IBM Global Security Kit

Secure Sockets Layer Introduction and iKeyman User's Guide

for version 7

Note

Before using this information and the product it supports, read the information in Appendix. Notices.

Edition August 9, 2006

This edition applies to iKeyman version 7 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

This book is intended for network or system security administrators who install, administer, and use Secure Socket Layer (SSL) based systems. This book assumes the reader is familiar with their network architecture and e-business applications.

How this book is organized

This book contains the following chapters:

- Secure sockets layer overview provides an overview of SSL and digital certificates.
- Managing digital certificates with iKeyman describes the iKeyman utility, which is a tool you can use to manage your digital certificates. Also included in this chapter are example situations in which iKeyman can be used to manage certificates.
- Using the IKEYCMD Command Line Interface describes the IKEYCMD command line interface.
- GSKit iKeyman support for accessibility describes accessibility features.

Contacting software support

Before contacting IBM Tivoli Software Support with a problem, refer to the IBM Tivoli Software Support site by clicking the Tivoli support link at the following Web site: http://www.ibm.com/software/support/

If you need additional help, contact software support by using the methods described in the *IBM Software Support Guide* at the following Web site: http://techsupport.services.ibm.com/guides/handbook.html

The guide provides the following information:

- Registration and eligibility requirements for receiving support
- Telephone numbers, depending on the country in which you are located
- A list of information you should gather before contacting customer support

Conventions used in this book

This reference uses several conventions for special terms and actions and for operating system-dependent commands and paths.

Operating system differences

This book uses the UNIX(TM) convention for specifying environment variables and for directory notation. When using the Windows(TM) command line, replace *$variable* with *%variable%* for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Typeface conventions

The following typeface conventions are used in this reference:

Bold

Lowercase commands or mixed case commands that are difficult to distinguish from surrounding text, keywords, parameters, options, names of Java(TM) classes, and objects are in **bold**.

Italic

Variables, titles of publications, and special words or phrases that are emphasized are in *italic*.

`Monospace`

Code examples, command lines, screen output, file and directory names that are difficult to distinguish from surrounding text, system messages, text that the user must type, and values for arguments or command options are in `monospace`.

Secure sockets layer overview

Privacy and security are concepts that are more critical than ever in today's electronic business environment.

Every business professional needs to be concerned about security over open communication networks, such as the Internet. It is not enough to have a secure Web site; you also need to have secure communication *between* Web sites -- communication that cannot be monitored by outside parties. Both you and your users need to be confident that you have a secure environment in which to conduct your business.

Such secure communication requires encryption, and encryption is what the Secure Sockets Layer (SSL) provides; security for data communication connections.

SSL was developed jointly by Netscape(R) Communications and RSA(R) Data Security. Many companies worldwide have adopted the SSL communication protocol. For instance, many financial transactions on the Internet, including online banking, are now conducted using SSL.

This chapter consists of two sections:

- Digital certificates
- How SSL works

Digital certificates

Digital certificates allow unique identification of an entity; they are, in essence, electronic ID cards issued by trusted parties. Digital certificates allow a user to verify to whom a certificate is issued as well as the issuer of the certificate.

Digital certificates are the vehicle that SSL uses for public-key cryptography. Public-key cryptography uses two different cryptographic keys: a *private key* and a *public key*. Public-key cryptography is also known as *asymmetric cryptography*, because you can encrypt information with one key and decrypt it with the complement key from a given public-private *key pair*.

Public-private key pairs are simply long strings of data that act as keys to a user's encryption scheme. The user keeps the private key in a secure place (for example, encrypted on a computer's hard drive) and provides the public key to anyone with whom the user wants to communicate. The private key is used to digitally sign all secure communications sent from the user; the public key is used by the recipient to verify the sender's signature.

Public-key cryptography is built on trust; the recipient of a public key needs to have confidence that the key really belongs to the sender and not to an impostor. Digital certificates provide that confidence.

A digital certificate serves two purposes: it establishes the owner's identity, and it makes the owner's public key available. A digital certificate is issued by a trusted authority--a certificate authority (CA)--and it is issued only for a limited time. When its expiration date passes, the digital certificate must be replaced.

Format of digital certificates

The digital certificate contains specific pieces of information about the identity of the certificate owner and about the certificate authority, these being:

- The owner's distinguished name. A distinguished name is the combination of the owner's common name and its context (position) in the directory tree. In the simple directory tree shown in Figure 1, for example, `LaurenA` is the owner's common name (CN), the organization unit (OU) is `Engnring` and the Organization (O) is `XYZCorp`; therefore, the distinguished name is:

  ```
  .CN=LaurenA.OU=Engnring.O=XYZCorp
  ```

- The owner's public key.
- The date the digital certificate was issued.
- The date the digital certificate expires.
- The issuer's distinguished name. This is the distinguished name of the issuing CA.
- The issuer's digital signature.

Figure 1. A simple directory tree

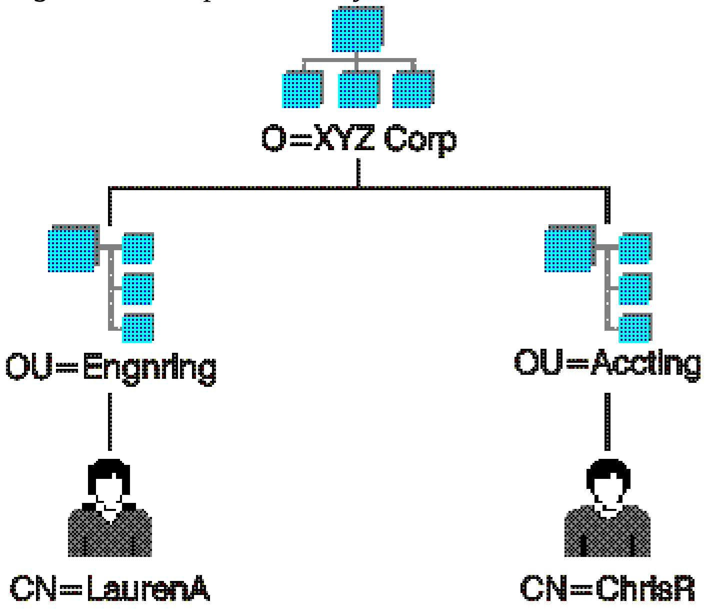
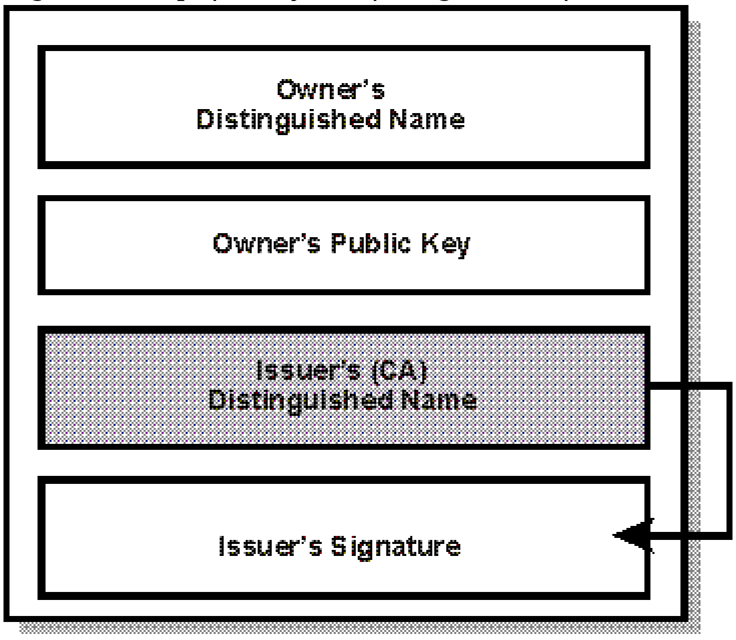


Figure 2 shows the layout of a typical digital certificate.

Figure 2. Simplified layout of a digital certificate



Security considerations for digital certificates

If you send your digital certificate containing your public key to someone else, what keeps that person from misusing your digital certificate and posing as you? The answer is *your private key*.

A digital certificate alone can never be proof of anyone's identity. The digital certificate just allows you to verify the identity of the digital certificate owner by providing the public key that is needed to check the digital certificate owner's digital signature. Therefore, the digital certificate owner must protect the private key that belongs to the public key in the digital certificate. If the private key is stolen, the thief can pose as the legitimate owner of the digital certificate. Without the private key, a digital certificate cannot be misused.

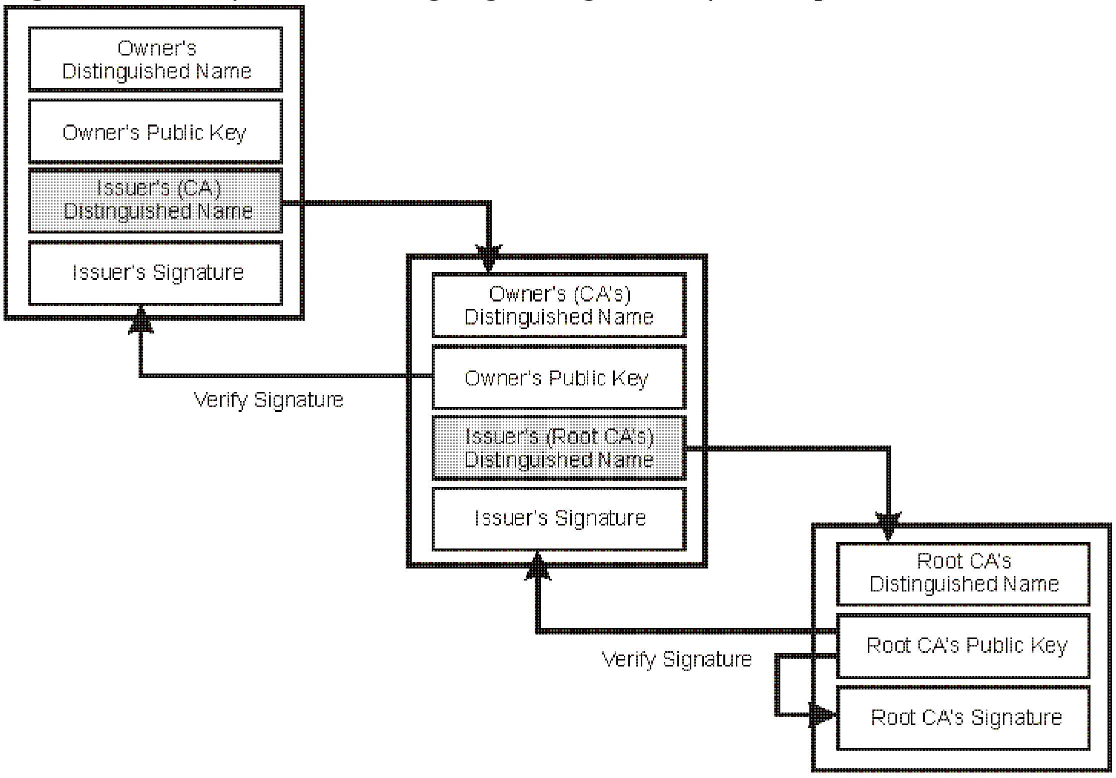
Certificate authorities and trust hierarchies

Trust is a very important concept in digital certificates. Each organization or user must determine which CAs can be accepted as trustworthy.

A user of a security service requiring knowledge of a public key generally needs to obtain and validate a digital certificate containing the required public key. Receiving a digital certificate from a remote party does not give the receiver any assurance about the authenticity of the digital certificate. To verify that the digital certificate is authentic, the receiver needs the public key of the certificate authority that issued the digital certificate.

If the public key user does not already hold an assured copy of the public key of the certificate authority that signed the digital certificate, then the user might need an additional digital certificate to obtain that public key. In general, a chain of multiple digital certificates might be needed, comprising a digital certificate of the public key owner (the end entity) signed by one CA, and optionally one or more additional digital certificates of CAs signed by other CAs. Figure 3 shows a chain of trust.

Figure 3. Chain of trust -- CAs signing CA digital certificates up to the root CA


Owner's Distinguished Name
Owner's Public Key
Issuer's (CA) Distinguished Name
Issuer's Signature
Verify Signature
Owner's (CA's) Distinguished Name
Owner's Public Key
Issuer's (Root CA's) Distinguished Name
Issuer's Signature
Verify Signature
Root CA's Distinguished Name
Root CA's Public Key
Root CA's Signature

Note that many applications that send a subject's digital certificate to a receiver send not only that digital certificate, but also send all the CA digital certificates necessary to verify the initial digital certificate up to the root CA.

The chain of trust begins at the root CA. The root CA's digital certificate is self-signed; that is, the certificate authority uses its own private key to sign the digital certificate. The public key used to verify the signature is the public key in the digital certificate itself. To establish a chain of trust, the public-key user must have received the digital certificate of the root CA in one of the following ways:

- On a diskette received by registered mail or picked up in person.
- Pre-loaded with software received from a reliable source or downloaded from an authenticated server.

Uses for digital certificates in Internet applications

Applications using public-key cryptography systems for key exchange or digital signatures need to use digital certificates to obtain the needed public keys. Internet applications of this kind are numerous. Following are brief descriptions of a few of the commonly used Internet applications that use public-key cryptography:

SSL

A protocol that provides privacy and integrity for communications. This protocol is used by

- Web servers to provide security for connections between Web servers and Web browsers,
- LDAP to provide security for connections between LDAP clients and LDAP servers,
- Host-on-Demand V2 to provide security for connections between the client and the host system.

Additional applications based on this protocol are in development.

SSL uses digital certificates for key exchange, server authentication, and optionally, client authentication.

Client Authentication

Client authentication is an option in SSL that requires a server to authenticate a client's digital certificate before allowing the client to log on or access certain resources. The server requests and authenticates the client's digital certificate during the SSL handshake. At that time the server can also determine whether it trusts the CA that issued the digital certificate to the client.

Secure Electronic Mail

Many electronic mail systems, using standards such as *Privacy Enhanced Mail* (PEM) or *Secure/Multipurpose Internet Mail Extensions* (S/MIME) for secure electronic mail, use digital certificates for digital signatures and for the exchange of keys to encrypt and decrypt messages.

Virtual Private Networks (VPNs)

Virtual private networks, also called *secure tunnels*, can be set up between firewalls to enable protected connections between secure networks over insecure communication links. All traffic destined to these networks is encrypted between the firewalls.

The protocols used in tunneling follow the *IP Security* (IPsec) standard. For the key exchange between partner firewalls, the Internet key exchange (IKE) standard, previously known as ISAKMP/Oakley, has been defined.

The standards also allow for a secure, encrypted connection between a remote client (for example, an employee working from home) and a secure host or network.

Secure Electronic Transaction (SET)

SET is a standard designed for secure credit card payments using insecure networks, for example, the Internet. Digital certificates are used for card holders (electronic credit cards) and merchants. The use of digital certificates in SET allows for secure, private connections between card holders, merchants, and banks. The transactions created are secure and indisputable, and they cannot be forged. The merchants receive no credit card information that can be misused or stolen.

Digital certificates and certificate requests

Simplified, a *signed* digital certificate contains the owner's distinguished name, the owner's public key, the certificate authority's (issuer's) distinguished name, and the signature of the certificate authority.

A *self-signed* digital certificate contains the owner's distinguished name, the owner's public key, and the owner's own signature over these fields.

A root CA's digital certificate is an example of a self-signed digital certificate. You can also create your own self-signed digital certificates to use when developing and testing a server product. See Creating a self-signed digital certificate for testing for details.

A certificate request that is sent to a certificate authority to be signed contains the owner's (requester's) distinguished name, the owner's public key, and the owner's own signature. The certificate authority verifies the owner's signature with the public key in the digital certificate to ensure tat:

- The certificate request was not corrupted in transit between the requester and the CA.
- The private key used to generate the request matches the public key in the certificate request.

The CA is also responsible for some level of identification verification. This can range from very little proof to absolute assurance of the owner's identity, the later being most important is financial systems.

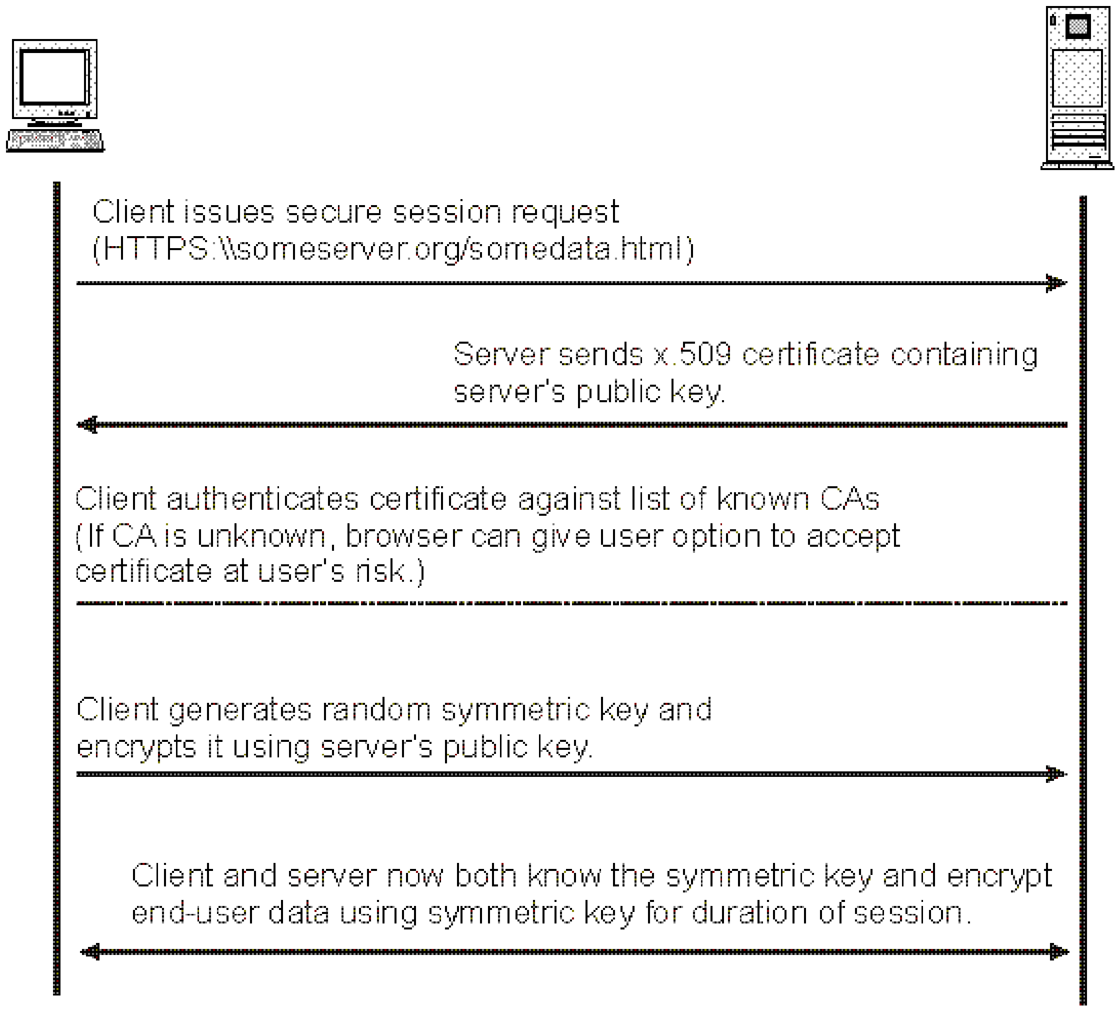
How SSL works

SSL is a protocol that provides privacy and integrity between two communicating applications using TCP/IP. The *Hypertext Transfer Protocol* (HTTP) for the World Wide Web uses SSL for secure communications.

The data going back and forth between client and server is encrypted using a symmetric algorithm such as DES or RC4. A public-key algorithm--usually RSA--is used for the exchange of the encryption keys and for digital signatures. The algorithm uses the public key in the server's digital certificate. With the server's digital certificate, the client can also verify the server's identity. Versions 1 and 2 of the SSL protocol provide only server authentication. Version 3 adds client authentication, using both client and server digital certificates.

The SSL handshake

A HTTP-based SSL connection is always initiated by the client using a URL starting with `https://` instead of with `http://`. At the beginning of an SSL session, an SSL handshake is performed. This handshake produces the cryptographic parameters of the session. A simplified overview of how the SSL handshake is processed is shown in the diagram below.


Client issues secure session request
(HTTPS:\\someserver.org/somedata.html)
Server sends x.509 certificate containing server's public key.
Client authenticates certificate against list of known CAs
(If CA is unknown, browser can give user option to accept certificate at user's risk.)
Client generates random symmetric key and encrypts it using server's public key.
Client and server now both know the symmetric key and encrypt end-user data using symmetric key for duration of session.

The client sends a client "hello" message that lists the cryptographic capabilities of the client (sorted in client preference order), such as the version of SSL, the cipher suites supported by the client, and the data compression methods supported by the client. The message also contains a 28-byte random number.

The server responds with a server "hello" message that contains the cryptographic method (cipher suite) and the data compression method selected by the server, the session ID, and another random number.

Note:

The client and the server must support at least one common cipher suite, or else the handshake fails. The server generally chooses the strongest common cipher suite.

The server sends its digital certificate. (In this example, the server uses X.509 V3 digital certificates with SSL.)

If the server uses SSL V3, and if the server application (for example, the Web server) requires a digital certificate for client authentication, the server sends a "digital certificate request" message. In the "digital certificate request" message, the server sends a list of the types of digital certificates supported and the distinguished names of acceptable certificate authorities.

The server sends a server "hello done" message and waits for a client response.

Upon receipt of the server "hello done" message, the client (the Web browser) verifies the validity of the server's digital certificate and checks that the server's "hello" parameters are acceptable.

If the server requested a client digital certificate, the client sends a digital certificate, or if no suitable digital certificate is available, the client sends a "no digital certificate" alert. This alert is only a warning, but the server application can fail the session if client authentication is mandatory.

The client sends a "client key exchange" message. This message contains the *pre-master secret*, a 46-byte random number used in the generation of the symmetric encryption keys and the *message authentication code* (MAC) keys, encrypted with the public key of the server.

If the client sent a digital certificate to the server, the client sends a "digital certificate verify" message signed with the client's private key. By verifying the signature of this message, the server can explicitly verify the ownership of the client digital certificate.

Note:

An additional process to verify the server digital certificate is not necessary. If the server does not have the private key that belongs to the digital certificate, it cannot decrypt the pre-master secret and create the correct keys for the symmetric encryption algorithm, and the handshake fails.

