IBM Global Security Kit
GSKit version 8

GSKCapiCmd Users Guide

Edition 17 March 2015

IBM Global Security Kit
GSKit version 8

GSKCapiCmd Users Guide

Edition 17 March 2015

IBM

Note

Before using this information and the product it supports, read the information in Appendix F, “Notices,” on page 99.

This edition applies to GSKCapiCmd version 8.0.15 and to all subsequent releases and modifications until otherwise indicated in new editions.

Note: The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2219].

Contents

Preface

This guide describes how to use the **GSKCapiCmd** utility to manage keys, certificates, and certificate requests within a key database.

This document assumes that Global Security Kit (GSKit) is installed, configured, and running on your network.

Who should read this book

This manual is intended for network or system security administrators who have installed GSKit and want to use the **GSKCapiCmd** program to modify Certificate Management System (CMS) or PKCS#11 key databases. This manual assumes that the reader is familiar with the GSKit product range and the functionality of the CMS key database.

Before continuing to read this manual, ensure that you have read and understood the following prerequisite readings. This will ensure that you understand the required concepts and terms used throughout the manual:

- Appendix A, “CMS key databases,” on page 85.
- Appendix B, “A Simple Example,” on page 89.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. Standard shortcut and accelerator keys are used by the product and are documented by the operating system. See the documentation provided by your operating system for more information.

IBM and accessibility

See the IBM® Human Ability and Accessibility Center (http://www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

Contacting software support

Before contacting IBM Tivoli® Software Support with a problem, see the IBM Tivoli Software Support portal http://www.ibm.com/software/sysmgmt/products/support/.

If you need additional help, contact software support by using the methods described in the Software Support Handbook http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.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, non-specific command-line options or identifiers, and special words 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`.

Symbol conventions

[ ] - Identifies an option that is optional, if an option is not surrounded by this style of brackets the option is required.

| - Indicates an "OR" relationship between the options on either side of it.

{} – Identifies mutually exclusive set of options.

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.

Chapter 1. Using the GSKCapiCmd program

GSKCapiCmd is a tool that can be used to manage keys, certificates, and certificate requests within a key database. The following chapters go into detail for each of the functions supported by GSKCapiCmd.

GSKCapiCmd uses some encoding rules, and implements aspects of certain RFCs and standards. It is not strictly necessary for users to have a full understanding of these items in order to use this utility. However, if you want to learn more then you can examine the resources contained in: Appendix D, "Resources," on page 95.

Language support overview

IBM Global Security Kit (GSKit) software is built using the International Components For Unicode toolkit (ICU) to provide Internationalization support.

Notwithstanding the lack of translated message catalogs, ICU provides other localization functions that are independent of translation, notably the formatting of date and time strings.

Locale environment variables

For most current operating systems, localized behavior is obtained by specifying the desired locale in the user environment. For **gsk8capicmd** on UNIX like systems, you can set the **LANG** environment variable to the desired locale name as specified by POSIX, X/Open, or other open systems standards.

If you are in a Windows environment, you can modify the language setting in the **Regional Settings** of the Control Panel.

LANG variable on UNIX or Linux systems

Most UNIX or Linux systems use the LANG variable to specify the desired locale. However, different UNIX and Linux operating systems require different locale name values to specify the same language. Always use a value for LANG that is supported by the UNIX or Linux operating system you are using. To obtain the locale names for your UNIX or Linux system, enter the following command: `locale –a`.

If you specify the LANG environment variable and also modify the regional settings then the LANG environment variable will override the regional setting. As specified by open systems standards, other environment variables override LANG for some or all locale categories. These variables include the following:

- `LC_COLLATE`
- `LC_CTYPE`
- `LC_MONETARY`
- `LC_NUMERIC`
- `LC_TIME`
- `LC_MESSAGES`
- `LC_ALL`

If any of the previous variables are set, you must remove their setting for the LANG variable to have full effect.

Forcing output to a different locale

You can use the **-locale** command-line option to select the desired display language if:

- The operating system does not support the LANG environment variable, or
- You want to display messages in a different locale to the current environment settings.

To do this, set the locale option to the appropriate canonical name, based on the ISO language or territory codes.

For example, to display the help message in German issue the command:

```
gsk8capicmd —help —locale de
```

Some example ISO language codes are:

Table 1. ISO Language Codes

ISO Language Code	**Language**
de	German
en	English
es	Spanish
fr	French
it	Italian
ja	Japanese
ko	Korean
pt_BR	Portuguese (Brazil)
zh_CN	Simplified Chinese
zh_TW	Traditional Chinese

Using locale variants

Although **gsk8capicmd** currently provides only one translated version for each language, you can specify a preferred locale variant to find the corresponding language translation if it is available.

If a message catalog is not found for the desired language, the English message catalogs are used. For example, suppose you specify the AIX® locale for German in Switzerland as follows:

```
LANG=De_CH.IBM-850
```

In this example, the catalogs are searched in the following order to locate the specified locale:

1. de_CH
2. de
3. en

Since **gsk8capicmd** does not provide a German in Switzerland language pack, **de_CH** is not found. If the German language package is available, **de** is used. Otherwise, the default locale **en** is used, causing text to be displayed in English.

Text encoding (code set) support

Different operating systems encode text in different ways. For example, Windows systems use SJIS (code page 932) for Japanese text, but UNIX or Linux systems often use eucJP.

In addition, you can provide multiple locales for the same language so that different code sets are used for the same language on the same machine. Message catalogs are encoded using UTF-8, and the text is converted to the locale encoding before being presented to the user. In this way, the same French message catalog files can be used to support a variety of Latin 1 code sets, such as ISO8859-1, Microsoft 1252, IBM PC 850, and IBM MVS™ 1047.

Interoperability across your domain depends on code set files, which are used to perform UTF-8 conversion and other types of encoding-specific text processing.

For messages to display correctly on some platforms, you might need to specify the correct code set that supports your locale.

Warning: KeyStore Backups

KeyStores often contain keying material (private keys) related to commercially sourced (paid for) certificates. If the keying material is lost, the certificate must be replaced. If the keying material is not backed up and a KeyStore management operation fails or otherwise results in the loss of the KeyStore or key material, then this may result in additional cost and time to replace the certificate/key. It is a good practice to keep regular backups of KeyStores to avoid the risk of loss of keying material.

Note: Backed-up KeyStores need the same physical protection as online KeyStores. Passwords of backed-up KeyStores must be managed with the same care as online KeyStores.

KeyStore Overview

KeyStores are databases used to store Private Keys and Public Keys contained in X.509 Certificates. The database may exist as a file or Hardware storage device e.g. Smart Card. The database may also be used to store certificate requests, secret keys, and key generation parameters. The KeyStore used maybe one of a number of supported formats. The following table lists the format and relevant capabilities:

	CMS[1]	**P12**[2]	**P11**[3]	**MSCAPI**[4]
Private Key Storage	Yes	Yes	Yes	Yes
Trust Anchor Source	Yes	Yes	Yes	Yes
Default Attribute[5]	Yes	No	No	No
Trusted Attribute[6]	Yes	No	No	No
Support Secondary CMS/P12 KeyStore for Trust Anchors	No	No	Yes[7]	No
Stash File Support	Yes	Yes	No	No
Password[8] Expiry[9]	Yes	No	No	No

	CMS[1]	P12[2]	P11[3]	MSCAPI[4]
GSKCapiCmd Supported	Yes	Yes	Yes	No[10]
Vendor Tools Supported	N/A	N/A	No[11]	Yes

Footnotes:

- [1] IBM Proprietary Format
- [2] PKCS#12 Format transparently supported and interchanged with CMS format by GSKit
- [3] PKCS#11 Format
- [4] Microsoft Key Store
- [5] Deprecated
- [6] Deprecated
- [7] Secondary KeyStore must be CMS or P12
- [8] Password does not apply/exist when P12 KeyStore is empty
- [9] Deprecated
- [10] Must use Vendor Tools
- [11] Must Not use Vendor Tools

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
MAIL	rfc822Mailbox	0.9.2342.19200300.100.1.3
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	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.

GSKCapiCmd command-line syntax

The syntax for the **GSKCapiCmd** program is as follows:

```
gsk8capicmd <modifiers> <object> <action> <options>
```

where:

modifiers
May include the following:

-fips [<true>|<false>]

Enable or disable forced Federal Information Processing Standards (FIPS) mode.

In FIPS mode, **gsk8capicmd** initializes the underlaying cryptographic provider in FIPS mode so that it only uses algorithms that have been FIPS 140-2 validated.

The program runs in FIPS mode by default. If however the -fips true is set and the provider cannot be initialized in FIPS mode then the **gsk8capicmd** operation will fail. If FIPS mode is not forced and the provider cannot be initialized in FIPS mode then the utility will fall back to a non-fips mode of operation.

-locale <language>
Set the display language preference.

-trace <pathname>
Enable trace logging to the named file.

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.

-random
Generates a random string of characters that can be used as a password for other commands.

-version
Displays version information for GSKCapiCmd.

-help Displays help for the GSKCapiCmd commands.

action Is the specific action to be taken on the object.

options Are the options associated with the specified object and task.

The following chapters of this manual describe each particular object, its associated actions, and what options are available.

Chapter 2. Key database commands

The key database commands are associated with the **-keydb** object. This object supports the following actions:

- "Create a key database (-create)" on page 8.
- "Delete a key database (-delete)" on page 10.
- "Change the password of an existing key database (-changepw)" on page 11.
- "Stash the password of an existing key database (-stashpw)" on page 12.
- "List the supported key databases (-list)" on page 13.
- "Convert a key database (-convert)" on page 13.
- "Display the expiry date associated with a key database (-expiry) [deprecated]" on page 15.

  Note: This feature is **deprecated** as CMS keystores no longer have this capability. Keystore password lifetime management must be done outside of the **gskcapicmd** utility.

Each of the following sections details the key database commands and the options available for each command.

After creating a key database (or keystore), it is the user's responsibility to maintain the contents of the keystore in order to maintain security of any application using that keystore. The trusted default CA certificates are particularly important as they are the trust anchors for all user certificates. The presence of a CA certificate in the keystore is enough to make it, and all valid certificates that it signs, trusted by the application using the keystore.

Specifically, the user must:

1. Monitor certificates for expiry and remove any expired certificates. The presence of expired CA certificates does not compromise security as they will fail validation if used.
2. Review the default CA certificates and remove any unnecessary ones. If a CA certificate does not need to be trusted by the application using the keystore then it should be removed.

Keystore access control

The keystore content is protected by encryption and MAC by using keys derived from a password that is chosen for each keystore. The keystore owner is responsible for this password. The password must be managed with respect to its purpose of protecting the keystore contents from unauthorized use or disclosure.

The access control settings of the keystore file default to those settings of the user running the create operation. The user can change the access control settings to provide additional protection to the default encryption by using the appropriate operating system services. That is, services such as "caclc" on Windows before Windows Vista and "icacls" for Windows Vista+ or "chmod" on UNIX operating systems.

You can stash the keystore password in a file that can automatically provide the password when required. Stashing the password removes the need for an administrator to manually enter the password to open a keystore. When accessing a key database, the system first checks for the existence of a stash file. If one exists, the contents of the file is decrypted and used as input for the password.

If you elect to create a stash file when creating a key database, the password is stashed into a file named as follows: `<`*`key_database_name`*`>.sth`. The access control settings for the stash file are set so that the stash file is readable only by the owner of the file. It is not recommended that multiple users be given access to the stash file. However, it is possible to change the access control settings on the stash file by using the appropriate operating system services. That is, services such as "caclcs" on Windows before Windows Vista and "icacls" for Windows Vista+ or "chmod" on UNIX operating systems.

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

Create a key database (-create)

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

The **create** command creates a new CMS or PKCS#12 key database.

The syntax for creating a key database with **GSKCapiCmd** is as follows:

```
gsk8capicmd -keydb -create -db <name> [-pw <passwd>]
[-type <cms|kdb|pkcs12|p12>] [-expire <days>] [-stash]
[-strong] [-empty | -populate] [-f]
```

where:

object -keydb

action -create

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
Fully qualified path name of a key database. A good example of a key database file name might be `/home/<`*`user_name`*`>/keydb.db`.

-pw <passwd>
The password for the key database identified by the –db tag. If you want to create a keystore without a password simply leave the -pw tag out of the command.

-type <cms | kdb | pkcs12 | p12>
The type of the key database to be created. This tool only supports

the creation of a CMS or PKCS#12 format key database. If this tag is omitted then the tool creates a CMS key database by default. The value '**kdb**' can be used as a synonym for '**cms**' and '**p12**' for '**pkcs12**'.

A CMS key database consists of three files:

- The first file is the certificate key database itself. By convention, the name of this file should include the .kdb extension (for example, `key.kdb`). This extension is not required, but it is a good idea as it makes it easy to identify the file as a key database.
- The second file created is used to store certificate requests associated with the key database. This file is created with the same name as given to the key database, but with a .rdb extension.
- The third file is used to hold the certificate revocation list used by the key database. This file has become obsolete and is no longer used. This file is created with the same name as the key database, but with a .crl extension.

