the list of custom properties, click `useServerSubject`
c. Change the **Value** for the `useServerSubject` property from “true” to “false”.
d. Click **OK**.

5. Save your changes to the master configuration.
6. Close the administrative console.

Bus-enabled Web services default configuration for accessing a secure service integration bus

By default, service integration bus-enabled Web services works when WebSphere Application Server security is enabled and every installed bus is secured. This topic describes the default configuration, and also the effect of modifying or overriding these defaults.

The default configuration that bus-enabled Web services use to access a secure bus is as follows:

- Access to a bus is configured through the *bus connector role*. By default, every bus connector role includes a group called *server*. Members of this group are authorized to connect to the bus.
- The service integration technologies resource adapter uses a J2C activation specification to communicate with the bus. By default, this activation specification has a Boolean custom property *useServerSubject* that is set to “true”. This property allows the service integration technologies resource adapter to connect to the bus as a subject (a member) of the server group.

The server group in the bus connector role

This group controls whether a user is authorized to connect to the bus. The server group can be added or removed by using the administrative console:

Service integration → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**

This group can also be set using the following AdminTask scripting commands from the wsadmin prompt:

```
addGroupToBusConnectorRole
removeGroupFromBusConnectorRole
```

The useServerSubject property

This boolean property is found in the custom properties panel of the J2C activation specification associated with the inbound, outbound or gateway service:

Resources → **Resource Adapters** → **J2C activation specifications** → ***activation_specification_name*** → → **[Additional Properties] J2C activation specification custom properties**

This property can also be set using scripting commands.

Disabling and overriding the default configuration

To disable the default configuration, set the useServerSubject property to “false” rather than removing the server group, because the service integration technologies resource adapter is not the only system resource that uses the server subject. If you remove the server group from the bus connector role, then no system resources can use the server subject.

You can also override the default configuration by configuring bus-enabled Web services to use an authentication alias to access a secure service integration bus.Using an authentication alias does not make your configuration more secure. However, you might want to use an alias for consistency of approach if you have other application servers running under WebSphere Application Server Version 6 or later, or to support your internal business controls for use of IDs and passwords.

If you configure an authentication alias you need not also disable the default configuration. If an authentication alias exists, it overrides the default configuration. However if you subsequently remove the authentication alias from the activation specification, the default configuration will again take control and (if not disabled) will allow the service integration technologies resource adapter to continue to access the bus.

The following chart shows whether the service integration technologies resource adapter can connect to the secured bus, depending on the state of the different properties:

Table 70. Summary of expected behavior for accessing a secure service integration bus

Valid authentication alias	useServerSubject	Server group on bus connector role	resource adapter can connect?
Yes	No	No	Yes
No	Yes	Yes	Yes
No	No	Yes	No
No	No	No	No
No	Yes	No	No
Yes	Yes	Yes	Yes (using the authentication alias)

Configuring secure transmission of SOAP messages using WS-Security

Configure service integration technologies for secure transmission of SOAP messages using tokens, keys, signatures and encryption in accordance with the Web Services Security (WS-Security) specification.

Before you begin

You can configure the service integration bus for secure transmission of SOAP messages using tokens, keys, signatures and encryption in accordance with the Web Services Security (WS-Security) 1.0 specification.

Alternatively, you can configure the bus in accordance with the previous WS-Security specification, WS-Security Draft 13 (also known as the Web Services Security Core Specification).

Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

You can only use WS-Security with Web service applications that comply with the *Web services for Java Platform, Enterprise Edition (Java EE)* or *Java Specification Requirements (JSR) 109* specification. For more information, see Web services security and Java Platform, Enterprise Edition security relationship. For information about how to make your Web service applications JSR-109 compliant, see Developing and deploying JAX-RPC Web services clients or Developing and deploying JAX-WS Web services clients .

About this task

To protect a service integration bus-deployed Web service, you can apply the following types of WS-Security resource to the inbound or outbound ports that the service uses:
- WS-Security bindings.
- WS-Security configurations.

The *configurations* resource type specifies the level of security that you require (for example “The body must be signed”), and the *bindings* resource type provides the information that the run-time environment needs to implement the configuration (for example “To sign the body, use this key”),

When you associate a WS-Security resource with a port, you choose from a list of WS-Security resources that you have previously configured as described in the following topics:
- Creating a new WS-Security binding.
- Creating a new WS-Security configuration.

What to do next

Note: You can associate any binding with any configuration, so you must ensure that you choose a valid combination.

For an overview of how WS-Security is applied to service integration bus-deployed Web services, see Service integration technologies and WS-Security. For detailed information about how WS-Security is implemented in WebSphere Application Server, see Securing Web services applications at the message level (WS-Security). For more information about the WS-Security standard, see the Web Services Security (WS-Security) 1.0 specification.

Getting WS-Security information from the owning parties

About this task

You get, from the owning parties, the WS-Security configurations for the client (in the case of an inbound service) and the target Web service (in the case of an outbound service). This information is found in the following files on the owners systems:
- Key stores (`.ks`, `.jks` and `.jceks` files).
- Certificate stores (`.cer` files).
- Security settings (the `ibm-webservicesclient-ext.xmi` file for the client, and the `ibm-webservices-ext.xmi` file for the Web service).
- Binding information - for example the location of a keystore file on the file system (the `ibm-webservicesclient-bnd.xmi` file for the client, and the `ibm-webservices-bnd.xmi` file for the Web service).

If the client is hosted on WebSphere Application Server, and the Web service security settings are created using IBM Web services tooling (for example IBM Rational Application Developer), then the files that contain the security settings and binding information have the exact file names (`*.xmi`) noted previously. For clients and Web services from other vendors, these files have different file names.

