Red Hat Certificate System 7.3

Command-Line Tools Guide

7.3

ISBN: N/A
Publication date:

This book covers important, Certificate System-specific, command-line tools that you can use to create, remove, and manage subsystem instances and to create and manage keys and certificates.

Red Hat Certificate System 7.3: Command-Line Tools Guide

The GPG fingerprint of the security@redhat.com key is:

CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

1801 Varsity Drive
Raleigh, NC 27606-2072
USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park, NC 27709
USA

About This Guide

The *Certificate System Command-Line Tools Guide* describes the command-line tools and utilities bundled with Red Hat Certificate System and provides information such as command syntax and usage examples to help use these tools.

1. Who Should Read This Guide

This guide is intended for experienced system administrators who are planning to deploy the Certificate System. Certificate System agents should use the *Certificate System Agent's Guide* for information on how to perform agent tasks, such as handling certificate requests and revoking certificates.

2. Required Information

This guide assumes familiarity with the following concepts:

- Public-key cryptography and the Secure Sockets Layer (SSL) protocol
  - SSL cipher suites
  - The purpose of and major steps in the SSL handshake
- Intranet, extranet, Internet security, and the role of digital certificates in a secure enterprise, including the following topics:
  - Encryption and decryption
  - Public keys, private keys, and symmetric keys
  - Significance of key lengths
  - Digital signatures
  - Digital certificates
  - The role of digital certificates in a public-key infrastructure (PKI)
  - Certificate hierarchies

3. What Is in This Guide

This guide contains the following topics:

Chapter 1, Create and Remove Instance Tools	Describes the tools used to create and remove subsystem instances.
Chapter 2, Silent Installation	Describes the tool used for a silent instance creation.

Chapter 3, TokenInfo	Describes the utility which can be used to identify tokens on a machine, which shows whether the Certificate System can detect those tokens to use for a subsystem.
Chapter 4, SSLGet	Describes a tool used by the Certificate System to help configure and use security domains.
Chapter 5, AuditVerify	Describes how to use the tool used to verify signed audit logs.
Chapter 6, PIN Generator	Describes how to use the tool for generating unique PINs for end users and for populating their directory entries with PINs.
Chapter 7, ASCII to Binary	Describes how to use the tool for converting ASCII data to its binary equivalent.
Chapter 8, Binary to ASCII	Describes how to use the tool for converting binary data to its ASCII equivalent.
Chapter 9, Pretty Print Certificate	Describes how to use the tool for printing or viewing the contents of a certificate stored as ASCII base-64 encoded data in a human-readable form.
Chapter 10, Pretty Print CRL	Describes how to use the tool for printing or viewing the contents of a CRL stored as ASCII base-64 encoded data in a human-readable form.
Chapter 11, TKS Tool	Describes how to manipulate symmetric keys, including keys stored on tokens, the TKS master key, and related keys and databases.
Chapter 12, CMC Request	Describes how to construct a Certificate Management Messages over Cryptographic Message Syntax (CMC) request.
Chapter 13, CMC Enrollment	Describes how to sign a CMC certificate enrollment request with an agent's certificate.
Chapter 14, CMC Response	Describes how to parse a CMC response.
Chapter 15, CMC Revocation	Describes how to sign a CMC revocation request with an agent's certificate.
Chapter 16, CRMF Pop Request	Describes how to generate Certificate Request Message Format (CRMF) requests with proof of possession (POP).
Chapter 17, Extension Joiner	Describes how to use the tool for joining MIME-64 encoded formats of certificate extensions to create a single blob.
Chapter 18, Key Usage Extension	Describes how to generate a distinguished

	encoding rules (DER)-encoded Extended Key Usage extension.
Chapter 19, Issuer Alternative Name Extension	Describes how to generate an Issuer Alternative Name extension in base-64 encoding.
Chapter 20, Subject Alternative Name Extension	Describes how to generate a Subject Alternative Name extension in base-64 encoding.
Chapter 21, HTTP Client	Describes how to communicate with any HTTP/HTTPS server.
Chapter 22, OCSP Request	Describes how to verify certificate status by submitting Online Certificate Status Protocol (OCSP) requests to an instance of an OCSP subsystem.
Chapter 23, PKCS #10 Client	Describes how to generate a Public-Key Cryptography Standards (PKCS) #10 enrollment request.
Chapter 24, Bulk Issuance Tool	Describes how to send either a KEYGEN or CRMF enrollment request to the bulk issuance interface to create certificates automatically.
Chapter 25, Revocation Automation Utility	Describes how to automate user management scripts to revoke certificates.

Table 1. List of Contents

4. Common Tool Information

All of the tools in this guide are located in the `/usr/bin` directory, except for the Silent Install tool which is downloaded separately and installed to any directory. These tools can be run from any location without specifying the tool location.

5. Additional Reading

The documentation for the Certificate System also contains the following guides:

- *Certificate System Administrator's Guide* explains all administrative functions for the Certificate System, such as adding users, creating and renewing certificates, managing smart cards, publishing CRLs, and modifying subsystem settings like port numbers.
- *Certificate System Agent's Guide* details how to perform agent operations for the CA, DRM, OCSP, and TPS subsystems through the Certificate System agent services interfaces.

- *Certificate System Enterprise Security Client Guide* explains how to install, configure, and use the Enterprise Security Client, the user client application for managing smart cards, user certificates, and user keys.
- *Certificate System Migration Guide* provides detailed migration information for migrating all parts and subsystems of previous versions of Certificate System to Red Hat Certificate System 7.3.

Additional Certificate System information is provided in the Certificate System SDK, an online reference to HTTP interfaces, javadocs, samples, and tutorials related to Certificate System; a downloadable zip file of this material is available for user interaction with the tutorials.

For the latest information about Certificate System, including current release notes, complete product documentation, technical notes, and deployment information, see the Red Hat documentation page:

```
http://www.redhat.com/docs/manuals/cert-system/
```

6. Examples and Formatting

All of the examples for Red Hat Certificate System commands, file locations, and other usage are given for Red Hat Enterprise Linux 5 systems. Be certain to use the appropriate commands and files for your platform. For example:

To start the Red Hat Directory Server:

```
service dir-server start
```

Example 1. Example Command

Certain words are represented in different fonts, styles, and weights. Different character formatting is used to indicate the function or purpose of the phrase being highlighted.

Formatting Style	Purpose
`Monospace font`	Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt.
`Monospace with a background`	This type of formatting is used for anything entered or returned in a command prompt.
Italicized text	Any text which is italicized is a variable, such as *instance_name* or *hostname*. Occasionally, this is also used to

Formatting Style	Purpose
	emphasize a new term or other phrase.
Bolded text	Most phrases which are in bold are application names, such as **Cygwin**, or are fields or options in a user interface, such as a **User Name Here:** field or **Save** button.

Other formatting styles draw attention to important text.

NOTE

A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

TIP

A tip is typically an alternative way of performing a task.

IMPORTANT

Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.

CAUTION and WARNING

A caution indicates an act that would violate your support agreement.

A warning indicates potential data loss, as may happen when tuning hardware for maximum performance.

7. Giving Feedback

If there is any error in this *Command-Line Tools Guide* or there is any way to improve the documentation, please let us know. Bugs can be filed against the documentation for Red Hat Certificate System through Bugzilla, *http://bugzilla.redhat.com/bugzilla*. Make the bug report as specific as possible, so we can be more effective in correcting any issues:

- Select the Red Hat Certificate System product.
- Set the component to `Doc - cli-tools-guide`.
- Set the version number to 7.3.
- For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.

  For enhancements, put in what information needs to be added and why.
- Give a clear title for the bug. For example, `"Incorrect command example for setup script options"` is better than `"Bad example"`.

We appreciate receiving any feedback — requests for new sections, corrections, improvements, enhancements, even new ways of delivering the documentation or new styles of docs. You are welcome to contact Red Hat Content Services directly at *mailto:docs@redhat.com*.

8. Revision History

Revision History

Revision 7.3.2	Tuesday, August 5, 2008	Ella Deon Lackey<dlackey@redhat.com>

Edited setpin options per Bugzilla #224748 and Bugzilla #224930

Revision 7.3.0-1	Fri May 25 2007	DavidO'Brien<david.obrien@redhat.com>

Removed ecc as key type option
Added pkisilent tool syntax for RA

Revision 7.3.0-0	Thu May 24 2007	DavidO'Brien<david.obrien@redhat.com>

Added revision history.
Updated version for 7.3.0
Updates to presentation and layout for commands.

Create and Remove Instance Tools

The Certificate System includes two tools to create and remove subsystem instances, `pkicreate` and `pkiremove`.

NOTE

The `pkicreate` tool does not install the Certificate System system; this is done through installing the packages or running the Red Hat Enterprise Linux `up2date` command. This tool creates new instances after the default subsystems have been installed.

Likewise, the `pkiremove` utility does not uninstall the Certificate System subsystem; it removes a single instance.

1. pkicreate

The `pkicreate` tool creates instances of Certificate System subsystems and does a minimal configuration of the new instance, such as setting the configuration directory and port numbers. Further configuration is done through the HTML configuration page, as with configuring the default instances.

The following sections explain the syntax and usage of the `pkicreate` tool.

1.1. Syntax

This tool has the following syntax:

```
pkicreate -pki_instance_root=/directory/path -subsystem_type=type
 -pki_instance_name=instance_ID [-secure_port=SSLport]
 [-unsecure_port=port] -tomcat_server_port=port
 -user=user_name -group=group_name [-verbose] [-help]
```

NOTE

The `pkicreate` tool also accepts an environment variable, `DONT_RUN_PKICREATE`; if this is set, the `pkicreate` utility is prevented from doing anything. When the `DONT_RUN_PKICREATE` variable is set before installing the default subsystem instance (before running the `rhpki-install` script), this allows the default instance to be installed in a user-defined location instead of the default location.

Parameter	Description
`pki_instance_root`	Gives the full path to the new instance configuration directory.
`subsystem_type`	Gives the type of subsystem being created. The possible values are as follows: • `ca`, for a Certificate Authority • `ra`, for a Registration Authority • `kra`, for a DRM • `ocsp`, for an OCSP • `tks`, for a TKS • `tps`, for a TPS
`pki_instance_name`	Gives the name of the new instance.The name must be unique within the security domain. Even cloned subsystems must have different instance names for cloning to succeed.
`secure_port`	*Optional.* Sets the SSL port number. If this is not set, the number is randomly generated.
`unsecure_port`	*Optional.* Sets the regular port number. If this is not set, the number is randomly generated.
`tomcat_server_port`	Sets the port number for the Tomcat web server. This option must be set for CA, OCSP, TKS, and DRM instances. `tomcat_server_port` is not used when creating a TPS instance since it does not use a Tomcat web server.
`user`	Sets the user as which the Certificate System instance will run. This option must be set.
`group`	Sets the group as which the Certificate System instance will run. This option must be set.
`verbose`	*Optional.* Runs the new instance creation in verbose mode.
`help`	Shows the help information.

Table 1.1.

1.2. Usage

In the following example, the `pkicreate` is used to create a new DRM instance running on ports `10543` and `10180`, named `rhpki-drm2`, in the `/var/lib/rhpki-drm2` directory.

```
pkicreate -pki_instance_root=/var/lib -subsystem_type=kra
-pki_instance_name=rhpki-drm2 -secure_port=10543 \
        -unsecure_port=10180 -tomcat_server_port=1802 -user=pkiuser
-group=pkigroup -verbose
```

To keep the `pkicreate` script from creating a new instance when it is run, set the `DONT_RUN_PKICREATE` environment variable to 1.

```
export DONT_RUN_PKICREATE=1
```

2. pkiremove

The `pkiremove` tool removes subsystem instances. This tool removes the single subsystem instance specified; it does not uninstall the Certificate System packages.

2.1. Syntax

This tool has the following syntax:

```
pkiremove -pki_instance_root=/directory/path -pki_instance_name=instance_ID
```

Parameter	Description
`pki_instance_root`	Gives the full path to the instance configuration directory.
`pki_instance_name`	Gives the name of the instance.

Table 1.2.