A PKCS#12 keystore is a single file, which by convention is created with the .p12 extension.

-expire <days>
(Deprecated). The number of days before the password for the key database is to expire. If this tag is not used then the key database password will never expire. If specified, the duration must be from 1 to 7300 days (20 years).

Note: This parameter is ignored for **PKCS#12** keystores.

-stash Stash the password for the key database after creation. See "Keystore access control" on page 7 for further details.

-strong
Check that the password entered satisfies the following minimum requirements for password strength:

- The minimum password length is 14 characters.
- A password must have at least one lower case character, one uppercase character, and one digit or special character (for example, *$#% etc.). A space is classified as a special character.
- Each character must not occur more than three times in a password.
- No more than two consecutive characters of the password can be identical.
- All characters are in the standard ASCII printable character set within the range from 0x20 to 0x7E inclusive.

-empty | -populate
The **–empty** option has no action and is **deprecated**. A keystore is empty when created.

The keystore can optionally be populated with a number of predefined trusted certificate authority (CA) certificates. To load the default CA certificates the **–populate** option must be given. Alternatively the options "-cert -populate" may be used to add trusted CA certificates to an existing keystore.

The CA certificates loaded should be reviewed by inspecting the output of the list certificates command. Any or all of these CA certificates can be removed from the key database. If you want to remove any of the certificates, use the delete certificate command in this manual.

-f The **gsk8capicmd** utility will not normally let you overwrite an existing database. Use this option to force the removal of an existing keystore before creating a new one of the same name.

Delete a key database (-delete)

Note: Refer to the "Warning: KeyStore Backups" on page 3 section before updating a keystore.

The **delete key database** command simply deletes the identified key database. To identify the key database, simply specify the correct file name of the key database. The request database (.rdb) and certificate revocation list (.crl) files are removed automatically during the process. If a stash file was created, it is not removed.

If a password was provided for this command, it 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.

The syntax for deleting a key database with GSKCapiCmd is as follows:

```
gsk8capicmd -keydb -delete -db <name> [-pw <passwd>]
```

where:

object -keydb

action -delete

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database that has been identified by the –db tag. The –pw tag is required if the key database was created with a password. It is an additional check to ensure that the user deleting the key database is authorized to do so. If the key database does not have a password, the –pw tag is not required.

If a password is provided and it does not match the password for the identified key database, the key database is not deleted.

Change the password of an existing key database (-changepw)

Note: Refer to the "Warning: KeyStore Backups" on page 3 section before updating a keystore.

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 encryption key that will be used during the encryption process.

Note: This command has no effect on an empty PKCS#12 type keystore.

The syntax for changing the password of an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -keydb -changepw {-db <name>|-crypto <module_name> -tokenlabel
<token_label>} [-type <cms|kdb|pkcs12|p12>] [-pw <passwd>] -new_pw
<new_passwd> [-expire <days>] [-stash] [-strong]
```

where:

object -keydb

action -changepw

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db tag.

-new_pw <new_passwd>
The new password for the key database.

-expire <days>
(Deprecated.) The number of days before the new password is to expire. If this tag is not specified the key databases password never expires. If specified the duration must be within the range of 1 to 7300 days (20 years).

Note: This parameter is ignored for PKCS#12 type keystores.

-stash
Stash the password for the key database. See "Keystore access control" on page 7 for further details.

-strong
Check that the password entered satisfies the following minimum requirements for password strength:

- The minimum password length is 14 characters.
- A password must have at least one lower case character, one uppercase character, and one digit or special character (for example, *$#% etc.). A space is classified as a special character.
- Each character must not occur more than three times in a password.
- No more than two consecutive characters of the password can be identical.
- All characters are in the standard ASCII printable character set within the range from 0x20 to 0x7E inclusive.

Stash the password of an existing key database (-stashpw)

The **stash password** command takes an existing key databases password and stashes it to a specified file. Stashing the password for a key database allows the password to be recovered from the file when automatic login is required. See "Keystore access control" on page 7 for further details regarding the stash file.

The syntax for stashing the password of an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -keydb -stashpw -db <name> [-pw <passwd>]
```

where:

object -keydb

action -stashpw

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database identified by the –db tag.

List the supported key databases (-list)

The **list supported key databases** command performs one of the following functions depending on the options that you specify:

- Lists all of the key database types that the GSKCapiCmd supports. For example, CMS and PKCS#11.
- Lists the token labels associated with a specified PKCS#11 cryptographic driver.
- Verifies that a specified keystore is in a usable format.

The syntax for listing the key databases supported by GSKCapiCmd is as follows:

```
gsk8capicmd -keydb –list [-crypto <driver_name> | -db <name>
[-pw <password>] [-stashed]]
```

where:

object -keydb

action -list

options

-crypto <driver_name>
Lists the token labels for the named PKCS#11 cryptographic device.

-db <name>
Keystore name for format validation. The program checks that the named keystore is usable.

-pw <passwd>
The password for the key database identified by the -db tag.

-stashed
The password for the key database is recovered from the stash file.

Convert a key database (-convert)

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

The **convert key database** command converts an old version CMS key database to the new version of CMS key database. The latest version of CMS is more secure because it uses more secure algorithms to protect the contents of the key databases during creation.

This command requires that you assign a name to the new key database that is different to the existing old key database. That is, the name cannot be the same as the existing one. This requirement is to ensure that the old key database is not destroyed until the user destroys it. Once all testing of the new version key database has been completed, the user can remove the old key database and rename the new key database to the old key databases name (if required).

The syntax for converting a key database to the latest CMS version by GSKCapiCmd is as follows:

```
gsk8capicmd -keydb –convert –db <name> [-pw <passwd>] [{-type|-old_format}
<cms|kdb|pkcs12|p12>] [{–new_db|-target} <name>][-new_pw <passwd>]
[-new_format <cms|kdb|pkcs12|p12>] [-preserve|-populate] [-expire <days>]
[-strong] [-stash]
```

where:

object -keydb

action -convert

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type | -old_format <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file extension of the database path name to determine the keystore type.

-pw <passwd>
The password for the key database identified by the –db tag.

-new_db | -target <filename>
Fully qualified path name of a new key database to be created during the conversion.

-new_pw <passwd>
The password for the key database identified by the –new_db tag.

-new_format <cms | kdb | pkcs12 | p12>
The type of the new keystore. If this option is not specified, the program uses the file name suffix of the new database path name to determine the keystore type.

-preserve | -populate
The preserve option has no action and is **deprecated**. The newly created key database will include the same certificates as the old key database, unless the populate option is selected.

The populate option adds a number of predefined trusted certificate authority (CA) certificates to the newly created key database. The CA certificates loaded should be reviewed by inspecting the output of the list certificates command. Any or all of the added CA certificates can be removed from the key database. If you want to remove any of the certificates, use the delete certificate command in this manual.

-expire <days>
(Deprecated.) The number of days before the password is to expire. If this tag is not specified the key databases password never expires. If specified the duration must be within the range of 1 to 7300 days (20 years).

Note: This parameter is ignored when converting to PKCS#12 type keystores.

-strong
Check that the password entered satisfies the following minimum requirements for the password strength:

- The minimum password length is 14 characters.
- A password must have at least one lower case character, one uppercase character, and one digit or special character (for example, *$#% etc.). A space is classified as a special character.
- Each character must not occur more than three times in a password.
- No more than two consecutive characters of the password can be identical.
- All characters are in the standard ASCII printable character set within the range from 0x20 to 0x7E inclusive.

-stash Stash the password for the new key database. See "Keystore access control" on page 7 for further details regarding the stash file.

Display the expiry date associated with a key database (-expiry) [deprecated]

This command is **deprecated**. The **expiry key database** command simply displays the date that the password associated with the identified key database will expire. When identifying the key database you need to specify the file name of the key database.

The syntax for displaying the expiry of the password associated with a key database with GSKCapiCmd is as follows:

```
gsk8capicmd -keydb -expiry -db <name> -type <cms | kdb| pkcs12 | p12>
[-pw <passwd>]
```

where:

object -keydb

action -expiry (**Deprecated.**)

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database identified by the –db tag. The

–pw tag is required if the key database was created with a password. If the key database does not have a password the –pw tag is not required.

-type <cms | kdb| pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file name suffix of the database path name to determine the keystore type.

IMPORTANT: An expiry of 0 means that the password associated with the key database does not expire.

Chapter 3. Certificate commands

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

The certificate commands are associated with the -cert object. This object supports the following actions:

- “Create a self-signed certificate in a keystore (-create)” on page 19
- “Add a certificate to a keystore (-add)” on page 22
- “Delete a certificate from a keystore (-delete)” on page 24
- “Display details of a certificate (-details)” on page 25
- “Export a certificate (-export)” on page 26
- “Receive a certificate into a keystore (-receive)” on page 28
- “Import a certificate (-import)” on page 29
- “Extract a certificate from a keystore (-extract)” on page 31
- “List details of the default certificate (-getdefault) [deprecated]” on page 32
- “Set default certificate in a keystore (-setdefault) [deprecated]” on page 33
- “Rename a certificate in a keystore (-rename)” on page 34
- “List the certificates stored in a keystore (-list)” on page 36
- “Modify a certificate in a keystore (-modify)” on page 38
- “Sign a certificate (-sign)” on page 39
- “Validate a certificate (-validate)” on page 41

The following sections describe how to use each of the identified certificate actions and what options are available. Supporting information to assist with generating and manipulating certificates is also included:

Signature algorithms

The following signature algorithms are supported for use with commands that accept the **-sigalg** parameter:

```
md5 | MD5_WITH_RSA | MD5WithRSA | sha1 | SHA_WITH_RSA | SHAWithRSA | SHA1WithRSA |
sha224 | SHA224_WITH_RSA | SHA224WithRSA | sha256 | SHA256_WITH_RSA |
SHA256WithRSA | SHA2WithRSA (deprecated) | sha384 | SHA384_WITH_RSA | SHA384WithRSA |
SHA3WithRSA (deprecated) | sha512 | SHA512_WITH_RSA | SHA512WithRSA |
SHA5WithRSA (deprecated) | SHA_WITH_DSA | SHA1WithDSA | SHAWithDSA |
SHA1WithECDSA | EC_ecdsa_with_SHA1 | SHA224WithECDSA | EC_ecdsa_with_SHA224 |
SHA256WithECDSA | EC_ecdsa_with_SHA256 | SHA384WithECDSA | EC_ecdsa_with_SHA384 |
SHA512WithECDSA | EC_ecdsa_with_SHA512
```

Note that the deprecated algorithms SHA2WithRSA, SHA3WithRSA and SHA5WithRSA were just aliases for SHA256WithRSA, SHA384WithRSA and SHA512WithRSA respectively, which have not been deprecated. SHA3 is a new hash algorithm and the name clash caused confusion.

The following table shows the key types matched with their corresponding signature algorithms.

Table 2. Signature algorithms

Algorithm	Signing algorithm
RSA	md5 | MD5_WITH_RSA | MD5WithRSA | sha1 | SHA_WITH_RSA | SHAWithRSA | SHA1WithRSA | sha224 | SHA224_WITH_RSA | SHA224WithRSA | sha256 | SHA256_WITH_RSA | SHA256WithRSA | sha384 | SHA384_WITH_RSA | SHA384WithRSA | sha512 | SHA512_WITH_RSA | SHA512WithRSA
EC	EC_ecdsa_with_SHA1 | SHA224WithECDSA | EC_ecdsa_with_SHA224 | SHA256WithECDSA | EC_ecdsa_with_SHA256 | SHA384WithECDSA | EC_ecdsa_with_SHA384 | SHA512WithECDSA | EC_ecdsa_with_SHA512
DSA	SHA_WITH_DSA | SHA1WithDSA | SHAWithDSA
DH	DH (key/cert gen only)

Information about key sizes

The following table indicates the key sizes that are available for each of the supported algorithms. If a key size is not specified in the API calls to generate keys then a default key size is used.

Table 3. Key sizes

Algorithm	Size (bits)	Default value (bits)
RSA	512-4096; key sizes in this range should be selected as per NIST SP800-131; 8192 is supported for validation only. **Note:** Available key sizes might vary according to security configurations. For example, you cannot generate 512 bit RSA keys in FIPS mode.	1024
EC	256, 384 and- 512 **Note:** GSKit EC key generation only supports P256, P384, and P521 curves. P521 curve keys use a 512 bit SHA2 hash.	256 (SHA256); 384 (SHA384); 512 (SHA512)
DSA	512-4096; key sizes in this range should be selected as per NIST SP800-131; 8192 is supported for validation only.	1024
DH	512-4096; key sizes in this range should be selected as per NIST SP800-131; 8192 is supported for validation only.	1024

Information about elliptic curves

GSKit uses the elliptic curves P-256, P-384, and P-521 as defined by FIPS186-3 for EC signature algorithms. For clarity the RFC4492-equivalent names are also shown:

Table 4. Elliptic curves

EC Key Size (bits)	FIPS186-3 curve name	RFC4492 curve name
256	P-256	secp256r1
384	P-384	secp384r1
512	P-521	secp521r1

Suite B algorithm and key size selection

GSKit *might* optionally operate in Suite B mode. Suite B Mode is an NSA/NIST mode of operation that has specific key length, hash size, and cipher suite requirements as specified by RFC 5430. Suite B is a subset of FIPS-Approved mode.

