IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6 and higher

iKeyman User's Guide

for version 8.0
SC32-1700-03
Note

Before using this information and the product it supports, read the information in Appendix C. A PKCS#11 Example.

This edition applies to iKeyman version 8.0 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

This manual is intended for network or system security administrators who want to use the iKeyman program to modify key databases.

This manual assumes the reader is familiar with general key database concepts.

Contacting software support

Before contacting IBM Software Support with a problem, refer to the IBM Software Support site by clicking the 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.

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™ 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`.

Operating system differences

This book uses the UNIX™ convention for specifying environment variables and for directory notation. When using the Windows™ 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.

Using the iKeyman Program

iKeyman is a certificate and key management application. It can be used to manage symmetric and asymmetric keys, digital certificates and certificate requests in various different types of key databases. It can also be used to manage the key databases themselves. It is advised that users make backups of keystores prior to modification as lost keys may need to be regenerated and certificates re-issued.

The application can be run in two modes, Graphic User Interface (GUI) and Command Line (CLI). Both modes support essentially the same command set, which is discussed in Commands.

iKeyman uses several encoding rules which are specified by aspects of certain RFCs and standards. While it is not strictly necessary for users to have a full understanding of these items in order to use this utility, those wishing to learn more should examine the following resources:

Basic Encoding Rules (BER)
: BER encoding is defined in the specification ITU-T Rec. X.690 (2002).

Distinguished Encoding Rules (DER)
: DER encoding is defined in the specification ITU-T Rec. X.690 (2002).

PKCS#10
: RFC 2986: PKCS #10: Certification Request Syntax Specification, Version 1.7, November 2000

X.509
: RFC 3280: Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL), obsoletes RFC 2459, April 2002.

PKCS#11
: Cryptographic Token Interface Standard

PKCS#12
: PKCS 12 v1.0: Personal Information Exchange Syntax, RSA Laboratories, June 24, 1999

PKCS#7

- Cryptographic Message Syntax Standard (RFC 2315),
- Used to sign and/or encrypt messages under a PKI,
- Used also for certificate dissemination (for instance as a response to a PKCS#10 message).

Supported Key database types

iKeyman currently supports the following key database types that are implemented as Java KeyStore Providers:

CMS

This is an IBM proprietary key database format used by IBM's GSKit library for SSL communications. It consists of the following two files:

Key Database File
: This file contains the certificates and private/public key pairs for personal certificates. It has a .kdb extension.

Certificate Request File

This file contains the certificate requests and associated public/private key pairs. It has the same name as the key database file with an extension of .rdb.

Other file based types

Each of the following types shares a common certificate request file format. The certificate request database consists of an index with a .qer file extension which lists the label of each certificate request appended by a random hex string.

For each certificate request there are also two files; one with a .bdr extension which contains the encrypted private key of the certificate request, and one with a .crq extension which contains the certificate request itself.

JKS
: Java Key Store format - The main file has a .jks extension.

JCEKS
: Java Cryptographic Extension Key Store format - A more secure extension of the JKS format. The main file has a .jck extension.

PKCS#12
: A file format used to transfer private keys and associated public key certificates. The main file has a .p12 extension.

PKCS#12S2
: This is a second version of PKCS12 type keystore. It can be read by the keytool program in a Oracle/Sun JVM. Note that PKCS12S2 uses stronger protection algorithms and should therefore be preferred over the PKCS12 type for key transport.
: PKCS12S2 is a cross platform keystore based on the RSA PKCS12 Personal Information Exchange Syntax Standard. Note that PKCS12S2 does not support the addition of trust anchors, but only private keys and their associated certificates.

Limitation: JKS and PKCS12 support multi-password keystores, i.e. the keystore password may be different from the private/secret key passwords. These are supported by Java's keytool and OpenSSL but IKeyMan will generate errors for keys where the password is different from the keystore password. To workaround you must use the native tool to make the passwords the same. Note that where key passwords must be separate you can use separate keystores.

PKCS#11

iKeyman is able to interface with key databases stored on smart cards via the PKCS#11 API. These key databases are basically the same as the file-based types above, however some functions are not supported, including exporting of private keys and changing the password of the key database.

Key Generation

All key generation occurs on the PKCS#11 device. Private keys are created as sensitive and not extractable, which means that once the key has been created on the device, it cannot be extracted and the key material cannot be accessed. Secret (symmetric) keys are created as sensitive but extractable, which means the key material itself cannot be accessed, however the keys can be exported by wrapping them with an asymmetric key.

Modes of Operation

iKeyman uses the IBM **PKCS11Impl** provider to access PKCS#11 key databases. This provider can be used in two modes:

config
: For this mode a configuration file needs to be created, which specifies the information for the PKCS#11 device being accessed. Each configuration file also contains a token label which is used by iKeyman to reference the particular configuration. This is the preferred mode of operation for PKCS#11 devices.

direct
: Key databases are accessed by specifying the module used to manage the PKCS#11 device (.dll or .so file) and a slot number. For details on enabling this mode, see Setup. This mode is included for backward compatibility with iKeyman version 7.

Secondary key databases

Some tokens have limited capacity, making them unable to hold signer certificates required to receive or import personal certificates. If a token has such a restriction, the user may choose to open a secondary key database to hold the signer certificates. The secondary database can be any file-based type.

GUI Display

When working with a PKCS#11 key database using the GUI, all entries on the token are preceded by the slot-number/token label used to open the database. This makes it easy to distinguish between entries stored on the key database, and those of the secondary key database, if one is used.

Setup

In order to run the iKeyman application no specialized setup is required. By default some features and functionality may however not be enabled. These are discussed below.

java.security file

The **java.security** file is located in the `$JAVA_HOME\jre\lib\security` folder of the Java installation. This file contains (amongst other information) the installed Java security providers; each provider is listed on a line starting with **security.provider** followed by an index.

The IBMJCE provider MUST be included in the **java.security** provider list even if iKeyman shall only be used to process CMS keystores. Because it uses crypto algorithms from the IBMJCE. IBMJCEFIPS might also be present, but it is not sufficient on its own. If the intention is for iKeyman to use FIPS algorithms in preference where available, then IBMJCEFIPS should be listed before IBMJCE.

The following table details the entry required for each supported key database type. To add a Provider, create a new line at the end of the list using the next index followed by the provider class. For example, to add the CMS provider you may need to add a line like the following:

```
security.provider.10=com.ibm.security.cmskeystore.CMSProvider

It takes an argument to specify whether it creates a V3 CMS keystore or
a V4 CMS keystore by default. The default is V4. V3 format may be
required in some cases for backwards compatibility. V4 format is
preferred for security reasons.

For example, this sets the keystore creation default to V3.
security.provider.10=com.ibm.security.cmskeystore.CMSProvider V3
```

Table 1. java.security file parameters

Key database type	**Security Provider**
JKS	com.ibm.crypto.provider.IBMJCE
JCEKS	com.ibm.crypto.provider.IBMJCE
PKCS#12 PKCS#12S2	com.ibm.crypto.provider.IBMJCE
CMS	com.ibm.security.cmskeystore.CMSProvider
MS-CAPI	sun.security.mscapi.SunMSCAPI
PKCS#11 - config mode	com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl followed by the location of a configuration file (see note below)
PKCS#11 - direct more	com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl

Note:
The **PKCS#11 configuration file** contains details used to interface with a PKCS#11 key database. The details consist of key-value pairs listed on separate rows of the file. Consult the PKCS11Impl Provider documentation (http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.security.component.60.doc/security-component/pkcs11implDocs/hardwareconfig.html) for more information. At a minimum, name, library and slot should be specified.

Supported Distinguished Name (DN) attributes

Certificates include a subject name and issuer name, as defined by X.509, to contain various name components in a structured and formatted way. The following table lists all GSKit, and certificate management tools, supported DN attributes that may be included when specifying a DN for a new certificate or request. Note, that although supported, MAIL and EMAIL should be avoided and subject alternative name email address (SAN) should be used instead.

Attribute Name	**Description**	**OID**
EMAIL	EmailAddress	1.2.840.113549.1.9.1
UID	Userid	0.9.2342.19200300.100.1.1
CN	CommonName	2.5.4.3
T	Title	2.5.4.12
OU *	OrganizationUnitName	2.5.4.11
DC *	DomainComponent	0.9.2342.19200300.100.1.25
O	OrganizationName	2.5.4.10
STREET	Street	2.5.4.9
L	LocalityName	2.5.4.7
ST	StateOrProvinceName	2.5.4.8
POSTALCODE (PC)	PostalCode	2.5.4.17
C	CountryName	2.5.4.6
DNQ	DNQualifier	2.5.4.46
GIVENNAME	GivenName	2.5.4.42
INITIALS	Initials	2.5.4.43
GENERATION	Generation	2.5.4.44

* attribute allows multiple values to be specified

Policy Files

By default the JRE may be using restricted policy files. These files are located in the directory `$JAVA_HOME\jre\lib\security` (**local_policy.jar** and **US_export_policy.jar**).

Some cryptographic algorithms and key sizes may not be available if the restricted policy files are being used. If an operation requires the unrestricted files, the error message will indicate this.

To update the policy files:

1. Download the Unrestricted JCE Policy files for 1.4.1 (1.3.1, 1.5 and later use the same Policy files) from:

   https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk

   The download includes the files, `local_policy.jar` and `US_export_policy.jar`

2. Copy `local_policy.jar` to `$JAVA_HOME/jre/lib/ext`
3. Copy `US_export_policy.jar` to `$JAVA_HOME/jre/lib/ext`

Settings file

iKeyman supports the use of a settings file to customize some functions of the program. The settings file can be specified on the command line invoking the application using the **-Dkeyman.settings** system property. The file name must end in ".properties" to be used by ikeyman. The supported settings are shown in the following table.