2.2. Usage

The following example removes a DRM instance named `rhpki-drm2` which was installed in the `/var/lib/rhpki-drm2` directory.

```
        pkiremove -pki_instance_root=/var/lib -pki_instance_name=rhpki-drm2
```

Silent Installation

The Certificate System includes a tool, `pkisilent`, which can completely create and configure an instance in a single step. Normally, adding instances requires running the `pkicreate` utility to create the instance and then accessing the subsystem HTML page to complete the configuration. The `pkisilent` utility creates and configures the instance in a single step. The `pkisilent` tool must be downloaded independently. It is available through the **Red Hat Certificate System 7.3** Red Hat Network channel.

NOTE

Run this tool on a system which already has a subsystem installed, since this tool depends on having libraries, JRE, and core jar files already installed.

Two files are installed for the `pkisilent` tool:

- `pkisilent`, the Perl wrapper script.
- `pkisilent.jar`, the jar files containing the Java™ classes to perform a silent installation.

The utility can be downloaded and saved to any location and is then executed locally.

1. Syntax

This tool has the following syntax for a CA:

```
perl pkisilent ConfigureCA -cs_hostname hostname
                           -cs_port SSLport
                           -client_certdb_dir certDBdir
                           -client_certdb_pwd password
                           -preop_pin preoppin
                           -domain_name domain_name
                           -admin_user adminUID
                           -admin_email admin@email
                           -admin_password password
                           -agent_name agentName
                           -agent_key_size keySize
                           -agent_key_type keyType
                           -agent_cert_subject cert_subject_name
                           -ldap_host hostname
                           -ldap_port port
                           -bind_dn bindDN
                           -bind_password password
                           -base_dn search_base_DN
                           -db_name dbName
                           -key_size keySize
                           -key_type keyType
```

```
                            -token_name HSM_name
                            -token_pwd HSM_password
                            -save_p12 export-p12-file
                            -backup_pwd password
```

This tool has the following syntax for the RA subsystem:

```
perl pkisilent ConfigureRA
                         -help,-?     displays help information
                         -cs_hostname CS Hostname
                         -cs_port CS SSL port
                         -sd_hostname Security Domain Hostname
                         -sd_ssl_port Security Domain SSL port
                         -sd_admin_name Security Domain username
                         -sd_admin_password Security Domain password
                         -ca_hostname CA Hostname
                         -ca_port CA non SSL port
                         -ca_ssl_port CA SSL port
                         -client_certdb_dir Client CertDB dir
                         -client_certdb_pwd client certdb password
                         -preop_pin pre op pin
                         -domain_name domain name
                         -admin_user Admin User Name
                         -admin_email Admin email
                         -admin_password Admin password
                         -agent_name Agent Cert Nickname
                         -token_name HSM/Software Token name
                         -token_pwd HSM/Software Token password
                         -key_size Key Size
                         -key_type Key type [rsa]
                         -agent_key_size Agent Cert Key Size
                         -agent_key_type Agent cert Key type [rsa]
                         -agent_cert_subject Agent cert Subject
                         -ra_subsystem_cert_subject_name RA subsystem cert
subject name
                         -ra_server_cert_subject_name RA server cert subject
name
                         -subsystem_name RA subsystem name
```

This tool has the following syntax for the DRM, OCSP, and TKS subsystems:

```
perl pkisilent Configure subsystemType -cs_hostname hostname
                            -cs_port SSLport
                            -ca_hostname hostname
                            -ca_port port
                            -ca_ssl_port SSLport
                            -ca_agent_name agentName
                            -ca_agent_password password
                            -client_certdb_dir certDBdir
                            -client_certdb_pwd password
                            -preop_pin preoppin
                            -domain_name domain_name
```

```
                            -admin_user adminUID
                            -admin_email admin@email
                            -admin_password password
                            -agent_name agentName
                            -ldap_host hostname
                            -ldap_port port
                            -bind_dn bindDN
                            -bind_password password
                            -base_dn search_base_DN
                            -db_name dbName
                            -key_size keySize
                            -key_type keyType
                            -agent_key_size keySize
                            -agent_key_type keyType
                            -agent_cert_subject cert_subject_name
                            -backup_pwd password
```

This tool has the following syntax for the TPS subsystem:

```
perl pkisilent ConfigureTPS -cs_hostname hostname
                            -cs_port SSLport
                            -ca_hostname hostname
                            -ca_port port
                            -ca_ssl_port SSLport
                            -ca_agent_name agentName
                            -ca_agent_password password
                            -client_certdb_dir certDBdir
                            -client_certdb_pwd password
                            -preop_pin preoppin
                            -domain_name domain_name
                            -admin_user adminUID
                            -admin_email admin@email
                            -admin_password password
                            -agent_name agentName
                            -ldap_host hostname
                            -ldap_port port
                            -bind_dn bindDN
                            -bind_password password
                            -base_dn search_base_DN
                            -db_name dbName
                            -key_size keySize
                            -key_type keyType
                            -agent_key_size keySize
                            -agent_key_type keyType
                            -agent_cert_subject cert_subject_name
                            -ldap_auth_host ldap_auth_host
                            -ldap_auth_port ldap_auth_port
                            -ldap_auth_base_dn ldap_auth_base_dn
```

Java™ Class Name	Subsystem
`ConfigureCA`	For the CA.
`ConfigureRA`	For the RA.

Java™ Class Name	Subsystem
`ConfigureDRM`	For the DRM.
`ConfigureOCSP`	For the OCSP.
`ConfigureTKS`	For the TKS.
`ConfigureTPS`	For the TPS.

Table 2.1. Subsystem Java™ Classes for pkisilent

NOTE

The `ConfigureCA` script is used to create a security domain or to add the new CA to an existing domain. The other scripts only add the subsystem to an existing security domain.

Parameter	Description
`cs_hostname`	The hostname for the Certificate System machine.
`cs_port`	The SSL port number of the Certificate System.
`ca_hostname`	The hostname for the CA subsystem which will issue the certificates for the DRM, OCSP, TKS, or TPS subsystem.
`ca_port`	The non-SSL port number of the CA.
`ca_ssl_port`	The SSL port number of the CA.
`ca_agent_name`	The UID of the CA agent.
`ca_agent_password`	The password of the CA agent.
`client_certdb_dir`	The directory for the subsystem certificate databases.
`client_certdb_pwd`	The password to protect the certificate database.
`preop_pin`	The preoperation PIN number used for the initial configuration.
`domain_name`	The name of the security domain to which the subsystem will be added.
`admin_user`	The new admin user for the new subsystem.
`admin_email`	The email address of the admin user.
`admin_password`	The password for the admin user.

Parameter	Description
`agent_name`	The new agent for the new subsystem.
`agent_key_size`	The key size to use for generating the agent certificate and key pair.
`agent_key_type`	The key type to use for generating the agent certificate and key pair.
`agent_cert_subject`	The subject name for the agent certificate.
`ldap_host`	The hostname of the Directory Server machine.
`ldap_port`	The non-SSL port of the Directory Server.
`bind_dn`	The bind DN which will access the Directory Server; this is normally the Directory Manager ID.
`bind_password`	The bind DN password.
`base_dn`	The entry DN under which to create all of the subsystem entries.
`db_name`	The database name.
`key_size`	The size of the key to generate. The recommended size for an RSA key is 1024 bits for regular operations and 2048 bits for sensitive operations.
`key_type`	The type of key to generate; the only option is RSA.
`save_p12`	Sets whether to export the keys and certificate information to a backup PKCS #12 file. `true` backs up the information; `false` does not back up the information. *Only for the CA subsystem.*
`backup_pwd`	The password to protect the PKCS #12 backup file containing the subsystem keys and certificates. *Not for use with TPS installation.*
`token_name`	Gives the name of the HSM token used to store the subsystem certificates. *Only for the CA subsystem.*
`token_password`	Gives the password for the HSM. *Only for the CA subsystem.*
`ldap_auth_host`	Gives the hostname of the LDAP directory database to use for the TPS subsystem token database. *Only for the TPS subsystem.*
`ldap_auth_port`	Gives the port number of the LDAP directory

Parameter	Description
	database to use for the TPS subsystem token database. *Only for the TPS subsystem.*
`ldap_auth_base_dn`	Gives the base DN in the LDAP directory tree of the TPS token database under which to create token entries. *Only for the TPS subsystem.*

Table 2.2. Parameters for pkisilent

2. Usage

The options are slightly different between the subsystems; all subsystems except for the CA subsystem require extra options specifying the Certificate Authority to which to submit the certificate requests.

This silent installation script example installs a CA subsystem:

```
perl pkisilent ConfigureCA -cs_hostname localhost  -cs_port 9543
 -client_certdb_dir /tmp/ -client_certdb_pwd password -preop_pin
sYY8er834FG9793fsef7et5
 -domain_name "testca" -admin_user admin -admin_email "admin@example.com"
 -admin_password password -agent_name "rhpki-ca2 agent" -agent_key_size 2048
 -agent_key_type rsa -agent_cert_subject "ca agent cert" -ldap_host server
 -ldap_port 389 -bind_dn "cn=directory manager" -bind_password password
 -base_dn "o=rhpki-ca2" -db_name "rhpki-ca2" -key_size 2048
 -key_type rsa -save_p12 true -backup_pwd password
```

This silent installation script example installs a TKS subsystem; this script has extra options to point to the CA server:

```
perl pkisilent ConfigureTKS -cs_hostname localhost  -cs_port 13543
 -ca_hostname server.example.com -ca_port 9080 -ca_ssl_port 9443
 -ca_agent_name agent -ca_agent_password password
 -client_certdb_dir /tmp/ -client_certdb_pwd password -preop_pin
fS44I6SASGF34FD76WKJHIW4
 -domain_name "testca" -admin_user admin -admin_email "admin@example.com"
 -admin_password password -agent_name "rhpki-tks2 agent" -ldap_host server
 -ldap_port 389 -bind_dn "cn=directory manager" -bind_password password
 -base_dn "o=rhpki-tks2" -db_name "rhpki-tks2" -key_size 2048
 -key_type rsa -agent_key_size 2048  -agent_key_type rsa
 -agent_cert_subject "tks agent cert" -backup_pwd password
```

This silent installation script example installs a TPS subsystem; this script has extra options to point to the LDAP authentication database used for storing token information:

```
perl pkisilent ConfigureTPS -cs_hostname localhost  -cs_port 7988
 -ca_hostname server.example.com -ca_port 9080 -ca_ssl_port 9443
 -ca_agent_name agent -ca_agent_password password
 -client_certdb_dir /tmp/ -client_certdb_pwd password -preop_pin
fS44I6SASGF34FD76WKJHIW4
 -domain_name "testca" -admin_user admin -admin_email "admin@example.com"
 -admin_password password -agent_name "rhpki-tks2 agent" -ldap_host server
 -ldap_port 389 -bind_dn "cn=directory manager" -bind_password password
 -base_dn "o=rhpki-tks2" -db_name "rhpki-tks2" -key_size 2048
 -key_type rsa -agent_key_size 2048  -agent_key_type rsa
 -agent_cert_subject "tps agent cert" -ldap_auth_host server
 -ldap_auth_port 389 -ldap_auth_base_dn "o=TPS DB,dc=example,dc=com"
```

TokenInfo

This tool is used to determine which external hardware tokens are visible to the Certificate System subsystem. This can be used to diagnose whether problems using tokens are related to the Certificate System being unable to detect it.

1. Syntax

The `TokenInfo` tool has the following syntax:

```
TokenInfo /directory/alias
```

Option	Description
/directory/alias	Specifies the path and file to the certificate and key database directory; for example, `/var/lib/rhpki-ca/alias/`.

Table 3.1.

SSLGet

This tool is similar to the the `wget` command, which downloads files over HTTP. `sslget` supports client authentication using NSS libraries. The configuration wizard uses this utility to retrieve security domain information from the CA.

