Red Hat Certificate System 10

Administration Guide

Updated for Red Hat Certificate System 10.4

Last Updated: 2023-08-25

Red Hat Certificate System 10 Administration Guide

Updated for Red Hat Certificate System 10.4

Florian Delehaye
Red Hat Customer Content Services
fdelehay@redhat.com

Marc Muehlfeld
Red Hat Customer Content Services

Petr Bokoč
Red Hat Customer Content Services

Filip Hanzelka
Red Hat Customer Content Services

Tomáš Čapek
Red Hat Customer Content Services

Ella Deon Ballard
Red Hat Customer Content Services

Legal Notice

Abstract

This manual covers all aspects of installing, configuring, and managing Certificate System subsystems. It also covers management tasks such as adding users; requesting, renewing, and revoking certificates; publishing CRLs; and managing smart cards. This guide is intended for Certificate System administrators.

Table of Contents

CHAPTER 1. OVERVIEW OF RED HAT CERTIFICATE SYSTEM SUBSYSTEMS

Every common PKI operation – issuing, renewing and revoking certificates; archiving and recovering keys; publishing CRLs and verifying certificate status – are carried out by interoperating subsystems within Red Hat Certificate System. The functions of each individual subsystem and the way that they work together to establish a robust and local PKI is described in this chapter.

1.1. USES FOR CERTIFICATES

The purpose of certificates is to establish trust. Their usage varies depending on the kind of trust they are used to ensure. Some kinds of certificates are used to verify the identity of the presenter; others are used to verify that an object or item has not been tampered with.

For information on how certificates are used, the types of certificates, or how certificates establish identities and relationships, see the *Certificates and Authentication* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

1.2. A REVIEW OF CERTIFICATE SYSTEM SUBSYSTEMS

Red Hat Certificate System provides five different subsystems, each focusing on different aspects of a PKI deployment. These subsystems work together to create a public key infrastructure (PKI). Depending on what subsystems are installed, a PKI can function as a token management system (TMS) or a non token management system. For descriptions of the subsystems and TMS and non-TMS environments, see the *A Review of Certificate System Subsystems* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

Enterprise Security Client

The Enterprise Security Client is not a subsystem since it does not perform any operations with certificates, keys, or tokens. The Enterprise Security Client is a user interface which allows people to manage certificates on smart cards very easily. The Enterprise Security Client sends all token operations, such as certificate requests, to the token processing system (TPS), which then sends them to the certificate authority (CA). For more information, see For more information, see *Red Hat Certificate System Managing Smart Cards with the Enterprise Security Client*.

1.3. A LOOK AT MANAGING CERTIFICATES (NON-TMS)

A conventional PKI environment provides the basic framework to manage certificates stored in software databases. This is a *non-TMS* environment, since it does not manage certificates on smart cards. At a minimum, a non-TMS requires only a CA, but a non-TMS environment can use OCSP responders and KRA instances as well.

For information on this topic, see the following sections in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*:

- Managing Certificates
- Using a Single Certificate Manager
- Planning for Lost Keys: Key Archival and Recovery
- Balancing Certificate Request Processing
- Balancing Client OCSP Requests

1.4. A LOOK AT THE TOKEN MANAGEMENT SYSTEM (TMS)

Certificate System creates, manages, renews, and revokes certificates, and it also archives and recovers keys. For organizations which use smart cards, the Certificate System has a token management system – a collection of subsystems with established relationships – to generate keys and requests and receive certificates to be used for smart cards.

For information on this topic, see the following sections in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*:

- Working with Smart Cards (TMS)
- Using Smart Cards

1.5. RED HAT CERTIFICATE SYSTEM SERVICES

There are various different interfaces for managing certificates and subsystems, depending on the user type: administrators, agents, auditors, and end users. For an overview of the different functions that are performed through each interface, see the *User Interfaces* section.

PART I. RED HAT CERTIFICATE SYSTEM USER INTERFACES

CHAPTER 2. USER INTERFACES

There are different interfaces for managing certificates and subsystems, depending on the user's role: administrators, agents, auditors, and end users.

2.1. USER INTERFACES OVERVIEW

Administrators can use the following interfaces to securely interact with a completed Certificate System installation:

- The PKI command-line interface and other command-line utilities
- The PKI Console graphical interface
- The Certificate System web interface.

These interfaces require configuration prior to use for secure communication with the Certificate System server over TLS. Using these clients without proper configuration is not allowed. Some of these tools use TLS client authentication. When required, their required initialization procedure includes configuring this. Which interface is used depends on the administrator's preferences and functionality available. Common actions using these interfaces are described in the remainder of the guide after this chapter.

By default, the PKI command-line interface uses the NSS database in the user's **~/.dogtag/nssdb/** directory. Section 2.5.1.1, "pki CLI Initialization" provides detailed steps for initializing the NSS database with the administrator's certificate and key. Some examples of using the PKI command-line utility are described in Section 2.5.1.2, "Using "pki" CLI". Additional examples are shown through the rest of the guide.

Interfacing with Certificate System (as an administrator in other user roles) can be done using various command-line utilities to submit CMC requests, manage generated certificates, and so on. These are described briefly in Section 2.5, "Command Line Interfaces", such as Section 2.5.2, "AtoB". These utilities are utilized in later sections such as Section 5.2.1.2, "Creating a CSR Using **PKCS10Client**".

The Certificate System web interface allows administrative access through the Firefox web browser. Section 2.4.1, "Browser Initialization" describes instructions about configuring the client authentication. Other sections in Section 2.4, "Web Interface" describe using the web interface of Certificate System.

The Certificate System's PKI Console is a graphical interface. **Please note that it is being deprecated.** Section 2.3.1, "**pkiconsole** Initialization" describes how to initialize this console interface. Section 2.3.2, "Using **pkiconsole** for CA, OCSP, KRA, and TKS Subsystems" gives an overview of using it. Later sections, such as Section 3.2.2, "Managing Certificate Enrollment Profiles Using the Java-based Administration Console" go into greater detail for specific operations.

NOTE

To terminate a PKI Console session, click the **Exit** button. To terminate a web browser session, close the browser. A command-line utility terminates itself as soon as it performs the action and returns to the prompt, so no action is needed on the administrator's part to terminate the session.

2.2. CLIENT NSS DATABASE INITIALIZATION

On Red Hat Certificate System, certain interfaces may need to access the server using TLS client certificate authentication (mutual authentication). Before performing server-side admin tasks, you need to:

1. Prepare an NSS database for the client. This can be a new database or an existing one.
2. Import the CA certificate chain and trust them.
3. Have a certificate and corresponding key. They can be generated in the NSS database or imported from somewhere else, such as from a PKCS #12 file.

Based on the utility, you need to initialize the NSS database accordingly. See:

- Section 2.5.1.1, "pki CLI Initialization"
- Section 2.3.1, "**pkiconsole** Initialization"
- Section 2.4.1, "Browser Initialization"

2.3. GRAPHICAL INTERFACE

IMPORTANT

pkiconsole is being deprecated.

The Certificate System console,**pkiconsole**, is a graphical interface that is designed for users with the Administrator role privilege to manage the subsystem itself. This includes adding users, configuring logs, managing profiles and plug-ins, and the internal database, among many other functions. This utility communicates with the Certificate System server via TLS using client-authentication and can be used to manage the server remotely.

2.3.1. pkiconsole Initialization

To use the **pkiconsole** interface for the first time, specify a new password and use the following command:

```
$ pki -c password -d ~/.redhat-idm-console client-init
```

This command creates a new client NSS database in the **~/.redhat-idm-console/** directory.

To import the CA certificate into the PKI client NSS database, see the *Importing a certificate into an NSS Database* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

To request a new client certificate, see Chapter 5, *Requesting, Enrolling, and Managing Certificates*.

Execute the following command to extract the admin client certificate from the **.p12** file:

```
$ openssl pkcs12 -in file -clcerts -nodes -nokeys -out file.crt
```

Validate and import the admin client certificate as described in the *Managing Certificate/Key Crypto Token* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*:

```
$ PKICertImport -d ~/.redhat-idm-console -n "nickname" -t ",," -a -i file.crt -u C
```

IMPORTANT

Make sure all intermediate certificates and the root CA certificate have been imported before importing the CA admin client certificate.

To import an existing client certificate and its key into the client NSS database:

```
$ pki -c password -d ~/.redhat-idm-console pkcs12-import --pkcs12-file file --pkcs12-password
pkcs12-password
```

Verify the client certificate with the following command:

```
$ certutil -V -u C -n "nickname" -d ~/.redhat-idm-console
```

2.3.2. Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems

The Java console is used by four subsystems: the CA, OCSP, KRA, and TKS. The console is accessed using a locally-installed **pkiconsole** utility. It can access any subsystem because the command requires the host name, the subsystem's administrative TLS port, and the specific subsystem type.

```
pkiconsole https://server.example.com:admin_port/subsystem_type
```

If DNS is not configured, you can use an IPv4 or IPv6 address to connect to the console. For example:

```
https://192.0.2.1:8443/ca
https://[2001:DB8::1111]:8443/ca
```

This opens a console, as in Figure 2.1, “Certificate System Console” .

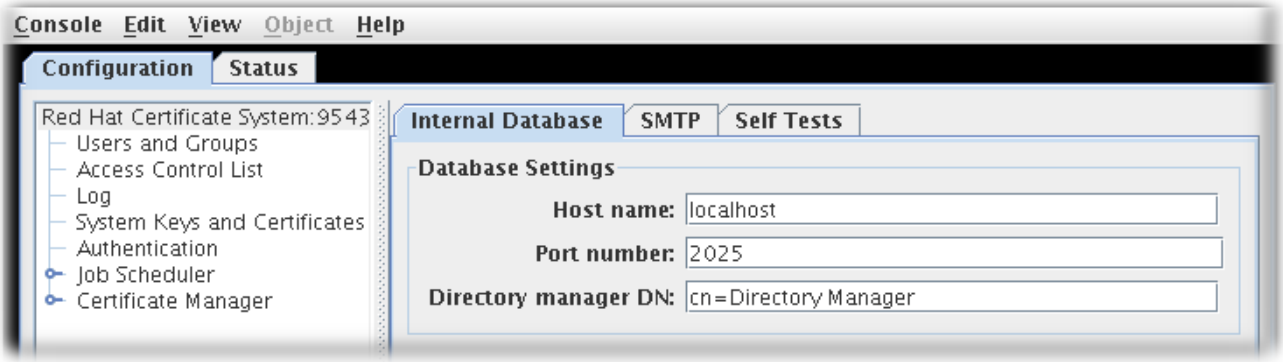


Figure 2.1. Certificate System Console

The **Configuration** tab controls all of the setup for the subsystem, as the name implies. The choices available in this tab are different depending on which subsystem type the instance is; the CA has the most options since it has additional configuration for jobs, notifications, and certificate enrollment authentication.

All subsystems have four basic options:

- Users and groups
- Access control lists

- Log configuration
- Subsystem certificates (meaning the certificates issued to the subsystem for use, for example, in the security domain or audit signing)

The **Status** tab shows the logs maintained by the subsystem.

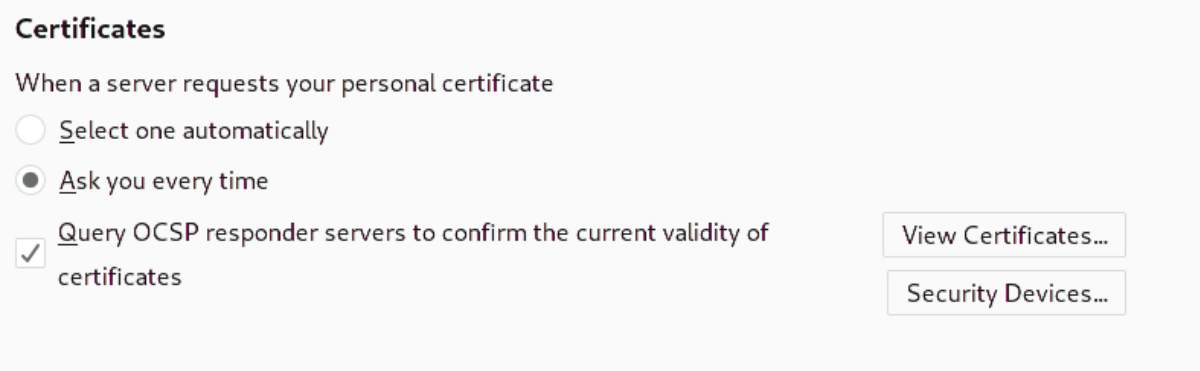
2.4. WEB INTERFACE

2.4.1. Browser Initialization

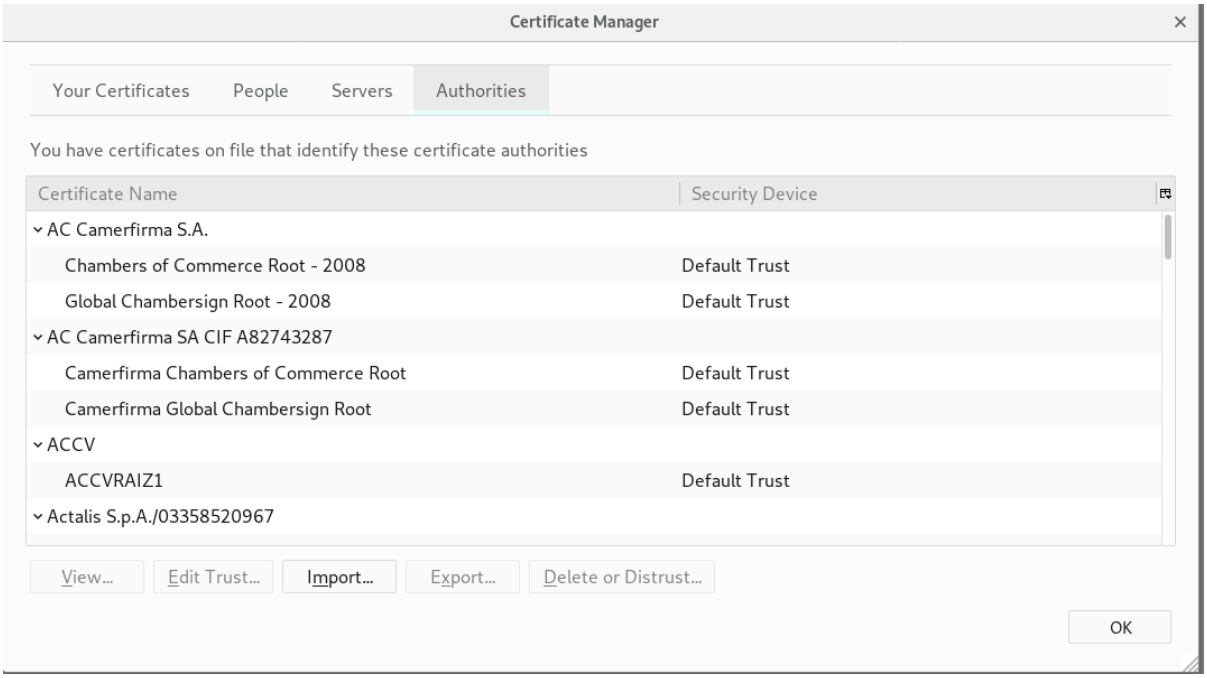
This section explains browser initialization for Firefox to access PKI services.

Importing a CA Certificate

1. Click **Menu → Preferences → Privacy & Security → View certificates**.



2. Select the **Authorities** tab and click the **Import** button.

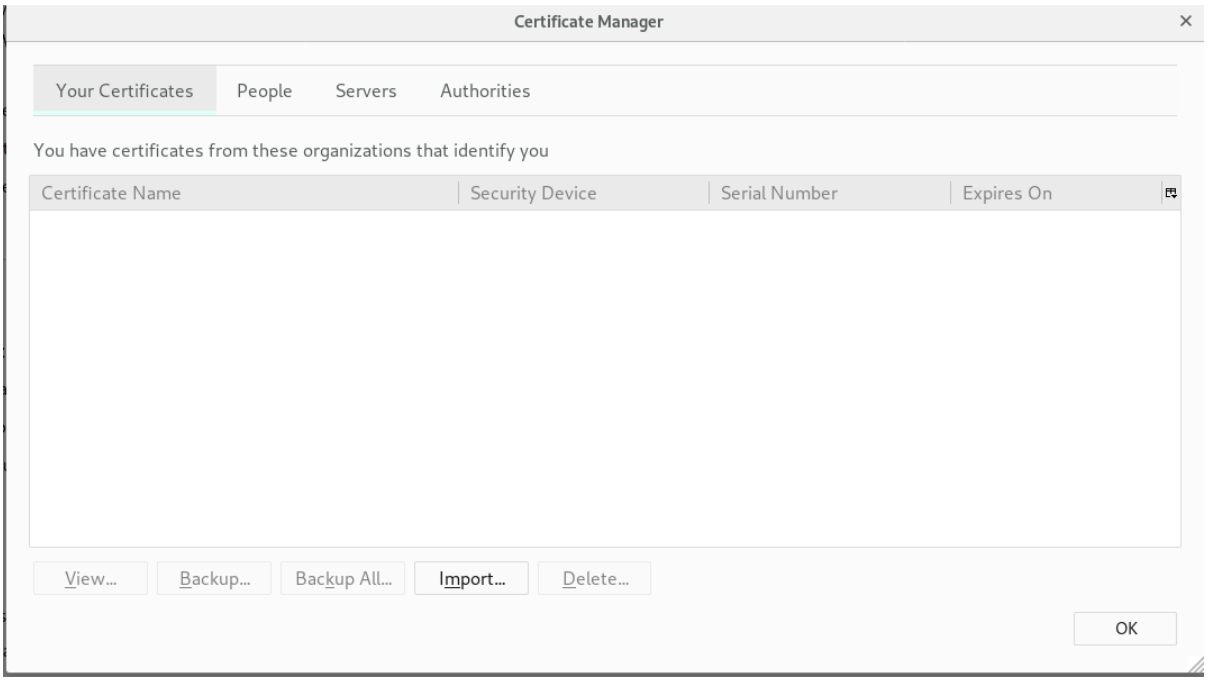


3. Select the **ca.crt** file and click **Import**.

Importing a Client Certificate

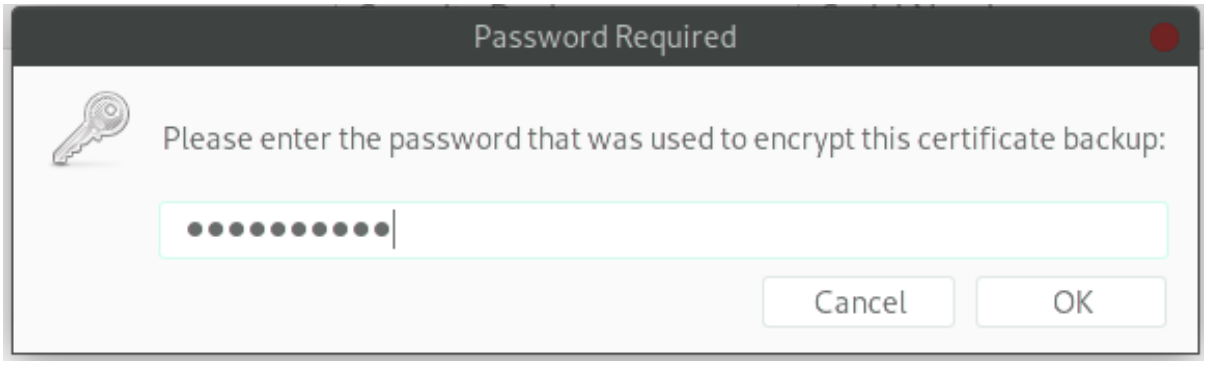
1. Click **Options → Preferences → Privacy & Security → View certificates**.

2. Select the **Your Certificates** tab.



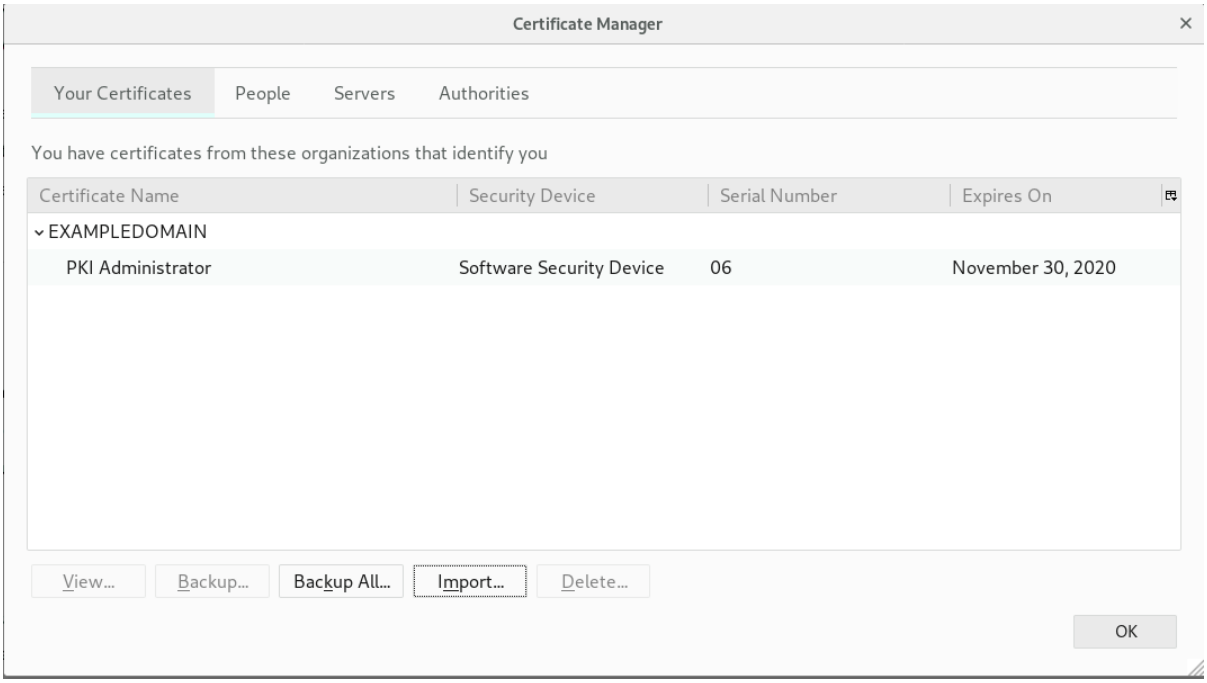
3. Click on **Import** and select the client p12 file, such as **ca_admin_cert.p12**.

4. Enter the password for the client certificate on the prompt.



5. Click **OK**.

6. Verify that an entry is added under **Your Certificates**.

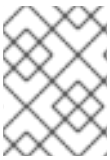


Accessing the Web Console

You can access the PKI services by opening **https://*host_name*:*port*** in your browser.

2.4.2. The Administrative Interfaces

The all subsystems use HTML-based administrative interface. It is accessed by entering the host name and secure port as the URL, authenticating with the administrator's certificate, and clicking the appropriate **Administrators** link.



NOTE

There is a single TLS port for all subsystems which is used for both administrator and agent services. Access to those services is restricted by certificate-based authentication.

The HTML admin interface is much more limited than the Java console; the primary administrative function is managing the subsystem users.

The TPS only allows operations to manage users for the TPS subsystem. However, the TPS admin page can also list tokens and display all activities (including normally-hidden administrative actions) performed on the TPS.