To operate in Suite B Mode, certificates must be created with specific key and algorithm choices as specified in RFC 5430. Refer to 4.1 and 4.2 of RFC 5430 for specific details on the available choices. The following table provides a mapping of the security levels to signing algorithms:

Table 5. Suite B algorithm security levels

Suite B security level	Signing algorithm
128 bit	EC_ecdsa_with_SHA256 | SHA256WithECDSA
192 bit	EC_ecdsa_with_SHA384 | SHA384WithECDSA

Create a self-signed certificate in a keystore (-create)

A self-signed certificate provides a certificate that can be used for testing while waiting for the 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.

The syntax for creating a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -create {-db <name> [-type <cms | kdb| pkcs12 | p12>] |
-crypto <module_name> -tokenlabel <token_label> [-secondarydb <name>]
[-secondarydbpw <passwd>] [-secondarydbtype <cms | kdb| pkcs12 | p12>]}
[-pw <passwd> | -stashed] -label <label> -dn <dist_name> [-size <key_size>]
[-x509version <1 | 2 | 3>] [-default_cert <yes | no>] [-expire <days>]
[-ca <true | false>] [{-sigalg | -sig_alg} <algorithm_name>] [-ca_label
<label>] [-san_dnsname <name>] [-san_emailaddr <address>] [-san_ipaddr
<address>] [-certpolicy <policy>] [-ku <name>] [-eku <name>]
```

where:

object -cert

action -create

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>

Fully qualified path name of a key database to store the self-signed certificate.

-type <cms | kdb| pkcs12 | p12>

Type of the keystore.

-crypto <module_name>

Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>

The PKCS#11 cryptographic device token label.

-pw <passwd>

The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed

The password for the key database will be recovered from the stash file.

-label <label>

Label attached to the certificate. The label is used to uniquely identify the certificate by a human user.

-dn <dist_name>

The X.500 distinguished name that uniquely identifies the certificate. The input must be a quoted string of the following format (only **CN** is required):

```
CN=common name
O=organization
OU=organization unit
L=location
ST=state, province
C=country
DC=domain component
EMAIL=email address
```

For Example: “`CN=weblinux.Raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,L=RTP,ST=NC,C=US`”

Multiple OU values are now supported. Simply add additional OU key\value pairs to the specified distinguished name. If the OU value requires a comma (‘,’) then you must escape it with ‘\\’

For Example: "`CN=weblinux.Raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,OU=GSKit\\, Gold Coast,L=RTP,ST=NC,C=US`"

-size <key_size>
The size of the new key pair to be created. This size ranges in value depending on the key type. Consult Table 3 on page 18 for valid values.

Note: For some algorithms, you can specify a zero (0) value to use the default key size. This is typically the minimum size that is considered secure.

-x509version <1 | 2 | 3>
The version of X.509 certificate to create, default is 3.

-default_cert <yes | no>
(**Deprecated.**). Sets the newly created certificate as the default certificate for the key database. By default the newly created self-signed certificate is not set as the default (`no`). A default certificate in the key database is used when a specific certificate is not specified for an operation.

-expire <days>
Expiration time of the certificate in days, default 365 days. The duration is 1 to 7300 days (20 years).

Note: To avoid possible timezone issues, the actual **valid-from** time for the certificate will be set one day in the past.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
The type of the secondary key database.

-ca <true | false>
This tag adds the Basic Constraint extension to the self-signed certificate. The Basic Constraint extension value is set to true or false depending on what value is associated with the tag.

-san_dnsname <name>
The Subject Alternative Name (SAN) DNS name(s) for the entry being created.

-san_emailaddr <address>
The Subject Alternative Name (SAN) email address(es) for the entry being created. Multiple values can be separated with commas.

-san_ipaddr <address>
The Subject Alternative Name (SAN) IP address(es) for the entry being created.

-certpolicy <policy>
The certificate policy. A named set of rules limiting the applicability of the certificate. Multiple values can be separated with commas.

-ku <list>
Key usage property list. Specifies the valid uses for the private key or certificate.

-eku <list>
Extended key usage property list. Specifies the valid uses for the certificate.

-sigalg | -sig_alg <signature_algorithm>
The signing algorithm used during the creation of the self-signed certificate. This algorithm is used to create the signature associated with the new self-signed certificate. The generated key type is chosen to match this signing algorithm. See “Signature algorithms” on page 17 for the allowed values.

-ca_label <label>
The label of the CA key to use to sign the certificate.

Add a certificate to a keystore (-add)

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 as either Base64 (ASCII) or binary. It is important to identify the correct format of the file otherwise the operation will fail.

The syntax for adding a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -add {-db <name> -type <cms | kdb| pkcs12 | p12> |
-crypto <module_name> -tokenlabel <token_label>} [-pw <passwd> |
-stashed] -label <label> -file <name> [-format <ascii | binary>]
[-trust <enable | disable>] [-secondaryDB <filename> -secondaryDBpw
<password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -add

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12>
Type of the key database. If this option is not present, the type is implied by the file extension of database path name.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate.

-file <name>
File name of the certificate to add. If the extension is ".p7", ".smime" or ".eml" then it is assumed to be a PKCS#7 encoding. The first certificate will take the 'label' given and all other certificates that are present, will be labelled with their subject name. PEM (extension ".pem") single or multi certificate encodings may also be added.

-format <ascii | binary>
Format of a certificate The default is Base64 encoded ASCII. Additional information about Base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the DER encoded certificate structure. For additional information, see ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002.

-trust <enable | disable>
(Deprecated). Trust status of a CA certificate, where the default is 'enable'. When a CAs certificate trust status is enabled then that CA certificate is permitted to be involved in a certificate chain validation. If the CAs certificate 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 then the validation of "ABC" will fail.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Delete a certificate from a keystore (-delete)

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

The **delete certificate** command 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.

The syntax for deleting a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -delete {-db <name>  -type <cms | kdb| pkcs12 | p12> |
-crypto <module_name> -tokenlabel <token_label>} [-pw <passwd> | -stashed]
-label <label> [-secondaryDB <filename> -secondaryDBpw <password>
-secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -delete

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:
`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | p12 | pkcs12>
Type of the key database. If this option is not present, the type is implied by the database file extension.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate that is to be deleted.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Display details of a certificate (-details)

The **display certificate details** command displays the different details associated with the identified certificate. The details displayed include:

- 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 certificates validity period.
- The fingerprint of the certificate.
- The signature of the algorithm used during creation of the certificate.
- An indication of the certificates trust status.

If more details for the certificate are required, use the **–showOID** option. This option displays a more detailed listing of the certificate details.

The syntax for displaying the details for a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -details [-showOID] [-file <name>] {-db <name> -type
<cms | kdb| pkcs12 | p12> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] -label <label>
[-secondaryDB <filename> -secondaryDBpw <password>
-secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -details

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-showOID
Display a more in-depth listing of the certificate.

-file <name>
File name of the certificate to display. ASCII or binary encoded certificate.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12 | pkcs7>
Type of the key database. If this option is not present, the type is implied by the file extension of the database path name.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate that is to be displayed.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Export a certificate (-export)

The **export certificate** command exports a single certificate and its private key (if one exists) from one key database to another key database. Use the label to identify the certificate that you want to export.

During this process no key generation occurs. On successful completion, the identified certificate will be in both the source and destination key databases.

The syntax to export a certificate from an existing key database to another key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -export -db <name> [-pw <passwd> | -stashed] -label <label>
[-type <cms | kdb| pkcs12 | p12 | pkcs7 | pkcs11> ] -target <name>
[-target_pw <passwd>] [-target_type <cms | kdb| pkcs12 | p12>]
[-encryption <strong | weak>]
```

where:

object -cert

action -export

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>

The fully qualified file name of the key database that contains the certificate to export. If the supplied file name has an extension of either ".p12" or ".pfx" then it is assumed that it is in PKCS#12 format. If it is “.p7”, “.smime” or “.eml” then it is assumed to be a PKCS#7 encoding.

-pw <passwd>

The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed

The password for the key database will be recovered from the stash file.

-label <label>

Label attached to the certificate that is to be exported.

-type <cms |kdb | pkcs12 | p12>

The type of the key database that contains the certificate to export. The default is **cms**.

-target <name>

Destination key database or file where the certificate is to be exported. If the supplied file name has an extension of either “.p12” or .pfx” then it is assumed that it is in PKCS#12 format. If the target keystore does not exist, it will be created.

-target_pw <passwd>

The password of the destination key database or file.

-target_type <cms | kdb| pkcs12 | p12>
The type of the destination key database or file where the certificate is to be exported. The default is **cms**.

-encryption <strong | weak>
The strength of encryption used during the export. The default is **strong**. This tag is no longer used as the export restrictions in the USA have eased. This tag is simply added to this command-line tool for backward compatibility reasons. It has no effect on the operation. **Strong** is always used.

Receive a certificate into a keystore (-receive)

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 either binary or Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the DER encoded certificate structure. For additional information, see ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002. During the receive process, 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: "Re-create certificate requests (-recreate)" on page 53.

The syntax for receiving a certificate to an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -receive -file <name>  [-format <ascii | binary>] { -db <name>
-type <cms | kdb| pkcs12 | p12> | -crypto <module_name> -tokenlabel <token_label>}
[-pw <passwd> | -stashed] [-default_cert <yes | no>] [-t61]
```

where:

object -cert

action -receive

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-file <name>
The file name of the certificate that is to be received. This file can be either binary or base64 encoded.

-format <ascii | binary>
Format of a certificate. The default is Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of

the DER encoded certificate structure. For additional information, see ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name extension to determine the keystore type.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-default_cert <yes|no>
(Deprecated). Sets the newly created certificate as the default certificate for the key database. By default the newly created self-signed certificate is not set as the default (`no`). A default certificate in a key database is used during operations where a specific certificate is not specified.

-t61 Substitute ISO8859-1 character set encodings for malformed TELETEX strings.

Import a certificate (-import)

The **import certificate** command imports certificates from either one key database (CMS or PKCS#12) to another key database (CMS, PKCS#12 or PKCS#11). During this process no key generation occurs. On successful completion, the identified certificates will be in both the source and destination key databases.

The syntax for importing a certificate from an existing key database to another key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -import { -db <name> | -file <name> } [-pw <passwd> | -stashed]
[-label <label>] [-type <cms | kdb| pkcs7 | pkcs12 | p12>]  [-pfx ]
{ -target <name> | -crypto <module_name> -tokenlabel <token_label>} [-secondaryDB
<filename> -secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
[-target_pw <passwd>] [-target_type <cms | kdb| pkcs11 | pkcs12 | p12>]
[-new_label <label>] [-t61]
```

where:

object -cert

action -import

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>

The fully qualified path name of the source key database that contains the certificate to be imported. If the supplied file name has an extension of either “.p12” or .pfx” then it is assumed that it is in PKCS#12 format. If it is “.p7”, “.smime” or “.eml” then it is assumed to be a PKCS#7 encoding.

-file <filename>

The fully qualified path name of a PKCS#12, PKCS#7 or PFX format file of:

- The certificate to be imported, or
- An import script (.txt file).

-pw <passwd>

The password for the key database or PKCS#11 cryptographic device identified by either the -db or -crypto tags respectively. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed

The password for the key database will be recovered from the stash file.

-label <label>

Label attached to the certificate that is to be imported. If the label tag is missing from the command line then the operation will transfer all certificates from the source key database to the target key database. If a certificate in the source key database already exists in the target key database then that certificate is not imported.

-type <cms | kdb| pkcs12 | p12 | pkcs7>

The type of the source key database. The default is **cms**.

-pfx

A switch indicating whether the import file is a .pfx file. Use of this option is unnecessary if the file extension of the file name is .pfx.

-target <name>

Destination key database to which the certificate is to be imported. If the supplied file name has an extension of either “.p12” or .pfx” then it is assumed that it is in PKCS#12 format.

-crypto <module_name>

Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

-target_pw <passwd>
The password of the destination key database.

-target_type <cms | kdb| pkcs11 | pkcs12 | p12>
The type of the destination key database. The default is **cms**.

-new_label <label>
The label to be used in the destination key database to identify the imported certificate.

-t61 Substitute ISO8859-1 character set encodings for malformed TELETEX strings.

Extract a certificate from a keystore (-extract)

The **extract certificate** command simply extracts the certificate data from the key database and places it into the identified file. If the file does not exist then it will be created. If the file already exists, an error indicating this will be returned. The data will be saved as either base64 encoding or binary. No private key components are extracted.