1. Syntax

The `sslget` tool has the following syntax:

```
sslget [-e profile information] -n rsa_nickname [-p password | -w pwfile]
 [-d dbdir] [-v] [-V] -r url hostname[:port]
```

Option	Description
`e`	*Optional.* Submits information through a subsystem form by specifying the form name and the form fields. For example, this can be used to submit certificate enrollments through a certificate profile.
`n`	Gives the CA certificate nickname.
`p`	Gives the certificate database password. Not used if the `-w` option is used.
`w`	*Optional.* Gives the password file path and name. Not used if the `-p` option is used.
`d`	*Optional.* Gives the path to the security databases.
`v`	*Optional.* Sets the operation in verbose mode.
`V`	*Optional.* Gives the version of the `sslget` tool.
`r`	Gives the URL of the site or server from which to download the information.
hostname	Gives the hostname of the server to which to send the request.
port	*Optional.* Gives the port number of the server.

Table 4.1.

2. Usage

It is possible to use `sslget` to submit information securely to Certificate System subsystems.

For example, to submit a certificate request through a certificate profile enrollment for to a CA, the command is as follows:

```
sslget -e
"profileId=caInternalAuthServerCert&cert_request_type=pkcs10
&requestor_name=TPS-server.example.com-7889
&cert_request=MIIBGTCBxAIBADBfMSgwJgYDVQQKEx8yMDA2MTEwNngxMi
BTZmJheSBSZWRoYXQgRG9tYWluMRIwEAYDVQQLEwlyaHBraS10cHMxHzAdBgNVBA
MTFndhdGVyLnNmYmF5LnJlZGhhdC5jb20wXDANBgkqhkiG9w0BAQEFAANLADBIAk
EAsMcYjKD2cDJOeKjhuAiyaC0YVh8hUzfcrf7ZJlVyROQx1pQrHiHmBQbcCdQxNz
YK7rxWiR62BPDR4dHtQzj8RwIDAQABoAAwDQYJKoZIhvcNAQEEBQADQQAKpuTYGP
%2BI1k50tjn6enPV6j%2B2lFFjrYNwlYWBe4qYhm3WoA0tIuplNLpzP0vw6ttIMZ
kpE8rcfAeMG10doUpp
&xmlOutput=true&sessionID=-4771521138734965265
&auth_hostname=server.example.com&auth_port=9443"
 -d "/var/lib/rhpki-tps/alias" -p "password123" -v -n "Server-Cert
cert-rhpki-tps"
 -r "/ca/ee/ca/profileSubmit" server.example.com:9443
```

AuditVerify

1. About the AuditVerify Tool

The `AuditVerify` tool is used to verify that signed audit logs were signed with the private signing key and that the audit logs have not been compromised.

Auditors can verify the authenticity of signed audit logs using the `AuditVerify` tool. This tool uses the public key of the signed audit log signing certificate to verify the digital signatures embedded in a signed audit log file. The tool response indicates either that the signed audit log was successfully verified or that the signed audit log was not successfully verified. An unsuccessful verification warns the auditor that the signature failed to verify, indicating the log file may have been tampered with (compromised).

2. Setting up the Auditor's Database

`AuditVerify` needs access to a set of security databases containing the signed audit log signing certificate and its chain of issuing certificates. One of the CA certificates in the issuance chain must be marked as trusted in the database.

The auditor should import the audit signing certificate into certificate and key databases before running `AuditVerify`. The auditor should not use the security databases of the Certificate System instance that generated the signed audit log files. If there are no readily accessible certificate and key database, the auditor must create a set of certificate and key databases and import the signed audit log signing certificate chain.

To create the security databases and import the certificate chain, do the following:

1. Create the security database directory in the filesystem.

```
mkdir /var/lib/instance_ID/logs/signedAudit/dbdir
```

2. Use the `certutil` tool to create an empty set of certificate databases.

```
certutil -d /var/lib/instance_ID/logs/signedAudit/dbdir -N
```

3. Import the CA certificate and log signing certificate into the databases, marking the CA certificate as trusted. The certificates can be obtained from the CA in ASCII format.

   If the CA certificate is in a file called `cacert.txt` and the log signing certificate is in a file called `logsigncert.txt`, both in the Certificate System `alias/` directory, then the `certutil` is used to set the trust for the new audit security database directory pointing to those files, as follows:

```

```

```
certutil -d /var/lib/instance_ID/logs/signedAudit/dbdir -A -n "CA
Certificate" -t \
        "CT,CT,CT" -a -i /var/lib/instance_ID/alias/cacert.txtcertutil -d \
        /var/lib/instance_ID/logs/signedAudit/dbdir -A -n "Log Signing
Certificate" -a -i \
        /var/lib/instance_ID/alias/logsigncert.txt
```

3. Syntax

The `AuditVerify` tool has the following syntax:

```
AuditVerify -d dbdir -n signing_certificate_nickname -a logListFile [-P
cert/key_db_prefix] [-v]
```

Option	Description
d	Specifies the directory containing the security databases with the imported audit log signing certificate.
n	Gives the nickname of the certificate used to sign the log files. The nickname is whatever was used when the log signing certificate was imported into that database.

a	Specifies the text file containing a comma separated list (in chronological order) of the signed audit logs to be verified. The contents of the *logListFile* are the full paths to the audit logs. For example:
	`/var/lib/rhpki-ca/logs/signedAudit/ca_cert-ca_audit, \`
	`/var/lib/rhpki-ca/logs/signedAudit/ca_cert-ca_audit. \`
	`/var/lib/rhpki-ca/logs/signedAudit/ca_cert-ca_audit.`

P	*Optional.* The prefix to prepend to the certificate and key database filenames. If used, a value of empty quotation marks (“”) should be specified for this argument, since the auditor is using separate certificate and key databases from the Certificate System instance and it is unlikely that the prefix

Option	Description
	should be prepended to the new audit security database files.
v	*Optional.* Specifies verbose output.

Table 5.1.

4. Return Values

When `AuditVerify` is used, one of the following codes is returned:

Return Value	Description
0	Indicates that the signed audit log has been successfully verified.
1	Indicates that there was an error while the tool was running.
2	Indicates that one or more invalid signatures were found in the specified file, meaning that at least one of the log files could not be verified.

Table 5.2.

5. Usage

After a separate audit database directory has been configured, do the following:

1. Create a text file containing a comma-separated list of the log files to be verified. The name of this file is referenced in the `AuditVerify` command.

   For example, this file could be `logListFile` in the `/etc/audit` directory. The contents are the comma-separated list of audit logs to be verified, such as "`auditlog.1213`, `auditlog.1214`, `auditlog.1215`."

2. If the audit databases do not contain prefixes and are located in the user home directory, such as `/usr/home/smith/.redhat`, and the signing certificate nickname is “auditsigningcert”, the `AuditVerify` command is run as follows:

```
AuditVerify -d /usr/home/smith/.redhat -n auditsigningcert -a
/etc/audit/logListFile -P "" -v
```

PIN Generator

For the Certificate System to use the `UidPwdPinDirAuth` authentication plug-in module, the authentication directory must contain unique PINs for each end entity which will be issued a certificate. The Certificate System provides a tool, the *PIN Generator*, which generates unique PINs for end-entity entries in an LDAP directory. The tool stores these PINs as hashed values in the same directory against the corresponding user entries. It also copies the PINs to a text file so that the PINs can be sent to the end entities.

1. The setpin Command

This chapter describes the syntax and arguments of the `setpin` tool and the expected responses. For information on generating and storing PINs in the user authentication directory, see the *Certificate System Administration Guide*.

1.1. Editing the setpin.conf Configuration File

The `setpin` tool can use a configuration file, `setpin.conf`, to store some of its required options. Before running `setpin`, modify this file to reflect the directory information, and set the `setpin` tool to use this file by doing the following:

1. Open the `setpin.conf` file.

```
cd /usr/lib/rhpki/native-tools
vi setpin.conf
```

2. Edit the directory parameters in the file to match the directory installation information.

```
#------- Enter the hostname of the LDAP server
host=localhost

#------- Enter the port number of the LDAP server
port=389

#------- Enter the DN of the Directory Manager user
binddn=CN=Directory Manager

#------- Enter the password for the Directory manager user
bindpw=

#    Enter the DN and password for the new pin manager user
pinmanager=cn=pinmanager,o=example.com
pinmanagerpwd=

#    Enter the base over which this user has the power
#    to remove pins
basedn=ou=people,o=example.com
```

```
## This line switches setpin into setup mode.
## Please do not change it.
setup=yes
```

3. Run `setpin`, and set the option file to `setpin.conf`.

```
setpin optfile=/usr/lib/rhpki/native-tools/setpin.conf
```

1.2. Syntax

The `setpin` has the following syntax:

```
setpin host=host_name [port=port_number] binddn=user_id
 [bindpw=bind_password] filter="LDAP_search_filter" [basedn=LDAP_base_DN]
 [length=PIN_length | minlength=minimum_PIN_length |
maxlength=maximum_PIN_length]
 [gen=character_type] [case=upperonly] [hash=algorithm]
 [saltattribute=LDAP_attribute_to_use_for_salt_creation] [input=file_name]
 [output=file_name] [write] [clobber] [testpingen=count]
 [debug] [optfile=file_name] [setup [pinmanager=pinmanager_user]
 [pinmanagerpwd=pinmanager_password]]
```

Option	Description
`host`	*Required*. Specifies the LDAP directory to which to connect.
`port`	Specifies the LDAP directory port to which to bind. The default port number is the default LDAP port, `389`.
`binddn`	*Required.* Specifies the user as whom the PIN Generator binds to the LDAP directory. This user account must have read/write access to the directory.
`bindpw`	Gives the password for the user ID set in the `binddn` option. If the bind password is not given at the command line, the tool prompts for it.
`filter`	*Required.* Sets the search filter for those DNs in the directory for which the tool should generate PINs.
`basedn`	Specifies the base DN under which to search for DNs. If this argument is not specified, the

Option	Description
	filter searches from the root.
`length`	Specifies the exact number a PIN must contain; the default is 6. Do not use with `minlength` or `maxlength`.
`minlength`	Sets the minimum length of the generated PINs. If used with `maxlength`, this sets the lower end of the range of the PIN length. Do not use with `length`.
`maxlength`	Sets the maximum length of the generated PINs. If used with `minlength`, this sets the upper end of the range of the PIN length. Do not use with `length`.
`gen`	Specifies the character type for PINs. The characters in the password can be constructed out of alphabetic characters (`RNG-alpha`), alphanumeric characters (`RNG-alphanum`), or any printable ASCII characters (`printableascii`).
`case`	Restricts the character cases to uppercase only; otherwise, the case is mixed. Restricting alphabetic characters to uppercase reduces the overall combinations for the password space significantly. Use `case` with `gen`.
`hash`	Specifies the message digest algorithm with which to hash the PINs before storing them in the authentication directory. If SHA-1 or MD5 is used, set an output file for storing PINs in plain text. A user needs the PINs in plain text for delivering them to end entities. The default is `sha1`, which produces a 160-bit message digest. `md5` produces a 128-bit message digest. `none` does not hash the PINs.
`saltattribute`	Specifies the LDAP attribute to use for salt creation. If an attribute is set, the tool integrates the value of the attribute with each PIN and hashes the resulting string with the hash routine. The default is to use the entry DN. For details, refer to *Section 2.3, "How PINs Are Stored in the Directory"*.
`input`	Specifies the file that contains the list of DNs to process. If this is used, the tool compares the filtered DNs to the ones in the input file

Option	Description
	and generates PINs for only those DNs .
`output`	Specifies the absolute path to the file to write the PINs as `setpin` generates them. If a file is not set, then the output is written to the standard output. Regardless of whether an output file is set, all error messages are directed to the standard error.
`write`	Sets whether the tool should write PINs to the directory. If specified, the PINs are written to the directory as they are generated. Otherwise, the tool does not make any changes to the directory. Do not write PINs to the directory if the PINs are to be checked. The PINs can be viewed in the output file to make sure that they are being assigned to the correct users and that they conform to the length and character restrictions. For more information, see *Section 2.2, "Output File"*.
`clobber`	Overwrites pre-existing PINs, if any, associated with a DN. If this option is not used, any existing PINs are left in the directory.
`testpingen`	Tests the PIN-generation mode. *count* sets the total number of PINs to generate for testing.
`debug`	Writes debugging information to the standard error. If `debug=attrs` is specified, the tool writes more detailed information about each entry in the directory.
`optfile`	Sets the tool to read options, one per line, from a file. This allows all arguments to be put in a file, instead of typing them at the command line. One configuration file, `setpin.conf`, is located in the `/usr/lib/rhpki/native-tools` directory.
`setup`	Switches to setup mode, which allows the tool to add to the directory schema.
`pinmanager`	Specifies the PIN manager user that has permission to remove the PIN for the `basedn` specified. Used with the `setup` option.
`pinmanagerpwd`	Gives the password for the PIN manager user. Used with the `setup` option.