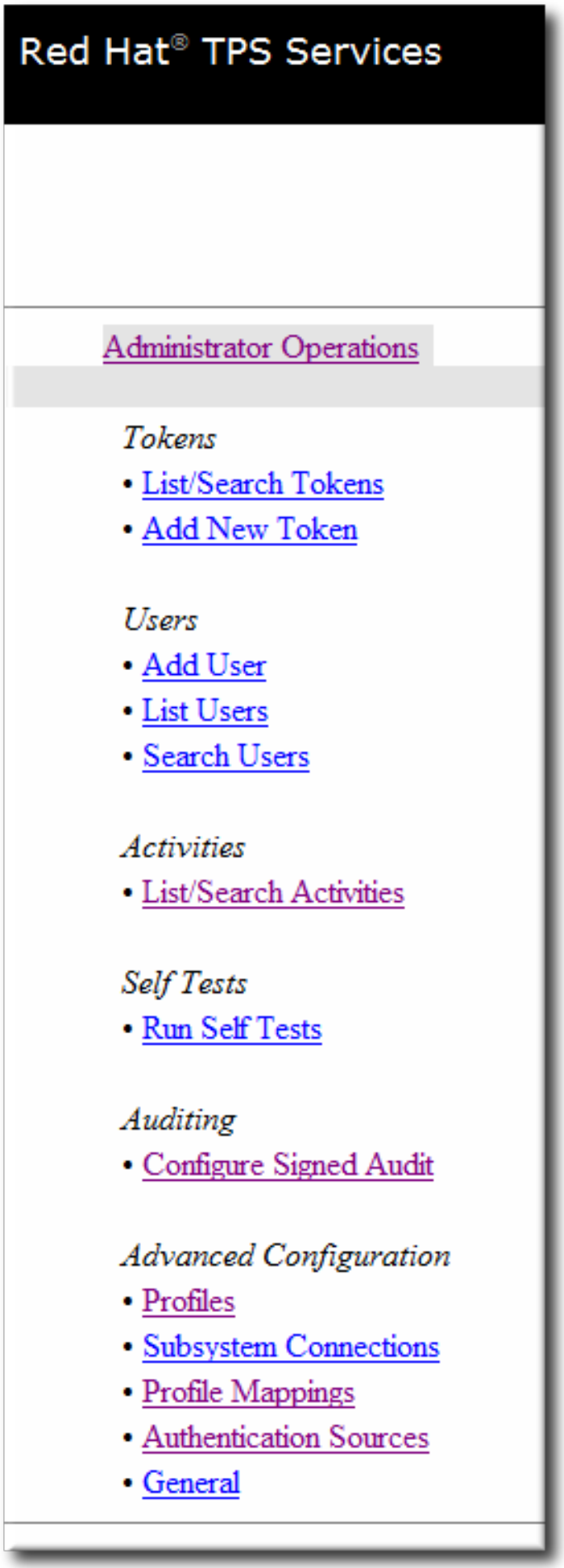

Red Hat® TPS Services
Administrator Operations
Tokens
• List/Search Tokens
• Add New Token
Users
• Add User
• List Users
• Search Users
Activities
• List/Search Activities
Self Tests
• Run Self Tests
Auditing
• Configure Signed Audit
Advanced Configuration
• Profiles
• Subsystem Connections
• Profile Mappings
• Authentication Sources
• General

Figure 2.2. TPS Admin Page

2.4.3. Agent Interfaces

The agent services pages are where almost all of the certificate and token management tasks are performed. These services are HTML-based, and agents authenticate to the site using a special agent certificate.

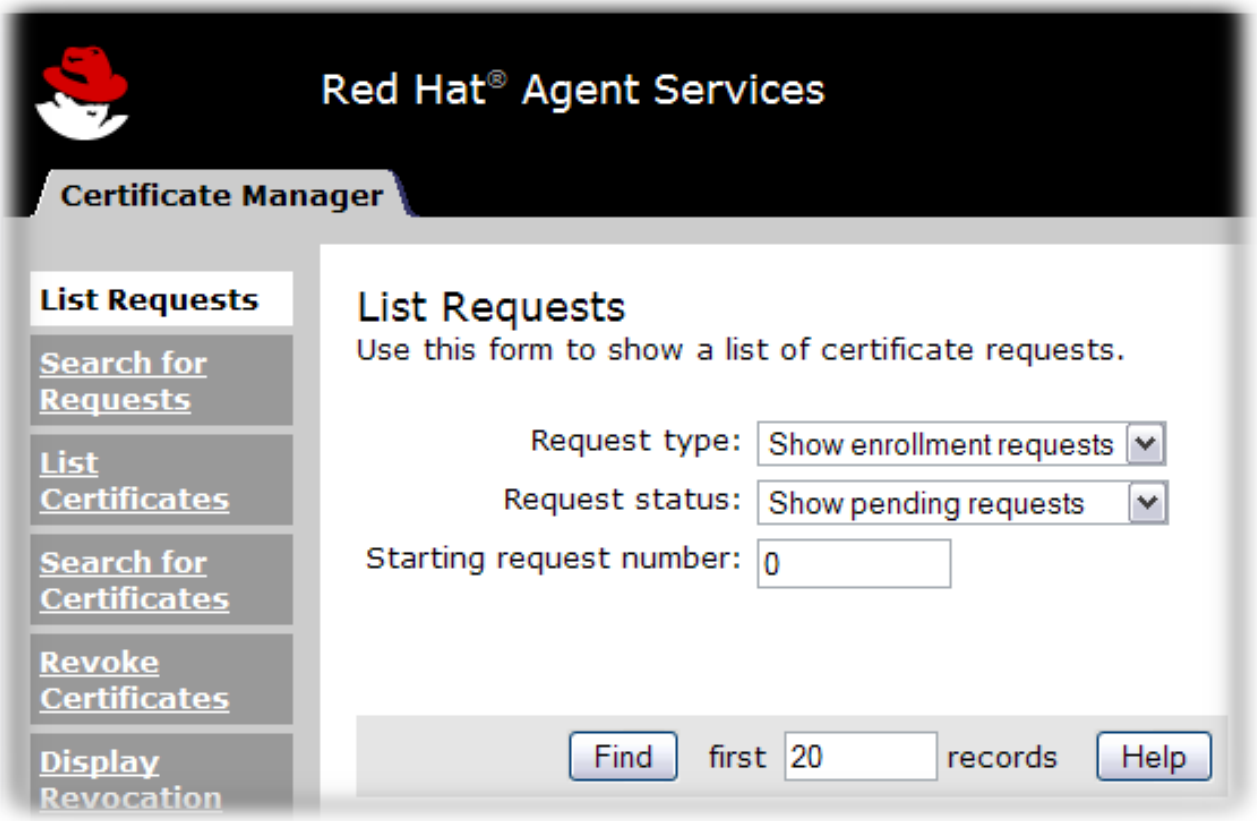


Figure 2.3. Certificate Manager's Agent Services Page

The operations vary depending on the subsystem:

- The Certificate Manager agent services include approving certificate requests (which issues the certificates), revoking certificates, and publishing certificates and CRLs. All certificates issued by the CA can be managed through its agent services page.
- The TPS agent services, like the CA agent services, manages all of the tokens which have been formatted and have had certificates issued to them through the TPS. Tokens can be enrolled, suspended, and deleted by agents. Two other roles (operator and admin) can view tokens in web services pages, but cannot perform any actions on the tokens.
- KRA agent services pages process key recovery requests, which set whether to allow a certificate to be issued reusing an existing key pair if the certificate is lost.
- The OCSP agent services page allows agents to configure CAs which publish CRLs to the OCSP, to load CRLs to the OCSP manually, and to view the state of client OCSP requests.

The TKS is the only subsystem without an agent services page.

2.4.4. End User Pages

The CA and TPS both process direct user requests in some way. That means that end users have to have a way to connect with those subsystems. The CA has end-user, or *end-entities*, HTML services. The TPS uses the Enterprise Security Client.

The end-user services are accessed over standard HTTP using the server's host name and the standard port number; they can also be accessed over HTTPS using the server's host name and the specific end-entities TLS port.

For CAs, each type of TLS certificate is processed through a specific online submission form, called a *profile*. There are about two dozen certificate profiles for the CA, covering all sorts of certificates — user TLS certificates, server TLS certificates, log and file signing certificates, email certificates, and every kind of subsystem certificate. There can also be custom profiles.

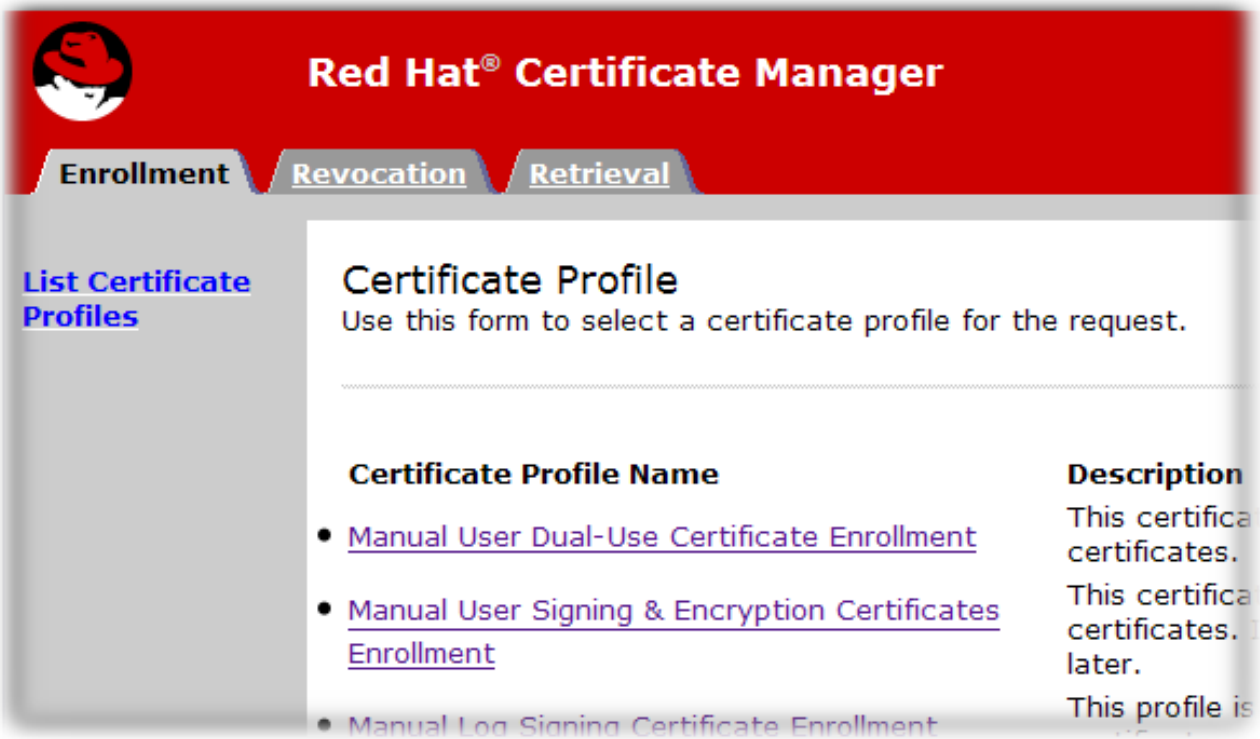


Figure 2.4. Certificate Manager's End-Entities Page

End users retrieve their certificates through the CA pages when the certificates are issued. They can also download CA chains and CRLs and can revoke or renew their certificates through those pages.

2.5. COMMAND LINE INTERFACES

This section discusses command-line utilities.

2.5.1. "pki" CLI

The **pki** command-line interface (CLI) provides access to various services on the server using the REST interface (see the *REST Interface* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*. The CLI can be invoked as follows:

```
$ pki [CLI options] <command> [command parameters]
```

Note that the CLI options must be placed before the command, and the command parameters after the command.

2.5.1.1. pki CLI Initialization

To use the command line interface for the first time, specify a new password and use the following command:

```
$ pki -c <password> client-init
```

This will create a new client NSS database in the **~/.dogtag/nssdb** directory. The password must be specified in all CLI operations that uses the client NSS database. Alternatively, if the password is stored in a file, you can specify the file using the **-C** option. For example:

```
$ pki -C password_file client-init
```

To import the CA certificate into the client NSS database refer to the *Importing a certificate into an NSS Database* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

Some commands may require client certificate authentication. To import an existing client certificate and its key into the client NSS database, specify the PKCS #12 file and the password, and execute the following command:

Execute the following command to extract the admin client certificate from the **.p12** file:

```
$ openssl pkcs12 -in file -clcerts -nodes -nokeys -out file.crt
```

Validate and import the admin client certificate as described in the *Managing Certificate/Key Crypto Token* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*:

```
$ PKICertImport -d ~/.dogtag/nssdb -n "nickname" -t ",," -a -i file.crt -u C
```

IMPORTANT

Make sure all intermediate certificates and the root CA certificate have been imported before importing the CA admin client certificate.

To import an existing client certificate and its key into the client NSS database, specify the PKCS #12 file and the password, and execute the following command:

```
$ pki -c <password> pkcs12-import --pkcs12-file <file> --pkcs12-password <password>
```

Verify the client certificate with the following command:

```
certutil -V -u C -n "nickname" -d ~/.dogtag/nssdb
```

2.5.1.2. Using "pki" CLI

The command line interface supports a number of commands organized in a hierarchical structure. To list the top-level commands, execute the **pki** command without any additional commands or parameters:

```
$ pki
```

Some commands have subcommands. To list them, execute **pki** with the command name and no additional options. For example:

```
$ pki ca
```

```
$ pki ca-cert
```

To view command usage information, use the **--help** option:

```
$ pki --help
```

```
$ pki ca-cert-find --help
```

To view manual pages, specify the command line **help** command:

```
$ pki help
```

```
$ pki help ca-cert-find
```

To execute a command that does not require authentication, specify the command and its parameters (if required), for example:

```
$ pki ca-cert-find
```

To execute a command that requires client certificate authentication, specify the certificate nickname, the client NSS database password, and optionally the server URL:

```
$ pki -U <server URL> -n <nickname> -c <password> <command> [command parameters]
```

For example:

```
$ pki -n jsmith -c password ca-user-find ...
```

By default, the CLI communicates with the server at **http://*local_host_name*:8080**. To communicate with a server at a different location, specify the URL with the **-U** option, for example:

```
$ pki -U https://server.example.com:8443 -n jsmith -c password ca-user-find
```

2.5.2. AtoB

The AtoB utility decodes the Base64-encoded certificates to their binary equivalents. For example:

```
$ AtoB input.ascii output.bin
```

For further details, more options, and additional examples, see the AtoB(1) man page.

2.5.3. AuditVerify

The AuditVerify utility verifies integrity of the audit logs by validating the signature on log entries.

Example:

```
$ AuditVerify -d ~jsmith/auditVerifyDir -n Log Signing Certificate -a ~jsmith/auditVerifyDir/logListFile -
P "" -v
```

The example verifies the audit logs using the **Log Signing Certificate** (**-n**) in the **~jsmith/auditVerifyDir** NSS database (**-d**). The list of logs to verify (**-a**) are in the **~jsmith/auditVerifyDir/logListFile** file, comma-separated and ordered chronologically. The prefix (**-P**) to prepend to the certificate and key database file names is empty. The output is verbose (**-v**).

For further details, more options, and additional examples, see the AuditVerify(1) man page or Section 16.3.2, "Using Signed Audit Logs" .

2.5.4. BtoA

The BtoA utility encodes binary data in Base64. For example:

```
$ BtoA input.bin output.ascii
```

For further details, more options, and additional examples, see the BtoA(1) man page.

2.5.5. CMCRequest

The CMCRequest utility creates a certificate issuance or revocation request. For example:

```
$ CMCRequest example.cfg
```

NOTE

All options to the **CMCRequest** utility are specified as part of the configuration filed passed to the utility. See the CMCRequest(1) man page for configuration file options and further information. Also see 4.3. Requesting and Receiving Certificates Using CMC and Section 7.2.1, "Revoking a Certificate Using **CMCRequest**".

2.5.6. CMCRevoke

Legacy. Do not use.

2.5.7. CMCSharedToken

The CMCSharedToken utility encrypts a user passphrase for shared-secred CMC requests. For example:

```
$ CMCSharedToken -d . -p myNSSPassword -s "shared_passphrase" -o cmcSharedTok2.b64 -n
"subsystemCert cert-pki-tomcat"
```

The shared passphrase (**-s**) is encrypted and stored in the **cmcSharedtok2.b64** file (**-o**) using the certificate named **subsystemCert cert-pki-tomcat** (**-n)** found in the NSS database in the current directory (**-d**). The default security token **internal** is used (as **-h** is not specified) and the token password of **myNSSPassword** is used for accessing the token.

For further details, more options, and additional examples, see the CMCSharedtoken(1) man page and also Section 7.2.1, "Revoking a Certificate Using **CMCRequest**".

2.5.8. CRMFPopClient

The **CRMFPopClient** utility is Certificate Request Message Format (CRMF) client using NSS databases and supplying Proof of Possession.

Example:

```
$ CRMFPopClient -d . -p password -n "cn=subject_name" -q POP_SUCCESS -b kra.transport -w
"AES/CBC/PKCS5Padding" -t false -v -o /user_or_entity_database_directory/example.csr
```

This example creates a new CSR with the **cn=subject_name** subject DN (**-n)**, NSS database in the current directory (**-d**), certificate to use for transport **kra.transport** (**-b**), the **AES/CBC/PKCS5Padding** key wrap algorithm verbose output is specified (**-v**) and the resulting CSR is written to the **/user_or_entity_database_directory/example.csr** file (**-o)**.

For further details, more options, and additional examples, see the output of the **CRMFPopClient --help** command and also Section 7.2.1, “Revoking a Certificate Using **CMCRequest**”.

2.5.9. HttpClient

The **HttpClient** utility is an NSS-aware HTTP client for submitting CMC requests.

Example:

```
$ HttpClient request.cfg
```

NOTE

All parameters to the **HttpClient** utility are stored in the **request.cfg** file. For further information, see the output of the **HttpClient --help** command.

2.5.10. OCSPClient

An Online Certificate Status Protocol (OCSP) client for checking the certificate revocation status.

Example:

```
$ OCSPClient -h server.example.com -p 8080 -d /etc/pki/pki-tomcat/alias -c "caSigningCert cert-pki-
ca" --serial 2
```

This example queries the **server.example.com** OCSP server (**-h**) on port **8080** (**-p**) to check whether the certificate signed by **caSigningcet cert-pki-ca** (**-c**) with serial number **2** (**--serial**) is valid. The NSS database in the **/etc/pki/pki-tomcat/alias** directory is used.

For further details, more options, and additional examples, see the output of the **OCSPClient --help** command.

2.5.11. PKCS10Client

The **PKCS10Client** utility creates a CSR in PKCS10 format for RSA and EC keys, optionally on an HSM.

Example:

```
$ PKCS10Client -d /etc/dirsrv/slapd-instance_name/ -p password -a rsa -l 2048 -o ~/ds.csr -n
"CN=$HOSTNAME"
```

This example creates a new RSA (**-a**) key with 2048 bits (**-l**) in the **/etc/dirsrv/slapd-instance_name/** directory (**-d** with database password **password** (**-p**). The output CSR is stored in the **~/ds.cfg** file (**-o**) and the certificate DN is **CN=$HOSTNAME** (**-n**).

For further details, more options, and additional examples, see the PKCS10Client(1) man page.

2.5.12. PrettyPrintCert

The PrettyPrintCert utility displays the contents of a certificate in a human-readable format.

Example:

```
$ PrettyPrintCert ascii_data.cert
```

This command parses the output of the **ascii_data.cert** file and displays its contents in human readable format. The output includes information like signature algorithm, exponent, modulus, and certificate extensions.

For further details, more options, and additional examples, see the PrettyPrintCert(1) man page.

2.5.13. PrettyPrintCrl

The PrettyPrintCrl utility displays the content of a CRL file in a human readable format.

Example:

```
$ PrettyPrintCrl ascii_data.crl
```

This command parses the output of the **ascii_data.crl** and displays its contents in human readable format. The output includes information, such as revocation signature algorithm, the issuer of the revocation, and a list of revoked certificates and their reason.

For further details, more options, and additional examples, see the PrettyPrintCrl(1) man page.

2.5.14. TokenInfo

The TokenInfo utility lists all tokens in an NSS database.

Example:

```
$ TokenInfo ./nssdb/
```

This command lists all tokens (HSMs, soft tokens, and so on) registered in the specified database directory.

For further details, more options, and additional examples, see the output of the **TokenInfo** command

2.5.15. tkstool

The **tkstool** utility is interacting with the token Key Service (TKS) subsystem.

Example:

```
$ tkstool -M -n new_master -d /var/lib/pki/pki-tomcat/alias -h token_name
```

This command creates a new master key (**-M**) named **new_master** (**-n**) in the **/var/lib/pki/pki-tomcat/alias** NSS database on the HSM **token_name**

For further details, more options, and additional examples, see the output of the **tkstool -H** command.

2.6. ENTERPRISE SECURITY CLIENT

The **Enterprise Security Client** is a tool for Red Hat Certificate System which simplifies managing smart cards. End users can use security tokens (smart cards) to store user certificates used for applications such as single sign-on access and client authentication. End users are issued the tokens containing certificates and keys required for signing, encryption, and other cryptographic functions.

The **Enterprise Security Client** is the third part of Certificate System's complete token management system. Two subsystems — the Token Key Service (TKS) and Token Processing System (TPS) — are used to process token-related operations. The **Enterprise Security Client** is the interface which allows the smart card and user to access the token management system.

After a token is enrolled, applications such as Mozilla Firefox and Thunderbird can be configured to recognize the token and use it for security operations, like client authentication and S/MIME mail. Enterprise Security Client provides the following capabilities:

- Supports JavaCard 2.1 or higher cards and Global Platform 2.01-compliant smart cards like Safenet's 330J smart card.
- Supports Gemalto TOP IM FIPS CY2 tokens, both the smart card and GemPCKey USB form factor key.
- Supports SafeNet Smart Card 650 (SC650).
- Enrolls security tokens so they are recognized by TPS.
- Maintains the security token, such as re-enrolling a token with TPS.
- Provides information about the current status of the token or tokens being managed.
- Supports server-side key generation so that keys can be archived and recovered on a separate token if a token is lost.

The Enterprise Security Client is a client for end users to register and manage keys and certificates on smart cards or tokens. This is the final component in the Certificate System token management system, with the Token Processing System (TPS) and Token Key Service (TKS).

The Enterprise Security Client provides the user interface of the token management system. The end user can be issued security tokens containing certificates and keys required for signing, encryption, and other cryptographic functions. To use the tokens, the TPS must be able to recognize and communicate with them. Enterprise Security Client is the method for the tokens to be enrolled.

Enterprise Security Client communicates over an SSL/TLS HTTP channel to the back end of the TPS. It is based on an extensible Mozilla XULRunner framework for the user interface, while retaining a legacy web browser container for a simple HTML-based UI.

After a token is properly enrolled, web browsers can be configured to recognize the token and use it for security operations. Enterprise Security Client provides the following capabilities:

- Allows the user to enroll security tokens so they are recognized by the TPS.
- Allows the user to maintain the security token. For example, Enterprise Security Client makes it possible to re-enroll a token with the TPS.
- Provides support for several different kinds of tokens through default and custom token profiles. By default, the TPS can automatically enroll user keys, device keys, and security officer keys; additional profiles can be added so that tokens for different uses (recognized by attributes such as the token CUID) can automatically be enrolled according to the appropriate profile.
- Provides information about the current status of the tokens being managed.