Table 2. iKeyman settings

Setting	Description	Default Value	Accepted Values
DEFAULT_CMS_PASSWORD _REQUIRED	If set to false, new CMS key databases are created with a default password.	true	true or false
DEFAULT_SUBJECT _ALTERNATIVE_NAME _SUPPORT	Whether SAN support is enabled. This affects the certificate creation and display dialogs.	false	true or false
RESOURCE_WARNING _ENABLED	If set to true, displays a message when the resource bundles for the default locale can't be found.	false	true or false
DEFAULT_FILE_LOCATION	The default file location used by the file chooser dialog.	.	any directory
DEFAULT_CMS_FILE _LOCATION	The default file location used for CMS key databases.	.	any directory
DEFAULT_KEYDB_LOCATION _CMS	Same as DEFAULT_CMS_ FILE_LOCATION	.	any directory
DEFAULT_JKS_FILE_LOCATION	The default file location used for JKS key databases.	.	any directory
DEFAULT_PKCS12_FILE_ LOCATION	The default file location used for PKCS12 key databases.	.	any directory

Table 2. iKeyman settings

Setting	Description	Default Value	Accepted Values
DEFAULT_JCEKS_FILE_ LOCATION	The default file location used for JCEKS key databases.	.	any directory
DEFAULT_CMS_FILE_NAME	The default file name used for CMS key databases.	key	any file name without extension
DEFAULT_KEYDB_NAME_CMS	Same as DEFAULT_CMS_ FILE_NAME	key	any file name without extension
DEFAULT_JKS_FILE_NAME	The default file name used for JKS key databases.	key	any file name without extension
DEFAULT_PKCS12_FILE_NAME	The default file name used for PKCS12 key databases.	key	any file name without extension
DEFAULT_JCEKS_FILE_NAME	The default file name used for JCEKS key databases.	key	any file name without extension
DEFAULT_CMS_FILE_NAME _EXT	The default file extension used for CMS key databases.	.kdb	any extension
DEFAULT_KEYDB_NAME_EXT _CMS	Same as DEFAULT_CMS_FILE _NAME_EXT	.kdb	any extension
DEFAULT_JKS_FILE_NAME_EXT	The default file extension used for JKS key databases.	.jks	any extension
DEFAULT_PKCS12_FILE_NAME _EXT	The default file extension used for PKCS12 key databases.	.p12	any extension
DEFAULT_JCEKS_FILE_NAME _EXT	The default file extension used for JCEKS key	.jck	any extension

Table 2. iKeyman settings

Setting	Description	Default Value	Accepted Values
	databases.		
USE_LAST_OPENED_LOCATION _CMS	Whether the last file location for CMS key databases will be remembered.	true	true or false
USE_LAST_OPENED_LOCATION _JKS	Whether the last file location for JKS key databases will be remembered.	true	true or false
USE_LAST_OPENED_LOCATION _PKCS12	Whether the last file location for PKCS12 key databases will be remembered.	true	true or false
USE_LAST_OPENED_LOCATION _JCEKS	Whether the last file location for JCEKS key databases will be remembered.	true	true or false
DEFAULT_CERTREQ_NAME	The default file name used for certificate requests.	certreq.ar m	any valid file name
DEFAULT_CERTIFICATE_ NAME_ARM	The default file name used for certificates in base64 format.	cert.arm	any valid file name
DEFAULT_CERTIFICATE_ NAME_DER	The default file name used for certificates in binary format.	cert.der	any valid file name
DEFAULT_CRYPTOGRAPHIC _MODULE	The default cryptographic module file name.	null	any valid file name and location
DEFAULT_CRYPTOGRAPHIC _TOKEN_SECONDARY _KEYDB	The default secondary database used for cryptographic modules.	null	any valid file name and location
DEFAULT_PASSWORD_ STASHING_STATE	The default value of the password stashing check box.	false	true or false
DEFAULT_PASSWORD_ V1STASHING_STATE	The default value of the password stashing	false	true or false

Table 2. iKeyman settings

Setting	**Description**	**Default Value**	**Accepted Values**
	in old format check box.		
DEFAULT_SSCERT_ BASIC_CONSTRAINTS	The default value for the basic constraints extension.	false	true or false
DEFAULT_FIPS_MODE _PROCESSING	If set to true, forces the use of the FIPS JCE Provider. If set to false, uses whichever JCE Provider comes first in the **java.security** file.	false	true or false
VIEW_HIDDEN_FILES	Whether hidden files are displayed in the file chooser.	false	true or false
DEFAULT_CMS_PASSWORD	A default password used for CMS keystore creation.	null	any valid password
DEFAULT_CERTIFICATE _KEYSIZE	The default value for certificate keysize.	1024	512 or 1024 or 2048
DEFAULT_CERTIFICATE_LIST _OPTION	The default list option for the **-cert -list** command.	all	all or personal or CA
DEFAULT_CERTIFICATE _DEFAULT	The default value for the certificate default status.	false	true or false
DEFAULT_CERTIFICATE_TRUST	The default value for the certificate trust status.	true	true or false
DEFAULT_CERTIFICATE_EXPIRE	The default certificate expiry time (in days).	365	0 to 7300
DEFAULT_CERTIFICATE _FORMAT	The default certificate input/output format.	ascii	ascii or binary
DEFAULT_CERTIFICATE_COUNTRY	The default country code for the subject name “C=” item.	Blank or selected based on locale.	E.g. “AU”, “DE” etc.
DEFAULT_ENGLISH_	If set to true, error	false	true or

Table 2. iKeyman settings

Setting	Description	Default Value	Accepted Values
ERROR_MESSAGE	messages will be displayed in English, no matter what the locale.		false
SECRET_KEY_FUNCTION _ENABLED	Whether the secret key functionality is enabled.	true	true or false
RENAME_CERT_FUNCTION _ENABLED	Whether the certificate rename functionality is enabled.	true	true or false
SIGNATURE_ALGORITHM _FUNCTION_ENABLED	Whether the signature algorithm specification function is enabled.	true	true or false
NO_DEFAULT_CERTS_ FUNCTION_ENABLED	Whether new key databases will be populated by default.	true	true or false
DEFAULT_FILE_CREATE _PERMISSION[1]	The default permissions string (format: rwxrwxrwx) for newly created files.	Let OS default decide.	format: rwxrwxrwx
ADD_CMS_SERVICE_PROVIDER_ENABLED	Adds the "IBM CMS provider" to the next position available provided the CMS provider is not already installed.	false	true or false

[1] Note for DEFAULT_FILE_CREATE_PERMISSION:

1. It only affects all newly created file (including keystore DB file, keystore stash file, cert request db file, cert file, and cert request file) permissions which are created by the current command. It doesn't affect existing files.
2. If this parameter is not set or set to an empty string "", then by default the OS decides the permissions of all new files.
3. If Java version is 6 or lower, then the OS decides the permissions of all new files.
4. If this parameter is set to a value other than an empty string and Java version is 7 or higher, then:

- On Linux, the permissions of all new files are set according to the PERMISSION string. iKeycmd/iKeyman just passes those permissions through to the OS. It works in the same way as the "chmod" command.
- On Windows, if the PERMISSION string starts with "rw", then the current user (the owner) permission is set as full permissions (read and write) and all other users or groups have no access to the new files. No other file permission setting is supported for Windows. If the PERMISSION string does not start with "rw", it will be ignored as an error (logged to the trace) and let OS decide the permissions.

System properties

When launching iKeyman it is possible to specify a number of Java system properties. These properties are specified as name-value pairs prepended by **-D**.

For example, to run ikeyman with tracing then run:

```
ikeyman -Dkeyman.debug=true
```

Then in the directory where ikeyman was run will be the file debugTrace.log.N (N is 0, 1 …)

Every property supported by the settings file is also supported as a system property,

for example, we can run iKeyman as follows:

```
ikeyman - DADD_CMS_SERVICE_PROVIDER_ENABLED =true
```

as well as the following is supported:

keyman.settings=<settings file location>

Specifies the name and location of the settings file to use. The settings file name must end in ".properties".

keyman.debug=true

Turns on debug mode, which displays the stack trace of exceptions when run in CLI mode. Also enables trace logging.

keyman.logging=true

Enables trace logging.

Starting iKeyman

You can start iKeyman in either GUI or Command Line mode.

GUI

To run iKeyman in GUI mode, execute the following command:

```
$JAVA_HOME\jre\bin\ikeyman.exe [properties]
```

where [properties] can be zero or more system properties.

Command Line

To run iKeyman in CLI mode, execute the following command:

```
$JAVA_HOME\jre\bin\ikeycmd.exe [properties] [options]
```

where [properties] can be zero or more system properties, and [options] are the command line parameters.

Commands

This chapter outlines the commands that are supported by iKeyman. It contains the following sections:

- GUI Commands
- CLI Commands
- Key Database Commands
- Certificate command
- Certificate Request Commands
- Secret (Symmetric) Key Commands
- Other commands
- Options

Each command is described in a separate section, including available modes (GUI/CLI) for each command and how to invoke it in the supported mode(s).

GUI Commands

All GUI commands are invoked by pressing a button or selecting a menu item. Screenshots outlining the dialogs a user should expect to see are shown for each command.

Most commands require the user to press a button in the **Key Database Content Panel**. This panel is used to display the various different types of entries in a key database. The different entry types can be displayed by selecting a value in the dropdown box at the top of the panel. Selecting a different value in the list also changes the buttons displayed along the right side of the panel.

The values in the dropdown list are:

Personal Certificates

This displays the entries in the key database that have a certificate and associated private-public key pair. The buttons shown down the right side are referred to as the **Personal Certificates Panel** in the command descriptions.

Signer Certificates

This displays the entries in the key database that have a certificate and associated public key. The buttons shown down the right side are referred to as the **Signer Certificates Panel** in the command descriptions.

Personal Certificate Requests

This displays the entries in the key database that have a certificate request and associated private/public key pair. The buttons shown down the right side are referred to as the **Personal Certificate Requests Panel** in the command descriptions.

Secret Keys

This displays the entries in the key database that have a secret (symmetric) key. The buttons shown down the right side are referred to as the **Secret Keys Panel** in the command descriptions.

CLI Commands

CLI commands are invoked using the following basic structure:

```
ikeycmd object action options
```

where:

- *options* are the options associated with the specified object and task;
- *action* is the specific action to be taken on the object;
- *object* is one of the following:

  -keydb
  Actions acted on a key database.
  -cert
  Actions acted on a certificate stored within an identified key database.
  -certreq
  Actions acted on a certificate request stored within an identified key database.
  -seckey
  Actions acted on a secret (symmetric) key stored within an identified key database.
  -version
  Displays version information for iKeyman.
  -help
  Displays help for the CLI commands.

Key Database Commands

The key database commands are associated with the **-keydb** object in CLI mode. The following key database actions are supported:

- Create a Key Database
- Change the Password of an existing Key Database
- Convert a Key Database to another format (Save As)
- Open a Key Database
- Delete a Key Database
- Display the password expiry date of a CMS Key Database
- Stash the password of an existing Key Database
- List the supported Key Database types

The sections below describe how to use each of the key database commands, and what options are available for each command.

Create a Key Database

Note:
Not available for PKCS#11.

The create command creates a new key database. During the creation process several files may be created. If the new key database is of the CMS type, the database itself (by default with a .kdb extension) as well as a certificate request store (by default with a .rdb extension) is created. For other database types only the database itself is created; the certificate request store is created when needed.

