IBM Integration Bus

Web Services Security:
Authentication and Encryption
Using X.509 Certificates

Featuring:

- Policy Sets and Policy Set Bindings
- Full and Partial Message Encryption
- Client Authentication and TLS

January 2016
Hands-on lab built at product
Version 10.0.0.3

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

                         Authentication and Encryption using X.509 Certificates
                                      Provided by IBM BetaWorks

1. Introduction

This lab guide covers how IBM Integration Bus Integration Services can be secured using (message level) Web Services security and Transport Level Security (TLS) using X.509 certificates.

In all scenarios throughout this lab guide a Provider service, "EmployeeService" is called by a JSON client.

The TCP/IP Monitor tool, provided in the IIB Toolkit, is used to see the effects of the message level authentication and encryption that WS-Security configuration has on SOAP messages. A final scenario shows how the service can be further secured by using Transport Level Security (TLS) by defining the service to use https.

The guide is split into the following scenarios:

1) **Initially** the environment is prepared ready for the security scenarios. The JSON client and EmployeeService are deployed and tested without being configured to use security. The Public Key Infrastructure (PKI) is defined in this section, however no applications or services are configured to use the certificates defined in the PKI. In an ideal situation it is easier to resolve non security related problems without security working. At the end of this part the environment will look like this:

Web Services Security   Version 10.0.0.3
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
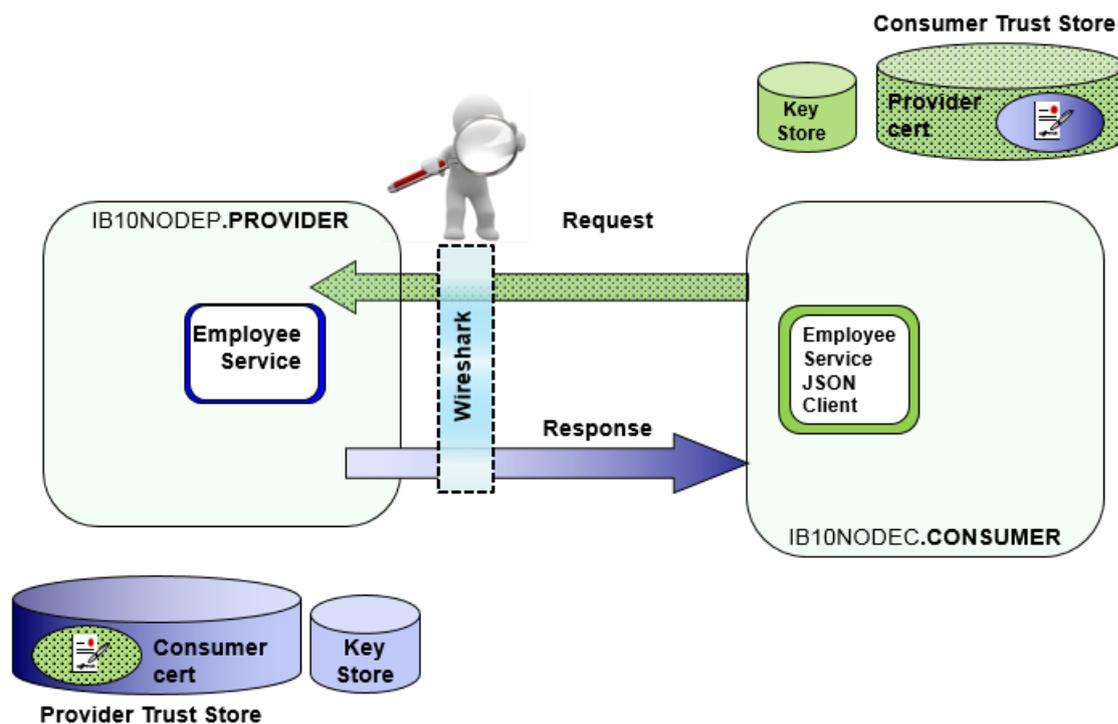
2) The next part covers the configuration necessary to enable **Authentication** so that successful communication between the JSON client and the EmployeeService is only possible given successful authentication between the two components. The following diagram shows at a high level the configuration required:

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
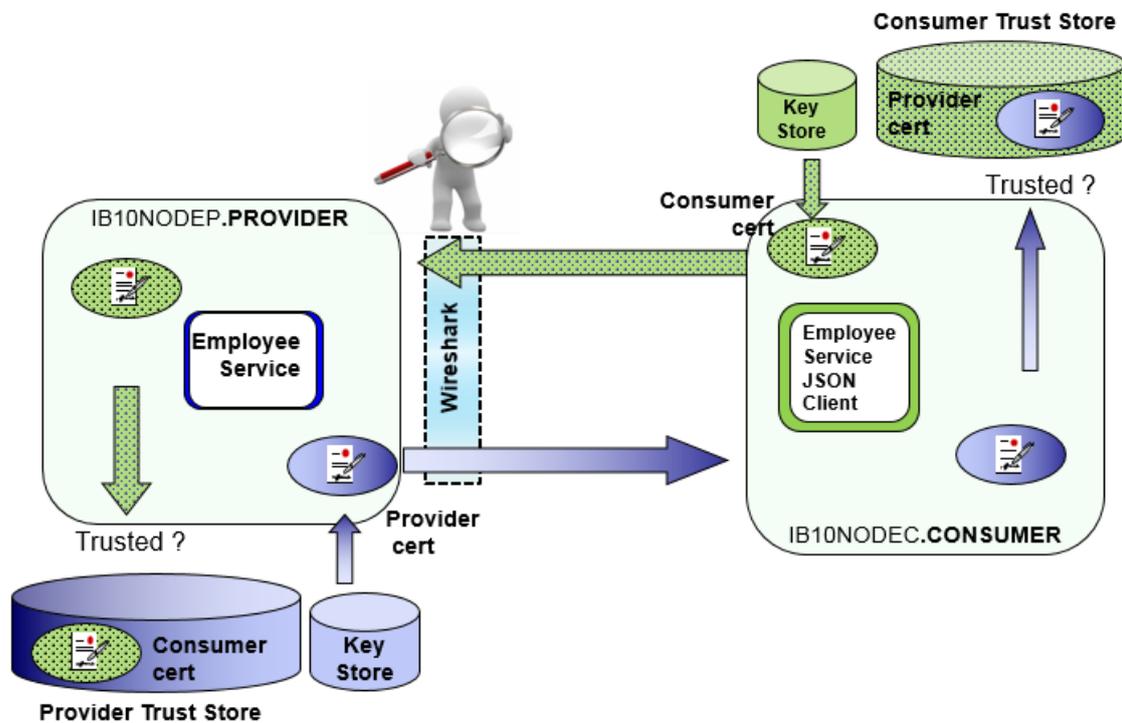
3) **Encryption** covers the configuration necessary to enable **Encryption** of the message content between the JSON client and the EmployeeService using X.509 certificates. The following diagram shows at a high level the configuration required:
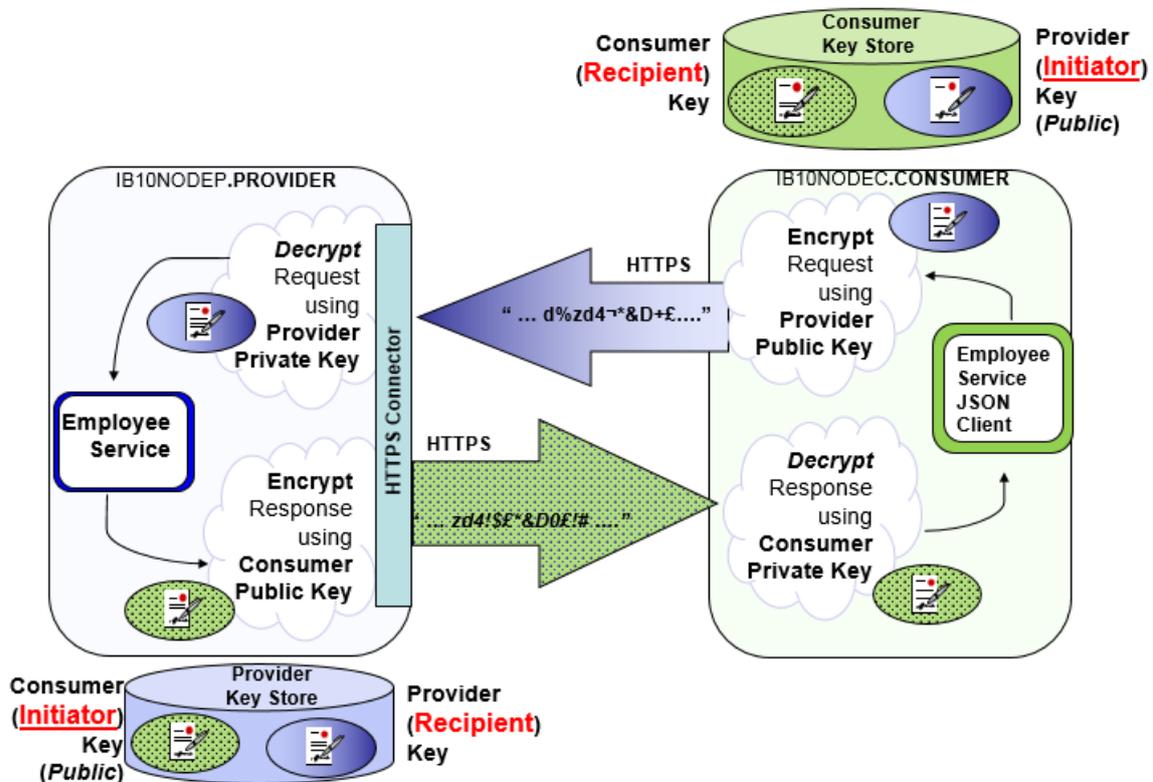
In this part you will see how IIB can be configured to support:

a) Full request and response message encryption

b) Partial message encryption (for example encryption of part of the message response containing sensitive data).

c) The addition of https Transport Level Security configuration between the JSON Client and the EmployeeService

2. Prepare the Environment

In the following sections you will prepare your environment to handle Web Services security using a public key infrastructure based on self-signed certificates. You will use a Provider based on the solution for the Integration Services Lab guide (EmployeeService), with the Consumer being based on the JSON client. The client is run on a separate Integration Node and Server and is used to call the EmployeeService, both these services are provided as a completed solution, there will be no need to develop these components.

The Consumer IIB environment will host the JSON Client application. The provider environment will host the EmployeeService.

2.1.1 The "Consumer"

A JSON-based client called EmployeeService_JSONClient will be used to call "the provider". The provider will retrieve employee details from the EMPLOYEE table, and return them to the EmployeeService_JSONClient.

The client will run in an Integration Server called CONSUMER running on node IB10NODEC.

2.1.2 The "Provider"

The "EmployeeService" is an Integration Service that will be used to obtain details registered in the EMPLOYEE (db2) table for a given key.

EmployeeService will run in an Integration Server called PROVIDER on node IB10NODEP.

2.1.3 Configuration scripts

Each part of this lab uses a number of scripts (Windows CMD files) to configure various parts of the IIB nodes. These use "mqsi" commands to perform the configuration. These are provided for simplicity of running the lab scenario, but you are encouraged to look in detail at the contents of these script files, and in particular the mqsi commands that are used.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

2.2 Set up IIB Nodes

In this section you will prepare the IIB nodes IB10NODEP and IB10NODEC. These nodes will be used to deploy the Provider web service and Consumer application.

1. Use the Integration Nodes view in the Toolkit to stop all Integration nodes that you have running.

2. Use "File > Switch Workspace > other" to create a new workspace, call it WSSEC. (*The Integration Toolkit will restart*).

3. Open an **Integration Console** and navigate to

    "C:\student10\WSSecurity\commands"

4. Run the command:

    00CreateNodes.10.0.0.3.cmd

    Accept the defaults when prompted to start the script.

    The script will create the IIB nodes used in this lab guide and configure them to run the Consumer and Provider web services you will use in this guide.

    Note: the script will attempt to delete the two nodes that it is adding before adding them. If the nodes do not exist in your environment you will see an error message (BIP8013E) saying that each node does not exist; you can ignore this message.

5. When prompted to establish if the node "IB10NODEP" is fully started, check the status of the nodes using the Toolkit (you may need to refresh the node in the Integration Nodes view to see the latest status).

    Press the enter key, in the Integration Console, when the node is fully started to continue the progress of the script.

    Wait for the script to finish.

6. When the script has successfully completed (you will see "00CreateNodes script complete") in the console. Check using IIB Toolkit to see the following new Integration Nodes and servers:
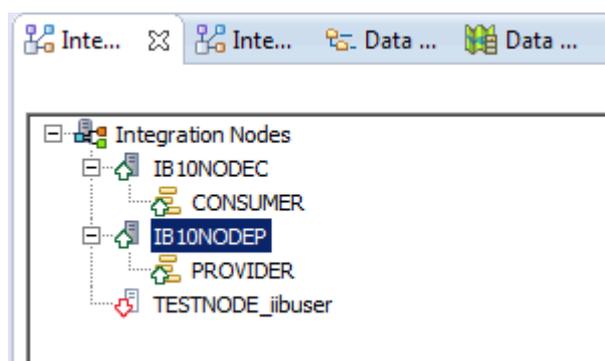
2.3 Set up TCP/IP Monitor

This lab will use the TCP/IP Monitor that is provided as part of the standard eclipse installation for IIB. This tool is not shown by default, so perform the following steps to display this tool.

The monitor captures and records TCP/IP traffic by acting as a proxy for the required endpoint, as shown below.