PART II. SETTING UP CERTIFICATE SERVICES

CHAPTER 3. MAKING RULES FOR ISSUING CERTIFICATES (CERTIFICATE PROFILES)

The Certificate System provides a customizable framework to apply policies for incoming certificate requests and to control the input request types and output certificate types; these are called *certificate profiles*. Certificate profiles set the required information for certificate enrollment forms in the Certificate Manager end-entities page. This chapter describes how to configure certificate profiles.

3.1. ABOUT CERTIFICATE PROFILES

A certificate profile defines everything associated with issuing a particular type of certificate, including the authentication method, the authorization method, the default certificate content, constraints for the values of the content, and the contents of the input and output for the certificate profile. Enrollment and renewal requests are submitted to a certificate profile and are then subject to the defaults and constraints set in that certificate profile. These constraints are in place whether the request is submitted through the input form associated with the certificate profile or through other means. The certificate that is issued from a certificate profile request contains the content required by the defaults with the information required by the default parameters. The constraints provide rules for what content is allowed in the certificate.

For details about using and customizing certificate profiles, see Section 3.2, “Setting up Certificate Profiles”.

The Certificate System contains a set of default profiles. While the default profiles are created to satisfy most deployments, every deployment can add their own new certificate profiles or modify the existing profiles.

- *Authentication.* In every certification profile can be specified an authentication method.
- *Authorization.* In every certification profile can be specified an authorization method.
- *Profile inputs.* Profile inputs are parameters and values that are submitted to the CA when a certificate is requested. Profile inputs include public keys for the certificate request and the certificate subject name requested by the end entity for the certificate.
- *Profile outputs.* Profile outputs are parameters and values that specify the format in which to provide the certificate to the end entity. Profile outputs are CMC responses which contain a PKCS#7 certificate chain, when the request was successful.
- *Certificate content.* Each certificate defines content information, such as the name of the entity to which it is assigned (the subject name), its signing algorithm, and its validity period. What is included in a certificate is defined in the X.509 standard. With version 3 of the X509 standard, certificates can also contain extensions. For more information about certificate extensions, see Section B.3, “Standard X.509 v3 Certificate Extension Reference” .

  All of the information about a certificate profile is defined in the **set** entry of the profile policy in the profile's configuration file. When multiple certificates are expected to be requested at the same time, multiple set entries can be defined in the profile policy to satisfy needs of each certificate. Each policy set consists of a number of policy rules and each policy rule describes a field in the certificate content. A policy rule can include the following parts:

  - *Profile defaults.* These are predefined parameters and allowed values for information contained within the certificate. Profile defaults include the validity period of the certificate, and what certificate extensions appear for each type of certificate issued.

- *Profile constraints.* Constraints set rules or policies for issuing certificates. Amongst other, profile constraints include rules to require the certificate subject name to have at least one CN component, to set the validity of a certificate to a maximum of 360 days, to define the allowed grace period for renewal, or to require that the **subjectaltname** extension is always set to **true**.

3.1.1. The Enrollment Profile

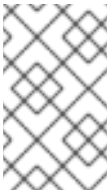
The parameters for each profile defining the inputs, outputs, and policy sets are listed in more detail in Table 11.1. Profile Configuration File Parameters in Red Hat Certificate System Planning, Installation and Deployment Guide.

A profile usually contains inputs, policy sets, and outputs, as illustrated in the **caUserCert** profile in Example 3.1, "Example caCMCUserCert Profile".

Example 3.1. Example caCMCUserCert Profile

The first part of a certificate profile is the description. This shows the name, long description, whether it is enabled, and who enabled it.

```
desc=This certificate profile is for enrolling user certificates by using the CMC certificate request
with CMC Signature authentication.
visible=true
enable=true
enableBy=admin
name=Signed CMC-Authenticated User Certificate Enrollment
```



NOTE

The missing **auth.instance_id=** entry in this profile means that with this profile, authentication is not needed to submit the enrollment request. However, manual approval by an authorized CA agent will be required to get an issuance.

Next, the profile lists all of the required inputs for the profile:

```
input.list=i1
input.i1.class_id=cmcCertReqInputImp
```

For the **caCMCUserCert** profile, this defines the certificate request type, which is CMC.

Next, the profile must define the output, meaning the format of the final certificate. The only one available is **certOutputImpl**, which results in CMC response to be returned to the requestor in case of success.

```
output.list=o1
output.o1.class_id=certOutputImpl
```

The last – largest – block of configuration is the policy set for the profile. Policy sets list all of the settings that are applied to the final certificate, like its validity period, its renewal settings, and the actions the certificate can be used for. The **policyset.list** parameter identifies the block name of the policies that apply to one certificate; the **policyset.userCertSet.list** lists the individual policies to apply.

For example, the sixth policy populates the Key Usage Extension automatically in the certificate, according to the configuration in the policy. It sets the defaults and requires the certificate to use those defaults by setting the constraints:

```
policyset.list=userCertSet
policyset.userCertSet.list=1,10,2,3,4,5,6,7,8,9
...
policyset.userCertSet.6.constraint.class_id=keyUsageExtConstraintImpl
policyset.userCertSet.6.constraint.name=Key Usage Extension Constraint
policyset.userCertSet.6.constraint.params.keyUsageCritical=true
policyset.userCertSet.6.constraint.params.keyUsageDigitalSignature=true
policyset.userCertSet.6.constraint.params.keyUsageNonRepudiation=true
policyset.userCertSet.6.constraint.params.keyUsageDataEncipherment=false
policyset.userCertSet.6.constraint.params.keyUsageKeyEncipherment=true
policyset.userCertSet.6.constraint.params.keyUsageKeyAgreement=false
policyset.userCertSet.6.constraint.params.keyUsageKeyCertSign=false
policyset.userCertSet.6.constraint.params.keyUsageCrlSign=false
policyset.userCertSet.6.constraint.params.keyUsageEncipherOnly=false
policyset.userCertSet.6.constraint.params.keyUsageDecipherOnly=false
policyset.userCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.userCertSet.6.default.name=Key Usage Default
policyset.userCertSet.6.default.params.keyUsageCritical=true
policyset.userCertSet.6.default.params.keyUsageDigitalSignature=true
policyset.userCertSet.6.default.params.keyUsageNonRepudiation=true
policyset.userCertSet.6.default.params.keyUsageDataEncipherment=false
policyset.userCertSet.6.default.params.keyUsageKeyEncipherment=true
policyset.userCertSet.6.default.params.keyUsageKeyAgreement=false
policyset.userCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.userCertSet.6.default.params.keyUsageCrlSign=false
policyset.userCertSet.6.default.params.keyUsageEncipherOnly=false
policyset.userCertSet.6.default.params.keyUsageDecipherOnly=false
...
```

3.1.2. Certificate Extensions: Defaults and Constraints

An extension configures additional information to include in a certificate or rules about how the certificate can be used. These extensions can either be specified in the certificate request or taken from the profile default definition and then enforced by the constraints.

A certificate extension is added or identified in a profile by adding the *default* which corresponds to the extension and sets default values, if the certificate extension is not set in the request. For example, the Basic Constraints Extension identifies whether a certificate is a CA signing certificate, the maximum number of subordinate CAs that can be configured under the CA, and whether the extension is critical (required):

```
policyset.caCertSet.5.default.name=Basic Constraints Extension Default
policyset.caCertSet.5.default.params.basicConstraintsCritical=true
policyset.caCertSet.5.default.params.basicConstraintsIsCA=true
policyset.caCertSet.5.default.params.basicConstraintsPathLen=-1
```

The extension can also set required values for the certificate request called *constraints*. If the contents of a request do not match the set constraints, then the request is rejected. The constraints generally correspond to the extension default, though not always. For example:

-

```
policyset.caCertSet.5.constraint.class_id=basicConstraintsExtConstraintImpl
policyset.caCertSet.5.constraint.name=Basic Constraint Extension Constraint
policyset.caCertSet.5.constraint.params.basicConstraintsCritical=true
policyset.caCertSet.5.constraint.params.basicConstraintsIsCA=true
policyset.caCertSet.5.constraint.params.basicConstraintsMinPathLen=-1
policyset.caCertSet.5.constraint.params.basicConstraintsMaxPathLen=-1
```

NOTE

To allow user supplied extensions to be embedded in the certificate requests and ignore the system-defined default in the profile, the profile needs to contain the User Supplied Extension Default, which is described in Section B.1.32, "User Supplied Extension Default".

3.1.3. Inputs and Outputs

Inputs set information that must be submitted to receive a certificate. This can be requester information, a specific format of certificate request, or organizational information.

The outputs configured in the profile define the format of the certificate that is issued.

In Certificate System, profiles are accessed by users through *enrollment forms* that are accessed through the end-entities pages. (Even clients, such as TPS, submit enrollment requests through these forms.) The inputs, then, correspond to fields in the enrollment forms. The outputs correspond to the information contained on the certificate retrieval pages.

3.2. SETTING UP CERTIFICATE PROFILES

In Certificate System, you can add, delete, and modify enrollment profiles:

- Using the PKI command-line interface
- Using the Java-based administration console

This section provides information on each method.

3.2.1. Managing Certificate Enrollment Profiles Using the PKI Command-line Interface

This section describes how to manage certificate profiles using the **pki** utility. For further details, see the pki-ca-profile(1) man page.

NOTE

Using the raw format is recommended. For details on each attribute and field of the profile, see the section Creating and Editing Certificate Profiles Directly on the File System in Red Hat Certificate System Planning, Installation and Deployment Guide.

3.2.1.1. Enabling and Disabling a Certificate Profile

Before you can edit a certificate profile, you must disable it. After the modification is complete, you can re-enable the profile.

NOTE

Only CA agents can enable and disable certificate profiles.

For example, to disable the **caCMCECserverCert** certificate profile:

```
# pki -c password -n caagent ca-profile-disable caCMCECserverCert
```

For example, to enable the **caCMCECserverCert** certificate profile:

```
# pki -c password -n caagent ca-profile-enable caCMCECserverCert
```

3.2.1.2. Creating a Certificate Profile in Raw Format

To create a new profile in raw format:

```
# pki -c password -n caadmin ca-profile-add profile_name.cfg --raw
```

NOTE

In raw format, specify the new profile ID as follows:

```
profileId=profile_name
```

3.2.1.3. Editing a Certificate Profile in Raw Format

CA administrators can edit a certificate profile in raw format without manually downloading the configuration file.

For example, to edit the **caCMCECserverCert** profile:

```
# pki -c password -n caadmin ca-profile-edit caCMCECserverCert
```

This command automatically downloads the profile configuration in raw format and opens it in the **VI** editor. When you close the editor, the profile configuration is updated on the server.

You do not need to restart the CA after editing a profile.

IMPORTANT

Before you can edit a profile, disable the profile. For details, see Section 3.2.1.1, “Enabling and Disabling a Certificate Profile”.

Example 3.2. Editing a Certificate Profile in RAW Format

For example, to edit the **caCMCserverCert** profile to accept multiple user-supplied extensions:

1. Disable the profile as a CA agent:

```
# pki -c password -n caagemt ca-profile-disable caCMCserverCert
```

2. Edit the profile as a CA administrator:

   a. Download and open the profile in the **VI** editor:

   ```
   # pki -c password -n caadmin ca-profile-edit caCMCserverCert
   ```

   b. Update the configuration to accept the extensions. For details, see Example B.3, "Multiple User Supplied Extensions in CSR".

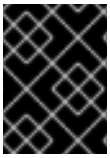
3. Enable the profile as a CA agent:

   ```
   # pki -c password -n caagent ca-profile-enable caCMCserverCert
   ```

3.2.1.4. Deleting a Certificate Profile

To delete a certificate profile:

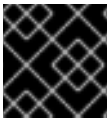
```
# pki -c password -n caadmin ca-profile-del profile_name
```



IMPORTANT

Before you can delete a profile, disable the profile. For details, see Section 3.2.1.1, "Enabling and Disabling a Certificate Profile".

3.2.2. Managing Certificate Enrollment Profiles Using the Java-based Administration Console



IMPORTANT

pkiconsole is being deprecated.

3.2.2.1. Creating Certificate Profiles through the CA Console

For security reasons, the Certificate Systems enforces separation of roles whereby an existing certificate profile can only be edited by an administrator after it was allowed by an agent. To add a new certificate profile or modify an existing certificate profile, perform the following steps as the administrator:

1. Log in to the Certificate System CA subsystem console.

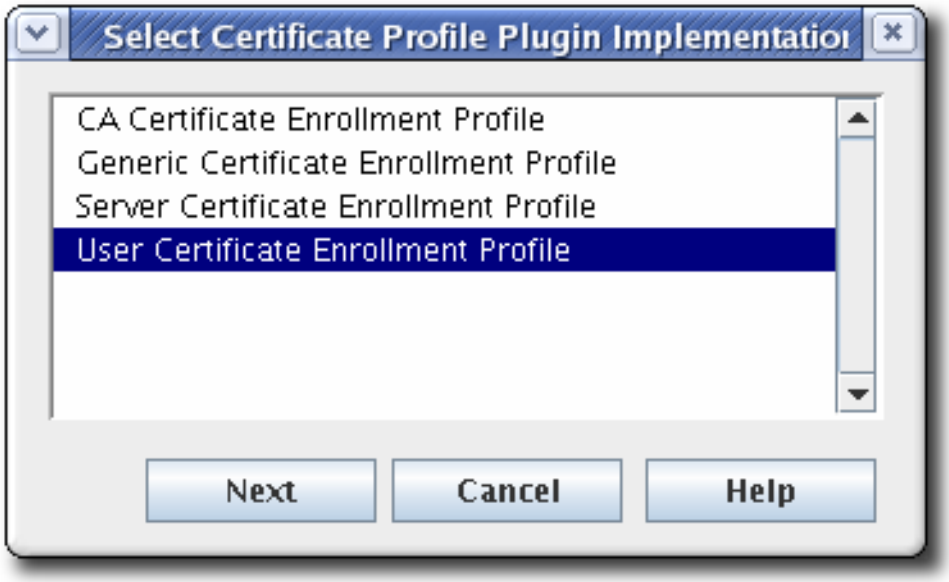
   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, select **Certificate Manager**, and then select **Certificate Profiles**.

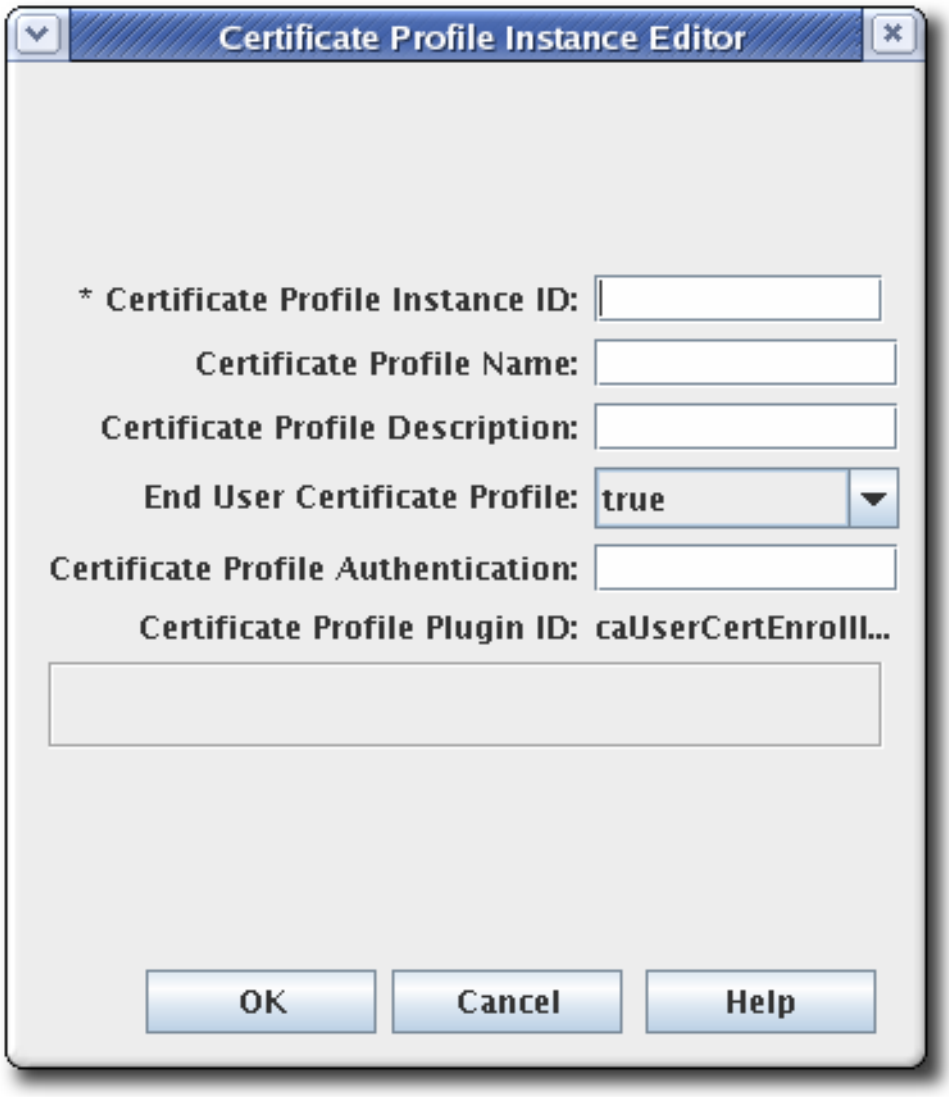
   The **Certificate Profile Instances Management** tab, which lists configured certificate profiles, opens.

3. To create a new certificate profile, click **Add**.

In the **Select Certificate Profile Plugin Implementation** window, select the type of certificate for which the profile is being created.



4. Fill in the profile information in the **Certificate Profile Instance Editor**.



- **Certificate Profile Instance ID**. This is the ID used by the system to identify the profile.

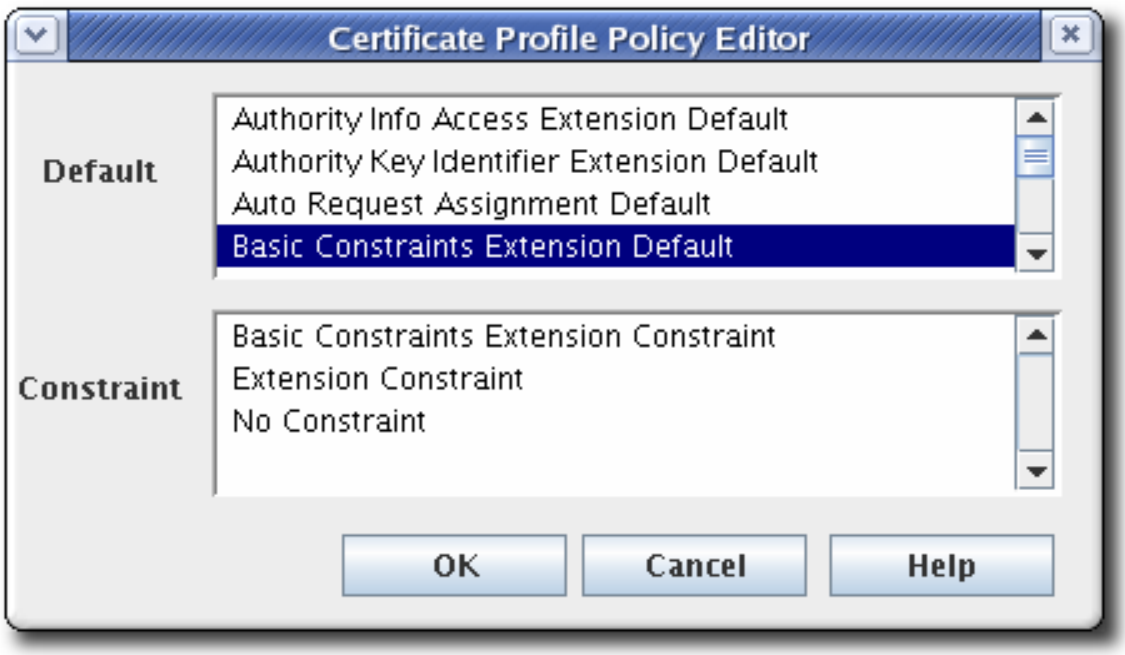
- **Certificate Profile Name**. This is the user-friendly name for the profile.
- **Certificate Profile Description**.
- **End User Certificate Profile**. This sets whether the request must be made through the input form for the profile. This is usually set to **true**. Setting this to **false** allows a signed request to be processed through the Certificate Manager's certificate profile framework, rather than through the input page for the certificate profile.
- **Certificate Profile Authentication**. This sets the authentication method. An automated authentication is set by providing the instance ID for the authentication instance. If this field is blank, the authentication method is agent-approved enrollment; the request is submitted to the request queue of the agent services interface.

  Unless it is for a TMS subsystem, administrators must select one of the following authentication plug-ins:

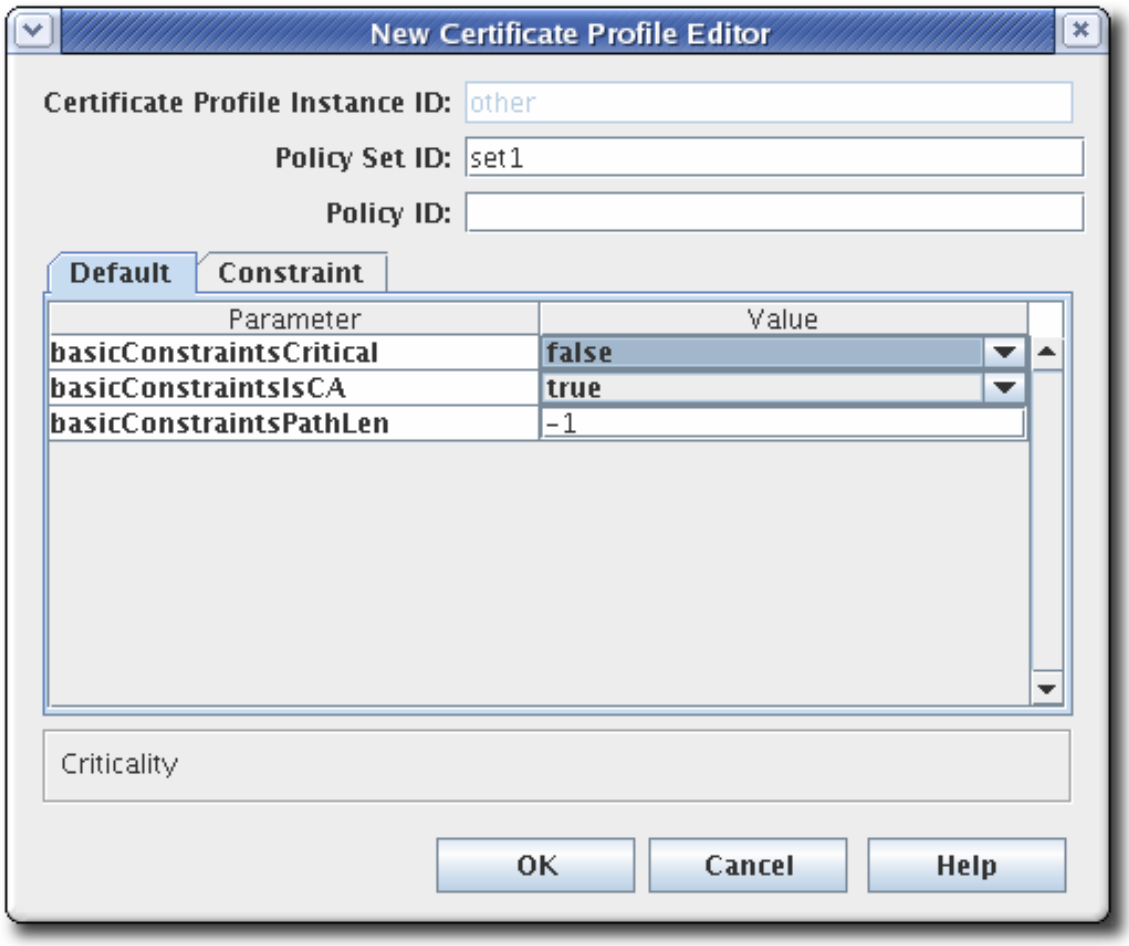
  - **CMCAuth**: Use this plug-in when a CA agent must approve and submit the enrollment request.
  - **CMCUserSignedAuth**: Use this plug-in to enable non-agent users to enroll own certificates.