The client uses a series of cryptographic operations to convert the pre-master secret into a *master secret*, from which all key material required for encryption and message authentication is derived. Then the client sends a "change cipher spec" message to make the server switch to the newly negotiated cipher suite. The next message sent by the client (the "finished" message) is the first message encrypted with this cipher method and keys.

The server responds with a "change cipher spec" and a "finished" message of its own.

The SSL handshake ends, and encrypted application data can be sent.

Digital certificates and trust chains with SSL

Secure Sockets Layer V3 can use server digital certificates as well as client digital certificates. As previously explained, server digital certificates are mandatory for an SSL session, while client digital certificates are optional, depending on client authentication requirements.

The *public key infrastructure* (PKI) used by SSL allows for any number of root certificate authorities. An organization or end user must decide for itself which CAs it will accept as being *trusted*. To be able to verify the server digital certificates, client Web browsers require possession of the root CA digital certificates used by servers. Popular Web browsers usually come with a *key ring* where a number of CA digital certificates, called trusted roots, are already installed. This list can be edited, and the digital certificates of untrusted CAs can be deleted.

If an SSL session is about to be established with a server that sends a digital certificate whose root CA digital certificate is not in the key ring, the browser displays a warning window and presents options either to import the digital certificate or to abort the session. To avoid this situation, import the root CA digital certificate from a Web page or use a JavaScript(TM) program that imports the digital certificate.

If client authentication is used, the Web server requires possession of the root CA digital certificates used by clients. Because it is not possible to import root CA digital certificates into the server application dynamically, all root CA digital certificates that are not part of the server key ring at delivery time must be installed using the iKeyman utility before any client digital certificates are issued by these CAs. For more information on iKeyman, see Managing digital certificates with iKeyman.

SSL with global server certificates

When an SSL session is established between the international version of Netscape Navigator(R)/Communicator(R) V4, Microsoft Internet Explorer(TM) V4, IBM SSL-enabled client applications, or client applications written using the SSL Toolkit and a Web server equipped with a global server certificate, a normal SSL handshake (described in The SSL Handshake) is performed initially. Usually, the Web server and the Web browser settle on the cipher suite, **SSL_RSA_EXPORT_WITH_RC4_40_MD5**. They use 512-bit RSA keys for key exchange, RC4 with 40-bit keys for encryption, and MD5 for message authentication.

During the handshake, the browser receives and verifies the server digital certificate and realizes that this digital certificate authorizes stronger encryption. Remember that the browser sends the list of cryptographic suites it supports with the "client hello" message before the server sends its digital certificate. At this point, the browser has no knowledge of the server's global server certificate.

The first handshake is completed with the "change cipher specification" and "finished" messages from both client and server. At this point, the client initiates another SSL handshake. This time, in the "client hello" message, the client includes the strong cryptographic suites such as:

SSL_RSA_WITH_RC4_128_MD5 (1024-bit RSA keys for key exchange, RC4 with 128-bit keys for encryption, and MD5 for message authentication).

SSL_RSA_WITH_3DES_EDE_CBC_SHA (1024-bit RSA keys for key exchange, triple DES with 168-bit keys for encryption, and SHA-1 for message authentication).

After the second handshake is completed, one of the stronger cryptographic suites is used. This double handshake is also known as the *SSL step-up protocol*. Actually, all application data exchanged in the SSL session are encrypted with the stronger encryption protocol. Compared to the use of a United States browser, the only drawback is the higher overhead of performing an SSL handshake twice.

Managing digital certificates with iKeyman

The iKeyman utility is a tool you can use to manage your digital certificates. With iKeyman, you can:

Create a new key database

Create a test digital certificate,

Add CA roots to your database,

Copy certificates from one database to another,

Request and receive a digital certificate from a CA,

Set default keys,

Change passwords.

You can also use the iKeyman utility to perform many of these functions for digital certificates on a smart card.

The iKeyman utility is automatically installed with the SSL Toolkit.

Towards the end of this chapter, Example iKeyman usage scenarios provides some real-world examples of iKeyman usage which you might find aids understanding of the utility.

Starting iKeyman

There are three packages for iKeyman version 7:

- GSKit iKeyman for GSKit users,
- GSKit Java-only iKeyman for Java-only users, and
- JDK iKeyman (shipped with IBM JDK 1.4.1) for Java-only users.

To start GSKit iKeyman:

1. Complete the installation steps and setting environment values as described in your product documentation.
2. Invoke iKeyman with the following:

   Windows:
   Click the iKeyman shortcut from the Startup menu.

   UNIX:
   To invoke iKeyman on 32-bit installations, type:

```
gsk7ikm
```

To invoke iKeyman on 64-bit installations, type:

```
gsk7ikm_64
```

To start GSKit Java-only iKeyman:

1. Complete installation steps and setting environment values as described in your product documentation.
2. Change directory to the location where the iKeyman classes files were extracted. For example:

   ```
   cd /usr/local/ibm/gsk7/classes
   ```

3. Invoke iKeyman with the following command:

   ```
   $JAVA_HOME/bin/java -classpath gsk7cls.jar
   com.ibm.gsk.ikeyman.Ikeyman
   ```

To start JDK iKeyman:

1. Complete installation steps and setting environment values as described in your product documentation.
2. Invoke iKeyman with the following command:

   ```
   $JAVA_HOME/bin/java com.ibm.gsk.ikeyman.Ikeyman
   ```

Note:

If, as described in the Installation Guide, you are using IBM JDK 1.4.1 and `JAVA_HOME/jre/lib/ext/gskikm.jar` exists, IBM JDK 1.4.1 iKeyman is run regardless of how iKeyman is started. If you are unsure which version of iKeyman you are running, check the **About** window under the **Help** menu for the version details. If it shows Version 7.0.0.0 or Version 6.0.0.0, you are running IBM JDK iKeyman.

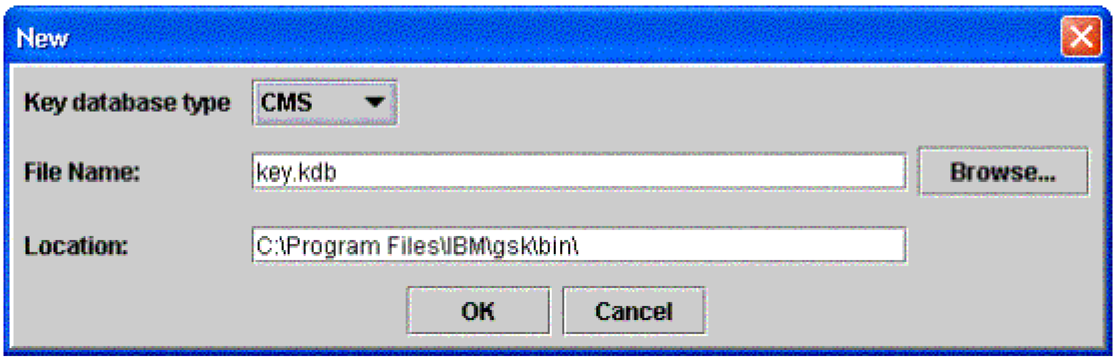
Creating a key database

A key database enables a client application to connect to trusted servers. A trusted server has digital certificates signed by the same CA as your digital certificates.

To create a CMS key database file, follow these steps:

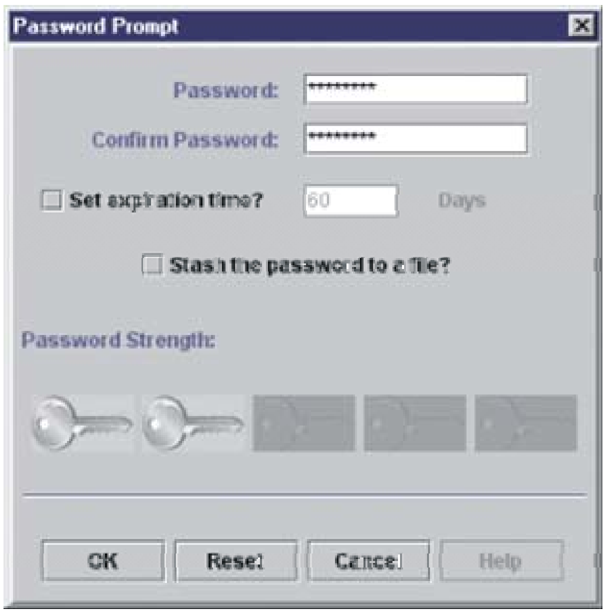
1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **New**. The **New** window is displayed.

   Figure 4. New Key Database File window



3. Select **CMS** in the **Key database type** field.
4. Type a **File Name**, such as `key.kdb`.
5. Accept the default value for the **Location** field, or choose a new location.
6. Click **OK**. The **Password Prompt** window is displayed..

Figure 5. Password Prompt window



7. Enter a password in the **Password** field, and confirm it again in the **Confirm Password** field. Click **OK**.
8. A confirmation window is displayed, verifying that you have created a key database. Click **OK**.

After successful key database creation, the **IBM Key Management** window is displayed which should now reflect your new CMS key database file (for example, `C:\Program Files\ibm\gsk\bin\key.kdb`), and your signer digital certificates.

The following signer digital certificates are provided with iKeyman:

- VeriSign Class 2 OnSite Individual CA
- VeriSign International Server CA -- Class 3

- VeriSign Class 3 Public Primary Certification Authority -- G2
- VeriSign Class 2 Public Primary Certification Authority -- G2
- VeriSign Class 1 Public Primary Certification Authority -- G2
- VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
- VeriSign Class 1 Public Primary Certification Authority
- VeriSign Class 2 Public Primary Certification Authority
- VeriSign Class 3 Public Primary Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freemail CA
- Thawte Personal Basic CA
- Thawte Premium Server CA
- Thawte Server CA
- RSA Secure Server Certification Authority

These signer digital certificates enable your clients to connect to servers that have valid digital certificates from these signers.

Now that a key database has been created, you can use it on your client and connect to a server that has a valid digital certificate from one of the signers.

If you need to use a signer digital certificate that is not on the list above, you need to request it from the CA and add it to your key database (see Adding a CA root digital certificate).

Creating a self-signed digital certificate for testing

When developing a production application, you might want to wait until testing is complete before purchasing a true digital certificate. With iKeyman, you can create a self-signed digital certificate to use for testing purposes. A self-signed digital certificate is a temporary digital certificate issued to yourself, with yourself as the CA.

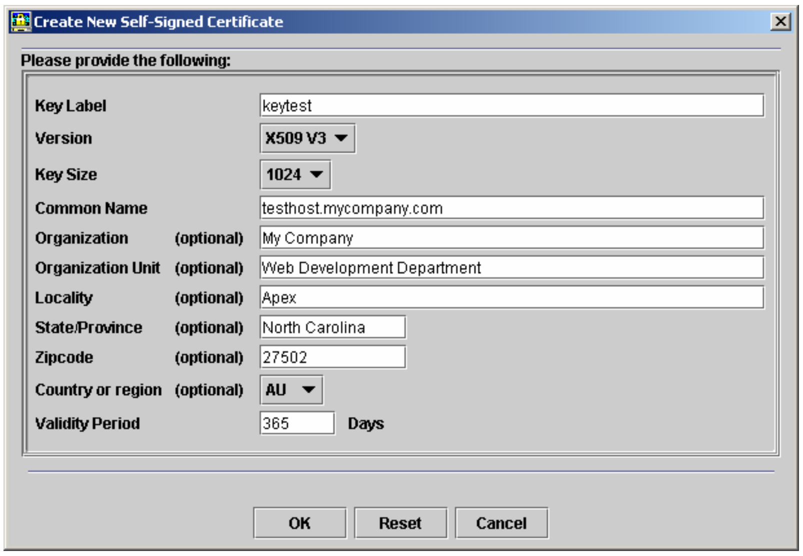
Note:

Do not release a production application with a self-signed digital certificate; no browser or client will be able to recognize or communicate with your server.

To create a self-signed digital certificate in a key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file where you want to add the self-signed digital certificate. Click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the key database file you selected, indicating that the file is open and ready.
5. Select **Personal Certificates** from the pulldown list.
6. Click **New Self-Signed**. The **Create New Self-Signed Certificate** window is displayed.

Figure 6. Create New Self-Signed Certificate window


Create New Self-Signed Certificate
Please provide the following:

Key Label		keytest
Version		X509 V3
Key Size		1024
Common Name		testhost.mycompany.com
Organization	(optional)	My Company
Organization Unit	(optional)	Web Development Department
Locality	(optional)	Apex
State/Province	(optional)	North Carolina
Zipcode	(optional)	27502
Country or region	(optional)	AU
Validity Period		365 Days

7. Type a **Key Label**, such as `keytest`, for the self-signed digital certificate.
8. Type a **Common Name** and **Organization**, and select a **Country**. For the remaining fields, either accept the default values, or enter new values.
9. Click **OK**. The **IBM Key Management** window is displayed. The **Personal Certificates** field shows the name of the self-signed digital certificate you created.

The default validity period for new self-signed digital certificates is 365 days. The minimum is 1 day. The maximum is 7300 days (twenty years).

Adding a CA root digital certificate

Once a root digital certificate has been received from a CA, it should be added to your database.

Note:
Most root digital certificates have the extension `.arm`; for example, `cert.arm`.

To add a CA root digital certificate to a database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file where you want to add the CA root digital certificate and click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Signer Certificates** from the pulldown list.
6. Click **Add**. The **Add CA's Certificate from a File** window is displayed.
7. Click **Data type** and select a data type, such as **Base64-encoded ASCII data**.
8. Type a **Certificate file name** and **Location** for the CA root digital certificate, or click **Browse** to select the name and location.

9. Click **OK**. The **Enter a Label** window is displayed.
10. Type a label for the CA root digital certificate; for example, `VeriSign Test CA Root Certificate`. Click **OK**. The **IBM Key Management** window is displayed.

    The **Signer Certificates** field now shows the label of the newly added CA root digital certificate.

Deleting a CA root digital certificate

If you no longer want to support one of the signers in your signer digital certificate list, you need to delete the CA root digital certificate.

Note:

Create a backup copy of the CA root digital certificate by extracting it from iKeyman. This is a precaution in case you need to re-create the CA root.

To delete a CA root digital certificate from a database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file from which the CA root digital certificate will be deleted. Click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the key database file you selected indicating that the file is open and ready.
5. Select **Signer Certificates** from the pulldown list.
6. Select the CA root digital certificate you want to delete and click **Delete**. The **Confirm** window is displayed. Click **Yes**.
7. The **IBM Key Management** window is displayed. The label of the CA root digital certificate you just deleted should no longer appear in the **Signer Certificates** field.

Copying certificates from one key database to another

When setting up a private trust network or using self-signed certificates for testing purposes, you might find it necessary to extract a certificate from a database and add it to another database as a signer certificate. There are two scenarios. The first scenario *extracts* a personal, or signer certificate from one database and *adds* it to another as a signer certificate. The second scenario *exports* a personal certificate from a source database and *imports* it to a target database as a personal certificate.

The tasks involved in achieving these scenarios are detailed in the following sections.

Scenario 1:

To extract a certificate from a source key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the source key database. This is the database that contains certificate you want to add to another database as a signer certificate. Click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select the type of certificate you want to export: **Personal**, or **Signer**.
6. Select the certificate that you want to add to another database.
7. If you selected **Personal,** click **Extract Certificate**. If you selected **Signer** click **Extract**. The **Extract a Certificate to a File** window is displayed.
8. Click **Data type** and select a data type, such as **Base64-encoded ASCII data**. The data type needs to match the data type of the certificate stored in the certificate file. The iKeyman tool supports Base64-encoded ASCII files and binary DER-encoded certificates.
9. Type the certificate file name and location where you want to store the certificate, or click **Browse** to select the name and location.
10. Click **OK**. The certificate is written to the specified file, and the **IBM Key Management** window is displayed. This completes the first part of the scenario.
11. The second part involves adding a certificate as a signer certificate to the target database. From the **IBM Key Management** window, click **Key Database File** -> **Open**. The **Open** window is displayed.
12. Select the key database that will receive the certificate extracted in the above steps. Click **Open**. The **Password Prompt** window is displayed.
13. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
14. Select the type of certificate you would like to add: **Signer**.
15. Click **Add**. The **Add CA's Certificate from a File** window is displayed.
16. Type the certificate file name used when you extracted the certificate in step 9 above.
17. The **Enter a Label** window is displayed.
18. Specify a name for the certificate. Click **OK**. The certificate is added to the target database.

Scenario 2:

In the previous scenario, a personal, or signer certificate was extracted from a source database and added to a target database as a signer certificate. This scenario exports a personal certificate from a source database and imports it to a target database as a personal certificate.

To export a personal certificate from a source key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the source key database containing the personal certificate to be extracted. Click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Personal Certificates** from the pulldown list.
6. Select the personal certificate you want to export.
7. Click **Export/Import** to transfer keys between the current database and a PKCS#12 file or another database. The **Export/Import Key** window is displayed.
8. Select **Export Key** from *Choose Action Type*.
9. Select **Key File Type** (for example, PKCS12 file) from the pulldown to export list.
10. Type the name of the file (for example, `copy.p12`) to create for the exported certificate, or click **Browse** to select the file name and location, and click **OK**. The **Password Prompt** window is displayed.
11. Enter a password for the certificate file, confirm the password, and click **OK.** The certificate is now exported from the (source) database. This completes the first part of the scenario.

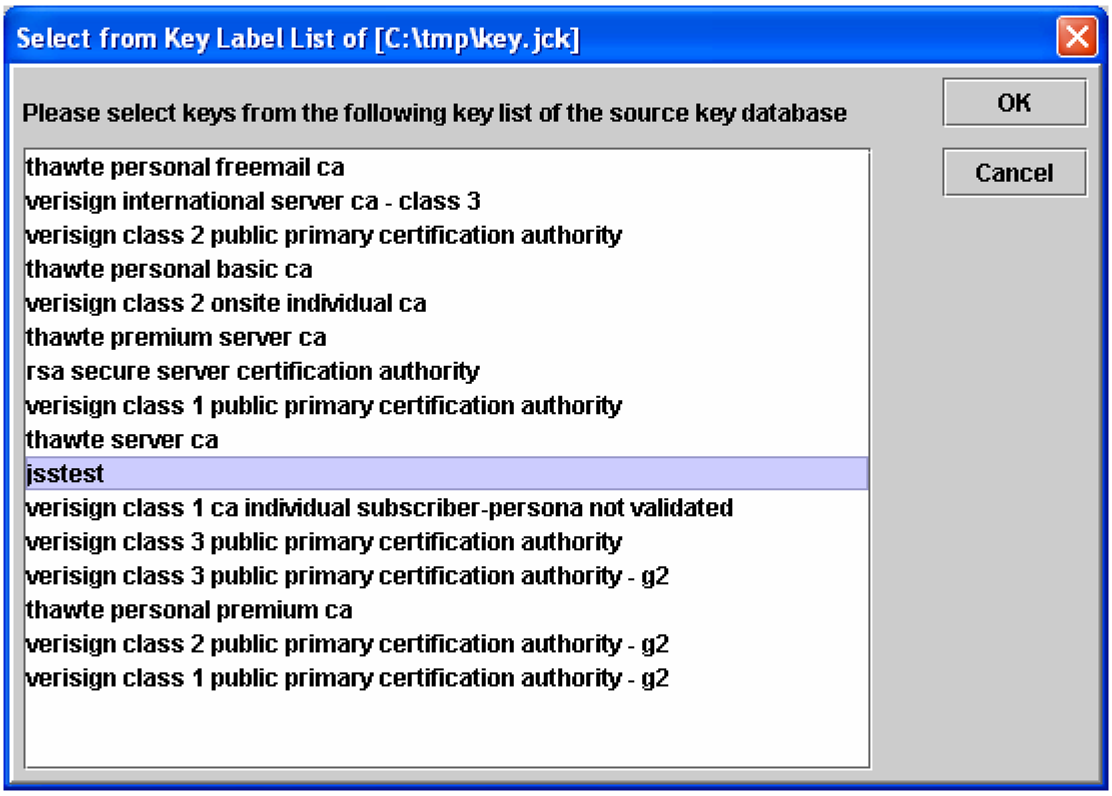
    Note:PKCS#12 files should be considered temporary and deleted after use.

12. The second part of the scenario involves importing a personal certificate to the target key database. To begin, start iKeyman. The **IBM Key Management** window is displayed.
13. Click **Key Database File** -> **Open**. The **Open** window is displayed.
14. Select the target key database for importing the certificate. Click **Open**. The **Password Prompt** window is displayed.
15. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
16. Select **Personal Certificates** from the pulldown list.
17. If the target key database has no personal certificate, click **Import** to import keys from a PKCS#12 file or other database. The **Import Key** window is displayed. If target key database has one or more personal certificates, do the following:

    - Click **Export/Import key**. The **Export/Import key** window is displayed.
    - Select **Import** from **Choose Action Type**.

18. Select the same key file type that you specified from the export. For more information, see step 9.
19. Type the name of the file containing the certificate you exported, or click **Browse** to select the name and location. For more information, see step 10. Click **OK**. The **Password Prompt** window is displayed.
20. Specify the same password that you specified when you exported the certificate. For more information, see step 11. Click **OK**.