In this scenario, the monitor will listen for traffic on port 7802, and will forward it on to port 7800. Responses are automatically captured and recorded. To allow this, the Client application has to be changed to send the request to the port of the TCP/IP Monitor.
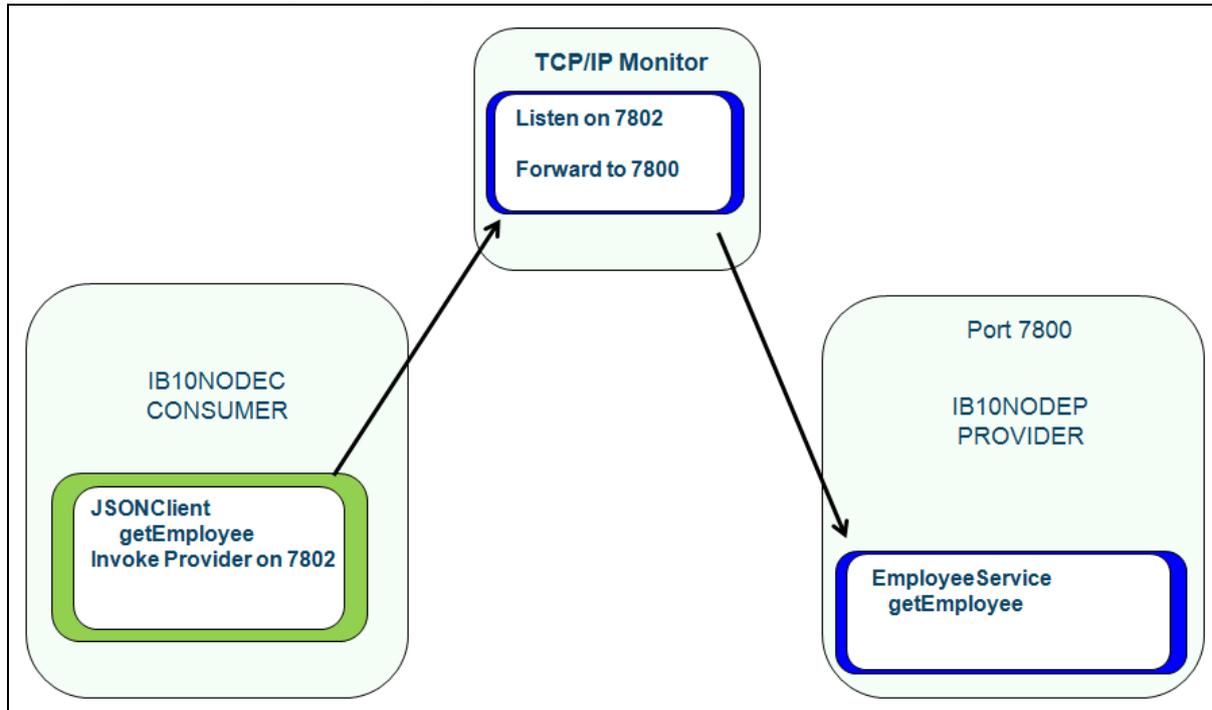
1.   In the IIB Toolkit, click Window, Show View, Other.
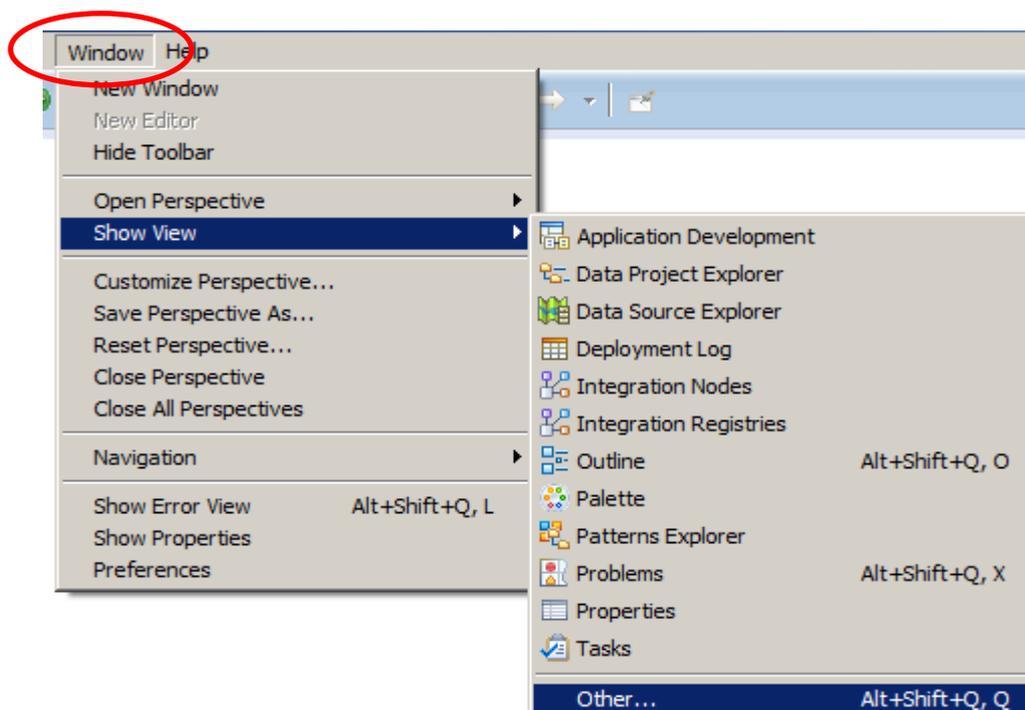
2.   In the Filter field, type TCP, select TCP/IP Monitor, and click OK.
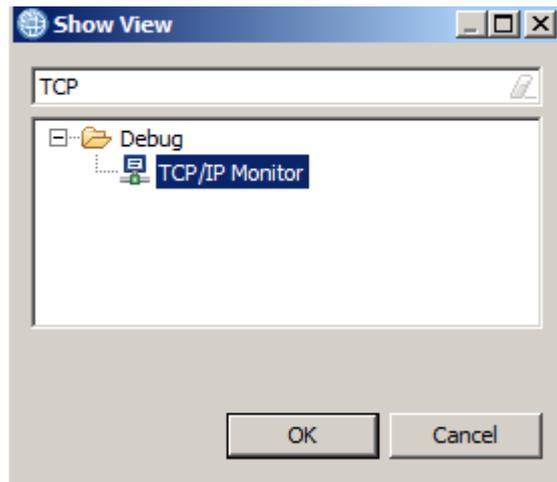
3.   The TCP/IP Monitor will open.
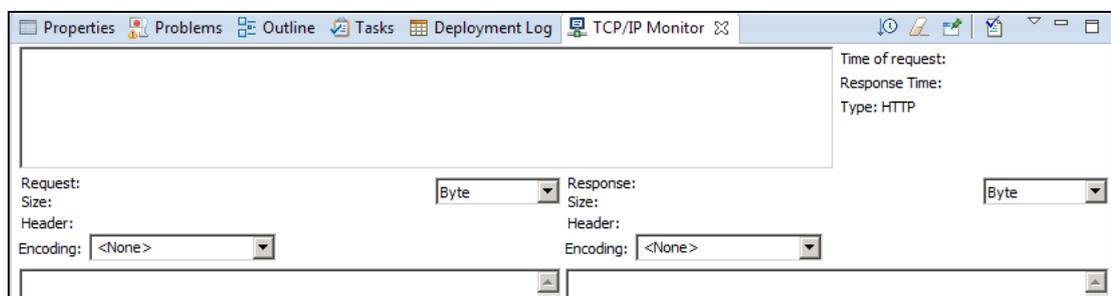
4.   Right-click in the open pane to create a new monitoring entry.
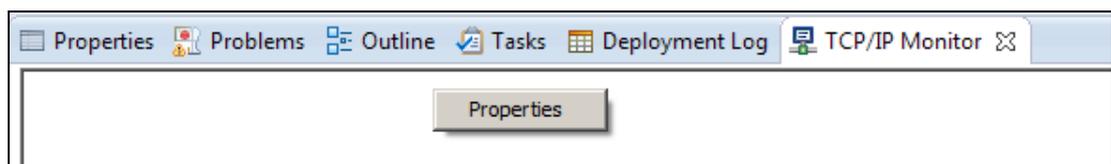
5.   Click Add to add a new entry for monitoring.
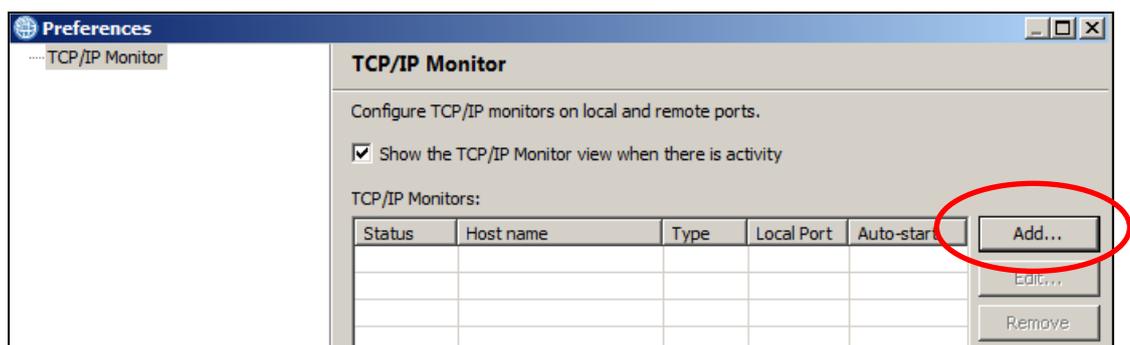
6. Set the following values:
    -   Local monitoring port: 7802 (this is the port that will be monitored by the Monitor).
    -   Host name: localhost
    -   Port: 7800   (this is the port that the monitor will forward TCPIP traffic to).

Click OK.
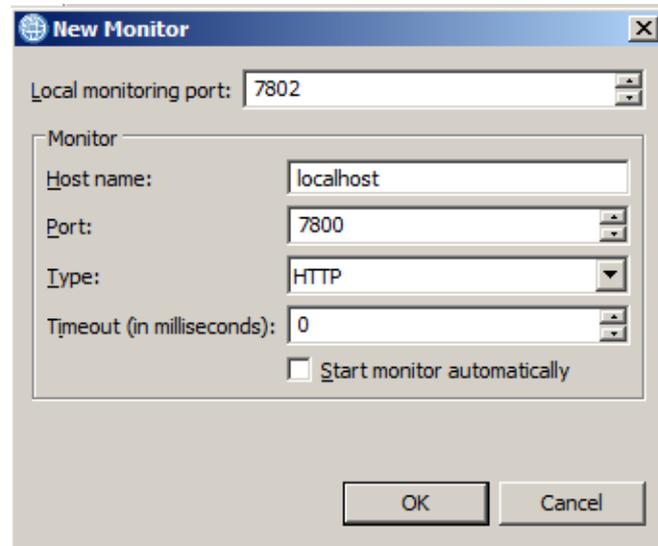
7. Now click Start to start the TCP/IP Monitor.
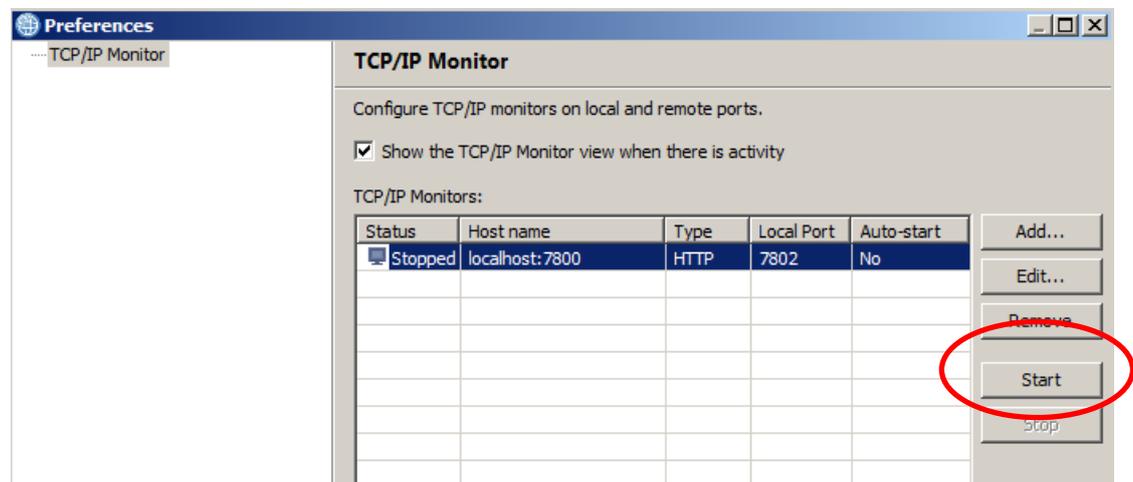
Click OK to close the TCPIP Monitor preferences.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

3. Prepare the Consumer and Provider

1.   Import the JSON Client application and EmployeeService provider into your workspace:

   - "**EmployeeService_JSONClient.getEmployee.10.0.0.3.zip**"

     (in "**C:\student10\Integration_service_JSONClient\solution**")

2.   Ensure all projects are selected, and click Finish.

Web Services Security   Version 10.0.0.3
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
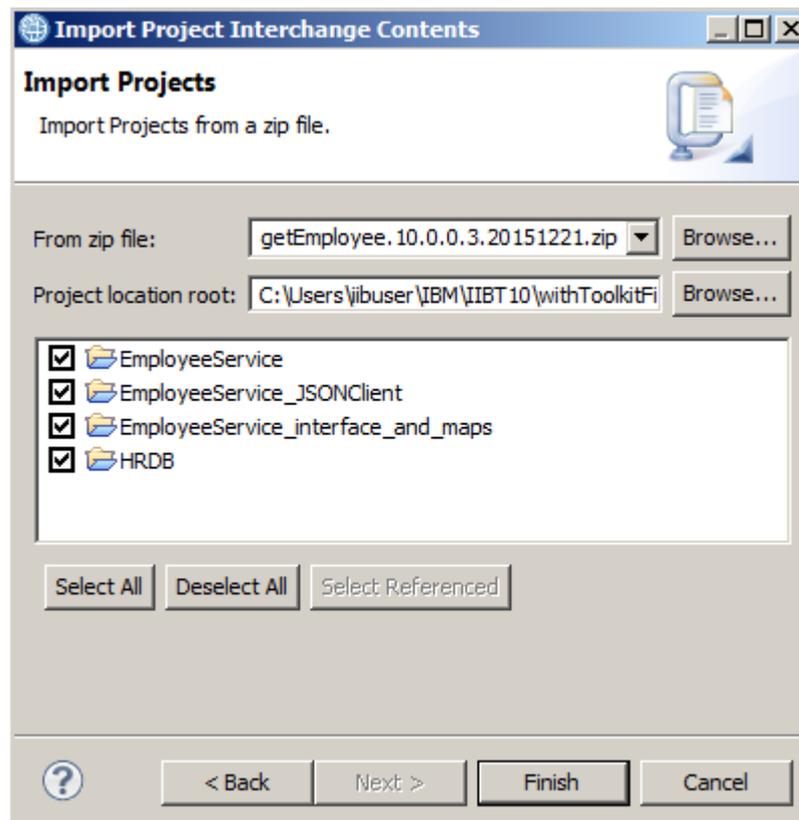
3.1 Prepare the Provider

In IBM Integration Bus, use of a web services security related Policy Set and binding is configured using the IIB bar file editor. You will now create and store the bar files that you will use to configure the authentication and encryption scenarios later in this guide.

3.1.1 Create EmployeeService bar file

1. In Integration Toolkit, right click on the EmployeeService service and select "New > Bar File" from this list of options.

    The "Create a new bar file" editor will open.

2. Call the bar file "ManualBuildEmployeeService" and specify the EmployeeService as the Container, click Finish:

Web Services Security
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
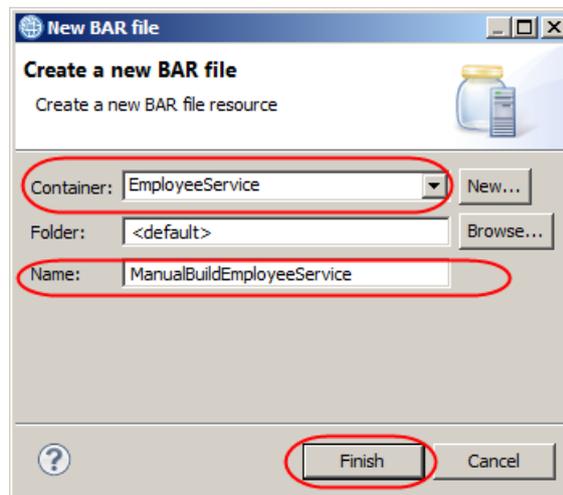
3.   Tick the check box next to "EmployeeService" and "EmployeeService_interface_and_maps", and click the "Build and Save" button to create the bar file:
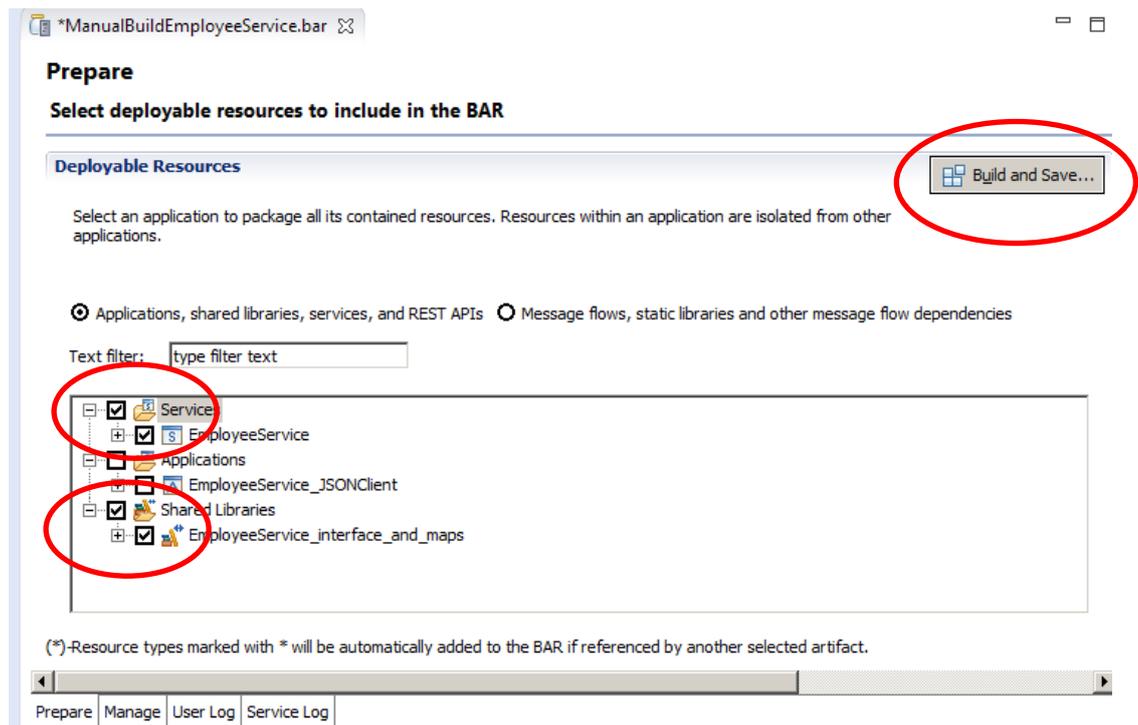
4.   Click OK on the window which says "Operation completed successfully".