Table 6.1.

1.3. Usage

The following command generates PINs for all entries that have the `CN` attribute in their distinguished name in an LDAP directory named `csldap` listening on port `19000`. The PIN Generator binds to the directory as `Directory Manager` and starts searching the directory from the base DN `dn=o=example.com` in the directory tree. Any existing PINs are overwritten with the new ones.

```
setpin host=csldap port=19000 binddn="CN=directory manager" bindpw=password
filter="(cn=*)" \
        basedn="o=example.com" clobber write
```

2. How setpin Works

The PIN Generator generates PINs for user entries in an LDAP directory and updates the directory with these PINs. To run the `setpin` command, the following five options are required:

- The host name (`host`) and port number (`port`) of the LDAP server
- The bind DN (`binddn`) and password (`bindpw`)
- An LDAP filter (`filter`) for filtering out the user entries that require PINs

The `setpin` command looks like the following:

```
setpin host=csldap port=19000 binddn="CN=Directory Manager" bindpw=redhat
filter="(ou=employees)" \
        basedn="o=example.com"
```

This example queries the directory for all the entries in the `employees` organizational unit (`ou`). For each entry matching the filter, information is printed out to standard error and to the standard output.

Note

Because the PIN Generator makes a lot of changes to the directory, it is important to use the correct filter, or the wrong entries are modified. Using the `write` option is a safeguard because no changes are made to the directory

unless that option is used. This allows the PINs to be verified before any entries are modified.

The information can be written to a different output file by using the `output` option; see *Section 2.2, "Output File"* for more information. The entries returned by the LDAP search filter can be further restricted by using an ASCII input file which lists the entry DNs; only entries matching those in the file are updated. The input file is set with the `input` option. The input file is not a substitute for the LDAP directory entries; the filter attribute must still be provided. For more information about the input file, refer to *Section 2.1, "Input File"*. *Figure 6.1, "Using an Input and Output File When Generating PINs"* shows how the input and output files work with the `setpin` tool.

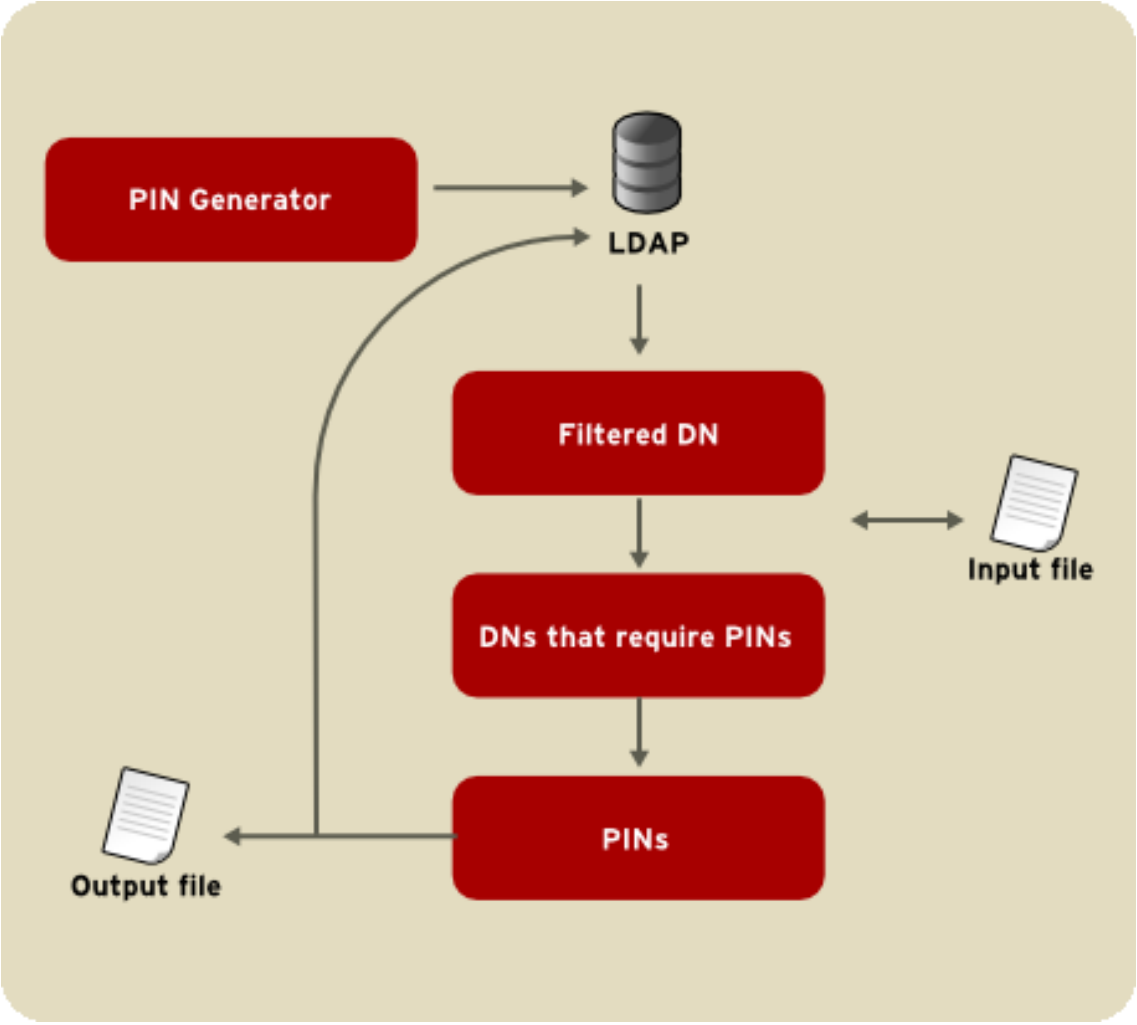

Figure 6.1. Using an Input and Output File When Generating PINs

The output file contains the entry and PIN information from running `setpin`, as shown in the following example:

```
Processing: cn=QA Managers,ou=employees,o=example.com
Adding new pin/password
dn:cn=QA Managers,ou=employees,o=example.com
pin:lDWynV
status:notwritten

Processing: cn=PD Managers,ou=employees,o=example.com
Adding new pin/password
dn:cn=PD Managers,ou=employees,o=example.com
pin:G69uV7
status:notwritten
```

The output also contains the status of each entry in the directory. The status values are listed in *Table 6.2, "PIN Generator Status "*.

Exit Code	Description
`notwritten`	The PINs were not written to the directory because the `write` option was not used.
`writefailed`	The tool tried to modify the directory, but the write operation was unsuccessful.
`added`	The tool added the new PIN to the directory successfully.
`replaced`	The tool replaced an old PIN with a new one; this means the `clobber` option was used.
`notreplaced`	The tool did not replace the old PIN with a new one; this means the `clobber` option was not used.

Table 6.2. PIN Generator Status

If a PIN already exists for a user, it is not changed if the `setpin` command is run a second time. This allows new PINs to be created for new users without overwriting PINs for users who have already received a PIN. To overwrite a PIN, use the `clobber` option.

After making sure that the filter is matching the right users, run the `setpin` command again with the `write` option and with `output` set to the name of the file to capture the unhoused PINs. For details about the output file, refer to *Section 2.2, "Output File"*.

2.1. Input File

The PIN Generator can receive a list of DNs to modify in a text file specified by the `input` argument. If an input file is specified, then the tool compares the DNs returned by the filtered to the ones in the input file and updates only those DNs that match in the input file.

The input enables the user to provide the PIN Generator with an exact list of DNs to modify; it is also possible to provide the PIN Generator with PINs in plain text for all DNs or for specific DNs.

There are two common situations when using an input file is useful:

- If PINs have been set for all entries in the user directory, and new users join the organization. For the new users to get certificates, the directory must contain PINs. PINs should be generated for only those two entries without changing any of the other user entries. Instead of constructing a complex LDAP filter, using an input file allows using a general filter, and the modified entries are restricted to the DNs of the two users listed in the input file.
- If a particular values, such as Social Security numbers, should be used as PINs, then the Social Security numbers can be put in the input file and provide those numbers as PINs to the PIN Generator. These are then stored as hashed values in the directory.

The format of the input file is the same as that of the output file (refer to *Section 2.2, "Output File"*) except for the status line. In the input file, PINs can be set for all the DNs in the file, for specific DNs, or for none of the DNs. If the PIN attribute is missing for a DN, the tool automatically generates a random PIN.

An input file looks like the following example:

```
dn:cn=user1, o=example.com

dn:cn=user2, o=example.com

...
dn:cn=user3, o=example.com
```

PINs can also be provided for the DNs in plain-text format; these PINs are hashed according to the command-line arguments.

```
dn:cn=user1, o=example.com
pin:pl229Ab

dn:cn=user2, o=example.com
pin:9j65dSf

...
dn:cn=user3, o=example.com
pin:3knAg60
```

NOTE

Hashed PINs cannot be provided to the tool.

2.2. Output File

The PIN Generator can capture the output to a text file specified by the `output` option.

The output contains a sequence of records in the following format:

```
dn: user_dn1
pin: generated_pin1
status: status1

dn: user_dn2
pin: generated_pin2
status: status2

...
dn: user_dn#
pin: generated_pin#
status: status#
```

where *user_dn* is a distinguished name matching the DN filter or listed in the input file. By default, the delimiter is a semi-colon (;) or the character defined on the command line. *generated_pin* is a string of characters of fixed or variable length, depending on the length parameters used. *status* is one of the values listed *Table 6.2, "PIN Generator Status "*.

The first line in each record is always the DN. The subsequent lines for `pin` and `status` are optional. The record ends with a blank line, using the Unix end of line sequence, (`\n`).

2.3. How PINs Are Stored in the Directory

Each PIN is concatenated with the corresponding LDAP attribute named in the `saltattribute` argument. If this argument is not specified, the DN is used. That string is hashed with the routine specified in the hash argument; the default algorithm is SHA-1. One byte is prepended to indicate the hash type used. The PIN is stored as follows:

```
byte[0] = X
```

The value of `X` depends on the hash algorithm chosen during the PIN generation process.

X	Hash Algorithm
0	SHA-1
1	MD5
45	none

Table 6.3.

The PIN is stored in the directory as a binary value, not as a base-64 encoded value.

2.4. Exit Codes

When the PIN Generator is finished running, it returns a result code showing how it ended. These result codes are listed in *Table 6.4, "Result Codes Returned by the PIN Generator"*.

Result Code	Description
`0`	The PIN generation was successful; PINs were set for all the DNs in the specified directory.
`4`	The tool could not bind to the directory as the user specified in the `binddn` parameter.
`5`	The tool could not open the output file specified in the `output` parameter.
`7`	There was an error parsing command-line arguments.
`8`	The tool could not open the input file specified in the `input` parameter.
`9`	The tool encountered an internal error.
`10`	The tool found a duplicate entry in the input file.
`11`	The tool did not find the salt attribute specified in the `saltattribute` parameter in the directory.

Table 6.4. Result Codes Returned by the PIN Generator

ASCII to Binary

The Certificate System ASCII to binary tool converts ASCII base-64 encoded data to binary base-64 encoded data.

1. Syntax

The ASCII to binary tool, `AtoB`, has the following syntax:

```
AtoB input_file output_file
```

Option	Description
input_file	Specifies the path and file to the base-64 encoded ASCII data.
output_file	Specifies the file where the utility should write the binary output.

Table 7.1.

2. Usage

The example command takes the base-64 ASCII data in the `ascii_data.in` file and writes the binary equivalent of the data to the `binary_data.out` file.

```
AtoB /usr/home/smith/test/ascii_data.in /usr/home/smith/test/binary_data.out
```

Binary to ASCII

The Certificate System binary to ASCII tool, `BtoA` converts binary base-64 encoded data to ASCII base-64 encoded data.