Figure 7. Select from Key Label List window



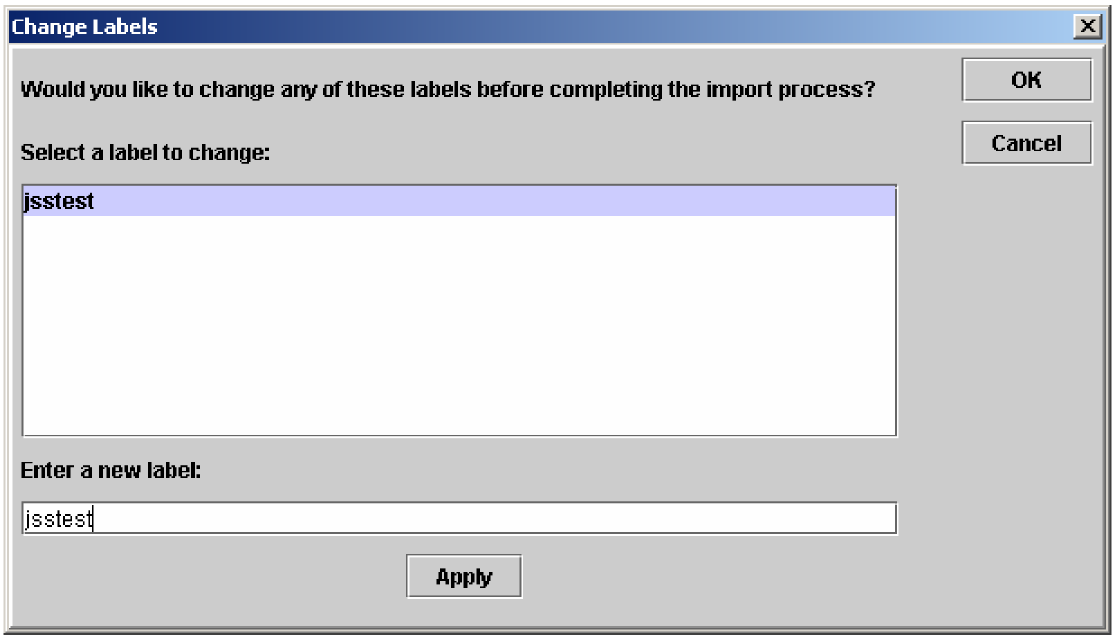
Select from Key Label List of [C:\tmp\key.jck]

Please select keys from the following key list of the source key database

thawte personal freemail ca
verisign international server ca - class 3
verisign class 2 public primary certification authority
thawte personal basic ca
verisign class 2 onsite individual ca
thawte premium server ca
rsa secure server certification authority
verisign class 1 public primary certification authority
thawte server ca
jsstest
verisign class 1 ca individual subscriber-persona not validated
verisign class 3 public primary certification authority
verisign class 3 public primary certification authority - g2
thawte personal premium ca
verisign class 2 public primary certification authority - g2
verisign class 1 public primary certification authority - g2

21. The **Select from Key Label List** window is displayed. From the list of certificate labels displayed select those you wish to import. Be sure to include any signer certificates that might be necessary to form a trust chain for any personal certificates you are importing. These will not be necessary if they are already in the target key database. Click **OK**.

Figure 8. Change Labels window



Change Labels

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

Select a label to change:

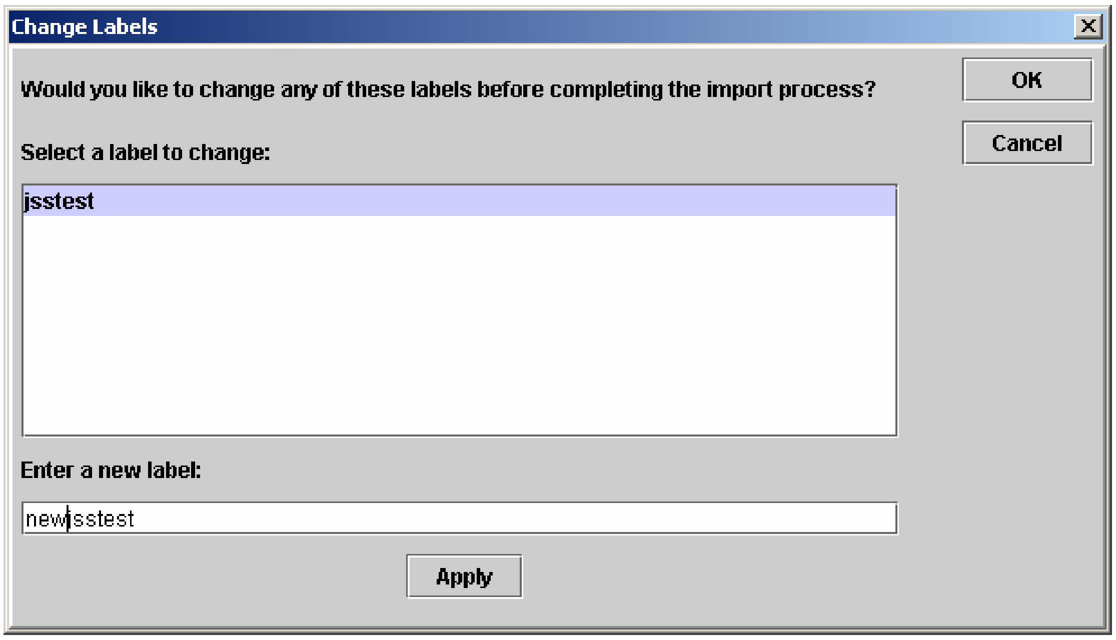
jsstest

Enter a new label:

jsstest

22. The **Change Labels** window will be displayed. This window allows the labels of certificates being imported to be changed if for example, a certificate with the same label already exists in the target key database. Changing certificate labels has no effect on trust chain validation.

Figure 9. Change Labels window with new label entered



Change Labels

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

Select a label to change:

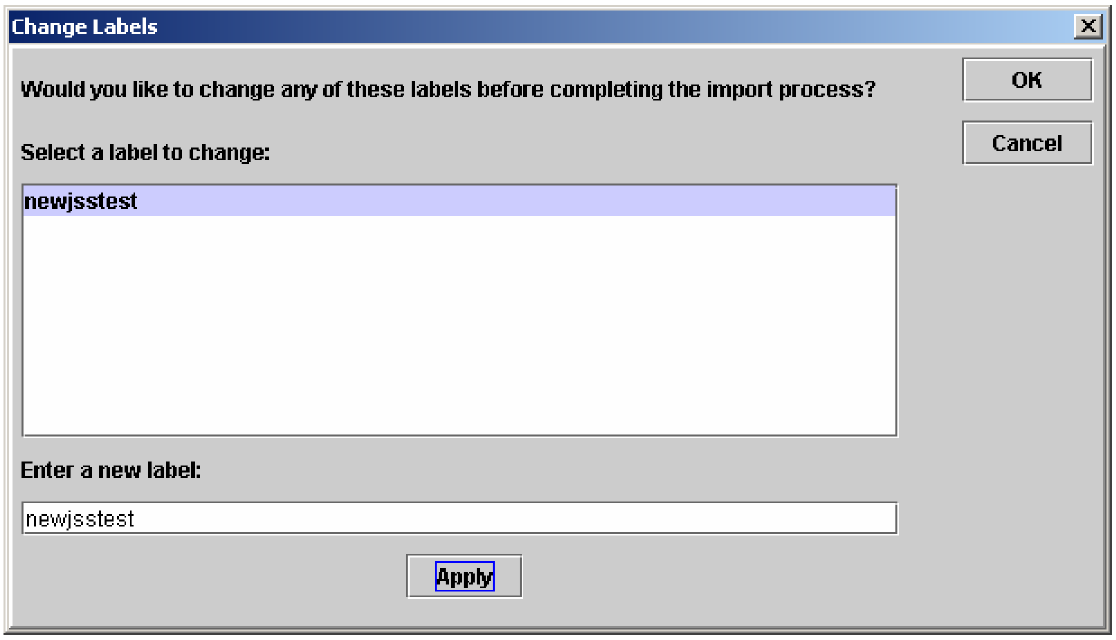
jsstest

Enter a new label:

newjsstest

23. To change a label select the required label from the **Select a label to change**: entry panel. The label will be replicated into the **Enter a new label**: entry line. Replace the label text with that of the new label and click **Apply.**

Figure 10. Change Labels window with new label applied



Change Labels

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

Select a label to change:

newjsstest

Enter a new label:

newjsstest

24. The text in the **Enter a new label:** entry line is replicated back into the **Select a label to change**: panel replacing the originally selected label and so relabelling the corresponding certificate.
25. When all required relabelling is done click **OK**. The **Change Labels** window is now removed and the original **IBM Key Management** window reappears with the **Personal Certificates** and **Signer Certificates** panels updated with the correctly labeled certificates.

    The certificate is now imported to the (target) database.

Scenario 4:

In the previous scenario, a personal, or signer certificate was imported into a key database from another key database with the certificate being relabeled in the process. In this scenario the special case of importing from a Microsoft pfx file is considered.

This is special because a pfx file may contain two certificates relating to the same key. One is a personal or site certificate (contains both a public and private key). The other is a signer certificate (contains only a public key). These certificates cannot co-exist in the same CMS keystore so only one or the other can be imported. As well the “friendly name” or label is attached to only the signer certificate. The personal certificate is identified by a system generated Unique User Identifier or UUID.

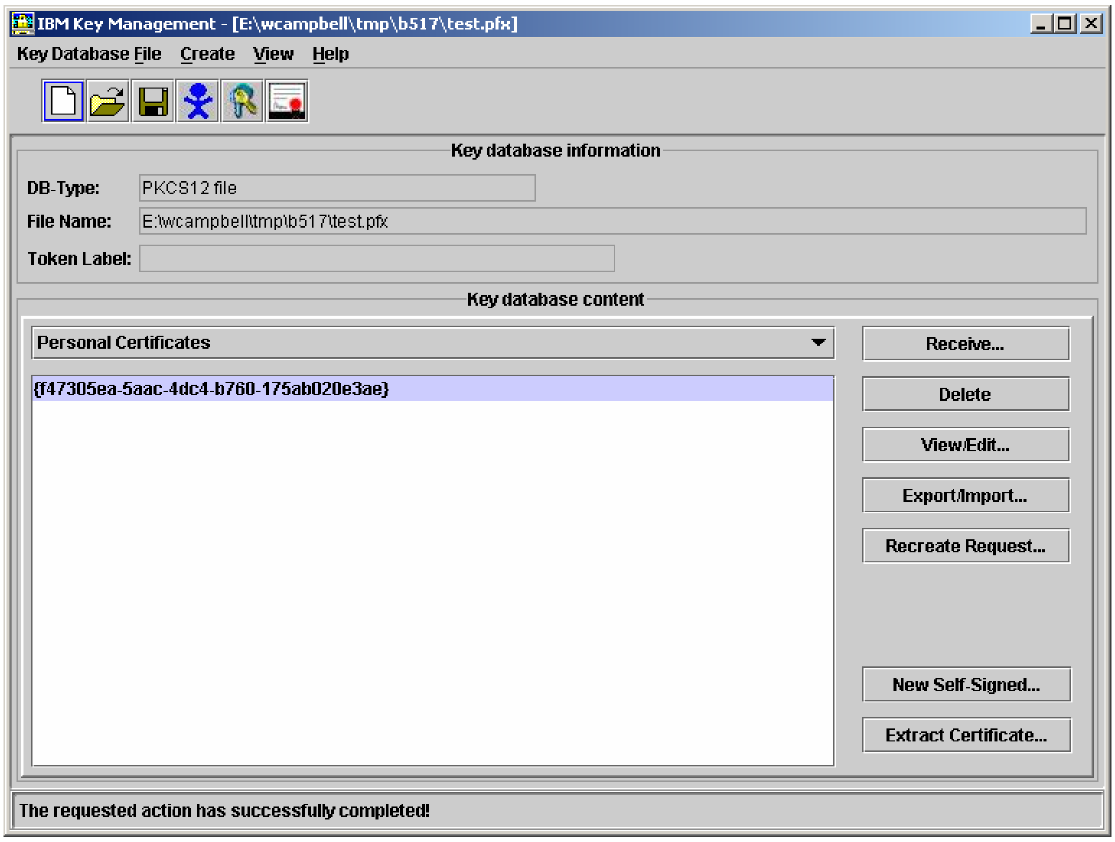
This scenario involves the import of a personal certificate from a pfx file while labeling it with the “friendly name” previously assigned to the signer certificate. The trust chain signer certificates should already be added to the target key database.

To import a personal certificate from a source pfx key database:

Note:PKCS#12 files should be considered temporary and deleted after use.

1. The scenario involves importing a personal certificate to the target key database. To begin, start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select a key database type of **PKCS12**.
4. Select the pfx key database that is intended for import. Click **Open**. The **Password Prompt** window is displayed.

Figure 11. Personal Certificates in PFX keystore



DB-Type: PKCS12 file
File Name: E:\wcampbell\tmp\b517\test.pfx
Token Label:

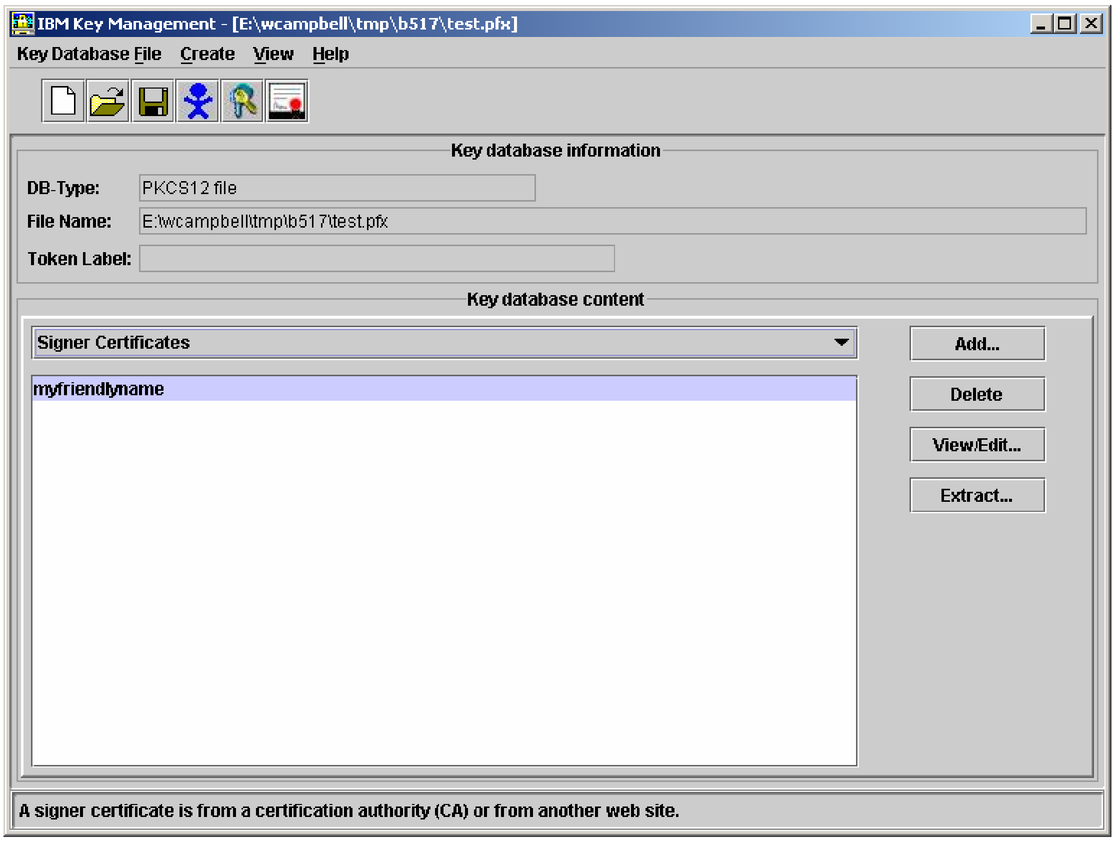
Personal Certificates

{f47305ea-5aac-4dc4-b760-175ab020e3ae}

The requested action has successfully completed!

5. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected pfx key database file, indicating that the file is open and ready.

Figure 12. Signer Certificates in PFX keystore



DB-Type: PKCS12 file

File Name: E:\wcampbell\tmp\b517\test.pfx

Token Label:

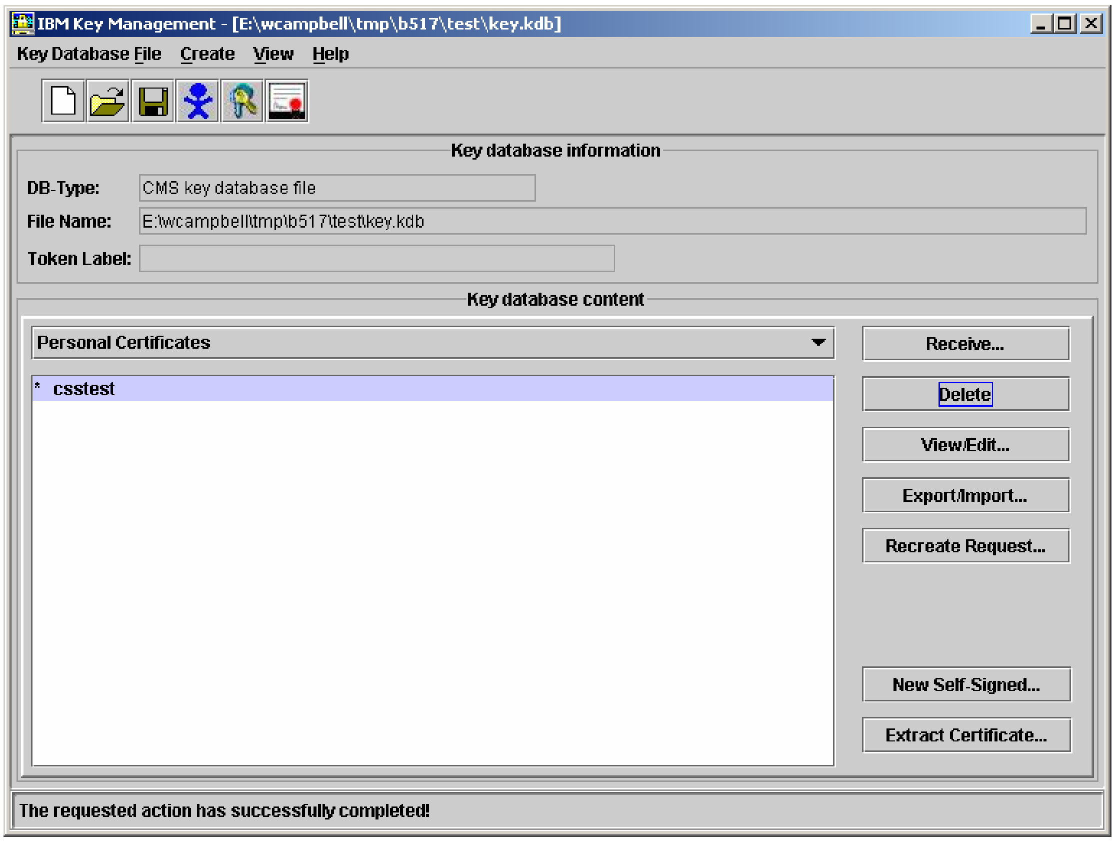
Signer Certificates

myfriendlyname

A signer certificate is from a certification authority (CA) or from another web site.

6. Select **Signer Certificates** from the pulldown list. The “friendly name” of the required certificate should be displayed as a label in the **Signer Certificates** panel.
7. Select the label entry and click **Delete** to remove the signer certificate. The **Confirm** dialog box is displayed.
8. Click **Yes**. The dialog box is removed and the selected label is no longer displayed in the **Signer Certificates** panel.
9. Click **Key Database File** -> **Open**. The **Open** window is displayed.
10. Select the target key CMS database which the pfx file is being imported into. Click **Open**. The **Password Prompt** window is displayed.

Figure 13. Personal Certificates in CMS key database before import


IBM Key Management - [E:\wcampbell\tmp\b517\test\key.kdb]

Key database information

DB-Type: CMS key database file

File Name: E:\wcampbell\tmp\b517\test\key.kdb

Token Label:

Key database content

Personal Certificates

* csstest

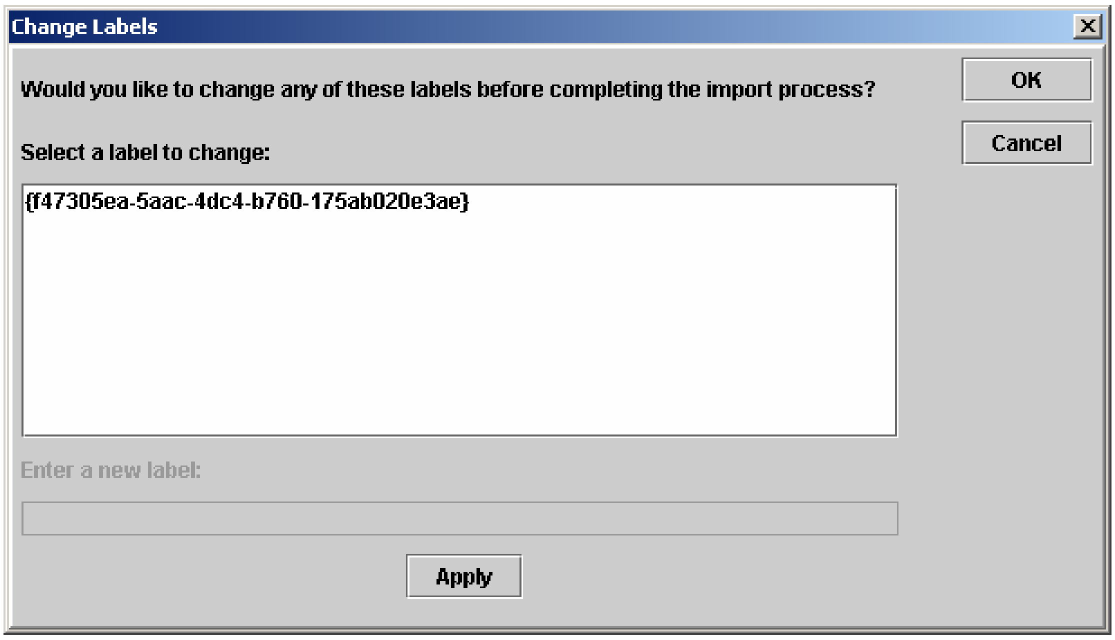
The requested action has successfully completed!

11. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
12. Select **Personal Certificates** from the pulldown list
13. Click **Import** to import keys from the pfx key database. The **Import Key** window is displayed.:

    - Click **Export/Import key**. The **Export/Import key** window is displayed.
    - Select **Import** from **Choose Action Type**.

14. Select the **PKCS12** file.
15. Enter the name of the pfx file as used in Step 4.. Click **OK**. The **Password Prompt** window is displayed.
16. Specify the same password that you specified when you deleted the signer certificate.. Click **OK**.