5.   Click the Manage tab to see the resources built to the bar file. You will see the subflow that implements the getEmployee operation under the Resources folder.
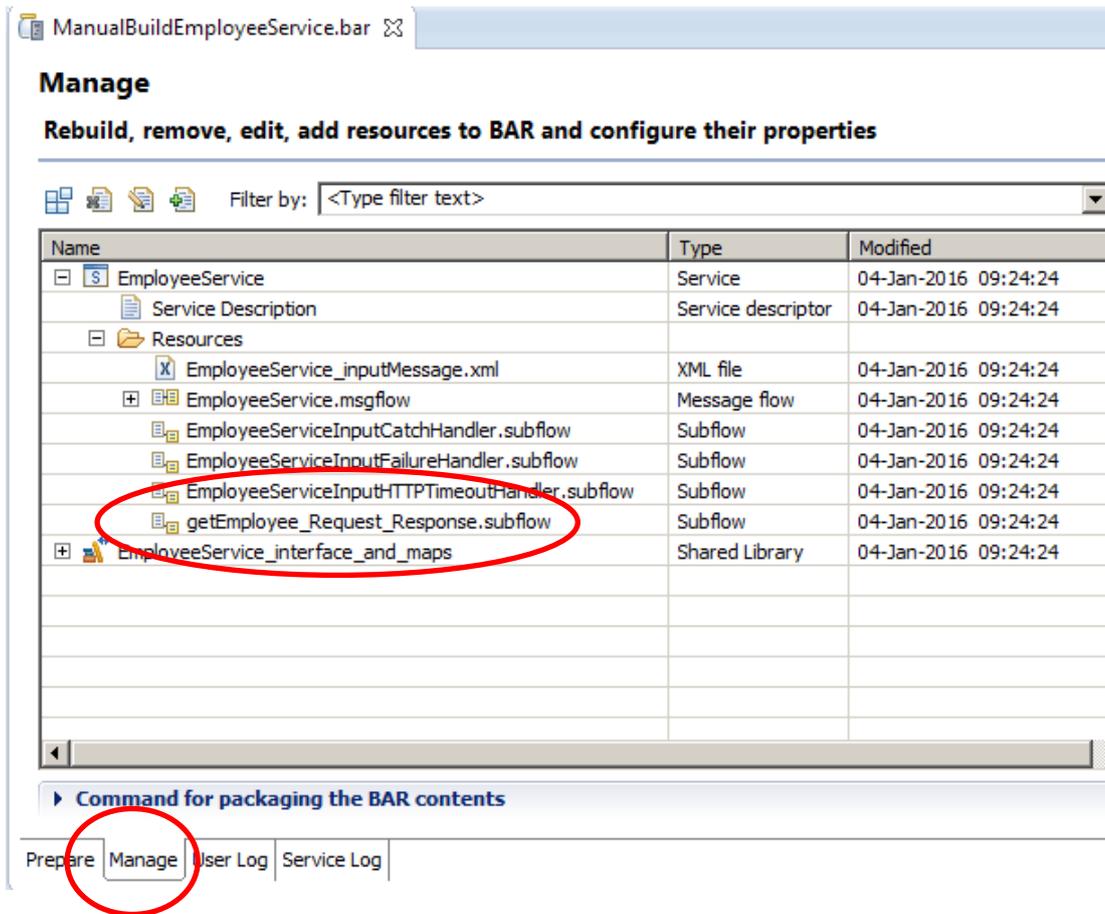
6.   Deploy the bar file ManualBuildEmployeeService to the **PROVIDER** Integration Server.

3.1.2 Note the Service Query URL for the provider

The EmployeeService is now running and can be called. However in order to call the service you need to know the URL and port that the service is running on. You can find this information from the properties tab associated with the running EmployeeService.

1. Using the Toolkit, note the "Service Query URL" port that the EmployeeService is running on: Click the EmployeeService in the Integration Nodes view and look at the properties for the running service:
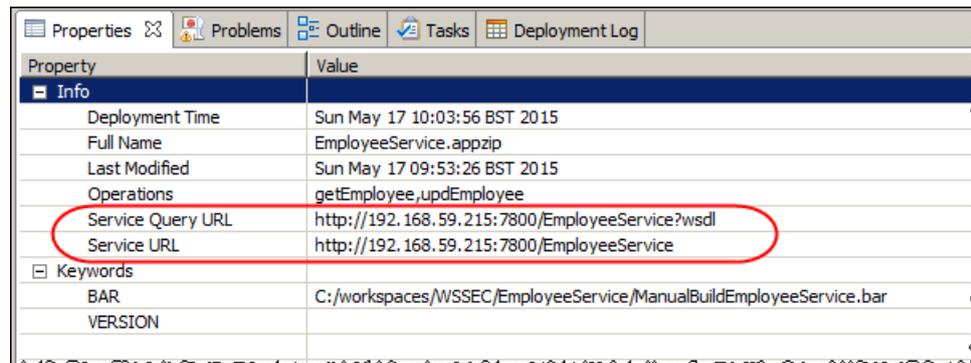
Property	Value
Info	
Deployment Time	Sun May 17 10:03:56 BST 2015
Full Name	EmployeeService.appzip
Last Modified	Sun May 17 09:53:26 BST 2015
Operations	getEmployee,updEmployee
Service Query URL	http://192.168.59.215:7800/EmployeeService?wsdl
Service URL	http://192.168.59.215:7800/EmployeeService
Keywords	
BAR	C:/workspaces/WSSEC/EmployeeService/ManualBuildEmployeeService.bar
VERSION	

This is required to ensure the requests made by EmployeeService_JSONClient are made to the correct endpoint.

Note the IP address and port number details for your environment here:

_____

You will use these details in the next section to configure the SOAP Request node used to call the EmployeeService from the JSON Client.

2. The IP address will be used in order to see communications between the Consumer and Provider application in the TCP/IP Monitor.

You will now set up the JSON Client application on the IB10NODEC.CONSUMER integration node.

3.2 Prepare the Consumer

1.   In the Integration Toolkit, expand EmployeeService_JSONClient > message flows

2.   Open the message flow **EmpServ_JSON_getEmployee.**

3.   Double click on the subflow called **getEmployee_EmployeeService** to open it in the Integration Toolkit subflow editor:

Web Services Security   Version 10.0.0.3
                    Authentication and Encryption using X.509 Certificates
                         Provided by IBM BetaWorks
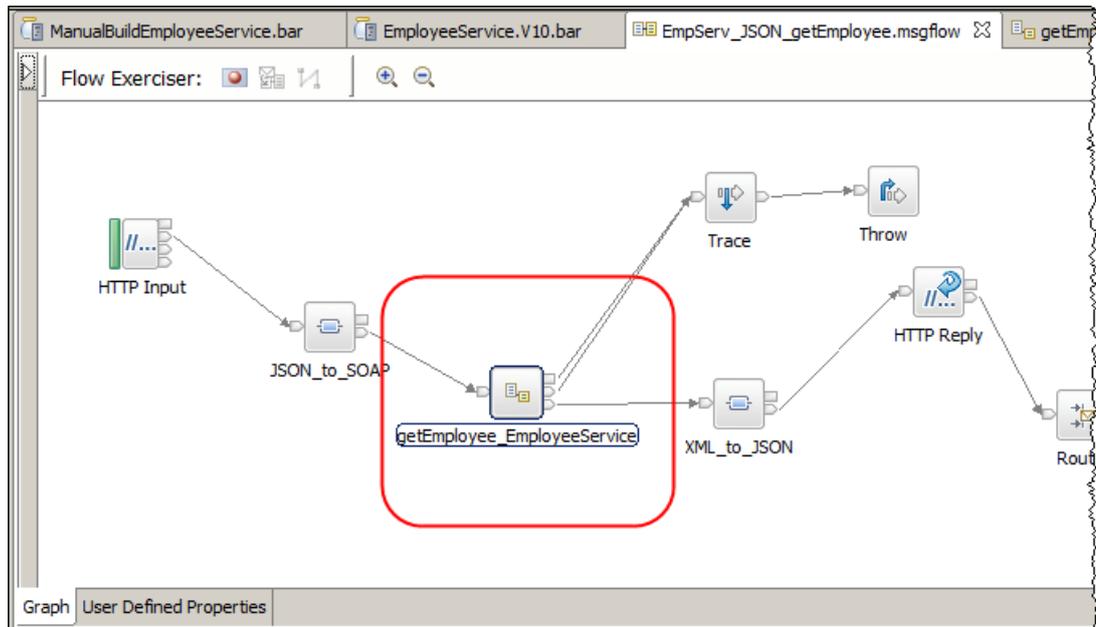
4. Highlight the Soap Request Node called "**Request**" and click the properties tab.

   On the **HTTP Transport** tab, set the port of the SOAP Request to 7802. This means that the request will be directed to the TCP/IP Monitor, which will in turn forward the request to 7800.

5. Save the changes to the subflow (Ctrl-S) and close the subflow editor.

6. Close the message flow editor (there should be no changes to the message flow).

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

3.2.1 Create JSON Client bar file

You will now create and build a bar file to deploy the JSON Client application. This will be used to configure the authentication and encryption scenarios later in this guide.

1. In the Integration Toolkit, right click on the **EmployeeService_JSONClient** application and select "New > Bar File" from this list of options.

    The "Create a new bar file" editor will open.

    Call the bar file "**ManualBuildEmployeeService_JSONClient**".

    Save the new bar file as part of the EmployeeService_JSONClient application:
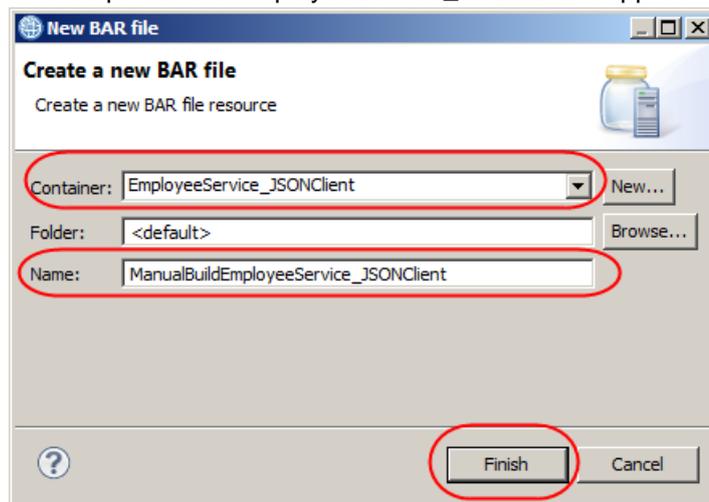
2. In the Prepare tab, select the both the "**EmployeeService_JSONClient**" and **EmployeeService_interface_and_maps**. Click the "Build and Save" button:

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
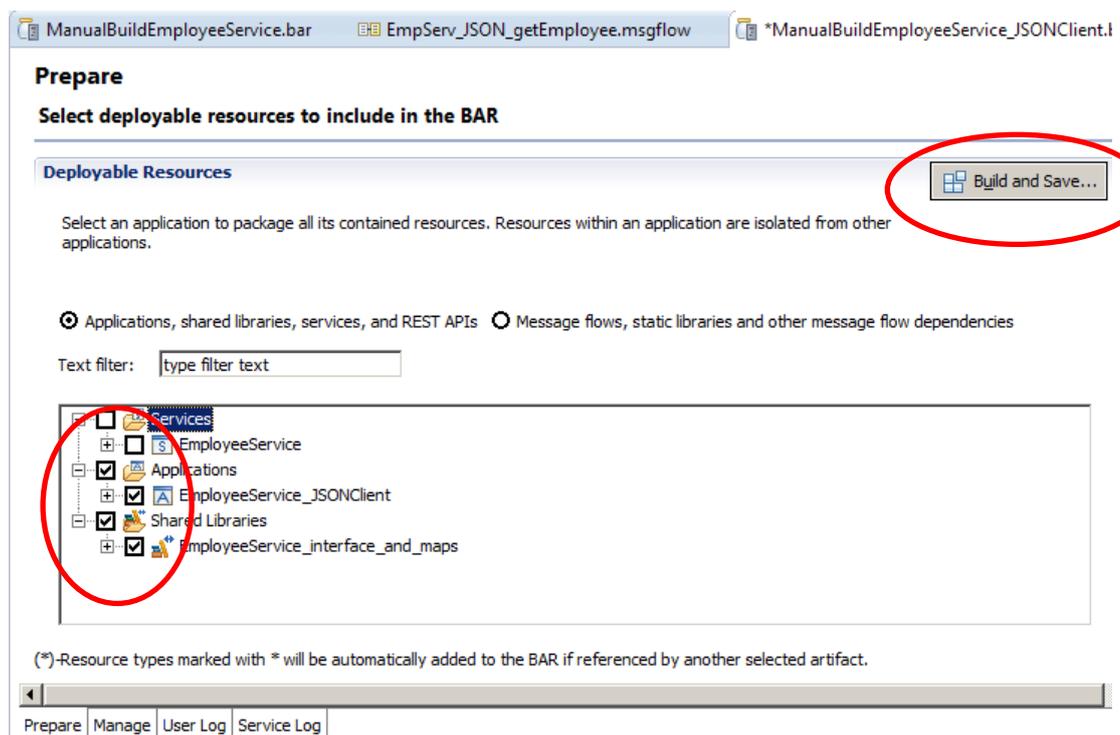
3.   Click the Manage tab to check the resources have been included in the bar file:
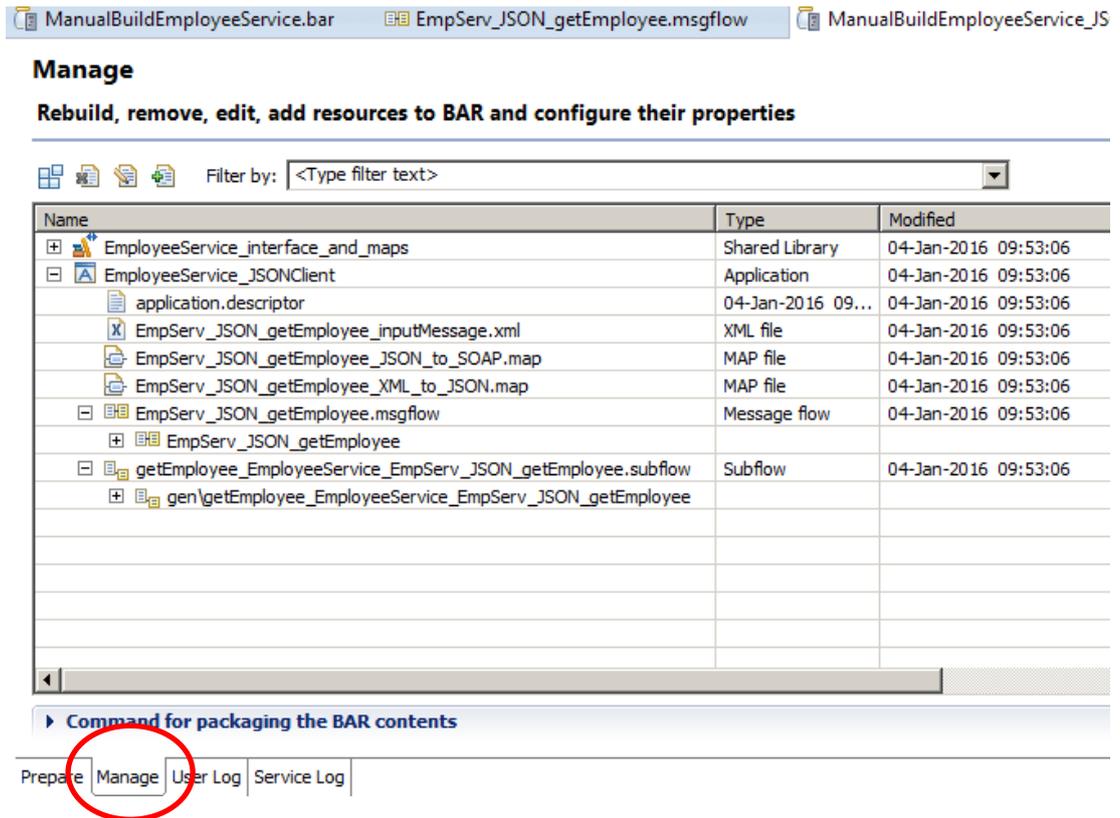
4.   Close the bar file editor.

5.   Deploy the bar file **ManualBuildEmployeeService_JSONClient** to the CONSUMER Integration Server.

4. Set up IIB to use a Key and Trust store

For both the Authentication and Encryption scenarios, X.509 certificates will be used in the configuration. You will now configure the Provider and Consumer IIB nodes to use a predefined key store and trust store.

Note: at the end of this section, the nodes will be configured to be capable of using a trust store and key store. However the data in the key store and trust store will not be used until a Policy Set and corresponding Policy Set Binding have been configured for the provider and consumer.

1. In an IIB V10 Integration console, change directory to

   c:\student10\WSSecurity\commands

   Run the command:

   01SetupPKI.cmd

2. Accept the defaults when prompted.

3. The script configures the PKI infrastructure used by the two nodes in this lab guide. For the changes to take effect the nodes are restarted by the script.

   Wait until you see "Script 01SetupPKI Complete".

5. Scenario: Test the service (no active security)

The purpose of this section is to ensure that the JSON Client application can communicate with the EmployeeService provider without any security or encryption defined.