The syntax to extract a certificate from an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -extract {-db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] -label <label> -param <dsa|dh>
-target <name> [-format
<ascii | binary>] [-secondaryDB <filename> -secondaryDBpw <password>
-secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -extract

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create`

`–db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
: The fully qualified path name of a key database.

-crypto <module_name>
: Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

tokenlabel <token_label>
: The PKCS#11 cryptographic device token label.

-pw <passwd>
: The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
: The password for the key database will be recovered from the stash file.

-label <label>
: Label attached to the certificate that is to be extracted.

-param <dsa|dh>
: Extract Diffie Hellman or DSA parameters.

-target <name>
: Destination file to which the certificate or parameters is to be extracted.

-format <ascii | binary>
: Format of an output file. The default is Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the DER encoded certificate structure. For additional information, see ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002

-secondaryDB <filename>
: A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
: Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
: Keystore type of the secondary key database.

List details of the default certificate (-getdefault) [deprecated]

This feature is **deprecated**. Use the explicit label of the desired certificate/key instead.

The **list default certificate details** command lists the following details for the default certificate of the identified key database:

- The label of the default certificate.
- The size of the key associated with the default certificate.
- The X509 version that the default certificate was created.
- The serial number for the default certificate.
- The issuer and subject distinguished names.
- The default certificates validity period.
- The fingerprint of the default certificate.
- The signature of the algorithm used during creation of the default certificate.
- An indication of the default certificates trust status.

The syntax for listing the details for the default certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -getdefault -db <name> [-type <cms | kdb| pkcs12 | p12>]
[-pw <passwd> | -stashed]
```

where:

object -cert

action -getdefault (**deprecated**)

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file extension of the database file name to determine the keystore type.

Set default certificate in a keystore (-setdefault) [deprecated]

This feature is **deprecated**. Use the explicit label of the desired certificate/key instead.

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.

The syntax for setting the default certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -setdefault -db <name> [-pw <passwd> | -stashed] -label
<label> [-type <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -setdefault (**deprecated**)

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label that uniquely identifies the certificate that is to be set as the default certificate in the identified key database.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file extension of the database file name to determine the keystore type.

Rename a certificate in a keystore (-rename)

The **rename certificate** command changes the label attached to a certificate contained in a CMS keystore.

The syntax for changing a certificate label name in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -rename {-db <filename> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] [-type <cms | kdb| pkcs12 | p12>]
-label <label> -new_label <name> [-secondaryDB <filename> -secondaryDBpw
<password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -rename

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate that is to be renamed.

-new_label <new_name>
A new label name to uniquely identify the certificate in the key database.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file extension of the database file name to determine the keystore type.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

List the certificates stored in a keystore (-list)

The **list certificate** command lists all of the certificates and secret keys stored within the identified key database.

The syntax to list the certificates in an existing key database is as follows:

```
gsk8capicmd -cert -list [<all | personal | CA>] [-expiry [<number of days>]
{-db <name> | -crypto <module_name> -tokenlabel <token_label>}[-pw <passwd> |
-stashed]  [-type <cms | kdb| pkcs12 | p12>] [-secondaryDB <filename> -secondaryDBpw
<password> secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -list

The list command has optional special tags that can be associated with it. These tags are used to identify what type of certificates you are requesting to be displayed. The tags are not required. By default all certificate stored within the key database will be displayed. The following list describes these tags:

all List the labels of all certificates in the identified key database. This is the default for the list command.

personal
List all personal certificates in the identified key database.

CA List all of the certificate authority (CA) certificates in the identified key database.

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:
`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-expiry <number of days>
The expiry tag identifies the number of days from today that a certificate remains valid. If a certificate is due to expire within this number of days then its validity range ("Not Before" and "Not After") displays along with the certificate name. The list also includes the name and validity range details for expired certificates. A certificate that is not expiring within this number of days has only its name listed. You can enter the value `0` to list

expired certificates. If you do not specify this tag, it is not applied during the execution of the command.

-db <filename>
The fully qualified path name of a key database. If the supplied file name has an extension of either ".p12" or .pfx" then it is assumed that it is in PKCS#12 format. If it is ".p7", ".smime" or ".eml" then it is assumed to be a PKCS#7 encoding.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The key database type. The default is **cms**.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

As an example, the following certificate list is displayed for a new key database created with the GSKCapiCmd program. The command used to create this list is as follows:

```
gsk8capicmd –cert –list –db <database name> [-pw <password>]
```

```
Certificates found:
* default, - has private key, ! trusted, # secret key
Entrust.net Global Secure Server Certification Authority
Entrust.net Global Client Certification Authority
Entrust.net Client Certification Authority
Entrust.net Certification Authority (2048)
Entrust.net Secure Server Certification Authority
VeriSign Class 3 Public Primary Certification Authority
VeriSign Class 2 Public Primary Certification Authority
VeriSign Class 1 Public Primary Certification Authority
VeriSign Class 4 Public Primary Certification Authority - G2
VeriSign Class 3 Public Primary Certification Authority - G2
VeriSign Class 2 Public Primary Certification Authority - G2
VeriSign Class 1 Public Primary Certification Authority - G2
VeriSign Class 4 Public Primary Certification Authority - G3
VeriSign Class 3 Public Primary Certification Authority - G3
VeriSign Class 2 Public Primary Certification Authority - G3
VeriSign Class 1 Public Primary Certification Authority - G3
Thawte Personal Premium CA
Thawte Personal Freemail CA
Thawte Personal Basic CA
Thawte Premium Server CA
Thawte Server CA
RSA Secure Server Certification Authority
```

The default key is marked with the '*' symbol (**deprecated**) and all trusted self-signed (root) certs are listed with a '!' symbol. (**deprecated**) The '-' symbol is used to show where a private key is present.

Modify a certificate in a keystore (-modify)

The **modify certificate** command allows a CAs certificate trust status to be enabled or disabled. When a CAs certificate trust status is enabled then that CA certificate is permitted to be involved in a certificate chain validation. If the CAs certificate 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 then the validation of "ABC" will fail. You are able to have any number of trusted CA certificates in the single key database.

The syntax for modifying the trust status of a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -modify -db <name> [-pw <passwd> | -stashed] -label <label>
-trust <enable | disable>
```

where:

object -cert

action -modify

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate.

-trust <enable | disable>
(**Deprecated**). Trust status of a CA certificate. The default is **enable**. When a CAs certificate trust status is enabled then that CA certificate is permitted to be involved in a certificate chain validation. If the CAs certificate 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 then the validation of “ABC” will fail.

Sign a certificate (-sign)

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 a specified file format and details of the certificate that contains the private key to be used during the signing process.

If a certificate is not identified, the private key of the default certificate in the key database is used during the signing process. (**deprecated**)

The syntax for signing a certificate with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -sign {-db <name> -type <cms | kdb| pkcs12 | p12> |
-crypto <module_name> -tokenlabel <label>} [-pw <passwd> | -stashed]
-label <label> -target <name> [-format <ascii | binary>] [-expire
<number of days>]  -file <name> [-secondaryDB <filename> -secondaryDBpw
<password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
[-ca <true | false>] [-san_dnsname <name>] [-san_emailaddr <address>]
[-san_ipaddr <address>] [-certpolicy <policy>] [-ku <name>] [-eku <name>] -preserve
[{-sigalg | -sig_alg} <algorithm>] [-sernum <serial_number>]
```

where:

object -cert

action -sign

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:
`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file name extension to determine the keystore type.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label of the certificate that has the private key to use for the signing operation.

-target <name>
The name of the file that will contain the signed certificate.

-format <acsii | binary>
The format of the signed certificate. The default is Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the DER encoded certificate structure. For additional information, see ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002

-expire <number of days>
The expiry tag identifies the number of days from today that a certificate is valid. The default is **365** days.

Note: To avoid possible timezone issues the actual valid-from time for the certificate will be set one day in the past.

-file <name>
The name and location of the certificate request to be signed.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

-ca <true | false>
This tag adds the Basic Constraint extension to the self-signed certificate. The Basic Constraint extension value is set to true or false depending on what value is associated with the tag.

-san_dnsname <name>
The Subject Alternative Name (SAN) DNS name(s) for the entry being created.

-san_emailaddr <address>
The Subject Alternative Name (SAN) email address(es) for the entry being created. Multiple values can be separated with commas.

-san_ipaddr <address>
The Subject Alternative Name (SAN) IP address(es) for the entry being created.

-certpolicy <policy>
The certificate policy. A named set of rules limiting the applicability of the certificate. Multiple values can be separated with commas.

-ku <list>
Key usage property list. Specifies the valid uses for the private key or certificate.

-eku <list>
Extended key usage property list. Specifies the valid uses for the certificate.

-preserve
Preserve/merge the certificate request attributes in the final certificate extensions.

-sigalg | -sig_alg <signature_algorithm>
The signing algorithm to be used during the signing of the certificate. This algorithm is used to create the signature associated with the new signed certificate. This algorithm must match the keytype of the key being used for signing. That is, the key contained in the certificate that is specified by the –label parameter. See “Signature algorithms” on page 17 for the allowed values.

-sernum
The serial number in conjunction with the issuers name uniquely identifies a certificate. A serial number is normally assigned to a certificate by the certificate authority (CA) that signed the certificate request. This tag has been included to allow the emulation of this process. The -sernum tag accepts two types of values:

1. **Hexadecimal** - A hexadecimal value can be passed as the -sernum tags value by prepending a “0x” to the front of the serial number.
2. **String** - A string representation of the serial number. The string representation of the serial number is normally displayed in ASCII format.

If the –sernum tag is not passed, a random serial number is assigned to the signed certificate.

Validate a certificate (-validate)

The **validate certificate** command is used to validate a certificate held in the keystore. The validation includes ensuring that:

- All necessary intermediate and root certificates used to validate the certificate are present, and
- These certificates have not expired.

The syntax for validating a certificate in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -cert -validate {-db <filename> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] [-type <cms | kdb| pkcs12 | p12>]
-label <label> -ldap <location> [-secondaryDB <filename>
-secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -cert

action -validate

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate that is to be validated.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program uses the file extension of the database file name to determine the keystore type.

-ldap <location>
TCP/IP name or address of the LDAP server that is to be used for certificate revocation checking.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Chapter 4. Certificate request commands

Note: Refer to the "Warning: KeyStore Backups" on page 3 section before updating a keystore.

The certificate request commands are associated with the -certreq object. This object supports the following actions:

- "Create a certificate request (-create)"
- "Delete certificate request (-delete)" on page 47
- "List certificate request details (-details)" on page 49
- "Extract certificate request (-extract)" on page 50
- "List all certificate requests (-list)" on page 51
- "Re-create certificate requests (-recreate)" on page 53

The following sections provide details on how to use each of the identified certificate request actions and what options are available.

Create a certificate request (-create)

The **create certificate request** command creates a new RSA private-public key pair and a PKCS10 certificate request in the specified key database. For CMS key databases, the certificate request information is stored in the file with the ".rdb" extension that is associated with the key database. During the creation process, the certificate request is also extracted to a file that can be used to send the certificate request to a CA for signing.

The syntax for creating a certificate request in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -certreq -create {-db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] [-type <cms | kdb| pkcs12 | p12>]
-label <label> -dn <dist_name> [-size <key_size>] {-target | -file} <name>
[{-sigalg | -sig_alg} <algorithm>] [-secondaryDB <filename> -secondaryDBpw
<password> -secondaryDBtype <cms | kdb| pkcs12 | p12>] [-san_dns_name <name>]
[-san_emailaddr <address>] [-san_ipaddr <address>] [-certpolicy <policy>]
[-ku <name>] [-eku <name>]
```

where:

object -certreq

action -create

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-label <label>
Label to be attached to the certificate request on creation. The user uses this label to uniquely identify the certificate request.

-dn <dist_name>
The X.500 distinguished name that will uniquely identify the certificate. The input must be a quoted string of the following format (only **CN** is required):