You need to copy the key store and certificate store files to the WebSphere Application Server file system, and to enter and configure (as WS-Security bindings and configurations) the security settings that are contained in the `.xmi` files. There are tools available (for example IBM Rational Application Developer) that can parse the `.xmi` files for you.

Working with password-protected components

Configure user ID and password authentication and authorization for inbound services, and for individual operations within a Web service. Invoke password-protected outbound services, and access password-protected proxy servers.

About this task

In addition to the security options described in Configuring secure transmission of SOAP messages using WS-Security, you can also use the broader security features of WebSphere Application Server to work with the password-protected components that are described in the sub-topics listed below:

- "Password-protecting inbound services."
- "Password-protecting a Web service operation" on page 715.
- "Invoking a password-protected outbound service" on page 718.
- "Accessing a password-protected proxy server" on page 718.

Password-protecting inbound services

Password-protect a set of inbound services by requiring user authentication for access to the associated HTTP endpoint listener, or (for JMS) to the associated JMS queue destination.

Before you begin

This topic covers the two main areas in which you might want to change the HTTP endpoint listener authentication settings:

- Changing the HTTP endpoint listener security role.
- Mapping the HTTP endpoint listener security role to users or groups.

If you want to change the HTTP endpoint listener security role, do so before you create the HTTP endpoint listener configuration.

For a SOAP over JMS endpoint listener, you can achieve similar results by securing the underlying destination for each JMS queue.

About this task

When WebSphere Application Server administrative security is enabled, clients that access an HTTP endpoint listener can be prompted for a user ID and password, which are authenticated against the registry defined within the security configuration. The HTTP endpoint listeners that are supplied with WebSphere Application Server are configured with a security role named `AuthenticatedUsers`. By default this role is mapped to the special group **Everyone**, so even if security is enabled all users can access any inbound service deployed to the HTTP endpoint listener.

You need not change the default security role. You would only choose to do so if you wanted to use a role name that is more specific, or more meaningful in the context of your organization. To change the security role, you modify the endpoint listener application EAR file before you configure the endpoint listener.

After you configure the endpoint listener application, you can map the security role to specific users or groups so that, when WebSphere Application Server security and service integration bus security are enabled, access to the HTTP endpoint listener is restricted. For more information about why you might want to do this, see Endpoint listeners and inbound ports: Entry points to the service integration bus.

To configure HTTP endpoint listener authentication, complete the following steps:

1. Optional: If you want to change the HTTP endpoint listener security role, use an assembly tool to modify the endpoint listener application by completing the following steps:
   a. In the endpoint listener enterprise application, edit the Web application deployment descriptor to add a new role with a name of your choice.
   b. Remove the existing role (for example `AuthenticatedUsers`) from the authorized roles within the security constraint, then add the role you created in the previous step.
   c. Save the modified endpoint listener application.
2. Create the HTTP endpoint listener configuration.

3. Map the HTTP endpoint listener security role to users or groups by completing the following steps:

   Note: The default security role `AuthenticatedUsers` is mapped to the special group **Everyone**. That is, even if WebSphere Application Server security is enabled all users can access any inbound service deployed to the HTTP endpoint listener. To restrict access to just authenticated users, map the role to the special group named **All authenticated**.

   a. Turn on WebSphere Application Server security.
   b. Start the WebSphere Application Server administrative server.
   c. Start the administrative console.
   d. In the navigation pane, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***

      where *application_name* is the name of the EAR file for this listener. For example `soaphttpchannel1`. In the additional properties for this listener application, an option to map security roles to users and groups is displayed.
   e. Assign users and groups to the security role. For example, map the `AuthenticatedUsers` role to the **All authenticated** group.
   f. Click **OK**.
   g. Save your changes to the master configuration.

Password-protecting a Web service operation

Password-protect individual operations (methods) in a Web service by creating an enterprise bean with methods matching the Web service operations, then applying WebSphere Application Server authentication mechanisms to the enterprise bean so that, before a Web service operation is invoked, a call is made to the EJB method for authorization.

Before you begin

As well as password-protecting a Web service operation as described in this topic, you must also configure the service as either an inbound or outbound service, and select the option to **Enable operation-level security** as described in Modifying an existing inbound service configuration or Modifying an existing outbound service configuration.

In order for an application deployed to the service integration bus to use operation-level security, you must set the application server class-loader policy to `single`, as described in Configuring class loaders of a server.

About this task

For operation-level authorization you create an enterprise bean with methods matching the Web service operations. These EJB methods perform no operation and are just entities for applying security. You then apply existing WebSphere Application Server authentication mechanisms to the enterprise bean. Before any Web service operation is invoked, a call is made to the EJB method. If authorization is granted, the Web service is invoked.

Your target Web service is protected by wrapping it in an EAR file (*your_webservice*`.ear`), then applying role-based authorization to the EAR file. This process is explained in general terms in Operation-level security: Role-based authorization. The *your_webservice*`.ear` file is then imported into the `sibwsauthbean.ear` file and the `sibwsauthbean.ear` file is modified to set the roles and assign them to methods. The modified `sibwsauthbean.ear` file is then deployed in WebSphere Application Server, and users are assigned to the previously-defined roles.

The installation version of the `sibwsauthbean.ear` file is in the *app_server_root*`/installableApps` directory, where *app_server_root* is the root directory for the installation of WebSphere Application Server.

The sibwsauthbean.ear file contains an EAR file for each Web service that you protect. For the first Web service that you protect through operation-level authorization, you copy the installation version of the sibwsauthbean.ear file and store your copy outside of the application server file system. For each subsequent Web service that you protect, you further modify the same copy of the sibwsauthbean.ear file.