1. Open the EmpServ_JSON_getEmployee message flow and click the record icon (red button).

2. When prompted, select the **CONSUMER** Integration Server on **IB10NODEC**, and deploy the application.

3. Click the send message icon. Highlight the Input message "Get employee 000010" and click the send button:

Web Services Security   Version 10.0.0.3
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
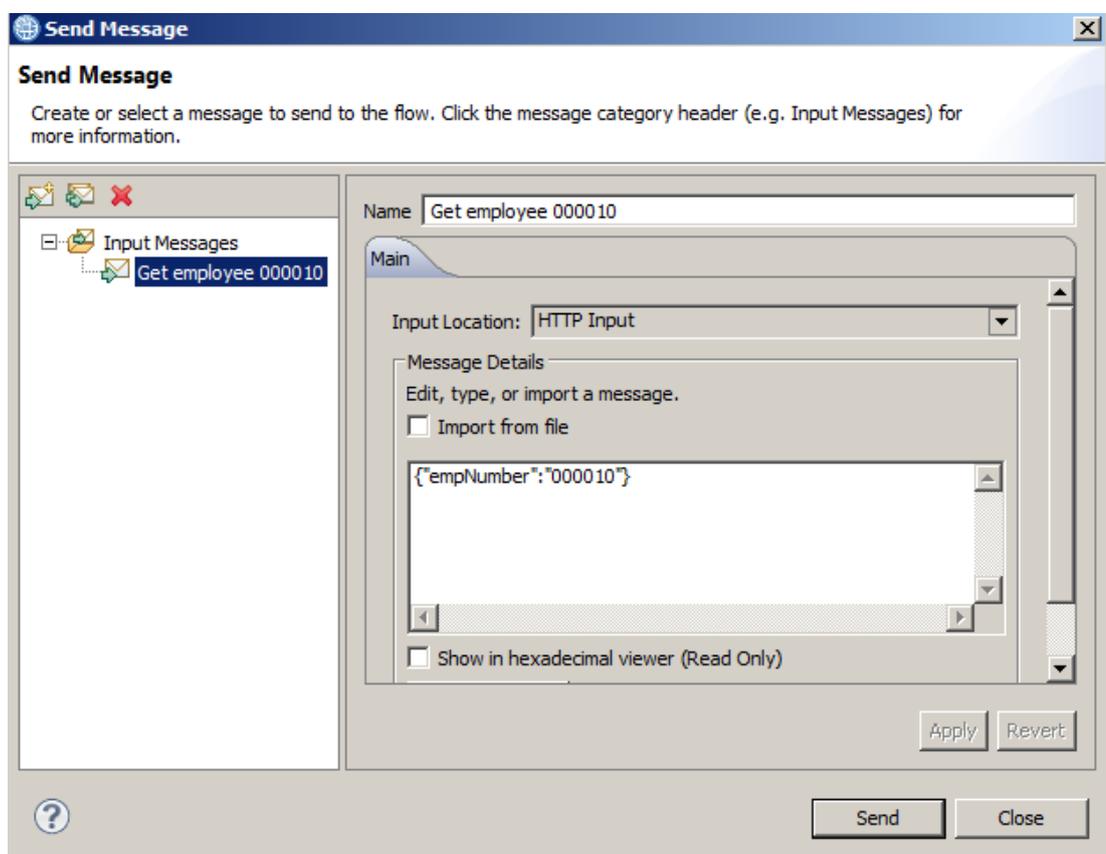
4.   The EmployeeService will respond with the details for "Christine Haas":
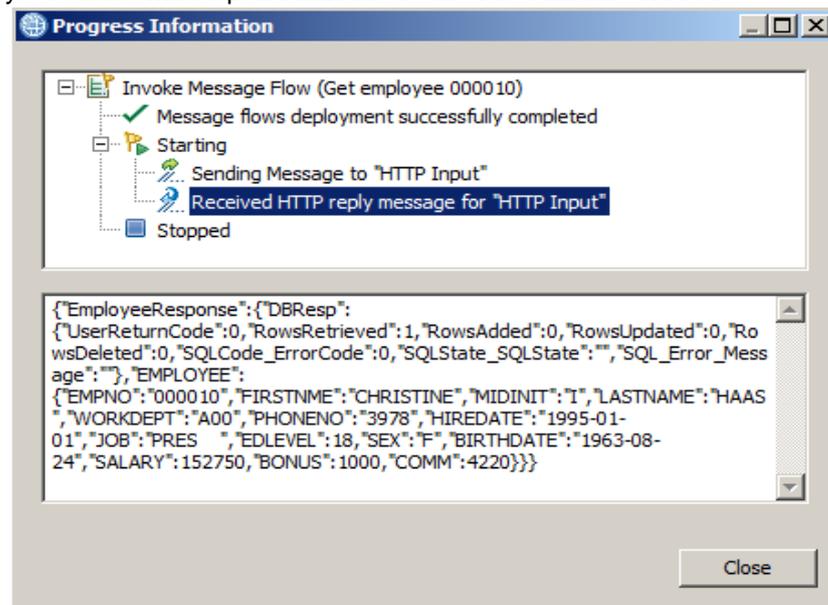
5.   Close the Progress Information window.

6.   Switch to the TCP/IP Monitor in the IIB Toolkit.

You will see the two panels (send and receive) have been populated with the data that has been sent and received.

Both messages are in clear text, and you will be able to see the requested key (000010) that was sent (in the left pane), and the data that was returned (right pane). Note the data in the section above shows the data exchange between the EmployeeService_JSONClient **request** and the **response** provider by EmployeeService. Both request and response messages appear in clear text and provide no authentication or privacy.

To view more data, expand the TCP/IP Monitor page (double-click the tab). You can also move the divider bar between the input and output messages.
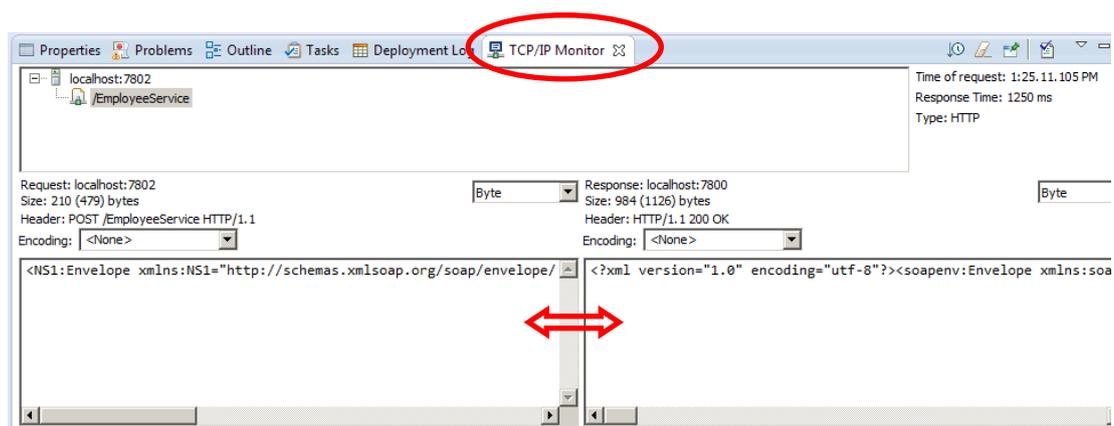
7.   In the IIB Toolkit, click the red cross to stop the Flow Exercise on the JSON Client.

6. Policy sets and bindings in IIB

The remaining sections of this lab guide show how IIB support of web services authentication and encryption, can be used to secure the message exchange between the JSON client and the EmployeeService.

This will involve editing the bar files prefix "ManualBuild" that you created earlier. The current implementation of the Flow Exerciser does not permit being able to run a manually built bar file (with bar file overrides). As a result the rest of the lab guide will use SOAPUI to send a message into the JSON client.

In IIB, policy sets and bindings define and configure WS-Security requirements for SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

A policy set is a container for the WS-Security policy types.

A policy set binding is associated with a policy set and contains information that is environment specific, such as information about keys. In each of the testing scenarios throughout this lab guide, a policy set and corresponding binding file is provided to show the security feature.

In the case of the scenarios used by this lab guide, activity on the following nodes will be affected by the rules in the policy set and binding:

1. For the **EmployeeService_JSONClient**, the SOAPRequest node

2. For the **EmployeeService**, the SOAPInput and SOAPReply nodes

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

7. Scenario: Authentication using WS-Security

In this scenario, web services authentication using X.509 certificates will be configured using a Policy Set and Policy Set bindings file provided for this lab guide.

The configuration has been automated using previously exported configuration files to save you time. However you will have the chance to optionally review the configuration required to implement Authentication using X.509 certificates.

7.1 Policy set and bindings for Authentication

1. In an **Integration console,** change directory to:

> `c:\student10\WSSecurity\commands`

Run the command:

> `02SetupAuthPSB.cmd`

Accept the defaults when prompted.

2. When the script completes you will see "**02SetupAuthPSB command complete.**"

The Policy set and Policy set bindings have now been defined as configurable services in the IIB nodes. In the next section you will configure both the EmployeeService (service) and EmployeeService_JSONClient (application) to use the respective policy set and policy set bindings that you added in this section.

7.2 Activating a Policy Set and Policy Set Binding configuration

To configure an application or service to use a policy set and corresponding policy set binding file, an IIB administrator specifies the names of the policy sets and bindings in the Bar file.

You will now configure EmployeeService and EmployeeService_JSONClient to use the policy set containing authentication configuration.

It is possible to specify a policy set and bindings rule either at the message flow level or the individual SOAP node level. In the following section you will specify the policy set and binding at the message flow level.

This is a one-time only configuration as the same policy set and policy set binding file is used for each scenario. Note if you decide to modify either the application or the service you may need to redo this configuration in the bar file.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

7.2.1 Activate the Policy Set and Binding on the Consumer

1. Using Integration Toolkit, edit the bar file you created earlier called (double click to open).

   ManualBuildEmployeeService_JSONClient.bar

2. Click the Manage tab and expand the application until you see the message flow, click on the message flow name, EmpServ_JSON_getEmployee.

3. In the properties tab for the message flow, use the edit button to configure the Consumer Policy Set and Policy Set Bindings as follows:

4. Save the bar file.

5. Redeploy the bar file to the CONSUMER Integration Server.

Web Services Security
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

7.2.2 Activate the Policy Set and Binding on the Provider

1. Using Integration Toolkit, double click (to edit) the **ManualBuildEmployeeService.bar** file.

2. Click the Manage tab and expand the EmployeeService until you see the message flow **EmployeeService.msgflow**. Click on the message flow name.

3. In the properties tab for the message flow, configure the consumer policy and policy set bindings as follows (the edit button will provide the valid names):

(NB take care not to specify the Provider policy sets on the Consumer fields)

4. Save the bar file.

5. Close the bar file editor.

6. Redeploy the bar file to the PROVIDER Integration Server.

7.3 Test the service: Authentication defined and active

The purpose of this section is to show that the JSON Client can communicate with the EmployeeService with authentication defined, with each environment knowing authentication details for the other.

7.3.1 Find the HTTPConnector port number on the CONSUMER server

In order to invoke the JSON Client that we are using as a Consumer, we need to identify the port number used by the HTTP Connector on the server CONSUMER (on the IIB node IB10NODEC).

1. In an Integration Console window, type the following command:

    ```
    mqsireportproperties IB10NODEC
                          -e CONSUMER
                          -o HTTPConnector
                          -a
    ```

2. The response of the command will be similar to the following (the port number in your environment may be different – results are truncated)

    ```
    HTTPConnector
    uuid='HTTPConnector'
    userTraceLevel='none'
    traceLevel='none'
    userTraceFilter='none'
    traceFilter='none'
    port='7801'
    address=''
    maxPostSize=''
    acceptCount=''
    compressableMimeTypes=''
    compression=''
    connectionLinger=''
    connectionTimeout=''
    ```

    Note the value of the port number   (*probably 7801 in the workshop system*   )

    You will need this value when you use SOAPUI to start the JSON Client.

Web Services Security
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

7.3.2 Reset the TCP/IP Monitor

1. In the TCP/IP Monitor window, click the Clear button (highlighted below).
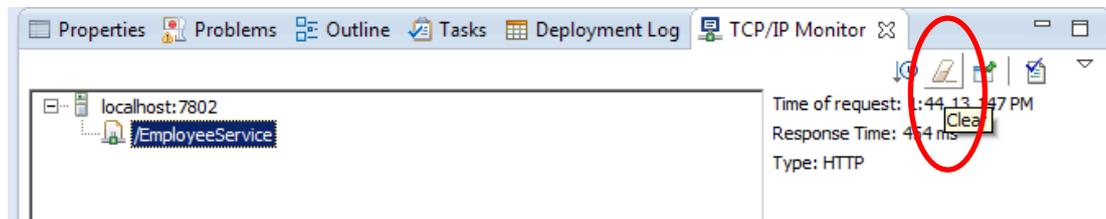
This will clear the messages written during the current live capture.

7.3.3 Send a message using SOAPUI

1. Open SOAPUI from the Windows Start menu and expand :

   - EmployeeService_JSONClient – Sync
     - http://localhost:7801   (since the target endpoint is most likely to be 7801)
       - getEmployee
         - Employee 000010

Make sure the port on the "Endpoint" window matches the port in your environment (edit it if it doesn't) and click the green arrow to send the message:

Web Services Security
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
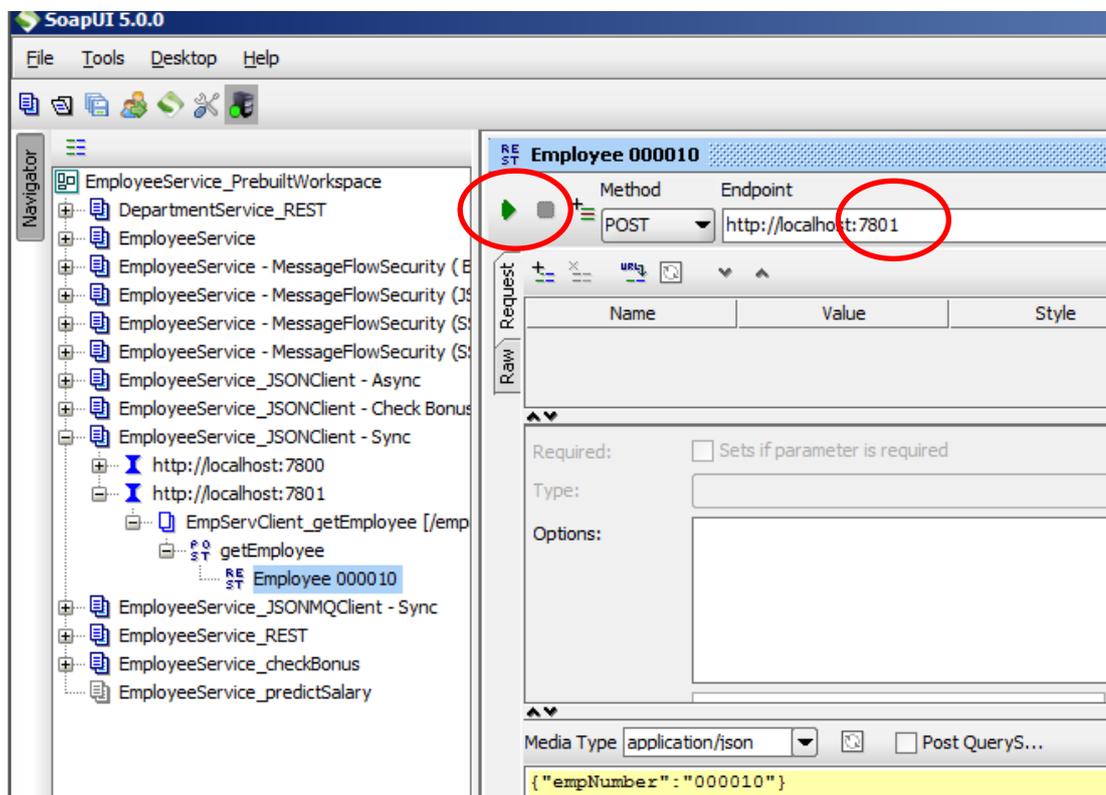
2.   After a few seconds the window will update with the response from the database:

Click the JSON tab to see the data in JSON format:

```json
{"EmployeeResponse": {
   "DBResp":      {
      "UserReturnCode": 0,
      "RowsRetrieved": 1,
      "RowsAdded": 0,
      "RowsUpdated": 0,
      "RowsDeleted": 0,
      "SQLCode_ErrorCode": 0,
      "SQLState_SQLState": "",
      "SQL_Error_Message": ""
   },
   "EMPLOYEE":      {
      "EMPNO": "000010",
      "FIRSTNME": "CHRISTINE",
      "MIDINIT": "I",
      "LASTNAME": "HAAS",
      "WORKDEPT": "A00",
      "PHONENO": "3978",
      "HIREDATE": "1995-01-01",
      "JOB": "PRES      ",
      "EDLEVEL": 18,
      "SEX": "F",
      "BIRTHDATE": "1963-08-24",
      "SALARY": 152750,
      "BONUS": 1000,
      "COMM": 4220
   }
}}
```

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

7.3.4 Verify the network traffic using TCP/IP Monitor

1. Switch to the TCP/IP Monitor view.

    The data shown on both the request and response messages will be shown.
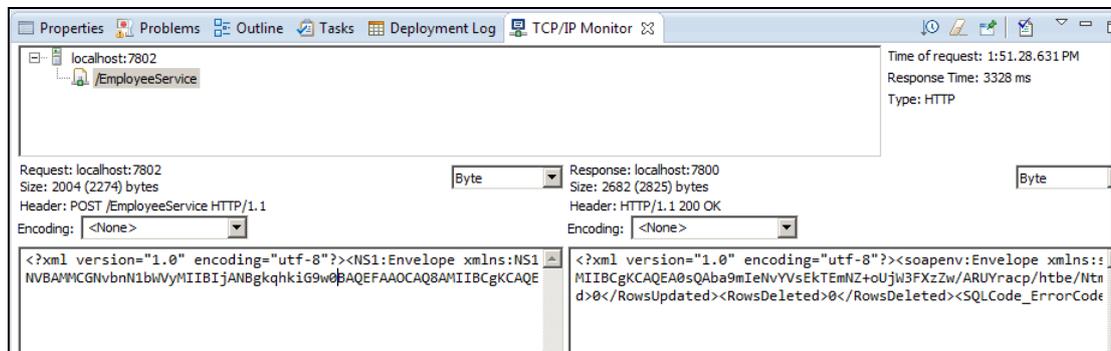
2. Ensure that the data is displayed in XML format (dropdown selection).