5. Click **OK**. The plug-in editor closes, and the new profile is listed in the profiles tab.
6. Configure the policies, inputs, and outputs for the new profile. Select the new profile from the list, and click **Edit/View**.
7. Set up policies in the **Policies** tab of the **Certificate Profile Rule Editor** window. The **Policies** tab lists policies that are already set by default for the profile type.

   1. To add a policy, click **Add**.

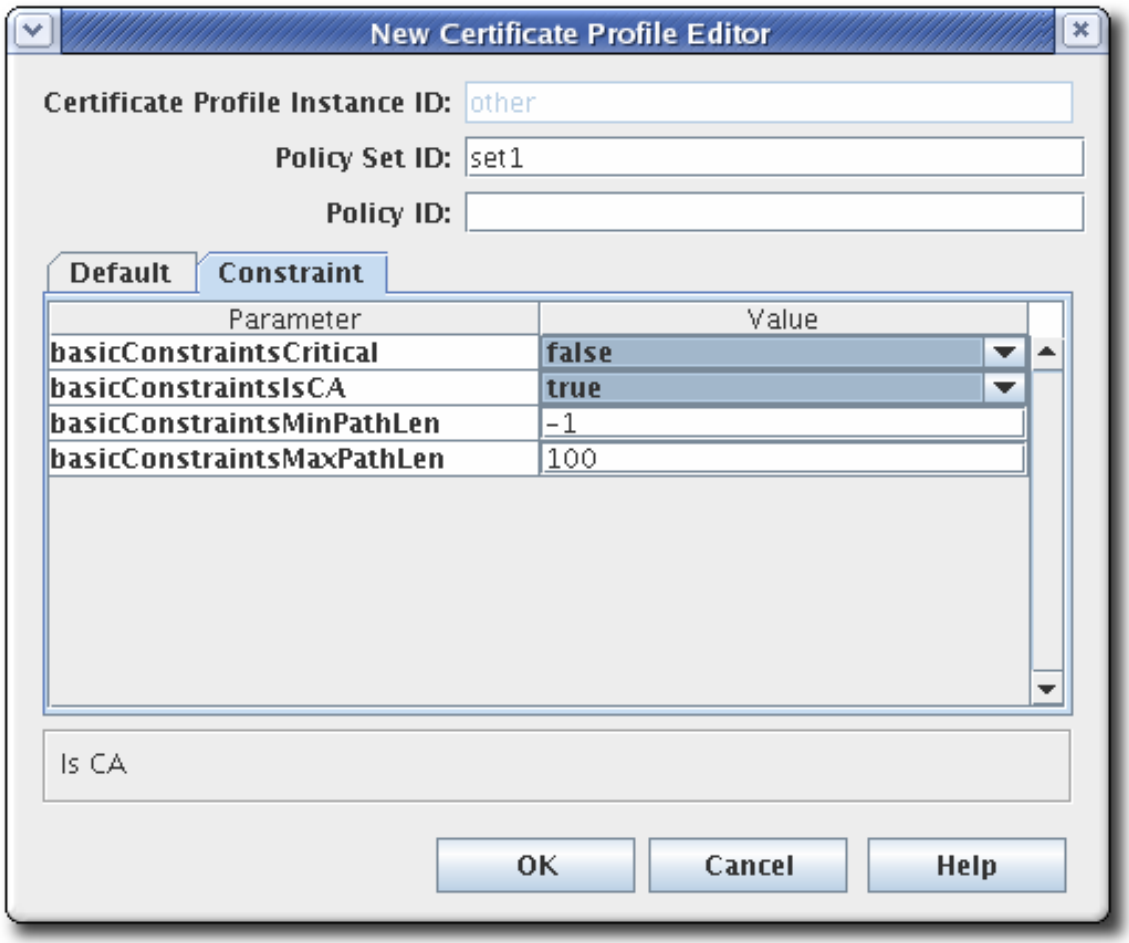


   2. Choose the default from the **Default** field, choose the constraints associated with that policy in the **Constraints** field, and click **OK**.



3. Fill in the policy set ID. When issuing dual key pairs, separate policy sets define the policies associated with each certificate. Then fill in the certificate profile policy ID, a name or identifier for the certificate profile policy.

4. Configure any parameters in the **Defaults** and **Constraints** tabs.



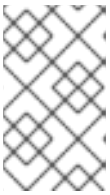
Defaults defines attributes that populate the certificate request, which in turn determines the content of the certificate. These can be extensions, validity periods, or other fields contained in the certificates. **Constraints** defines valid values for the defaults.

See Section B.1, "Defaults Reference" and Section B.2, "Constraints Reference" for complete details for each default or constraint.

To modify an existing policy, select a policy, and click **Edit**. Then edit the default and constraints for that policy.

To delete a policy, select the policy, and click **Delete**.

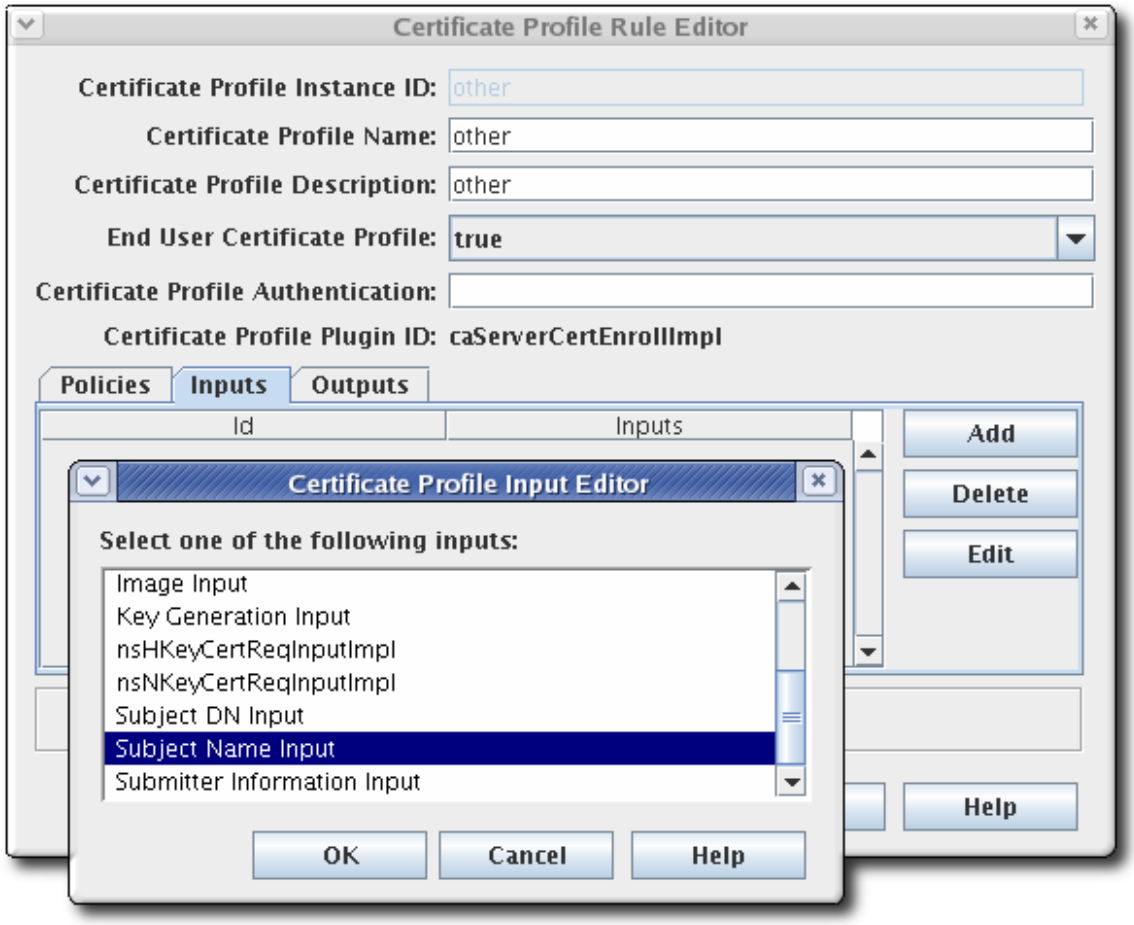
8. Set inputs in the **Inputs** tab of the **Certificate Profile Rule Editor** window. There can be more than one input type for a profile.



NOTE

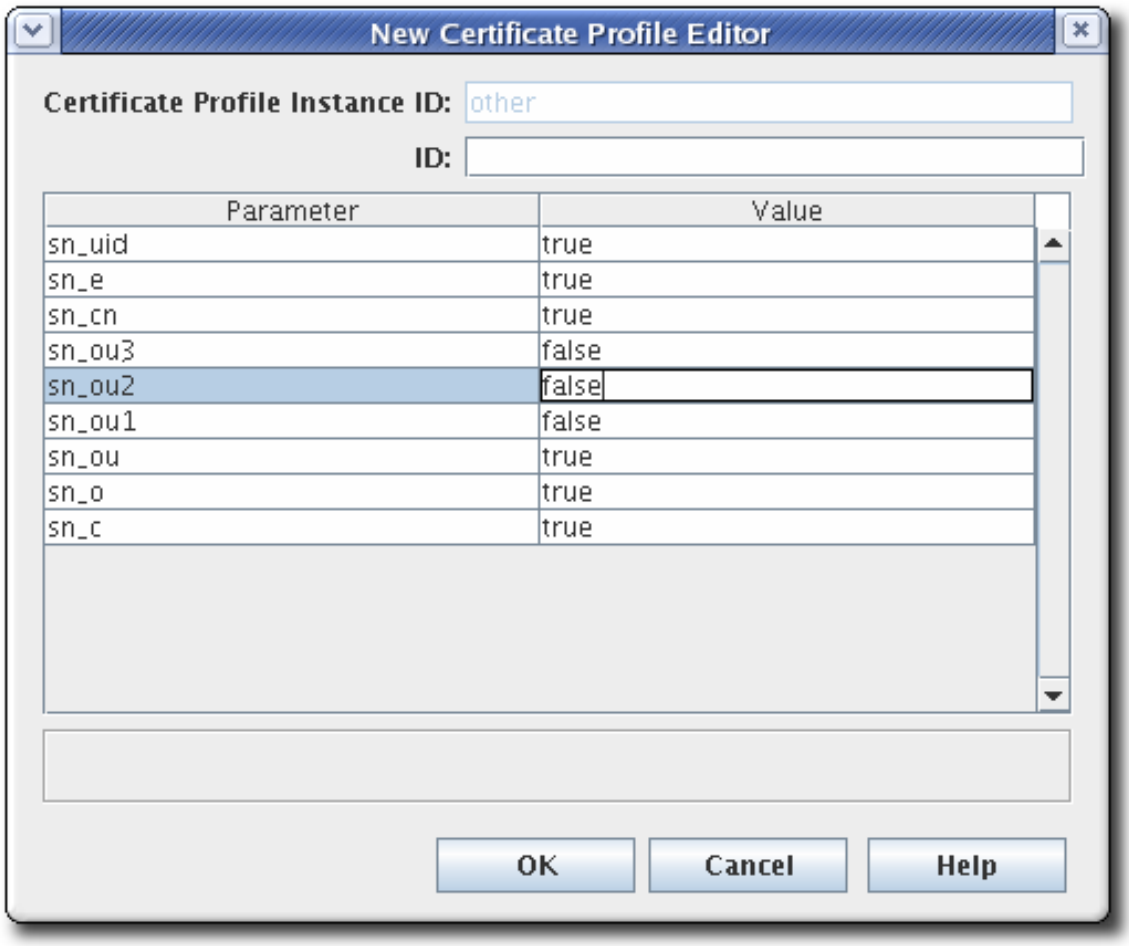
Unless you configure the profile for a TMS subsystem, select only **cmcCertReqInput** and delete other profiles by selecting them and clicking the **Delete** button.

1. To add an input, click **Add**.



2. Choose the input from the list, and click **OK**. See Section A.1, "Input Reference" for complete details of the default inputs.

3. The **New Certificate Profile Editor** window opens. Set the input ID, and click **OK**.



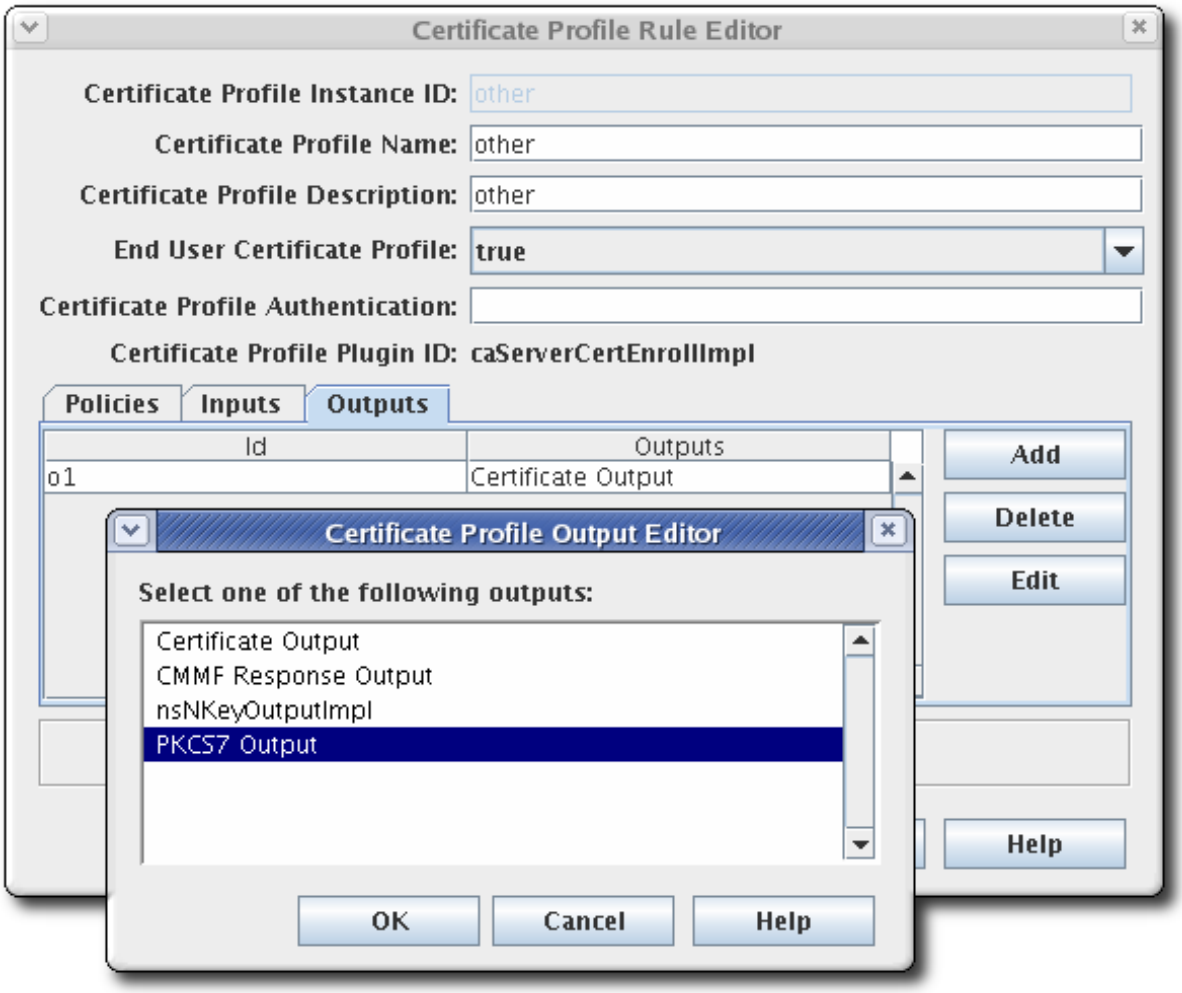
Inputs can be added and deleted. It is possible to select edit for an input, but since inputs have no parameters or other settings, there is nothing to configure.

To delete an input, select the input, and click **Delete**.

9. Set up outputs in the **Outputs** tab of the **Certificate Profile Rule Editor** window.

   Outputs must be set for any certificate profile that uses an automated authentication method; no output needs to be set for any certificate profile that uses agent-approved authentication. The Certificate Output type is set by default for all profiles and is added automatically to custom profiles.

   Unless you configure the profile for a TMS subsystem, select only **certOutput**.



Outputs can be added and deleted. It is possible to select edit for an output, but since outputs have no parameters or other settings, there is nothing to configure.

1. To add an output, click **Add**.
2. Choose the output from the list, and click **OK**.
3. Give a name or identifier for the output, and click **OK**.

   This output will be listed in the output tab. You can edit it to provide values to the parameters in this output.

To delete an output, select the output from list, and click **Delete**.

10. Restart the CA to apply the new profile.

    ```
    systemctl restart pki-tomcatd-nuxwdog@instance_name.service
    ```

11. After creating the profile as an administrator, a CA agent has to approve the profile in the agent services pages to enable the profile.

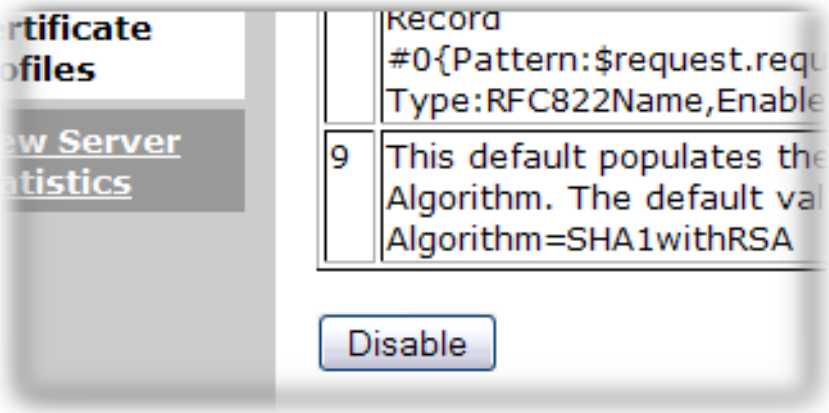
    1. Open the CA's services page.

       ```
       https://server.example.com:8443/ca/services
       ```

2. Click the **Manage Certificate Profiles** link. This page lists all of the certificate profiles that have been set up by an administrator, both active and inactive.

3. Click the name of the certificate profile to approve.

4. At the bottom of the page, click the **Enable** button.



NOTE

If this profile will be used with a TPS, then the TPS must be configured to recognized the profile type. This is in 11.1.4. Managing Smart Card CA Profiles in Red Hat Certificate System's Planning, Installation, and Deployment Guide.

Authorization methods for the profiles can only be added to the profile using the command line, as described in the section Creating and Editing Certificate Profiles Directly on the File System in Red Hat Certificate System Planning, Installation and Deployment Guide.

3.2.2.2. Editing Certificate Profiles in the Console

To modify an existing certificate profile:

1. Log into the agent services pages and disable the profile.

   Once a certificate profile is enabled by an agent, that certificate profile is marked enabled in the **Certificate Profile Instance Management** tab, and the certificate profile cannot be edited in any way through the console.

2. Log in to the Certificate System CA subsystem console.

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

3. In the **Configuration** tab, select **Certificate Manager**, and then select **Certificate Profiles**.

4. Select the certificate profile, and click **Edit/View**.

5. The **Certificate Profile Rule Editor** window appears. Many any changes to the defaults, constraints, inputs, or outputs.

NOTE

The profile instance ID cannot be modified.

If necessary, enlarge the window by pulling out one of the corners of the window.

6. Restart the CA to apply the changes.

7. In the agent services page, re-enable the profile.

NOTE

Delete any certificate profiles that will not be approved by an agent. Any certificate profile that appears in the **Certificate Profile Instance Management** tab also appears in the agent services interface. If a profile has already been enabled, it must be disabled by the agent before it can be deleted from the profile list.

3.2.3. Listing Certificate Enrollment Profiles

The following pre-defined certificate profiles are ready to use and set up in this environment when the Certificate System CA is installed. These certificate profiles have been designed for the most common types of certificates, and they provide common defaults, constraints, authentication methods, inputs, and outputs.

To list the available profiles on the command line, use the **pki** utility. For example:

```
# pki -c password -n caadmin ca-profile-find
-----------------
59 entries matched
-----------------
  Profile ID: caCMCserverCert
  Name: Server Certificate Enrollment using CMC
  Description: This certificate profile is for enrolling server certificates using CMC.

  Profile ID: caCMCECserverCert
  Name: Server Certificate wth ECC keys Enrollment using CMC
  Description: This certificate profile is for enrolling server certificates with ECC keys using CMC.

  Profile ID: caCMCECsubsystemCert
  Name: Subsystem Certificate Enrollment with ECC keys using CMC
  Description: This certificate profile is for enrolling subsystem certificates with ECC keys using CMC.

  Profile ID: caCMCsubsystemCert
  Name: Subsystem Certificate Enrollment using CMC
  Description: This certificate profile is for enrolling subsystem certificates using CMC.

  ...
----------------------------
Number of entries returned 20
```

For further details, see the pki-ca-profile(1) man page. Additional information can also be found at *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

3.2.4. Displaying Details of a Certificate Enrollment Profile

For example, to display a specific certificate profile, such as **caECFullCMCUserSignedCert**:

```
$ pki -c password -n caadmin ca-profile-show caECFullCMCUserSignedCert
```

```
-----------------------------------
Profile "caECFullCMCUserSignedCert"
-----------------------------------
  Profile ID: caECFullCMCUserSignedCert
  Name: User-Signed CMC-Authenticated User Certificate Enrollment
  Description: This certificate profile is for enrolling user certificates with EC keys by using the CMC
certificate request with non-agent user CMC authentication.

  Name: Certificate Request Input
  Class: cmcCertReqInputImpl

    Attribute Name: cert_request
    Attribute Description: Certificate Request
    Attribute Syntax: cert_request

  Name: Certificate Output
  Class: certOutputImpl

    Attribute Name: pretty_cert
    Attribute Description: Certificate Pretty Print
    Attribute Syntax: pretty_print

    Attribute Name: b64_cert
    Attribute Description: Certificate Base-64 Encoded
    Attribute Syntax: pretty_print
```

For example, to display a specific certificate profile, such as **caECFullCMCUserSignedCert**, in raw format:

```
$ pki -c password -n caadmin ca-profile-show caECFullCMCUserSignedCert --raw
#Wed Jul 25 14:41:35 PDT 2018
auth.instance_id=CMCUserSignedAuth
policyset.cmcUserCertSet.1.default.params.name=
policyset.cmcUserCertSet.4.default.class_id=authorityKeyIdentifierExtDefaultImpl
policyset.cmcUserCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.cmcUserCertSet.10.default.class_id=noDefaultImpl
policyset.cmcUserCertSet.10.constraint.name=Renewal Grace Period Constraint
output.o1.class_id=certOutputImpl

...
```

For further details, see the pki-ca-profile(1) man page.