The key database is used to store all personal certificates and private keys, signer certificates, and symmetric keys (if these are supported by the key database type). The certificate request file stores all certificate requests and associated private keys.

Once the key database has been created it may be automatically populated with a number of predefined trusted certificate authority (CA) certificates. By default this does not occur, however if the setting `NO_DEFAULT_CERTS_FUNCTION_ENABLED` is set to *false* the predefined certificates will be added to every new key database. The list of CA certificates that are added can be determined using the list signer certificates command. Any of these CA certificates can be removed from the key database using the delete certificate command in this manual.

Creating a new key database in the GUI closes any currently open key database and opens the newly created database.

Attention: It is the user's responsibility to manage and maintain the certificates in all key databases. The user must monitor certificates for expiry and should only include those CA certificates that they will need. They must also ensure that all predefined certificates in iKeyman match those supplied by their respective issuer in order to ensure that they have not been tampered with. By default no CA certificates are added to the key database

and adding these becomes a manual process in order to improve the security of the application, as it reminds the user to only add the CA certificates that are required.
COMMAND LINE

The command is invoked as follows:

```
ikeycmd -keydb -create [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** (optional)
- **-type** (optional)
- **-expire** (optional)
- **-stash** (optional)
- **-populate** (optional)
- **-label** (optional, one or some of entrust,verisign,thawte)
- **-v1stash** (optional)

For details of these options, see Options.

GUI

The GUI command can be invoked in either of the following ways:

- From the menu, select **Key Database File** then **New**.
- From the toolbar, click the new file icon.

A dialog prompts you to enter the file name, location and type of the new key database that will be created. A further dialog gathers password data. For CMS key databases, you can specify a password expiration time (deprecated: see Appendix A. Differences from iKeyman 7) and whether or not to stash the password to a file.

Change the Password of an existing Key Database

Note:
Not available for PKCS#11.

The change password command allows the user to change the password associated with the specified key database. When changing the password for a key database, all key records containing encrypted private key information have the private key data re-encrypted. The new password is used as input to create the new encryption key used during the encryption process.

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -changepw [options]
```

The permitted options for this command are:

- **-db** (required)
- **-new_pw** (required)
- **-pw** or -stashed(optional)
- **-type** (optional)
- **-expire** (optional)
- **-stash** (optional)
- **-v1stash** (optional)

GUI

The command can be invoked from the menu by selecting **Key Database File** then **Change Password**. Invoking the command displays a dialog for either CMS key databases or other file-based key database types. The information gathered by the forms is used to modify the password of the currently open key database.

Convert a Key Database to another format (Save As)

Note:
Not available for PKCS#11.

The convert (Save As) command creates a new key database containing the same entries as the source database. The command allows for the new database to be of a different type and use a different password. If the target type doesn't support certain entries in the source database, these are ignored. The source database is not altered or deleted.

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -convert [options]
```

The permitted options for this command are:

- **-db** (required)
- **-new_format** (required)
- **-pw** or -stashed (optional)
- **-new_pw** (optional)
- **-target** (optional)
- **-old_format** or **-type** (optional)
- **-expire** (optional)
- **-stash** (optional)
- **-v1stash** (optional)

GUI

The command can be invoked from the menu by selecting **Key Database File** then **Save As**, or by selecting the **Save** icon from the toolbar.

Invoking the command displays the same sequence of dialogs as the **create new key database** command. Rather than creating a blank key database of the specified type, the new key database will be a copy of the currently open database. Once the command has completed, the newly created key database is opened.

Open a Key Database

The open command is a GUI specific operation which opens a particular key database and allows the user to perform operations on it. As long as the key database remains open it does not need to be manually saved when it is modified, as this happens automatically. The iKeyman GUI can only open one key database at a time (except for a secondary database for a PKCS11 token).

Command Line

Not applicable - the CLI runs one command at a time and therefore does not keep key databases open.

GUI

The command can be invoked from the menu by selecting **Key Database File** then **Open**, or by clicking the **Open** icon on the toolbar. A dialog prompts you to enter the data required to open a key database. A further dialog prompts for the password that will be used to open the key database. Additional dialogs gather details for PKCS11Direct or PKCS11Config key databases, if selected.

Delete a Key Database

Note:
Not available for PKCS#11.

The delete key database command deletes the identified key database. When identifying the key database you simply need to specify the file name of the key database. The request database files are removed automatically during the process. If a stash file was created it is not removed.

If a password is required for this key database, it is prompted for if not provided, and is used to ensure that the user is actually allowed to delete the key database. If the password is not correct the key database is not deleted.

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -delete [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** or -stashed (optional)
- **-type** (optional)

GUI

Not applicable - key databases cannot be deleted in GUI mode.

Display the password expiry date of a CMS Key Database

Notes:

1. Only applicable to CMS key databases.
2. The database password expiry function has been deprecated.

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -expiry [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** (optional)
- **-type** (optional)

GUI

The command can be invoked from the menu by selecting **Key Database File** then **Display Password Expiry**. If there is no password expiry date set for the key database, the following message is displayed:

```
The password doesn't expire.
```

If the key database does have a password expiry date, a dialog window indicates the date and time when the password will expire.

Stash the password of an existing Key Database

Note:
Only applicable to CMS and PKCS12 key databases.

The stash password command takes an existing key databases password and stashes it to a file. The reason that a user would want to stash a password for a key database is to allow the password to be recovered from the file when automatic login is required. The output of the command is a single file with the name of the key database with a ".sth" extension.

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -stashpw [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** (optional)
- **-type** (optional)
- **-v1stash** (optional)

GUI

The command can be invoked from the menu by selecting **Key Database File** then **Stash Password**. If the command succeeds, a confirmation message indicates the name of the stash file that was created.

List the supported Key Database types

This command lists the database types that are currently supported by iKeyman. This only includes file-based types (not PKCS11, MS-CAPI).

Command Line

The command is invoked as follows:

```
ikeycmd -keydb -list
```

This command takes no additional options.

GUI

Not applicable - there is no command to display the supported types in GUI mode. The supported types can be seen in the **Key database type** field of the **open** dialog (see Open a Key Database).

Certificate command

The certificate commands are associated with the -cert object in CLI mode. The following certificate actions are supported:

- Add a certificate
- Populate a key store
- Create a self-signed certificate
- Delete a certificate
- Display details of a certificate
- Export a certificate
- Extract a certificate
- Import a certificate
- List Certificates
- List details of default certificate
- List Signer Certificates
- Modify a Certificate
- Receive a Certificate
- Set Default Certificate
- Sign a Certificate Request
- Rename a Certificate

The following sections go into details on how to use each of the certificate commands and what options are available for each command.

Add a certificate

The add certificate command stores a CA certificate in the identified key database. The CA certificate is received as a file with the data encoded in one of the following formats:

- Base64
- DER
- PKCS#7
- S/MIME

The encoding type does not need to be supplied, as it will be automatically determined.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -add [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required)
- **-file** (required)
- **-label** (optional)
- **-pw** or -stashed (optional)
- **-type** (optional)
- **-format** (optional)
- **-trust** (optional)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

To add a certificate:

1. Click the **Add** button on the Signer Certificates panel.
2. Enter the file name and location of the certificate to open and click **OK**.
3. A dialog displays the contents of the selected certificate source (either a file or a key database) and allows you to choose which ones to add. The list contains the labels of all certificates in the source. To select multiple entries, hold down the SHIFT (for a range of entries) or CTRL key (for multiple separated entries) while clicking the desired entries.
4. A further dialog displays the certificates selected. If required, you can rename labels by selecting a certificate, entering a new label and clicking **Apply**. Click **OK** to complete the import process.

Populate a key store

The populate command allows the user to add predefined CA certificates (the same ones as the key database create command may add) to a key database. This command is only available if NO_DEFAULT_CERTS_FUNCTION_ENABLED is set to *true*.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -populate  [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (optional, one or some of entrust,verisign,thawte)
- **-pw** or -stashed (optional)
- **-type** (optional if **-db** present)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked by clicking the **Populate** button in the Signer Certificates Panel. A dialog allows the user to choose certificates from a list of predefined CA certificates to be added to the currently open key database.

Create a self-signed certificate

A self-signed certificate provides a certificate that can be used for testing while waiting for an officially signed certificate to be returned from the CA. Both a private and public key are created during this process.

The **create self-signed certificate** command creates a self-signed X509 certificate in the identified key database. A self-signed certificate has the same issuer name as its subject name.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -create [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -stashed (optional)
- **-dn** (optional)
- **-type** (optional if **-db** present)
- **-expire** (optional: default is *365*)
- **-size** (optional: default is *1024*)
- **-x509version** (optional: default is *3*)
- **-default_cert** (optional)
- **-ca** (optional: default is *false*)
- **-eku** (optional)
- **-ku** (optional)

- **-sig_alg** (optional: default is *SHA1WithRSA*)
- **-san_dnsname** (optional)
- **-san_emailaddr** (optional)
- **-san_ipaddr** (optional)

GUI

The command can be invoked in any of the following ways:

- by selecting **Create** then **New Self-Signed Certificate** from the menu, or
- clicking the **New Self-Signed** button on the toolbar, or
- clicking the **New Self-Signed** button on the Personal Certificates Panel.

A dialog allows the user to specify the certificate and key data for a new self-signed certificate.

Delete a certificate

The delete command simply removes the certificate with the identified label. Once the delete operation is complete, there is no way of recovering the certificate unless you add the certificate back into the key database.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -delete [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -stashed (optional)
- **-type** (optional if **-db** present)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked in any of the following ways (the buttons mentioned are only enabled if an entry in the list is selected):

- For personal certificates, click the **Delete** button on the Personal Certificates Panel.
- For signer certificates, click the **Delete** button on the Signer Certificates Panel.

A dialog confirms that you wish to delete the selected entry from the key database. Click **Yes** to delete it.

Display details of a certificate

The display certificate details command displays the different details associated with the identified certificate. The details displayed include (but not limited to):

- The label of the certificate.
- The size of the key associated with the certificate.
- The X509 version that the certificate was created.
- The serial number for the certificate.
- The issuer and subject distinguished names.
- The certificate's validity period.
- The fingerprint of the certificate (Both SHA1 and SHA256 fingerprints are displayed)
- The signature of the algorithm used during creation of the certificate.
- Certificate extensions.,
- An indication of the certificate's trust status.

If more details for the certificate are required use the **-showOID** option in the CLI, and the **View Details** button in the GUI. These options display a more detailed listing of the certificate details.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -details [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -stashed (optional)
- **-type** (optional if **-db** present)
- -**showOID** (optional)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked in one of the following ways (the buttons mentioned are only enabled if an entry in the list is selected):

- For personal certificates, click the **View/Edit** button in the Personal Certificates Panel.
- For signer certificates, click the **View/Edit** button in the Signer Certificates Panel.
- For either certificate type, double click the certificate in the list.

Invoking the command displays a dialog showing details of the certificate. Click **View Details** for additional information.

Export a certificate

The export command exports a single certificate specified by its label from one key database to another. During this process no key generation occurs. On successful completion the identified certificate will be in both the source and destination key databases.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -export [options]
```

The permitted options for this command are:

- **-db** (required)
- **-label** (required)
- **-target** (required)
- **-pw** or **-stashed** (optional)
- **-target_pw** or -**target_stashed** (optional)
- **-target_type** (optional)
- **-type** (optional)
- **-encryption** (optional)

GUI

To invoke the command from the menu, click the **Export/Import** button in the Personal Certificates Panel (the **Export/Import** button replaces the **Import** button when an entry in the list is selected).