    Note the Soap envelope now has a "wsse:security" section with a "BinarySecurityToken", on the request header.

    The value of this token is derived from the X.509 certificate passed by the consumer and is used by the provider IIB to authenticate the request from the consumer. (The end tag of the Binary Security Token will be shown if you use the slide bar to display further parts of the message).

    IIB trusts this request and allows it to progress as the public part of the consumer's X.509 certificate is stored in the trust store used by the PROVIDER Integration Server.

    Note that the payload of the message is in clear text.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

3.   Now look at the response message in the right-hand pane. Ensure the format is XML.

This message is the response back to the consumer from the provider.

Note the soapenv:Header also contains a (different) BinarySecurityToken derived from the provider's X.509 certificate. On the provider machine this too has been used to authenticate the response from the provider by verifying the token with the contents of the provider's trust store:

```
Response: localhost:7800
Size: 2682 (2825) bytes
Header: HTTP/1.1 200 OK
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oas
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-ws
fW/ONZm1JaWUD0rdMmByF6R97C/OWo4eL70QLOu2Rxv68E/8y7gbTrrl7Qr/8OsIad3vgX3+EPbzVcF8N+o0lPhPQIDAQABMA0GC
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <io2:getEmployeeResponse xmlns:io="http://hrdb/iibadmin" xmlns:io2="http://EmployeeService">
      <io:EmployeeResponse>
        <DBResp>
          <UserReturnCode>0</UserReturnCode>
          <RowsRetrieved>1</RowsRetrieved>
          <RowsAdded>0</RowsAdded>
          <RowsUpdated>0</RowsUpdated>
          <RowsDeleted>0</RowsDeleted>
          <SQLCode_ErrorCode>0</SQLCode_ErrorCode>
          <SQLState_SQLState/>
          <SQL_Error_Message/>
        </DBResp>
        <out:EMPLOYEE xmlns:out="http://hrdb/iibadmin">
          <EMPNO>000010</EMPNO>
          <FIRSTNME>CHRISTINE</FIRSTNME>
          <MIDINIT>I</MIDINIT>
          <LASTNAME>HAAS</LASTNAME>
          <WORKDEPT>A00</WORKDEPT>
          <PHONENO>3978</PHONENO>
          <HIREDATE>1995-01-01</HIREDATE>
          <JOB>PRES    </JOB>
          <EDLEVEL>18</EDLEVEL>
          <SEX>F</SEX>
          <BIRTHDATE>1963-08-24</BIRTHDATE>
          <SALARY>152750</SALARY>
          <BONUS>1000</BONUS>
          <COMM>4220</COMM>
        </out:EMPLOYEE>
      </io:EmployeeResponse>
    </io2:getEmployeeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The consumer JSON Client application and the provider EmployeeService now authenticate when communicating. The remainder of this lab guide will guide you through two encryption scenarios.

8. Message Level Encryption

The previous scenario showed you how to authenticate web services requests (and responses) using X.509 certificates.

As you can see from the captured data in the TCP/IP Monitor, the request message payload is "in the clear". The data on the network link can be viewed, for both the request and response messages.

An option to secure the exchange of data between two web services is to encrypt the data. With IBM Integration Bus this can be done using whole or partial message encryption. The following sections show how each of these scenarios can be used to secure the data being exchanged between two web services.

The configuration required to use encryption will require changes to the policy sets and bindings configuration used on both the CONSUMER and PROVIDER Integration Servers. The bar file configuration performed in the previous section will continue to be used so no changes will be required to the bar file in both scenarios.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

8.1 Prepare the PKI for encryption

For *authentication,* IIB requires the public/private key that is used for authentication purposes to exist in the keystore used by the Integration Server.

For example, in the case where the provider sends a response back to the consumer, the key used by the provider to authenticate with the consumer must exist in the provider **keystore**. When the key is passed over to the consumer machine, the consumer integration server checks for the existence of the public portion of the key in the **truststore** used by the consumer integration server. If the key is trusted then authentication succeeds and the request flows to the consumer.

For **Encryption,** the requirement for the placement of keys is slightly different. When encrypting data, the public key of the party which will be decrypting the data is required in the **keystore** of the party encrypting the data.

For example, in the case where the provider sends an encrypted response back to the consumer, the consumer's public key needs to exist in the keystore used by the provider.

The following scripts will prepare the PKI environment you will use for both encryption scenarios in this lab guide.

1. In an Integration console, change directory to:

    c:\student10\WSSecurity\commands

    Run the command:

    03SetupPKIForEncryption.cmd

    Accept the defaults when prompted.

2. When the script completes you will see "**03SetupPKIForEncryption command complete.**"

You now have the required X.509 certificates in place to perform the encryption scenarios.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

8.2 Whole Message Encryption

Full message encryption is possible on the request and response messages between a client and a web services provider. You will now configure your environment to **encrypt** both the request from the JSON Client and the response from the EmployeeService.

Note: encrypting the whole message exchange between the consumer and provider, although secure, is the more costly in terms of the volume of data being exchanged and the processing power/time required to encrypt the data.

8.2.1 Policy Set and Bindings for Whole Message Encryption

1.   In an Integration console, navigate to:

c:\student10\WSSecurity\commands.

Run the command:

04PSBSetupEncryptWholeMsg.cmd

Accept the defaults when prompted.

2.   When the script completes you will see "**04PSBSetupEncryptWholeMsg command complete.**"

3.   Note a restart of the node is required in order for the changes to take effect on IIB nodes. This is done automatically by the script.

The Policy set and Policy set bindings have now been defined as configurable services in the IIB nodes. In the next section you will configure both the EmployeeService (service) and EmployeeService_JSONClient (application) to use the respective policy set and policy set bindings that you added in this section.

8.2.2 Reset the TCP/IP Monitor

1.   In the TCP/IP Monitor view, click the Clear button.
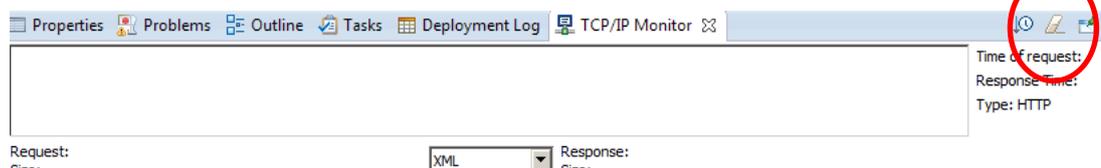
This will clear the messages written during the current live capture.

2.   If not already started, start the IIB Event Log monitor from the start menu.

8.2.3 Send a message using SOAPUI

1. In the SOAPUI window click the green arrow to send the message:
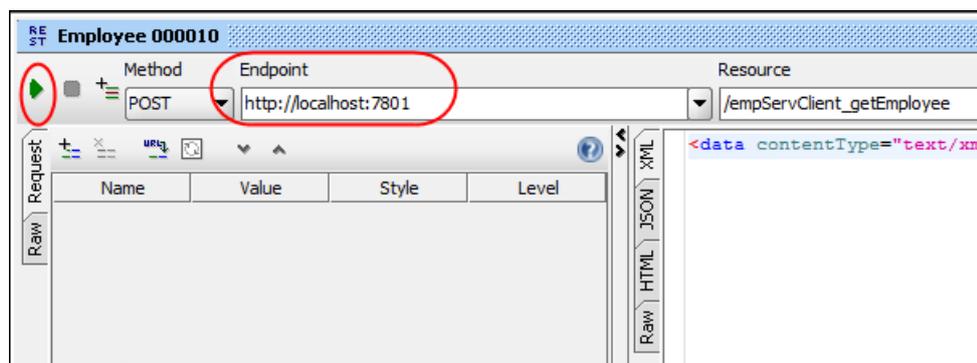
The request will fail.

2. In the IIB Event Log monitor (and in the SOAPUI output) you will see a number of Error messages. The crux of the Failure is an Illegal Key size error. Look for a Java security error message similar to the following:

```
org.apache.axis2.AxisFault: CWWSS5612E: Encrypting the data
produced the following exception: Illegal key size or default
parameters: java.security.InvalidKeyException: Illegal key
size or default parameters
```

When creating the "self-signed" certificates provided with this lab guide, a (default) key size of 1024 was used for both consumercert and providercert.

Due to export restrictions the IBM JDK's ship with a set of restricted policy files that limit the size of the cryptographic keys that are supported.

It is possible to remove this restriction by downloading the unrestricted policy files from the web. The policy files have been downloaded for your convenience and a scripts are provided that will replace the restricted policy files with "Unrestricted" policy files in the consumer and provider environments.

If you are using the lab in your own environment, you can obtain the unrestricted policy files from the URL: https://www-01.ibm.com/marketing/iwm/iwm/web/preLogin.do?source=jcesdk

8.2.4 Replace IBM JDK restricted policy files

1. In an IIB V10 Integration console, navigate to **c:\student10\WSSecurity\commands**, run the command:

   05SetJCEForEncryption.cmd

2. Accept the defaults when prompted.

   For the Fix Pack number, provide the appropriate fixpack (eg. 3).

3. The script replaces the default IBM JDK **restricted** policy files with **unrestricted** policy files on the provider and consumer IIB nodes.

   This affects the whole of this installation not just a specific IIB node.

   The IB10NODEP and IB10NODEC nodes are stopped and restarted to bring the changes into effect.

4. When the script completes you will see "**05SetJCEForEncryption Complete, ready to run scenario again.**"

8.2.5 Reset the TCP/IP Monitor

1. In the TCP/IP Monitor view, click the Clear button.

   This will clear the messages written during the current live capture.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
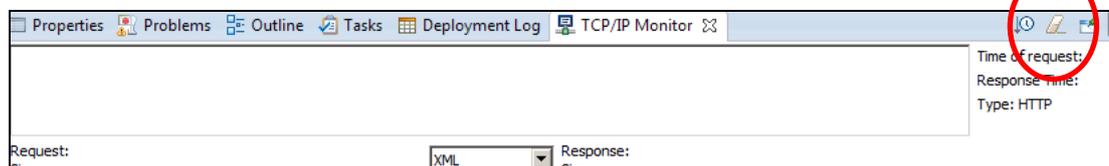
8.2.6 Send a message using SOAPUI

1. In SOAPUI, click the green arrow to send the message again. The port should be the same port as used earlier.
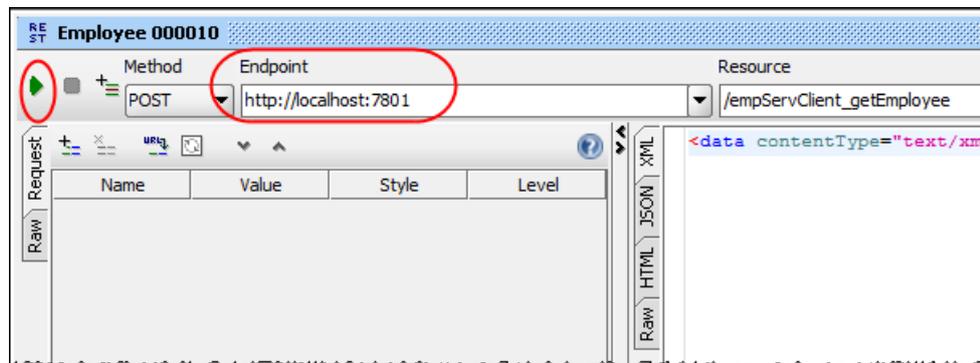
2. The request should now respond successfully (click the "JSON" tab to see the response data).

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

8.2.7 Verify the Encryption in TCP/IP Monitor

1.   Switch to the TCP/IP Monitor.

Make sure the format is XML. You will see the SOAP body with the encrypted request (wrapped in "CipherValue"):

```
Request: localhost:7802
Size: 3675 (3945) bytes
Header: POST /EmployeeService HTTP/1.1
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<NS1:Envelope xmlns:NS1="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-me
l9aKoGA8J/gn7stQfD/GKsSm2aQcz26gnJ88xaj/q5Xlhee9MYEEsRbyVYPrlIgSmgulKA4EhYM4KfA7SXctIHKUwIDAQABMA0GCSqGSIb3DQ
      <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
        <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="http://www.w3.org/200
        </enc:EncryptionMethod>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-me
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <enc:CipherData>
          <enc:CipherValue>PIJ1V821qJAp82PAemrXjO0wAIhNF9piEMZD3iDikr9cYW1MKQPgWJJvf4ePKZuNNcTUhXrYqt7+/l8yBE
        </enc:CipherData>
        <enc:ReferenceList>
          <enc:DataReference URI="#wssecurity_encryption_id_21"/>
        </enc:ReferenceList>
      </enc:EncryptedKey>
    </wsse:Security>
  </soapenv:Header>
  <NS1:Body>
    <enc:EncryptedData Id="wssecurity_encryption_id_21" Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns
      <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
      <enc:CipherData>
        <enc:CipherValue>M71ZfuP0xW6M+C2z+tVgzbrvGm0RO5ouxvhVNXWb6zakZBhUWWNYsFTCG25qdxYndhlQKFM1u3TfFH01P4Y4
      </enc:CipherData>
    </enc:EncryptedData>
  </NS1:Body>
</NS1:Envelope>
```

This is the encrypted request sent from the Consumer machine.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

2. In the right pane, you will see the response message which will show "**HTTP/1.1 200 OK**".

As with the request message, the SOAP message will show the encrypted response (also wrapped in "CipherValue"):