Figure 14. Change Labels window


Change Labels

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

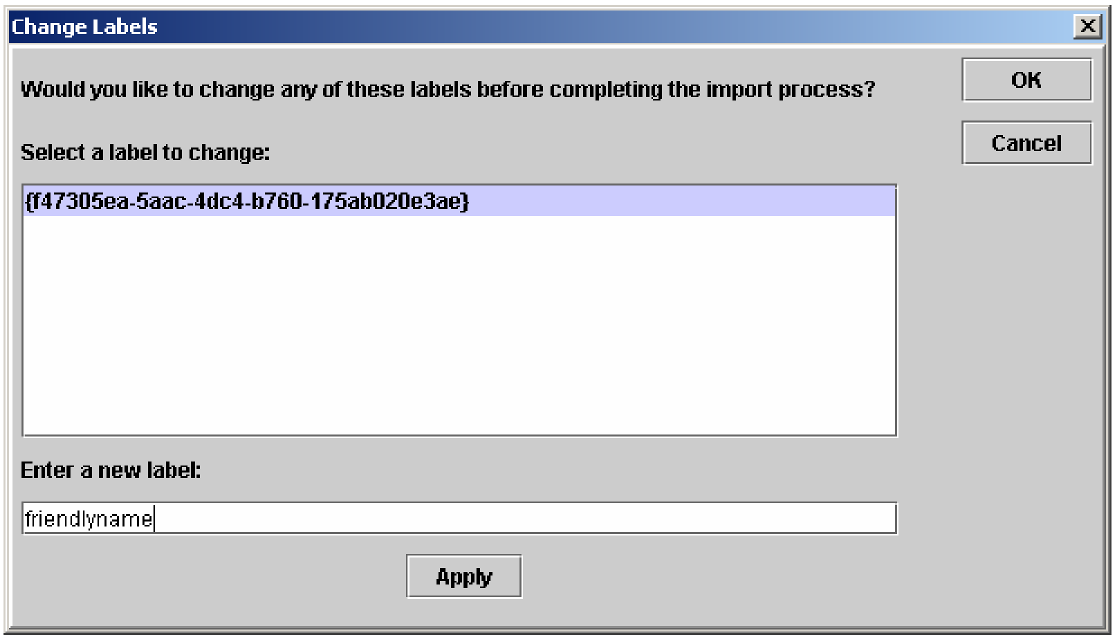
Select a label to change:

{f47305ea-5aac-4dc4-b760-175ab020e3ae}

Enter a new label:

17. The **Change Labels** window will be displayed as there should be only a single certificate available for import. The label of the certificate should be a UUID which has a format "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"

Figure 15. Change Labels window with new label entered


Change Labels

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

Select a label to change:

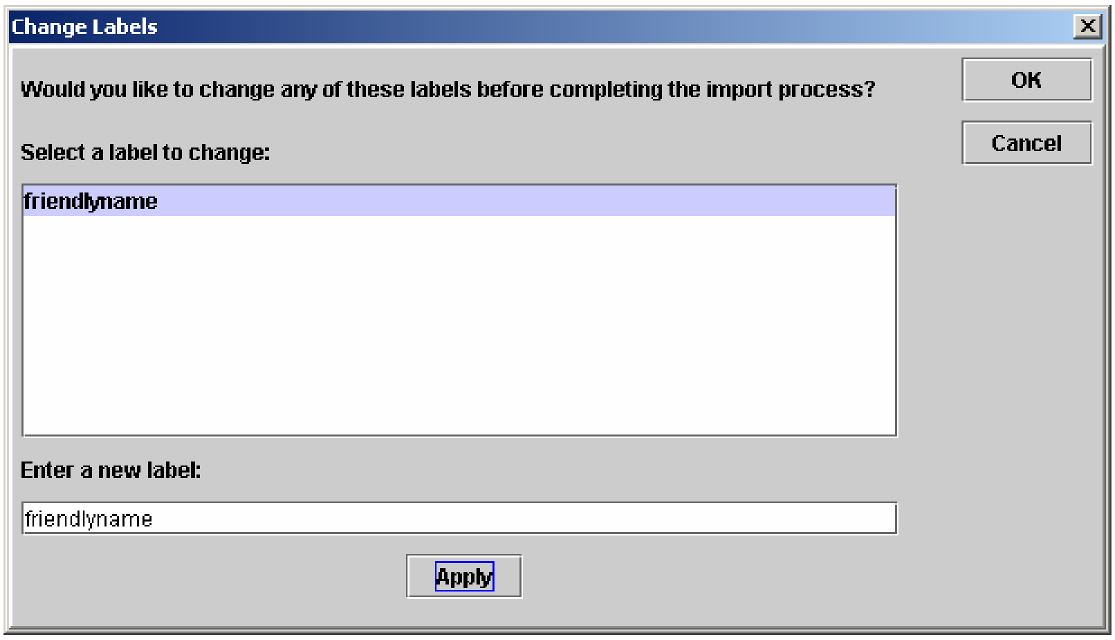
{f47305ea-5aac-4dc4-b760-175ab020e3ae}

Enter a new label:

friendlyname

18. To change the label select the UUID from the **Select a label to change**: entry panel. The label will be replicated into the **Enter a new label**: entry line. Replace the label text with that of the "friendly name" that was deleted in Step 7 and click **Apply.**

Figure 16. Change Labels window with new label applied



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

Select a label to change:

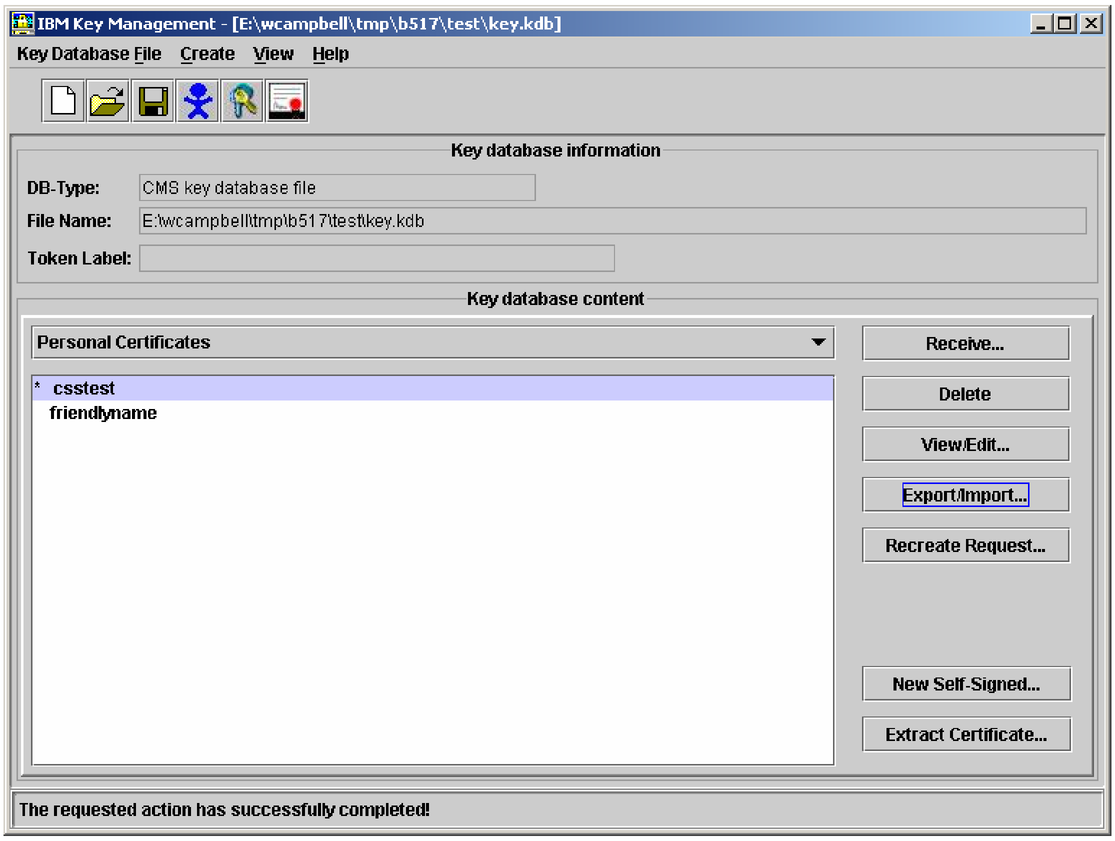
friendlyname

Enter a new label:

friendlyname

19. The text in the **Enter a new label:** entry line is replicated back into the **Select a label to change**: panel replacing the originally selected label and so relabelling the personal certificate with the required "friendly name".

Figure 17. Personal Certificates in CMS key database after import



DB-Type: CMS key database file

File Name: E:\wcampbell\tmp\b517\test\key.kdb

Token Label:

Personal Certificates

* csstest
friendlyname

The requested action has successfully completed!

20. Click **OK**. The **Change Labels** window is now removed and the original **IBM Key Management** window reappears with the **Personal Certificates** and **Signer Certificates** panels updated with the correctly labeled personal certificate.
21. The pfx personal certificate is now imported to the (target) database.

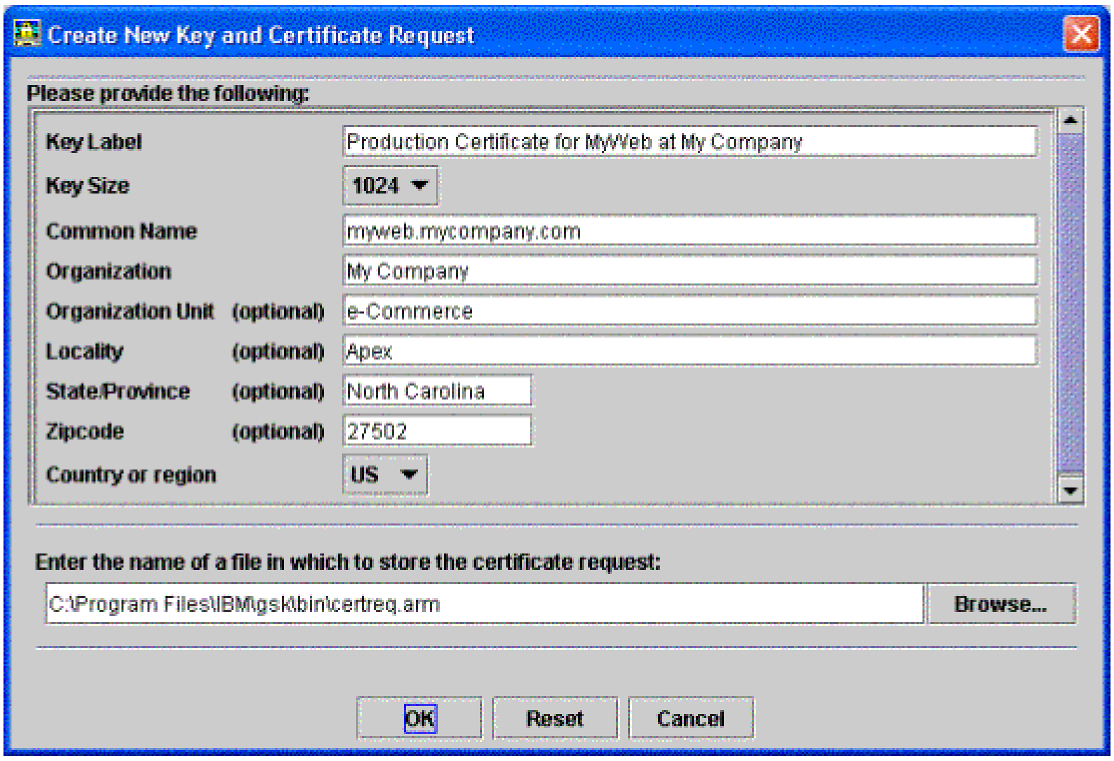
Requesting a digital certificate

A digital certificate is required to run SSL-enabled server code and might be required for client applications. To acquire a digital certificate, generate a request using iKeyman and submit the request to a CA. The CA will verify your identity and send you a digital certificate.

To request a digital certificate for a key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file to generate the request and click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Personal Certificate Requests** from the pulldown list.
6. Click **New**. The **Create New Key and Certificate Request** window is displayed.

Figure 18.7. Create New Key and Certificate Request window


Create New Key and Certificate Request
Please provide the following:
Browse...
OK
Reset
Cancel

Key Label		Production Certificate for MyWeb at My Company
Key Size		1024
Common Name		myweb.mycompany.com
Organization		My Company
Organization Unit	(optional)	e-Commerce
Locality	(optional)	Apex
State/Province	(optional)	North Carolina
Zipcode	(optional)	27502
Country or region		US

Enter the name of a file in which to store the certificate request:

C:\Program Files\IBM\gsk\bin\certreq.arm

7. Enter a **Key Label** for the digital certificate request, for example, `Production Certificate for MyWeb at My Company`.
8. Type a **Common Name** and **Organization**, and select a **Country**. For the remaining fields, either accept the default values, or type or select new values.
9. At the bottom of the window, type a name for the file, such as `certreq.arm`.
10. Click **OK**. A confirmation window is displayed, verifying that you have created a request for a new digital certificate.
11. Click **OK**. The **IBM Key Management** window is displayed. The **Personal Certificate Requests** field shows the key label of the new digital certificate request you created.
12. Send the file to a CA to request a new digital certificate, or cut and paste the request into the request form on the CA's Web site.

Receiving a digital certificate

Once the CA sends you a new digital certificate, you need to add it to the key database you used to generate the request.

To receive a digital certificate into a key database, follow these steps:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file you used to generate the request. Click **Open**. The **Password Prompt** window is displayed.

4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Personal Certificates** from the pulldown list.
6. Click **Receive**. The **Receive Certificate from a File** window is displayed.
7. Click **Data type** and select the data type of the new digital certificate, such as **Base64-encoded ASCII data**. If the CA sends the certificate as part of an e-mail message, then you might need to cut and paste the certificate into a separate file.
8. Type the **Certificate file name** and **Location** for the new digital certificate, or click **Browse** to select the name and location.
9. Click **OK**. The **Enter a Label** window is displayed.
10. Type a label, such as `Production Certificate for MyWeb at My Company`, for the new digital certificate and click **OK**. The **IBM Key Management** window is displayed. The **Personal Certificates** field shows the label of the new digital certificate.

Deleting a digital certificate

If you no longer need one of your digital certificates, you'll want to delete it from your database.

Note:

Before deleting a digital certificate, create a backup copy in case you later want to re-create it.

To delete a digital certificate from a key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file from which to delete the digital certificate and click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Personal Certificates** from the pulldown list.
6. Select the digital certificate you want to delete and click **Delete**. The **Confirm** window is displayed.
7. Click **Yes**. The **IBM Key Management** window is displayed. The label of the digital certificate you just deleted should no longer appear in the **Personal Certificates** field.

Setting a new default key (CMS only)

To make it easy to configure an SSL application, the iKeyman utility lets you specify a default digital certificate. This certificate is used when the key database contains more than one Personal Certificate entry. You might have more than one digital certificate in a database if you started using a self-signed digital certificate in the application (for testing) while waiting for the official digital certificate from your chosen CA.

After receiving the official digital certificate from the CA, you can leave the self-signed digital certificate in the database and begin using the CA-issued digital certificate by making it the default digital certificate. The default digital certificate is indicated by an asterisk (*) in front of the entry label.

Note:

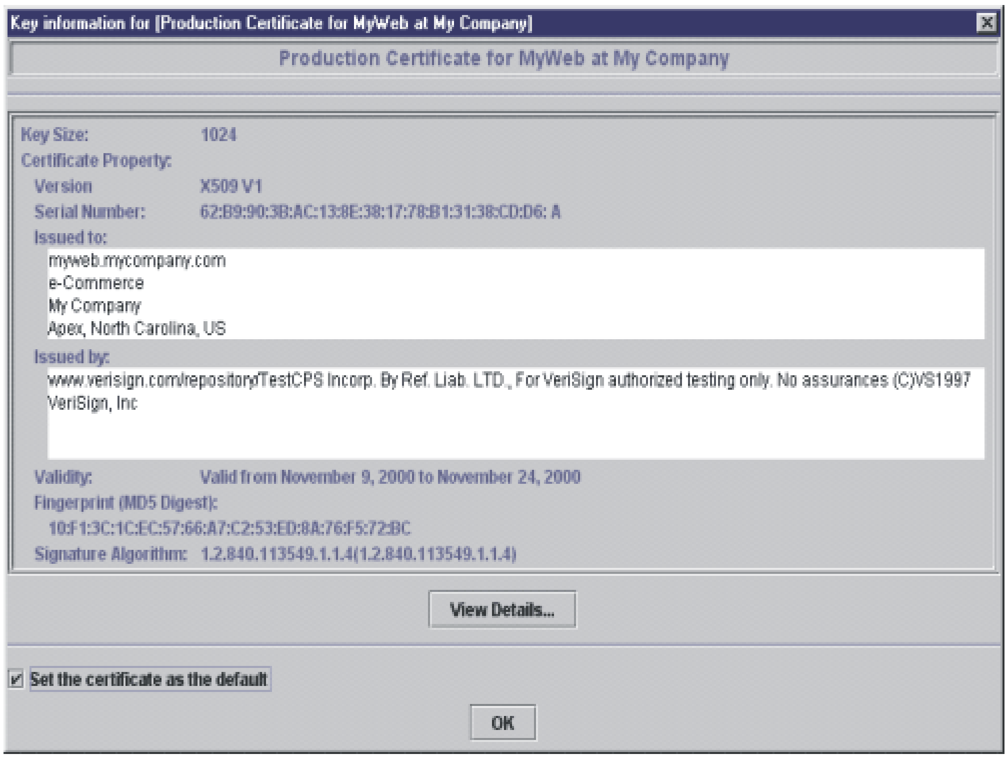
Do not release a production application with a self-signed digital certificate; no browser or client will be able to recognize or communicate with your server.

The first digital certificate that is received or created as self-signed is marked as the default digital certificate. Each time a new digital certificate is received or a self-signed digital certificate is created, you are given the option to make the new certificate the default. The digital certificate specified as the default can be changed at any time.

To change the default digital certificate in a key database:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File** -> **Open**. The **Open** window is displayed.
3. Select the key database file which holds the default digital certificate and click **Open**. The **Password Prompt** window is displayed.
4. Enter the key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Select **Personal Certificates** from the pulldown list. The default digital certificate is indicated by an asterisk (*) in front of the entry label.
6. Select the digital certificate you want to set as the default digital certificate and click **View/Edit**, or double-click on the entry. The **Key Information** window is displayed for the digital certificate entry, as shown in Figure 8.

Figure 19.8. Key Information window



Key information for [Production Certificate for MyWeb at My Company]

Production Certificate for MyWeb at My Company

Key Size: 1024
Certificate Property:
Version X509 V1
Serial Number: 62:B9:90:3B:AC:13:8E:38:17:78:B1:31:38:CD:D6: A
Issued to:
myweb.mycompany.com
e-Commerce
My Company
Apex, North Carolina, US
Issued by:
www.verisign.com/repository/TestCPS Incorp. By Ref. Liab. LTD., For VeriSign authorized testing only. No assurances (C)VS1997
VeriSign, Inc
Validity: Valid from November 9, 2000 to November 24, 2000
Fingerprint (MD5 Digest):
10:F1:3C:1C:EC:57:66:A7:C2:53:ED:8A:76:F5:72:BC
Signature Algorithm: 1.2.840.113549.1.1.4(1.2.840.113549.1.1.4)

7. Select **Set the certificate as the default** and click **OK**. The **IBM Key Management** window is displayed. The label of the digital certificate you just set as the default is identified with an asterisk (*) in front of the entry.

Changing a database password

To change a key database password:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Click **Key Database File -> Open**. The Open window is displayed.
3. Select the key database file requiring the change of password. Click **Open**. The **Password Prompt** window is displayed.
4. Enter the current key database password and click **OK**. The **IBM Key Management** window is displayed. The title bar shows the name of the selected key database file, indicating that the file is open and ready.
5. Click **Key Database File** -> **Change Password**. The **Change Password** window is displayed.
6. Type a new password in the **Password** field, and type it again in the **Confirm Password** field.
7. Click **OK**. A message in the status bar indicates that the request completed successfully.

Using iKeyman to manage smart cards

The iKeyman utility allows you to manage digital certificates on a smart card; that is, on any PKCS11 cryptographic token. Before using iKeyman, you will need to configure the module name for managing your smart card. There are two ways to do this:

1. From a property file
   a. Copy the *ikmuser.sample* file that is shipped with the SSL Toolkit (in the classes directory) to a file called *ikmuser.properties* (also in the classes directory).
   b. Edit the *ikmuser.properties* file and set the `DEFAULT_CRYPTOGRAPHIC_MODULE` property to the name of the module for managing your smart card. For example:

      ```
      DEFAULT_CRYPTOGRAPHIC_MODULE=C:\\Winnt\\System32\\W32pk2
      ig.dll
      ```

      The module is normally installed on your system when you install the software for your smart card. The above `.dll` is used here as an example only.

   c. Save the *ikmuser.properties* file.

      The above configuration steps need only be performed once. Every time the iKeyman utility is started, it will read the contents of your *ikmuser.properties* file.

2. From iKeyman GUI

   You can input the module name directly from the iKeyman GUI.

When the module name has been set in the *ikmuser.properties* file, this name will always be shown as a default in the GUI panel. You can change the name if it is not the preferred module name.

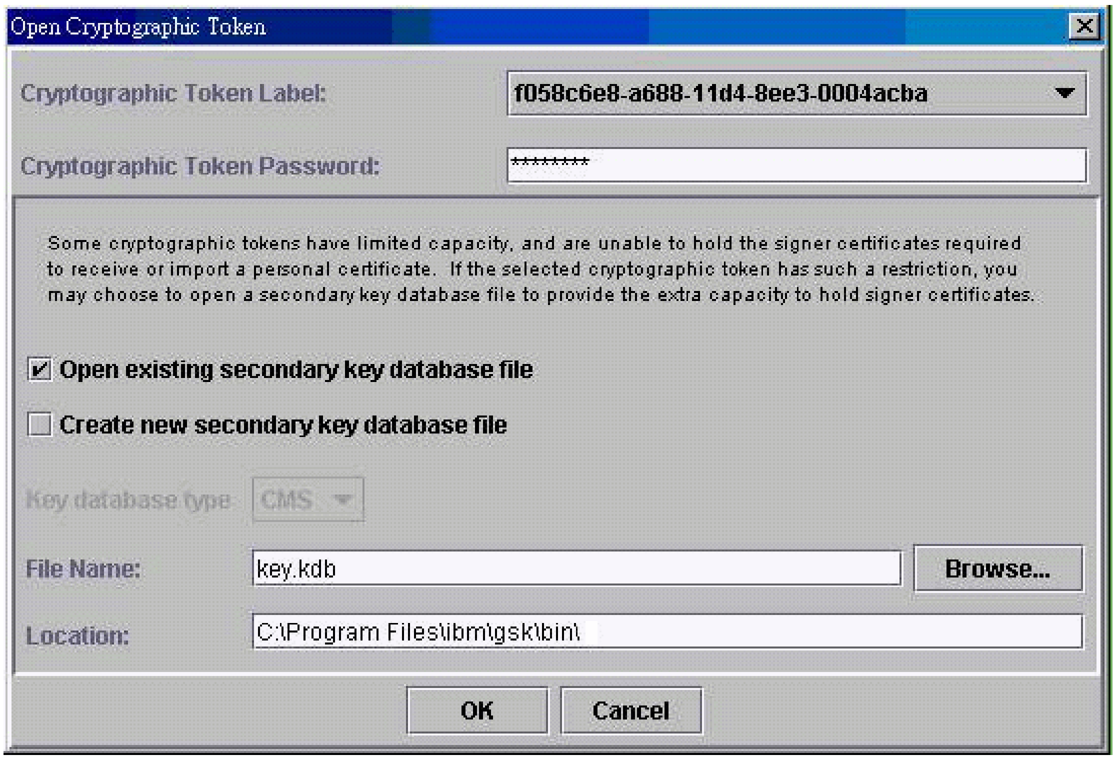
Note:
If using IBM JSSE, edit the `java.security` file in your JDK environment to set the IBM JCE to be a higher priority than CMS.