1. Syntax

The `BtoA` tool uses the following syntax:

```
BtoA input_file output_file
```

Option	Description
input_file	Specifies the path and file of the base-64 encoded binary data.
output_file	Specifies the path and file to which the tool should write the ASCII output.

Table 8.1.

2. Usage

The following example of the `BtoA` utility takes the base-64 encoded binary data in the `binary_data.in` file and writes the ASCII equivalent of the data to the `ascii_data.out` file.

```
BtoA /usr/home/smith/test/binary_data.in /usr/home/smith/test/ascii_data.out
```

Pretty Print Certificate

The Pretty Print Certificate utility, `PrettyPrintCert`, prints the contents of a certificate stored as ASCII base-64 encoded data to a readable format.

1. Syntax

The `PrettyPrintCert` command has the following syntax:

```
PrettyPrintCert [-simpleinfo] input_file [output_file]
```

Option	Description
`simpleinfo`	*Optional.* Prints limited certificate information in an easy to parse format.
input_file	Specifies the path to the file containing the ASCII base-64 encoded certificate.
output_file	*Optional.* Specifies the path and file to which the tool should write the certificate. If this option is not specified, the certificate information is written to the standard output.

Table 9.1.

2. Usage

The following example converts the ASCII base-64 encoded certificate in the `ascii_cert.in` file and writes the certificate in the pretty-print form to the output file `ascii_cert.out`.

```
PrettyPrintCert /usr/home/smith/test/ascii_cert.in
/usr/home/smith/test/ascii_cert.out
```

The base-64 encoded certificate data in the `ascii_cert.in` looks like the following:

```
-----BEGIN CERTIFICATE-----
MIIC2DCCAkGgAwIBAgICEAwwDQYJKoZIhvcNAQEFBQAwfDELMAkGA1UEBhMCVVMxIzA
hBgNVBAoTGlBhbG9va2FWaWxsZSBXaWRnZXRzLCBJbmMuMR0wGwYDVQQLExRXaWRnZX
QgTWFrZXJzICdSJyBVczEpMCcGA1UEAxMgVGVzdCBUZXN0IFRlc3QgVGVzdCBUZXN0I
FRlc3QgQ0EwHhcNOTkwMjE4MDMMzM5WhcNMDAwMjE4MDM0MzM5WjCBrjELMAkGA1UEB
hMCVVMxJjAkBgNVBAoTHU5ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnAuMRUwEwYD
VQQLEwOZXRzY2FwZSBDTVMxGDAWBEBEwhtaGFybXNlbjEfMB0GA1UEAxWaW50ZGV2Y2
EgQWRtaW5pcwp0frfJOObeiSsia3BuifRHBNw95ZZQR9NIXr1x5bE
```

```
-----END CERTIFICATE-----
```

The certificate in pretty-print format in the `ascii_cert.out` file looks like the following:

```
Certificate:
Data:
Version: v3
Serial Number: 0x100C
Signature Algorithm: OID.1.2.840.113549.1.1.5 -1.2.840.113549.1.1.5
Issuer: CN=Test CA,OU=Widget Makers 'R'Us,O=Example Corporation,
Widgets\,Inc.,C=US
Validity:
 Not Before: Wednesday, February 17, 1999 7:43:39 PM
 Not After: Thursday, February 17, 2000 7:43:39 PM
Subject: MAIL=admin@example.com,CN=testCA Administrator, UID=admin, OU=IS,
 O=Example Corporation,C=US
Subject Public Key Info:
 Algorithm: RSA - 1.2.840.113549.1.1.1
 Public Key:
   30:81:89:02:81:81:00:DE:26:B3:C2:9D:3F:7F:FA:DF:
   24:E3:9B:7A:24:AC:89:AD:C1:BA:27:D1:1C:13:70:F7:
   96:59:41:1F:4D:21:7A:F5:C7:96:C4:75:83:35:9F:49:
   E4:B0:A7:5F:95:C4:09:EA:67:00:EF:BD:7C:39:92:11:
   31:F2:CA:C9:16:87:B9:AD:B8:39:69:18:CE:29:81:5F:
   F3:4D:97:B9:DF:B7:60:B3:00:03:16:8E:C1:F8:17:6E:
   7A:D2:00:0F:7D:9B:A2:69:35:18:70:1C:7C:AE:12:2F:
   0B:0F:EC:69:CD:57:6F:85:F3:3E:9D:43:64:EF:0D:5F:
   EF:40:FF:A6:68:FD:DD:02:03:01:00:01:
Extensions:
 Identifier: 2.16.840.1.113730.1.1
 Critical: no
 Value: 03:02:00:A0:
Identifier: Authority Key Identifier - 2.5.29.35
 Critical: no
 Key Identifier:
   EB:B5:11:8F:00:9A:1A:A6:6E:52:94:A9:74:BC:65:CF:
 07:89:2A:23:
Signature:
 Algorithm: OID.1.2.840.113549.1.1.5 - 1.2.840.113549.1.1.5
 Signature:
   3E:8A:A9:9B:D1:71:EE:37:0D:1F:A0:C1:00:17:53:26:
   6F:EE:28:15:20:74:F6:C5:4F:B4:E7:95:3C:A2:6A:74:
   92:3C:07:A8:39:12:1B:7E:C4:C7:AE:79:C8:D8:FF:1F:
   D5:48:D8:2E:DD:87:88:69:D5:3A:06:CA:CA:9C:9A:55:
   DA:A9:E8:BF:36:BC:68:6D:1F:2B:1C:26:62:7C:75:27:
   E2:8D:24:4A:14:9C:92:C6:F0:7A:05:A1:52:D7:CC:7D:
   E0:9D:6C:D8:97:3A:9C:12:8C:25:48:7F:51:59:BE:3C:
   2B:30:BF:EB:0A:45:7D:A6:49:FB:E7:BE:04:05:D6:8F:
```

The following example command takes the ASCII base-64 encoded certificate in the `ascii_cert.in` file and writes the information contained within the certificate to the simple

format output file `cert.simple`.

```
PrettyPrintCert -simpleinfo /usr/home/smith/test/ascii_cert.in
/usr/home/smith/test/cert.simple
```

The base-64 encoded certificate data in `ascii_cert.in` file looks similar to the following:

```
-----BEGIN CERTIFICATE-----
MIIC2DCCAkGgAwIBAgICEAwwDQYJKoZIhvcNAQEFBQAwfDELMAkGA1UEBhMCVVMxIzA
hBgNVBAoTGlBhbG9va2FWaWxsZSBXaWRnZXRzLCBJbmMuMR0wGwYDVQQLExRXaWRnZX
QgTWFrZXJzICdSJyBVczEpMCcGA1UEAxMgVGVzdCBUZXN0IFRlc3QgVGVzdCBUZXN0I
FRlc3QgQ0EwHhcNOTkwMjE4MDMMzM5WhcNMDAwMjE4MDM0MzM5WjCBrjELMAkGA1UEB
hMCVVMxJjAkBgNVBAoTHU5ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnAuMRUwEwYD
VQQLEwOZXRzY2FwZSBDTVMxGDAWBEBEwhtaGFybXNlbjEfMB0GA1UEAxWaW50ZGV2Y2
EgQWRtaW5pcwp0frfJOObeiSsia3BuifRHBNw95ZZQR9NIXr1x5bE
-----END CERTIFICATE-----
```

The simple certificate information in the `cert.simple` output file looks like the following:

```
  MAIL=admin@example.com
  CN=testCA Administrator
  UID=admin
  OU=IS
  O=Example Corporation
  C=US
```

Pretty Print CRL

The Pretty Print CRL tool, `PrettyPrintCrl`, prints the contents of a certificate revocation list (CRL) in an ASCII base-64 encoded file in a readable form.

1. Syntax

The `PrettyPrintCrl` utility has the following syntax:

```
PrettyPrintCrl input_file [output-file]
```

Option	Description
input_file	Specifies the path to the file that contains the ASCII base-64 encoded CRL.
output_file	*Optional.* Specifies the path to the file to write the CRL. If the output file is not specified, the CRL information is written to the standard output.

Table 10.1.

2. Usage

The following example `PrettyPrintCrl` command takes the ASCII base-64 encoded CRL in the `ascii_crl.in` file and writes the CRL in the pretty-print form to the output file `ascii_crl.out`.

```
PrettyPrintCrl /usr/home/smith/test/ascii_crl.in
/usr/home/smith/test/ascii_crl.out
```

The base-64 encoded CRL in the `ascii_crl.in` file looks like the following:

```
-----BEGIN CRL-----
MIIBkjCBAIBATANBgkqhkiG9w0BAQQFADAsMREwDwYDVQQKEwhOZXRzY2FwZTEXMBUG
A1UEAxMOQ2VydDQwIFRlc3QgQ0EXDTk4MTIxNzIyMzcyNFowgaowIAIBExcNOTgxMjE
1MTMxODMyWjAMMAoGA1UdFQQDCgEBMCACARIXDTk4MTINTEzMjA0MlowDDAKBgNVHRU
EAwoBAjAgAgERFw05ODEyMTYxMjUxNTRaMAwwCgYDVR0VBAMKAQEwIAIBEBcNOTgxMj
E3MTAzNzI0WjAMMAoGA1UdFQQDCgEDMCACAQoXDTk4MTEyNTEzMTExOFowDDAKBgNVH
RUEAwoBATANBgkqhkiG9w0BQQFAAOBgQBCN85O0GPTnHfImYPROvoorx7HyFz2ZsuKs
VblTcemsX0NL7DtOa+MyY0pPrkXgm157JrkxEJ7GBOeogbAS6iFbmeSqPHj8+
-----END CRL-----
```

The CRL in pretty-print format in the `ascii_crl.out` output file looks like the following:

```
Certificate Revocation List:
Data:
Version: v2
Signature Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
Issuer: CN=Test CA,O=Example Corporation
This Update: Thu Dec 17 14:37:24 PST 1998
Revoked Certificates:
Serial Number: 0x13
 Revocation Date: Tuesday, December 15, 1998 5:18:32 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Serial Number: 0x12
 Revocation Date: Tuesday, December 15, 1998 5:20:42 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: CA_Compromise
Serial Number: 0x11
 Revocation Date: Wednesday, December 16, 1998 4:51:54 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Serial Number: 0x10
 Revocation Date: Thursday, December 17, 1998 2:37:24 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Affiliation_Changed
Serial Number: 0xA
 Revocation Date: Wednesday, November 25, 1998 5:11:18 AM
 Extensions:
   Identifier: Revocation Reason - 2.5.29.21
   Critical: no
   Reason: Key_Compromise
Signature:
 Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
 Signature:
   42:37:CE:4E:D0:63:D3:9C:77:C8:99:83:D1:3A:FA:28:
   AF:1E:C7:C8:5C:F6:66:CB:8A:B1:56:E5:4D:C7:A6:B1:
   7D:0D:2F:B0:ED:39:AF:8C:C9:8D:29:3E:B9:17:82:6D:
   79:EC:9A:E4:C4:42:7B:18:13:9E:A2:06:C0:4B:A8:85:
   6E:67:92:A8:F1:E3:F3:E2:41:1F:9B:2D:24:D9:DF:4C:
   2B:A1:68:CE:96:C7:AF:F7:5B:F7:3D:2F:06:57:39:74:
   CF:B2:FA:46:C6:AD:18:60:8D:3E:0C:F7:C1:66:52:37:
   CF:89:42:B0:D7:33:C4:95:7E:F4:D9:1E:32:B8:5E:12:
```

TKS Tool

The TKS utility, `tksTool`, manages keys, including keys stored on tokens, the TKS master key, and related keys and databases.

1. Syntax

The `tksTool` can be used to manage certificates and keys in several different ways. The syntax for these different operations is as follows:

- Deleting a key from a token.

```
tksTool -D -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Inputting shares to generate a new transport key.

```
tksTool -I -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Displaying the key check value (KCV) of the specified key.

```
tksTool -K -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Listing a specified key or all keys.

```
tksTool -L -n keyname -d dbdir [-h all | -h token_name]
 [-p dbprefix] [-f pwfile] [-x]
```

- Generating a new master key.