To password-protect Web service operations, complete the following steps for each Web service that you want to protect:

1. For the first Web service that you protect, make your own copy of the *app_server_root*/installableApps/SIBWSauthbean.ear file in a convenient location outside of the application server file system.
2. To create the *your_webservice*.ear file, complete the following steps:
   a. Open a command prompt.
   b. Go to the *app_server_root*/util directory.
   c. Enter the following command:

      sibwsAuthGen *location your_webservice_name*

      where
      - *location* is the service WSDL location. For an outbound service, you need the target WSDL file that is located at an external Web address. For an inbound service, you need the template WSDL file that is located at the endpoint listener endpoint for the service.
      - *your_webservice_name* is the name of the service that you are securing, as defined in the location field of the WSDL file. This is case-sensitive.

      Note: To get the location details for a given inbound service WSDL file, publish the WSDL file to a zip file as described in Modifying an existing inbound service configuration, then look up the location within the exported WSDL file. Alternatively, you can retrieve the inbound service WSDL file by using the following standard Web services query:

      http://*host_name*:*port_number*/*epl_context_root*/soaphttpengine/*bus_name*/*inbound_service_name*/*inbound_port_name*?wsdl

      where *host_name* and *port_number* are the host name and port number for this application server, and *epl_context_root* is the context root of the endpoint listener application as described in Modifying an existing endpoint listener configuration.

      Examples of using the sibwsAuthGen command:

      (outbound service):

      sibwsAuthGen http://*www.somecompany.com*/*targetservice*/wsdl/*targetservice*.wsdl *targetServiceName*

      (inbound service):

      sibwsAuthGen http://your.server.name:9080/wsgwsoaphttp1/soaphttpengine/yourbus/yourservice/inboundport1?wsdl yourservicename

   The *your_webservice*.ear file is created in the current directory. There is also a temporary directory *current_directory*/ejb that you can delete.
3. To finish assigning roles and protecting methods, complete the steps given in the topic Using assembly tools to Password-protect a Web service operation.
4. To install the modified copy of the sibwsauthbean.ear file, complete the following steps:
   a. Check that the modified sibwsauthbean.ear file is stored in the convenient location outside of the application server file system that you chose in step 1. Keep the sibwsauthbean.ear file in this location for subsequent reuse and further modification.
   b. Start the WebSphere Application Server administrative console.
   c. In the navigation pane, select **Applications > Install an Application**.
   d. Use **Install New Application** to install the modified copy of the sibwsauthbean.ear file. Select the users or groups to assign to the roles when prompted.

Using assembly tools to password-protect a Web service operation:
Before you begin

This task assumes that you have already completed the initial steps for Password-protecting a Web service operation.

About this task

As is explained in general terms in Operation-level security: Role-based authorization, your target Web service is protected by wrapping it in an EAR file and applying role-based authorization to the EAR file. In this task, the EAR file that contains your Web service (*your_webservice*.ear) is imported into the sibwsauthbean.ear file (which contains all of the protected Web services) and the sibwsauthbean.ear file is modified to set the roles and assign them to methods. This modified sibwsauthbean.ear file is then deployed in WebSphere Application Server and users are assigned to the previously defined roles.

Use an assembly tool to complete the following steps:

1. Start the assembly tool, then open the Java EE perspective.
2. From the File menu select **File > Import > EAR**, then browse to select your copy of the sibwsauthbean EAR file. On the Project Explorer tab these projects are created:
   - An enterprise application project called sibwsauthbean
   - An EJB project called Authorization
3. From the File menu select **File > Import > EAR**, specify a new EAR project name, then browse to select the *your_webservice* EAR file. On the Project Explorer tab these projects are created:
   - An enterprise application project called *your_webservice*.
   - An EJB project called *your_webservice* ejb.
4. Select the EJB project *your_webservice* ejb, then edit the **EJB Deployment Descriptor**. For every security role that you want to create, repeat the following steps:
   a. On the Assembly tab, add the required security role (for example READER).
   b. Use the Add Method Permission wizard to add one or more method permissions to the security role.
   c. Save your changes.
5. To import the enterprise application *your_webservice* into the sibwsauthbean EAR file, complete the following steps:
   a. Select the enterprise application project sibwsauthbean, then edit the **EAR Deployment Descriptor**.
   b. On the Module tab, add the *your_webservice* ejb enterprise bean from the EJB project *your_webservice* ejb.
   c. Save your changes.
6. To ensure that the authorization enterprise bean can reference the newly-imported enterprise bean, complete the following steps to add an EJB reference:
   a. Select the EJB project Authorization, then edit the **EJB Deployment Descriptor**.
   b. On the Reference tab, select the Authorization reference then click **Add**. The Add Reference wizard is displayed.
   c. Select **EJB Reference > Next**.
   d. Select the **Enterprise beans in the workspace** radio button, then browse to select the *your_webservice* ejb enterprise bean.
   e. Save your changes.
7. To assign users to roles, complete the following steps:
   a. Select the enterprise application project sibwsauthbean, then edit the **EAR Deployment Descriptor**.

b. On the Security tab, select **Gather**. For every security role that you want to assign, repeat the following steps:
   1) Select a security role.
   2) Under **WebSphere Bindings**, select the required access level from the following choices:
      - Everyone
      - All authenticated
      - Users/Groups

8. Export the enterprise application project `sibwsauthbean` as an EAR file.

What to do next

You are now ready to install the modified copy of the `sibwsauthbean` EAR file as described in the final step of Password-protecting a Web service operation.