```
CN=common name
O=organization
OU=organization unit
L=location
ST=state, province
C=country
DC=domain component
EMAIL=email address
```

For example: “`CN=weblinux.Raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,L=RTP,ST=NC,C=US`”

Multiple OU values are now supported. Simply add additional OU key\value pairs to the specified distinguished name. If the OU value requires a comma (‘,’) then you must escape it with ‘\\’.

For example: “`CN=weblinux.Raleigh.ibm.com,O=ibm,OU=IBM HTTP Server,OU=GSKit\\, Gold Coast,L=RTP,ST=NC,C=US`”

-size <key_size>
The size of the new key pair to be created. This size ranges in value depending on the key type. Consult Table 3 on page 18 for valid values.

Note: For some algorithms, you can specify a zero (0) value to use the default key size. This is typically the minimum size that is considered secure.

-target | -file <name>
The file name that the certificate request will be extracted to during the certificate request creation process.

Note: "-file <name> continues to operate for this command for backwards compatibility.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

-san_dnsname <name>
The Subject Alternative Name (SAN) DNS name(s) for the entry being created.

-san_emailaddr <address>
The Subject Alternative Name (SAN) email address(es) for the entry being created. Multiple values can be separated with commas.

-san_ipaddr <address>
The Subject Alternative Name (SAN) IP address(es) for the entry being created.

-certpolicy <policy>
The certificate policy. A named set of rules limiting the applicability of the certificate. Multiple values can be separated with commas.

-ku <list>
Key usage property list. Specifies the valid uses for the private key or certificate.

-eku <list>
Extended key usage property list. Specifies the valid uses for the certificate.

-sigalg | -sig_alg <signature_algorithm>
The signing algorithm to be used during the creation of the certificate request. This algorithm is used to create the signature associated with the new certificate request. The generated key type will be chosen to match this signing algorithm. See “Signature algorithms” on page 17 for the allowed values.

Delete certificate request (-delete)

Note: Refer to the “Warning: KeyStore Backups” on page 3 section before updating a keystore.

The **delete certificate request** removes the certificate request from the identified key database. This means that the entry in the ".rdb" associated with the certificate request is deleted.

The syntax for deleting a certificate request in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -certreq -delete {-db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed]  [-type <cms | kdb| pkcs12 | p12>]
-label <label> [-secondaryDB <filename> -secondaryDBpw <password>
-secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -certreq

action -delete

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

`'!', '\', '"', '''`

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-label <label>
Label attached to the certificate request that is to be deleted.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space

available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

List certificate request details (-details)

The **list certificate request details** command simple 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 request details use the **-showOID** option in the command.

The syntax for listing a certificate requests details in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -certreq -details [-showOID] [-file <name>] {-db <name> |
-crypto <module_name> -tokenlabel <token_label>}  [-pw <passwd> | -stashed]
[-type <cms | kdb| pkcs12 | p12>] -label <label> [-secondaryDB <filename>
-secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -certreq

action -details

options
IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-showOID
Display a more in-depth listing of the certificate requests.

-file <name>
File name of the certificate request to display. Ascii or binary encoded certificate request.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>

Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>

The PKCS#11 cryptographic device token label.

-pw <passwd>

The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed

The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>

The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-label <label>

Label attached to the certificate request that is to be displayed.

-secondaryDB <filename>

A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>

Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>

Keystore type of the secondary key database.

Extract certificate request (-extract)

The **extract certificate request** command extracts an existing certificate request stored in the specified key database to the identified file in base64 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.

The syntax for extracting a certificate request from an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -certreq -extract {-db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed]  [-type <cms | kdb| pkcs12 | p12>]
-label <label> -target <name> [-secondaryDB <filename>
-secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -certreq

action -extract

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-token_label <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db or –tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-label <label>
Label attached to the certificate request that is to be extracted.

-target <name>
Destination file to which the certificate request is to be extracted.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

List all certificate requests (-list)

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

The syntax for listing the certificate requests stored within an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -certreq -list { -db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] [-type <cms | kdb| pkcs12 | p12>]
[-secondaryDB <filename> -secondaryDBpw <password> -secondaryDBtype
<cms | kdb| pkcs12 | p12>]
```

where:

object -certreq

action -list

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the –db tag or the -crypto tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Re-create certificate requests (-recreate)

The **re-create 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 if there was a problem with the CA that originally signed it.

The syntax to recreate a certificate request in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd  -certreq -recreate { -db <name> | -crypto <module_name> -tokenlabel
<token_label>} [-pw <passwd> | -stashed] [-type <cms | kdb| pkcs12 | p12>] -label
<label> -target <name> [-secondaryDB <filename> -secondaryDBpw <password>
-secondaryDBtype <cms | kdb| pkcs12 | p12>] [-san_dnsname <name>] [-san_emailaddr
<address>] [-san_ipaddr <address>] [-certpolicy <policy>] [-ku <list>] [-eku <list>]
[-sigalg | -sig_alg <signature_algorithm>]
```

where:

object -certreq

action -recreate

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-type <cms | kdb | pkcs12 | p12>
The keystore type. If this option is not specified, the program will use the database path name suffix to determine the keystore type.

-pw <passwd>
The password for the key database identified by the –db tag or the -crypto tag. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the certificate request that is to be recreated.

-target <name>
Destination file to which the certificate request is to be recreated.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device. A PKCS#11 device does not normally have a large amount of space available to store a lot of signer certificates. The signer certificates are used for the validation of certificates when they are added to the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

-san_dnsname <name>
The Subject Alternative Name (SAN) DNS name(s) for the entry being created.

-san_emailaddr <address>
The Subject Alternative Name (SAN) email address(es) for the entry being created. Multiple values can be separated with commas.

-san_ipaddr <address>
The Subject Alternative Name (SAN) IP address(es) for the entry being created.

-certpolicy <policy>
The certificate policy. A named set of rules limiting the applicability of the certificate. Multiple values can be separated with commas.

-ku <list>
Key usage property list. Specifies the valid uses for the private key or certificate.

-eku <list>
Extended key usage property list. Specifies the valid uses for the certificate.

-sigalg | -sig_alg <signature_algorithm>
The signing algorithm to be used during the creation of the self-signed certificate. This algorithm is used to create the signature associated with the new self-signed certificate. The key type must match this signing algorithm. See “Signature algorithms” on page 17 for the allowed values.

Chapter 5. Secret Key commands

Note: : "-cert –list" and "–cert –details" will function on secret keys providing information regarding the specific secret key.

Create a random secret key to a keystore (-create)

Note that the key value created here will be a random binary value of the specified length. No processing of the random data shall be done and its use as a secret key of a specific type is up to the application using the value. E.g. If the key value shall be used for triple DES then the user must parity correct the value.

The syntax for creating a random secret key in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -secretkey -create {-db <name> -type <cms | kdb| pkcs12 |
p12> -crypto <module_name> -tokenlabel <token_label>} [-pw <passwd> |
-stashed] -label <label> -size <key length in bytes> [-secondaryDB
<filename> -secondaryDBpw <password> -secondaryDBtype <cms | kdb|
pkcs12 | p12>]
```

where:

object -secretkey

action -create

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (""). You must also use a backslash ('\') character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12>
The type of the key database. If this option is not present, the type is implied by the file extension of database path name.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the -db or

-tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the key.

-size <key length>
Key length in bytes.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Add a secret key to a keystore (-add)

The syntax for adding a secret key in an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -secretkey -add {-db <name> -type <cms | kdb| pkcs12 | p12>
-crypto <module_name> -tokenlabel <token_label>} [-pw <passwd> | -stashed]
-label <label> -file <name> [-format <ascii | binary>] [-secondaryDB <filename>
-secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -secretkey

action -add

options

IMPORTANT: On UNIX operating systems, always encapsulate string values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-type <cms | kdb | pkcs12 | p12>
Type of the key database. If this option is not present, the type is implied by the file extension of database path name.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the -db or -tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the key.

file <name>
File name of the key to add.

-format <ascii | binary>
Format of a certificate The default is Base64 encoded ASCII. Additional information about Base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of key.

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Extract a Secret Key from a keystore (-extract)

The **extract secret key** command simply extracts the key data from the key database and places it into the identified file. If the file does not exist then it will be created. If the file already exists, an error indicating this will be returned. The data will be saved as either base64 encoding or binary.

The syntax to extract a secret key from an existing key database with GSKCapiCmd is as follows:

```
gsk8capicmd -secretkey -extract {-db <name> | -crypto <module_name>
-tokenlabel <token_label>} [-pw <passwd> | -stashed] -label <label>
-target <name> [-format <ascii | binary>] [-secondaryDB <filename>
-secondaryDBpw <password> -secondaryDBtype <cms | kdb| pkcs12 | p12>]
```

where:

object -secretkey

action -extract

options
IMPORTANT: On UNIX operating systems, always encapsulate string

values associated with all tags in double quotation marks (“”). You must also use a backslash (‘\’) character to escape the following characters if they appear in the string values:

```
'!', '\', '"', '''
```

This will prevent some command-line shells from interpreting specific characters within these values. For example: `gsk8capicmd –keydb –create –db "/tmp/key.kdb" –pw "j\!jj"`. When prompted by **gsk8capicmd** for a value (for example, a password) do not quote the string and add the escape characters, as the shell is no longer influencing this input.

-db <filename>
The fully qualified path name of a key database.

-crypto <module_name>
Indicates a PKCS#11 cryptographic device operation, where <module_name> is the path to the module to manage the crypto device.

-tokenlabel <token_label>
The PKCS#11 cryptographic device token label.

-pw <passwd>
The password for the key database identified by the -db or -tokenlabel tags. Specify a hyphen (-) as the password to cause the program to read the password from stdin. This allows you to pipe in the password.

-stashed
The password for the key database will be recovered from the stash file.

-label <label>
Label attached to the key that is to be extracted.

-target <name>
Destination file to which the key is to be extracted.

-format <ascii | binary>
Format of data. The default is Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the key data .

-secondaryDB <filename>
A CMS key database used to support the PKCS#11 device.

-secondaryDBpw <password>
Password for the secondary CMS key database supporting the PKCS#11 device.

-secondaryDBtype <cms | kdb| pkcs12 | p12>
Keystore type of the secondary key database.

Chapter 6. Random commands

The GSKCapiCmd program provides its users with the ability to generate random passwords. Users can specify the password length and whether the generated password is required to conform to GSKits minimum password requirements.

The random commands are associated with the -random object. This object supports the following action:

"Create a random password of a specified length (-create)"

The following section describes how to use and what options are available for this random action.

Create a random password of a specified length (-create)

The **create random password** command creates a random string of characters that can be used with other GSKCapiCmd commands that require a password. This command can be used if the user is looking for a truly random password.

The syntax for creating a random password with GSKCapiCmd is as follows:

```
gsk8capicmd -random -create -length <password_length> -strong
```

where:

object -random

action -create

options

-length <password_length>
The length of the random password. There is a maximum length when the **-strong** tag is used for this command. The maximum length is 125 character.

-strong
Check that the password entered satisfies the minimum requirements for the passwords strength. The minimum requirements for a password are as follows:

- The minimum password length is 14 characters.
- A password must have at least one lower case character, one uppercase character, and one digit or special character (for example, *$#% etc). A space is classified as a special character.
- Each character must not occur more than three times in a password.
- No more than two consecutive characters of the password can be identical.
- All characters are in the standard ASCII printable character set within the range from 0x20 to 0x7E inclusive.

Chapter 7. Help commands

GSKCapiCmd has an extensive help command system. You are able to get help on what objects are available, what actions are associated with a particular object, and how to use each of the actions.

The **help** commands are associated with the -help object. The syntax for the help commands is as follows:

```
gsk8capicmd -help <object> <action>
```

where:

<*object*>
: The object you want to find out information about.

<*action*>
: The action you are wanting to find out information about. This action must be associated with the identified object. If it is not the system will display the help associated with the requested object.

Examples:

- Listing all of the objects and their associated actions: `gsk8capicmd -help`
- Listing the actions for the **-keybd** object: `gsk8capicmd -help -keydb`
- Listing the specific help for the **-create** action associated with the **-keydb** object. `gsk8capicmd -help -keydb -create`

To find out the different objects and their associated actions see:

- Chapter 2, “Key database commands,” on page 7
- Chapter 3, “Certificate commands,” on page 17
- Chapter 4, “Certificate request commands,” on page 45

Chapter 8. Version command

The **version** command displays version information associated with the currently installed GSKCapiCmd program.

The version command is associated with the **-version** object. The syntax is as follows:

```
gsk8capicmd -version
```

The version command has no associated actions or objects.

Chapter 9. Runtime messages

This chapter describes the messages displayed by GSKCapiCmd.