```
tksTool -M -n keyname -d dbdir [-h token_name] [-p dbprefix] [-f pwfile]
```

- Creating a new key database.

```
tksTool -N -d dbdir [-p dbprefix] [-f pwfile]
```

- Changing the key database password.

```
tksTool -P -d dbdir [-p dbprefix] [-f pwfile]
```

- Renaming a symmetric key.

```
tksTool -R -n keyname -r new_keyname -d dbdir [-h token_name]
 [-p dbprefix] [-f pwfile]
```

- Listing all security modules.

```
tksTool -S -d dbdir [-p dbprefix] [-x]
```

- Generating a new transport key.

```
tksTool -T -n keyname -d dbdir [-h token_name]
 [-p dbprefix] [-f pwfile] [-z noiseFile]
```

- Unwrapping a wrapped master key.

```
tksTool -U -n keyname -d dbdir -t transport_keyname -i inputFile
 [-h token_name] [-p dbprefix] [-f pwfile]
```

- Wrapping a new master key.

```
tksTool -W -n keyname -d dbdir -t transport_keyname -o outputFile
 [-h token_name] [-p dbprefix] [-f pwfile]
```

NOTE

Chrysalis-ITS version 2.3 is required to support version 1.0 of the `-R` option of the `tksTool`.

Transport keys residing on Chrysalis-ITS hardware tokens created by an earlier version of `tksTool` cannot have their KCV values determined with the `-K` option of the `tksTool` because the `CKA_ENCRYPT` and `CKF_ENCRYPT` bits were not set when they were created by the previous tool.

The `tksTool` options are as follows:

Option	Description
`D`	Deletes a key from the token.
`d`	*Required.* Gives the security module database (HSM, if allowed for that operation) or the key database directory (software).
`f`	Gives the path and filename of the password file, if one is used.
`h`	Gives the token name for the toke which contains the key to be managed. Some operations allow an `all` option to manage all keys in the token.
`I`	Inputs shares to generate a new transport key.
`i`	*Required with -U.* Gives the path and filename of the input file which contains the wrapped master key.
`K`	Displays the KCV of the specified key.
`L`	Lists the specified key or all keys.
`M`	Generates a new master key.
`N`	Creates a new key database (software).
`n`	*Required for every operation except -N, -P, and -S.* Gives the name of the key being managed.
`o`	*Required with -W.* Gives the path and filename for the file to which to output the new wrapped master key.
`P`	Changes the key database password (software).
`p`	Gives the prefix to the key database directory.
`R`	Renames a symmetric key.
`r`	*Required with -R*. Gives the new key name.
`S`	Lists all security modules.
`T`	Generates a new transport key.
`t`	*Required with -U and -W.* Gives the name of the transport key being managed.
`U`	Unwraps the wrapped master key.
`W`	Wraps the new master key.
`x`	Forces the database to be read/write.

Option	Description
z	Gives the path and filename of the noise file to generate the key.

Table 11.1.

There are two additional options which can be used with `tksTool` to get more information about the utility.

Option	Description
H	Displays the extended help information.
V	Display the version number of the `tksTool` tool.

Table 11.2.

2. Usage

1. Check the version of `tksTool` by running the following command:

```
tksTool -V
```

   This should return output similar to the following:

```
tksTool: Version 1.0
```

2. Create new software databases.

```
tksTool -N -d .
Enter a password which will be used to encrypt your keys.
The password should be at least 8 characters long,
and should contain at least one non-alphabetic character.

Enter new password:
Re-enter password:
```

NOTE

A hardware HSM can be used instead of the software database if the `modutil` utility is first used to insert the HSM slot and token into the `secmod.db` database.

If an HSM is used, then the option `-h` *hsm_token* must be added to each of commands below.

3. List the contents of the local software key database.

```
tksTool -L -d .

slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
tksTool: the specified token is empty
```

4. Create a transport key called `transport`.

```
tksTool -T -d . -n transport
```

5. When prompted, fill in the database password, then type in some noise to seed the random number generator.
6. The session key share and corresponding KCV are displayed. Write down both of these.
7. Run the following command to produce an identical transport key; this is generally used within another set of databases which need to use identical transport keys. When this is run, multiple session key shares and KCVs are generated. Write down all of this information.

```
tksTool -I -d . -n verify_transport
```

Responses similar to the following appear:

```
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .
     transport key KCV: A428 53BA
Storing transport key on final specified token . . .
Naming transport key "transport" . . .
```

```
Successfully generated, stored, and named the transport key!
```

8. List the contents of the key database again.

```
tksTool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 transport
```

9. Use the transport key to generate and wrap a master key, and store the master key in a file called `file`.

```
tksTool -W -d . -n wrapped_master -t transport -o file

Enter Password or Pin for "NSS Certificate DB":
Retrieving the transport key (for wrapping) from the specified token . . .
Generating and storing the master key on the specified token . . .
Naming the master key "wrapped_master" . . .
Successfully generated, stored, and named the master key!
Using the transport key to wrap and store the master key . . .
Writing the wrapped data (and resident master key KCV) into the file
called "file" . . .

       wrapped data:   47C0 06DB 7D3F D9ED
                       FE91 7E6F A7E5 91B9
       master key KCV: CED9 4A7B
       (computed KCV of the master key residing inside the wrapped data)
```

10. List the contents of the software key database again.

```
tksTool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 wrapped_master
 1 transport
```

NOTE

The order of the keys is not important, and some systems may display the keys in a different order.

11. Use the transport key to generate and unwrap a master key called `unwrapped_master` stored in a file called `file`.

```
tksTool -U -d . -n unwrapped_master -t transport -i file

Enter Password or Pin for "NSS Certificate DB":
Retrieving the transport key from the specified token (for unwrapping) . . .
Reading in the wrapped data (and resident master key KCV) from the file
called "file" . . .

     wrapped data:   47C0 06DB 7D3F D9ED
                     FE91 7E6F A7E5 91B9
     master key KCV: CED9 4A7B
     (pre-computed KCV of the master key residing inside the wrapped data)

Using the transport key to temporarily unwrap the master key to
recompute its KCV value to check against its pre-computed KCV value . . .
     master key KCV: CED9 4A7B
     (computed KCV of the master key residing inside the wrapped data)
     master key KCV: CED9 4A7B
     (pre-computed KCV of the master key residing inside the wrapped data)

Using the transport key to unwrap and store the master key on the
specified token . . .
Naming the master key "unwrapped_master" . . .
Successfully unwrapped, stored, and named the master key!
```

12. List the contents of the key database to show all keys.

```
tksTool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 unwrapped_master
 1 wrapped_master
 2 transport
```

13. Delete a key from the database.

```
tksTool -D -d . -n wrapped_master

Enter Password or Pin for "NSS Certificate DB":
tksTool: 1 key(s) called "wrapped_master" were deleted
```

14. List the contents of the key database again to show all keys.

```
tksTool -L -d .

 slot: NSS User Private Key and Certificate Services
token: NSS Certificate DB

Enter Password or Pin for "NSS Certificate DB":
 0 unwrapped_master
 1 transport
```

CMC Request

The CMC Request utility, `CMCRequest`, creates a CMC request from one or more PKCS #10 or CRMF requests. The utility can also be used to revoke certificates.

1. Syntax

The `CMCRequest` command uses a configuration file (`.cfg`) as a parameter. The `.cfg` file must include the path to the file of the formatted CMC request:

```
CMCRequest /path/to/file.cfg
```

For revocation requests, the `revRequest.enable` parameter must be set to `true`, and related parameters must contain the appropriate information.

The `.cfg` file contains the following parameters:

Parameters	Description
`numRequests`	The total number of PKCS #10 or CRMF requests. In some cases, the value of this parameter can be 0. For example, `numRequests=1`.
`input`	The full path and filename of the PKCS #10 or CRMF request, which must be in base-64 encoded format. Multiple filenames are separated by white space. This parameter is a required if the value for `numRequests` is greater than 0. For example, `input=crmf1`.
`output`	*Required.* The full path and filename for the generated binary CMC request. For example, `output=cmc`.
`nickname`	*Required.* The nickname of the agent certificate used to sign the full CMC request. For example, `nickname=CS Agent-102504a's 102504a ID`.
`dbdir`	

Parameters	Description
	Required. The full path to the directory where the `cert8.db`, `key3.db`, and `secmod.db` databases are located. For example, `dbdir=/u/smith/db/`.
`password`	*Required.* The token password for `cert8.db`, which stores the agent certificate. For example, `password=redhat`.
`format`	The request format, either `pkcs10` or `crmf`. For example, `format=crmf`.

Table 12.1.

The following `.cfg` file parameters set CMC controls:

Parameters	Description
`confirmCertAcceptance.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `confirmCertAcceptance.enable=false`.
`confirmCertAcceptance.serial`	The serial number for the `confirmCertAcceptance` control. For example, `confirmCertAcceptance.serial=3`.
`confirmCertAcceptance.issuer`	The issuer name for the `confirmCertAcceptance` control. For example, `confirmCertAcceptance.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us`.
`getCert.enable`	If set to `true`, then the request contains this attribute. If this parameter is not set, the value is assumed to be `false`.

Parameters	Description
	For example, `getCert.enable=false`.
`getCert.serial`	The serial number for the `getCert` control. For example, `getCert.serial=300`.
`getCert.issuer`	The issuer name for the `getCert` control. For example, `getCert.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us`.
`dataReturn.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `dataReturn.enable=false`.
`dataReturn.data`	The data contained in the `dataReturn` control. For example, `dataReturn.data=test`.
`transactionMgt.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `transactionMgt.enable=true`.
`transactionMgt.id`	The transaction identifier for `transactionMgt` control. VeriSign recommends that the transaction ID should be an MD5 hash of the public key.
`senderNonce.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `senderNonce.enable=false`.
`senderNonce.id`	The ID for the `senderNonce` control. For example, `senderNonce.id=testing`.
`revRequest.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value

Parameters	Description
	is assumed to be `false`. For example, `revRequest.enable=true`.
`revRequest.nickname`	The nickname for the certificate being revoked. For example, `revRequest.nickname=newuser's 102504a ID`.
`revRequest.issuer`	The issuer name for the certificate being revoked. For example, `revRequest.issuer=cn=Certificate Manager,ou=102504a,o=102504a,c=us`.
`revRequest.serial`	The serial number for the certificate being revoked. For example, `revRequest.serial=75`.
`revRequest.reason`	The reason for revoking this certificate. The allowed values are `unspecified`, `keyCompromise`, `caCompromise`, `affiliationChanged`, `superseded`, `cessationOfOperation`, `certificateHold`, and `removeFromCRL`. For example, `revRequest.reason=unspecified`.
`revRequest.sharedSecret`	The shared secret for the revocation request. For example, `revRequest.sharedSecret=testing`.
`revRequest.comment`	A text comment for the revocation request. For example, `revRequest.comment=readable comment`.
`revRequest.invalidityDatePresent`	If set to `true`, the current time is the invalidity date for the revoked certificate. If set to `false`, no invalidity date is present.

Parameters	Description
	For example, `revRequest.invalidityDatePresent=false`.
`identityProof.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `identityProof.enable=false`.
`identityProof.sharedSecret`	The shared secret for `identityProof` control. For example, `identityProof.sharedSecret=testing`.
`popLinkWitness.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `popLinkWitness.enable=false`.
`LraPopWitness.enable`	If set to `true`, then the request contains this control. If this parameter is not set, the value is assumed to be `false`. For example, `LraPopWitness.enable=false`.
`LraPopWitness.bodyPartIDs`	The space-delimited list of body part IDs for the `LraPopWtiness` control. For example, `LraPopWitness.bodyPartIDs=1 .`

Table 12.2.

2. Usage

Once a simple CMC request, a PKCS #10 request, has been generated, do the following to send it to the CA:

1. Run the `AtoB` tool to convert the base-64-encoded PKCS #10 request to binary.
2. Use the `HttpClient` utility to send the request.

By default, the URI of the servlet that processes a simple CMC request is `/ca/ee/ca/profileSubmitCMCSimple`; this must be specified in the `HttpClient` configuration.

CMC Enrollment

The CMC Enrollment utility, `CMCEnroll`, is used to sign a certificate request with an agent's certificate. This can be used in conjunction with the CA end-entity CMC Enrollment form to sign and enroll certificates for users.

1. Syntax

This utility has the following syntax:

```
CMCEnroll -d directory_containing_agent_cert -h db_password -n
certificate_nickname
 -r certificate_request_file -p certificate_DB_passwd [-c comment]
```

Option	Description
`d`	The directory containing the `cert8.db`, `key3.db`, and `secmod.db` files associated with the agent certificate.
`h`	Password to the directory specified in the `d` option.
`n`	The nickname of the certificate.
`r`	The filename of the certificate request.
`p`	The password to the browser certificate database.
`c`	*Optional.* Includes comments about the request.

Table 13.1.

NOTE

Surround values that include spaces with quotation marks.

2. Usage

Signed requests must be submitted to the CA, either by sending them directly to the Certificate Authority or by using the CA agent page. Certificate System provides a Certificate Authority Certificate Enrollment form called `CMCEnrollment.html`. The default configuration of this form does not include the necessary field to paste an enrollment request. To use this form to submit

requests, change the configuration so that this field is available.

To enable the CMC Enrollment form for the CA end-entity interface, do the following:

1. Open the CA's web directory in `/var/lib/rhpki-ca/web-apps/ca/ee/ca`.
2. Open the `CMCEnrollment.html` file.
3. Find the following line:

```
form method="post" action="/enrollment" onSubmit="return
validate(document.forms[0])"
```

4. Add the following line below that line:

```
input type="hidden" name="authenticator" value="CMCAuth"
```

5. After configuring the HTML form, test `CMCEnroll` and the form by doing the following:
   a. Create a certificate request using `certutil`.
   b. Copy the PKCS #10 ASCII output to a text file.
   c. Run the `CMCEnroll` command to sign the certificate request. If the input file is `request34.txt`, the agent's certificate is stored in the `/export/certs` directory, the certificate common name for this CA is `CertificateManagerAgentsCert`, and the password for the certificate database is `1234pass`, the command is as follows:

```
CMCEnroll -d "/export/certs" -n "CertificateManagerAgentsCert" -r
"/export/requests/request34.txt" -p "1234pass"
```

      The output of this command is stored in a file with the same filename and `.out` appended to the filename.

   d. Submit the signed certificate through the CA end-entities page.
      i. Open the end-entities page.
      ii. Select the CMC Enrollment profile form.
      iii. Paste the content of the output file into the first text area of this form.
      iv. Remove `-----BEGIN NEW CERTIFICATE REQUEST-----` and `----END NEW CERTIFICATE REQUEST-----` from the pasted content.
      v. Select **Certificate Type User Certificate**, fill in the contact information, and submit the

form.

e. The certificate is immediately processed and returned since a signed request was sent and the `CMCAuth` plug-in was enabled.

f. Use the agent page to search for the new certificates.

CMC Response

The CMC Response utility, `CMCResponse`, parses a CMC response received by the utility.

1. Syntax

The CMC Response utility uses the following syntax:

```
CMCResponse -d directoryName -i /path/to/CMCResponse.file
```

Options	Description
`d`	Specifies the path to the `cert8.db` directory.
`i`	Specifies the path and filename of the CMC response file.

Table 14.1.

The parsed output is printed to the screen.

CMC Revocation

The CMC Revocation utility, `CMCRevoke`, signs a revocation request with an agent's certificate.

1. Syntax

This utility has the following syntax:

```
CMCRevoke -d directoryName -n nickname -i issuerName -s serialName
 -m reasonToRevoke -c comment
```

Option	Description
`d`	The path to the directory where the `cert8.db`, `key3.db`, and `secmod.db` databases containing the agent certificates are located.
`n`	The nickname of the agent's certificate.
`i`	The issuer name of the certificate being revoked.
`s`	The decimal serial number of the certificate being revoked.
`m`	The reason the certificate is being revoked. The reason code for the different allowed revocation reasons are as follows: • *0* - Unspecified. • *1* - Key compromised. • *2* - CA key compromised. • *3* - Affiliation changed. • *4* - Certificate superseded. • *5* - Cessation of operation. • *6* - Certificate is on hold.
`c`	Text comments about the request.

Table 15.1.

NOTE

Surround values that include spaces in quotation marks.

2. Testing CMC Revocation

Test that CMC revocation is working properly by doing the following:

1. Create a CMC revocation request for an existing certificate. For example, if the directory containing the agent certificate is `/var/lib/rhpki-ca/alias/`, the nickname of the certificate is `CertificateManagerAgentCert`, and the serial number of the certificate is `22`, the command is as follows:

```
CMCRevoke -d "/var/lib/rhpki-ca/alias" -n "CertificateManagerAgentCert" -i
"cn=agentAuthMgr" -s 22 -m 0 -c "test comment"
```

2. Open the CA's end-entities page.
3. Select the **Revocation** tab.
4. Select the **CMC Revoke** link in the menu.
5. Paste the output from the `CMCRevoke` operation into the text box. Remove the `-----BEGIN NEW CERTIFICATE REQUEST-----` and `----END NEW CERTIFICATE REQUEST-----` lines from the pasted content.
6. Click **Submit**.
7. The results page displays that certificate 22 has been revoked.

CRMF Pop Request

The `CRMFPopClient` utility is a tool to send a Certificate Request Message Format (CRMF) request to a Certificate System CA with the request encoded with proof of possession (POP) data that can be verified by the CA server. If a client provides POP information with a request, the server can verify that the requester possesses the private key for the new certificate.

The tool does all of the following:

1. Has the CA enforce or verify POP information encoded within a CRMF request.
2. Makes simple certificate requests without using the standard Certificate System agent page or interface.
3. Makes a simple certificate request that includes a transport certificate for key archival from the DRM.

1. Syntax

There are two syntax styles for the `CRMFPopClient` utility, depending on the intended use:

```
CRMFPopClient token_password authenticator host port username password
 [pop_option] subject_dn [OUTPUT_CERT_REQ]
```

```
CRMFPopClient token_password [pop_option] OUTPUT_CERT_REQ subject_dn
```

Option	Description
token_password	The password for the cryptographic token.
authenticator	The authentication manager within the Certificate System; this is most often set to `nullAuthMgr`
host	The hostname of the CA instance.
port	The non-SSL port of the Certificate System CA.
username	The Certificate System user for whom the certificate request is issued.

Option	Description
password	The password of the Certificate System user.
pop_option	*Optional*. Sets the type of POP request to generate; since this can generate invalid requests, this option can be used for testing. There are three values: • `POP_SUCCESS`. Generates a request with the correct POP information; the server verifies that the information is correct. • `POP_FAIL`. Generates a request with incorrect POP information; the server rejects this request if it is submitted. This is used to test server configuration. • `POP_NONE`. Generates a CRMF request with no POP information. If the server is configured to verify all the POP information, then it rejects this request. In that case, it can be used to test the server configuration.
subject_dn	The distinguished name of the requested certificate.
`OUTPUT_CERT_REQ`	*Optional*. Prints the generated certificate request to the screen.

Table 16.1.

2. Usage

The following example generates a CRMF/POP request for the Certificate System user `admin`, has the server verify that the information is correct, and prints the certificate request to the screen:

```
CRMFPopClient password123 nullAuthMgr host.redhat.com 1026 admin redhat \
        POP_SUCCESS CN=MyTest,C=US,UID=MyUid OUTPUT_CERT_REQ
```

The following example generates a CRMF/POP request that includes a transport for key archival in the DRM. The `transport.txt` file containing the base-64 encoded transport

certificate must be in the same directory from which the utility is launched; the tool picks up this file automatically.

```
CRMFPopClient password123 POP_SUCCESS OUTPUT_CERT_REQ
CN=MyTest,C=US,UID=MyUid
```

NOTE

A file named `transport.txt` containing the transport certificate in base-64 format *must* be created in the directory from which the utility is launched. This file must be available for archival to a DRM.

Extension Joiner

The Certificate System provides policy plug-in modules that allow standard and custom X.509 certificate extensions to be added to end-entity certificates that the server issues. Similarly, the Certificate Setup Wizard that generates certificates for subsystem users allows extensions to be selected and included in the certificates. The wizard interface and the request-approval page of the agent interface contain a text area to paste any extension in its MIME-64 encoded format.

The text field for pasting the extension accepts a single extension blob. To add multiple extensions, they must first be combined into a single extension blob, then pasted into the text field. The `ExtJoiner` tool joins multiple extensions together into a single MIME-64 encoded blob. This new, combined blob can then be pasted in the wizard text field or the request-approval page of the agent interface to specify multiple extensions at once.

1. Syntax

The `ExtJoiner` utility has the following syntax:

```
ExtJoiner ext_file0 ext_file1 ... ext_fileN
```

Option	Description
ext_file#	Specifies the path and names for files containing the base-64 DER encoding of an X.509 extension.

Table 17.1.

2. Usage

`ExtJoiner` does not *generate* an extension in its MIME-64 encoded format; it joins existing MIME-64 encoded extensions. To join multiple custom extensions and add the extensions to a certificate request using `ExtJoiner`, do the following:

1. Find and note the location of the extension program files.
2. Run `ExtJoiner`, specifying the extension files. For example, if there are two extension files named `myExt1` and `myExt2` in a directory called `/etc/extensions`, then the command would be as follows:

```
ExtJoiner /etc/extensions/myExt1 /etc/extensions/myExt2
```

This creates a base-64 encoded blob of the joined extensions, similar to this example:

```
MEwwLgYDVR0lAQHBCQwIgYFKoNFBAMGClGC5EKDM5PeXzUGBi2CVyLNCQYFU
iBakowGgYDVR0SBBMwEaQPMA0xCzAJBgNVBAYTAlVT
```

3. Copy the encoded blob, without any modifications, to a file.

4. Verify that the extensions are joined correctly before adding them to a certificate request by converting the binary data to ASCII using the `AtoB` utility and then dumping the contents of the base-64 encoded blob using the `dumpasn1` utility. For information on the `AtoB` utility, see *Chapter 7, ASCII to Binary*. The `dumpasn1` tool can be downloaded at *http://fedoraproject.org/extras/4/i386/repodata/repoview/dumpasn1-0-20050404-1.fc4.html*.

   a. Run the `AtoB` utility to convert the ASCII to binary.

```
AtoBinput_file output_file
```

      where *input_file* is the path and file containing the base-64 encoded data in ASCII and *output_file* is the path and file for the utility to write the binary output.

   b. Run the `dumpasn1` utility.