Extract a certificate

The extract certificate command extracts the certificate data from the key database and places it into a file. If the file does not exist it will be created. If it does exist, it will be overwritten. The data will be saved in either Base-64 or binary DER encoding.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -extract [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-target** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

- **-format** (optional: default is *ascii*)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked in one of the following ways:

- For personal certificates, click the **Extract Certificate** button in the Personal Certificates Panel.
- For signer certificates, click the **Extract** button in the Signer Certificates Panel.

Note:
these buttons are only enabled if an entry in the list is selected.
A dialog allows you to choose the data type, file name and location of the certificate.

Import a certificate

The import command imports certificates from one key database to another. The target key database may be either a file-based key database, or a PKCS#11 one. During this process no key generation occurs. On successful completion the identified certificates will be in both the source and destination key databases.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -import [options]
```

The permitted options for this command are:

- **-db** or **-file** (required)
- **-target** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-pw** or -**stashed** (optional)
- **-pw (second password)** (optional if **-crypto** present)
- **-target_pw** or -**target_stashed** (optional if **-target** present)
- **-type** (optional if **-db** present)
- **-label** (optional)
- **-target_type** (optional if **-target** present)
- **-new_label** (optional if **-label** present)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked by clicking the **Import** button in the Personal Certificates Panel. Select the key file type, file name and location to import a certificate from another key database.
Note:
the **Import** button appears as **Import/Export** if an entry in the list has been selected. From the Import/Export dialog, you can select the **Import Key** radio button and then enter details as described above.

List Certificates

The list certificate command lists certificates stored within the identified key database. By default all certificates are displayed, however the `<list filter>` parameter can be used to display a subset of the certificates. The expiry parameter can also be used to filter the certificates being displayed.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -list [options]
```

The permitted options for this command are:

- **<list filter>** (optional: default is *all*)
- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-pw** or -stashed (optional)
- **-type** (optional if **-db** present)
- **-expiry** (optional)
- **-label** (optional)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

Not applicable - refer to *Open a Key Database*. When a key database is opened, all items are shown in the various lists. See GUI Commands for details of viewing the various lists.

List details of default certificate

Note:
Only applicable to CMS.

This command selects the default certificate (if there is one) and displays the same details as the **display details** command. If there is no default certificate, a message indicating this is displayed.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -getdefault [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

Not applicable - the default certificate in a CMS key database (if there is one) is indicated by an asterisk (*) next to the alias in the **Personal Certificates** list. For details of viewing the default certificate, see Display details of a certificate.

List Signer Certificates

The list signers command displays a list of the CA certificates that are included in iKeyman and can be added with the populate command. The certificates are displayed in a tree format, grouped by their issuer.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -listsigners
```

This command takes no additional options.

GUI

Not applicable - refer to Populate a key store. The available signers are listed in the selection dialog.

Modify a Certificate

Note:
Only applicable to CMS key databases.

The **modify** certificate command allows a CA certificate's trust status to be enabled or disabled. When a CA certificate's trust status is enabled, then that CA certificate is permitted to be involved in a certificate chain validation. If the CA certificate's trust status is disabled then it cannot be used to validate any certificates.

For example, if certificate "ABC" is signed by the CA certificate "VeriSign CA" and "VeriSign CA" is not marked as trusted, the validation of "ABC" will fail.

You are able to have any number of trusted CA certificates in the single key database.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -modify [options]
```

The permitted options for this command are:

- **-db** (required)
- **-label** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional)
- **-trust** (optional: default value is determined by settings in the settings file)

GUI

This command is invoked via the certificate details dialog of a signer certificate in a CMS key database (see Display details of a certificate). For CMS key databases, the details dialog for signer certificates has an additional option: to modify the trust of a signer certificate, select the **Set the certificate as a trusted root** checkbox and click **OK**.

Receive a Certificate

The receive certificate command stores a certificate received from a CA that was requested to sign a certificate request. The certificate being received can be in any of the following formats:

- Base64
- DER
- PKCS#7
- S/MIME

The type does not need to be specified, as it will be automatically determined. Specifying a type has no effect and is only included for backward compatibility.

While receiving the certificate, the certificate is matched to its corresponding certificate request. This certificate request is removed from the key database as it is no longer needed.

If the certificate request is required after receiving the certificate, you will need to use the recreate certificate request command that can be found in chapter four of this manual.

This command can also be used in a certificate renewal operation. Rather than replacing the corresponding certificate request, there will be a certificate which is replaced.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -receive [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-file** (required)

- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-format** (required)
- **-default_cert** (optional)

GUI

The command can be invoked by clicking the **Receive** button on the Personal Certificates Panel.

Set Default Certificate

Note:
Only applicable to CMS key databases.

The set default certificate command sets a certificate to be used as the default certificate for the identified key database. During this command the current default certificate, if there is one, has its default setting removed. The new certificate is then set as the default certificate. There can only ever be one default certificate in a key database.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -setdefault [options]
```

The permitted options for this command are:

- **-db** (required)
- **-label** (required)
- **-pw** (optional)
- **-type** (optional)

GUI

This command is invoked via the certificate details dialog of a personal certificate in a CMS key database (see Display details of a certificate). For CMS key databases, the details dialog for personal certificates has an additional option: to modify the default status of a signer certificate, select the **Set the certificate as the default** checkbox and click **OK**.

Sign a Certificate Request

The sign certificate command allows the signing of a certificate request by an existing certificate stored within a key database. The command accepts a certificate request in its file format and a label of a certificate stored within the key database that contains the private key to be used during the signing process.

Command Line

The command is invoked as follows:

```
ikeycmd -cert -sign [options]
```

The permitted options for this command are:

- **-db** (required)
- **-label** (required)
- **-file** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional)
- **-expire** (optional: default is *365*)
- **-format** (optional: default is *ascii*)
- **-target** (optional: default is *cert.arm*)
- **-eku** (optional)
- **-ku** (optional)
- **-sernum** (optional)

GUI

This function is not available in the GUI.

Rename a Certificate

The rename command renames a certificate entry in the key database and leaves it otherwise unchanged. The new label must not already exist in the key database.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -rename [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-label** (required)
- **-new_label** (required)

GUI

The command can be invoked in one of the following ways:

- For personal certificates, click the **Rename** button in the Personal Certificates Panel.
- For signer certificates, click the **Rename** button in the Signer Certificates Panel.

Note:
these buttons are only enabled if an entry in the list is selected.
A dialog prompts you to enter the new label.

Validate a Certificate

The validate command validates the certificate path or certificate chain. Note: SSL validation of the certificate may give different results depending on environment. e.g. access to CRL distribution points.

Command Line

The command is invoked as follows:

```
 ikeycmd -cert -validate [options]
```

The permitted options for this command are:

- **-db** (required)
- **-pw** or -**stashed** (optional)
- **-label** (required)

GUI

The command can be invoked by clicking the **Validate** button on the Personal Certificates Panel.

Certificate Request Commands

The certificate request commands are associated with the **-certreq** object in CLI mode. The following certificate request actions are supported:

- Create Certificate Request
- Delete Certificate Request
- Display details of a Certificate Request
- Extract Certificate Request
- List all Certificate Requests
- Recreate Certificate Requests

The following sections detail how to use each of the certificate request commands, and what options are available for each command.

Create Certificate Request

The create certificate request command creates a new private-public key pair using the specified algorithm and a PKCS10 certificate request in the specified key database. The certificate request is stored in the certificate request database of the specified key database and is also extracted to a file that can be used to send the certificate request to a CA for signing. This file doesn't contain any private key material; this is only stored in the certificate request database.

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -create [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-file** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-dn** (optional)
- **-size** (optional: default is *1024*)
- **-eku** (optional)
- **-ku** (optional)
- **-sig_alg** (optional: default is *SHA1WithRSA*)
- **-san_dnsname** (optional)
- **-san_emailaddr** (optional)
- **-san_ipaddr** (optional)

GUI

The command can be invoked in one of the following ways:

- From the menu. select **Create** then **New Certificate Request**.
- From the toolbar, click the **New Certificate Request** icon.
- In the Personal Certificate Requests panel, click the **New** button.

A dialog prompts you to specify the certificate request and key data for the new certificate request.

Delete Certificate Request

The delete certificate request removes the certificate request from the identified key database. This means that the entry in the certificate request database associated with the certificate request is deleted. The key/certificate request cannot be recovered after this operation.

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -delete [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **Delete** button on the Personal Certificate Request panel.
Note:
this button is only enabled when a certificate request is selected in the list.

Display details of a Certificate Request

The list certificate request details command lists the identified certificate requests details. These details include:

- The label of the certificate request.
- The size of the key associated with the certificate request.
- The subject distinguished name.
- The fingerprint of the certificate.
- The signature of the algorithm used during creation of the certificate.

For a more detailed listing of the certificate requests details use the **-showOID** option in the command (for the CLI) or press the **View Details** button (for the GUI).

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -details [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-showOID** (optional)

GUI

The command can be invoked by clicking the **View** button in the Personal Certificate Request Panel, or by double-clicking the certificate request entry in the list.
Note:
the **View** button is only enabled when a certificate request is selected in the list.
A dialog displays details of the certificate request.

Extract Certificate Request

The extract certificate request command extracts an existing certificate request stored in the specified key database to the identified file in base64 or binary format. The certificate request will still exist within the key database so you are able to extract it as many times as needed. The file that is extracted is the file that is dispatched to a CA for signing.

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -extract [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-target** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-format** (optional)

GUI

The command can be invoked by clicking the **Extract** button in the Personal Certificate Requests panel.

Note:

the **Extract** button is only enabled when an entry in the list is selected.

List all Certificate Requests

The list certificate request command lists all of the certificate request labels stored within the identified key database.

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -list [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

Not applicable - refer instead to Open a Key Database. When a key database is opened, all items are displayed in the various lists. See the GUI Commands section for details of viewing the various lists.

Recreate Certificate Requests

The recreate certificate request command recreates a certificate request from an existing certificate stored within the specified key database. The recreation of a certificate may be required to allow a certificate to be signed by another CA in case there was a problem with the CA that originally signed it.

Command Line

The command is invoked as follows:

```
ikeycmd -certreq -recreate [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-target** (required)
- **-pw** or -**stashed** (optional)
- **-eku** (optional)
- **-ku** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **Recreate Request** button in the Personal Certificates panel. A dialog prompts you to choose the target file for the recreate certificate request operation.