Managing digital certificates on a smart card

To manage digital certificates on your smart card:

1. Click **Key Database File -> Open**, select **CMS Cryptographic Token** as the key database type, and ensure that the module name is correct. Click **OK**.
2. The **Open Cryptographic Token** window is displayed, an example of which for CMS cryptographic token support is displayed, as shown in Figure 9. You are required to enter information such as the password.

Figure 20.9. Open Cryptographic Token window


Open Cryptographic Token

Cryptographic Token Label: f058c6e8-a688-11d4-8ee3-0004acba

Cryptographic Token Password: ********

Some cryptographic tokens have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If the selected cryptographic token has such a restriction, you may choose to open a secondary key database file to provide the extra capacity to hold signer certificates.

[✓] Open existing secondary key database file

[ ] Create new secondary key database file

Key database type: CMS

File Name: key.kdb

Location: C:\Program Files\ibm\gsk\bin\

Some smart cards have limited capacity, and are unable to hold the signer certificates required to receive or import a personal certificate. If this restriction applies to your smart card, you may open a secondary key database (which is supported by CMS only), in addition to the smart card. The secondary key database serves as a container for the signer certificates associated with receiving and importing personal certificates onto your smart card. If your smart card has sufficient capacity, you may not need to open a secondary key database.

Requesting a digital certificate for a smart card

The steps for requesting a digital certificate for a smart card are the same as those for requesting a digital certificate for a key database. Clicking **Key Database -> Open** and select **CMS Cryptographic Token** for the key database type. After specifying the module name and password for the smart card, proceed with the steps as specified in Requesting a digital certificate.

Adding a digital certificate to a smart card

After the CA sends you a new digital certificate, you need to add it to the same smart card for which you generated the request.

To add a digital certificate to a smart card:

1. Click **Key Database File -> Open**,

2. Select **CMS Cryptographic Token** for the key database type, specify the module name, and click **OK**.

   At this point that you may need to open a secondary key database, if your smart card does not have the capacity to hold the signer certificate(s) associated with the request.

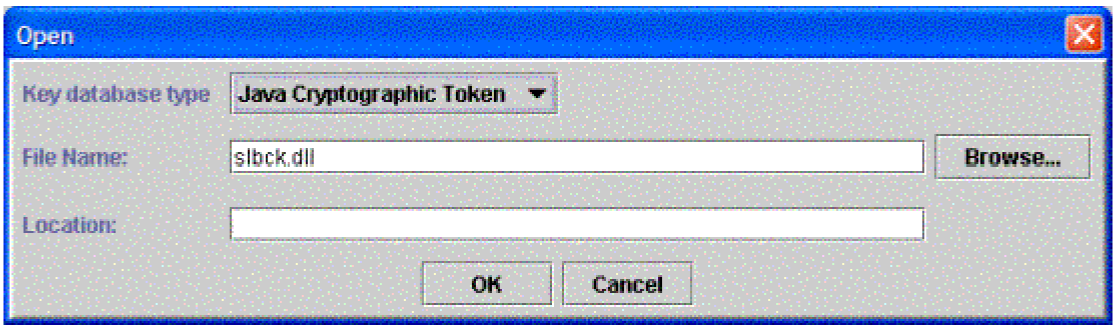
3. Once you have opened the cryptographic token and (optionally) a secondary key database, proceed with the steps as if you had opened a key database -- Receiving a digital certificate.

Opening Cryptographic Tokens using IBM JSSE

The examples above deal with opening a cryptographic token using CMS. To open the cryptographic token using IBM JSSE, do the following:

1. Edit the `java.security` file in your JDK environment to set the IBM JCE to be a higher priority than CMS.
2. Click **Key Database File** then **Open**, select **Java Cryptographic Token** and make sure the module name is correct. The open window is displayed as shown in Figure 10 below:

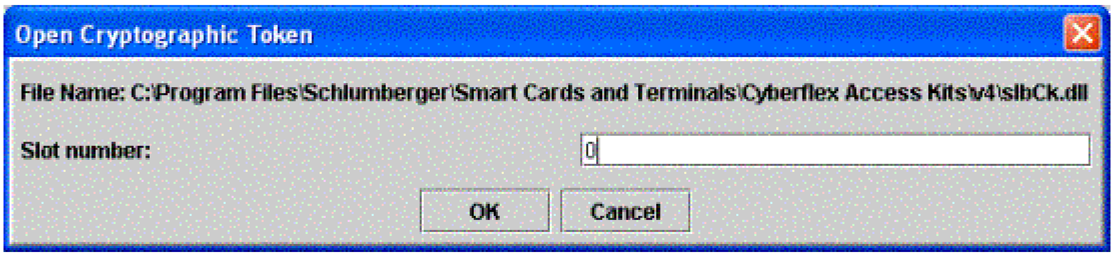
   Figure 21.. Opening a Key Database File using IBM JSSE



Key database type: Java Cryptographic Token
File Name: slbck.dll
Location:

3. Click the **OK** button. The following screen is displayed:

   Figure 22.11. Selecting slot number of cryptographic token



File Name: C:\Program Files\Schlumberger\Smart Cards and Terminals\Cyberflex Access Kits\v4\slbCk.dll
Slot number: 0

4. Enter the slot number and press **OK**.
5. You are then presented with a password dialog box for the cryptographic token. Enter the Smart Card PIN number and click **OK**.

Opening Cryptographic Tokens using MS CryptoAPI

To open the cryptographic token using MS CryptoAPI:

1. Click the **Key Database File** -> **Open**.
2. Select **CMS Cryptographic Token** and ensure the module name is correct. Enter the cryptographic service provider and click **OK**.
3. A password window is displayed. Enter the password for the cryptographic token.

Storing the encrypted database password in a stash file

For a secure network connection, store the encrypted database password in a stash file. To do this:

1. Start iKeyman. The **IBM Key Management** window is displayed.
2. Select **Key Database File -> Stash File**. An information dialog is displayed advising that the password was stashed and its location. Click **OK**.

Enabling FIPS mode

To use FIPS certified cryptography the following steps should be taken.

1. Ensure the JVM being used has been enabled to support FIPS cryptography . To do this review and revise the JVM configuration as discussed in the *Environment Requirements and Configuration* section of *Global Security Kit Install and Packaging Guid, Version 7*.
2. Modify the appropriate properties file to add the following properties

   a. DEFAULT_FIPS_MODE_PROCESSING=ON
   b. DEFAULT_CRYPTOGRAPHIC_BASE_LIBRARY=ICC
   c. DEFAULT_SIGNATURE_ALGORITHM=SHA1_WITH_RSA

Note: Property (c) may be excluded altogether from the file in which case the default will be selected according to the value of property (a). If DEFAULT_FIPS_MODE_PROCESSING is set to “off” then the default algorithm is “MD5_WITH_RSA”. If on then the default will be “SHA1_WITH_RSA”.

Example iKeyman usage scenarios

This section presents some usage scenarios that demonstrate how a typical user may prepare the necessary PKI keystores to SSL enable their application.

Scenario 1:

Kay, a Systems Administrator at Anchor Shipbuilders has been asked to enable SSL communication between their public Web site and customer browsers. To do this, Kay must obtain a PKI certificate for her company. Kay has been told that she may purchase this certificate from Harry's Web Certificates. Harry's Web Certificates' own public certificate is signed by Verisign's root certificate. In order to create a CMS certificate keystore for use in her SSL installation, Kay must:

- Create a CMS keystore.
- Obtain the Verisign root certificate.
- Obtain Harry's Web Certificates' certificate.
- Obtain a certificate for her company.
- Add the Verisign's root certificate, Harry's Web Certificates certificate, and her newly obtained certificate to the keystore.
- Configure her Web server to use her newly created certificate in the CMS key store (this is application specific and beyond the scope of this scenario).

The steps taken are:

1. Kay follows the instructions in, Creating a key database, to create a new CMS keystore. Once complete, Kay notices that the Verisign root certificate is already present in the the new keystore (iKeyman ships various root certificates for customer convenience). If this was not the case then Kay would need to obtain the Verisign root certificate from Verisign and follow the instructions in Adding a CA root digital certificate to add it to her newly created keystore.
2. Kay visits the Harry's Web Certificates' Web site and downloads their public certificate as an `*.arm` file. She then follows the instructions in Adding a CA root digital certificate to add it to her newly created keystore.
3. Following the instructions in Requesting a digital certificate Kay creates a certificate request and sends it to Harry's Web Certificates for signing. Sometime later her newly signed certificate is returned and she follows the instructions in Receiving a digital certificate to add her company's new certificate to the keystore.
4. Kay wishes to ensure that her new certificate is the default one used by her Web installation. She reads the section Setting a new default key (CMS only) and ensures that her new certificate is the default.
5. Kay consults her Web server documentation to determine how to enable it for SSL.
6. Kay devises a plan to periodically examine the expiry status of the certificates in her keystore. The expiry dates of certificates can be determined using the command detailed in Listing Expired Certificates.

Scenario 2

Rob works for a small company that wishes to evaluate the benefits of using SSL to secure the network traffic between their main test servers and the company LDAP server. Rob has been told that for this evaluation he must not spend any money on certificates and that the keystore he creates must be as small as possible. Rob discovers that for the evaluation he can use a self-signed certificate. He realizes though, and informs his employer, that the self-signed certificate will be for evaluation testing only and should they proceed into production they must purchase a certificate from a reliable Certificate Authority.

In order to create a CMS certificate keystore for use in the evaluation Rob must:

- Create a CMS keystore.
- Create a self-signed certificate.
- Take steps to minimize the size of the keystore.

The steps taken are:

1. Rob follows the instructions in Creating a key database to create a new CMS keystore.
2. Rob follows the instructions in Creating a self-signed digital certificate for testing.
3. Rob follows the instructions in Deleting a CA root digital certificate to delete all the Signer certificates in the keystore. Rob now knows that he has a minimum size keystore to use in his evaluation.
4. Rob uses his newly created keystore during his evaluation.
5. Rob includes as part of his evaluation a plan to periodically examine the expiry status of certificates in their keystores. Certificates about to expire in a given number of days can be listed by following the instructions in Listing Expired Certificates.

Scenario 3

Jan works for a medium-size company that has just purchased another smaller organization. Jan has been asked to import the certificates from the smaller company's OpenSSL keystore into the company standard CMS format. Jan decides to create a new CMS keystore and add the certificates from the OpenSSL store to it. Jan does not like to use a GUI tool and decides to use the IKEYCMD command line tool to perform the operation. In order to import the OpenSSL certificates into a new CMS keystore Jan must:

- Create a new CMS keystore.
- Add the root certificate from OpenSSL.
- Export the certificates from the OpenSSL key store.
- Import the OpenSSL certificates into the CMS keystore.

The steps taken are:

1. Jan creates the new CMS keystore using the command:

   ```
   gsk7cmd -keydb -create -db test.kdb -pw jjj -type cms
   ```

2. Jan adds the test root certificate from OpenSSL. This must be done before other certificates can be added (command entered as one line):

3.
   ```
   gsk7cmd -cert -add -db test.kdb -pw jjj -label
   OpenSSLroot -format binary -trust enable -file
   OpenSSLrootca.crt
   ```

4. Jan exports the OpenSSL Server `pem` files to a `pkcs12` file using OpenSSL (command entered as one line):

5.
   ```
   openssl pkcs12 -export -in OpenSSL_server_cert.pem -inkey
   OpenSSL_server_key.pem -out test.p12
   ```

6. Jan imports the `pkcs12` file (command entered as one line):

7.
   ```
   gsk7cmd -cert -import -file test.p12 -pw jjj -type pkcs12
   -target test.kdb -target_pw jjj -target_type cms
   ```

8. Jan devises a plan to periodically examine the expiry status of the certificates in her keystore. The expiry dates of certificates can be determined using the command detailed in Listing Expired Certificates.

Using the IKEYCMD Command Line Interface

IKEYCMD is a tool, in addition to iKeyman, that can be used to manage keys, certificates and certificate requests. It is functionally similar to iKeyman though it employs command line syntax and doesn't use a graphical user interface. It can be called from native shell scripts and programs for applications that prefer to add custom interfaces to certificate and key management tasks. It can create all types of key database files that iKeyman currently supports. It can create certificate requests, import CA signed certificates and manage self-signed certificates. It is Java-based and supported on all of the platforms that the SSL toolkit supports.

Use IKEYCMD for configuration tasks related to public-private key creation and management. You cannot use IKEYCMD for configuration options that update the server configuration file, *httpd.conf*. For options that update the server configuration file, you must use the IBM Administration Server.

IKEYCMD uses the Java language and native command line invocation, enabling IKEYMAN task scripting.

Environment set up for IKEYCMD command line interface

IKEYMAN can be run with the **gsk7cmd** command (a UNIX script, or Windows executable, which invokes the Java virtual machine with the correct parameters). Alternatively, you can use the Java command to run the class directly.

The preferred way of using the IKEYMAN command line interface is to use **gsk7cmd**. To run IKEYCMD using **gsk7cmd**, set up environmental variables as follows:

1. Set your PATH to the location of your Java or JRE executable:

   UNIX
   ```
   export PATH=/opt/IBMJava/bin:/usr/local/ibm/gsk7/bin:$PATH
   ```
   Windows
   Right click on *My Computer*. Select *Properties*. Click on the *Advanced* tab. Click on the *Environment Variables* button. In the *System Variables* section, create the new entry:
   ```
   JAVA_HOME=C:\Program Files\IBM\Java
   ```
   and modify the existing PATH entry:
   ```
   PATH=<existingentries>;C:\Program Files\IBM\GSK7\bin;C:\Program
             Files\IBM\Java\bin;
   ```

   Once completed, **gsk7cmd** should run from any directory. To run a **gsk7cmd** command, use the following syntax:

   ```
   gsk7cmd command
   ```

(where *command* is the required command name).

2. To run IKEYCMD by specifying the class directly, set up environment variables to use the IKEYCMD command line interface as follows:
   a. Set your PATH to the location of your Java or JRE executable:

   UNIX

```
export PATH=/opt/IBMJava/bin:$PATH
```

   Windows

   Right click on *My Computer*. Select *Properties*. Click on the *Advanced* tab. Click on the *Environment Variables* button. In the *System variables* section, edit the entry:

```
PATH=<existingentries>;C:\Program Files\IBM\Java\bin;C:\Program
   Files\IBM\GSK7\bin;C:\Program Files\IBM\GSK7\lib;
```

   b. Set the following CLASSPATH environment variable:

   UNIX

   Entered as one line:

```
export
CLASSPATH=/usr/local/ibm/gsk7/classes/cfwk.zip:/usr/local/ibm
         /gsk7/classes/gsk7cls.jar:$CLASSPATH
```

   Note:

   Depending on your UNIX you may require more settings in CLASSPATH environment variable. If necessary, look at the **gsk7cmd** script to determine these.

   Windows

   Right click on *My Computer*. Select *Properties*. Click on the *Advanced* tab. Click on the *Environment Variables* button. In the *System variables* section, create the new entry (entered as one line):

```
CLASSPATH=C:\Program Files\IBM\GSK7\classes\cfwk.zip;C:\Program
Files\IBM\GSK7\classes\gsk7cls.jar;<existingentries>
```

Once completed, IKEYCMD should run from any directory. To run an IKEYCMD command, use the following syntax:

```
 java com.ibm.gsk.ikeyman.ikeycmd command
```

Notes:

1. You can substitute `jre` for `java`, depending on whether you are using a JRE or JDK. For example:

```
jre com.ibm.gsk.ikeyman.ikeycmd <command>
```

2. If, as described in the Installation Manual, you are using IBM JDK 1.4.1 and the `gskikm.jar` has not been removed from the IBM JDK directory tree you will always use the IBM JDK Ikeyman (no matter how you setup your path or CLASSPATH). This is because a different version of Ikeyman is packaged with IBM JDK 1.4.1.
3. The setup above assumes you are using the 32-bit version of the Java VM and SSL Toolkit. If installed, the 64-bit version of the Ikeyman command line

interface can be invoked using **gsk7cmd_64**, but the environment variables need to include the path to the 64-bit libraries and executables.

IKEYCMD command line syntax

The syntax of the Java CLI is:

```
gsk7cmd [-Dikeycmd.properties=<properties_file>] <object> <action>
[options]
```

where:

-Dikeycmd.properties
: Specifies the name of an optional properties file to use for this Java invocation. A default properties file, *ikeycmd.properties*, is provided as a sample file that can be modified and used by any Java application.
Note: Properties in this file override those in the *ikminit.properties* file.

object
: Is one of the following:
-keydb
Actions taken on the key database (either a CMS key database file, a WebDB keyring file, or SSLight class)
-version
Displays version information for IKEYCMD
-cert
Actions taken on a certificate
-certreq
Actions taken on a certificate request
-help
Displays help for the IKEYCMD invocations

action
: The specific action to be taken on the object, and *options* are the options, both required and optional, specified for the *object* and *action* pair. For a complete set of options, see IKEYCMD command line options overview. For a complete set of actions allowed for a given object, see IKEYCMD command line parameter overview.
Note:
The *object* and *action* keywords are positional and must be in the specified order. However, *options* are not positional and can be in any order, provided that they are specified as an option and operand pair.

-cert
: Actions taken on a certificate

-certreq
: Actions taken on a certificate request

-Dikeycmd.properties
: Specifies the name of an optional properties file to use for this Java invocation. A default properties file, *ikeycmd.properties*, is provided as a sample file that can be modified and used by any Java application.

-help

Displays help for the IKEYCMD invocations

Error code returns

By default the command line interface returns a boolean result - 0 or true for success and 1 or false for failure. If a more detailed result is required then the option “-usereasoncode” may be included in any command line option string. This option causes the command line to return a multi-valued result – 0 or true for success and an integer in the range 1-255 for failure. The interpretation of this value is described in the Error Codes Appendix.

This return code can be accessed from a shell by entering the following –

For Unix: echo $?

For Windows: echo %ERRORLEVEL%

User interface task reference

IKEYCMD command line interface tasks are summarized in the following table:

IKEYMAN and IKEYCMD task	**For instructions, see:**
Supporting no password	Supporting no password
Create a new key database and specify the database password	Creating a new key database
Create a new key pair and certificate request	Creating a new key pair and certificate request
Create a self-signed certificate	Creating a self-signed certificate
Export a key to another database or PKCS12 file	Exporting keys
Import a key from another database or PKCS12 file	Importing keys
List certificate authorities (CAs) and certificate requests	Listing CAs

IKEYMAN and IKEYCMD task	**For instructions, see:**
Open a key database	Opening a key database
Receive a CA-signed certificate into a key database	Receiving a CA-signed certificate
Managing a digital certificate on a smart card	Managing a digital certificate on a smart card
Show the default key in a key database	Showing the default key in a key database
Listing expired certificates	Listing Expired Certificates
Showing the entire certificate	Showing the entire certificate
Store the root certificate of a CA	Storing a CA certificate
Store the encrypted database password in a stash file	Storing the encrypted database password in a stash file

Supporting no password

The Certificate Management System (CMS) key database supports no-password operation. That is, the **-pw** parameter can be optional. You can specify this information using the `-Dikeycmd.properties` file. If you want to create a CMS key database without a password, you must add `DEFAULT_CMS_PASSWORD_REQUIRED=false` to your `ikeycmd.properties` file and type the following on one line:

```
gsk7cmd -Dikeycmd.properties=<properties file> -keydb
 -create -db <filename> -type cms -expire <days> -stash
```

where:

-create
    Creates the database.
-db <*filename*>
    Name of the database.

-expire <*days*>
Days before password expires. This parameter is only valid for CMS key databases.

-keydb
Specifies the command is for the key database.

<*properties file*>
Contains the information if the **-pw** parameter is needed. For a CMS key database, use **DEFAULT_CMS_PASSWORD_REQUIRED**.

-stash
Stashes the password for the key database. When the **-stash** option is specified during the key database creation, the password is stashed in a file with a filename built as follows: `<filename of key database>.sth`

This parameter is only valid for CMS key databases. For example, if the database being created is named *keydb.kdb*, the stash filename is *keydb.sth*.

Note:
Stashing the password is required for the IBM HTTP Server.

-type cms
The "no password" operation is only supported with CMS.
Note:
IBM HTTP Server only handles a CMS key database.

Creating a new key database

A key database is a file that the server uses to store one or more key pairs and certificates. You can use one key database for all your key pairs and certificates, or create multiple databases.

To create a new key database using the IKEYCMD command line interface, enter the following command (as one line):

```
gsk7cmd -keydb -create -db <filename> -pw <password> -type
              <cms | jks | jceks | pks12> -expire <days> -stash
```

where:

-db <*filename*>
Name of the database.

-expire <*days*>
Days before password expires. This parameter is only valid for CMS key databases.

-keydb
Specifies the command is for the key database.

-pw <password>
The password to access the key database.

-type <cms | jks | jceks | pkcsk>
The database type.
Note:
IBM HTTP Server only handles a CMS key database.

-stash

Stashes the password for the key database. When the **-stash** option is specified during the key database creation, the password is stashed in a file with a filename built as follows: `<`*`filename_of_key_database`*`>.sth`

This parameter is only valid for CMS key databases. For example, if the database being created is named `keydb.kdb`, the stash filename is `keydb.sth`.

Note:

Stashing the password is required for the IBM HTTP Server.