```
Response: localhost:7800
Size: 4597 (4740) bytes
Header: HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-s
fW/ONZm1JaWUD0rdMmByF6R97C/OWo4eL70QLOu2Rxv68E/8y7gbTrrl7Qr/8OsIad3vgX3+EPbzVcF8N+o0lPhPQIDAQABMA0GCSqG
      <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
        <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="http://www.w3.o
        </enc:EncryptionMethod>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-s
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <enc:CipherData>
          <enc:CipherValue>of/EPlLq/ByyJstIeU+DJtzdxovRXjes+1ZMFyLRA2NgjRcuaXUZmIWPxk5VCjEr3m6FzdX0YxFX
        </enc:CipherData>
        <enc:ReferenceList>
          <enc:DataReference URI="#wssecurity_encryption_id_21"/>
        </enc:ReferenceList>
      </enc:EncryptedKey>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <enc:EncryptedData Id="wssecurity_encryption_id_21" Type="http://www.w3.org/2001/04/xmlenc#Content"
      <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
      <enc:CipherData>
        <enc:CipherValue>hlkSbTf4K6puXkkIr7w+GyujClpszlmt9TRbb+KbJJCDvIBkzWq5+7A0LgOuWcKevrGasFZaKcBM6A
AVzLL2Ira8t4mkwPtcV8LIsMf3QjREVhBCGv0Z66xKPZkurhgJ8LTAKm9vs84k3E5KaP6Ra4A2PKaVlX/uw7LWS77NYN4ljJwU5NiUU
      </enc:CipherData>
    </enc:EncryptedData>
  </soapenv:Body>
</soapenv:Envelope>
```

8.3 Partial Message encryption

An alternative to encrypting the whole request and response message is to encrypt only a part of the request or response message. In the next section you will see how to encrypt a sensitive part of the response sent back to the consumer.

Partial message encryption is possible on the request and response conversation between two web services. The EmployeeService provides sensitive information back to the requestor. In this next section you will see how the "SALARY" portion of the message can be encrypted leaving the rest of the message untouched.

Note: *encrypting only sensitive parts of the message exchange between the consumer and provider can provide benefits in terms of less volume of data being exchanged and less processing power/time required to encrypt the data.*

Partial message encryption is configured using a policy set and policy set binding. The configuration of these has been automated for your convenience using command line scripts. You will have a chance to review the policy set and policy set binding to see what is required to encrypt part of the response data.

8.3.1 Policy Set and Bindings for Whole Message Encryption

1.   In an **Integration console,** navigate to:

c:\student10\WSSecurity\commands.

Run the command:

06SetupPSBEncryptPartialMsg.cmd

Accept the defaults when prompted.

2.   When the script completes you will see "**06SetupPSBEncryptPartialMsg command complete.**"

The PKI, policy set and policy set binding configuration is now complete to allow for the SALARY field in the response message to be encrypted leaving the rest of the message in clear text. You will now test the partial message encryption scenario.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

8.3.2 Reset the TCP/IP Monitor

1. In the TCP/IP Monitor window, click the Clear button.
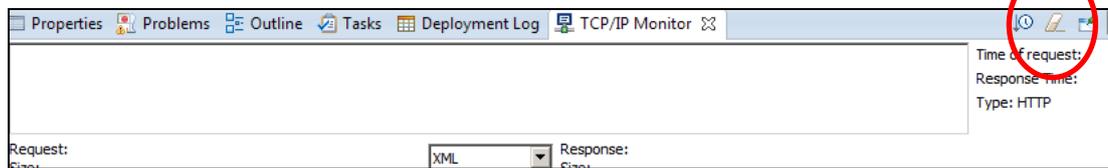
This will clear the messages written during the current live capture.

8.3.3 Send a message using SOAPUI

1. In SOAPUI, click the green arrow to send the message again.
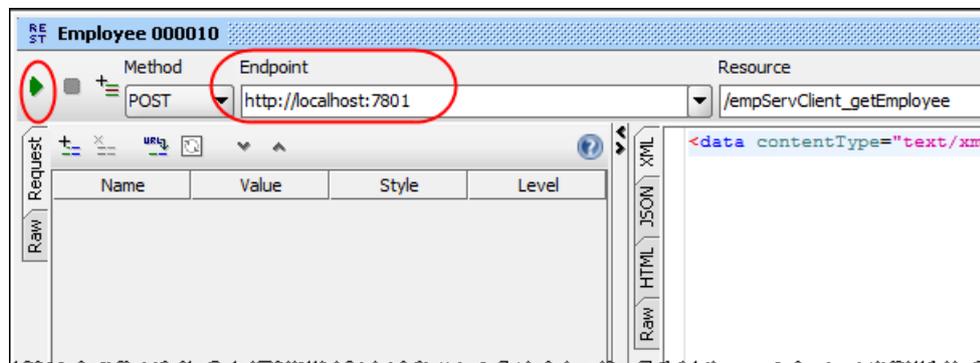
2. As before, the request should respond successfully (click the "JSON" tab to see the response data).

8.3.4 Verify the Encryption in the TCP/IP Monitor

1.   Switch to the TCP/IP Monitor. Look at the data in the Request pane. Select the XML format.

The header still has the security authentication, however the request is now back to being "in the clear text. In particular, you will see the employeeNumber element, and the value of 000010.

```
Request: localhost:7802                                                    XML  ▼
Size: 2004 (2274) bytes
Header: POST /EmployeeService HTTP/1.1

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<NS1:Envelope xmlns:NS1="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/:
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasi:
l9aKoGA8J/gn7stQfD/GKsSm2aQcz26gnJ88xaj/q5Xlhee9MYEEsRbyVYPrlIgSmgulKA4EhYM4KfA7SXctIHKUw.
    </wsse:Security>
  </soapenv:Header>
<NS1:Body>
  <io:getEmployee xmlns:io="http://EmployeeService">
    <employeeNumber>000010</employeeNumber>
  </io:getEmployee>
</NS1:Body>
/NS1:Envelope>
```

2.   Now look at the Response pane. First, you will see that the Provider responded with an
       HTTP 200 code.
       You will see the authentication information at the top of the response message.

```
Response: localhost:7800                                                    XML ▼
Size: 4328 (4471) bytes
Header: HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/:
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasi:
fW/ONZm1JaWUD0rdMmByF6R97C/OWo4eL70QLOu2Rxv68E/8y7gbTrrl7Qr/8OsIad3vgX3+EPbzVcF8N+o0lPhPQ1
      <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
        <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p":
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns:ds="h1
        </enc:EncryptionMethod>
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="http://docs.oasis-open.org/wss/2004/01/oasi:
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <enc:CipherData>
          <enc:CipherValue>jGCOBlFMkXFzfQX3adCAsitJUlo/KzOboJSWgzJ5VnTf8YX9X9fK7YsK0rMz5/)
        </enc:CipherData>
        <enc:ReferenceList>
          <enc:DataReference URI="#wssecurity_encryption_id_21"/>
```

Further down the response message, you will see the SOAP Body. You will see that most of
the message is in clear text. However, the SALARY element (positioned between
BIRTHDATE and BONUS) no longer appears as a readable field. The configuration of the
XPATH statement in the Policy Set used by the message flow has encrypted (just) the
SALARY field:

```
<soapenv:Body>
  <io2:getEmployeeResponse xmlns:io="http://hrdb/iibadmin" xmlns:io2="http://EmployeeSe1
    <io:EmployeeResponse>
      <DBResp>
        <UserReturnCode>0</UserReturnCode>
        <RowsRetrieved>1</RowsRetrieved>
        <RowsAdded>0</RowsAdded>
        <RowsUpdated>0</RowsUpdated>
        <RowsDeleted>0</RowsDeleted>
        <SQLCode_ErrorCode>0</SQLCode_ErrorCode>
        <SQLState_SQLState/>
        <SQL_Error_Message/>
      </DBResp>
      <out:EMPLOYEE xmlns:out="http://hrdb/iibadmin">
        <EMPNO>000010</EMPNO>
        <FIRSTNME>CHRISTINE</FIRSTNME>
        <MIDINIT>I</MIDINIT>
        <LASTNAME>HAAS</LASTNAME>
        <WORKDEPT>A00</WORKDEPT>
        <PHONENO>3978</PHONENO>
        <HIREDATE>1995-01-01</HIREDATE>
        <JOB>PRES     </JOB>
        <EDLEVEL>18</EDLEVEL>
        <SEX>F</SEX>
        <BIRTHDATE>1963-08-24</BIRTHDATE>
        <enc:EncryptedData Id="wssecurity_encryption_id_21" Type="http://www.w3.org/200:
          <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc",
          <enc:CipherData>
            <enc:CipherValue>Iw1cA6SxlSd64luhoK+XBgtNuP5hIX4Rsxy0A6FKMcl7p6RAt4MkM+kTdr,
          </enc:CipherData>
        </enc:EncryptedData>
        <BONUS>1000</BONUS>
        <COMM>4220</COMM>
      </out:EMPLOYEE>
    </io:EmployeeResponse>
```

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

9. Transport Level Security (TLS)

Encryption support in WS-Security concentrates on providing encryption for the content of the SOAP message exchange between a consumer and provider. The http request from the JSON client (consumer) to the EmployeeService (provider) is a standard non secure http request. You will now secure the provider by configuring the EmployeeService to use https and change the consumer request configuration to use an authenticated TLS request.

9.1.1 Configure the provider to use https

Perform the following tasks on the PROVIDER machine:

1. In an IIB V10 Integration Console, navigate to `c:\student10\WSSecurity\commands`, run the command

    `07SetupHTTPS.cmd`.

2. Accept the defaults when prompted.

3. The script:

    a) configures the **HTTPSConnector** defined at the Integration Server level on a specific port (7848)
    b) sets the "**clientAuth**" property to "true" enabling client authentication based on X.509 certificates.
    c) stops and starts IB10NODEC for the changes to take effect.

    Make sure IB10NODEP starts ok and that the EmployeeService is running.

4. In the IIB Toolkit, expand EmployeeService and double click on the "Integration Service Description":
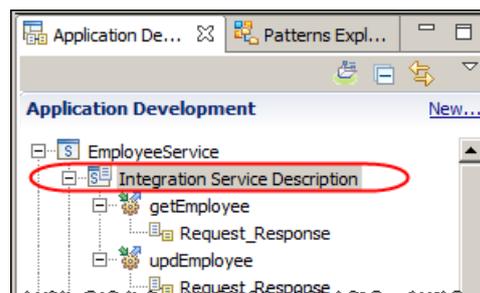
5. This will open the Service at the "Service" tab:
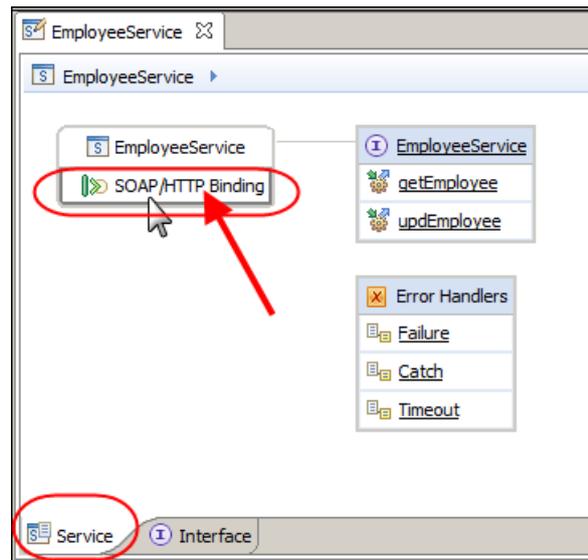
Click the service tab to show the SOAP HTTP Binding properties

6. Tick the "Use HTTPS" check box:
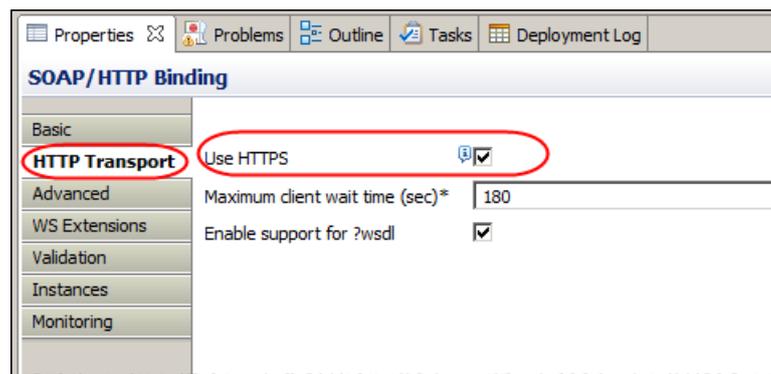
7. Save the change (Ctrl –s).

8. Close the window.

9. Open **ManualBuildEmployeeService.bar** file in the bar file editor.

10. In the bar file editor, on the Prepare tab, Click the Build and Save button to rebuild the bar file.
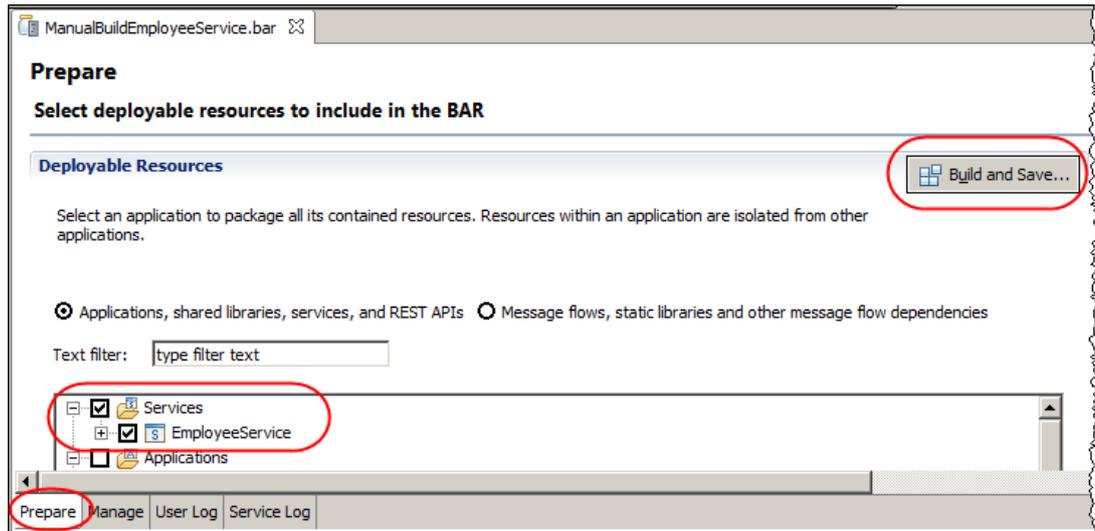
11. When the bar file has been rebuilt with the changes applied, click the Manage tab.

    In the Name column of the table, expand EmployeeService > Resources and click EmployeeService.msgflow:

                      Authentication and Encryption using X.509 Certificates
                                  Provided by IBM BetaWorks
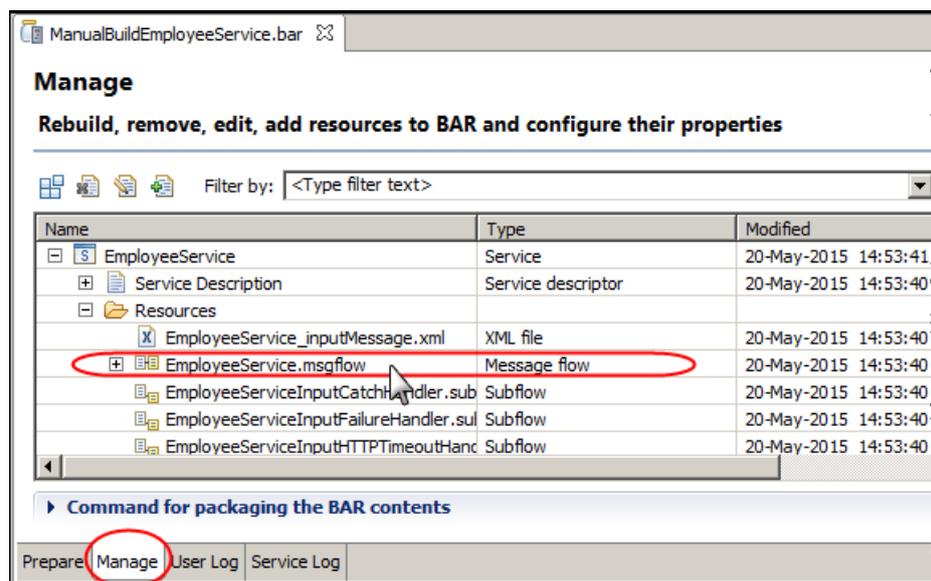
12. In the "Configure" tab, use the Edit button to set the Provider Policy Set and Provider Policy Set Bindings values to myProviderPolicySet and myProviderPolicySetBinding respectively:
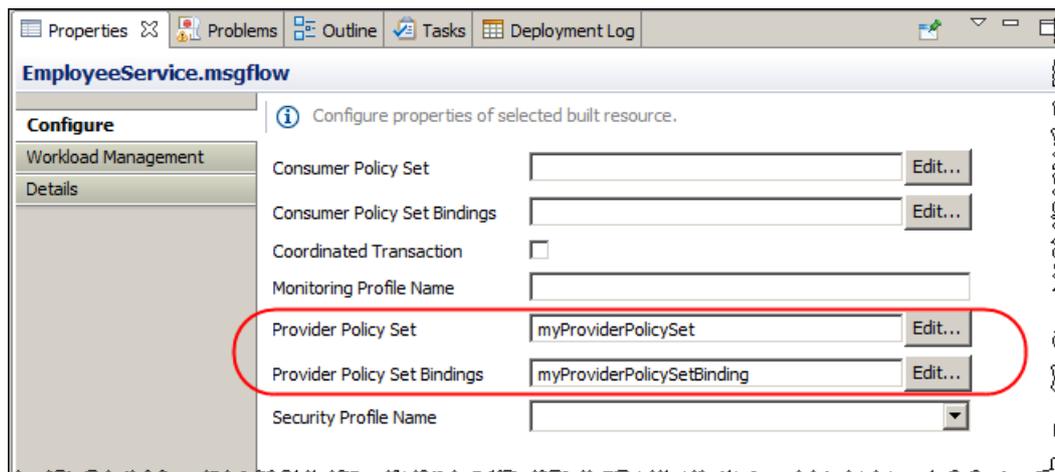
(note: do not specify anything in the Consumer Policy set and Bindings fields)

13. Save the bar file (ctrl –s)

14. Close the bar file editor.

15. Deploy the bar file to IB10NODEP.PROVIDER.

16. In the Integration Nodes window, navigate down to the EmployeeService and highlight the service by clicking on it. In the properties tab, make sure the "Service URL" is now using "https" and the port the service uses is "7848":
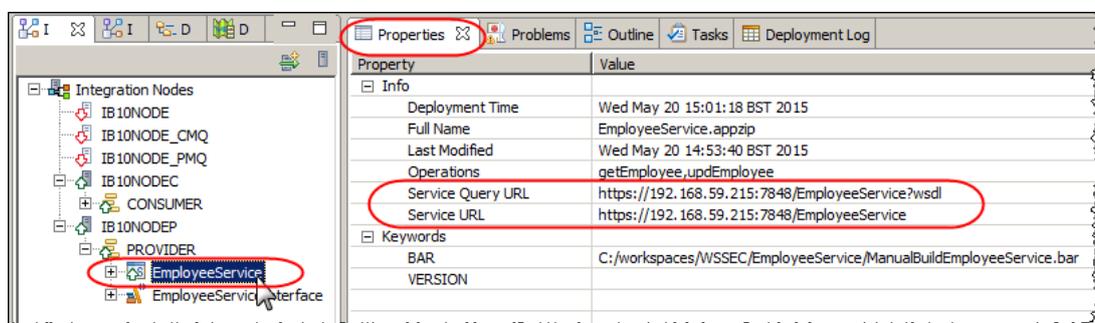
(EmployeeService is now running on an https connection).
Write down (or copy using ctrl c) the Service URL, you will use this in the next section:

Service URL :

_____

9.2 Call the Provider using HTTPS and Client authentication

There is no requirement to configure the HTTPSConnector on the consumer side. However you will need to modify the following details on the SOAP Request node in the JSON client:

a)   the Web service URL

b)   the SSL client authentication key alias

Both are on the "HTTP transport" tab on the SOAP Request Node Properties.

1.   Open the **EmpServ_JSON_getEmployee** message flow.

2.   In the message flow double click on the **getEmployee_EmployeeService** sub flow.

3.   In the sub flow, click on the Soap Request node called "Request":
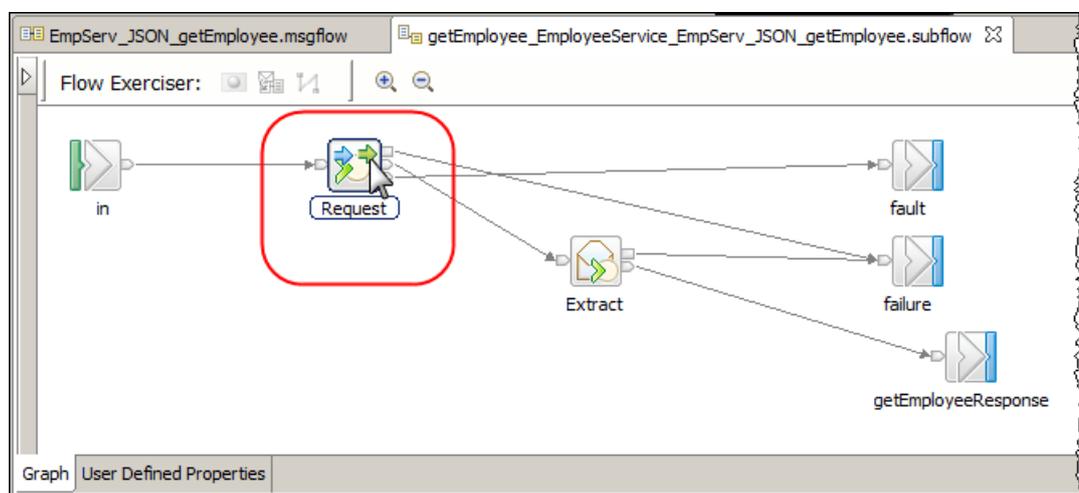
4.   In the properties tab, click on "HTTP Transport" to show the Web service URL settings.

5.   Make the following changes:

    a)   Set Web service URL to :   **https**://localhost:7802/EmployeeService

    b)   Set SSL client authentication key alias to : consumercert

Note that the target port, 7802, is owned by the TCP/IP Monitor. This will forward the request on to the ultimate destination, on port 7848.
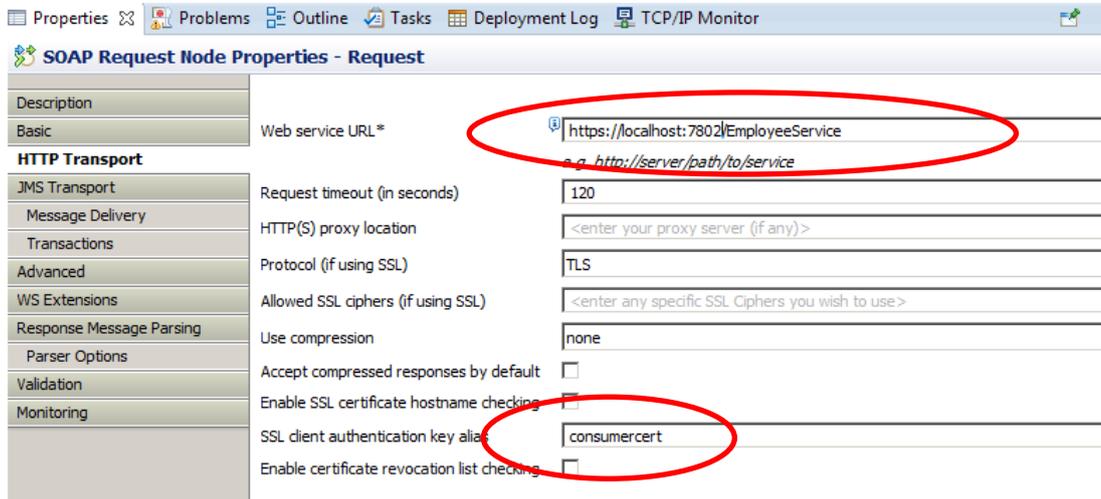
6.   Save the sub flow (ctrl S).

7.   Close the subflow and message flow editors.

8.   Open the "**ManualBuild_EmployeeService-JSONClient**" bar file and click the prepare tab.

9.   Click the Build and Save button to rebuild the bar file.

Web Services Security   Version 10.0.0.3
Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
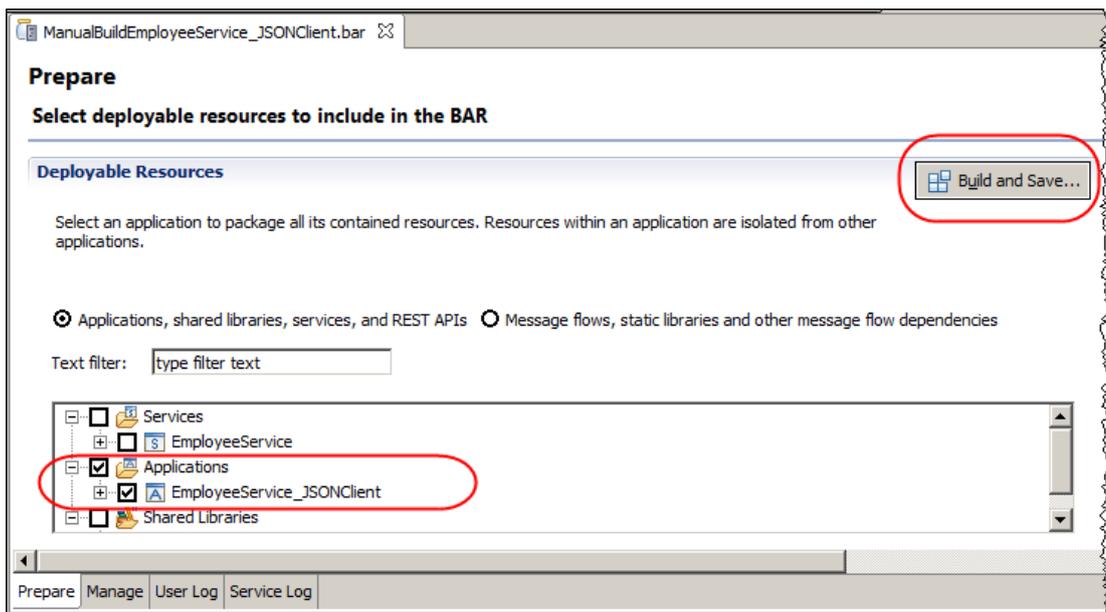
10. Click the manage tab, navigate down to the message flow called

EmpServ_JSON_getEmployee.msgflow

Click the message flow name, then click the properties tab.

11. In the "Configure" (horizontal) tab, use the Edit button to reset the Consumer Policy Set and Consumer Policy Set Bindings values to **myConsumerPolicySet** and **myConsumerPolicySetBinding** respectively:
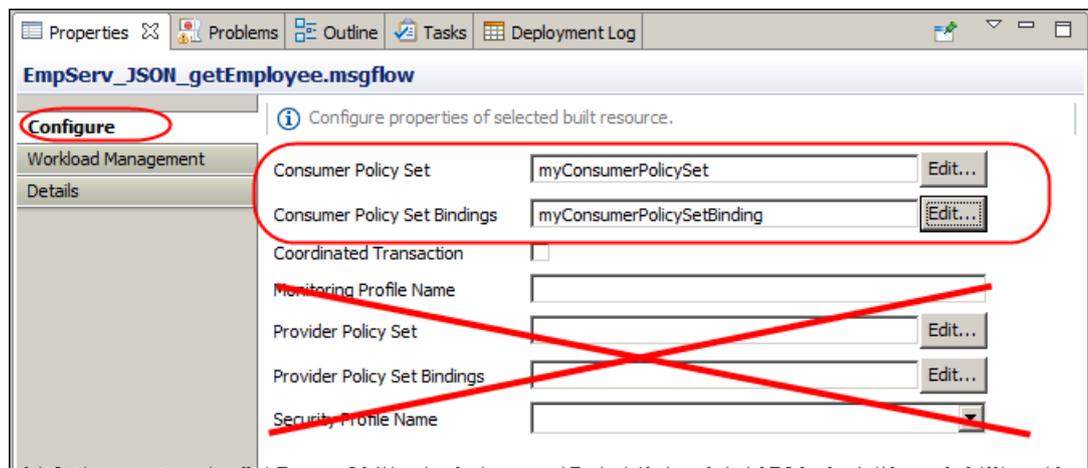
(do not specifying anything in the Provider Policy settings).

12. Save (ctrl-s) and close the bar file.

13. Deploy the bar file to IB10NODEC.CONSUMER.

The JSON client is now configured to request the EmployeeService on an https connection and pass "consumercert" as credentials to authenticate with the service. The Policy Set configuration is (still) set to partially secure the message response from the EmployeeService, however the remaining part of the message content response and the request from the JSON client would otherwise be in the clear.

The TLS configuration will ensure the data exchanged between the consumer and provider is not in the clear.

9.3 Test TLS scenario

The JSON client is now configured to request the EmployeeService on an https connection and pass "consumercert" as credentials to authenticate with the service. The Policy Set configuration is (still) set to partially secure the message response from the EmployeeService, however the remaining part of the message content response and the request from the JSON client would otherwise be in the clear.

The TLS configuration will ensure the data exchanged between the consumer and provider is not in the clear.

9.3.1 Reconfigure the TCP/IP Monitor properties

The connection between the Consumer and Provider is now secured using TLS. This means that the TCP/IP Monitor will not be able to detect that the transmission is using http. To accommodate this change, you will need to reconfigure the TCP/IP Monitor.

1.   In the TCP/IP Monitor window, right-click and select Properties.
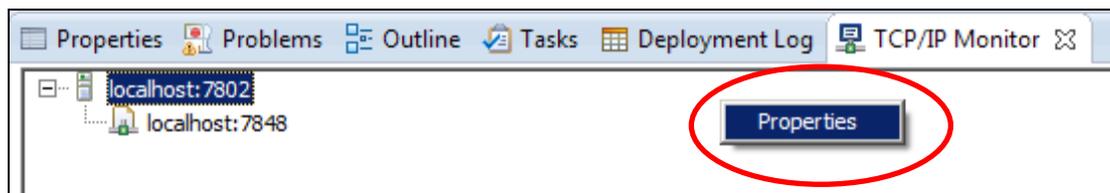
2.   Stop the current Monitor:

                      Authentication and Encryption using X.509 Certificates
                                  Provided by IBM BetaWorks
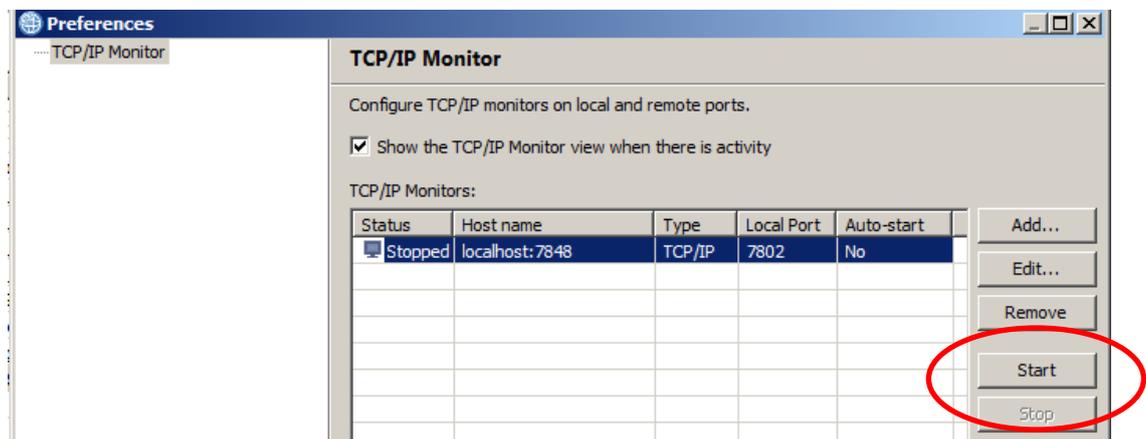
3.   Click Add to add a new monitor.

Set the following values:
- Local monitoring port: 7802
- Host name :   localhost
- Port :   7848
- Type :   TCP/IP

Click OK, then Start the Monitor.

Click OK to close the Properties window.

9.3.2 Send a message using SOAPUI

1.   In the SOAPUI window click the green arrow to send the message again.

                    Authentication and Encryption using X.509 Certificates
                                Provided by IBM BetaWorks

2.   The request should respond successfully (click the "JSON" tab to see the response data).

```json
{"EmployeeResponse": {
    "DBResp":      {
        "UserReturnCode": 0,
        "RowsRetrieved": 1,
        "RowsAdded": 0,
        "RowsUpdated": 0,
        "RowsDeleted": 0,
        "SQLCode_ErrorCode": 0,
        "SQLState_SQLState": "",
        "SQL_Error_Message": ""
    },
    "EMPLOYEE":      {
        "EMPNO": "000010",
        "FIRSTNME": "CHRISTINE",
        "MIDINIT": "I",
        "LASTNAME": "HAAS",
        "WORKDEPT": "A00",
        "PHONENO": "3978",
        "HIREDATE": "1995-01-01",
        "JOB": "PRES      ",
        "EDLEVEL": 18,
        "SEX": "F",
        "BIRTHDATE": "1963-08-24",
        "SALARY": 152750,
        "BONUS": 1000,
        "COMM": 4220
    }
}}
```

9.3.3 Verify the test in the TCP/IP Monitor

1.   Switch to the Monitor.

This time, with TLS, there will be no evidence of the data or protocols that the Consumer and Provider have used to talk to each other. Nothing will be "in the clear".

However, you will see that several thousand bytes have been sent and received.

End of Lab

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
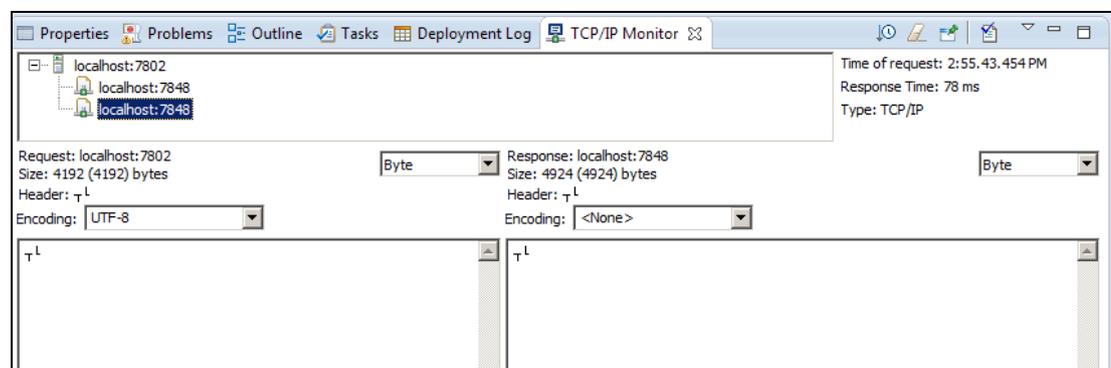
10. Appendix

10.1 Defining the self signed certificates used in this lab guide

The self-signed certificates used in this lab guide were created using the Open SSL package.
A key and self-signed certificate were first created for a Root Authority called rootCA.
Private keys and certificates (signed by the rootCA) where then created for use by the consumer and provider.
These were then saved as in PKCS12 format before being imported into the JKS key stores used by the IBM Integration Servers.
The following section outlines the commands that were used to create the rootCA and consumer certificates.
The provider certificates were created in the same way as the consumer certificates.

10.1.1   Preparing your openssl environment:

1.   Install OpenSSL. (the details in this section were done on SSL Light downloaded from the openssl.org web site)

2.   If you are using windows, and you see an error stating that the openssl.cfg file cannot be located, you will need to set an environment variable OPENSSL_CONF, the variable needs setting to (by default C:\OpenSSL-Win32\bin\openssl.cfg")

3.   Ensure the OpenSSL application can write to the directory you were using the openssl.exe file in. In windows open a command prompt with the "Run As administrator" option.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

10.1.2   Create the RootCA:

1.   Enter the command to create an RSA 2048 bit key:

   ```
   C:\OpenSSL-Win32\bin>openssl genrsa -out c:\sample\rootCA.key 2048

   The response will be similar to the following:

   Loading 'screen' into random state - done

   Generating RSA private key, 2048 bit long modulus

   ..................................................................