Invoking a password-protected outbound service

Invoke a password-protected external Web service by configuring and deploying a JAX-RPC handler to set the associated user ID and password.

About this task

Providers of external Web services can use HTTP basic authentication to secure their services. When you configure an outbound service to invoke an external Web service that requires HTTP basic authentication, you configure and deploy a JAX-RPC handler at the outbound port to provide the required user ID and password in the form of an HTTP Basic Authentication header. To configure and deploy this handler, complete the following steps.

1. Create a new JAX-RPC handler class that sets the properties `javax.xml.rpc.security.auth.username` and `javax.xml.rpc.security.auth.password`. For example:

```
public class BasicAuthHandler extends GenericHandler {

    public QName[] getHeaders() {
        return null;
    }

    public boolean handleRequest(MessageContext mc) {

        // Insert basic auth properties
        mc.setProperty("javax.xml.rpc.security.auth.username", "bob");
        mc.setProperty("javax.xml.rpc.security.auth.password", "xy129bge");
        return super.handleRequest(mc);
    }
}
```

2. Create a new JAX-RPC handler configuration for the handler.
3. Create a new JAX-RPC handler list, then select the handler that sets the HTTP basic authentication properties for this service and add it to the handler list.
4. Use the instructions given in Modifying an existing outbound service configuration to navigate to the administrative console page **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services** → ***service_name*** → **Outbound Ports** → ***port_name***, where `service` and `port` indicate the outbound port at which you apply the HTTP basic authentication properties.
5. Set the **JAX-RPC Handler list** property by selecting, from the selection list, the handler list that sets the HTTP basic authentication properties for this service.
6. Save your changes to the master configuration.

Accessing a password-protected proxy server

Configure access to an external Web service or WSDL file through a password-protected proxy server.

About this task

Service integration technologies requires access to the Internet for invoking outbound services and for retrieval of external WSDL files. Many enterprise installations use a proxy server in support of Internet routing, and many proxy servers require authentication before they grant access to the Internet. This requirement is supported in HTTP messaging by a Proxy-Authorization message header that contains encoded user ID and password credentials.

To enable service integration technologies to invoke an outbound service you configure, for each outbound port, a proxy host, port and J2C authentication alias.

When you create or modify inbound or outbound services, the service integration bus might also need to pass messages through an authenticating proxy server to retrieve WSDL documents. Consequently you must configure the proxy host and port that are used.

Note: Neither the administrative console panels used to create a new Web service configuration, nor the **Reload WSDL** button provided on the panels used to modify an existing Web service configuration, allow you to enter an authentication alias for WSDL retrieval. If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you must use command-line tools to retrieve the WSDL.

1. Start the WebSphere Application Server administrative server.
2. Start the administrative console.
3. To enable invocation of an outbound service through a password-protected proxy server, complete the following steps:
   a. In the administrative console navigation pane, click **Service integration** → **Buses** → ***security_value*** → **[Related Items] JAAS - J2C authentication data**.
   b. Create a J2C authentication alias, providing an alias name, and the user ID and password required by the authenticating proxy server.
   c. Click **OK**.
   d. In the administrative console navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services** → ***service_name*** → **Outbound Ports** → ***port_name***.
   e. Type into the appropriate fields the authenticating proxy host name, port, and the authentication alias you created.
   f. Click **OK**.
4. To enable the service integration bus to pass messages through an authenticating proxy server to retrieve WSDL documents, complete the following steps:
   a. In the administrative console navigation pane, select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server Infrastructure] Java and Process Management** → **Process Definition** → **[Additional Properties] Java Virtual Machine** → **[Additional Properties] Custom Properties**.
   b. Set the following properties:
      - **http.proxySet** - Set this to `true` to tell the application server that it is required to work with an authenticating proxy.
      - **http.proxyHost** - Set this to the machine name of the authenticating proxy.
      - **http.proxyPort** - Set this to the port through which the authenticating proxy is accessed. For example `8080`.
      - **http.nonProxyHosts** - List the internal machines for which authentication is not required for routing through the proxy. Separate each machine name in the list with a vertical bar ("|").
      - This list must include the machine on which the bus is installed.

   Note: If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you must use command-line tools to retrieve the WSDL.
5. Save your changes to the master configuration.

6. Stop then restart the application server.
7. Close the administrative console.

Invoking outbound services over HTTPS

Use Secure Sockets Layers (SSL) to allow the service integration bus to invoke external Web services that include `https://` in their addresses.

About this task

There are two ways to set the bus to use SSL with SOAP over HTTPS messages:

- Configure SSL certificate and key management for a managed endpoint.
- Use a JAX-RPC handler to set the SSL configuration.

By default, each managed endpoint is already configured to use SSL. However you will need to modify the default configuration, for example to add information about the keys and keystores that the external Web service uses.

Alternatively, you can use a JAX-RPC handler to set the SSL configuration. You might want to do this because you are upgrading from a previous version of WebSphere Application Server and your configuration is already set to work in this way, or because you need to target an SSL configuration very precisely; for example to secure each service or each invocation.

To configure SSL certificate and key management for a managed endpoint, see Creating a Secure Sockets Layer configuration.