```
dumpasn1output_file
```

      where *output_file* is the path and file containing the binary data. The output looks similar to this:

```
0 30 76: SEQUENCE {
2 30 46: SEQUENCE {
4 06 3: OBJECT IDENTIFIER extKeyUsage (2 5 29 37)
9 01 1: BOOLEAN TRUE
12 04 36: OCTET STRING
 : 30 22 06 05 2A 83 45 04 03 06 0A 51 82 E4 42 83
 : 33 93 DE 5F 35 06 06 2D 82 57 22 CD 09 06 05 51
 : 38 81 6A 4A
 : }
50 30 26: SEQUENCE {
52 06 3: OBJECT IDENTIFIER issuerAltName (2 5 29 18)
57 04 19: OCTET STRING
 : 30 11 A4 0F 30 0D 31 0B 30 09 06 03 55 04 06 13
 : 02 55 53
 : }
 : }
```

```
0 warnings, 0 errors.
```

If the output data do not appeat to be correct, check that the original Java™ extension files are correct, and repeat converting the files from ASCII to binary and dumping the data until the correct output is returned.

5. When the extensions have been verified, copy the base-64 encoded blob that was created by running `ExtJoiner` to the Certificate System wizard screen, and generate the certificate or the certificate signing request (CSR).

Key Usage Extension

The `GenExtKeyUsage` tool creates a base-64 encoded blob that adds `ExtendedKeyUsage` (OID 2.5.29.37) to the certificate. This blob is pasted into the certificate approval page when the certificate is created.

1. Syntax

The `GenExtKeyUsage` tool has the following syntax:

```
GenExtKeyUsage [true|false] OID ...
```

Option	Description
`true | false`	Sets the criticality. `true` means the extension is critical; `false` means it is not critical. The criticality value is used during the certificate validation process. If an extension is marked as critical, then the path validation software must be capable of interpreting that extension.
OID	The OID numbers that represent each certificate type selected for the certificate.

Table 18.1.

For more information on the OIDs that can be used for each certificate type, refer to appendix A, "Certificate and CRL Extensions," in the *Certificate System Administrator's Guide*.

Issuer Alternative Name Extension

The `GenIssuerAltNameExt` creates a base-64 encoded blob that adds the issuer name extensions, `IssuerAltNameExt` (OID 2.5.29.18), to the new certificate. This blob is pasted into the certificate approval page when the certificate is created.

1. Syntax

The `GenIssuerAltNameExt` tool uses parameter pairs where the first parameter specifies the general type of name attribute which is used for the issuer and the second parameter gives that name in that format. The tool has the following syntax:

```
GenIssuerAltNameExt general_type0 general_name0 ... general_typeN
general_nameN
```

Parameter	Description
general_type	Sets the type of name. It can be one of the following strings: • `RFC822Name` • `DirectoryName` • `DNSName` • `EDIPartyName` • `URIName` • `IPAddress` • `OIDName` • `OtherName`
general_name	A string, conforming to the name type, that gives the name of the issuer. • For `RFC822Name`, the value must be a valid Internet mail address. For example, `testCA@example.com`. • For `DirectoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example, `cn=SubCA, ou=Research Dept,`

Parameter	Description
	`o=Example Corporation, c=US`. • For `DNSName`, the value must be a valid fully-qualified domain name. For example, `testCA.example.com`. • For `EDIPartyName`, the value must be an IA5String. For example, `Example Corporation`. • For `URIName`, the value must be a non-relative URI following the URL syntax and encoding rules. The name must include both a scheme, such as `http`, and a fully qualified domain name or IP address of the host. For example, `http://testCA.example.com`. • For `IPAddress`, the value must be a valid IP address. An IPv4 address must be in the format `n.n.n.n` or `n.n.n.n,m.m.m.m`. For example, `128.21.39.40` or `128.21.39.40,255.255.255.00`. An IPv 6 address with netmask is separated by a comma. For example, `0:0:0:0:0:0:13.1.68.3`, `FF01::43`, `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:2` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`. • For `OIDName`, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, `1.2.3.4.55.6.5.99`. • `OtherName` is used for names with any other format; this supports `PrintableString`, `IA5String`, `UTF8String`, `BMPString`, `Any`, and `KerberosName`. `PrintableString`, `IA5String`, `UTF8String`, `BMPString`, and `Any` set a string to a base-64 encoded file specifying the subtree, such as `/var/lib/rhpki-ca/othername.txt`. `KerberosName` has the format *Realm*|*NameType*|*NameStrings*, such as

Parameter	Description
	`realm1|0|userID1,userID2`.

Table 19.1.

2. Usage

The following example sets the issuer name in the `RFC822Name` and `DirectoryName` formats:

```
GenIssuerAltNameExt RFC822Name TomTom@redhat.com DirectoryName cn=TomTom
```

Subject Alternative Name Extension

The `GenSubjectAltNameExt` creates a base-64 encoded blob to add the alternate subject name extension, `SubjectAltNameExt` (OID 2.5.29.17), to the new certificate. This blob is pasted into the certificate approval page when the certificate is created.

1. Syntax

The `GenSubjAltNameExt` tool uses parameter pairs where the first parameter specifies the type of name format, and the second parameter gives that name in the specified format.

This tool has the following syntax:

```
GenSubjectAltNameExt general_type0 general_name0 ... general_typeN
general_nameN
```

Parameter	Description
general_type	Sets the type of name that is used. This can be any of the following strings: • `RFC822Name` • `DirectoryName` • `DNSName` • `EDIPartyName` • `URIName` • `IPAddress` • `OIDName` • `OtherName`
general_name	A string, conforming to the specified format, of the subject name. • For `RFC822Name`, the value must be a valid Internet mail address. For example, `testCA@example.com`. • For `DirectoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example,

Parameter	Description
	`cn=SubCA, ou=Research Dept, o=Example Corporation, c=US`. • For `DNSName`, the value must be a valid fully-qualified domain name. For example, `testCA.example.com`. • For `EDIPartyName`, the value must be an IA5String. For example, `Example Corporation`. • For `URIName`, the value must be a non-relative URI following the URL syntax and encoding rules. The name must include both a scheme, such as `http`, and a fully qualified domain name or IP address of the host. For example, `http://testCA.example.com`. • For `IPAddress`, the value must be a valid IP address. An IPv4 address must be in the format `n.n.n.n` or `n.n.n.n,m.m.m.m`. For example, `128.21.39.40` or `128.21.39.40,255.255.255.00`. An IPv 6 address with netmask is separated by a comma. For example, `0:0:0:0:0:0:13.1.68.3`, `FF01::43`, `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:2` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`. • For `OIDName`, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, `1.2.3.4.55.6.5.99`. • `OtherName` is used for names with any other format; this supports `PrintableString`, `IA5String`, `UTF8String`, `BMPString`, `Any`, and `KerberosName`. `PrintableString`, `IA5String`, `UTF8String`, `BMPString`, and `Any` set a string to a base-64 encoded file specifying the subtree, such as `/var/lib/rhpki-ca/othername.txt`. `KerberosName` has the format

Parameter	Description
	Realm|*NameType*|*NameStrings*, such as `realm1|0|userID1,userID2`.

Table 20.1.

2. Usage

In the following example, the subject alternate names are set to the `RFC822Name` and `DirectoryName` types.

```
GenSubjectAltNameExt RFC822Name TomTom@redhat.com DirectoryName cn=TomTom
```

HTTP Client

The HTTP Client utility, `HttpClient`, sends a CMC request (created with the CMC Request utility) or a PKCS #10 request to a CA.

1. Syntax

This utility takes a single `.cfg` configuration file as a parameter. The syntax is as follows:

```
HttpClient /path/to/file.cfg
```

The `.cfg` file has the following parameters:

Parameters	Description
`host`	The hostname for the Certificate System server. For example: `host=server.com`
`port`	The port number for Certificate System server. For example: `port=1028`
`secure`	`true` for an HTTPS connection, `false` for an HTTP connection. For example: `secure=true`
`input`	The full path and filename for the enrollment request, which must be in binary format. For example: `input=cmcReqCRMFBin`
`output`	The full path and filename for the response in binary format. For example: `output=cmcResp`
`dbdir`	The full path to the directory where the `cert8.db`, `key3.db`, and `secmod.db` databases are located. This parameter is ignored if `secure=false`. For example: `dbdir=/usr/bin`
`clientmode`	`true` for client authentication, `false` for no client authentication. This parameter is ignored if `secure=false`. For example: `clientmode=true`
`password`	The password for the `cert8.db` database. This parameter is ignored if `secure=false` and `clientauth=false`. For example: `password=redhat`
`nickname`	The nickname of the client certificate. This parameter is ignored if `clientmode=false`. For example: `nickname=CS Agent-102504a's 102504a ID`

Parameters	Description
`servlet`	The URI of the servlet that processes full CMC requests. The default value is `/ca/profileSubmitCMCFull`. For example: `servlet=/ca/profileSubmitCMCFull`

Table 21.1.

OCSP Request

The OCSP request utility, `OCSPClient`, creates an OCSP request conforming to RFC 2560, submits it to the OCSP server, and saves the OCSP response in a file.

1. Syntax

The `OCSPClient` tool has the following syntax:

```
OCSPClient host port dbdir nickname serial_number output times
```

Option	Description
host	Specifies hostname of the OCSP server.
port	Gives the port number of the OCSP server.
dbdir	Gives the location of the security databases (`cert8.db`, `key3.db`, and `secmod.db`) which contain the CA certificate that signed the certificate being checked.
nickname	Gives the CA certificate nickname.
serial_number	Gives the serial number of the certificate that's status is being checked.
output	Gives the path and file to which to print the DER-encoded OCSP response.
times	Specifies the number of times to submit the request.

Table 22.1.

PKCS #10 Client

The PKCS #10 utility, `PKCS10Client`, generates a 1024-bit RSA key pair in the security database, constructs a PKCS#10 certificate request with the public key, and outputs the request to a file.

PKCS #10 is a certification request syntax standard defined by RSA. A CA may support multiple types of certificate requests. The Certificate System CA supports KEYGEN, PKCS#10, CRMF, and CMC.

To get a certificate from the CA, the certificate request needs to be submitted to and approved by a CA agent. Once approved, a certificate is created for the request, and certificate attributes, such as extensions, are populated according to certificate profiles.

1. Syntax

The `PKCS10Client` tool has the following syntax:

```
PKCS10Client -p certDBPassword -d certDBDirectory -o outputFile -s subjectDN
```

Option	Description
p	Gives the password for the security databases.
d	Gives the path to the security databases.
o	Sets the path and filename to output the new PKCS #10 certificate.
s	Gives the subject DN of the certificate.

Table 23.1.

Bulk Issuance Tool

The `bulkissuance` utility sends a KEYGEN or a CRMF enrollment request to the bulk issuance interface of a CA to create certificates automatically. The `bulkissuance` utility does not generate the certificate request itself. It submits the content in the input file to the CA server's bulk issuance interface.

The bulk issuance interface is part of the agent interface of the CA. If the request is submitted through the agent interface, the request is processed, and the certificate is created immediately.

1. Syntax

The `bulkissuance` command has the following syntax:

```
bulkissuance -n rsa_nickname [-p password | -w passwordFile]
 [-d dbdir] [-v] [-V] -f inputFile hostname:[port]
```

Option	Description
`n`	Gives the agent certificate nickname.
`p`	Gives the certificate database password. Not used if the `-w` option is used.
`w`	*Optional.* Gives the path to the password file. Not used if the `-p` option is used.
`d`	*Optional.* Gives the path to the security databases.
`v`	*Optional.* Sets the operation in verbose mode.
`V`	*Optional.* Gives the version of the `bulkissuance` tool.
`f`	Gives the path and filename of the input file containing an HTTP request to send to the specified *hostname*.
hostname	Gives the hostname of the server to which to send the request.
port	*Optional.* Gives the port number of the server.

Table 24.1.

This utility requires an input file which includes the URI to the CA's bulk issuance interface and the certificate request.

Revocation Automation Utility

The `revoker` utility sends revocation requests to the CA agent interface to revoke certificates. To access the interface, revoker needs to have access to an agent certificate that is acceptable to the CA.

The `revoker` tool can do all of the following:

- Specify which certificate or a list of certificates to revoke by listing the hexadecimal serial numbers.
- Specify a revocation reason.
- Specify an invalidity date.
- Unrevoke a certificate that is currently on hold.

1. Syntax

The `revoker` utility has the following syntax:

```
revoker  -s serialNumber -n rsa_nickname [-p password
 | -w passwordFile] [-d dbdir] [-v] [-V] [-u] [-r reasoncode]
 [-i numberOfHours] hostname:[port]
```

Option	Description
`s`	Gives the serial numbers in hexadecimal of the certificates to revoke.
`n`	Gives the agent certificate nickname.
`p`	Gives the certificate database password. Not used if the `-w` option is used.
`w`	*Optional.* Gives the path to the password file. Not used if the `-p` option is used.
`d`	*Optional.* Gives the path to the security databases.
`v`	*Optional.* Sets the operation in verbose mode.
`V`	*Optional.* Gives the version of the `revoker` tool.
`u`	
`r`	Gives the reason to revoke the certificate. The following are the possible reasons:

Option	Description
	• *0* - Unspecified (default). • *1* - The key was compromised. • *2* - The CA key was compromised. • *3* - The affiliation of the user has changed. • *4* - The certificate has been superseded. • *5* - Cessation of operation. • *6* - The certificate is on hold.
`i`	Sets the invalidity date in hours from current time for when to revoke the certificate.
hostname	Gives the hostname of the server to which to send the request.
port	*Optional.* Gives the port number of the server.

Table 25.1.

Index