   ...........................................................+++

   ....+++

   e is 65537 (0x10001)

   C:\OpenSSL-Win32\bin>
   ```

   This will create a private key called "rootCA.key" in C:\sample which we can now use to base
   the certificate requests on.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

2.   Now create the certificate request:

openssl req -x509 -new -nodes -key c:\sample\rootCA.key -days 2048 -out c:\sample\rootCA.crt

The response to the command does require input, the input in the following is in the larger text:

```
Loading 'screen' into random state – done

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:GB

State or Province Name (full name) [Some-State]:warwickshire

Locality Name (eg, city) []:warwick

Organization Name (eg, company) [Internet Widgits Pty Ltd]:ibm

Organizational Unit Name (eg, section) []:betaworks

Common Name (e.g. server FQDN or YOUR name) []:rootCA

Email Address []:
```

At the end of this command you have a certificate request for the rootCA

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

10.1.3   Create the signed CONSUMER certificate:

1.   Enter the command to create an RSA 2048 bit key:

C:\OpenSSL-Win32\bin>openssl genrsa -out c:\sample\consumer.key 2048

The response will be similar to the following:

Loading 'screen' into random state - done

Generating RSA private key, 2048 bit long modulus

...

..+++

....+++

e is 65537 (0x10001)

C:\OpenSSL-Win32\bin>

This will create a private key called "consumer.key" in C:\sample which we can now use to base the certificate requests on.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

2.   Create the certificate request for the consumer

openssl req -new -key c:\sample\consumer.key -out c:\sample\consumer.csr

The response to the command does require input, the input in the following is in the larger text

```
Loading 'screen' into random state - done

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:GB

State or Province Name (full name) [Some-State]:warwickshire

Locality Name (eg, city) []:warwick

Organization Name (eg, company) [Internet Widgits Pty Ltd]:ibm

Organizational Unit Name (eg, section) []:betaworks

Common Name (e.g. server FQDN or YOUR name) []:consumer

Email Address []:


Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:
```

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

3. Now generate the consumer signed certificate using the rootCA certificate and key:

```
openssl  x509  -req  -in  c:\sample\consumer.csr  -CA  c:\sample\rootCA.crt  -CAkey
c:\sample\rootCA.key -CAcreateserial -days 2048 -out c:\sample\consumer.crt
```

Response:

```
Loading 'screen' into random state - done

Signature ok

subject=/C=GB/ST=warwickshire/L=warwick/O=ibm/OU=betaworks/CN=consumer

Getting CA Private Key
```

4. Store the key and certificate in PKCS12 format to enable import into JKS key stores used by IBM Integration Bus.

```
openssl pkcs12 -export -in c:\sample\consumer.crt -inkey c:\sample\consumer.key -out
c:\sample\consumerkeystore.p12 -name consumercert
```

Response:

```
Loading 'screen' into random state - done

Enter Export Password:

Verifying - Enter Export Password:
```

Give the PKCS12 format file a password as part of the response otherwise the import will fail when you come to import the key.

10.1.4   Import the keys and certificates:

The following section show how you to import a key saved in PKCS12 format into a JKS key store used by IIB.

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks

1. Using IKEYMAN, open the Consumer key store:
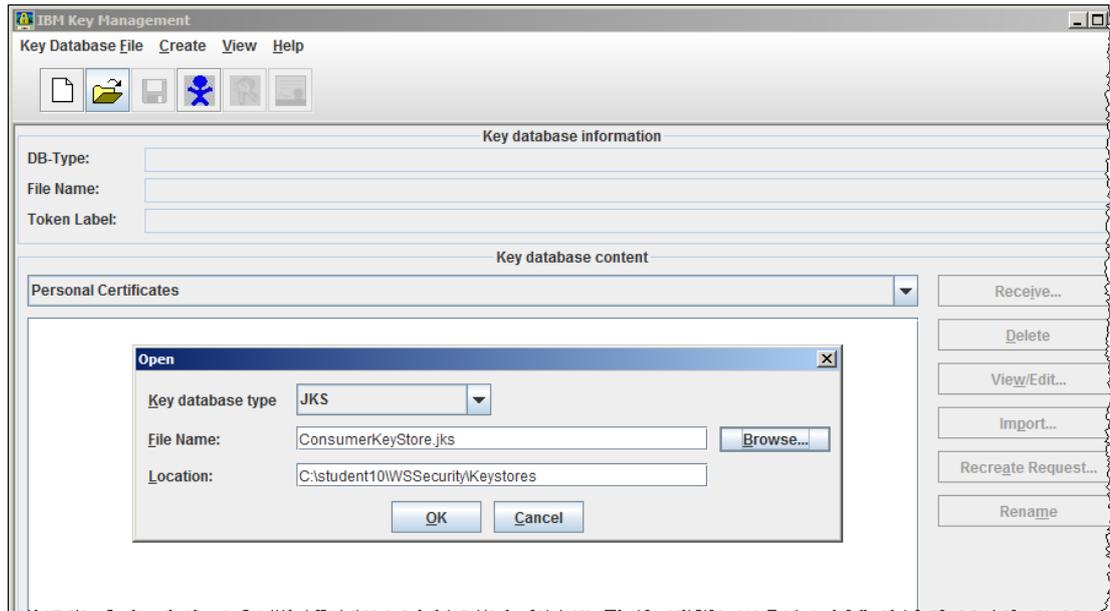
2. When prompted, enter the password specified for the key store:

3. Once the keystore is open click the Import button (if there are keys already stored in the key store this will also appear as an Import/Export button):
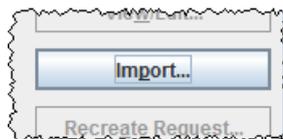
4. Browse for the consumerkeystore.p12 file that you created in the previous section:

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
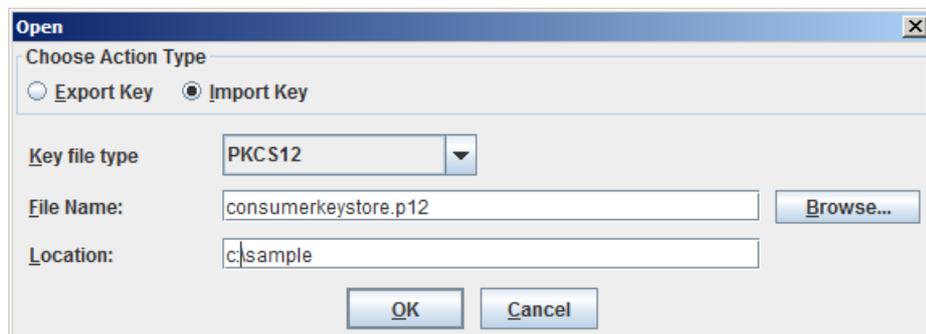
5.   Key in the password you defined for the PKCS12 file:

Password Prompt

Password: ●●●●●●●●

OK   Reset   Cancel

6.   Click "ok" on the "Change Labels" screen:

Change Labels

Would you like to change any of these labels before completing the import process?   OK

Select a label to change: Cancel

consumercert

Enter a new label:

Apply

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
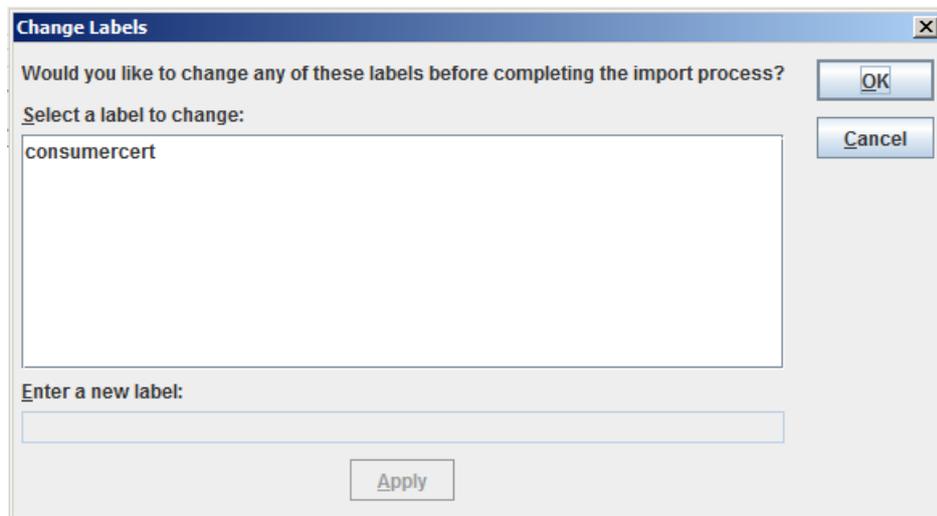
7.   The consumercert certificate and key will be imported into the key store:

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks
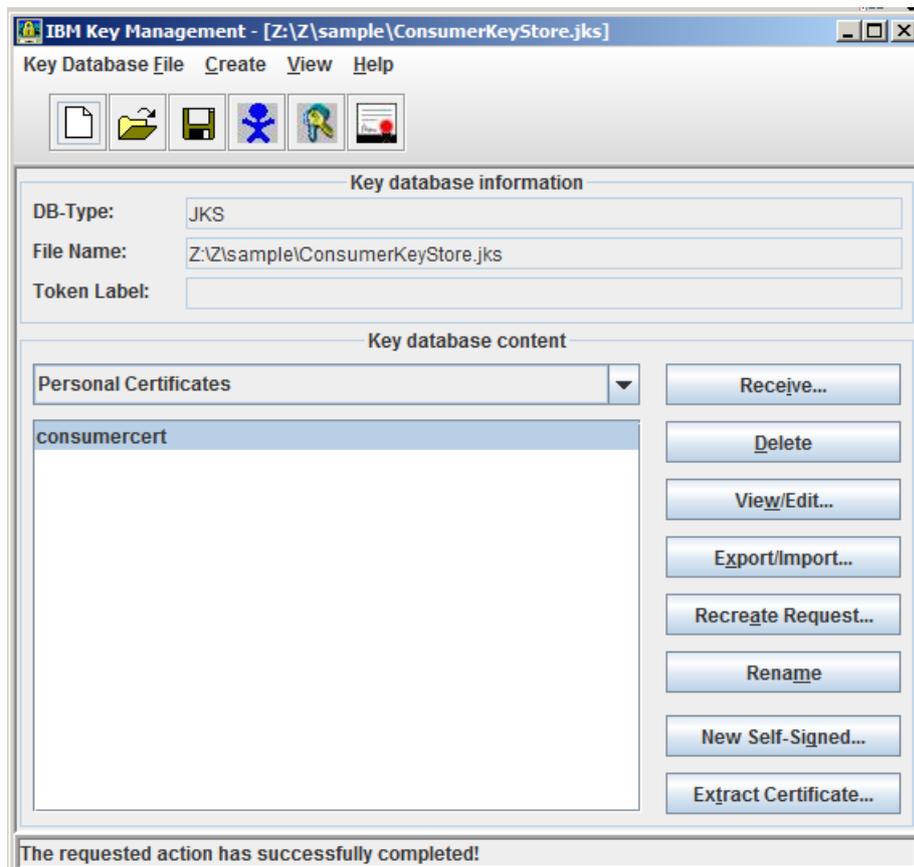
8.   Take a look at the certificate, click the View/Edit button when the consumercert is highlighted:
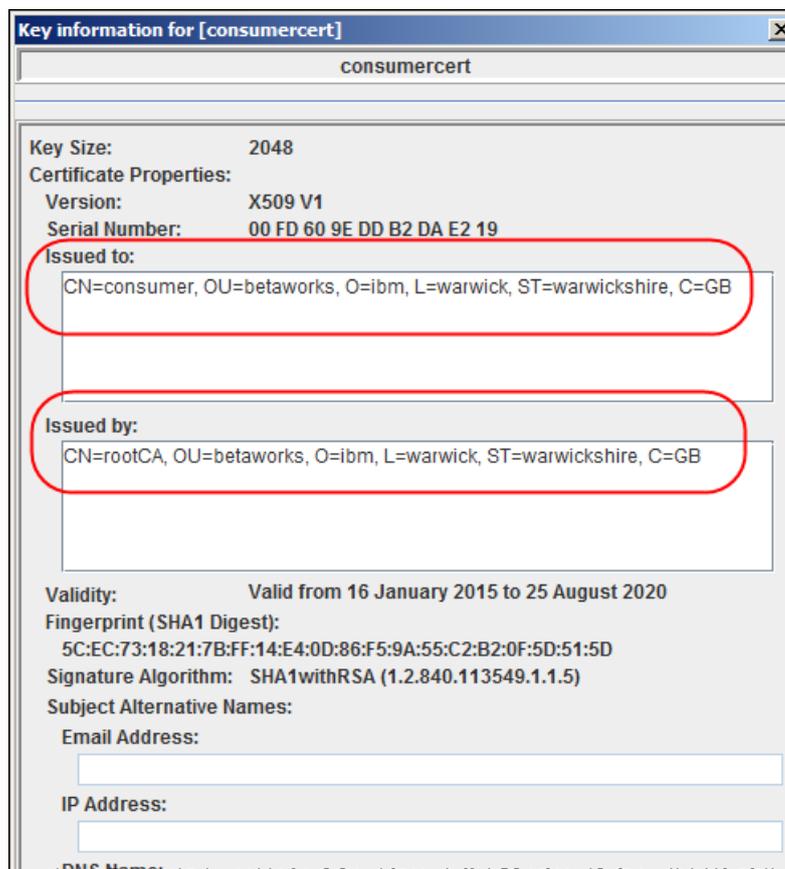
Notice:

1) the Distinguished Name defined in the **"Issued to"** field reflect the details you entered when creating the *consumer* certificate request in OpenSSL

2) the Distinguished Name defined in the **"Issued by"** field reflect the details of the certificate Authority (rootCA) which signed the *consumer* certificate request in OpenSSL.

9.   You can now Extract the Certificate as an ".arm" file enabling you to then "add" it (for example) to trust store used by the provider (enabling an integration server used as a "provider" to trust certificates sent from an application running on the integration server used as a "consumer").

Similarly it is likely that you will need to configure a "consumer" to trust responses back from the provider (in this case you would "Add" the extracted provider certificate *as a signer* to the "consumer" trust store).

The key stores and trust stores used throughout this lab guide were set up and configured using the above techniques.

10. In order to avoid certificate verification chain errors you will also need the rootca certificate defined in the key store. Add (as a "Signer certificate") the rootCA.crt file created in Open SSL to the key store:
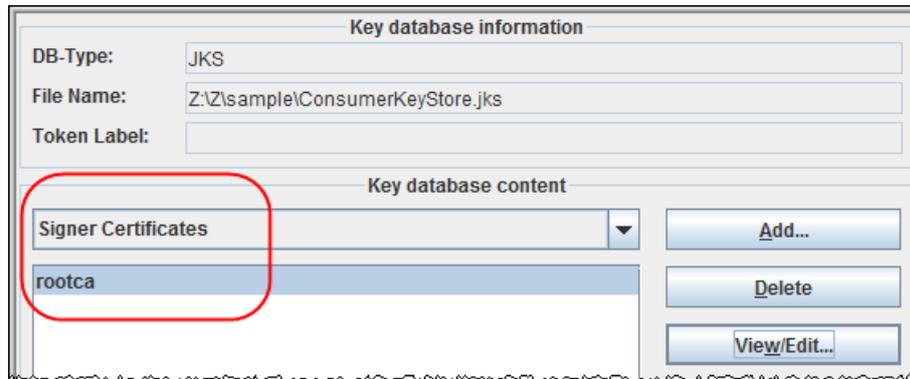
11. If you look at this certificate using the View/Edit button you will see the details of the Certificate request for the "mock" root certificate authority you specified in "open SSL" above:

End of Lab Guide

Authentication and Encryption using X.509 Certificates
Provided by IBM BetaWorks