Setting the database password

When you create a new key database, you specify a key database password. This password protects the private key. The private key is the only key that can sign documents or decrypt messages encrypted with the public key. It is good practice to change the key database password frequently.

Use the following guidelines when specifying the password:

- The password must be from the U.S. English character set.
- The password should be at least six characters and contain at least two nonconsecutive numbers. Make sure the password does not consist of publicly obtainable information about you, such as the initials and birth date for you, your partner, or children.
- Stash the password.

Note:
For IBM HTTP Server, keep track of expiration dates for the password. If the password expires, a message is written to the error log. The server will start, but there will not be a secure network connection if the password has expired.

Changing the database password

To change the database password, type (as one line):

```
gsk7cmd -keydb -changepw -db <filename>.kdb -pw <password> -new_pw
             <new_password> -expire <days> -stash
```

where:

-db <*filename*>
Name of the database.
-changepw
Changes the password.
-keydb
Specifies the command is for the key database
-new_pw <new_password>
New key database password; this password must be different than the old password and this password cannot be a NULL string.
-pw <password>
The password to access the key database.
-expire <*days*>
Days before password expires. This parameter is only valid for CMS key databases.
-stash
Stashes the password for the key database. This parameter is only valid for CMS key databases.
Note:
Stashing the password is required for the IBM HTTP Server.

Display the expiry of a database password

To display the expiry date of a password for a CMS key database, type (as one line):

```
gsk7cmd -keydb -expiry -db <filename>.kdb -pw <password>
```

where:

-keydb
Specifies the command is for the key database

-expiry
Request to display the date that the password will expire.

-db <*filename*>
Name of the database.

-pw <password>
The password to access the key database.

IMPORTANT: This option is only supported for CMS key databases. An expiry of 0 means that the password associated with the key database does not expire.

Creating a new key pair and certificate request

To create a public-private key pair and certificate request:

1. Enter the following command (as one line):

```
gsk7cmd -certreq -create -db <filename> -pw <password> -label
<label>
   -dn <distinguished_name> -size <1024 | 512 | 2048> -file
<filename> -san_dnsname <DNS name value>[,<DNS name value>] –
san_emailaddr <email address value>[,<email address value>] –
san_ipaddr <IP address value>[,<IP address value>]
```

where:

-certreq
Specifies a certificate request
-create
Specifies a create action
-db <*filename*>
Name of the database
-pw <*password*>
The password to access the key database
-label <*label*>
Label attached to certificate or certificate request
-dn <*distinguished_name*>

Enter an X.500 distinguished name. This is input as a quoted string of the following format (only CN is required): CN=common_name, O=organization, OU=organization_unit, L=location, ST=state, province, C=country. For example: `"CN=weblinux.raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,L=RTP,ST=NC,C=US"`

-size <1024 | 512 | 2048>

Key size of 512, 1024 or 2048

-file <*filename*>

Name of file where the certificate request will be stored

-san_* <subject alternate name attribute value> [,<subject alternate name attribute value>]

These options are only valid if the line DEFAULT_SUBJECT_ALTERNATE_NAME_SUPPORT =true is entered in the *ikminit.properties* file. The * can have the following values

dnsname : The value must be formatted using the "preferred name syntax" according to RFC 1034. An example is "test.mine.ibm.com".

emailaddr : The value must be formatted as an "addr-spec" according to RFC 822. An example is myname@test.mine.ibm.com.

ipaddr : The value is a string representing an IP address formatted according to RFC 1338 and RFC 1519. An example is "192.168.100.112"

The values of these options are accumulated into a Subject Alternate Name extended attribute which is itself accumulated into an extendedRequest attribute. The resulting attribute is added to the certificate request.
Note: It is not mandatory that a signing CA recognise the extendedRequest attribute or its contents. If it is ignored then the Subject Alternate Name extended attribute will not appear in the signed certificate.

2. Verify that the certificate was successfully created.
   a. View the contents of the certificate request file you created.
   b. Make sure the key database recorded the certificate request:

   ```
   gsk7cmd -certreq -list -db <filename> -pw <password>
   ```

   You should see the label listed that you just created.

3. Send the newly created file to a certificate authority.

Creating a self-signed certificate

It usually takes two to three weeks to get a certificate from a well-known CA. While waiting for an issued certificate, use IKEYMAN to create a self-signed server certificate to enable SSL sessions between clients and the server. Use this procedure if you are acting as your own CA for a private Web network.

To create a self-signed certificate, enter the following command (as one line):

```
gsk7cmd -cert -create -db <filename> -pw <password> -size <1024 |
512 | 2048> -dn
 <distinguished name> -label <label> -default_cert <yes | no> -
expire <days> -san_dnsname <DNS name value>[,<DNS name value>] –
```

```
san_emailaddr <email address value>[,<email address value>] –
san_ipaddr <IP address value>[,<IP address value>] [-ca <true |
false]
```

where:

-cert
Specifies a self-signed certificate.

-create
Specifies a create action.

-db <*filename*>
Name of the database.

-pw <password>
The password to access the key database.

-dn <*distinguished name*>
Enter an X.500 distinguished name. This is input as a quoted string of the following format (only CN is required): CN=common_name, O=organization, OU=organization_unit, L=location, ST=state, province, C=country. For example: `"CN=weblinux.raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,L=RTP,ST=NC,C=US"`

-label <*label*>
A descriptive comment used to identify the key and certificate in the database.

-size
Key size 512, 1024, or 2048

-default_cert <yes | no>
Whether this is the default certificate in the key database.

-expire <days>
The default validity period for new self-signed digital certificates is 365 days. The minimum is 1 day. The maximum is 7300 days (twenty years).

-san_* <subject alternate name attribute value> [,<subject alternate name attribute value>]
These options are only valid if the line DEFAULT_SUBJECT_ALTERNATE_NAME_SUPPORT =true is entered in the ikminit.properties file. The * can have the following values
dnsname : The value must be formatted using the "preferred name syntax" according to RFC 1034. An example is "test.mine.ibm.com".
emailaddr : The value must be formatted as an "addr-spec" according to RFC 822. An example is "myname@test.mine.ibm.com".
ipaddr : The value is a string representing an IP address formatted according to RFC 1338 and RFC 1519. An example is "192.168.100.112".

The values of these options are accumulated into the Subject Alternate Name extended attribute of the generated certificate. If the options are not used then this extended attribute is not added to the certificate.

-ca <true | false>
Add the Basic Constraint extension to the self-signed certificate. The extension will be added with a CA:true and PathLen:<max int> if the value passed is true or not added at all if the value passed is false.

Exporting keys

To export keys to another key database, or to export keys to a PKCS12 file, enter the following command:

```
gsk7cmd -cert -export -db <filename> -pw <password> -label <label>
       -type <cms | jks | jceks | pkcs12> -target <filename> -
target_pw <password>
       -target_type <cms | jks | jceks | pkcs12>
```

where:

-cert
: Specifies a personal certificate.

-export
: Specifies an export action.

-db <*filename*>
: Name of the database.

-pw <password>
: The password to access the key database.

-label <label>
: Label attached to the certificate.

-target <filename>
: Destination file or database. If the **target_type** is JKS, CMS or JCEKS, the database specified here must exist.

-target_pw
: Password for the target key database.

-target_type <cms | jks | jceks | pkcs12>
: The type of database specified by the -target operand.

-type <cms | jks | jceks | pkcs12>
: The type of database key.

Importing keys

To import certificates from another key database, enter the following command:

```
gsk7cmd -cert -import -db|-file> <filename>
-pw <password> -label <label>] [> -type <cms | JKS | JCEKS | pkcs12>
-new_label <label>
-target <filename> -target_pw <password> -target_type <cms | JKS |
JCEKS | pkcs12>] [-pfx]
```

where:

-cert
: Specifies a certificate.

-import
: Specifies an import action.

-db <*filename*>
: Name of the database.(defaults to CMS type)

-file <*filename*>

Name of the database (defaults to PKCS12 type)

-pw <*password*>
    The password to access the key database.

-label <*label*>
    Label attached to the certificate.

-new_label <*label*>
    Re-labels certificate in target key database.

-type <cms | JKS | JCEKS | pkcs12>
    Type of the database.

-target <*filename*>
    Destination database.

-target_pw <*password*>
    Password for the key database if -target specifies a key database.

-target_type <cms | JKS | JCEKS | pkcs12>
    Type of the database specified by -target operand.

-pfx
    Imported file in Microsoft .pfx format. PKCS12 files generated from a Microsoft KeyStore often contain a keypair object and a signer certificate object both with the same public key. Only the signer certificate has a friendly name. With this option only the keypair object will be imported to the target keystore and it will be labeled with the friendly name from the signer certificate. An attempt to import both objects will fail since a given key can only be represented once in a CMS keystore.

To import all keys from a PKCS12 file, enter the following command:

```
gsk7cmd -cert -import -file <filename> -pw <password> -type pkcs12 –
-target -pfx
       <filename> -target_pw <password> -target_type <cms | JKS |
JCEKS | pkcs12>
```

If the source file has been created from a Microsoft Certificate Store then the `–pfx` parameter may be added.

To import a specific key from any keystore to another keystore, enter the following:

```
gsk7cmd –cert –import –db <filename> -pw <password> -type <cms | JKS
| JCEKS | PKCS12> -label <source label> -target <filename> -
target_pw <password> -target_type <cms | JKS | JCEKS | PKCS12>
```

If the certificate is to be relabeled in the target keystore then `the –new_label <target_label>` parameter may be added to the line.

Listing CAs

To display a list of trusted CAs in a key database (*command should be entered as one line*):

```
gsk7cmd -cert -list CA -db <dbname> -pw <password> -type  <cms | jks
| jceks | pkcs12>
```

Opening a key database

There is no explicit opening of a key database. For each command, database and password options are specified and these specifications provide the information needed to operate in a key database.

Receiving a CA-signed certificate

Use this procedure to receive an electronically mailed certificate from a certificate authority (CA), designated as a trusted CA on your server. By default, the following CA certificates are stored in the key database and marked as trusted CA certificates:

- Verisign Class 2 OnSite Individual CA
- Verisign International Server CA -- Class 3
- VeriSign Class 1 Public Primary CA -- G2
- VeriSign Class 2 Public Primary CA -- G2
- VeriSign Class 3 Public Primary CA -- G2
- VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
- VeriSign Class 2 CA Individual Subscriber-Persona Not Validated
- VeriSign Class 3 CA Individual Subscriber-Persona Not Validated
- RSA Secure Server CA (from RSA)
- Thawte Personal Basic CA
- Thawte Personal Freemail CA
- Thawte Personal Premium CA
- Thawte Premium Server CA
- Thawte Server CA

The Certificate Authority may send more than one certificate. In addition to the certificate for your server, the CA may also send additional Signing certificates or Intermediate CA Certificates. For example, Verisign includes an Intermediate CA Certificate when sending a Global Server ID certificate. Before receiving the server certificate, receive any additional Intermediate CA certificates. Follow the instructions in Storing a CA certificate to receive Intermediate CA Certificates.

Note:

If the CA who issues your CA-signed certificate is not a trusted CA in the key database, you must first store the CA certificate and designate the CA as a trusted CA. Then you can receive your CA-signed certificate into the database. You cannot receive a CA-signed certificate from a CA who is not a trusted CA. For instructions, see Storing a CA certificate .

To receive the CA-signed certificate into a key database, enter the following command (as one line):

```
gsk7cmd -cert -receive -file <filename> -db <filename> -pw
<password>
      -format <ascii | binary> -label <label> -default_cert <yes |
no>
```

where:

-cert

Specifies a self-signed certificate.

-receive
: Specifies a receive action.

-file <*filename*>
: File containing the CA certificate

-db <*filename*>
: Name of the database.

-pw <*password*>
: The password to access the key database.

-format <ascii | binary>
: Certificate Authority might provide CA Certificate in either ASCII or binary format.

-default_cert <yes | no>
: Whether this is the default certificate in the key database.

-label <*label*>
: Label attached to a CA certificate.

-trust
: Indicates whether this CA can be trusted. Use enable options when receiving a CA certificate.

Showing the default key in a key database

To display the default key entry, enter the following command:

```
gsk7cmd -cert -getdefault -db <dbname> -pw <password>
```

Listing Expired Certificates

To display a list of certificates in a key database and their expiry dates:

```
gsk7cmd -cert -list -expiry <days> -db <filename> -pw <paswsword>  -
type <type>
```

where:

-cert
: Indicates the operation applies to a certificate.

-list <all | personal | CA | site>
: Specifies a list action, default is to list all certificates

-expiry <*days*>
: Indicates that validity dates should be displayed. Specifying the number of days is optional, though when used will result in displaying all certificates that expire within that amount of days. To list certificates that have already expired, enter the value 0.

-db <*filename*>
: Name of the key database. Used when you want to list certificate for a specific key database.

-pw <*password*>
: The password to access the key database.

-type <cms | JKS | JCEKS | pkcs12>
: The type of database.

Showing the entire certificate

To display the entire key entry, type the following on one line:

```
gsk7cmd -cert -details -showOID -db <filename> -pw <password> -label
<label>
```

where:

-cert
Indicates the operation applies to a certificate.
-details
Displays the entire key entry.
-showOID
Displays the entire certificate or certificate request.
-db <*filename*>
Name of the database.
-pw <*password*>
The password to access the key database.
-label <*label*>
Label attached to the certificate or certificate request.

Storing a CA certificate

To store a certificate from a CA who is not a trusted CA:

```
gsk7cmd -cert -add -db <filename>.kdb -pw <password>
      -label <label> -format <ascii | binary> -trust <enable
|disable>
      -file <filename>
```

where:

-add
Specifies an add action.
-cert
Indicates the operation applies to a certificate.
-db <*filename*>
Name of the database.
-file <*filename*>
File containing the CA certificate.
-format <ascii | binary>
Certificate Authorities might supply a binary or an ASCII file.
-label <*label*>
Label attached to certificate or certificate request.
-pw <password>
The password to access the key database.
-trust <enable | disable>
Indicate whether this CA can be trusted. Should be *Yes*.

Storing the encrypted database password in a stash file

For a secure network connection, store the CMS encrypted database password in a stash file. To store the password when creating a CMS database:

```
gsk7cmd -keydb -create -db <path_to_db>/<db_name> -pw <password> -
type cms
             -expire <days> -stash
```

To store the password after a CMS database has been created:

```
gsk7cmd -keydb -stashpw -db <db_name> -pw <password>
```

Managing a digital certificate on a smart card

The iKeyman CLI allows you to manage digital certificates on a smart card (or more generically, on a PKCS11 cryptographic device). To do so from either CMS, IBM JSSE, or Microsoft CryptoAPI, you must first perform the following steps to inform the iKeyman CLI of the name of the module for managing your smart card:

1. Edit the `java.security` file in your JDK environment with one of the following settings stored in the `$JAVA_HOME/jre/lib/security/` directory:
    - security.provider.1=sun.security.provider.Sun -- *versions greater than JDK 1.3.1*
    - security.provider.2=com.ibm.spi.IBMCMSProvider
    - security.provider.3=com.ibm.crypto.provider.IBMJCE
2. Edit the `ikeycmd.properties` file to set the **DEFAULT_CRYPTOGRAPHIC_MODULE** property to the name of the module for managing your smart card. For example:

    ```
    DEFAULT_CRYPTOGRAPHIC_MODULE=C:\\Winnt\\System32\\W32pk2ig.dll
    ```

    The module is normally installed on your system when you install the software for your smart card. The `.dll` in the above example is used here for illustration only and may differ from that for your system.

3. Save the `ikeycmd.properties` file
4. Use the following for PKCS11 related operations:

    ```
    -crypto <module_name> -tokenlabel <token_label> -pw <password>
    ```

For example, to display the certificate contained on your PKCS11 cryptographic device, type:

```
gsk7cmd -Dikeycmd.properties=<propertiesfile>  -cert -list all -
crypto <module_name>
             -tokenlabel <token_label> -pw <password>
```

where:

-cert
Indicates the operation applies to a certificate.
-crypto <*module_name*>

Specifies PKCS11 cryptographic device usage, where <*module_name*> is the name of the module to manage your smart card. Optional if specified in the `ikmcmd.properties` file.

-list all
: Displays the certificate.

-pw <*password*>
: The password for this PKCS11 cryptographic device.

-tokenlabel <*token_label*>
: The PKCS11 cryptographic device token label.

If you want to open a cryptographic hardware device using IBM JSSE implementation, do the following:

1. Edit the `java.security` file in your JDK environment with one of the following settings stored in the *`$JAVA_HOME/`*`jre/lib/security/` directory:
    - security.provider.1=sun.security.provider.Sun -- *JDK 1.3.1 only*
    - security.provider.2=com.ibm.crypto.provider.IBMJCE
    - security.provider.3=com.ibm.spi.IBMCMSProvider

   That is, set the IBM JCE to be a higher priority than CMS in the `java.security` file.

2. Edit the `ikeycmd.properties` file to set the **DEFAULT_CRYPTOGRAPHIC_MODULE** property to the name of the module for managing your smart card. For example:

   ```
   DEFAULT_CRYPTOGRAPHIC_MODULE=C:\\Winnt\\System32\\W32pk2ig.dll
   ```

   The module is normally installed on your system when you install the software for your smart card. The `.dll` in the above example is used here for illustration only and may differ from that for your system.

3. Save the `ikeycmd.properties` file.
4. Use the following for PKCS11 related operations:

   ```
   -crypto <module_name> -relativeSlotNumber <slot_number> -pw
   <password>
   ```

For example, to display the certificate contained on your PKCS11 cryptographic device, type:

```
gsk7cmd -Dikeycmd.properties=<propertiesfile> -cert -list all -
crypto <module_name>
       -relativeSlotNumber <slot_number> -pw <password>
```

where:

-cert
: Indicates the operation applies to a certificate

-crypto <*module_name*>

Specifies PKCS11 cryptographic device usage, where <*module_name*> is the name of the module to manage your smart card. Optional if specified in the `ikmcmd.properties` file.

-list all
: Displays the certificate

-pw <*password*>
: The password for this PKCS11 cryptographic device

-relativeSlotNumber <*slot_number*>
: The PKCS11 cryptographic device relative slot number

If you want to open a cryptographic hardware device using MS CryptoAPI, do the following:

1. Edit the `ikeycmd.properties` file. Set the cryptographic service provider (CSP) name to the **DEFAULT_CRYPTOGRAPHIC_MODULE** property. For example:

   ```
   DEFAULT_CRYPTOGRAPHIC_MODULE=Schlumberger Cryptographic
   Service Provider
   ```

   The CSP is normally installed on your system when you install the software for your smart card.

2. Save the `ikeycmd.properties` file.
3. Use the following to display the certificate contained on your cryptographic device:

4.
   ```
   gsk7cmd -Dikeycmd.properties=<propertiesfile> -cert -list all
             -crypto <CSP_module_name
   ```

   where:

   -cert
   Indicates the operation applies to a certificate.

   -crypto <*CSP_module_name*>
   Specifies the cryptographic service provider (CSP) name.

   -list all
   Displays the certificate.

Managing the Microsoft Certificate Store

The iKeyman CLI allows you to manage the Microsoft Certificate Store, which is supported only on Windows 2000 with Service Pack 2 or later. To work on Microsoft Certificate Store, use **-db MSCertificateStore**. A password (**-pw**) is NOT needed in **-db MSCertificateStore** related operations.

For example, to view the details of a certificate stored in Microsoft Certificate Store, type the following on one line:

```
gsk7cmd -cert -details -db MSCertificateStore -label <label>
```

where:

-cert
Specifies certificate information.

-details
Specifies verbose mode.

-db MSCertificateStore
Specifies Microsoft Certificate Store.

-label <*label*>
The label of the certificate used in the operation.

Managing the Java Key Store

The iKeyman CLI supports various key store providers that register in the `java.security` registry. For example, if you have the following settings in your `java.security` registry, you will be able to use JKS, JCEKS, PKCS12, and CMS

- security.provider.1=sun.security.provider.Sun -- *versions greater than JDK 1.3.1*
- security.provider.2=com.ibm.crypto.provider.IBMJCE
- security.provider.3=com.ibm.spi.IBMCMSProvider

Note:

The following functions are for a CMS key store only:

Table 1. CMS key store functions

Functions	**Key words**
Stash the password of a key database into a file	**-keydb -stashpw** **-stash**
Setting/getting default personal certificates	**-cert -setdefault** **-cert -getdefault** **default_cert <no | yes>**

Table 1. CMS key store functions

Functions	**Key words**
Setting trusted certificate	**-cert -modify** **-trust <enable | disable>**
Setting expiration time for password	**-expire <days>**
No password support	**-pw <password>**

IKEYCMD command line parameter overview

The following table describes each *action* that can be performed on a specified *object*.

Object	**Action**	**Description**
-keydb	-changepw	Change the password for a key database
	-convert	Convert the key database from one format to another
	-create	Create a key database
	-delete	Delete the key database
	-stashpw	Stash the password of a key database into a file
	-expiry	Display the expiry date of the password associated with a key database

Object	**Action**	**Description**
-cert	-add	Add a CA certificate from a file into a key database
	-create	Create a self-signed certificate
	-delete	Delete a CA certificate
	-details	List the detailed information for a specific certificate
	-export	Export a personal certificate and its associated private key from a key database into a PKCS#12 file, or to another key database
	-extract	Extract a certificate from a key database
	-getdefault	Get the default personal certificate
	-import	Import a certificate from a key database or PKCS#12 file
	-list	List all certificates
	-modify	Modify a certificate (NOTE: Currently, the only field that can be modified is the Certificate Trust field)
	-receive	Receive a certificate from a file into a key database
	-setdefault	Set the default personal certificate

Object	Action	Description
	-sign	Sign a certificate stored in a file with a certificate stored in a key database and store the resulting signed certificate in a file
-certreq	-create	Create a certificate request
	-delete	Delete a certificate request from a certificate request database
	-details	List the detailed information of a specific certificate request
	-extract	Extract a certificate request from a certificate request database into a file
	-list	List all certificate requsts in the certificate request database
	-recreate	Recreate a certificate request
	-crypto	Indicate a PKCS11 cryptographic device operation. The value after **-crypto** can be optional if specified in the properties file.
-help		Display help information for the IKEYCMD command
-version		Display IKEYCMD version information

IKEYCMD command line options overview

The following table shows each option that can be present on the command line. The options are listed as a complete group. However, their use is dependent on the *object* and *action* specified on the command line.