Table 6. Runtime Messages

Message ID	Message Details	Explanation
CTGSK2000W	The task completed successfully.	See message.
CTGSK2001W	Unknown error occurred	See message.
CTGSK2002W	An ASN.1 encoding/decoding error occurred.	See message.
CTGSK2003W	An error occurred while initializing ASN.1 encoder/decoder.	See message.
CTGSK2004W	An ASN.1 encoding/decoding error occurred because of an out-of-range index or non-existent optional field.	See message.
CTGSK2005W	A database error occurred.	See message.
CTGSK2006W	An error occurred while opening the database file, check for file existence and permission.	See message.
CTGSK2007W	An error occurred while re-opening the database file.	See message.
CTGSK2008W	Database creation failed.	See message.
CTGSK2009W	The database already exists.	See message.
CTGSK2010W	An error occurred while deleting the database file.	See message.
CTGSK2011W	The database could not be opened.	See message.
CTGSK2012W	An error occurred while reading the database file.	See message.
CTGSK2013W	An error occurred while writing data to the database file.	See message.
CTGSK2014W	A database validation error occurred.	See message.
CTGSK2015W	An invalid database version was encountered.	See message.
CTGSK2016W	An invalid database password was encountered.	See message.
CTGSK2017W	An invalid database file type was encountered.	See message.
CTGSK2018W	The specified database has been corrupted.	See message.
CTGSK2019W	An invalid password was provided or the key database has been tampered or corrupted.	See message.
CTGSK2020W	A database key entry integrity error occurred.	See message.
CTGSK2021W	A duplicate certificate already exists in the database.	See message.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK2022W	A duplicate key already exists in the database (Record ID).	See message.
CTGSK2023W	A certificate with the same label already existed in the key database.	See message.
CTGSK2024W	A duplicate key already exists in the database (Signature).	See message.
CTGSK2025W	A duplicate key already exists in the database (Unsigned Certificate).	See message.
CTGSK2026W	A duplicate key already exists in the database (Issuer and Serial Number).	See message.
CTGSK2027W	A duplicate key already exists in the database (Subject Public Key Info).	See message.
CTGSK2028W	A duplicate key already exists in the database (Unsigned Certificate Revocation List (CRL)).	See message.
CTGSK2029W	The label has been used in the database.	See message.
CTGSK2030W	A password encryption error occurred.	See message.
CTGSK2031W	An LDAP related error occurred.	Reserved for future use.
CTGSK2032W	A cryptographic error occurred.	See message.
CTGSK2033W	An encryption/decryption error occurred.	See message.
CTGSK2034W	An invalid cryptographic algorithm was found.	See message.
CTGSK2035W	An error occurred while signing data.	See message.
CTGSK2036W	An error occurred while verifying data.	See message.
CTGSK2037W	An error occurred while computing digest of data.	See message.
CTGSK2038W	An invalid cryptographic parameter was found.	See message.
CTGSK2039W	An unsupported cryptographic algorithm was encountered.	See message.
CTGSK2040W	The specified input size is greater than the supported modulus size.	See message.
CTGSK2041W	An unsupported modulus size was found.	See message.
CTGSK2042W	A database validation error occurred.	See message.
CTGSK2043W	Key entry validation failed.	See message.
CTGSK2044W	A duplicate extension field exists.	See message.
CTGSK2045W	The version of the key is wrong.	See message.
CTGSK2046W	A required extension field does not exist.	See message.
CTGSK2047W	The validity period does not include today or does not fall within its issuer's validity period.	See message.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK2048W	The validity period does not include today or does not fall within its issuer's validity period.	See message.
CTGSK2049W	An error occurred while validating validity private key usage extension.	See message.
CTGSK2050W	The issuer of the key was not found.	See message.
CTGSK2051W	A required certificate extension is missing.	See message.
CTGSK2052W	An invalid basic constraint extension was found.	See message.
CTGSK2053W	The key signature validation failed.	See message.
CTGSK2054W	The root key of the key is not trusted.	See message.
CTGSK2055W	The key has been revoked.	See message.
CTGSK2056W	An error occurred while validating authority key identifier extension.	See message.
CTGSK2057W	An error occurred while validating private key usage extension.	See message.
CTGSK2058W	An error occurred while validating subject alternative name extension.	See message.
CTGSK2059W	An error occurred while validating issuer alternative name extension.	See message.
CTGSK2060W	An error occurred while validating key usage extension.	See message.
CTGSK2061W	An unknown critical extension was found.	See message.
CTGSK2062W	An error occurred while validating key pair entries.	See message.
CTGSK2063W	An error occurred while validating CRL.	See message.
CTGSK2064W	A mutex error occurred.	See message.
CTGSK2065W	An invalid parameter was found.	See message.
CTGSK2066W	A null parameter or memory allocation error was encountered.	See message.
CTGSK2067W	Number or size is too large or too small.	See message.
CTGSK2068W	The old password is invalid.	See message.
CTGSK2069W	The new password is invalid.	See message.
CTGSK2070W	The password has expired.	See message.
CTGSK2071W	A thread related error occurred.	See message.
CTGSK2072W	An error occurred while creating threads.	See message.
CTGSK2073W	An error occurred while a thread was waiting to exit.	See message.
CTGSK2074W	An I/O error occurred.	See message.
CTGSK2075W	An error occurred while loading CMS.	See message.
CTGSK2076W	A cryptography hardware related error occurred.	See message.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK2077W	The library initialization routine was not successfully called.	See message.
CTGSK2078W	The internal database handle table is corrupted.	See message.
CTGSK2079W	A memory allocation error occurred.	See message.
CTGSK2080W	An unrecognized option was found.	See message.
CTGSK2081W	An error occurred while getting time information.	See message.
CTGSK2082W	Mutex creation error occurred.	See message.
CTGSK2083W	An error occurred while opening message catalog.	See message.
CTGSK2084W	An error occurred while opening error message catalog.	See message.
CTGSK2085W	An null file name was found.	See message.
CTGSK2086W	An error occurred while opening files, check for file existence and permissions.	See message.
CTGSK2087W	An error occurred while opening files to read.	See message.
CTGSK2088W	An error occurred while opening files to write.	See message.
CTGSK2089W	There is no such file.	See message.
CTGSK2090W	The file cannot be opened because of its permission setting.	See message.
CTGSK2091W	An error occurred while writing data to files.	See message.
CTGSK2092W	An error occurred while deleting files.	See message.
CTGSK2093W	Invalid Base64-encoded data was found.	See message.
CTGSK2094W	An invalid Base64 message type was found.	See message.
CTGSK2095W	An error occurred while encoding data with Base64 encoding rule.	See message.
CTGSK2096W	An error occurred while decoding Base64-encoded data.	See message.
CTGSK2097W	An error occurred while getting a distinguished name tag.	See message.
CTGSK2098W	The required common name field is empty.	See message.
CTGSK2099W	The required country or region name field is empty.	See message.
CTGSK2100W	An invalid database handle was found.	See message.
CTGSK2101W	The key database does not exist.	See message.
CTGSK2102W	The request key pair database does not exist.	See message.
CTGSK2103W	The password file does not exist.	See message.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK2104W	The new password is identical to the old one.	See message.
CTGSK2105W	No key was found in the key database.	See message.
CTGSK2106W	No request key was found.	See message.
CTGSK2107W	No trusted CA was found	See message.
CTGSK2108W	No request key was found for the certificate.	See message.
CTGSK2109W	There is no private key in the key database	See message.
CTGSK2110W	There is no default key in the key database.	See message.
CTGSK2111W	There is no private key in the key record.	See message.
CTGSK2112W	There is no certificate in the key record.	See message.
CTGSK2113W	There is no CRL entry.	See message.
CTGSK2114W	An invalid key database file name was found.	See message.
CTGSK2115W	An unrecognized private key type was found.	See message.
CTGSK2116W	An invalid distinguished name input was found.	See message.
CTGSK2117W	No key entry was found that has the specified key label.	See message.
CTGSK2118W	The key label list has been corrupted.	See message.
CTGSK2119W	The input data is not valid PKCS#12 data.	See message.
CTGSK2120W	The password is invalid or the PKCS#12 data has been corrupted or been created with a later version of PKCS#12.	See message.
CTGSK2121W	An unrecognized key export type was found.	See message.
CTGSK2122W	An unsupported password-based encryption algorithm was found.	See message.
CTGSK2123W	An error occurred while converting the key ring file to a CMS key database.	See message.
CTGSK2124W	An error occurred while converting the CMS key database to a keyring file.	See message.
CTGSK2125W	An error occurred while creating a certificate for the certificate request.	See message.
CTGSK2126W	A complete issuer chain cannot be built.	See message.
CTGSK2127W	Invalid WEBDB data was found.	See message.
CTGSK2128W	There is no data to be written to the key ring file.	See message.
CTGSK2129W	The number of days that you entered extends beyond the permitted validity period.	See message.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK2130W	The password is too short.	See message.
CTGSK2131W	A password must contain at least one numeric digit.	See message.
CTGSK2132W	All characters in the password are either alphabetic or numeric characters.	See message.
CTGSK2133W	An unrecognized or unsupported signature algorithm was specified.	See message.
CTGSK2134W	An invalid database type was encountered.	See message.
CTGSK2135W	The specified secondary key database is in use by another PKCS#11 device.	See message.
CTGSK2136W	No secondary key database was specified.	See message.
CTGSK2137W	The label does not exist on the PKCS#11 device.	See message.
CTGSK2138W	Password required to access the PKCS#11 device.	See message.
CTGSK2139W	Password not required to access the PKCS#11 device.	See message.
CTGSK2140W	Unable to load the cryptographic library.	See message.
CTGSK2141W	PKCS#11 is not supported for this operation.	See message.
CTGSK2142W	An operation on a PKCS#11 device has failed.	See message.
CTGSK2143W	The LDAP user is not a valid user.	See message.
CTGSK2144W	The LDAP user password is not valid.	See message.
CTGSK2145W	The LDAP query failed.	See message.
CTGSK2146W	An invalid certificate chain was found.	See message.
CTGSK2147W	The root certificate is not trusted.	See message.
CTGSK2148W	A revoked certificate was encountered.	See message.
CTGSK2149W	A cryptographic object function failed.	See message.
CTGSK2150W	There is no certificate revocation list data source available.	See message.
CTGSK2151W	There is no cryptographic token available.	See message.
CTGSK2152W	FIPS mode is not available.	See message.
CTGSK2153W	There is a conflict with the FIPS mode settings.	See message.
CTGSK2154W	The password entered does not meet the minimum required strength.	See message.
CTGSK3000W	An action must be specified for this object.	An expected command line option was missing. **Operator response:** Reissue the command and include all of the required options.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK3001W	Cannot load keystore: *filename*	The nominated keystore cannot be opened. **Operator response:** Ensure that the keystore is valid and of the correct type.
CTGSK3002W	Error creating certificate.	An error occurred while trying to create a certificate with the selected options. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3003W	A certificate error has occurred.	An error occurred trying to set the trust state of the certificate. **Operator response:** Review the details of the certificate and ensure the attempted action matches the certificate details.
CTGSK3004W	Error decoding certificate request for label *name*.	The certificate request file may have been corrupted. **Operator response:** Ensure the certificate request file is valid and readable.
CTGSK3005W	Error storing one or more certificates.	The certificate could not be stored. **Operator response:**Ensure the certificate data is valid and readable.
CTGSK3006W	The private key for entry "*filename*" cannot be decrypted. It may have been corrupted.	There is no private key in the key record. **Operator response:** Ensure the certificate data is valid and readable.
CTGSK3007W	The database password has expired.	The password of the key database has expired or the database cannot be accessed. **Operator response:**The password must be reset before the keystore can be opened.
CTGSK3008W	The entry for label "*name*" could not be deleted.	An attempt to delete an object from a database failed. **Operator response:**Ensure the database is accessible and the correct object has been nominated for deletion.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK3009W	One or more certificates in the keystore could not be loaded.	An attempt to load or renew a certificate failed. **Operator response:** Ensure the key database is accessible and the certificate has not been corrupted.
CTGSK3010W	Error loading entry "*name*"	Cannot load the named certificate from the keystore. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3011W	Failed to delete file "*name*"	Cannot delete the specified file. **Operator response:** Check the file name and file permissions are correct.
CTGSK3012W	The input file "*name*" could not be found.	Cannot find the specified file. **Operator response:** Check the database path.
CTGSK3013W	"*other*" is not a valid action for this object.	The specified action does not apply to the selected object. **Operator response:**Review the command line options selected, correct any problems and retry the operation.
CTGSK3014W	An invalid parameter was specified: *name*	An unknown or invalid algorithm was specified. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3015W	Invalid certificate details: *name*	The certificate inspected was not valid. **Operator response:** No action is required.
CTGSK3016W	Cannot parse certificates in file "*name*". It is not a valid certificate file.	The wrong file has been specified or it has been corrupted. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3017W	The database type "*name*" is not recognized.	The database type is not known or not supported for this command. **Operator response:** Change the database type and try again.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK3018W	The action "*name*" is not supported for database type "*other*".	The attempted action was not supported for the database. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3019W	Invalid DN.	The DN was not complete or contained errors. **Operator response:** A valid DN consists of a comma delimited list of attribute value pairs; e.g. "CN=John Smith, OU=Tivoli, O=IBM, C=US"
CTGSK3020W	Invalid object: *other*	An unknown command line option was specified. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3021W	Invalid parameter: *name*	An illegal value was specified for a command line parameter. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3022W	An invalid parameter was provided for the command: *name*	An invalid value was specified for a command line parameter. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3023W	An invalid password was provided, the key database has been corrupted or it is of the wrong type.	The supplied password did not unlock the selected database. **Operator response:** Ensure that the database is of the correct type and that the correct password was given.
CTGSK3024W	Invalid value for parameter "*other*" (*other*).	The supplied value is not valid for the named parameter. **Operator response:** Review the command line options selected, correct any problems and retry the operation.