3.3. DEFINING KEY DEFAULTS IN PROFILES

When creating certificate profiles, *the Key Default must be added before the Subject Key Identifier Default*. Certificate System processes the key constraints in the Key Default before creating or applying the Subject Key Identifier Default, so if the key has not been processed yet, setting the key in the subject name fails.

For example, an object-signing profile may define both defaults:

```
policyset.set1.p3.constraint.class_id=noConstraintImpl
policyset.set1.p3.constraint.name=No Constraint
policyset.set1.p3.default.class_id=subjectKeyIdentifierExtDefaultImpl
```

```
policyset.set1.p3.default.name=Subject Key Identifier Default
...
policyset.set1.p11.constraint.class_id=keyConstraintImpl
policyset.set1.p11.constraint.name=Key Constraint
policyset.set1.p11.constraint.params.keyType=RSA
policyset.set1.p11.constraint.params.keyParameters=1024,2048,3072,4096
policyset.set1.p11.default.class_id=userKeyDefaultImpl
policyset.set1.p11.default.name=Key Default
```

In the **policyset** list, then, the Key Default (**p11**) must be listed before the Subject Key Identifier Default (**p3**).

```
policyset.set1.list=p1,p2,p11,p3,p4,p5,p6,p7,p8,p9,p10
```

3.4. CONFIGURING PROFILES TO ENABLE RENEWAL

This section discusses how to set up profiles for certificate renewals. For more information on how to renew certificates, see Section 5.4, “Renewing Certificates”.

A profile that allows renewal is often accompanied by the **renewGracePeriodConstraint** entry. For example:

```
policyset.cmcUserCertSet.10.constraint.class_id=renewGracePeriodConstraintImpl
policyset.cmcUserCertSet.10.constraint.name=Renewal Grace Period Constraint
policyset.cmcUserCertSet.10.constraint.params.renewal.graceBefore=30
policyset.cmcUserCertSet.10.constraint.params.renewal.graceAfter=30
policyset.cmcUserCertSet.10.default.class_id=noDefaultImpl
policyset.cmcUserCertSet.10.default.name=No Default
```

3.4.1. Renewing Using the Same Key

A profile that allows the same key to be submitted for renewal has the ***allowSameKeyRenewal*** parameter set to **true** in the **uniqueKeyConstraint** entry. For example:

```
policyset.cmcUserCertSet.9.constraint.class_id=uniqueKeyConstraintImpl
policyset.cmcUserCertSet.9.constraint.name=Unique Key Constraint
policyset.cmcUserCertSet.9.constraint.params.allowSameKeyRenewal=true
policyset.cmcUserCertSet.9.default.class_id=noDefaultImpl
policyset.cmcUserCertSet.9.default.name=No Default
```

3.4.2. Renewal Using a New Key

To renew a certificate with a new key, use the same profile with a new key. Certificate System uses the **subjectDN** from the user signing certificate used to sign the request for the new certificate.

3.5. SETTING THE SIGNING ALGORITHMS FOR CERTIFICATES

The CA's signing certificate can sign the certificates it issues with any public key algorithm supported by the CA. For example, an ECC signing certificate can sign both ECC and RSA certificate requests as long as both ECC and RSA algorithms are supported by the CA. An RSA signing certificate can can sign a PKCS #10 request with EC keys, but may not be able to sign CRMF certificate requests with EC keys if the ECC module is not available for the CA to verify the CRMF proof of possession (POP).

ECC and RSA are public key encryption and signing algorithms. Both public key algorithms support different cipher suites, algorithms used to encrypt and decrypt data. Part of the function of the CA signing certificate is to issue and sign certificates using one of its supported cipher suites.

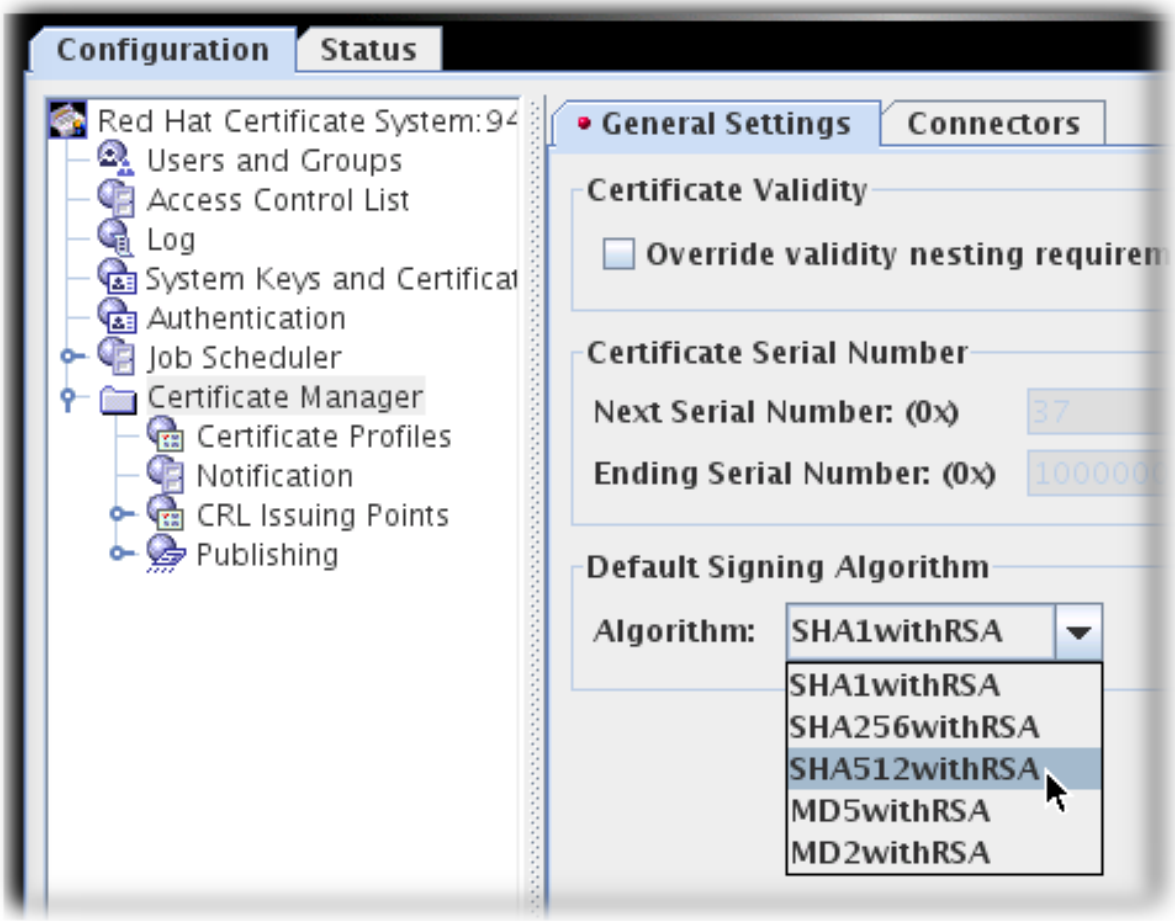
Each profile can define which cipher suite the CA should use to sign certificates processed through that profile. If no signing algorithm is set, then the profile uses whatever the default signing algorithm is.

3.5.1. Setting the CA's Default Signing Algorithm

1. Open the CA console.

```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, expand the **Certificate Manager** tree.
3. In the **General Settings** tab, set the algorithm to use in the **Algorithm** drop-down menu.



NOTE

pkiconsole is being deprecated.

3.5.2. Setting the Signing Algorithm Default in a Profile

Each profile has a Signing Algorithm Default extension defined. The default has two settings: a default algorithm and a list of allowed algorithms, if the certificate request specifies a different algorithm. If no signing algorithms are specified, then the profile uses whatever is set as the default for the CA.

In the profile's **.cfg** file, the algorithm is set with two parameters:

```
policyset.cmcUserCertSet.8.constraint.class_id=signingAlgConstraintImpl
policyset.cmcUserCertSet.8.constraint.name=No Constraint
policyset.cmcUserCertSet.8.constraint.params.signingAlgsAllowed=SHA256withRSA,SHA512withRSA,S
HA256withEC,SHA384withRSA,SHA384withEC,SHA512withEC
policyset.cmcUserCertSet.8.default.class_id=signingAlgDefaultImpl
policyset.cmcUserCertSet.8.default.name=Signing Alg
policyset.cmcUserCertSet.8.default.params.signingAlg=-
```

To configure the Signing Algorithm Default through the console:

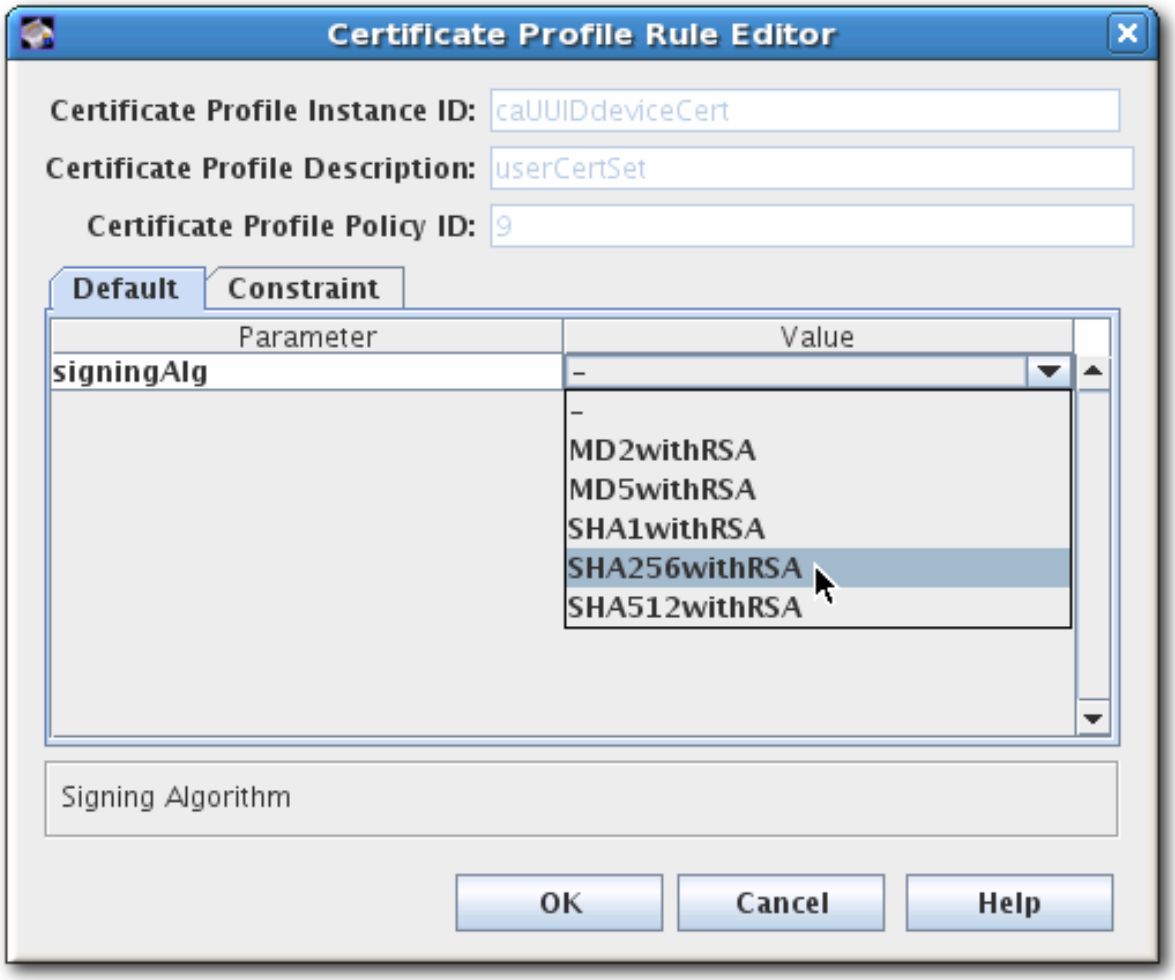
NOTE

Before a profile can be edited, it must first be disabled by an agent.

1. Open the CA console.

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, expand the **Certificate Manager** tree.
3. Click the **Certificate Profiles** item.
4. Click the **Policies** tab.
5. Select the **Signing Alg** policy, and click the **Edit** button.
6. To set the default signing algorithm, set the value in the **Defaults** tab. If this is set to **-**, then the profile uses the CA's default.



7. To set a list of allowed signing algorithms which can be accepted in a certificate request, open the **Constraints** tab, and set the list of algorithms in the **Value** field for **signingAlgsAllowed**.

   The possible values for the constraint are listed in Section B.2.10, "Signing Algorithm Constraint".

NOTE

pkiconsole is being deprecated.

3.6. MANAGING CA-RELATED PROFILES

Certificate profiles and extensions must be used to set rules on how subordinate CAs can issue certificates. There are two parts to this:

- Managing the CA signing certificate
- Defining issuance rules

3.6.1. Setting Restrictions on CA Certificates

When a subordinate CA is created, the root CA can impose limits or restrictions on the subordinate CA. For example, the root CA can dictate the maximum depth of valid certification paths (the number of subordinate CAs allowed to be chained below the new CA) by setting the pathLenConstraint field of the Basic Constraints extension in the CA signing certificate.

A certificate chain generally consists of an entity certificate, zero or more intermediate CA certificates, and a root CA certificate. The root CA certificate is either self-signed or signed by an external trusted CA. Once issued, the root CA certificate is loaded into a certificate database as a trusted CA.

An exchange of certificates takes place when performing a TLS handshake, when sending an S/MIME message, or when sending a signed object. As part of the handshake, the sender is expected to send the subject certificate and any intermediate CA certificates needed to link the subject certificate to the trusted root. For certificate chaining to work properly the certificates should have the following properties:

- CA certificates must have the Basic Constraints extension.
- CA certificates must have the keyCertSign bit set in the Key Usage extension.
- When the CAs generate new keys, they must add the Authority Key Identifier extension to all subject certificates. This extensions helps distinguish the certificates from the older CA certificates. The CA certificates must contain the Subject Key Identifier extension.

For more information on certificates and their extensions, see *Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile (RFC 5280)*, available at RFC 5280.

These extensions can be configured through the certificate profile enrollment pages. By default, the CA contains the required and reasonable configuration settings, but it is possible to customize these settings.

NOTE

This procedure describes editing the CA certificate profile used by a CA to issue CA certificates to its subordinate CAs.

The profile that is used when a CA instance is first configured is **/var/lib/pki/*instance_name*/ca/conf/caCert.profile**. This profile cannot be edited in **pkiconsole** (since it is only available before the instance is configured). It is possible to edit the policies for this profile in the template file before the CA is configured using a text editor.

To modify the default in the CA signing certificate profile used by a CA:

1. If the profile is currently enabled, it must be disabled before it can be edited. Open the agent services page, select **Manage Certificate Profiles** from the left navigation menu, select the profile, and click **Disable profile**.
2. Open the CA Console.

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

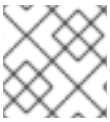
3. In the left navigation tree of the **Configuration** tab, select **Certificate Manager**, then **Certificate Profiles**.
4. Select caCACert, or the appropriate CA signing certificate profile, from the right window, and click **Edit/View**.
5. In the **Policies** tab of the **Certificate Profile Rule Editor**, select and edit the Key Usage or Extended Key Usage Extension Default if it exists or add it to the profile.

6. Select the Key Usage or Extended Key Usage Extension Constraint, as appropriate, for the default.

7. Set the default values for the CA certificates. For more information, see Section B.1.13, "Key Usage Extension Default" and Section B.1.8, "Extended Key Usage Extension Default" .

8. Set the constraint values for the CA certificates. There are no constraints to be set for a Key Usage extension; for an Extended Key Usage extension, set the appropriate OID constraints for the CA. For more information, see Section B.1.8, "Extended Key Usage Extension Default" .

9. When the changes have been made to the profile, log into the agent services page again, and re-enable the certificate profile.



NOTE

pkiconsole is being deprecated.

For more information on modifying certificate profiles, see Section 3.2, "Setting up Certificate Profiles".

3.6.2. Changing the Restrictions for CAs on Issuing Certificates

The restrictions on the certificates issued are set by default after the subsystem is configured. These include:

- Whether certificates can be issued with validity periods longer than the CA signing certificate. The default is to disallow this.
- The signing algorithm used to sign certificates.
- The serial number range the CA is able to use to issue certificates.

Subordinate CAs have constraints for the validity periods, types of certificates, and the types of extensions which they can issue. It is possible for a subordinate CA to issue certificates that violate these constraints, but a client authenticating a certificate that violates those constraints will not accept that certificate. Check the constraints set on the CA signing certificate before changing the issuing rules for a subordinate CA.

To change the certificate issuance rules:

1. Open the Certificate System Console.

```
pkiconsole https://server.example.com:8443/ca
```

2. Select the **Certificate Manager** item in the left navigation tree of the **Configuration** tab.

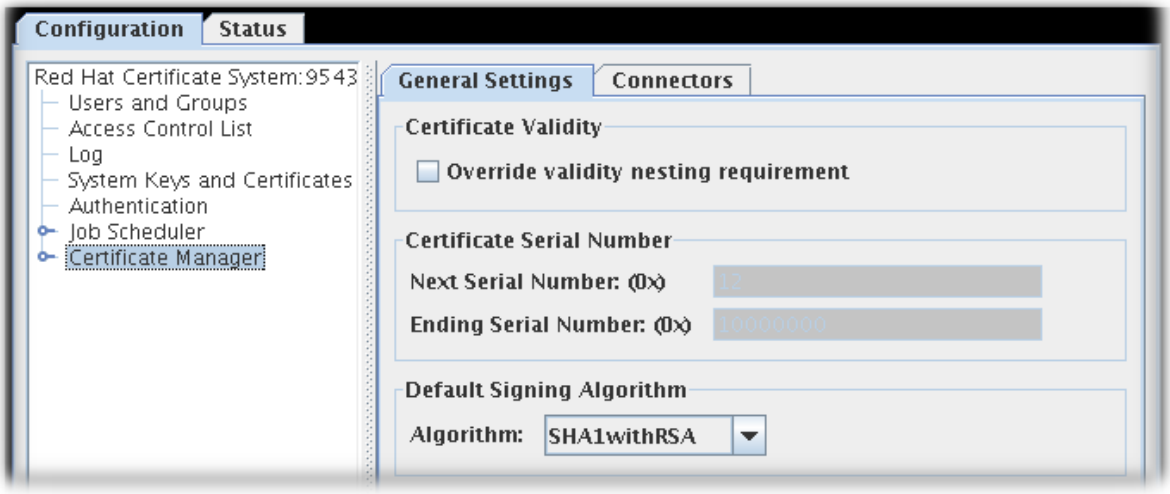


Figure 3.1. The General Settings Tab in non-subordinate CAs by default

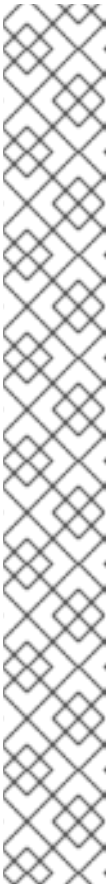
3. By default, in non-cloned CAs, the **General Settings** tab of the **Certificate Manager** menu item contains these options:

   - **Override validity nesting requirement.** This checkbox sets whether the Certificate Manager can issue certificates with validity periods longer than the CA signing certificate validity period.

     If this checkbox is not selected and the CA receives a request with validity period longer than the CA signing certificate's validity period, it automatically truncates the validity period to end on the day the CA signing certificate expires.

   - **Certificate Serial Number.** These fields display the serial number range for certificates issued by the Certificate Manager. The server assigns the serial number in the **Next serial number** field to the next certificate it issues and the number in the **Ending serial number** to the last certificate it issues.

     The serial number range allows multiple CAs to be deployed and balances the number of certificates each CA issues. The combination of an issuer name and a serial number uniquely identifies a certificate.



NOTE

The serial number ranges with cloned CAs are fluid. All cloned CAs share a common configuration entry which defines the next available range. When one CA starts running low on available numbers, it checks this configuration entry and claims the next range. The entry is automatically updated, so that the next CA gets a new range.

The ranges are defined in ***begin*Number*** and ***end*Number*** attributes, with separate ranges defined for requests and certificate serial numbers. For example:

```
dbs.beginRequestNumber=1
dbs.beginSerialNumber=1
dbs.enableSerialManagement=true
dbs.endRequestNumber=9980000
dbs.endSerialNumber=ffe0000
dbs.ldap=internaldb
dbs.newSchemaEntryAdded=true
dbs.replicaCloneTransferNumber=5
```

Serial number management can be enabled for CAs which are not cloned. However, by default, serial number management is disabled unless a system is cloned, when it is automatically enabled.

The serial number range cannot be updated manually through the console. The serial number ranges are read-only fields.

- **Default Signing Algorithm.** Specifies the signing algorithm the Certificate Manager uses to sign certificates. The options are **SHA256withRSA**, and **SHA512withRSA**, if the CA's signing key type is RSA.

  The signing algorithm specified in the certificate profile configuration overrides the algorithm set here.

4. By default, in cloned CAs, the **General Settings** tab of the **Certificate Manager** menu item contains these options:

   - **Enable serial number management**
   - **Enable random certificate serial numbers**

   Select both check boxes.

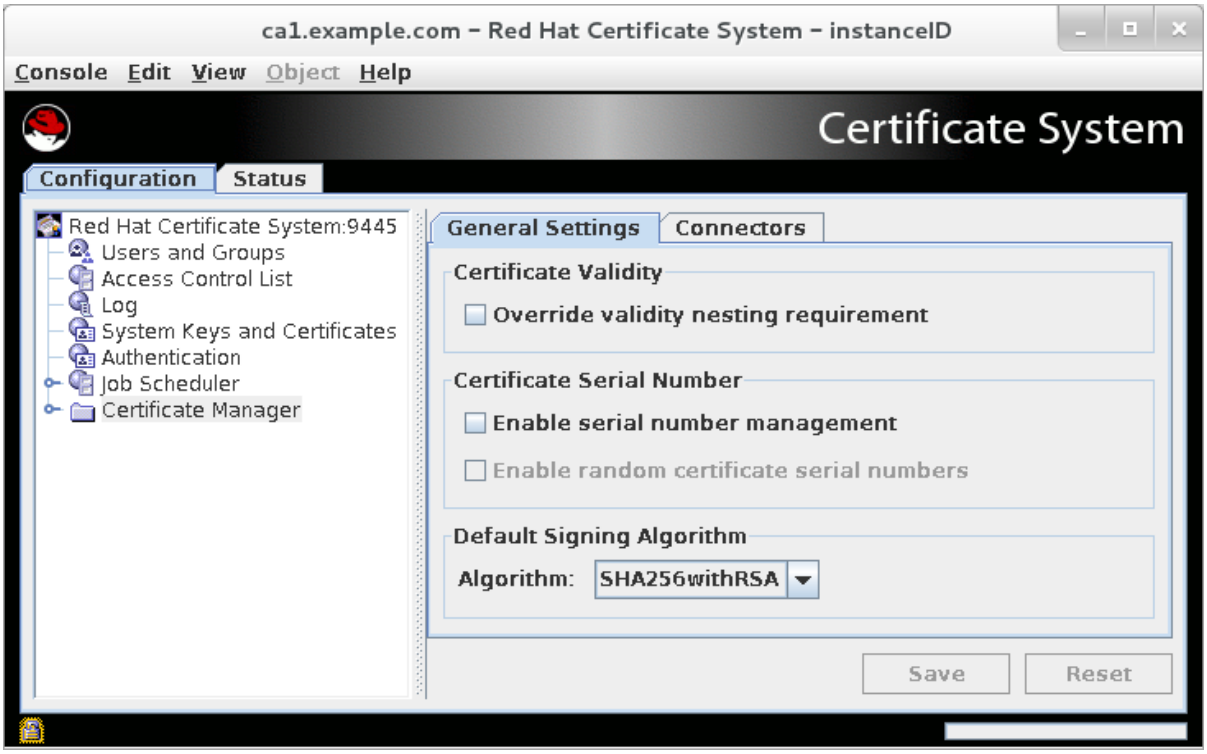


Figure 3.2. The General Settings Tab in cloned CAs by default

5. Click **Save**.

NOTE

pkiconsole is being deprecated.

3.6.3. Using Random Certificate Serial Numbers

Red Hat Certificate System contains a serial number range management for requests, certificates, and replica IDs. This allows the automation of cloning when installing **Identity Management** (IdM).

There are these ways to reduce the likelihood of hash-based attacks:

- making part of the certificate serial number unpredictable to the attacker
- adding a randomly chosen component to the identity
- making the validity dates unpredictable to the attacker by skewing each one forwards or backwards

The *random certificate serial number assignment* method adds a randomly chosen component to the identity. This method:

- works with cloning
- allows resolving conflicts
- is compatible with the current serial number management method
- is compatible with the current workflows for administrators, agents, and end entities
- fixes the existing bugs in sequential serial number management

NOTE

Administrators must enable random certificate serial numbers.

3.6.3.1. Enabling Random Certificate Serial Numbers

You can enable automatic serial number range management either from the command line or from the console UI.

To enable automatic serial number management from the console UI:

1. Tick the **Enable serial number management** option in the **General Settings** tab.

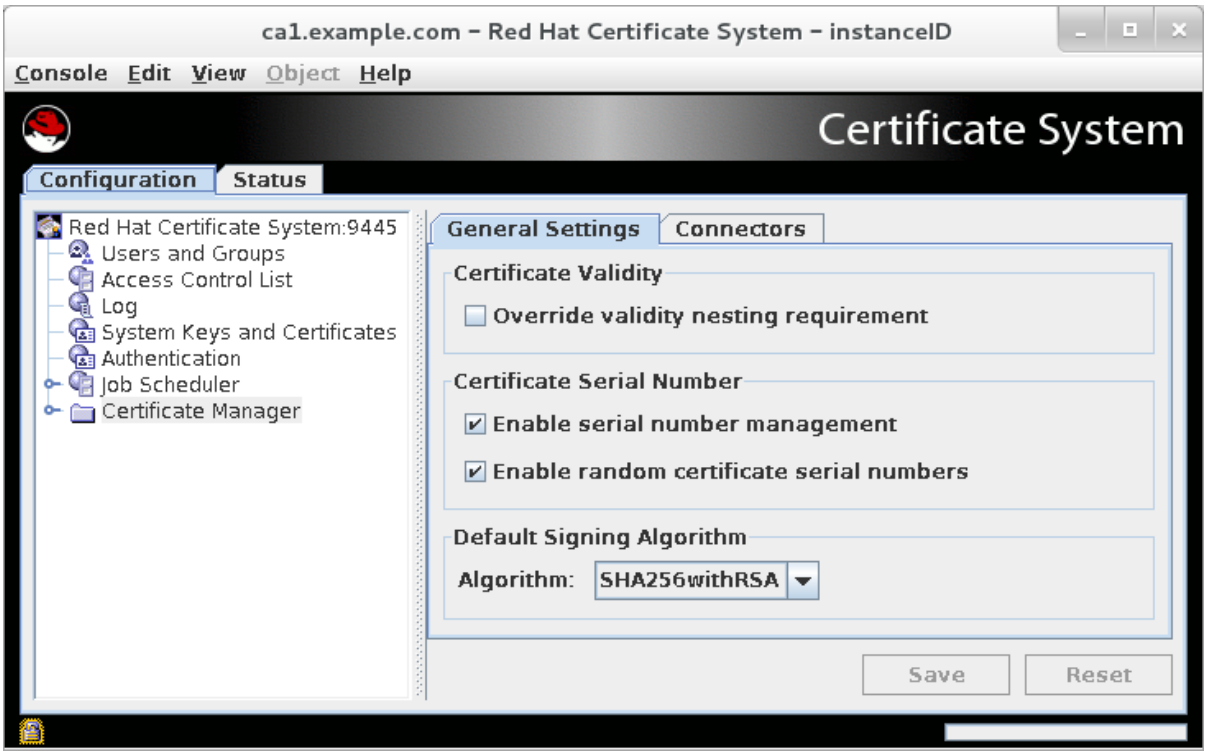


Figure 3.3. The General Settings Tab when Random Serial Number Assignment is enabled

2. Tick the **Enable random certificate serial numbers** option.

NOTE

pkiconsole is being deprecated.

3.6.4. Allowing a CA Certificate to Be Renewed Past the CA's Validity Period

Normally, a certificate cannot be issued with a validity period that ends *after* the issuing CA certificate's expiration date. If a CA certificate has an expiration date of December 31, 2015, then all of the certificates it issues must expire by or before December 31, 2015.

This rule applies to other CA signing certificates issued by a CA — and this makes renewing a root CA certificate almost impossible. Renewing a CA signing certificate means it would necessarily have to have a validity period past its own expiration date.

This behavior can be altered using the CA Validity Default. This default allows a setting (**bypassCAnotafter**) which allows a CA certificate to be issued with a validity period that extends past the issuing CA's expiration (*notAfter*) date.

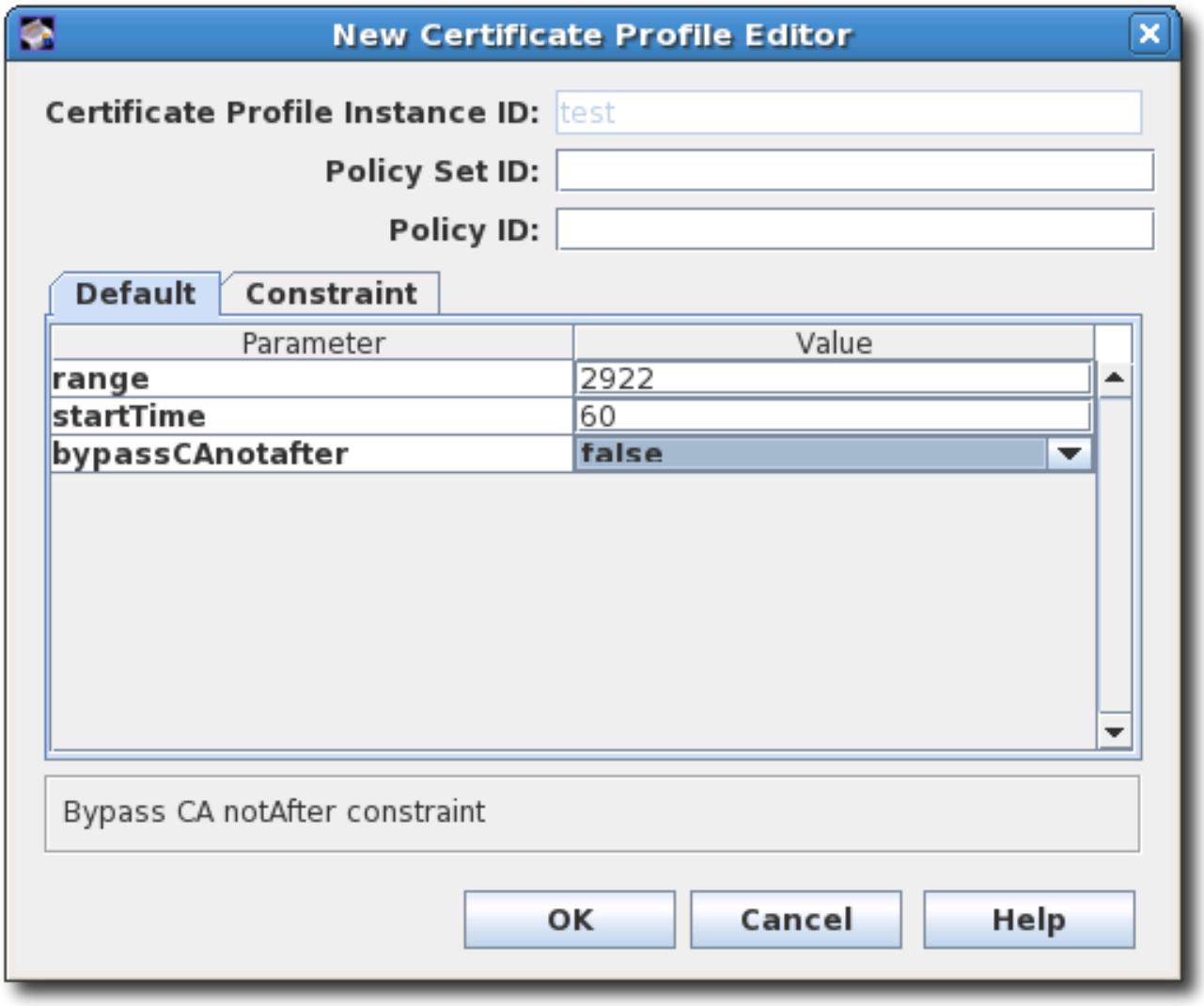


Figure 3.4. CA Validity Default Configuration

In real deployments, what this means is that a CA certificate for a root CA can be renewed, when it might otherwise be prevented.

To enable CA certificate renewals past the original CA's validity date:

1. Open the **caCACert.cfg** file.

```
vim /var/lib/pki/instance_name/ca/conf/caCACert.cfg
```

2. The CA Validity Default should be present by default. Set the value to **true** to allow a CA certificate to be renewed past the issuing CA's validity period.

```
policyset.caCertSet.2.default.name=CA Certificate Validity Default
policyset.caCertSet.2.default.params.range=2922
policyset.caCertSet.2.default.params.startTime=0
policyset.caCertSet.2.default.params.bypassCAnotafter=true
```

3. Restart the CA to apply the changes.

When an agent reviews a renewal request, there is an option in the **Extensions/Fields** area that allows the agent to choose to bypass the normal validity period constraint. If the agent selects **false**, the constraint is enforced, even if **bypassCAnotafter=true** is set in the profile. If the agent selects true when the **bypassCAnotafter** value is not enabled, then the renewal request is rejected by the CA.

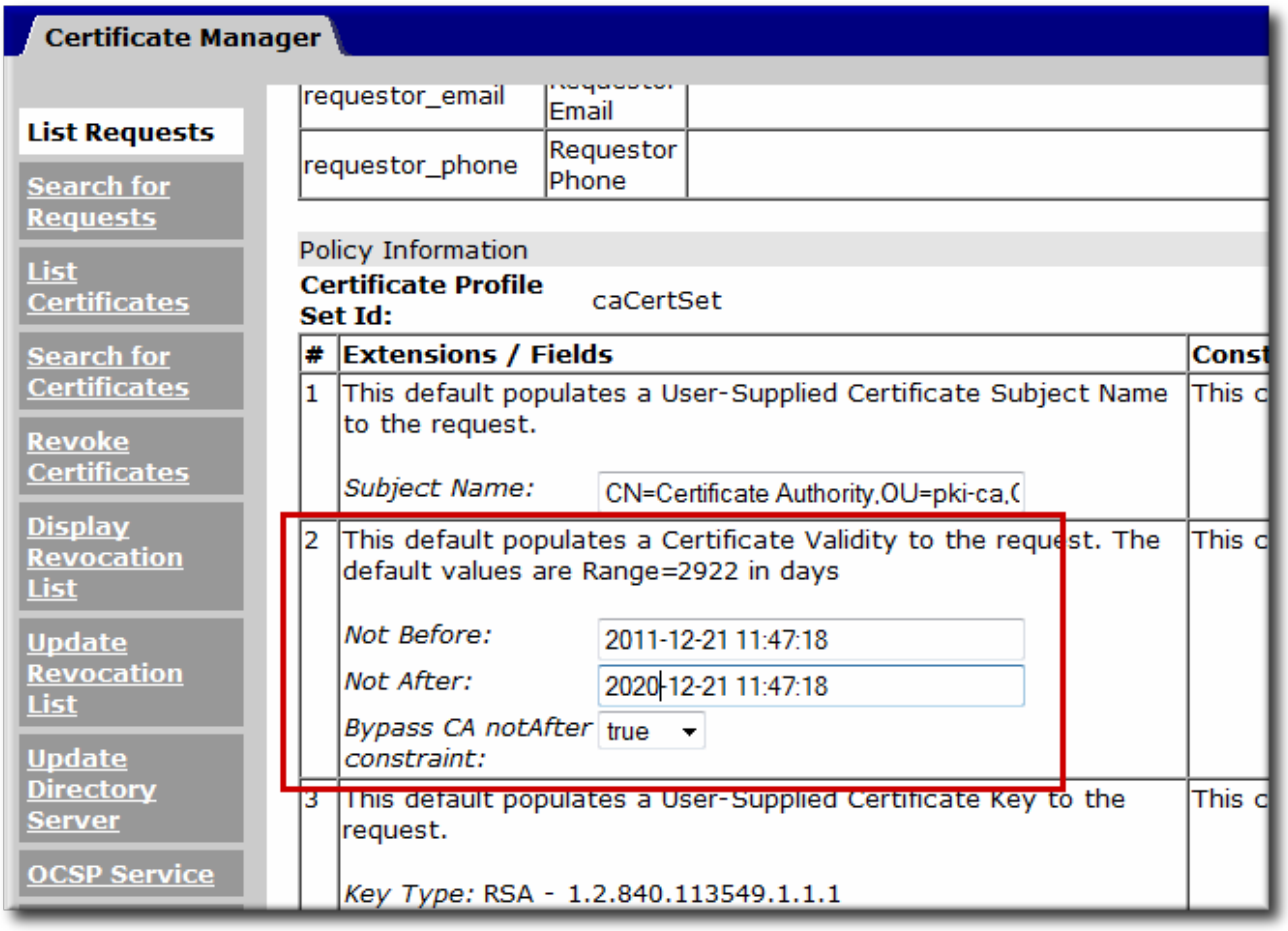
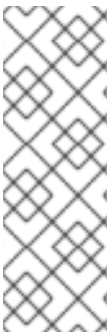


Figure 3.5. Bypass CA Constraints Option in the Agent Services Page



NOTE

The CA Validity Default only applies to CA signing certificate renewals. Other certificates must still be issued and renewed within the CA's validity period.

A separate configuration setting for the CA, **ca.enablePastCATime**, can be used to allow certificates to be renewed past the CA's validity period. However, this applies to *every* certificate issued by that CA. Because of the potential security issues, this setting is not recommended for production environments.

3.7. MANAGING SUBJECT NAMES AND SUBJECT ALTERNATIVE NAMES

The *subject name* of a certificate is a distinguished name (DN) that contains identifying information about the entity to which the certificate is issued. This subject name can be built from standard LDAP directory components, such as common names and organizational units. These components are defined in X.500. In addition to – or even in place of – the subject name, the certificate can have a *subject alternative name*, which is a kind of extension set for the certificate that includes additional information that is not defined in X.500.

The naming components for both subject names and subject alternative names can be customized.

IMPORTANT

If the subject name is empty, then the Subject Alternative Name extension must be present and marked critical.

3.7.1. Using the Requester CN or UID in the Subject Name

The ***cn*** or ***uid*** value from a certificate request can be used to build the subject name of the issued certificate. This section demonstrates a profile that requires the naming attribute (CN or UID) being specified in the Subject Name Constraint to be present in the certificate request. If the naming attribute is missing, the request is rejected.

There are two parts to this configuration:

- The CN or UID format is set in the **pattern** configuration in the Subject Name Constraint.
- The format of the subject DN, including the CN or UID token and the specific suffix for the certificate, is set in the Subject Name Default.

For example, to use the CN in the subject DN:

```
policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policyset.serverCertSet.1.constraint.name=Subject Name Constraint
policyset.serverCertSet.1.constraint.params.pattern=CN=[^,]+,.+
policyset.serverCertSet.1.constraint.params.accept=true
policyset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policyset.serverCertSet.1.default.name=Subject Name Default
policyset.serverCertSet.1.default.params.name=CN=$request.req_subject_name.cn$,DC=example,
DC=com
```

In this example, if a request comes in with the CN of **cn=John Smith**, then the certificate will be issued with a subject DN of **cn=John Smith,DC=example, DC=com**. If the request comes in but it has a UID of **uid=jsmith** and no CN, then the request is rejected.

The same configuration is used to pull the requester UID into the subject DN:

```
policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policyset.serverCertSet.1.constraint.name=Subject Name Constraint
policyset.serverCertSet.1.constraint.params.pattern=UID=[^,]+,.+
policyset.serverCertSet.1.constraint.params.accept=true
policyset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policyset.serverCertSet.1.default.name=Subject Name Default
policyset.serverCertSet.1.default.params.name=UID=$request.req_subject_name.uid$,DC=example,
DC=com
```

The format for the ***pattern*** parameter is covered in Section B.2.11, “Subject Name Constraint” and Section B.1.27, “Subject Name Default”.

3.7.2. Inserting LDAP Directory Attribute Values and Other Information into the Subject Alt Name

Information from an LDAP directory or that was submitted by the requester can be inserted into the

subject alternative name of the certificate by using matching variables in the Subject Alt Name Extension Default configuration. This default sets the type (format) of information and then the matching pattern (variable) to use to retrieve the information. For example:

```
policyset.userCertSet.8.default.class_id=subjectAltNameExtDefaultImpl
policyset.userCertSet.8.default.name=Subject Alt Name Constraint
policyset.userCertSet.8.default.params.subjAltNameExtCritical=false
policyset.userCertSet.8.default.params.subjAltExtType_0=RFC822Name
policyset.userCertSet.8.default.params.subjAltExtPattern_0=$request.requestor_email$
policyset.userCertSet.8.default.params.subjAltExtGNEnable_0=true
```

This inserts the requester's email as the first CN component in the subject alt name. To use additional components, increment the **Type_**, **Pattern_**, and **Enable_** values numerically, such as **Type_1**.

Configuring the subject alt name is detailed in Section B.1.23, "Subject Alternative Name Extension Default", as well.

To insert LDAP components into the subject alt name of the certificate:

1. Inserting LDAP attribute values requires enabling the user directory authentication plug-in, **SharedSecret**.

    1. Open the CA Console.

        ```
        pkiconsole https://server.example.com:8443/ca
        ```

    2. Select **Authentication** in the left navigation tree.

    3. In the **Authentication Instance** tab, click **Add**, and add an instance of the **SharedSecret** authentication plug-in.

    4. Enter the following information:

        ```
        Authentication InstanceID=SharedToken
        shrTokAttr=shrTok
        ldap.ldapconn.host=server.example.com
        ldap.ldapconn.port=636
        ldap.ldapconn.secureConn=true
        ldap.ldapauth.bindDN=cn=Directory Manager
        password=password
        ldap.ldapauth.authtype=BasicAuth
        ldap.basedn=ou=People,dc=example,dc=org
        ```

    5. Save the new plug-in instance.

    NOTE

    pkiconsole is being deprecated.

    For information on setting a CMC shared token, see Section 10.4.2, "Setting a CMC Shared Secret".

2. The **ldapStringAttributes** parameter instructs the authentication plug-in to read the value of the ***mail*** attribute from the user's LDAP entry and put that value in the certificate request.

When the value is in the request, the certificate profile policy can be set to insert that value for an extension value.

The format for the ***dnpattern*** parameter is covered in Section B.2.11, "Subject Name Constraint" and Section B.1.27, "Subject Name Default".

3. To enable the CA to insert the LDAP attribute value in the certificate extension, edit the profile's configuration file, and insert a policy set parameter for an extension. For example, to insert the ***mail*** attribute value in the Subject Alternative Name extension in the **caFullCMCSharedTokenCert** profile, change the following code:

```
policyset.setID.8.default.params.subjAltExtPattern_0=$request.auth_token.mail[0]$
```

For more details about editing a profile, see Section 3.2.1.3, "Editing a Certificate Profile in Raw Format".

4. Restart the CA.