To use a JAX-RPC handler to set the SSL configuration, complete the following steps:

1. Start the administrative console.
2. Create a new Secure Sockets Layer repertoire configuration entry.
3. Create a new JAX-RPC handler class that sets the property `ssl.configName` to a value that is the name of the SSL repertoire configuration that you have just created. For example:

```
public class SSLHandler extends GenericHandler {

    public QName[] getHeaders() {
        return null;
    }

    public boolean handleRequest(MessageContext mc) {

        // Insert SSL property
        mc.setProperty("ssl.configName", "myNode/SSLConfig");
        return super.handleRequest(mc);
    }
}
```

4. Create a new JAX-RPC handler configuration for the handler.
5. Create a new JAX-RPC handler list, then select the handler that sets the SSL configuration name property and add it to the handler list.
6. Use the instructions given in Modifying an existing outbound service configuration to navigate to the administrative console page **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services** → ***service_name*** → **Outbound Ports** → ***port_name***, where `service` and `port` indicate the outbound port that is to use SSL.
7. Set the **JAX-RPC Handler list** property by selecting, from the selection list, the handler list that sets the SSL configuration name property.
8. Save your changes to the master configuration.

Securing WS-Notification

The WS-Notification security implementation requires that a user identity is flowed in requests for WS-Notification services. This identity is used to authenticate the client application and check that the client is authorized to invoke the requested operation, and to access the underlying service integration bus topic spaces and topic resources.

About this task

WS-Notification uses the same mechanisms as other Web services to provide an authenticated identity. For example WS-Security or HTTP Basic Authentication.

There are three parts to configuring secure access to WS-Notification:

- Securing the communication channel between the application and the server.
- Authorizing the application to invoke the NotificationBroker.
- Authorizing the application to access the resources of the service integration bus.

If messaging security is enabled, and the WS-Security or HTTP Basic Authentication components are not configured to flow a user identity in WS-Notification requests, then all such requests are treated as unauthenticated and can only access messaging resources that are accessible by the WebSphere Application Server “everyone” group.

1. Secure the communication between the application and the server:
   a. Provide security for inbound requests and associated responses by configuring the WS-Notification service point.
      - For Version 7.0 WS-Notification service points, attach security-enabled policy sets to the application associated with the service point. For Version 6.1 WS-Notification service points, configure security for the inbound ports associated with the service point.
      - If you are using SOAP over HTTP as the binding for your WS-Notification service point, modify it to use SOAP over HTTPS as described in “Configuring secure access to WS-Notification service points using SOAP over HTTPS” on page 722.
      - If you are using SOAP over JMS as the binding (for Version 6.1 WS-Notification services), configure the JMS connection factory used by the client application to use a secure communication protocol to communicate with the JMS provider. Exactly how you do this depends upon the JMS provider. If you are using the service integration bus as the JMS provider, configure the client to use SSL to communicate with the server by setting the **target inbound transport chain** to `InboundSecureMessaging` as described in How JMS applications connect to a messaging engine on a bus and its related tasks.
   b. Provide security for outbound requests (for example notifications from the server to subscribed consumers) by configuring the WS-Notification service.
      - For Version 7.0 WS-Notification services, the steps involved are similar to those for applying security to JAX-WS Web service clients except that any binding or configuration created is applied to the WS-Notification service. For more information, see Securing JAX-WS Web services using message level security.
      - For Version 6.1 WS-Notification services, the steps involved are similar to those for applying security to service integration bus-enabled Web services outbound ports except that any binding or configuration created is applied to the WS-Notification service. For more information, see “Securing bus-enabled Web services” on page 709 and its sub-topics, notably “Invoking outbound services over HTTPS” on page 720.
   c. You can also use WS-Security to sign or encrypt SOAP messages.
      - For Version 7.0 WS-Notification services, see Signing and encrypting message parts using policy sets.

- For Version 6.1 WS-Notification services, see “Configuring secure transmission of SOAP messages using WS-Security” on page 712.

2. Authorize the application to invoke the NotificationBroker:
   a. Configure the client application to provide an appropriate identity.

      To authorize a Web service application to communicate with the server, the application must identify itself as running as a particular authenticated identity. The mechanism for doing this depends upon the type of Web service binding you are using:
      - If you are using SOAP over HTTP Web service bindings, use either HTTP Basic Authentication or WS-Security to provide the authenticated identity.
      - If you are using SOAP over JMS Web service bindings (for Version 6.1 WS-Notification services), use WS-Security to provide an authenticated identity.

   b. Configure the server to authorize the client application identity to carry out the required operations.
      - For Version 7.0 WS-Notification services, you can use Web services policy sets such as the “Username WS-I RSP default” or “Username WSSecurity default” policy sets to apply authorization to the Web services that are deployed in the enterprise application associated with a service point.
      - For Version 6.1 WS-Notification services, you can apply authorization to the whole of an inbound service (for example the NotificationBroker endpoint of a WS-Notification service point) as described in “Password-protecting inbound services” on page 714, or configure authorization constraints independently for each Web service operation as described in “Password-protecting a Web service operation” on page 715.

3. Authorize the application to access the resources of the service integration bus.

   Service integration bus security uses role-based authorization. When a user is assigned to a role, the user is granted all of the permissions that the role contains. By administering authorization permissions, you can control user access to a bus and to its resources when messaging security is enabled.
   a. Authorize the application identity to be able to connect to the service integration bus, as described in “Administering the bus connector role” on page 675.
   b. When the application can connect to the bus, grant the application access to the appropriate destinations on the bus.

      You can determine which service integration bus topic spaces are required, by checking which WS-Notification topic namespaces are used by the application then looking at the appropriate WS-Notification permanent topic namespace to find the service integration bus topic space to which it maps. You can then grant authorization (for example the Sender or Receiver roles) for the authenticated identity to access that topic space as described in “Administering destination roles” on page 680.
   c. After the client application has been authorized to access the appropriate topic space destination, you might also need to authorize the client application to access the individual topics within the topic space destination as described in “Administering topic roles” on page 695.

   For general information about configuring access to the service integration bus, see bus security.