Secret (Symmetric) Key Commands

Note:
Only applicable to JCEKS and PKCS#11.

The secret key commands are associated with the **-seckey** object in CLI mode. The following secret key actions are supported:

- Create Secret Key
- Delete Secret Key
- Display Secret Key details
- Export Secret Keys
- Import Secret Keys
- List all Secret Keys
- Rename Secret Key

The following sections go into detail on how to use each of the secret key commands and what options are available for each command.

Create Secret Key

The create secret key command creates one or more symmetric keys and adds these to the key database. The command allows the user to specify what symmetric encryption algorithm to create the key(s) for, as well as the desired key size.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -create [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** or **-labelrange** (required)
- **-keyalg** (required)
- **-keysize** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **New** button in the Secret Key panel. A dialog prompts you to specify the data for a new secret key(s).

Delete Secret Key

The delete secret key command removes a symmetric key entry from a key database. The key cannot be recovered after deleting it.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -delete [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** or **-labelrange** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **Delete** button in the Secret Key panel.
Note:
the **Delete** button is only enabled if an entry is selected in the list.

Display Secret Key details

The details command displays the details of a symmetric key. The following details are displayed:

- Symmetric key label
- Algorithm OID
- Key Size

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -details [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **View** button in the Secret Key panel, or by double-clicking a secret key in the list.

Export Secret Keys

The export command encrypts one or more symmetric keys in a key database and outputs these to a file. The symmetric keys are encrypted using one of the public keys in the key database, so a public key must be chosen using the **-keyalias** option.

The command can be used to back up symmetric keys to another key database. The keys are encrypted before being output to a file so that they remain confidential. The key database that the symmetric keys will be imported to must contain the private key corresponding to the public key that the export command uses for encryption.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -export [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** or **-labelrange** (required)
- **-keyalias** (required)
- **-file** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **Export** button in the Secret Key panel. A dialog prompts you to choose the target file for the export secret key operation. A further dialog prompts you to choose the encryption or decryption key for the secret key export or import operation.

Import Secret Keys

The import command is used to add symmetric keys from a file into a key database. The file, which is produced using the export command, contains encrypted symmetric keys. The target key database of the import command must contain the private key corresponding to the public key that was used to encrypt the symmetric keys.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -import [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** or **-labelrange** (required)
- **-keyalias** (required)
- **-file** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

The command can be invoked by clicking the **Import** button in the Secret Key panel. A dialog prompts you to specify the source file for the import secret key operation.

List all Secret Keys

The list command displays the labels of all symmetric keys in the specified key database.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -list [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)

GUI

Not applicable - refer instead to *Open a Key Database*. When a key database is opened, all items are displayed in the various lists. See the GUI Commands section for details of viewing the various lists.

Rename Secret Key

The rename command changes the label of a symmetric key entry in a key database. The entry is otherwise unaffected.

Command Line

The command is invoked as follows:

```
ikeycmd -seckey -rename [options]
```

The permitted options for this command are:

- **-db** or **-crypto** (required)
- **-relativeSlotNumber** or **-tokenlabel** (required if **-crypto** present)
- **-label** (required)
- **-new_label** (required)
- **-pw** or -**stashed** (optional)
- **-type** (optional if **-db** present)
- **-secondaryDB** (optional if **-crypto** present)
- **-secondaryDBpw** (optional if **-secondaryDB** present)
- **-secondaryDBType** (optional if **-secondaryDB** present)

GUI

The command can be invoked by clicking the **Rename** button in the Secret Key panel. A dialog prompts you to specify the new label.

Other commands

Add provider

The add provider command can be used to temporarily add a Java Security Provider, which may contain required signature algorithms or key database types. Generally, all required providers should be added to the **java.security** file so that they do not need to be manually added each time the program is run (see java.security file).

A Provider is added by specifying the Java class name containing the Provider. Adding Providers that require parameters in their initialization is supported. In this case, the Provider name and parameters need to be entered. If the parameters contain spaces, they must be enclosed in double quotes. For example, to add the **PKCS11Impl** provider with a configuration file (`"c:\`*`example_directory`*`\example.conf"`), the following entry should be made in the add provider dialog:

```
com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl
"c:\example_directory\example.conf"
```

As the CLI can only execute one command at a time, this command does not apply to it. There are 2 ways to affect the Providers being used for the CLI - One is by modifying the **java.security** file and the other is using the iKeyman settings option "ADD_CMS_SERVICE_PROVIDER_ENABLED=true". Please refer to Table 2. iKeyman settings..

Command Line

Not applicable.

GUI

The command can be invoked in either of the following ways:

- Select **Key Database File** from the menu, then click the **New Provider** icon.
- Click the **New Provider** icon on the toolbar.

A dialog displays the currently installed Providers and prompts you to enter the class of the new Java Security Provider.

Version Help

The version command displays version information associated with the currently installed iKeyman program. It displays the following data:

- iKeyman version (**major.minor.build**)
- CMS Provider version (**major.minor**) or "No CMS Provider" if the provider is not installed
- Java version (product version)
- Copyright notice

Command Line

The command is invoked as follows:

```
ikeycmd -version
```

This command takes no additional parameters.

GUI

The command can be invoked from the menu by clicking **Help** then **About**. A dialog displaying the iKeyman version information is displayed.

Help

The help command displays a list of all `<object>` and `<action>` pairs that are available. The output is the same as running the command line without any parameters.

To get help on a specific command, enter the `<object>` and `<action>` for that command without any parameters.

Command Line

The command is invoked as follows:

```
ikeycmd -help
```

This command takes no additional parameters.

GUI

There is no comparable feature in GUI mode, although most controls and fields have tool tips available to explain their functions.

Options

The following is a list of all options the iKeyman CLI supports. For each option the following details are specified:

Option
: The command line tag used for the option.

Parameters
: The number of parameters this option takes. Some options can be used in different ways and therefore have multiple values in the Parameters column.

Commands
: The commands the option applies to. If the option is used in different ways for different commands, this cell is split into multiple rows with the associated use next to the commands.

Use
: How the option is used for the commands indicated in the Commands column

Accepted Values
: The values that are accepted for the option.

Table 3. options

Option	**Para-meters**	**Commands**	**Use**	**Accepted Values**
-ca	1	-cert -create	If *true*, adds the Basic Constraints extension to the certificate being created. The extension is marked as critical with a path length of 2147483647.	*true* or *false*
-crypto	0 or 1	-cert -add -create -delete -details -extract -import -list -populate -receive -rename -certreq -create -delete -details -extract -list -recreate	This option can be used in one of the following ways: • Used to specify the file name of the cryptographic token library. • A switch indicating that this is a token operation using the default library indicated in the settings file. • Indicates that a PKCS#11 token is accessed using the configuration	Any valid library file name

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
		-seckey	method with the label specified by the **-tokenlabel** parameter.	
-db	1	-keydb all except -list -cert all except -listsigners -certreq all -seckey all	The name of the primary key database used in the command.	Any valid file name.
-default_cert	1	-cert -create -receive	Whether to set a particular certificate as the default certificate (only applicable to CMS key databases).	*yes* or *no*
-dn	1	-cert -create -certreq -create	The distinguished name of the entity being created.	Comma-delimited string of DN components.
-encryption	1	-cert -export	**Obsolete: Included for backward compatibility.** Setting this value has no effect.Whether to use strong or weak encryption.	*strong* or *weak*
-expire	1	-keydb -changepw -convert -create	**Deprecated** (see Appendix A. Differences from iKeyman 7) The database password expiry (in days).	0 to 7300
		-cert -create -sign	The certificate expiry (in days).	
-expiry	0 or 1	-cert -list	Expiry of a certificate. Filters the certificates	Any positive integer.

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
			being listed by their expiration date and displays the validity ranges of the certificates. If a value is provided, lists all certificates that will be expired `<value>` days from now (a value of 0 will list the currently expired certificates). If the value is omitted, all certificates are listed.	
-file	1	-cert -add -receive	The file to retrieve one or more certificates from.	Any valid file name.
		-cert -import	The key database file to import certificates from. Must be a PKCS12 file.	
		-cert -sign	The file containing the certificate request to be signed.	
		-certreq -create	The file to which the certificate request will be written.	
		-seckey -import -export	The file to import/export secret keys to/from.	
-format	1	-cert -add -extract -receive -sign -certreq -extract	The input/output format of the certificate/certificate request file. The option is ignored for input format as the type is automatically determined in those cases.	*binary* or *ascii*
-keyalg	1	-seckey -create	The name of the algorithm of the symmetric key(s) to be created.	Any supported symmetric key algorithm (currently AES, DES, DESede).
-keyalias	1	-seckey -export	Alias of the public/private key to use for	Any public/private key alias in the key database.

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
		-import	encryption/decryption of the secret key(s).	
-keysize	1	-seckey -create	The size of the symmetric key(s) to be created.	A valid key size for the specified algorithm
-label	1	-cert -add -create -delete -details -export -extract -modify -setdefault -certreq -create -delete -details -extract -recreate -seckey -create -delete -details	The label of the key database entry to perform the command on.	A valid label, or a valid list of labels (if applicable).
		-cert -populate	A comma-delimited list of labels to add to the key database. The labels must match those returned by the **-cert -listsigners** command.	
		-seckey -export	A comma-delimited list of secret keys to export to a file.	
		-cert -import	The label of the entry to import. Imports all if omitted.	
		-cert -sign	The label of the private key item in the key database to sign the certificate request with.	
-labelrange	1	-seckey -create	The label range of the keys to be created.	A valid range consists of a prefix consisting of

Table 3. options

Option	**Para-meters**	**Commands**	**Use**	**Accepted Values**
				letters, followed by a range of hexadecimal values. The hexadecimal range must be separated by a hyphen (**-**) and the first value must be less than the second.
<list filter>	0	-cert -list	This parameter is different to all others, in that it is not preceded by a hyphen (**-**). If specified, it must appear immediately after **-cert -list**. It is used to filter which certificates are displayed.	all Displays all certificates. personal Displays all personal certificates (private key entries). ca Displays all CA (Certificate Authority) certificates.
-new_format	1	-keydb -convert	The key database format being converted to.	A valid file-based key database type.
-new_label	1	-cert -rename -seckey -rename	The new label to assign to the key database entry being renamed.	Any string.
-new_pw	1	-keydb -changepw -convert	The new password used to protect the key database.	Any valid password.
-old_format	1	-keydb -convert	The key database format being converted from.	A valid file-based key database type.
-pfx	0	-cert -import	A switch indicating whether the import file is a pfx file. This switch is unnecessary if the file extension of the file is .pfx.	N/A.
-pw	1	-keydb all except -list	The password used to access the key database. The **-cert -import**	The correct password.