```
systemctl restart pki-tomcatd-nuxwdog@instance_name.service
```

For this example, certificates submitted through the **caFullCMCSharedTokenCert** profile enrollment form will have the Subject Alternative Name extension added with the value of the requester's ***mail*** LDAP attribute. For example:

```
Identifier: Subject Alternative Name - 2.5.29.17
    Critical: no
    Value:
    RFC822Name: jsmith@example.com
```

There are many attributes which can be automatically inserted into certificates by being set as a token (*X*) in any of the **Pattern_** parameters in the policy set. The common tokens are listed in Table 3.1, "Variables Used to Populate Certificates", and the default profiles contain examples for how these tokens are used.

Table 3.1. Variables Used to Populate Certificates

Policy Set Token	Description
$request.auth_token.cn[0]$	The LDAP common name (***cn***) attribute of the user who requested the certificate.
$request.auth_token.mail[0]$	The value of the LDAP email (***mail***) attribute of the user who requested the certificate.
$request.auth_token.tokencertsubject$	The certificate subject name.
$request.auth_token.uid$	The LDAP user ID (***uid***) attribute of the user who requested the certificate.
$request.auth_token.userdn$	The user DN of the user who requested the certificate.

Policy Set Token	Description
$request.auth_token.userid$	The value of the user ID attribute for the user who requested the certificate.
$request.uid$	The value of the user ID attribute for the user who requested the certificate.
$request.requestor_email$	The email address of the person who submitted the request.
$request.requestor_name$	The person who submitted the request.
$request.upn$	The Microsoft UPN. This has the format *(UTF8String)1.3.6.1.4.1.311.20.2.3,$request.upn$*.
$server.source$	Instructs the server to generate a version 4 UUID (random number) component in the subject name. This always has the format *(IA5String)1.2.3.4,$server.source$*.
$request.auth_token.user$	Used when the request was submitted by TPS. The TPS subsystem trusted manager who requested the certificate.
$request.subject$	Used when the request was submitted by TPS. The subject name DN of the entity to which TPS has resolved and requested for. For example, **cn=John.Smith.123456789,o=TMS Org**

3.7.3. Using the CN Attribute in the SAN Extension

Several client applications and libraries no longer support using the Common Name (CN) attribute of the Subject DN for domain name validation, which has been deprecated in RFC 2818. Instead, these applications and libraries use the ***dNSName*** Subject Alternative Name (SAN) value in the certificate request.

Certificate System copies the CN only if it matches the preferred name syntax according to RFC 1034 Section 3.5 and has more than one component. Additionally, existing SAN values are preserved. For example, the ***dNSName*** value based on the CN is appended to existing SANs.

To configure Certificate System to automatically use the CN attribute in the SAN extension, edit the certificate profile used to issue the certificates. For example:

1. Disable the profile:

```
# pki -c password -p 8080 \
    -n "PKI Administrator for example.com" ca-profile-disable profile_name
```

2. Edit the profile:

```
# pki -c password -p 8080 \
    -n "PKI Administrator for example.com" ca-profile-edit profile_name
```

a. Add the following configuration with a unique set number for the profile. For example:

```
policyset.serverCertSet.12.constraint.class_id=noConstraintImpl
policyset.serverCertSet.12.constraint.name=No Constraint
policyset.serverCertSet.12.default.class_id=commonNameToSANDefaultImpl
policyset.serverCertSet.12.default.name=Copy Common Name to Subject
```

The previous example uses **12** as the set number.

b. Append the new policy set number to the ***policyset.userCertSet.list*** parameter. For example:

```
policyset.userCertSet.list=1,10,2,3,4,5,6,7,8,9,12
```

c. Save the profile.

3. Enable the profile:

```
# pki -c password -p 8080 \
    -n "PKI Administrator for example.com" ca-profile-enable profile_name
```

NOTE

All default server profiles contain the ***commonNameToSANDefaultImpl*** default.

3.7.4. Accepting SAN Extensions from a CSR

In certain environments, administrators want to allow specifying Subject Alternative Name (SAN) extensions in Certificate Signing Request (CSR).

3.7.4.1. Configuring a Profile to Retrieve SANs from a CSR

To allow retrieving SANs from a CSR, use the User Extension Default. For details, see Section B.1.32, “User Supplied Extension Default”.

NOTE

A SAN extension can contain one or more SANs.

To accept SANs from a CSR, add the following default and constraint to a profile, such as **caCMCECserverCert**:

```
prefix.constraint.class_id=noConstraintImpl
prefix.constraint.name=No Constraint

prefix.default.class_id=userExtensionDefaultImpl
prefix.default.name=User supplied extension in CSR
prefix.default.params.userExtOID=2.5.29.17
```

3.7.4.2. Generating a CSR with SANs

For example, to generate a CSR with two SANs using the **certutil** utility:

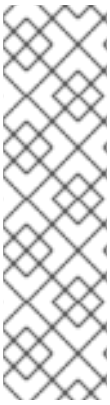
```
# certutil -R -k ec -q nistp256 -d . -s "cn=Example Multiple SANs" --extSAN
dns:www.example.com,dns:www.example.org -a -o /root/request.csr.p10
```

After generating the CSR, follow the steps described in Section 5.5.2, “The CMC Enrollment Process” to complete the CMC enrollment.

CHAPTER 4. SETTING UP KEY ARCHIVAL AND RECOVERY

For more information on Key Archival and Recovery, see the *Archiving, Recovering, and Rotating Keys* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

This chapter explains how to setup the Key Recovery Authority (KRA), previously known as Data Recovery Manager (DRM), to archive private keys and to recover archived keys for restoring encrypted data.

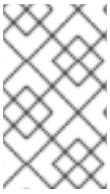


NOTE

This chapter only discusses archiving keys through client-side key generation. Server-side key generation and archivals, whether it's initiated through TPS, or through CA's End Entity portal, are not discussed here.

For information on smart card key recovery, see Section 6.11, “Setting Up Server-side Key Generation”.

For information on server-side key generation provided at the CA’s EE portal, see Section 5.2.2, “Generating CSRs Using Server-Side Key Generation” .

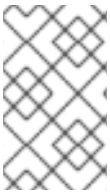


NOTE

Gemalto SafeNet LunaSA only supports PKI private key extraction in its CKE - Key Export model, and only in non-FIPS mode. The LunaSA Cloning model and the CKE model in FIPS mode do not support PKI private key extraction.

When KRA is installed, it joins a security domain, and is paired up with the CA. At such time, it is configured to archive and recover private encryption keys. However, if the KRA certificates are issued by an external CA rather than one of the CAs within the security domain, then the key archival and recovery process must be set up manually.

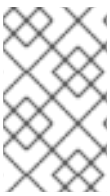
For more information, see the *Manually Setting up Key Archival* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.



NOTE

In a cloned environment, it is necessary to set up key archival and recovery manually. For more information, see the *Updating CA-KRA Connector Information After Cloning* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

4.1. CONFIGURING AGENT-APPROVED KEY RECOVERY IN THE CONSOLE



NOTE

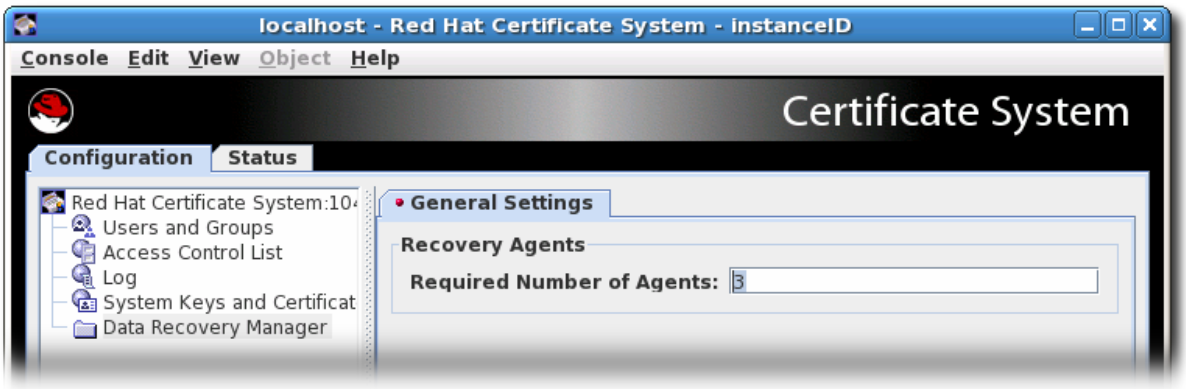
While the *number* of key recovery agents can be configured in the Console, the *group* to use can only be set directly in the **CS.cfg** file. The Console uses the **Key Recovery Authority Agents Group** by default.

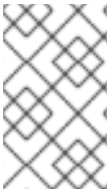
1. Open the KRA's console. For example:
   -

```
pkiconsole https://server.example.com:8443/kra
```

2. Click the **Key Recovery Authority** link in the left navigation tree.

3. Enter the number of agents to use to approve key recover in the **Required Number of Agents** field.

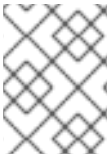




NOTE

For more information on how to configure agent-approved key recovery in the **CS.cfg** file, see the *Configuring Agent-Approved Key Recovery in the Command Line* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

4.2. TESTING THE KEY ARCHIVAL AND RECOVERY SETUP



NOTE

Newer browsers do not support key archival from the browser; for Step 1, one should substitute *CRMF generation clients* for those browsers.

To test whether a key can be successfully archived:

1. Enroll for dual certificates using the CA's **Manual User Signing & Encryption Certificates Enrollment** form.

2. Submit the request. Log in to the agent services page, and approve the request.

3. Log into the end-entities page, and check to see if the certificates have been issued. In the list of certificates, there should be two new certificates with consecutive serial numbers.

4. Import the certificates into the web browser.

5. Confirm that the key has been archived. In the KRA's agent services page, select **Show completed requests**. If the key has been archived successfully, there will be information about that key. If the key is not shown, check the logs, and correct the problem. If the key has been successfully archived, close the browser window.

6. Verify the key. Send a signed and encrypted email. When the email is received, open it, and check the message to see if it is signed and encrypted. There should be a security icon at the top-right corner of the message window that indicates that the message is signed and encrypted.

7. Delete the certificate. Check the encrypted email again; the mail client should not be able to decrypt the message.

8. Test whether an archived key can be recovered successfully:

    1. Open the KRA's agent services page, and click the **Recover Keys** link. Search for the key by the key owner, serial number, or public key. If the key has been archived successfully, the key information will be shown.

    2. Click **Recover**.

    3. In the form that appears, enter the base-64 encoded certificate that corresponds to the private key to recover; use the CA to get this information. If the archived key was searched for by providing the base-64 encoded certificate, then the certificate does not have to be supplied here.

    4. Make sure that the **Async Recovery** checkbox is selected to allow the browser session to be closed while recovery is ongoing.

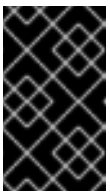
NOTE

An async recovery is the default and recommended way to perform a key recovery. If you want to perform a synchronous key recovery, the browser window cannot be shut and the KRA cannot be stopped during the recovery process.

    5. Depending on the agent scheme, a specified number of agents must authorize this key recovery. Have the agents search for the key to recover and then to approve the initiated recovery.

    6. Once all the agents have authorized the recovery, the next screen requests a password to encrypt the PKCS #12 file with the certificate.

    7. The next screen returns a link to download a PKCS #12 blob containing the recovered key pair. Follow the link, and save the blob to file.



IMPORTANT

Opening the PKCS #12 file directly from the browser in the **gcr-viewer** utility can fail in certain situations. To work around the problem, download the file and manually open it in **gcr-viewer**.

9. Restore the key to the browser's database. Import the **.p12** file into the browser and mail client.

10. Open the test email. The message should be shown again.

CHAPTER 5. REQUESTING, ENROLLING, AND MANAGING CERTIFICATES

Certificates are requested and used by end users. Although certificate enrollment and renewal are operations that are not limited to administrators, understanding the enrollment and renewal processes can make it easier for administrators to manage and create appropriate certificate profiles, as described in Section 3.2, "Setting up Certificate Profiles", and to use fitting authentication methods (described in Chapter 10, *Authentication for Enrolling Certificates*) for each certificate type.

This chapter discusses requesting, receiving, and renewing certificates for use outside Certificate System. For information on requesting and renewing Certificate System subsystem certificates, see Chapter 17, *Managing Subsystem Certificates*.

5.1. ABOUT ENROLLING AND RENEWING CERTIFICATES

Enrollment is the process for requesting and receiving a certificate. The mechanics for the enrollment process are slightly different depending on the type of certificate, the method for generating its key pair, and the method for generating and approving the certificate itself. Whatever the specific method, certificate enrollment, at a high level, has the same basic steps:

1. A certificate request (CSR) is generated.
2. The certificate request is submitted to the CA.
3. The request is verified by authenticating the entity which requested it and by confirming that the request meets the certificate profile rules which were used to submit it.
4. The request is approved.
5. The requesting party retrieves the new certificate.

When the certificate reaches the end of its validity period, it can be renewed.

5.2. CREATING CERTIFICATE SIGNING REQUESTS

Traditionally, the following methods are used to generate Certificate requests (CSRs):

- Generating CSRs using command line utilities
- Generating CSRs inside a supporting browser
- Generating CSRs inside an application, such as the installer of a server

Some of these methods support direct submission of the CSRs, while some do not.

Starting from RHCS 9.7, Server-Side key generation is supported to overcome the inconvenience brought on by the removal of the key generation support inside newer versions of browsers, such as Firefox v69 and up, as well as Chrome. For this reason, in this section, we will not discuss browser support for key generation. Although there is no reason to believe that older versions of those browsers should not continue to function as specified in older RHCS documentation.

CSRs generated from an application generally take the form of PKCS#10. Provided that they are generated correctly, they should be supported by RHCS.

In the following subsections, we are going to go over the following methods supported by RHCS:

- Command-line utilities
- Server-Side Key Generation

5.2.1. Generating CSRs Using Command-Line Utilities

Red Hat Certificate System supports using the following utilities to create CSRs:

- **certutil**: Supports creating PKCS #10 requests.
- **PKCS10Client**: Supports creating PKCS #10 requests.
- **CRMFPopClient**: Supports creating CRMF requests.
- **pki client-cert-request**: Supports both PKCS#10 and CRMF requests.

The following sections provide some examples on how to use these utilities with the feature-rich enrollment profile framework.

5.2.1.1. Creating a CSR Using certutil

This section describes examples on how to use the **certutil** utility to create a CSR.

For further details about using **certutil**, see:

- The certutil(1) man page
- The output of the **certutil --help** command

5.2.1.1.1. Using certutil to Create a CSR with EC Keys

The following procedure demonstrates how to use the **certutil** utility to create an Elliptic Curve (EC) key pair and CSR:

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

   ```
   $ cd /user_or_entity_database_directory/
   ```

2. Create the binary CSR and store it in the ***/user_or_entity_database_directory/request.csr*** file:

   ```
   $ certutil -d . -R -k ec -q nistp256 -s "CN=subject_name" -o
   /user_or_entity_database_directory/request-bin.csr
   ```

   Enter the required NSS database password when prompted.

   For further details about the parameters, see the certutil(1) man page.

3. Convert the created binary format CSR to PEM format:

   ```
   $ BtoA /user_or_entity_database_directory/request-bin.csr
   /user_or_entity_database_directory/request.csr
   ```

4. Optionally, verify that the CSR file is correct:

```
$ cat /user_or_entity_database_directory/request.csr

MIICbTCCAVUCAQAwKDEQMA4GA1UEChMHRXhhbXBsZTEUMBIGA1UEAxMLZXhhbXBs
...
```

This is a PKCS#10 PEM certificate request.

5.2.1.1.2. Using certutil to Create a CSR With User-defined Extensions

The following procedure demonstrates how to create a CSR with user-defined extensions using the **certutil** utility.

Note that the enrollment requests are constrained by the enrollment profiles defined by the CA. See Example B.3, “Multiple User Supplied Extensions in CSR” .

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /user_or_entity_database_directory/
```

2. Create the CSR with user-defined Key Usage extension as well as user-defined Extended Key Usage extension and store it in the ***/user_or_entity_database_directory/request.csr*** file:

```
$ certutil -d . -R -k rsa -g 1024 -s "CN=subject_name" --keyUsage
keyEncipherment,dataEncipherment,critical --extKeyUsage
timeStamp,msTrustListSign,critical -a -o /user_or_entity_database_directory/request.csr
```

Enter the required NSS database password when prompted.

For further details about the parameters, see the certutil(1) man page.

3. Optionally, verify that the CSR file is correct:

```
$ cat /user_or_entity_database_directory/request.csr
  Certificate request generated by Netscape certutil
  Phone: (not specified)

  Common Name: user 4-2-1-2
  Email: (not specified)
  Organization: (not specified)
  State: (not specified)
  Country: (not specified)
```

This is a PKCS#10 PEM certificate request.

5.2.1.2. Creating a CSR Using PKCS10Client

This section describes examples how to use the **PKCS10Client** utility to create a CSR.

For further details about using **PKCS10Client**, see:

- The PKCS10Client(1) man page
- The output of the **PKCS10Client --help** command

5.2.1.2.1. Using PKCS10Client to Create a CSR

The following procedure explains how to use the **PKCS10Client** utility to create an Elliptic Curve (EC) key pair and CSR:

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /user_or_entity_database_directory/
```

2. Create the CSR and store it in the ***/user_or_entity_database_directory/example.csr*** file:

```
$ PKCS10Client -d . -p NSS_password -a ec -c nistp256 -o
/user_or_entity_database_directory/example.csr -n "CN=subject_name"
```

   For further details about the parameters, see the PKCS10Client(1) man page.

3. Optionally, verify that the CSR is correct:

```
$ cat /user_or_entity_database_directory/example.csr
  -----BEGIN CERTIFICATE REQUEST-----
  MIICzzCCAbcCAQAwgYkx
  ...
  -----END CERTIFICATE REQUEST-----
```

5.2.1.2.2. Using PKCS10Client to Create a CSR for SharedSecret-based CMC

The following procedure explains how to use the **PKCS10Client** utility to create an RSA key pair and CSR for SharedSecret-based CMC. Use it only with the CMC Shared Secret authentication method which is, by default, handled by the **caFullCMCSharedTokenCert** and **caECFullCMCSharedTokenCert** profiles.

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /user_or_entity_database_directory/
```

2. Create the CSR and store it in the ***/user_or_entity_database_directory/example.csr*** file:

```
$ PKCS10Client -d . -p NSS_password -o /user_or_entity_database_directory/example.csr -y
true -n "CN=subject_name"
```

   For further details about the parameters, see the PKCS10Client(1) man page.

3. Optionally, verify that the CSR is correct:

```
$ cat /user_or_entity_database_directory/example.csr
  -----BEGIN CERTIFICATE REQUEST-----
  MIICzzCCAbcCAQAwgYkx
```

```
...
-----END CERTIFICATE REQUEST-----
```

5.2.1.3. Creating a CSR Using CRMFPopClient

Certificate Request Message Format (CRMF) is a CSR format accepted in CMC that allows key archival information to be securely embedded in the request.

This section describes examples how to use the **CRMFPopClient** utility to create a CSR.

For further details about using **CRMFPopClient**, see the CRMFPopClient(1) man page.

5.2.1.3.1. Using CRMFPopClient to Create a CSR with Key Archival

The following procedure explains how to use the **CRMFPopClient** utility to create an RSA key pair and a CSR with the key archival option:

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /user_or_entity_database_directory/
```

2. Retrieve the KRA transport certificate:

```
$ pki ca-cert-find --name "DRM Transport Certificate"
  ---------------
  1 entries found
  ---------------
   Serial Number: 0x7
   Subject DN: CN=DRM Transport Certificate,O=EXAMPLE
   Status: VALID
   Type: X.509 version 3
   Key A    lgorithm: PKCS #1 RSA with 2048-bit key
   Not Valid Before: Thu Oct 22 18:26:11 CEST 2015
   Not Valid After: Wed Oct 11 18:26:11 CEST 2017
   Issued On: Thu Oct 22 18:26:11 CEST 2015
   Issued By: caadmin
  ----------------------------
  Number of entries returned 1
```

3. Export the KRA transport certificate:

```
$ pki ca-cert-show 0x7 --output kra.transport
```

4. Create the CSR and store it in the ***/user_or_entity_database_directory/example.csr*** file:

```
$ CRMFPopClient -d . -p password -n "cn=subject_name" -q POP_SUCCESS -b
kra.transport -w "AES/CBC/PKCS5Padding" -v -o
/user_or_entity_database_directory/example.csr
```

   To create an Elliptic Curve (EC) key pair and CSR, pass the **-a ec -t false** options to the command.

For further details about the parameters, see the CRMFPopClient(1) man page.

5. Optionally, verify that the CSR is correct:

```
$ cat /user_or_entity_database_directory/example.csr
 -----BEGIN CERTIFICATE REQUEST-----
 MIICzzCCAbcCAQAwgYkx
 ...
 -----END CERTIFICATE REQUEST-----
```

5.2.1.3.2. Using CRMFPopClient to Create a CSR for SharedSecret-based CMC

The following procedure explains how to use the **CRMFPopClient** utility to create an RSA key pair and CSR for SharedSecret-based CMC. Use it only with the CMC Shared Secret authentication method which is, by default, handled by the **caFullCMCSharedTokenCert** and **caECFullCMCSharedTokenCert** profiles.

1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /user_or_entity_database_directory/
```

2. Retrieve the KRA transport certificate:

```
$ pki ca-cert-find --name "DRM Transport Certificate"
 ---------------
 1 entries found
 ---------------
  Serial Number: 0x7
  Subject DN: CN=DRM Transport Certificate,O=EXAMPLE
  Status: VALID
  Type: X.509 version 3
  Key A    lgorithm: PKCS #1 RSA with 2048-bit key
  Not Valid Before: Thu Oct 22 18:26:11 CEST 2015
  Not Valid After: Wed Oct 11 18:26:11 CEST 2017
  Issued On: Thu Oct 22 18:26:11 CEST 2015
  Issued By: caadmin
 ----------------------------
 Number of entries returned 1
```

3. Export the KRA transport certificate:

```
$ pki ca-cert-show 0x7 --output kra.transport
```

4. Create the CSR and store it in the ***/user_or_entity_database_directory/example.csr*** file:

```
$ CRMFPopClient -d . -p password -n "cn=subject_name" -q POP_SUCCESS -b
kra.transport -w "AES/CBC/PKCS5Padding" -y -v -o
/user_or_entity_database_directory/example.csr
```

To create an EC key pair and CSR, pass the **-a ec -t false** options to the command.

For further details about the parameters, see the output of the **CRMFPopClient --help** command.

5. Optionally, verify that the CSR is correct:

```
$ cat /user_or_entity_database_directory/example.csr
 -----BEGIN CERTIFICATE REQUEST-----
 MIICzzCCAbcCAQAwgYkx
 ...
 -----END CERTIFICATE REQUEST-----
```

5.2.1.4. Creating a CSR using client-cert-request in the PKI CLI

The **pki**command-line tool can also be used with the **client-cert-request** command to generate a CSR. However, unlike the previously discussed tools, CSR generated with pki are submitted directly to the CA. Both PKCS#10 or CRMF requests can be generated.

Example on generating a PKCS#10 request:

```
pki -d user token db directory -P https -p 8443 -h host.test.com -c user token db passwd client-cert-request "uid=test2" --length 4096 --type pkcs10
```

Example on generating a CRMF request:

```
pki -d user token db directory -P https -p 8443 -h host.test.com -c user token db passwd client-cert-request "uid=test2" --length 4096 --type crmf
```

A request id will be returned upon success.

Once a request is submitted, an agent could approve it by using the pki **ca-cert-request-approve** command.

For example:

```
pki -d agent token db directory -P https -p 8443 -h host.test.com -c agent token db passwd -n <CA agent cert nickname> ca-cert-request-approve request id
```

For more information, see the man page by running the **pki client-cert-request --help** command.

5.2.2. Generating CSRs Using Server-Side Key Generation

Many newer versions of browsers, including Firefox v69 and up, as well as Chrome, have removed the functionality to generate PKI keys and the support for CRMF for key archival. On RHEL, CLIs such as **CRMFPopClient** (see **CRMFPopClient --help**) or **pki** (see **pki client-cert-request --help**) could be used as a workaround.

Server-Side Keygen enrollment has been around for a long time since the introduction of Token Key Management System (TMS), where keys could be generated on a KRA instead of locally on smart cards. Red Hat Certificate System now adopts a similar mechanism to resolve the browser keygen deficiency issue. Keys are generated on the server (specifically, on the KRA) and then transferred securely back to the client in PKCS#12.

NOTE

It is highly recommended to employ the Server-Side Keygen mechanism only for encryption certificates.