Configuring secure access to WS-Notification service points using SOAP over HTTPS

Modify the configuration of existing WS-Notification service points to require use of SOAP over HTTPS instead of SOAP over HTTP as the binding for inbound requests from notification providers.

Before you begin

This task assumes that you have an existing WS-Notification service and service points, and that you are using the SOAP over HTTP binding for inbound requests.

About this task

By default the SOAP over HTTP endpoints used by service points accept both HTTP and HTTPS requests. HTTPS can be used by changing the endpoint URL prefix to `https://` on each WS-Notification service point for the service, and modifying the port to the HTTPS port used by the server (default of 9443).

For Version 7.0 WS-Notification services, enterprise applications are used to expose the Web services associated with the WS-Notification service. For Version 6.1 WS-Notification services, service integration endpoint listeners are used.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**, then identify the WS-Notification service points for the WS-Notification service you want to secure.
3. Configure HTTPS access individually on each of the WS-Notification service points by repeating the following sub-steps:

   For Version 7.0 WS-Notification services:

   a. In the content pane, click the name of a Version 7.0 WS-Notification service point in the list.
   b. Navigate to the associated enterprise application by clicking **[Additional Properties] Service point application**. The enterprise application settings panel is displayed.

      Note: You can also reach this panel by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
   c. In the enterprise application settings panel, click **[Web Services Properties] Provide HTTP endpoint URL information**.
   d. Specify the endpoint URL prefix (that is the protocol (HTTPS), host name, and port number) to use in the endpoint URL. You can select the default HTTPS prefix (`https://`*`your_host_name`*`:9443`) from a predefined list, or you can create and use your own custom HTTPS prefix. For more information, see Configuring endpoint URL information for HTTP bindings.

   For Version 6.1 WS-Notification services:

   a. Create a new endpoint listener with an `https` URL as the URL root.
   b. Modify this WS-Notification service point to associate the inbound port for the new endpoint listener with this service point. The `https` URL appears in the published WSDL file.
   c. Prevent the new endpoint listener from accepting HTTP connections by modifying the virtual host settings. For more information, see Virtual hosts and Creating a Secure Sockets Layer configuration.

Chapter 7. Messaging resources

Configuring authorization security for a Version 5 default messaging provider

Use this task to configure authorization security for the default messaging provider on a WebSphere Application Server Version 5 node.

About this task

To configure authorization security for the Version 5 default messaging provider complete the following steps.

Note: Security for the Version 5 default messaging provider is enabled when you enable WebSphere Application Server security on the Version 5 node. For more information about enabling security, see Enabling security.

1. Configure authorization settings to access JMS resources owned by the embedded messaging subsystem.

   Authorization to access JMS resources owned by the embedded messaging subsystem is controlled by settings in the *profile_root*\config\cells*your_cell_name*\integral-jms-authorizations.xml file. The settings grant or deny authenticated users access to messaging resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Pub, sub, and persist to all topics.

   To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.
2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

   ```
   <queue-admin-userids>
       <userid>adminid1</userid>
       <userid>adminid2</userid>
   </queue-admin-userids>
   ```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.
3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

   ```
   <queue-default-permissions>
     <permission>write</permission>
   </queue-default-permissions>
   ```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.
4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

   ```
   <queue>
     <name>q1</name>
     <public>
   ```

© Copyright IBM Corp. 2008

```
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
    <authorize>
      <userid>useridrw</userid>
      <permission>read</permission>
      <permission>write</permission>
    </authorize>
  </queue>
```

In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

      ```
      <topic>
         <name></name>
         <public>
           <permission>+pub</permission>
         </public>
       </topic>
      ```

      In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

   b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

      ```
      <topic>
        <name>a/b/c</name>
        <public>
          <permission>+sub</permission>
        </public>
        <authorize>
          <userid>useridpub</userid>
          <permission>+pub</permission>
        </authorize>
      </topic>
      ```

      In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

Results

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic

update setting is not selected, you need to stop and restart the JMS server to make changes active.

Authorization settings for Version 5 default JMS resources

Use the integral-jms-authorisations.xml file to view or change the authorization settings for Java Message Service (JMS) resources owned by the default messaging provider on WebSphere Application Server Version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*\config\cells\ *your_cell_name*\integral-jms-authorisations.xml file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
```

```
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update

Controls whether or not the JMS Server checks dynamically for updates to this file.

true (Default) Enables dynamic update support.

false Disables dynamic update checking and improves authorization performance.

queue-admin-userids

This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:

<userid>*adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions

This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:

<permission>*read-write***</permission>**

Where *read-write* is one of the following keywords:

read By default, userids have read access to Version 5 default queue destinations.

write By default, userids have write access to Version 5 default queue destinations.

queue

This element contains the following authorization settings for a single queue destination:

name The name of the queue.