Table 6. Runtime Messages (continued)

Message ID	**Message Details**	**Explanation**
CTGSK3025W	An invalid version was provided for certificate "*name*".	The X509 version is not valid. Some features are not supported by all X509 versions. For example, SAN extensions are only supported by version 3 and above. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3026W	The key file "*name*" does not exist or cannot be read.	The specified key file could not be opened. **Operator response:** Check that the file exists and that you have read permissions.
CTGSK3027W	An error has occurred while closing the keystore.	An error occurred accessing the keystore file. **Operator response:** Check that the directory and file permissions allow access to the file.
CTGSK3028W	An error has occurred while accessing the keystore.	An unknown error occurred while accessing the keystore file. **Operator response:** Check that the directory and file permissions allow access to the file.
CTGSK3029W	The database does not contain a certificate with label "*name*"	No certificate exists with that name. **Operator response:** Check the label and try again.
CTGSK3030W	The database does not contain an entry with label "*name*"	No certificate exists with that name. **Operator response:** Check the label and try again.
CTGSK3031W	The file name cannot be null.	A legal file name must be supplied. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3032W	The database does not contain a key entry with label "*name*"	No certificate exists with that name. **Operator response:** Check the label and try again.

Table 6. Runtime Messages (continued)

Message ID	**Message Details**	**Explanation**
CTGSK3033W	No read permissions for file "*name*"	The file cannot be accessed. **Operator response:** Check the file path and ensure that you have read/write permissions to the directory and file.
CTGSK3034W	The certificate request created for the certificate is not in the key database.	No request exists for the named certificate. **Operator response:** Check that you are using the correct keystore.
CTGSK3035W	The database does not contain a certificate request with label "*name*"	No request exists for the named certificate. **Operator response:** Check the label and try again.
CTGSK3036W	The output file "*name*" already exists.	An existing file will not be over written. **Operator response:**Choose a different name or delete the file if it is no longer required.
CTGSK3037W	The output file "*name*" could not be created.	A file could not be created. **Operator response:** Check the file path and ensure that you have read/write permissions to the directory.
CTGSK3038W	An error occurred while trying to change the password of the request database.	The changed password could not be written to the password stash file. **Operator response:** Check the file path to the key database and stash files and ensure that you have appropriate read/write permissions to the directory.
CTGSK3039W	Certificate request "*name*" could not be created.	The certificate request could not be created. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3040W	A required value for the command was not specified: *name*	The request action is missing a required option. **Operator response:** Review the command line options selected, correct any problems and retry the operation.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK3041W	A signature error has occurred while signing item "*name*"	A problem occurred while attempting to issue a signed certificate. **Operator response:** Review the command line options to ensure the selected options are valid and compatible. Retry the operation.
CTGSK3042W	Invalid value for parameter "*name*" was specified: *other*	An invalid value was supplied for an option. **Operator response:** Replace the parameters that are not valid and try again.
CTGSK3043W	Unknown parameter "*name*"	An unknown parameter was supplied for an option. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3044W	No value for parameter "*name*" was provided.	A value was expected but none was found. **Operator response:** Add a valid value for the parameter and re-run the command.
CTGSK3045W	An error has occurred while reading the keystore.	The keystore cannot be read. **Operator response:** Check the path to the key database and ensure that you have read/write permissions to the file.
CTGSK3046W	The key file "*name*" could not be imported.	An error occurred reading the keystore. **Operator response:** Check the path to the key database and ensure that you have read/write permissions to the file.
CTGSK3047W	Option "*name*" is not supported in FIPS mode.	The operation is not supported in FIPS mode. **Operator response:** Review the command line options selected, correct any problems and retry the operation.
CTGSK3048W	The password is weak.	The supplied password is too weak. **Operator response:** Try a different password.

Table 6. Runtime Messages (continued)

Message ID	Message Details	Explanation
CTGSK3049W	An attempted operation has failed: *name other* : *detail*	An unexpected internal error occurred. **Operator response:** Contact your support team.
CTGSK3050W	An internal error occurred while attempting to work with an ASN extension: *error*	An unexpected internal error occurred. The first parameter is the reported GSK error code. **Operator response:** Contact your support team.
CTGSK3051W	The current codepage "*id*" is not supported, encoding for codepage "*name*" instead.	No ICU converter exists for the codepage the console has been set to. **Operator response:**Set the console codepage to something else compatible with the locale's output.
CTGSK3052W	"*name*" is not a supported signature algorithm.	The chosen signature algorithm is not supported for this operation. **Operator response:** Retry the operation using a different signature algorithm.
CTGSK3053W	The database already contains a certificate with label "*name*"	A certificate exists with that name. **Operator response:** Check the label and try again.
CTGSK3054W	An error occurred while trying to stash the password for the database "*name*".	The changed password could not be written to the password stash file. **Operator response:** Check the file path to the key database and stash files and ensure that you have appropriate read/write permissions to the directory.
CTGSK3055W	A "*error*" error occurred while formatting the date and time information for locale "*name*".	An error occurred formatting the date and time information for the locale. This is an error reported from the ICU package. **Operator response:** Retry the command specifying a different locale. Contact your support team.
CTGSK3056W	The file is not of the correct type.	The file type is not known or not supported for this command. **Operator response:** Change the file type and try again.

Table 6. Runtime Messages (continued)

Message ID	**Message Details**	**Explanation**
CTGSK3057W	Too many choices were selected, "*name*" is not valid in this context.	Conflicting options have been selected on the command line. **Operator response:** Review the command line options, correct any problems and retry the operation.
CTGSK3058W	There is no data to write to file "*name*". An empty file will not be created.	There is no data to write to the file and an empty file is not valid. **Operator response:** No action is required.

Chapter 10. Error codes and messages

GSKCapiCmd, returns GSKKM_OK (0) on success or a positive number indicating the error that has occurred. The following table lists all of the error codes and their associated error messages.

Table 7. Error Messages

Error Code	Error Message
1	Unknown error occurred
2	An asn.1 encoding/decoding error occurred.
3	An error occurred while initializing asn.1 encoder/decoder.
4	An asn.1 encoding/decoding error occurred because of an out-of-range index or non-existent optional field.
5	A database error occurred.
6	An error occurred while opening the database file, check for file existence and permission.
7	An error occurred while re-opening the database file.
8	Database creation failed.
9	The database already exists.
10	An error occurred while deleting the database file.
11	The database could not be opened.
12	An error occurred while reading the database file.
13	An error occurred while writing data to the database file.
14	A database validation error occurred.
15	An invalid database version was encountered.
16	An invalid database password was encountered.
17	An invalid database file type was encountered.
18	The specified database has been corrupted.
19	An invalid password was provided or the key database has been tampered or corrupted.
20	A database key entry integrity error occurred.
21	A duplicate certificate already exists in the database.
22	A duplicate key already exists in the database (Record ID).
23	A certificate with the same label already existed in the key database.
24	A duplicate key already exists in the database (Signature).
25	A duplicate key already exists in the database (Unsigned Certificate).
26	A duplicate key already exists in the database (Issuer and Serial Number).
27	A duplicate key already exists in the database (Subject Public Key Info).
28	A duplicate key already exists in the database (Unsigned CRL).
29	The label has been used in the database.
30	A password encryption error occurred.
31	An LDAP related error occurred.

Table 7. Error Messages (continued)

Error Code	Error Message
32	A cryptographic error occurred.
33	An encryption/decryption error occurred.
34	An invalid cryptographic algorithm was found.
35	An error occurred while signing data.
36	An error occurred while verifying data.
37	An error occurred while computing digest of data.
38	An invalid cryptographic parameter was found.
39	An unsupported cryptographic algorithm was encountered.
40	The specified input size is greater than the supported modulus size.
41	An unsupported modulus size was found.
42	A database validation error occurred.
43	Key entry validation failed.
44	A duplicate extension field exists.
45	The version of the key is wrong.
46	A required extension field does not exist.
47	The validity period does not include today or does not fall within its issuer's validity period.
48	The validity period does not include today or does not fall within its issuer's validity period.
49	An error occurred while validating validity private key usage extension.
50	The issuer of the key was not found.
51	A required certificate extension is missing.
52	An invalid basic constraint extension was found.
53	The key signature validation failed.
54	The root key of the key is not trusted.
55	The key has been revoked.
56	An error occurred while validating authority key identifier extension.
57	An error occurred while validating private key usage extension.
58	An error occurred while validating subject alternative name extension.
59	An error occurred while validating issuer alternative name extension.
60	An error occurred while validating key usage extension.
61	An unknown critical extension was found.
62	An error occurred while validating key pair entries.
63	An error occurred while validating CRL.
64	A mutex error occurred.
65	An invalid parameter was found.
66	A null parameter or memory allocation error was encountered.
67	Number or size is too large or too small.
68	The old password is invalid.
69	The new password is invalid.

Table 7. Error Messages (continued)

Error Code	Error Message
70	The password has expired. (**deprecated**)
71	A thread related error occurred.
72	An error occurred while creating threads.
73	An error occurred while a thread was waiting to exit.
74	An I/O error occurred.
75	An error occurred while loading CMS.
76	A cryptography hardware related error occurred.
77	The library initialization routine was not successfully called.
78	The internal database handle table is corrupted.
79	A memory allocation error occurred.
80	An unrecognized option was found.
81	An error occurred while getting time information.
82	Mutex creation error occurred.
83	An error occurred while opening message catalog.
84	An error occurred while opening error message catalog.
85	A null file name was found.
86	An error occurred while opening files, check for file existence and permissions.
87	An error occurred while opening files to read.
88	An error occurred while opening files to write.
89	There is no such file.
90	The file cannot be opened because of its permission setting.
91	An error occurred while writing data to files.
92	An error occurred while deleting files.
93	Invalid Base64-encoded data was found.
94	An invalid Base64 message type was found.
95	An error occurred while encoding data with Base64 encoding rule.
96	An error occurred while decoding Base64-encoded data.
97	An error occurred while getting a distinguished name tag.
98	The required common name field is empty.
99	The required country or region name field is empty.
100	An invalid database handle was found.
101	The key database does not exist.
102	The request key pair database does not exist.
103	The password file does not exist.
104	The new password is identical to the old one.
105	No key was found in the key database.
106	No request key was found.
107	No trusted CA was found
108	No request key was found for the certificate.

Table 7. Error Messages (continued)

Error Code	Error Message
109	There is no private key in the key database
110	There is no default key in the key database. (**deprecated**)
111	There is no private key in the key record.
112	There is no certificate in the key record.
113	There is no CRL entry.
114	An invalid key database file name was found.
115	An unrecognized private key type was found.
116	An invalid distinguished name input was found.
117	No key entry was found that has the specified key label.
118	The key label list has been corrupted.
119	The input data is not valid PKCS#12 data.
120	The password is invalid or the PKCS#12 data has been corrupted or been created with later version of PKCS#12.
121	An unrecognized key export type was found.
122	An unsupported password-based encryption algorithm was found.
123	An error occurred while converting the key ring file to a CMS key database.
124	An error occurred while converting the CMS key database to a key ring file.
125	An error occurred while creating a certificate for the certificate request.
126	A complete issuer chain cannot be built.
127	Invalid WEBDB data was found.
128	There is no data to be written to the key ring file.
129	The number of days that you entered extends beyond the permitted validity period.
130	The password is too short; it must consist of at least {0} characters.
131	A password must contain at least one numeric digit.
132	All characters in the password are either alphabetic or numeric characters.
133	An unrecognized or unsupported signature algorithm was specified.
134	An invalid database type was encountered.
135	The specified secondary key database is in use by another PKCS#11 device.
136	No secondary key database was specified.
137	The label does not exist on the PKCS#11 device.
138	Password required to access the PKCS#11 device.
139	Password not required to access the PKCS#11 device.
140	Unable to load the cryptographic library.
141	PKCS#11 is not supported for this operation.
142	An operation on a PKCS#11 device has failed.
143	The LDAP user is not a valid user.
144	The LDAP user password is not valid.
145	The LDAP query failed.
146	An invalid certificate chain was found.

Table 7. Error Messages (continued)

Error Code	Error Message
147	The root certificate is not trusted.
148	A revoked certificate was encountered.
149	A cryptographic object function failed.
150	There is no certificate revocation list data source available.
151	There is no cryptographic token available.
152	FIPS mode is not available.
153	There is a conflict with the FIPS mode settings.
154	The password entered does not meet the minimum required strength.
200	There was a failure during initialization of the program.
201	Tokenization of the arguments passed to the GSKCapiCmd Program failed.
202	The object identified in the command is not a recognized object.
203	The action passed is not a known -keydb action.
204	The action passed is not a known -cert action.
205	The action passed is not a known -certreq action.
206	There is a tag missing for the requested command.
207	The value passed with the –version tag is not a recognized value.
208	The value passed with the –size tag is not a recognized value.
209	The value passed in with the –dn tag is not in the correct format.
210	The value passed in with the –format tag is not a recognized value.
211	There was an error associated with opening the file.
212	PKCS#12 is not supported at this stage.
213	The cryptographic token you are trying to change the password for is not password protected.
214	PKCS#12 is not supported at this stage.
215	The password entered does not meet the minimum required strength.
216	FIPS mode is not available.
217	The number of days you have entered as the expiry date is out of the allowed range.
218	Password strength failed the minimum requirements.
219	No Default certificate was found in the requested key database. (**deprecated**)
220	An invalid trust status was encountered.
221	An unsupported signature algorithm was encountered. At this stage only MD5 and SHA1 are supported.
222	PKCS#11 not supported for that particular operation.
223	The action passed is not a known –random action.
224	A length than less than zero is not allowed.
225	When using the –strong tag the minimum length password is 14 characters.
226	When using the –strong tag the maximum length password is 300 characters.
227	The MD5 algorithm is not supported when in FIPS mode.
228	The site tag is not supported for the –cert –list command. This attribute is simply added for backward compatibility and potential future enhancement.