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
		-cert all except -listsigners -certreq all -seckey all	command can have two **-pw** options specified; one for the primary database, and one for the target cryptographic token. In this case, the first password applies to the primary database, and the second applies to the cryptographic token.	
-relativeSlot Number	1	same as -crypto	The slot number of the token being accessed.	A valid slot number.
-san_dnsname	1	-cert -create -certreq -create	The SAN DNS name(s) for the entry being created.	A comma- or space-delimited list of DNS names.
-san_emailaddr	1	-cert -create -certreq -create	The SAN email address(es) for the entry being created.	A comma- or space-delimited list of email addresses.
-san_ipaddr	1	-cert -create -certreq -create	The SAN IP address(es) for the entry being created.	A comma- or space-delimited list of IP addresses.
-secondaryDB	1	-cert -rename -add -delete -details -extract -import -list -populate	The name of the secondary key database for the cryptographic token operation.	A valid file name of a key database.
-secondaryDB pw	1	same as -secondaryDB	The password used to access the secondary key database.	The correct password to access the secondary key database.
-secondaryDB Type	1	same as -secondaryDB	The type of the secondary key database.	A valid file-based key database type.
-sernum	1	-cert -sign	The serial number for the certificate created in the	A positive integer.

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
			signing operation.	
-showOID	0	-cert -display -certreq -display	The switch is used to control the way an entry is displayed. If it is provided, the full details of the entry are displayed; if it is omitted, a summary is displayed.	N/A.
-sig_alg	1	-cert -create -certreq -create	The asymmetric signature algorithm used for the creation of the entry's key pair.	Any supported asymmetric signature algorithm. Currently: • SHA1WithRSA • SHA256WithRSA • SHA384WithRSA • SHA512WithRSA • MD2WithRSA • MD5WithRSA • SHA1WithDSA
-size	1	-cert -create -certreq -create	The signature algorithm key size used for the creation of the entry's key pair.	Any supported key size for the specified algorithm
-stash	0	-keydb -changepw -convert -create	Whether to create a stash file containing the keystore password. The stash file has the same name as the keystore, but with a ".sth" extension (Currently only supported for CMS and PKCS12 key databases).	N/A
-v1stash	0	-keydb -changepw -convert -create -stashpw	Generates an old format stash file (not recommended). Must be used with -stash parameter for commands -changepw, -convert and	

Table 3. options

Option	Para- meters	Commands	Use	Accepted Values
			-create. Note that if a stash file already exists for the keystore, and it is in a newer format than v1, then this parameter will be ignored i.e. you cannot back level a newer format stash file.	
-target	1	-keydb -convert	The target key database to convert to.	Any valid file name
		-cert -export -import	The target key database for the certificates to be imported/exported.	
		-cert -extract -sign -certreq -extract -recreate	The target certificate/certificate request file.	
-target_pw	1	-cert -export -import	The password used to access the target key database.	The correct password for the target key database.
-target_type	1	-cert -export -import	The type of the target key database.	Any supported file-based key database type.
-tokenlabel	1	same as -crypto	The label of the token being accessed.	A valid token label.
-trust	1	-cert -add -modify	Whether to trust the certificate being added/modified.	*enabled* or *disabled*
-type	1	-keydb all except -list -cert all except -listsigners -certreq all -seckey all	The type of the primary key database.	Any supported file-based key database type
-x509Version	1	-cert	The X.509 version of the	1 or 2 or 3

Table 3. options

Option	Para-meters	Commands	Use	Accepted Values
		-create	certificate being created.	

Error Codes

The following table of error codes can be returned by iKeyCmd.

Table 4. iKeyCmd error codes

Name	Return codes
OK	0
BLANK_PKCS12_CREATION	1
BLANK_PKCS12_STORE	2
CANNOT_CONVERT_PKCS11	3
CANNOT_INSTANTIATE_OBJECT	4
CANNOT_MODIFY_PERSONAL_CERT_TRUST	5
CERT_REQUEST_FILE_CORRUPTED	6
CERTIFICATE_CREATE_ERROR	7
CERTIFICATE_ENCODING_ERROR	8
CERTIFICATE_ERROR	9
CERTIFICATE_LOAD_ERROR	10
CERTIFICATE_PARSING_ERROR	11
CERTIFICATE_REQUEST_DECODING	12
CERTIFICATE_REQUEST_FILE_NOT_FOUND	13
CERTIFICATE_STORE_ERROR	14
CLASS_NOT_FOUND	15
CORRUPT_CERTIFICATE	16
DATABASE_LOCKED	17
DATABASE_PASSWORD_EXPIRED	18
DELETION_FAILED	19
DER_ENCODING_ERROR	20
ENTRY_EXISTS_FOR_KEY	21
ENTRY_EXISTS_FOR_LABEL	22
ENTRY_LOAD_ERROR	23
FILE_DELETION_FAILED	24
INPUT_FILE_NOT_FOUND	25

Table 4. iKeyCmd error codes

Name	**Return codes**
INPUT_STREAM_CLOSE	26
INVALID_ACTION	27
INVALID_ALGORITHM_PARAMETER	28
INVALID_ALGORITHM_PARAMETERS	29
INVALID_ALIAS_RANGE_FORMAT	30
INVALID_CERTIFICATE_FILE	31
INVALID_CERTIFICATE_VERSION	32
DATABASE_TYPE	33
INVALID_DATABASE_TYPE_FOR_PARAMETER	34
INVALID_DN	35
INVALID_KEY_FOR_SIGNING	36
INVALID_KEY_SIZE	37
INVALID_KEY_SIZE_FOR_ALGORITHM	38
INVALID_KEY_TYPE_FOR_KEYSTORE	39
INVALID_LIST_FILTER	40
INVALID_NUMBER_FORMAT	41
INVALID_OBJECT	42
INVALID_OPTIONAL_PARAMETER	43
INVALID_OUTPUT_MODE	44
INVALID_PARAMETER_FOR_COMMAND	45
INVALID_PASSWORD	46
INVALID_PFX_OPTION	47
INVALID_PIN	48
INVALID_REQUEST_FILE	49
INVALID_SAN	50
INVALID_SAN_IN_CERT	51
INVALID_SIGNATURE_ALGORITHM	52
INVALID_SIGNING_KEY_TYPE	53
INVALID_SLOT_NUMBER	54
INVALID_TYPE_FOR_ACTION	55
INVALID_VALUE_FOR_PARAMETER	56
IO_ERROR	57
IO_ERROR_MSG	58
KEY_DECRYPTION_ERROR	59

Table 4. iKeyCmd error codes

Name	**Return codes**
KEY_ENCRYPTION_ERROR	60
KEY_FILE_NOT_FOUND	61
KEY_STORE_ENTRY_SET_ERROR	62
KEY_STORE_INSTANTIATION_ERROR	63
KEY_STORE_LOAD_ERROR	64
KEY_STORE_TYPE_NOT_FOUND	65
KEYSTORE_CLOSE_ERROR	66
MISSING_ACTION	67
MISSING_VALUE_FOR_PARAMETER	68
NO_CA_CERT_FOR_LABEL	69
NO_CERTIFICATE_FOR_LABEL	70
NO_CRYPTO_MODULE_SPECIFIED	71
NO_DEFAULT_CERTIFICATE	72
NO_DN_OR_SAN	73
NO_ENTRY_FOR_KEY	74
NO_ENTRY_FOR_LABEL	75
NO_JCE_PROVIDER	76
NO_KEY_FOR_LABEL	77
NO_OPTIONS_SPECIFIED	78
NO_READ_PERMISSION	79
NO_REQUEST_FOR_CERTIFICATE	80
NO_REQUEST_FOR_LABEL	81
NO_SUCH_ALGORITHM	82
NO_WRITE_PERMISSION	83
NOT_A_PROVIDER	84
OUTPUT_FILE_CREATION_ERROR	85
OUTPUT_FILE_EXISTS	86
OUTPUT_STREAM_CLOSE_ERROR	87
OUTPUT_STREAM_WRITE_ERROR	88
PASSWORD_CANNOT_BE_NULL	89
PKCS11_ERROR	90
PKCS11_EXCEPTION	91
PKCS7_PARSING_ERROR	92
PRIVATE_KEY_DECRYPTION_ERROR	93

Table 4. iKeyCmd error codes

Name	Return codes
PRIVATE_KEY_ENCODING_ERROR	94
PRIVATE_KEY_ENCRYPTION_ERROR	95
PROVIDER_ALREADY_INSTALLED	96
PUBLIC_KEY_COPY_ERROR	97
RDB_PASSWORD_CHANGE_ERROR	98
REQUEST_SIGNING_ERROR	99
REQUIRED_DN_NOT_SPECIFIED	100
REQUIRED_VALUE_NOT_SPECIFIED	101
RESTRICTED_POLICY_FILES	102
SAN_ENCODING_ERROR	103
SAN_SUPPORT_NOT_ENABLED	104
SIGNATURE_ERROR	105
SOURCE_DB_IS_EMPTY	106
TOO_MANY_CHOICES_SELECTED	107
TOO_MANY_PARAMETERS_FOR_TAG	108
TOO_MANY_VALUES_FOR_TAG	109
UNKNOWN_PARAMETER	110
UNRECOVERABLE_ENTRY	111
UNRECOVERABLE_KEY	112

Accessibility

iKeyman contains several accessibility features and functions which allow individuals with disabilities to use the application. The IBM Accessibility Center and Sun Microsystems' Accessibility Group have combined efforts to design and build next-generation accessibility into the Java application. iKeyman is one of the products that currently comply with the accessibility support initiative. Features include the following:

Accessibility support for usability

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

[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
Mnemonics
All controls possess mnemonics which allow the user to access a field directly using the keyboard. The mnemonic character of each control is underlined and the control can be accessed by pressing **[Alt]+[Mnemonic Character]**.