public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
```

```
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
    <authorize>
      <userid>useridrw</userid>
      <permission>read</permission>
      <permission>write</permission>
    </authorize>
  </queue>
```

topic

This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission

+sub Grant subscribe permission

+persist

Grant persist permission

-pub Deny publish permission

-sub Deny subscribe permission

-persist

Deny persist permission

Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for Enterprise JavaBeans (EJB) Version 2.1 message-driven beans.

About this task

The association between connection factories, destinations, and message-driven beans is provided by listener ports. A listener port allows a deployed message-driven bean associated with the port to retrieve messages from the associated destination. You create listener ports by specifying their administrative name, the connection factory JNDI name, and the destination name (other optional properties are also configurable). Listener ports provide simplified administration of the associations between connection factories, destinations and message-driven beans, and are managed by a listener manager. The listener manager is provided by the message listener service to control and monitor the JMS listeners that are monitoring JMS destinations on behalf of deployed message-driven beans. For more information about listener ports, see Message-driven beans - listener port components

Messages handled by message-driven beans have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

Connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define an authentication alias on the J2C activation specification that the message-driven bean is configured with. If defined, the message-driven bean uses the authentication alias for its JMSConnection security credentials instead of any application-managed alias.

To set the authentication alias, you can use the administrative console to complete the following steps. This task description assumes that you have already created an activation specification. If you want to create a new activation specification, see the related tasks.

- For a message-driven bean listening on a JMS destination of the default messaging provider, set the authentication alias on a JMS activation specification.
  1. To display the JMS activation specification settings, click **Resources** → **JMS** → **JMS providers**
  2. In the content pane, click the name of a default messaging provider.
  3. In the content pane, under Additional Resources, click **Activation specifications**.
  4. If you have already created a JMS activation specification, click its name in the list displayed. Otherwise, click **New** to create a new JMS activation specification.
  5. Set the **Authentication alias** property.
  6. Click **OK**
  7. Save your changes to the master configuration.
- For a message-driven bean listening on a destination (or endpoint) of another Java EE Connector Architecture (JCA) provider, set the authentication alias on a J2C activation specification.
  1. To display the J2C activation specification settings, click **Resources** → **Resource Adapters** → **adapter_name** → **J2C Activation specifications** → **activation specification_name**
  2. Set the **Authentication alias** property.
  3. Click **OK**
  4. Save your changes to the master configuration.

Chapter 8. Mail, URLs, and other J2EE resources

JavaMail security permissions best practices

In many of its activities, the JavaMail API needs to access certain configuration files. The JavaMail and JavaBeans Activation Framework binary packages themselves already contain the necessary configuration files. However, the JavaMail API allows the user to define user-specific and installation-specific configuration files to meet special requirements.

The two locations where you can place these configuration files are the <user.home> and <java.home>/lib directories. For example, if the JavaMail API needs to access a file named mailcap when it sends a message, the API:

1. Tries to access <user.home>/mailcap.
2. If the first attempt fails due to a lack of security permission or a nonexistent file, the API searches in <java.home>/lib/mailcap.
3. If the second attempt also fails, the API searches in the META-INF/mailcap location in the class path. This location actually leads to the configuration files contained in the mail-impl.jar and activation-impl.jar files.

Application Server uses JavaMail API configuration files that are contained in the mail-impl.jar and activation-impl.jar files, and there are no mail configuration files in <user.home> and <java.home>/lib directories. To ensure proper functioning of the JavaMail API, Application Server grants *file read* permission for both the mail-impl.jar and activation-impl.jar files to all of the installed applications.

JavaMail code attempts to access configuration files at <user.home> and <java.home>/lib, which can cause an access control exception to be thrown, since the default configuration does not grant file read permission for those two locations by default. This activity does not affect the proper functioning of the JavaMail API, but you might see a large amount of mail-related security exceptions reported in the system log, and these errors could overshadow harmful errors for which you are looking. This is a sample of the security message, SECJ0314W:

```
[02/31/08 12:55:38:188 PDT] 00000058 SecurityManag W   SECJ0314W: Current Java 2 Security policy reported a
potential violation of Java 2 Security Permission.
Please refer to Problem Determination Guide for further information.

Permission:

      D:\o063919\java\jre\lib\javamail.providers : access denied (java.io.FilePermission
D:\o063919\java\jre\lib\javamail.providers read)


Code:

     com.ibm.ws.mail.SessionFactory  in  {file:/D:/o063919/lib/runtime.jar}



Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission D:\o063919\java\jre\lib\javamail.providers read)
 at java.security.AccessControlContext.checkPermission(AccessControlContext.java(Compiled Code))
 at java.security.AccessController.checkPermission(AccessController.java(Compiled Code))
 at java.lang.SecurityManager.checkPermission(SecurityManager.java(Compiled Code))
 at com.ibm.ws.security.core.SecurityManager.checkPermission(SecurityManager.java(Compiled Code))
 at java.lang.SecurityManager.checkRead(SecurityManager.java(Compiled Code))
 at java.io.FileInputStream.<init>(FileInputStream.java(Compiled Code))
 at java.io.FileInputStream.<init>(FileInputStream.java:89)
 at javax.mail.Session.loadFile(Session.java:1004)
 at javax.mail.Session.loadProviders(Session.java:861)
 at javax.mail.Session.<init>(Session.java:191)
 at javax.mail.Session.getInstance(Session.java:213)
 at com.ibm.ws.mail.SessionFactory.getObjectInstance(SessionFactory.java:67)
 at javax.naming.spi.NamingManager.getObjectInstance(NamingManager.java:314)
 at com.ibm.ws.naming.util.Helpers.processSerializedObjectForLookupExt(Helpers.java:894)
 at com.ibm.ws.naming.util.Helpers.processSerializedObjectForLookup(Helpers.java:701)
```

© Copyright IBM Corp. 2008