Option	Description
-db	Fully qualified path name of a key database
-default_cert	Sets a certificate to be used as the default certificate for client authentication (yes or no). Default is no.
-dn	X.500 distinguished name. Input as a quoted string of the following format (only CN is required): *`"CN=Jane Doe,O=IBM,OU=Java Development,L=Endicott, ST=NY,ZIP=13760,C=country"`*
-encryption	Strength of encryption used in certificate export command (strong or weak). Default is strong.
-expire	Expiration time of either a certificate or a database password (in days). Defaults are: 365 days for a certificate and 60 days for a database password. Duration is 0 to 7300 (20 years).
-file	File name of a certificate or certificate request (depending on specified *object*)
-format	Format of a certificate (either ascii for Base64_encoded ASCII or *binary* for Binary DER data). Default is ascii.
-label	Label attached to a certificate or certificate request
-new_format	New format of key database
-new_label	A new certificate label or alias to replace an existing label
-new_pw	New database password
-old_format	Old format of key database

Option	Description
-pfx	Interpret the PKCS12 file as a Microsoft pfx variant
-pw	Password for the key database or PKCS#12 file. See Creating a new key database.
-size	Key size (512, 1024, or 2048). Default is 1024.
-stash	Indicator to stash the key database password to a file. If specified, the password will be stashed in a file.
-san_dnsname	Add one or more DNS names to the Subject Alternate Name attribute. Must be in “preferred name syntax” according to RFC 1034.
-san_emailaddr	Add one or more email addresses to the Subject Alternate Name attribute. Must be an “addr-spec” as defined in RFC 822
-san_ipaddr	.Add one or more IP addresses to the Subject Alternate Name attribute. Must be a string according to RFC 1338 and RFC 1519.
-secondaryDB	Secondary key database support for PKCS11 device operations
-secondaryDBpw	Password for the secondary key database support for PKCS11 device operations
-showOID	Display the entire certificate or certificate request
-target	Destination file or database
-target_pw	Password for the key database if *-target* specifies a key database. See Creating a new key database.

Option	Description
-target_type	Type of database specified by -*target* operand (see -*type*).
-tokenlabel	Label of a PKCS11 cryptographic device
-trust	Trust status of a CA certificate (enable or disable). Default is enable.
-type	Type of database. Allowable values are: • cms (indicates a CMS key database), • jce (indicates Sun's proprietary Java Cryptography Extension), • jceks (indicates Sun's proprietary Java Cryptography Extension Key Store), • pkcs12 (indicates a PKCS#12 file).
-usereasoncode	Return a multi-valued error code if the command fails or 0 if it is successful.
-x509version	Version of X.509 certificate to create (1, 2 or 3). Default is 3.

Command line invocation for CMS key database only

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-keydb -changepw -db <filename> -pw <password> -new_pw
<new_password>
-stash -expire <days>
-keydb -create -db <filename> -pw <password> -type <cms>
-expire <days> -stash
-keydb -stashpw -db <filename> -pw <password>
-cert -getdefault -db <filename> -pw <password>
-cert -modify -db <filename> -pw <password> -label <label>
-trust <enable | disable>
-cert -setdefault -db <filename> -pw <password> -label <label>
```

Cryptographic command line invocation (CMS implementation)

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

> For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-keydb -changepw -crypto <module_name>
-tokenlabel <token_label> -pw <password> -new_pw <new_password>
-cert -add -crypto <module_name> -tokenlabel <token_label> -pw
<password>
-label <label> -file <filename> -format <ascii | binary>
-cert -create -crypto <module_name> -tokenlabel <token_label> -pw
<password>
-label <label> -dn <distinguished_name> -size <1024 | 512 | 2048>
-x509version <3  | 1 | 2> -default_cert <no | yes> -expire <days>
-cert -delete -crypto <module_name> -tokenlabel <token_label> -pw
<password>
-label <label>
-cert -details -crypto <module_name> -tokenlabel <token_label>
-pw <password> -label <label>
-cert -details -showOID -crypto <module_name> -tokenlabel
<token_label>
-pw <password> -label <label>
-cert -extract -crypto <module_name> -tokenlabel <token_label> -pw
<password>
-label <label> -target <filename> -format <ascii | binary>
```

The next command imports a certificate to a cryptographic device with secondary key database support.

```
-cert -import -db <filename> -pw <password> -label <label>
-type <cms> -crypto <module_name> -tokenlabel <token_label>
-pw <password> -secondaryDB <filename> -secondaryDBpw <password>
```

The next command imports a PKCS12 certificate to a cryptographic device with secondary key database support.

```
-cert -import -file <filename> -pw <password>
-type <pkcs12> -crypto <module_name> -tokenlabel <token_label>
-pw <password> -secondaryDB <filename> -secondaryDBpw <password>
-cert -list <all | personal | CA> -crypto <module_name>
-tokenlabel <token_label> -pw <password>
-cert -receive -file <filename> -crypto <module_name> -tokenlabel
<token_label>
-pw <password> -secondaryDB <filename> -secondaryDBpw <password>
-format <ascii | binary>
-certreq -create -crypto <module_name> -tokenlabel <token_label>
-pw <password> -label <label> -dn <distinguished_name>
-size <1024 | 512 | 2048> -file <filename>
-certreq -delete -crypto <module_name> -tokenlabel <token_label>
-pw <password> -label <label>
-certreq -details -crypto <module_name> -tokenlabel <token_label>
-pw <password> -label <label>
-certreq -details -showOID -crypto <module_name> -tokenlabel
<token_label>
-pw <password> -label <label>
```

```
-certreq -extract -crypto <module_name> -tokenlabel <token_label>
-pw <password> -label <label> -target <filename>
-certreq -list -crypto <module_name> -tokenlabel <token_label>
-pw <password>
```

Cryptographic command line invocation (IBM JSSE implementation)

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

> For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-keydb -list
-cert -add -crypto <module_name> -relativeSlotNumber <slot_number>
-pw <password> -label <label> -file <filename> -format <ascii |
binary>
-cert -create -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label> -dn <distinguished_name>
-size <1024 | 512 | 2048> -x509version <3  | 1 | 2>
-cert -delete -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
-cert -details -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
-cert -details -showOID -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
-cert -extract -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label> -target <filename>-format <ascii |
binary>
-cert -import -db <filename> -pw <password> -label <label>
-type <pkcs12 | cms | JKS | JCEKS> -crypto <module_name>
-relativeSlotNumber <slot_number> -pw <password>
-cert -list <all | personal | CA> -crypto <module_name>
-relativeSlotNumber <slot_number> -pw <password>
-cert -receive -file <filename> -crypto <module_name>
-relativeSlotNumber <slot_number> -pw <password>
-format <ascii | binary>
-certreq -create -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label> -dn <distinguished_name>
-size <1024 | 512 | 2048> -file <filename>
-certreq -delete -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
-certreq -details -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
-certreq -details -showOID -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password> -label <label>
```

```
-certreq -list -crypto <module_name> -relativeSlotNumber
<slot_number>
-pw <password>
```

PKCS11 cryptographic command line invocation (Microsoft CryptoAPI implementation)

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-cert -add -crypto <CSP_module_name> -label <label>
-file <filename> -format <ascii | binary>
-cert -create -crypto <CSP_module_name>
-label <label> -dn <distinguished_name> -size <1024 | 512 | 2048>
-x509version <3  | 1 | 2> -expire <days>
-cert -delete -crypto <CSP_module_name> -label <label>
-cert -details -crypto <CSP_module_name> -label <label>
-cert -details -showOID -crypto <CSP_module_name> -label <label>
-cert -extract -crypto <CSP_module_name> -label <label>
-target <filename> -format <ascii | binary>
```

Note:

In the next command, **-label** is not needed for a PKCS12 key database.

```
-cert -import -db <filename> -pw <password> -label <label>
-type <pkcs12 | cms | JKS | JCEKS> -crypto <CSP_module_name>
-cert -list <all | personal | CA> -crypto <CSP_module_name>
```

Microsoft Certificate Store command line invocation

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-cert -add -db MSCertificateStore -label <label> -file <filename>
-format <ascii | binary>
-cert -create -db MSCertificateStore -label <label> -dn
<distinguished_name>
-size <1024 | 512 | 2048> -x509version <3  | 1 | 2> -expire <days>
-cert -delete -db MSCertificateStore -label <label>
-cert -details -db MSCertificateStore -label <label>
-cert -details -showOID -db MSCertificateStore -label <label>
-cert -export -db MSCertificateStore -label <label>
```

```
-target <filename> -target_pw <password> -target_type <cms | jks |
jceks | pkcs12>
-cert -extract -db MSCertificateStore -label <label>
-target <filename> -format <ascii | binary>
-cert -import -db <filename> -pw <password> -label <label>
-type <cms | JKS | JCEKS> -target MSCertificateStore
-cert -import -file <filename> -pw <password>
-type <pkcs12> -target MSCertificateStore
-cert -list <all | personal | CA> -db MSCertificateStore
```

Command line invocation for all types of key store (CMS, JKS, JCEKS, PKCS12)

The following is a list of each of the command line invocations, with the optional parameters specified in *italics*. All commands are entered as one line.

Note:

For simplicity, the actual Java invocation, *gsk7cmd*, is omitted from each of the command invocations.

```
-keydb -changepw -db <filename> -pw <password> -newpw <new_password>
```

Note:

The following is only valid for CMS key databases:

```
    -keydb -changepw -db <filename> -pw <password> -newpw
      <new_password>  -expire <days>
-keydb -convert -db <filename> -pw <password>
-old_format <cms | JKS | JCEKS | pkcs12> -new_format <cms | JKS |
JCEKS | pkcs12>
-keydb -create -db <filename> -pw <password> -type <cms | JKS |
JCEKS | pkcs12>
-keydb -delete -db <filename> -pw <password>
-cert -add -db <filename> -pw <password> -label <label>
-file <filename> -format <ascii | binary>
-cert -create -db <filename> -pw <password> -label <label>
-dn <distinguished_name> -size <1024 | 512 | 2048> -x509version <3
| 1 | 2>
-expire <days> -san_dnsname <DNS name value>[,<DNS name value>] –
san_emailaddr <email address value>[,<email address value>] –
san_ipaddr <IP address value>[,<IP address value>]
-cert -delete -db <filename> -pw <password> -label <label>
-cert -details -db <filename> -pw <password> -label <label>
-cert -export -db <filename> -pw <password> -label <label>
-type <cms | JKS | JCEKS | pkcs12> -target <filename>
-target_pw <password> -target_type <cms | JKS | JCEKS | pkcs12>
-cert -extract -db <filename> -pw <password> -label <label>
-target <filename> -format <ascii | binary>
-cert -import -db <filename> -pw <password> -label <label>
-type <cms | JKS | JCEKS> -target <filename> -target_pw <password>
-target_type <cms | JKS | JCEKS | pkcs12>
-cert -import -file <filename> -pw <password>
-type <pkcs12> -target <filename> -target_pw <password>
```