Accessibility support for visual effect
The following accessibility options are available:

- 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.
- Enable accessibility support. Select **View** then **Windows Look and Feel**. The appearance of an iKeyman display will adopt the current system color and fonts.

Appendix A. Differences from iKeyman 7

The following features differentiate iKeyman version 8 from version 7:

Architecture
iKeyman 7 used JNI to use GSKit DLLs for access to .KDB keystore data. iKeyman8 uses a Java CMS provider to access .kdb keystores and does not use any JNI to GSKit for this.

Provider name change

iKeyman7

- provider class: com.ibm.spi.IBMCMSProvider
- keystore class: com.ibm.spi.CMSKeyStoreSpi
- provider name: IBMCMS
- keystore name: CMS

iKeyman 8

- provider class: com.ibm.security.cmskeystore.CMSProvider
- keystore class: com.ibm.security.cmskeystore.CMSKeyStoreSpi
- provider name: IBMCMSProvider
- keystore name: IBMCMSKS, and CMSKS

In particular the keystore name change will impact on applications that were written for the ikeyman 7 keystore. Changing "CMS" to "CMSKS" should resolve this in the application. For example, `KeyStore ks = KeyStore.getInstance("CMSKS");`

Key Database creation

In iKeyman 7, new key databases are automatically populated with a set of pre-defined CA certificates. In iKeyman 8, this is not automatically done. For more information, see Create a Key Database.

Password Expiry Functionality

Password expiry for CMS key databases is still supported in iKeyman 8 for backward compatibility, but the feature has been deprecated. Using this feature is not recommended, as it may be removed in future versions of iKeyman

No MS-CAPI support

iKeyman 7 provides access to the Microsoft Certificate store via MS-CAPI. This feature is currently not available in iKeyman 8.

PKCS#11 functionality

PKCS#11 access is now exclusively done through the **IBMPKCS11Impl** Provider, as opposed to the IBMPKCS11 Provider and native methods that existed in iKeyman 7. For more information, see PKCS#11

Certificate Validation

In iKeyman 7, certificates are validated before being added to a CMS keystore. The validation includes ensuring that all necessary intermediate and root certificates used to validate the certificate are present and that these have not expired. This validation has been removed from iKeyman 8 to make the various keystore types more consistent.

PKCS#12 Import

When importing a PKCS#12 file, iKeyman 7 generates an error if any of the certificates in the source file are already present in the target keystore. In iKeyman 8 these duplicates are ignored and not imported.

Changed command usage strings

Some command usage strings in iKeyman 7 don't clearly identify which parameters are optional and required. iKeyman 8 has an improved command usage structure that clearly identifies which parameters are supported.

Certificate request signing

Certificate requests in iKeyman 7 may only be signed by a certificate if the resultant certificate's validity period falls entirely within the validity of the signer certificate. This requirement has been removed in iKeyman 8.

Certificate creation

When creating a self-signed certificate in iKeyman 7, a common name (CN) value must always be specified. iKeyman 8 only requires any subject alternative name (SAN) or distinguished name (DN) element to be specified. If no DN is supplied, the SAN extension is marked as critical.

Secondary database for PKCS#11

iKeyman 7 only allows CMS keystores to be used as secondary databases for PKCS#11 operations. iKeyman 8 allows all file-based keystore types to be used.

Improved error messages

iKeyman 7 often generates unhelpful and generic error messages when an error occurs. This area has been improved in iKeyman 8 with more detailed error messages and a full stack trace for every error.

Converting a keystore

iKeyman 7 doesn't include certificate requests when converting from one database type to another. This limitation has been fixed in iKeyman 8.

Multiple OUs

iKeyman 7 doesn't display multiple OUs in the short CLI display or in the details view of the GUI. iKeyman 8 shows all OUs.

Signature Algorithm

The signature algorithm used when creating self-signed certificates or certificate requests is no longer specified in the settings file. It is now selected in the creation dialogs or passed as a command line parameter.

Signing certificate requests

iKeyman 7 only allows certificate requests to be signed by CMS keystores. In iKeyman 8 all file-based keystore types can be used for this operation.

Appendix B. A Simple Example

The example below offers an example scenario for a company setting up a Web site for its employees to access business sensitive information. It is assumed that the web server chosen by the company uses GSKit as its SSL provider. The example does not cover all issues for such a scenario but instead concentrates on what an administrator would typically need to do to set up a CMS keystore in such an environment.

The requirement

The ACME Company wishes to set up a Web site for its employees to access certain sensitive business information across a wide geographical area. Some employees are more senior than others and therefore will be allowed access to more resources on the server than the more junior employees. It is expected that employees can be assured they are connecting to their company Web site and not some fraudulent site pretending to be their company site. Employees use a customized web browser that can read CMS key databases to access certificates contained in them.

The CEO of ACME has asked the system administrator to implement this system in a manner that is secure and cost conscious.

Considerations for the administrator

The administrator makes some decisions based on the requirements:

- As the employees are located at different geographical locations a secure channel for the web traffic must be used. The administrator decides that this should be SSL.
- As employees will have different levels of access to web content the administrator decides that the server will operate in client authentication mode where each connecting client must present a valid certificate in order to gain access. Information from this presented certificate will be used to limit access to authorized areas of the web server only (this is outside the scope of this scenario)
- As cost is an issue the administrator decides that it is too costly to have every employee certificate signed by a CA. The administrator decides to make use of a company wide intermediate certificate that will then be used to sign all employee certificates.
- Employees must be able to validate the server's certificate thereby proving the authenticity of the web server.
- The administrator notes that it is bad practice to use a certificate for more than one purpose so decides that another certificate must be produced and signed by the CA. This certificate will be the server certificate used for the Web site. Using the Intermediate Certificate for this purpose would be poor practice.

Step 1: Obtain a company wide Intermediate Certificate

The administrator needs to create a certificate that can be used to sign each employee's certificate. This intermediate certificate itself may be publicly published so it needs to be signed by a trusted CA. To achieve this, the administrator performs the following actions:

1. The administrator creates a new CMS keystore using the "Create a Key Database" command:

   ```
   ikeycmd -keydb -create -db acme.kdb -pw offs64b -expire 365
   ```

2. The administrator notices that the new keystore, while containing a number of CA certificates, does not contain the certificate of the CA which he would like to use to sign his Intermediate Certificate. He obtains the CA certificate (this is usually done by visiting a well know site that publishes these) and adds it to his CMS keystore via the "Add a Certificate" command:

```
3.  ikeycmd -cert -add -db acme.kdb -pw offs64b -label OurCA
       -file CACert.arm -format ascii
```

4. The administrator then creates a new certificate request to be sent to the CA that he has chosen to sign our Intermediate Certificate using the "Create a Certificate Request" command:

```
5.  ikeycmd -certreq -create -db acme.kdb -pw offs64b  -label
6.      OurIntermediate  -dn "CN=acme.com,O=acme,C=US" -file
        certreq.arm -sig_alg  sha1
```

7. The administrator takes the request file (`certreq.arm` in this case) and sends it to the CA for signing. Sometime later the signed certificate is returned by the CA.

The administrator then receives the certificate into the CMS keystore using the "Receive a Certificate" command:

```
 ikeycmd -cert -receive -file  signedCert.arm -db acme.kdb -pw
offs64b
```

8. Make the new certificate the default one. This means that it will be used by default to sign other certificate request if no other certificate label is given. The administrator makes it the default certificate using the following command:

```
ikeycmd -cert -setdefault -db acme.kdb -pw offs64b -label
OurIntermediate
```

Step 2: Sign all employee certificates using the ACME Intermediate

The administrator now has a CMS keystore containing ACME's intermediate certificate and the CA certificate that signed that intermediate certificate. The administrator now needs to use ACME's intermediate certificate to sign all the employee certificates. To achieve this, the administrator performs the following actions:

1. The administrator asks each employee to obtain the CA certificate and add it to their respective CMS keystores. Note that employees may first need to create their own CMS keystore in the same manner as the administrator did in item 1 of step 1. The employee adds the CA certificate using the "Add a Certificate" command:

```
2. ikeycmd -cert -add -db Dave.kdb -pw Davepwd  -label OurCA
        -file CACert.arm -format  ascii
```

3. The administrator extracts the Intermediate Certificate (note that this does not extract the private key of the certificate) using the "Extract a Certificate" command:

```
4. ikeycmd -cert -extract -db acme.kdb -pw offs64b  -label acmeCert
        -target acmeCert.arm
```

5. The administrator sends the ACME intermediate certificate to each employee asking them to add it to their keystore. Employees do this via the "Add a Certificate" command:

```
6.  ikeycmd -cert -add -db Dave.kdb -pw Davepwd  -label acmeCert
        -file acmeCert.arm -format ascii
```

7. The administrator asks each employee to create a certificate request putting their employee email address in the CN field. The employees use the "Create a Certificate Request" command:

```
8. ikeycmd -certreq -create -db Dave.kdb -pw Davepwd  -label myCert
9.      -dn "CN=dave@acme.com,O=acme,C=US" -file DavesCertReq.arm
        -sig_alg  sha1
```

10. Upon receipt of each employee's certificate request the administrator signs it and returns the signed certificate to the employee. The administrator uses the "Sign a Certificate" command to achieve this:

```
11.   ikeycmd -cert -sign -db acme.kdb -pw  offs64b  -label acmeCert
12.       -target DavesCertReq.arm -expire 365 -file
   DavesSignedCert.arm
        -sig_alg  sha1
```

13. As each employee obtains their signed certificate they receive it into their CMS keystore. Employees use the "Receive a Certificate" command:

```
ikeycmd -cert -receive -file  DavesSignedCert.arm -db Dave.kdb
-pw Davepwd
```

14. Make the new certificate the default one. This means that it will be the certificate sent to the web server when it requests one via SSL for client authentication purposes. The employee makes it the default certificate using the following command:

```
ikeycmd -cert -setdefault -db Dave.kdb -pw Davepwd -label myCert
```

Step 3. Create the Web Server certificate

At this stage the administrator has a CMS database containing the CA certificate and the ACME Intermediate certificate (with its corresponding private key). The administrator puts this CMS keystore in a safe place using it only to sign new employee certificates.

Each employee has a CMS keystore containing the CA certificate, the ACME Intermediate Certificate (minus the corresponding private key), and their own certificate that has been signed by the ACME Intermediate Certificate.