```
 at com.ibm.ws.naming.jndicos.CNContextImpl.processResolveResults(CNContextImpl.java:1937)
 at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1792)
 at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1707)
 at com.ibm.ws.naming.jndicos.CNContextImpl.lookupExt(CNContextImpl.java:1412)
 at com.ibm.ws.naming.jndicos.CNContextImpl.lookup(CNContextImpl.java:1290)
 at com.ibm.ws.naming.util.WsnInitCtx.lookup(WsnInitCtx.java:145)
 at javax.naming.InitialContext.lookup(InitialContext.java:361)
 at emailservice.com.onlinebank.bpel.EmailService20060907T224337EntityAbstractBase$JSE_6.
execute(EmailService20060907T224337EntityAbstractBase.java:32)
 at com.ibm.bpe.framework.ProcessBase6.executeJavaSnippet(ProcessBase6.java:256)
 at emailservice.com.onlinebank.bpel.EmailService20060907T224337EntityBase.invokeSnippet
(EmailService20060907T224337EntityBase.java:40)
```

Note: If this situation is a problem, consider adding more read access permissions for more locations. This should eliminate most, if not all, JavaMail-related harmless security exceptions from the log file.

The permissions required by JavaMail are as follows:

```
grant codeBase "file:${application}" {
  // Allow access to default configuration files
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}javamail.address.map", "read";
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}javamail.providers", "read";
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}mailcap", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}javamail.address.map", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}javamail.providers", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
  permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}mail-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}plugins${/}com.ibm.ws.prereq.javamail.jar", "read";
  // If using an isolated mail provider,
  // add additional file read permissions for each jar defined
  // for the isolated mail provider
  // permission java.io.FilePermission "path${/}mail.jar, "read";

  // Allow connection to mail server using SMTP
  permission java.net.SocketPermission "*:25", "connect,resolve";
  // Allow connection to mail server using SMTPS
  permission java.net.SocketPermission "*:465", "connect,resolve";

  // Allow connection to mail server using IMAP
  permission java.net.SocketPermission "*:143", "connect,resolve";
  // Allow connection to mail server using IMAPS
  permission java.net.SocketPermission "*:993", "connect,resolve";

  // Allow connection to mail server using POP3
  permission java.net.SocketPermission "*:110", "connect,resolve";
  // Allow connection to mail server using POP3S
  permission java.net.SocketPermission "*:995", "connect,resolve";

  // Allow System.getProperties() to be used
  // permission java.util.PropertyPermission "*", "read,write";
  // Otherwise use the following to allow system properties to be read
  permission java.util.PropertyPermission "*", "read";
};
```

Chapter 9. Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

Your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application.

Scheduler

Securing scheduler tasks

Scheduled tasks are protected using application isolation and administrative roles. This topic describes how to secure scheduler tasks.

About this task

If a task is created using a Java Platform, Enterprise Edition (Java EE) server application, only applications with the same name can access those tasks. Tasks created with a WASScheduler MBean using the AdminClient interface or scripting are not part of a J2EE application and have access to all tasks regardless of the application with which they were created. Tasks created with a WASScheduler MBean are only accessible from the WASScheduler MBean API and are not accessible from the Scheduler API.

If the Use Administration Roles attribute is enabled on a scheduler and administrative security is enabled on the Application Server, all Scheduler API methods and WASScheduler MBean API operations enforce access based on the WebSphere Administration Roles. If either of these attributes are disabled, then all API methods are fully accessible by all users.

1. Enable security for all application servers.
2. Manage schedulers.

© Copyright IBM Corp. 2008

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - IBM i

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations.

app_client_root
The default installation root directory for the Java EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V7/client directory.

app_client_user_data_root
The default Java EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V7/client directory.

app_client_profile_root
The default Java EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V7/client/profiles/*profile_name* directory.

app_server_root
The default installation root directory for WebSphere Application Server - Express is the /QIBM/ProdData/WebSphere/AppServer/V7/Express directory.

cip_app_server_root
The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_profile_root
The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_user_data_root
The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

if_root This directory represents the root directory of the IBM WebSphere Installation Factory. Because you can download and unpack the Installation Factory to any directory on the file system to which you have write access, this directory's location varies by user. The Installation Factory is an Eclipse-based tool which creates installation packages for installing WebSphere Application Server in a reliable and repeatable way, tailored to your specific needs.

iip_root
This directory represents the root directory of an *integrated installation package* (IIP) produced by the IBM WebSphere Installation Factory. Because you can create and save an IIP to any directory on the file system to which you have write access, this directory's location varies by user. An IIP is an aggregated installation package created with the Installation Factory that can include one or more generally available installation packages, one or more customized installation packages (CIPs), and other user-specified files and directories.

© IBM Corporation 2005, 2007

java_home

The following directories are the root directories for all supported Java Virtual Machines (JVMs).

JVM	Directory
Classic JVM	/QIBM/ProdData/Java400/jdk6
32–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit
64–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit

plugins_profile_root

The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver/profiles/*profile_name* directory.

plugins_root

The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V7/webserver directory.

plugins_user_data_root

The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver directory.

product_library
product_lib

This is the product library for the installed product. The product library for each Version 7.0 installation on the system contains the program and service program objects (similar to .exe, .dll, .so objects) for the installed product. The product library name is QWAS7*x* (where *x* is A, B, C, and so on). The product library for the first WebSphere Application Server Version 7.0 product installed on the system is `QWAS7A`. The *app_server_root*/properties/product.properties file contains the value for the product library of the installation, `was.install.library`, and is located under the *app_server_root* directory.

profile_root

The default directory for a profile named *profile_name* for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express/profiles/*profile_name* directory.

shared_product_library

The shared product library, which contains all of the objects shared by all installations on the system, is `QWAS7`. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

updi_root

The default installation root directory for the Update Installer for WebSphere Software is the /QIBM/ProdData/WebSphere/UpdateInstaller/V7/updi directory.

user_data_root

The default user data directory for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express directory.

The profiles and profileRegistry subdirectories are created under this directory when you install the product.

web_server_root

The default web server path is /www/*web_server_name*.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

© Copyright IBM Corp. 2008

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

© Copyright IBM Corp. 2008