Table 7. Error Messages (continued)

Error Code	Error Message
229	The value associated with the -ca tag is not recognized. The value must be either ‘true’ or ‘false’.
230	The value passed in with the –type tag is not valid.
231	The value passed in with the –expire tag is below the allowed range.
232	The encryption algorithm used or requested is not supported.
233	The target already exists.

Appendix A. CMS key databases

What is a CMS key database?

Certificate Management System (CMS) is the native GSKit key database (keystore) containing:

- X.509 certificates
- Certificate requests (ones pending signing by an authority), and
- Private keys for the stored certificates where applicable.

Typically, only personal certificates contain private keys. If a certificate has an associated private key, it is stored encrypted in the keystore with its associated certificate. Private keys cannot be stored without an associated certificate.

How is a CMS key database organized?

A CMS keystore consists of a file with extension .kdb and optionally two other files with extension .rdb and .crl respectively.

A key record in a .kdb file is either a certificate on its own or a certificate plus its encrypted private key information. Private keys cannot be stored in a CMS keystore without a corresponding certificate.

When a certificate request is created, a .rdb file with the same file stem as the key database file is created. This file is used to store the requested key pair, along with the PKCS#10 certificate request data. The request entry is only deleted from the request key database when a signed certificate is obtained from a signing authority and received into the key database. The signed certificate is matched up with the private key in the .rdb file and together they are added to the .kdb file as a certificate with private key information.

A .crl file is also created, purely for legacy reasons (in the past it contained Certificate Revocation Lists (CRLs)). This file is no longer used and is always empty.

How is a CMS key database protected

GSKit implements password protection for access control, confidentiality, and integrity of the CMS key database. The password must be provided before any access to the keystore database.

The access control does not limit unauthorized users from reading and writing the file. GSKit relies on the OS for these protections, but access to sensitive data is effectively controlled because all sensitive data in the keystore is encrypted, all records hashed, and the index to all records is hashed. This ensures that any modification to the file is detectable. If tampering is detected, GSKit will deny access to the keystore (the behaviour is similar to receipt of an incorrect password).

What can I put in a CMS key database?

The CMS keystore contains X.509 certificates along with any associated private key information. For example when using the GSKCapiCmd tool to maintain a CMS keystore the following items may typically end up in the keystore:

- **CA Certificates.**

  Each valid X.509 certificate needs to be signed. Typically, this is done by a trusted Certificate Authority (CA). To validate a certificate signed by a CA, the public certificate for that CA must be in the CMS keystore. The GSKCapiCmd tool automatically populates a new CMS keystore with a number of CA certificates. If the CMS keystore does not already contain a required CA certificate, an administrator uses GSKCapiCmd to add it . Any valid X.509 certificate can be imported into a CMS keystore. For the import or add operation to succeed, the incoming certificate must be validated. For this reason, the certificate needed for the validation chain must already exist in the keystore.
- **Intermediate Certificates.**

  A valid certificate does not necessarily need to be signed by a CA. Instead it can be signed by what is known as an intermediate certificate that has itself been signed by a CA. If this is the case then both the CA certificate and the intermediate certificate must be in the CMS keystore in order to validate that certificate. This is known as a certificate validation chain. An administrator typically uses GSKCapiCmd to add their intermediate certificate. The keystore treats intermediate certificates in the same way as CA certificates.
- **Personal Certificates.**

  In a client-server relationship, the server may ask the client for its certificate. When acting as a client, GSKit attempts to use a personal certificate from the keystore to present to the server. Typically, GSKit picks the certificate that is marked as the default, or another certificate indicated by the client application. A personal certificate can also be used for signing other certificates. For example, an Intermediate certificate together with its private key may be in a CMS keystore and used to sign other certificates. An extracted version (without the private key) of the intermediate certificate is then added to other CMS keystores, to be used in a validation chain. An administrator typically uses GSKCapiCmd to add and extract their personal certificate(s). The final use for a personal certificate is for a server application. The server may present a personal certificate to the client during an SSL handshake.

GSKCapiCmd supports two certificate transfer formats (for commands such as add, import, extract, and so forth). These are referred to as ASCII and binary throughout this document. The default is Base64 encoded ASCII. Additional information about base64 encoding can be found in RFC 2045 and RFC 3548. The binary format is a binary dump of the DER encoded certificate structure. For additional information refer to ITU-T Rec. X.690 (2002) | ISO/IEC 8825-1:2002.

What is a label?

A label is a friendly name that an administrator can attach to a certificate that is contained in a CMS keystore. It is a convenient, human readable way to refer to a certificate. Only the UTF-8 character set is supported for keystore labels. Use of characters outside of UTF-8 is unsupported and behaviour is undefined.

How can I manipulate certificates in a CMS keystore?

Certificates in a CMS keystore are standard X.509 certificates. X.509 entities can be:

- imported (for personal certificates),
- added (for a certificate needed in a validation chain such as a CA or intermediate certificate),
- exported (from one CMS keystore to another - this takes the private key with it if one exists), or
- extracted (extract the public certificate – the private key is not extracted).

An administrator can also change the trust status of a certificate. When a certificates trust status is enabled, it is permitted to be involved in a certificate chain validation. If the certificates trust status is disabled then it cannot be used to validate any other certificates. For example, if certificate “ABC” is signed by the CA certificate “VeriSign CA”, but “VeriSign CA” is not marked as trusted, the validation of “ABC” will fail.

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 junior employees. It is expected that employees can be assured they are connecting to their company web site (not some fraudulent site pretending to be their company site). Employees use a customized web browser that can read CMS keystores 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 the following 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 to use 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 use a company wide intermediate certificate to sign all employee certificates.
- Employees must be able to validate the servers 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 in the default FIPS mode using the “Create a Key Database” command:

```
gsk8capicmd -keydb -create -db acme.kdb -pw offs64b
```

2. The administrator notices that the new keystore, while containing a number of CA certificates, does not contain the certificate of the CA 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:

   ```
   gsk8capicmd -cert -add -db acme.kdb -pw offs64b -label OurCA -file CACert.arm
   -format ascii
   ```

3. 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:

   ```
   gsk8capicmd -certreq -create -db acme.kdb –pw offs64b  -label OurIntermediate
   -dn "CN=acme.com,O=acme,C=US" -file certreq.arm -sigalg  sha1
   ```

4. 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:

   ```
   gsk8capicmd -cert -receive -file  signedCert.arm -db acme.kdb -pw offs64b
   ```

5. 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:

   ```
   gsk8capicmd -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 ACMEs intermediate certificate and the CA certificate that signed that intermediate certificate. The administrator now needs to use ACMEs 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:

   ```
   gsk8capicmd -cert -add -db Dave.kdb -pw Davepwd -label OurCA -file CACert.arm
   -format ascii
   ```

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

   ```
   gsk8capicmd -cert -extract -db acme.kdb -pw offs64b -label acmeCert -target
   acmeCert.arm
   ```

3. 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:

   ```
   gsk8capicmd -cert -add -db Dave.kdb -pw Davepwd -label acmeCert -file
   acmeCert.arm –format ascii
   ```

4. 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:

   ```
   gsk8capicmd -certreq -create -db Dave.kdb –pw Davepwd -label myCert -dn
   "CN=dave@acme.com,O=acme,C=US" -file DavesCertReq.arm -sigalg sha1
   ```

5. 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:

   ```
   gsk8capicmd -cert -sign -db acme.kdb -pw  offs64b -label acmeCert -target
   DavesCertReq.arm -expire 365 -file DavesSignedCert.arm -sigalg sha1
   ```

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

   ```
   gsk8capicmd -cert -receive -file  DavesSignedCert.arm -db Dave.kdb -pw Davepwd
   ```

7. 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:

   ```
   gsk8capicmd -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 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:

   ```
   gsk8capicmd -keydb -create -db acmeWebServer.kdb -pw ejed43dA
   ```

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

   ```
   gsk8capicmd -cert -add -db acmeWebServer.kdb -pw ejed43dA -label OurCA -file
   CACert.arm –format ascii
   ```

3. 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:

   ```
   gsk8capicmd -certreq -create -db acmeWebServer.kdb –pw ejed43dA -label
   OurServerCert -dn "CN=web.acme.com,O=acme,C=US" -file serverCertReq.arm
   -sigalg sha1
   ```

4. 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:

   ```
   gsk8capicmd -cert -receive -file signedServerCert.arm -db acmeWebServer.kdb
   -pw ejed43dA
   ```

5. 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:

```
gsk8capicmd -cert -setdefault -db acmeWebServer.kdb -pw ejed43dA -label
OurServerCert
```

6. 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?

Lets look 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 to use 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 labeled "OurServerCert" in the keystore acmeWebServer.kdb can 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 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`) contains both of these certificates. 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 use a company wide intermediate certificate 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 servers certificate to prove 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 certificates have been created. “OurServerCert” for use by the ACME web server and “OurIntermediate” for the administrator to use to sign employee certificates.

Appendix C. Troubleshooting

Use this troubleshooting information to help resolve any certificate management problems that you encounter.

Certificate not validating

GSKit validates all certificates when they are used. The validation process involves checking the digital signatures by following the certificate chain. The check usually starts with the *issuer* name and then continues back to a trusted root certificate. You must ensure that all the certificates in the chain are available in the keystore for this validation process to be successful.

A common error is to forget to install some certificates on the chain. There might be one or more certificates in the chain that are missing or invalid. For example, expired certificates.

You can use the following command to check a certificate chain that is in the keystore:

```
gskcapicmd -cert -validate -db key.kdb -pw **** -label endentity
```

You can use the following command to list a certificate chain from a selected label to the trusted root certificate:

```
gskcapicmd -cert -list -db key.kdb -pw **** -label endentity
```

You can find the name of the certificate issuer by listing the certificate details with the following command:

```
gskcapicmd -cert -details -db key.kdb -pw **** -label endentity
```

Case sensitivity

GSKit is fully case-sensitive on its certificate label lookups. Other tools such as iKeyman, OpenSSL for PKCS12 keystores, and PKCS11 vendor tools might be able to process keystores. These tools might not be case-sensitive. Some tools might even change the case of the key and certificate label names when displaying them. It is important to use the case-sensitive name exactly as displayed by **gskcapicmd**, which is the native GSKit utility for managing keystores.

You can list all certificate labels with the following command:

```
gskcapicmd -cert -list -db key.kdb -pw ****
```

Appendix D. Resources

ASN.1 Project

ASN.1 and OID project website http://asn1.elibel.tm.fr/en/standards/index.htm.

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). See the Internet RFC/STD/FYI/BCP Archives http://www.faqs.org/rfcs.

X.509

RFC 3280: Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL), obsoletes RFC 2459, April 2002. See RSA Security http://www.rsasecurity.com/rsalabs/.

PKCS#7

PKCS 7 v1.5: Cryptographic Message Syntax, RSA Laboratories, March 1998.

PKCS#10

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

PKCS#11

Cryptographic Token Interface Standard.

PKCS#12

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

Certificate Management System (CMS)

Appendix A, “CMS key databases,” on page 85 offers additional information concerning the format and use of a CMS keystore.

Appendix B, “A Simple Example,” on page 89 provides a simple example of how CMS keystores can be used to enable SSL communication between a server and client application.

Appendix E. FIPS mode operational considerations

Note: This section MUST be read in conjunction with the document " Using the Global Security Toolkit version 8 in FIPS-Approved Mode", which can be downloaded from the GSKit website https://w3.tap.ibm.com/w3ki/display/GSKit/Home

This section covers some of the FIPS considerations when using the gskcapicmd tool

FIPS considerations for the CAPICMD tool

The gsk8apicmd command line utility puts the underlaying ICC module into FIPS opertional mode whenever the “-FIPS” flag is passed on the command line. Should gsk8apicmd be unable to go into FIPS mode the operation will fail with the exit code value 152 (FIPS mode processing is unavailable). The solution for the user in this case is to perform the operation again but this time without the “-FIPS” flag. Assuming that gsk8apicmd could find another provider (eg. ICC in non-FIPS mode) then the operation will complete in the same manner as if gsk8apicmd had been in FIPS mode. Note: It is also possible for this utility to be configured to use a FIPS certified PKCS#11 module in which case the usage of ICC is not required.

Warning: When using the utility, ensure that the user or application only uses FIPS approved algorithms to be FIPS compliant. For example, do not use MD5 for hash and signature functions.

Appendix F. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. However, it is the user responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBMs future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBMs application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

Trademarks

IBM, the IBM logo, AIX, DB2®, IBMLink, Tivoli, Tivoli Enterprise Console®, and TME are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, the Adobe logo, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

IT Infrastructure Library is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Printed in USA