```
-cert -list <all | personal | CA> -db <filename>
-pw <password> -type <cms | JKS | JCEKS | pkcs12>
-cert -receive -file <filename> -db <filename>
-pw <password> -format <ascii | binary>
-certreq -create -db <filename> -pw <password>
-label <label> -dn <distinguished_name> -size <1024 | 512 | 2048> -
file <filename> -san_dnsname <DNS name value>[,<DNS name value>] –
san_emailaddr <email address value>[,<email address value>] –
san_ipaddr <IP address value>[,<IP address value>]
-certreq -delete -db <filename> -pw <password> -label <label>
-certreq -details -db <filename> -pw <password> -label <label>
-certreq -extract -db <filename>
-pw <password> -label <label> -target <filename>
-certreq -list -db <filename> -pw <password>
-certreq -recreate -db <filename>
-pw <password> -label <label> -target <filename>
```

User properties file

In order to eliminate some of the typing on the Java CLI invocations, user properties can be specified in a properties file. The properties file can be specified on the Java command line invocation via the *-Dikeycmd.properties=<filename>* Java option. A sample properties file, `ikeycmd.properties`, is supplied as a sample to enable Java applications to modify default settings for their application.

GSKit iKeyman support for accessibility

To accommodate individuals who have disabilities so they can use IBM products, important accessibility features and functions must be included in our products so they are either directly accessible or compatible with assistive technology. The IBM Accessibility Center and Sun Microsystems' Accessibility Group have combined efforts to design and build next-generation accessibility into the Java application. Follow the checklist at http://www-3.ibm.com/able/accessjava.html to make a Java2 application accessible. iKeyman is one of the products that currently comply with the accessibility support initiative.

Note:

> Current Java accessibility support is completely featured on the Windows platform only.

Features include the following:

- For the usability accessibility support:

  Users are able to operate iKeyman with the keyboard only, using the following:

  [Tab]
  Key to move focus forward
  [Shift]+[Tab]
  Keys to move focus backward
  [Space]
  Key to trigger action
  [up arrow]
  Key to move selectable item(s) up
  [down arrow]
  Key to move selectable item(s) down

- For the visual effect accessibility support:
  1. Modify the platform setting for colors and fonts. For example, on the Windows platform, modify the system color and font settings in Display properties on the Control Panel.
  2. Enable accessibility support. Select **View** -> **Windows Look and Feel**. The appearance of an iKeyman display will adopt the current system color and fonts.

GSKit iKeyman properties

To allow customization of some operational settings and to enable/disable some functionality a system of properties files has been provided. The default set of properties files installed with GSKit is usually sufficient for most users of Ikeyman.

There are 3 types of property file identified by their names.

1. `ikminit.properties` – there can be only one instance of this file and it must be in the directory <GSKit install path>/gsk7/classes . This file is provided as part of the initial installation of GSKit.
2. `ikmuser.properties` – this file contains properties which are local to a particular use of iKeyman. There can be many versions of this file. To use a version it must be present in the working directory of iKeyman. The working directory is the directory in which `gsk7cmd` or `gsk7ikm` are started. For example this might be a user's home directory. A sample file renamed as `ikmuser.sample` is provided as part of the initial installation of GSKit.
3. *-Dikeycmd.properties* option – the value of this option is a filename-path which is the properties file. This file is discussed further in the paragraph *User properties file* in this document. Note that this option is only applicable to the command line version of iKeyman.

The order of searching files for a property is (3), (2), (1).

Appendices.

Error Codes

The following codes are returned by the gsk7cmd command when used with the “-usereasoncode” option. The following terms are used in the description –

a. Internal – would only be caused by a software error. Contact your normal support channel.

b. Not used – Obsolete, should never appear.

Name	return codes	Description
OK	0	No error
KEYDB	1	Internal - mode action invalid
CERTIFICATE	2	Internal - mode action invalid
CERTREQUEST	3	Internal - mode action invalid
CREATEDB	4	Invalid parameter in –keydb action.
CHANGEPW	5	Invalid parameter in –keydb action
STASHPW	6	Invalid parameter in –keydb action
CONVERT	7	Invalid parameter in –keydb action
DELETEDB	8	Invalid parameter in –keydb action
RECREATE	9	Invalid parameter in –certreq action
LISTCERT	10	Invalid parameter in -cert action
DETAILSCERT	11	Invalid parameter in -cert action
EXPORT	12	Invalid parameter in –cert action
IMPORT	13	Invalid parameter in –cert action
EXTRACTCERT	14	Invalid parameter in -cert action
RECEIVE	15	Invalid parameter in -cert action
SIGN	16	Invalid parameter in –cert action
ADD	17	Invalid parameter in –cert action
MODIFY	18	Invalid parameter in -cert action
SETDEFAULT	19	Invalid parameter in –cert action
GETDEFAULT	20	Invalid parameter in -cert

		action
HELP	21	Internal - mode action invalid
VERSION	22	Internal - mode action invalid
CREATECERT	23	Invalid parameter in –cert action
CREATEREQ	24	Invalid parameter in –certreq action
DELETECERT	25	Invalid parameter in –cert action
DELETEREQ	26	Invalid parameter in –certreq action
LISTREQ	27	Invalid parameter in –certreq action
DETAILSREQ	28	Invalid parameter in –certreq action
EXTRACTREQ	29	Invalid parameter in –certreq action
LISTKEYDB	30	Invalid parameter in –keydb action
GENERAL_IO_EXCEPTION_ERROR	185	Internal - non-specific io error
GSKKM_ERR_CMN_KEYDB_OPEN	186	Internal - should never occur. Attemp to create a KeyStoreManager object failed unexpectedly
GSKKM_ERR_CMN_PASSWORD_NULL	187	Internal - null or no password provided when opening a non PKCS11 keystore
GSKKM_ERR_FILENAME_NULL	188	Internal - attempt to create a new key database with a null filename
GSKKM_ERR_CMN_KEYDB_GET_KEY_BY_LABEL	189	Internal - general error in getting key by label.
GSKKM_ERR_CMN_KEYDB_NEW_SSCERT	190	Error creating self-signed certificate.
GSKKM_ERR_DATABASE_DUPLICATE_KEY_LABEL	191	Certificate label already exists in target keystore for a. creation of self-signed certificate. The label already exists so use another one b. importing certificates. Use the –relabel option to change the label in the target keystore. c. exporting certificates Change the conflicting label in the target keystore.
GSKKM_ERR_CRYPTO	192	Error initialising or creating a cryptographic token keystore.
GENERAL_EXCEPTION_ERROR	193	Internal - non-specific error
GENERAL_KEYSTORE_MANAGER_ERROR	194	Internal - error using KeyStoreManager object

NO_KEYSTORE_OP_ERROR	195	Internal - operator for keystore was null
GENERAL_KEYSTORE_ERROR	196	Error using KeyDatabase object
MISSING_RESOURCE_FILE	197	Internal error – the language bundle containing the gui strings for Ikeyman could not be found nor could the default messages
FILE_IO_ERROR	198	Could not close the log. (by default ikmgdbg.log) file for some reason.
MISSING_FILE_ERROR	199	The log file(by default ikmgdbg.log) could not be accessed. It does not exist or does not have write permissions.
MISSING_RESOURCE_ERROR	200	Internal error - The language bundle(“ikmerr”) containing the error messages for Ikeyman and ikeycmd could not be found nor could the default messages.
NULL_PASSWORD_SPECIFIED	201	Not Used
INVALID_ALT_NAME_OPTION	202	An invalid Subject Alternative Name type was provided. Only email, ipAddr and dnsName are recognised.
SUBJECT_ALT_NAME_NOT_SUPPORTED	203	“DEFAULT_SUBJECT_ALTERNATE_NAME_SUPPORT" in properties file is false and -san_* parameter was provided. Uncomment the corresponding line in the properties file(by default ikminit.properties)
NO_ALT_NAME_SPECIFIED	204	No value provided for -san_* parameter
NOPWD_KDB	205	Password parameter not required for this file type so –stashpw invalid. Reenter the command without the –stashpw option.
GSKKM_ERR_CRYPTOGRAPHIC_TOKEN_NOT_SUPPORT	206	Internal - should never occur
GSKKM_CLI_WIN2K_SUPPORT	207	MSCertificate Store only supported on W2K SP2 and later. Upgrade your version of Windows and retry.
GSKKM_CLI_MICROSOFT_INVALID_OPERATION	208	These operations are not valid for MSCP 1. any -keydb actions. The MS Certificate Store is integral to the Windows OS. 2. any certreq actions. The Microsoft Certificate Store does not hold certificate

		requests.
NOT_SUPPORTED_PARAMETER	209	A command line parameter is not supported in the contect of the other parameters. 1. -secondaryDBFile when Java cryptotoken used in "cert import . The secondary keystore is not supported by the Java PKCS11 provider. 2. -keydb -changepw not allowed on Java Cryptographic token. The Java PKCS11 provider does not support changing the PIN on a token. 3. -certreq -extract not allowed for JAVA cryptographic token 5. -expire only allowed for CMS keystore type. Password expiry is only provided by the IBM CMS Keystore type. U must provide your own password aging capability for other types. 6. -stash only allowed for CMS keystore type. Stash files are not supported by Java keystores providers. You must provide your own password hiding capability. 7. -trust valid only for CMS. The –trust option is only supported by the CMS keystore. 8. -pfx and non-PKCS12 file as source keystore. Use a PKCS12 formatted file as the source for the import. 9. -pfx and -label value provided. The PFX option only applies when an entire keystore is being imported. Delete the –label option and retry.
CMN_INVALID_FILE_NAME	210	File name specified does not exist (keydb delete). Check the spelling of the file name or path and that the permissions along the path supply the correct access.
NO_CRYPTOKI_LABEL_SPECIFIED	211	1. No cryptographic token label provided for CMS crypto token. You must supply the token name with the –tokenlabel option. 2. No relative slot number was provided for Java crypto token. You must supply the hardware slot number where

		the token is inserted using the –relativeSlotNumber option.
NO_CRYPTOKI_MODULENAME_SPECIFIED	212	No Cryptographic module name either parameter or property was provided even though one was required – a token label was provided. Add the file path of the cryptographic device driver using the –crypto option.
NOPWD_PW_TARGETKDB_HASNO_PWD	213	Password is not required for this db type of target db. Remove the –pw option and retry.
GSKKM_CLI_PARAMETER_NOT_REQUIRED	214	Parameter not meaningful in the context other options
NOPWD_PW_NOT_REQUIRED_NEW	215	Password provided for MSCertificateStore keydb. Remove the –pw option and reenter the command.
NATIVE_NOT_ENABLED	216	Not Used
LABEL_NULL	217	Internal - label value present but is null - has length 0.
NO_ENCRYP_SPECIFIED	218	No value provided for -encryption parameter
NO_TARGET_PASSWORD_SPECIFIED	219	No value provided for -target_pw parameter
NO_TRUST_STATUS_SPECIFIED	220	No value specified for –trust parameter
NO_CERT_FORMAT_SPECIFIED	221	No value provided for -format parameter.
NO_DEFAULT_CERT_VALUE_SPECIFIED	222	No value provided for -default_cert parameter
NO_X509_VER_SPECIFIED	223	No value supplied for –x509version parameter
NO_KEY_SIZE_SPECIFIED	224	No value for -size parameter
NO_NEW_DB_FORMAT_SPECIFIED	225	No keystore type value provided for -new_format parameter
NO_OLD_DB_FORMAT_SPECIFIED	226	No keystore type value provided for -old_format option in convert
NO_NEW_PASSWORD_SPECIFIED	227	No -new_pw value provided for -changepw action
NO_EXPIRE_DAYS_SPECIFIED	228	No time interval provided for –expire parameter
NO_DB_TYPE_SPECIFIED	229	No keystore type was provided and the type could not be deduced from the keystore name.
NO_CERTIFICATE_ACTION_SPECIFIED	230	Internal – no certificate action detected
NO_CERTREQUEST_ACTION_SPECIFIED	231	Internal – no certificate request action detected
NO_KEYDB_ACTION_SPECIFIED	232	Internal – no keydb action

		detected
WEBDB_NOT_SUPPORTED	233	Not used - WEBDB database supported only for "-keydb -convert"
NO_FILE_SPECIFIED	234	No file specified for 1."certreq create" –an external filepath is required to write the certificate request to. 2. "cert sign" or "cert add”. An external file path is required which contains the certificate to be added or signed.
NO_LABEL_SPECIFIED	235	Label parameter required for “cert” or “certreq” actions. Add the –label option to specify the certificate being acted on.
NO_TARGET_SPECIFIED	236	No value provided for -target_pw parameter
NO_PASSWORD_SPECIFIED	237	No password value specified for –pw option
NO_DB_NAME_SPECIFIED	238	No default keystore name was found and no name was provided as a parameter. Reenter the command using the –db option.
INVALID_VERSION	239	Version value must be one of 1, 2 or 3
NEED_REQUIRED_OPTIONS	240	The required common name field is empty.
INVALID_ENCRYPTION	241	Not used -encryption strength must strong or weak
NO_PARAMS	242	Internal - no arguments from command line
INVALID_TRUST	243	Invalid trust parameter value (enable or disable)
INVALID_LIST_OPTION	244	Not used - replaced by LISTCERT
INVALID_TYPE	245	A valid key store type is required for -type or -target_type parameters
BAD_KEY_SIZE	246	Key -size value could not be parsed into an integer
INVALID_EXPIRE	247	Certificate days to expire less than 1
INVALID_FORMAT	248	Format value not a valid file format (ascii or binary)
TOO_MANY_OPTIONS	249	Too many options on command line for this operation
INVALID_OPTION	250	Value for “default_cert” parameter must be yes or no
BAD_CERTIFICATE_ACTION	251	Invalid -certificate action
BAD_CERTREQUEST_ACTION	252	Invalid -certificate request action
BAD_KEYDB_ACTION	253	Invalid -keydb command action

BAD_MODE	254	Unidentified mode entered.
ERROR	255	Not Used

Ikeyman property keys

Table heading key

Key name	*The text value of the property key. Case is significant.* *A "*" indicates the key can be used in a IKEYCMD User properties file.* *A "+" indicates the key is only used by the IKEYCMD program.*
Default value	*The value which is assumed if this property is not explicitly stated.*
Distribution value	*The value of this property in the ikminit.properties file distributed with GSKit.*
Possible values	*The range of possible values. Case is not significant for values. For boolean properties usually "true" is the only specified value. Any other string is interpreted as "false".*
Use	*A description of the purpose of the property and the meaning of different values.*

Program properties

The following program properties are listed in ikminit.properties and are read by Ikeyman but are not used
DEFAULT_KEYDB_LOCATION_WEBDB
DEFAULT_KEYDB_LOCATION_SSLIGHT
DEFAULT_KEYDB_LOCATION_SSLINK
DEFAULT_KEYDB_LOCATION_CDSA
DEFAULT_KEYDB_LOCATION_BERKELEY
DEFAULT_PKCS12_GUI_POPUPS
DEFAULT_PKCS12_EXPORT_VERSION
DEFAULT_1_BROWSER_ITERATION_COUNT
DEFAULT_1_BROWSER_ENC_ALGORITHM
DEFAULT_2_IKEYMAN_ITERATION_COUNT
DEFAULT_2_IKEYMAN_ENC_ALGORITHM
DEFAULT_PKCS12_ENCODE_TYPE
DEFAULT_SSLIGHT_EXTRACT_FILE_LOCATION
DEFAULT_SSLIGHT_EXTRACT_FILE_NAME
DEFAULT_SSLIGHT_PACKAGE_NAME
DEFAULT_SSLIGHT_PASSWORD_REQUIRED
DEFAULT_WEBDB_PASSWORD_REQUIRED
DEFAULT_SSLIGHT_CREATE_EMPTY
DEFAULT_KEYDB_NAME_EXT_WEBDB
DEFAULT_KEYDB_NAME_EXT_SSLIGHT
DEFAULT_KEYDB_NAME_EXT_SSLINK
DEFAULT_KEYDB_NAME_EXT_CDSA
DEFAULT_KEYDB_NAME_EXT_BERKELEY
DEFAULT_KEYDB_NAME_WEBDB
DEFAULT_KEYDB_NAME_SSLIGHT
DEFAULT_KEYDB_NAME_SSLINK
DEFAULT_KEYDB_NAME_CDSA
DEFAULT_KEYDB_NAME_BERKELEY
DEFAULT_PEM_FILE_NAME_EXT

DEFAULT_CERTIFICATE_NAME_PEM
DEFAULT_CERTIFICATE_REQUEST_KEY_SIZE
DEFUALT_CERTIFICATE_ENCRYPTION

*Key name	DEFAULT_FILE_LOCATION
Default value	User’s current working directory
Distribution value	-
Possible values	Any accessible directory
Use	This is the default directory for any keystore or file which is not allocated a location through another property. `gsk7cmd` will use this value to store non-keystore files ie `gsk7cmd -cert -extract -db test.kdb -label test -target test -format ascii` but `gsk7ikm` will use the default keystore location

*Key name	DEFAULT_KEYDB_LOCATION_CMS
Default value	DEFAULT_FILE_LOCATION
Distribution value	“.”
Possible values	Any accessible directory
Use	This is the default directory for any keystore of type CMS. This does not work with `gsk7cmd` except with the `-keydb -create`. Similar for other keystore types. This does not work with `gsk?cmd` except with the `-keydb -create`. Similar for other keystore types.

*Key name	DEFAULT_KEYDB_NAME_CMS
Default value	“key”
Distribution value	“key”
Possible values	Any valid filename root for the platform. The filename extension should be left off.
Use	This is the default name for any keystore of type CMS. There is no way of requesting the default named keystore using `gsk?cmd`. ie `-db <filename>` is always required. This holds for all keystore types

*Key name	DEFAULT_KEYDB_NAME_EXT_CMS
Default value	“.kdb”
Distribution value	“.kdb”

Possible values	Any valid filename extension for the platform. The "." separator is mandatory.
Use	This is the default extension for any keystore of type CMS. This string will be appended to the filename. This default extension is not reliably used in `gsk7cmd` except with the `-keydb` modifier ie `-keydb -create`. This holds for all keystore types

Key name	USE_LAST_OPENED_LOCATION_CMS
Default value	"true"
Distribution value	"true"
Possible values	"true" or "false".
Use	When used with "gsk7ikm" it will cause any directory entered in response to a "New" or "Open" form to become the new cms default directory for the duration of execution otherwise the default value is reused.

*Key name	DEFAULT_CMS_PASSWORD_REQUIRED
Default value	"true"
Distribution value	-
Possible values	"true" or "false".
Use	When "false" it allows a CMS keystore to be created and subsequently accessed without a password. If the `-pw` option is included with the `-keydb -create` action then the keydb is created with that password

*Key name	DEFAULT_PASSWORD_STASHING_STATE
Default value	"false"
Distribution value	-
Possible values	"true" or "false"
Use	If true then the check box to stash the password in a password dialog will always be checked by default. Only valid for CMS.

*Key name	DEFAULT_JKS_FILE_LOCATION
Default value	DEFAULT_FILE_LOCATION
Distribution value	"."
Possible values	Any accessible directory
Use	This is the default directory for any keystore of type JKS

*Key name	DEFAULT_JKS_FILE_NAME
Default value	“key”
Distribution value	“key”
Possible values	Any valid root filename for the platform..
Use	This is the default name for any keystore of type JKS

*Key name	DEFAULT_JKS_FILE_NAME_EXT
Default value	“.jks”
Distribution value	“.jks”
Possible values	Any valid filename extension for the platform. The “.” separator is mandatory.
Use	This is the default extension for any keystore of type jks. This string will be appended to any filename entered without an extension.

Key name	USE_LAST_OPENED_LOCATION_JKS
Default value	“true”
Distribution value	“true”
Possible values	“true” or “false”.
Use	When used with “gsk7ikm” it will cause any directory entered in response to a “New” or “Open” form to become the new jks default directory for the duration of execution otherwise the default value is reused.

*Key name	DEFAULT_JCEKS_FILE_LOCATION
Default value	DEFAULT_FILE_LOCATION
Distribution value	“.”
Possible values	Any accessible directory
Use	This is the default directory for any keystore of type JCK

*Key name	DEFAULT_JCEKS_FILE_NAME
Default value	“key”
Distribution value	“key”
Possible values	Any valid root filename for the platform.
Use	This is the default name for any keystore of type JCEKS

*Key name	DEFAULT_JCEKS_FILE_NAME_EXT
Default value	“.jck”

Distribution value	“.jck”
Possible values	Any valid filename extension for the platform. The “.” separator is mandatory.
Use	This is the default extension for any keystore of type JCEKS. This string will be appended to any filename entered without an extension.

Key name	USE_LAST_OPENED_LOCATION_JCEKS
Default value	“true”
Distribution value	“true”
Possible values	“true” or “false”.
Use	When used with “gsk7ikm” it will cause any directory entered in response to a “New” or “Open” form to become the new default jck directory for the duration of execution otherwise the default value is reused.

*Key name	DEFAULT_PKCS12_FILE_LOCATION
Default value	DEFAULT_FILE_LOCATION
Distribution value	“.”
Possible values	Any accessible directory
Use	This is the default directory for any keystore of type pkcs12

*Key name	DEFAULT_PKCS12_FILE_NAME
Default value	“key”
Distribution value	“key”
Possible values	Any valid root filename for the platform.
Use	This is the default name for any keystore of type P12

*Key name	DEFAULT_PKCS12_FILE_NAME_EXT
Default value	“.p12”
Distribution value	“.p12”
Possible values	Any valid filename extension for the platform. The “.” separator is mandatory.
Use	This is the default extension for any keystore of type P12. This string will be appended to any filename entered without an extension when the determined type is PKCS12

Key name	USE_LAST_OPENED_LOCATION_PKCS12

Default value	“true”
Distribution value	“true”
Possible values	“true” or “false”.
Use	When used with “gsk7ikm” it will cause any directory entered in response to a “New” or “Open” form to become the new pkcs12 default directory for the duration of execution otherwise the default value is reused.

+*Key name	DEFAULT_KEYDB_TYPE
Default value	1 (CMS)
Distribution value	-
Possible values	1 or 8 (PKCS12)
Use	Sets up a default keystore type. Looks like this has not been updated to deal with the Java keystore types. Really only works for -keydb -create. Did not work for -cert -list for example.

+*Key name	DEFAULT_KEYDB_PASSWORD_EXPIRE
Default value	60
Distribution value	-
Possible values	Integer > 0
Use	The default keystore password lifetime in days. Only valid for CMS keystores

+*Key name	DEFAULT_CMS_PASSWORD
Default value	-
Distribution value	-
Possible values	Any valid string
Use	The default password for a CMS keystore. Only seems to work when creating a keystore.

+*Key name	DEFAULT_PKCS12_PASSWORD
Default value	-
Distribution value	-
Possible values	Any valid string
Use	The default password for a PKCS12 keystore

+*Key name	DEFAULT_CERTIFICATE_EXPIRE

Default value	365
Distribution value	-
Possible values	Any integer > 0
Use	The default certificate lifetime in days for self-signed certificates. CMS keystore only.

+*Key name	DEFAULT_CERTIFICATE_FORMAT
Default value	“true”
Distribution value	-
Possible values	“false” or any other valid string/format
Use	If set false then default is binary mode format otherwise is ascii (b64) mode for self-signed certificates which are being extracted – CMS keystore only.

+*Key name	DEFAULT_CERTIFICATE_TRUST
Default value	“true”
Distribution value	-
Possible values	“true” or “false”
Use	If set to enable then any added CA certificates are marked as trusted by default otherwise they are not trusted.

+*Key name	DEFAULT_CERTIFICATE_KEYSIZE
Default value	1024
Distribution value	-
Possible values	512, 1024, 2048
Use	The default keysize for self-signed certificates or certificate requests. CMS type keystore only.

+*Key name	DEFAULT_CERTIFICATE_DEFAULT
Default value	“false”
Distribution value	-
Possible values	“true” or any other valid string/format
Use	If set then the created selfsigned cert will be set as the default certificate. Only applicable to CMS types.

+*Key name	DEFAULT_CERTIFICATE_LIST_OPTION
Default value	“all”
Distribution value	-
Possible values	“all”, “CA”, “personal”
Use	The default option for the –cert –list action.

*Key name	DEFAULT_BASE64_FILE_NAME_EXT
Default value	“.arm”
Distribution value	-
Possible values	Any valid file extension for the platform
Use	Used as the file extension wherever a base64 formatted file is specified. Does not override an existing extension. Didn't work with `-cert -extract`. or `gsk?ikm`. Caused "Invalid file" error when file using defaultfile name in `gsk?ikm`

*Key name	DEFAULT_DER_FILE_NAME_EXT
Default value	“.der”
Distribution value	-
Possible values	Any valid file extension for the platform
Use	Used as the file extension wherever a binary formatted file is specified. Does not override an existing extension. Didn't work with `-cert -extract`. or `gsk?ikm`. Caused "Invalid file" error when file using defaultfile name in `gsk?ikm`

Key name	DEFAULT_CERTREQ_FILE_NAME
Default value	“certreq.arm”
Distribution value	-
Possible values	Any valid file name for the platform
Use	If no extension provided then the value of DEFAULT_BASE64_FILE_NAME_EXT will be appended. This is the default name for the certificate request file. Only relevant in `gsk?ikm`.

*Key name	DEFAULT_CERTIFICATE_NAME_ARM
Default value	“cert.arm”

Distribution value	-
Possible values	Any valid file name for the platform
Use	The default file name for a certificate being extracted in b64 mode. Only relevant in `gsk?ikm`.

*Key name	DEFAULT_CERTIFICATE_NAME_DER
Default value	“cert.der”
Distribution value	-
Possible values	Any valid file name for the platform
Use	The default file name for a certificate being extracted in binary mode. Only relevant in `gsk?ikm`.

Key name	DEFAULT_CRYPTOGRAPHIC_TOKEN_SECONDARY_KEYDB
Default value	-
Distribution value	-
Possible values	Any valid directory/file name for the platform
Use	The default directory path and filename of a CMS keystore to be used as a secondary keystore to a cryptotoken.

*Key name	DEFAULT_CRYPTOGRAPHIC_MODULE
Default value	-
Distribution value	-
Possible values	Any valid directory/file name for the platform
Use	The directory path and filename of a vendor specific PKCS11 dll which will interface to a cryptographic device.

Key name	DEFAULT_FIPS_MODE_PROCESSING
Default value	“off”
Distribution value	-
Possible values	“on” or “off”
Use	If set on then attempts to do FIPS compliant cryptography. Requires the properties DEFAULT_SIGNATURE_ALGORITHM to be set to “SHA1_WITH_RSA” or to be left to default and DEFAULT_CRYPTOGRAPHIC_BASE_LIBRARY set to “ICC”

Key name	DEFAULT_CRYPTOGRAPHIC_BASE_LIBRARY
Default value	SYSTEM_DEFAULT
Distribution value	-
Possible values	“RSA” or “ICC”
Use	Determines the binary library to be loaded which will provide the cryptographic routines for the CMS security provider. If this is allowed to default then the underlying JNI binary libraries will load their default library. Either way the “ICC” library is the only library which is FIPS compliant. The “RSA” library is the RSA BSAFE library.

Key name	DEFAULT_SIGNATURE_ALGORITHM
Default value	“SHA1_WITH_RSA” if the property DEFAULT_FIPS_MODE_PROCESSING set to “on”. otherwise “MD5_WITH_RSA”
Distribution value	-
Possible values	“MD5_WITH_RSA”, “SHA1_WITH_RSA”, “SHA224_WITH_RSA”, “SHA256_WITH_RSA”, “SHA384_WITH_RSA”, or “SHA512_WITH_RSA” Notes: • On JRE 1.4.2 the longer SHA algorithms only work for CMS certificates (i.e. not for CMS certificate requests or any other key store formats) • On JRE 5.0 all algorithms work for certificates and certificate requests for all key store formats, with the exception of SHA224, which is only available for CMS certificates. • Whenever an unavailable algorithm is set in this setting, the default value SHA1_WITH_RSA will be used instead.
Use	The algorithm to be used to sign a self-signed certificate or certificate request

Key name	VIEW_HIDDEN_FILES
Default value	“false”
Distribution value	-
Possible values	“true” or any other string is false
Use	If true then platform specific hidden files will be visible during browsing.

Key name	DEFAULT_SUBJECT_ALTERNATIVE_NAME_SUPPORT
Default value	“false”
Distribution value	-
Possible values	“true” or “false”
Use	Allows the entry of Subject Alternate Name attributes as extentions in self-signed certs and as extension requests in certificate requests

Key name	DEFAULT_ENGLISH_ERROR_MESSAGE
Default value	“false”
Distribution value	-
Possible values	“true” or “false”
Use	If true forces the locale for the display of error messages to “English” otherwise the default locale for the JVM is used.

Key name	DEFAULT_LOGGING
Default value	-
Distribution value	“false”
Possible values	“true” or any other string is false
Use	If true then logging is turned on..

Key name	DEFAULT_LOGGING_FILTER
Default value	-
Distribution value	“ALL”
Possible values	“ALL” – log everything “INFO” – log everything except tracing “WARN” – log warning and error messages “ERROR” – log only error messages
Use	Determines the level of logging message to be allowed into the log.

Key name	DEFAULT_LOGGING_FILE
Default value	-
Distribution value	-
Possible values	Any valid platform path/filename
Use	Logging messages are written to this file.

Key name	DEFAULT_SSCERT_BASIC_CONSTRAINTS[1]
Default value	-
Distribution value	-
Possible values	“true” or “false”
Use	Add the Basic Constraint extension to a self-signed certificate on creation. The extension will be added with a CA:true and PathLen:<max int> if set to “true” or not added at all if set to “false”.

System properties

The following system properties are accessed but never used.
keyman.docpath - supplied in ikmguiw.cpp, the Windows version of gsk?ikm
keyman.helpClassName

These properties may be entered on the command line using the *–D<key name>=<key value>* syntax. Note that not all properties are applicable to both the command line and GUI versions of iKeyman.

Key name	keyman.fix.jfc.mouse.retarget
Default value	“false”
Possible values	“true”, any other string
Use	If true then a work-around for a JDK bug that causes mouseEventTarget to be lost in certain circumstances. Used in internal Unix scripts `gsk?cmd` and `gsk?ikm.` A holdover from early JDKs (pre 1.3)

Key name	gskit.lib
Default value	-
Possible values	A library suffix to be appended to the jni library root name
Use	If present this suffix is used to allow specification of a special library extension if the inbuilt extension checking is not enough. Used internally in `gsk7cmd_64` and `gsk7ikm_64` for Linux on 64 bit Intel and AMD platforms.

Key name	IKEYMAN_JNI_GCC_VERSION
Default value	-
Possible values	A version suffix to be appended to the “_gcc” suffix to the jni library root name.

[1] Only available for version 7.0.3.20 and onwards

Use	If present this version is appended to the suffix which is appended to the jni library root. If not present then the "_gcc" suffix is not appended either. Used internally in Unix scripts `gsk7cmd_gcc295`, `gsk7ikm_gcc295.`

Key name	keyman.lang
Default value	Default locale for this JVM
Possible values	A locale in format <ISO Language Code>_<ISO Country Code>
Use	If present this locale overrides the default locale for this JVM

Key name	keyman.verbose
Default value	"false"
Possible values	"true", any other valid string
Use	If present this enables verbose output during startup – doesn't appear to do anything. Used internally in Unix scripts `gsk?cmd` and `gsk?ikm`

Key name	keyman.debug
Default value	"false"
Possible values	"true", any other string
Use	If set to "true" then debug tracing is enabled. Trace will be written to 3 files – `ikmcdbg.log, ikmjdbg.log` and `ikmgdbg.log`. The file name `ikmgdbg.log` may be replaced with the value of the **keyman.debugDumpFileName** property defined below.

Key name	keyman.debugDumpFileName
Default value	"ikmgdbg.log"
Possible values	Any valid platform path/filename
Use	Logging messages are written to this file.

Key name	keyman.jnitracing
Default value	"off"
Possible values	"on", any other string

Use	If set to “on” then jni tracing is enabled. Jni tracing is a trace generated by the non-Java libraries which implement the CMS, MS Certificate Store and CMS PKCS11 keystores.

Key name	ikeycmd.properties
Default value	-
Possible values	Any valid platform path/filename
Use	Loads the contents of the file as system properties

Key name	initPropertyFile
Default value	<GSK Install path>”/gsk7/classes/ikminit.properties”
Possible values	Any valid filename. “.properties” will be appended if not already present.
Use	Loads initial properties from this file

Key name	userPropertyFile
Default value	“ikmuser.properties”
Possible values	Any valid filename. “.properties” will be appended if not already present.
Use	Loads user properties from this file. This file is treated not quite the same as the "real" ikmuser.properties file. More like the ikeycmd.properties file.

Key name	guiPropertyFile
Default value	-
Possible values	Any valid platform path/filename
Use	Loads the contents of the file as a gui messages resource bundle. Overrides the locale specific resource bundle `ikmgui.` Only works with `gsk?ikm` not `gsk?cmd`.

Key name	errPropertyFile
Default value	-
Possible values	Any valid platform path/filename
Use	Loads the contents of the file as an error messages resource bundle. Overrides the locale specific resource bundle `ikmerr.` Only works with `gsk?ikm` not `gsk?cmd`.

Key name	keyman.useDosPrompt
Default value	“false”
Possible values	“true”, any other string is false
Use	When true the iKeyman startup window is not displayed. iKeyman starts up directly into the “IBM Key Management” window.

Key name	keyman.timer
Default value	“off”
Possible values	“on”, any other string is off
Use	If set on displays the startup elapsed time in the debug log (ikmgdbg.log)if this is enabled.

Key name	keyman.javaOnly
Default value	“false”
Possible values	“true”, any other string
Use	When true iKeyman will not support the CMS, MS Certificate Store or CMS PKCS11 keystore types

Key name	keyman.keydatabase
Default value	-
Possible values	Any valid platform path/filename. Must be the name of a keystore with a valid type extension.
Use	When present iKeyman will start with the provided keystore ready to open. A password will still be required.

Notices

Trademarks