The remaining task for the administrator is to create a CMS keystore with a certificate to be used by the web server. Although the administrator could have used the ACME Intermediate Certificate for this purpose as stated previously it is considered bad practice to use a certificate for more than one purpose. The intermediate certificate is already being used to sign the employees' certificates. To create a keystore and server certificate the administrator performs the following actions:

1. The administrator creates a new CMS keystore using the "Create a Key Database" command:

```
ikeycmd -keydb -create -db acmeWebServer.kdb -pw ejed43dA -expire
365
```

2. The administrator adds the CA certificate to the keystore using the "Add a Certificate command:

```
3. ikeycmd -cert -add -db acmeWebServer.kdb -pw ejed43dA -label
   OurCA -file
```

```
      CACert.arm -format ascii
```

4. The administrator creates a new certificate request to be sent to the CA that he has chosen to sign our web server certificate using the "Create a Certificate Request" command:

```
5. ikeycmd -certreq -create -db acmeWebServer.kdb -pw ejed43dA
   -label
6.      OurServerCert  -dn "CN=web.acme.com,O=acme,C=US" -file
        serverCertReq.arm -sig_alg  sha1
```

7. The administrator takes the request file (`serverCertReq.arm` in this case) and sends it to the CA for signing. Sometime later the signed certificate is returned by the CA. The administrator then receives the certificate into the CMS keystore using the "Receive a Certificate command:

```
8. ikeycmd -cert -receive -file  signedServerCert.arm -db
        acmeWebServer.kdb -pw ejed43dA
```

9. Make the new certificate the default one. This means that when a client connects to the web server the server will offer this certificate to the client. The administrator makes it the default certificate using the following command:

```
10.  ikeycmd -cert -setdefault -db acmeWebServer.kdb -pw ejed43dA
        -label OurServerCert
```

11. The administrator now has a CMS keystore with a server certificate ready for use by the Web server application.

So do we meet the requirements?

Looking at each requirement in turn:

- As the employees are located at different geographical locations a secure channel for the web traffic must be used. The administrator decides that this should be SSL.

  For a web server to make use of SSL it must have a server side certificate to offer to clients during the SSL handshake. The certificate labelled **OurServerCert**" in the keystore `acmeWebServer.kdb` may be used for this purpose.

- As employees will have different levels of access to web content the administrator decides that the server will operate in client authentication mode where each connecting client must present a valid certificate in order to gain access. Information from this presented certificate will be used to limit access to authorized areas of the web server only (this is outside the scope of this scenario)

  Each employee has their own certificate to offer to the server when it requests one during the SSL handshake. The server can first validate the client certificate as it has the signer chain, that is, the client certificate is signed by the ACME

Intermediate Certificate, and the ACME intermediate certificate is in turn signed by the CA certificate. The server keystore (`acmeWebServer.kdb`) has both of these. Once the client certificate has been validated the application can inspect the CN of the certificate and extract the employee email address from it. The application can then use the employee email address to determine the level of access allowed for the connection.

- As cost is an issue the administrator decides that it so too costly to have every employee certificate signed by a CA. The administrator decides to make use of a company wide intermediate certificate that will then be used to sign all employee certificates.

  The administrator only incurred the expense of two CA signing operations. One for the Intermediate Certificate and one for the signer certificate.

- Employees must be able to validate the server's certificate thereby proving the authenticity of the web server.

  When the client application receives (as part of the SSL handshake) the server certificate it can verify the validity of that certificate as it has the CA certificate that signed it.

- The administrator notes that it is bad practice to use a certificate for more than one purpose so decides that another certificate must be produced and signed by the CA. This certificate will be the server certificate used for the Web site. Using the Intermediate Certificate for this purpose would be poor practice.

  Two certificate have been created. **OurServerCert** for use of the ACME web server, and **OurIntermediate** for the administrator to use to sign employee certificates.

Appendix C. A PKCS#11 Example

The following example offers an example scenario for setting up and using a PKCS#11 HSM or smartcard device on a Windows platform. It is the same for Linux except the filenames are platform specific, e.g. use '/' not '\' or '\\' for directory separators, and .so rather than .dll for the pkcs11 library.

The requirement

The user wants to set up an HSM for key/certificate storage and usage with GSKit.

Considerations for the administrator

The administrator makes some decisions based on the requirements:

- Selection of HSM device. For this example we will use the Eracom/Safenet "Orange" (SW emulation).
- Setup or management of the HSM device. Each HSM will come with its own vendors instructions on the correct setup and secure operation of the device.

Step 1: Configure jre's pkcs11 provider

The administrator needs to set up the jre so that the PKCS11 provider is configured for the selected HSM:

1. Edit `jre/lib/security/java.security`. Include this line in the list of providers:
2. `security.provider.n=com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl c:\\p11_eracom.cfg`

   where 'n' is the next available provider number.

3. Create or edit the file `c:\p11_eracom.cfg` to contain the following lines:
4. `library = C:\Program Files\Eracom\ProtectToolkit C SDK\bin\cryptoki.dll`
5. `name = xxx`
6. `tokenlabel = xxx`
7. `attributes(*,CKO_PRIVATE_KEY,*) = {`
8. ` CKA_PRIVATE = true`
9. ` CKA_TOKEN = true`
   `}`

   where 'xxx' is the token label to use.

Note:
The CKO_PRIVATE_KEY attributes are not always necessary on every pkcs11 provider. However some providers do need them and they cause no problem for providers that do not.

Step 2: iKeyman usage

1. In iKeyman, open the pkcs11 device by selecting "PKCS11config" from the keystore type dialog. If only "PKCS11direct" appears, then the configuration of the previous steps has failed or the DLL has not loaded correctly.
2. Keys pr certificates will be listed with as <tok label>:<name> where <tok label> will be the "name" from the configuration file. GSKit will match names using this convention where <tok label> will be the pkcs11 token label. These must match in the configuration file to make the names appear the same in iKeyman 8 as they appear in GSKit.

Appendix D. Keystore provider sample Java code

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.Security;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.util.Enumeration;

public class Ksdump {
    public static void main( String[] args ) {
        if (args.length < 2 || args.length > 3) {
            System.out.println( "Wrong arguments!!\n\nUsage: ksdump
<Keystore Name>
            <Password> <(Optional) Keystore Provider>" );
            System.exit( -1 );
        }

        // First arg is the keystore file name.
        String keyStoreFileName = args[ 0 ];

        // The second is the password
        String keyStorePassword = args[ 1 ];
        String provider = "";

        // The third is the provider, if provided in the command line
        if (args.length == 3) {
            provider = args[ 2 ];
        }

        KeyStore keyStoreRef = null;

        // If provided a provider
        if (!provider.equals( "" )) {
            keyStoreRef =
processWithSpecifiedProvider( keyStoreFileName, keyStorePassword,
            provider );
        } else {
            // Otherwise, try all available providers one bye one
            keyStoreRef = processWithProviderList( keyStoreFileName,
keyStorePassword );
        }

        // Print all certificates
        listAllCertificates( keyStoreFileName, keyStoreRef );
    }
    private static KeyStore processWithProviderList( String
keyStoreFileName,
    String keyStorePassword ) {
        String[] providerNames = getProviderNames();
        String provider = "";
        KeyStore keyStoreRef = null;

        /*
         * Due to the reason that sometimes .p12 keystores have a .kdb
suffix,
         * and in the case that the user cannot / doesn't specify the
exact
         * provider in the command line, so we are looping through each
possible
         * keystore type.
         */
        for (String possibleProvider : providerNames) {
            try {
                keyStoreRef = processKeyStoreFile( keyStoreFileName,
keyStorePassword,
                possibleProvider );
            } catch (Exception e) {
                // Invalid KeyStore Format
                continue;
            }

            // Found a proper provider
            provider = possibleProvider;
            break;
        }

        // If no proper provider has been found, then exit
        if (provider.equals( "" )) {
            System.out.println( "Sorry! We cannot process the keystore
file you
            provided: " + keyStoreFileName
                    + " - no provider found!" );
            System.exit( -10 );
        }
        return keyStoreRef;
    }

    private static KeyStore processWithSpecifiedProvider( String
keyStoreFileName,
    String keyStorePassword,
            String provider ) {
        KeyStore keyStoreRef = null;
        try {
            keyStoreRef = processKeyStoreFile( keyStoreFileName,
keyStorePassword, provider );
        } catch (KeyStoreException e) {
            System.out.println( "Sorry! There is no KeyStoreSpi
implementation found
            for the keystore: "
                    + keyStoreFileName + "! Please reinstall the
provider!" );
            System.exit( -3 );
        } catch (IOException e) {
            // an I/O or format problem with the keystore data or
password issue
            System.out.println( "Sorry! You may have given a wrong
password or a wrong
            provider for the keystore: "
                    + keyStoreFileName + "!" );
            System.exit( -4 );
        } catch (Exception e) {
            System.out.println( "Sorry! We cannot process the keystore:
" + keyStoreFileName
                    + " the file might be damaged!" );
            e.printStackTrace();
            System.exit( -5 );
        }
        return keyStoreRef;
    }

    private static KeyStore processKeyStoreFile( String
keyStoreFileName,
    String keyStorePassword,
            String possibleProvider ) throws KeyStoreException,
IOException,
            NoSuchAlgorithmException, CertificateException {
        // process the keystore with the provider
        KeyStore keyStoreRef = KeyStore.getInstance( possibleProvider );
        InputStream in = null;
        try {
            in = new FileInputStream( keyStoreFileName );
            keyStoreRef.load( in, keyStorePassword.toCharArray() );
        } catch (FileNotFoundException e1) {
            System.out.println( "Sorry! We cannot find the keystore
file you provided:
            " + keyStoreFileName + "!" );
            System.exit( -2 );
        } finally {
            in.close();
        }
        return keyStoreRef;
    }

    private static String[] getProviderNames() {
        String providers[] =
Security.getAlgorithms( "KeyStore" ).toArray( new String[ 0 ] );
        return providers;
    }

    private static void listAllCertificates( String keyStoreFileName,
KeyStore keyStoreRef ) {
        try {
            // Obtain all aliases
            Enumeration<String> allAliases = keyStoreRef.aliases();

            // List all certificates in the keystore
            System.out.println( "All certificates in the keystore " +
keyStoreFileName + ":" );
            int index = 0;
            while (allAliases.hasMoreElements()) {
                String alias = allAliases.nextElement();
                System.out.println( "#" + index++ + " - " + alias );

                // Display information of the certificate
                Certificate cert = keyStoreRef.getCertificate( alias );
                System.out.println( "     Type:      " + cert.getType()
);
                System.out.println( "     Algorithm: " +
cert.getPublicKey().getAlgorithm() );
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Appendix E. Notices

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Trademarks

Index