5.2.2.1. Functionality Highlights

- Certificate request keys are generated on the KRA (Note: a KRA must be installed to work with the CA)
- The profile default plugin, **serverKeygenUserKeyDefaultImpl**, provides selection to enable or disable key archival (i.e. the *enableArchival* parameter)
- Support for both RSA and EC keys
- Support for both manual (agent) approval and automatic approval (e.g. directory password-based)

5.2.2.2. Enrolling a Certificate Using Server-Side Keygen

The default Sever-Side Keygen enrollment profile can be found on the EE page, under the *List Certificate Profiles* tab:

Manual User Dual-Use Certificate Enrollment Using server-side Key generation

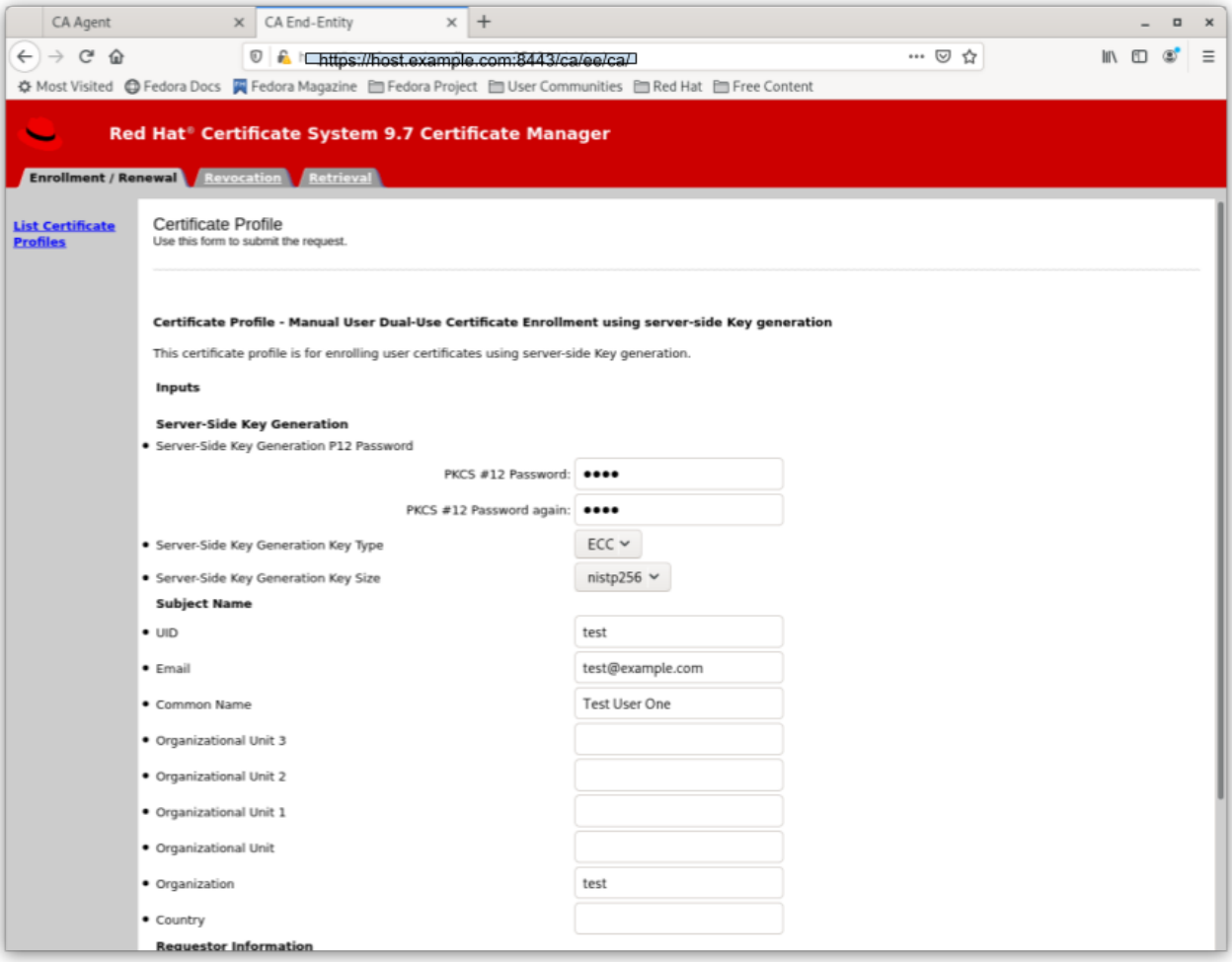


Figure 5.1. Server-Side Keygen Enrollment that requires agent manual approval

Directory-authenticated User Dual-Use Certificate Enrollment Using server-side Key generation

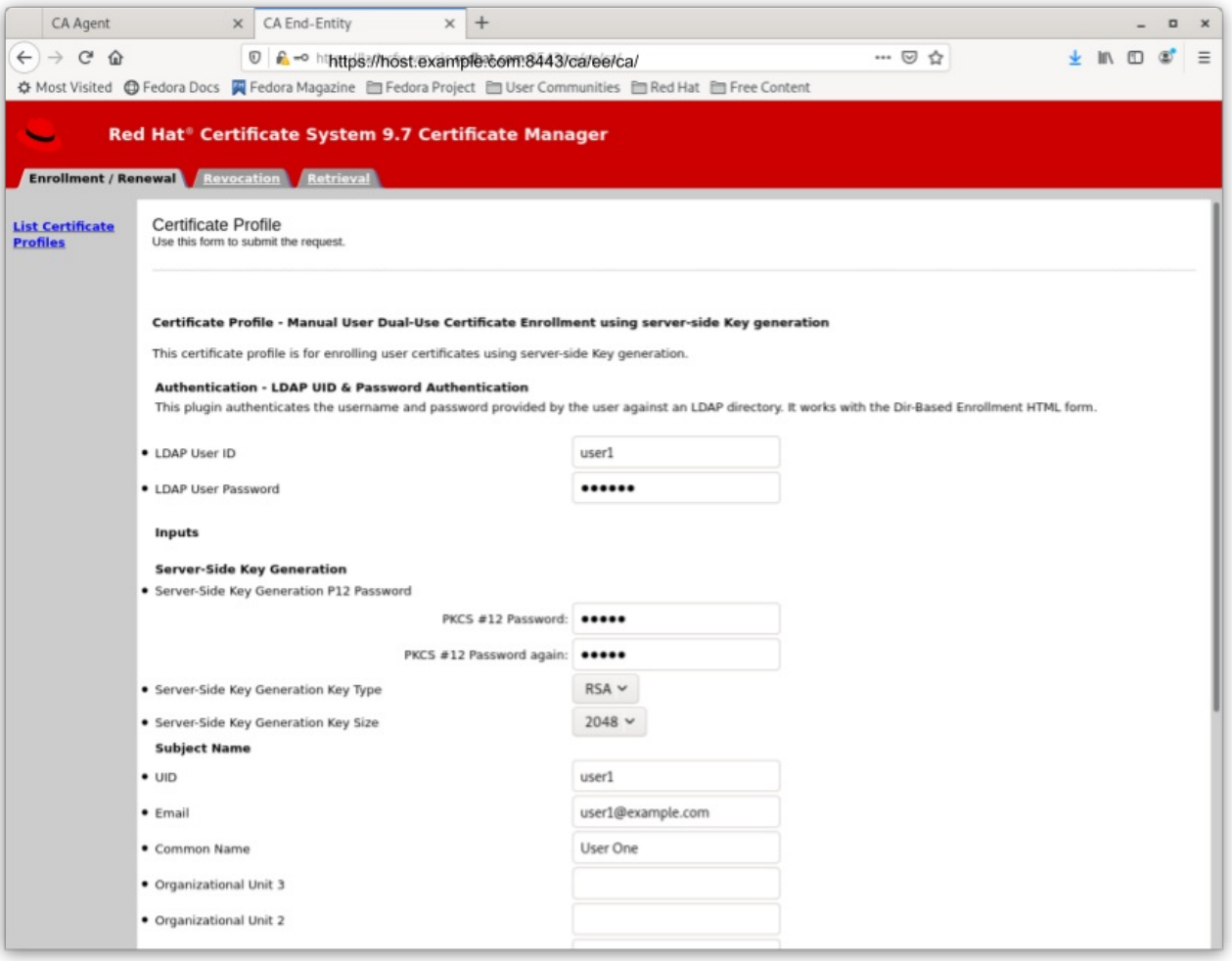


Figure 5.2. Server-Side Keygen Enrollment that will be automatically approved upon successful LDAP uid/pwd authentication

Regardless of how the request is approved, the Server-Side Keygen Enrollment mechanism requires the End Entity user to enter a password for the PKCS#12 package which will contain the issued certificate as well as the encrypted private key generated by the server once issued.

IMPORTANT

Users should not share their passwords with anyone. Not even the CA or KRA agents.

When the enrollment request is approved, the PKCS#12 package will be generated and,

- In case of manual approval, the PKCS#12 file will be returned to the CA agent that approves the request; the agent is then expected to forward the PKCS#12 file to the user.
- In case of automatic approval, the PKCS#12 file will be returned to the user who submitted the request

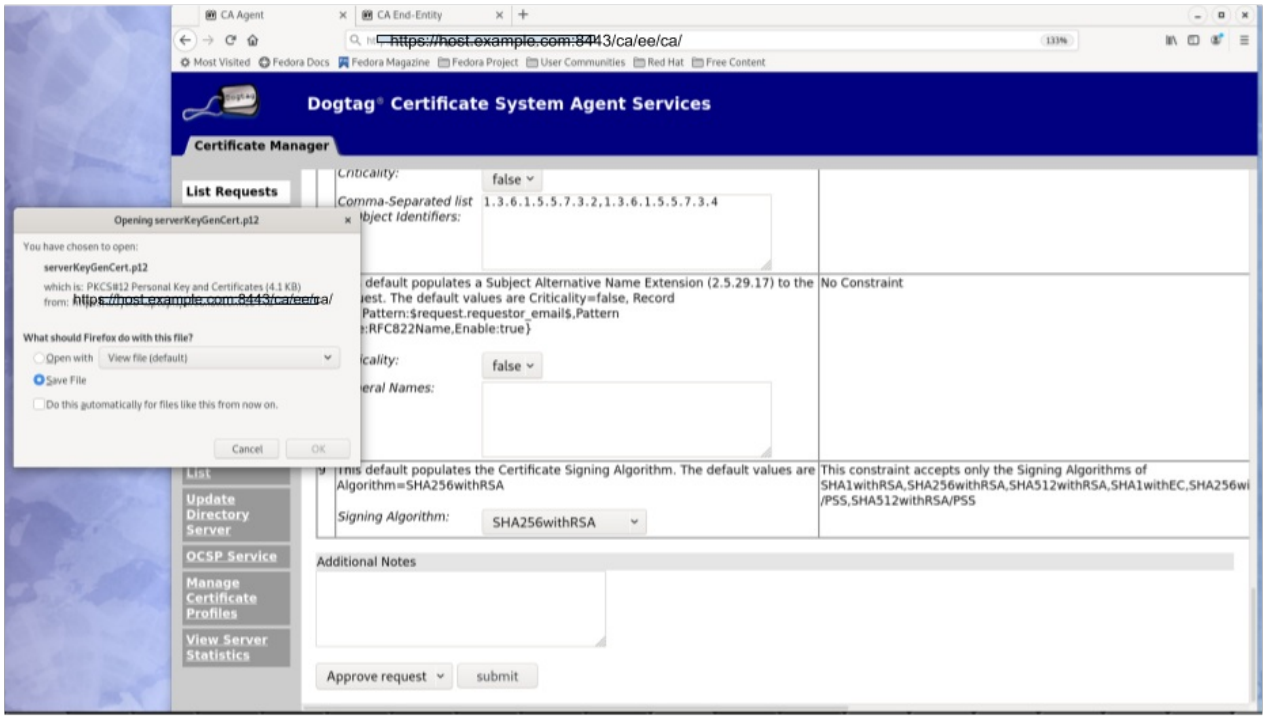


Figure 5.3. Enrollment manually approved by an agent

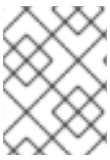
Once the PKCS#12 file is received, the user could use a CLI such as **pkcs12util** to import this file into their own user internal cert/key database for each application. E.g. the Firefox nss database of the user.

5.2.2.3. Key Recovery

If the *enableArchival* parameter is set to *true* in the certificate enrollment profile, then the private keys are archived at the time of Server-Side Keygen enrollment. The archived private keys could then be recovered by the authorized KRA agents.

5.2.2.4. Additional Information

5.2.2.4.1. KRA Request Records



NOTE

Due to the nature of this mechanism, in case the *enableArchival* parameter is set to *true* in the profile, there are two KRA requests records per Server-Side keygen request:

- One for the request type *asymkeyGenRequest*

  This request type cannot be filtered using **List Requests** on the KRA agent page; you can select **Show All Requests** to see them listed.

- One for the request type *recovery*

5.2.2.4.2. Audit Records

Some audit records could be observed if enabled:

CA

- SERVER_SIDE_KEYGEN_ENROLL_KEYGEN_REQUEST
- SERVER_SIDE_KEYGEN_ENROLL_KEY_RETRIEVAL_REQUEST

KRA

- SERVER_SIDE_KEYGEN_ENROLL_KEYGEN_REQUEST_PROCESSED
- SERVER_SIDE_KEYGEN_ENROLL_KEY_RETRIEVAL_REQUEST_PROCESSED (not yet implemented)

5.3. REQUESTING AND RECEIVING CERTIFICATES

As explained in Section 5.1, "About Enrolling and Renewing Certificates" , once CSRs are generated, they need to be submitted to the CA for issuance. Some of the methods discussed in Section 5.2, "Creating Certificate Signing Requests" submit CSRs to the CA directly, while some would require submission of the CSRs in a separate step, which could either be carried out by the user or pre-signed by an agent.

In this section, we are going to discuss the separate submission steps supported by the RHCS CA.

- Section 5.3.1, "Requesting and Receiving a Certificate through the End-Entities Page"
- Section 5.5, "Submitting Certificate requests Using CMC"

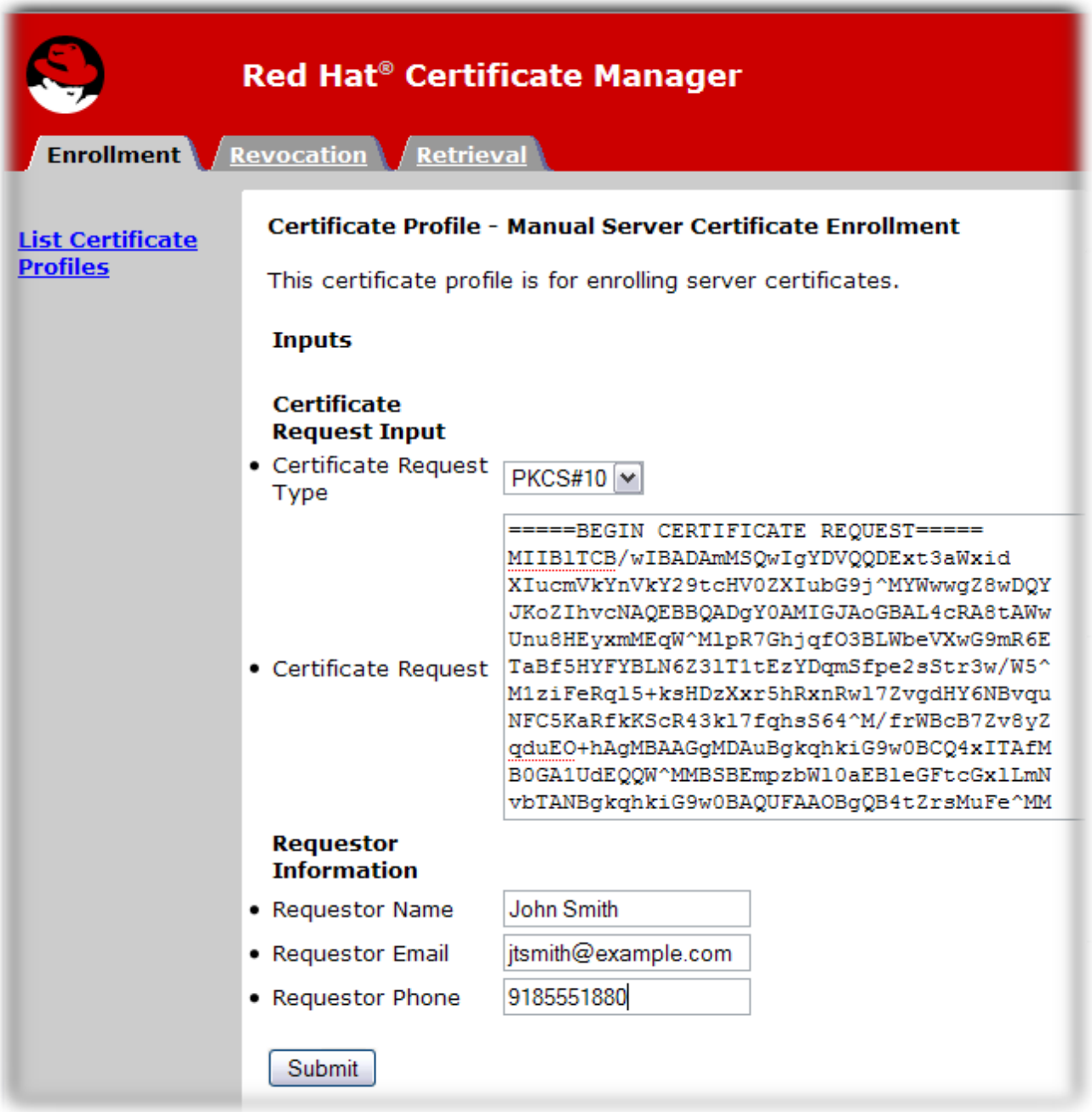
5.3.1. Requesting and Receiving a Certificate through the End-Entities Page

At the CA End Entity portal (i.e. https://*host.domain*:*port#*/ca/ee/ca), end entities can use the HTML enrollment forms presented at each applicable enrollment profile under the **Enrollment/Renewal** tab to submit their certificate requests (CSRs, see Section 5.2, "Creating Certificate Signing Requests" for how to generate CSRs).

This section assumes that you have the CSR in Base64 encoded format, including the marker lines -----BEGIN NEW CERTIFICATE REQUEST----- and -----END NEW CERTIFICATE REQUEST----- .

Many of the default enrollment profiles provide a **Certificate Request** text box where one could paste in the Base64 encoded CSR, along with a **Certificate Request Type** selection drop down list.

In the certificate enrollment form, enter the required information.



The standard requirements are as follows:

- **Certificate Request Type**. This is either PKCS#10 or CRMF. Certificate requests created through the subsystem administrative console are PKCS #10; those created through the **certutil** tool and other utilities are usually PKCS #10.
- **Certificate Request**. Paste the base-64 encoded blob, including the **-----BEGIN NEW CERTIFICATE REQUEST-----** and **-----END NEW CERTIFICATE REQUEST-----** marker lines.
- **Requester Name**. This is the common name of the person requesting the certificate.
- **Requester Email**. This is the email address of the requester. The agent or CA system will use this address to contact the requester when the certificate is issued. For example, **jdoe@someCompany.com**.
- **Requester Phone**. This is the contact phone number of the requester.

The submitted request is queued for agent approval. An agent needs to process and approve the certificate request.

NOTE

Some enrollment profiles may allow automatic approval such as by using the LDAP uid/pwd authentication method offered by Red Hat Certificate System. Enrollments through those profiles would not require manual agent approval in the next section. See Chapter 10, *Authentication for Enrolling Certificates* for supported approval methods.

In case of manual approval, once the certificate is approved and generated, you can retrieve the certificate.

1. Open the Certificate Manager end-entities page, for example:

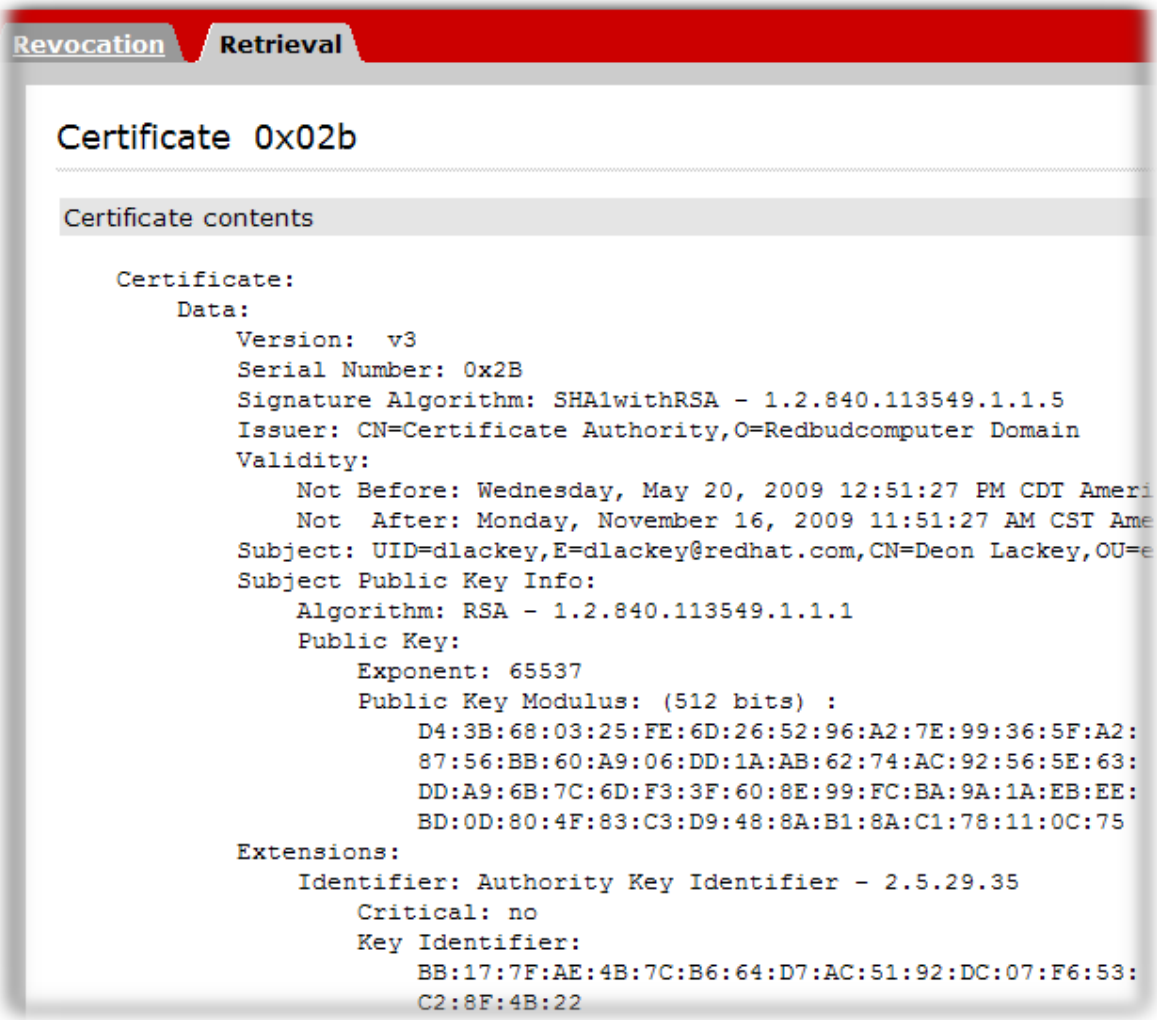
   ```
   https://server.example.com:8443/ca/ee/ca
   ```

2. Click the **Retrieval** tab.

3. Fill in the request ID number that was created when the certificate request was submitted, and click **Submit**.

4. The next page shows the status of the certificate request. If the status is **complete**, then there is a link to the certificate. Click the **Issued certificate** link.

5. The new certificate information is shown in pretty-print format, in base-64 encoded format, and in PKCS #7 format.



The following actions can be taken through this page:

- To install this certificate on a server or other application, scroll down to the **Installing This Certificate in a Server** section, which contains the base-64 encoded certificate.

6. Copy the base-64 encoded certificate, including the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** marker lines, to a text file. Save the text file, and use it to store a copy of the certificate in the security module of the entity where the private key resides. See Section 15.3.2.1, "Creating Users".

5.4. RENEWING CERTIFICATES

This section discusses how to renew certificates. For more information on how to set up certificate renewal, see Section 3.4, "Configuring Profiles to Enable Renewal".

Renewing a certificate consists in regenerating the certificate with the same properties to be used for the same purpose as the original certificate. In general, there are two types of renewals:

- *Same key Renewal* takes the original key, profile, and request of the certificate and recreates a new certificate with a new validity period and expiration date using the identical key. This can be done by either of the following methods:
  - resubmitting the original certificate request (CSR) through the original profile, or
  - regenerating a CSR with the original keys by using supporting tools such as certutil

- *Re-keying* a certificate requires regeneration of a certificate request with the same information, so that a new key pair is generated. The CSR is then submitted through the original profile.

5.4.1. Same Keys Renewal

5.4.1.1. Reusing CSR

There are three approval methods for same key renewal at the end entity portal.

- Agent-approved method requires submitting the serial number of the certificate to be renewed; This method would require a CA agent's approval.
- Directory-based renewal requires submitting the serial number of the certificate to be renewed, and the CA draws the information from its current certificate directory entry. The certificate is automatically approved if the ldap uid/pwd is authenticated successfully.
- Certificate-based renewal uses the certificate in the browser database to authenticate and have the same certificate re-issued.

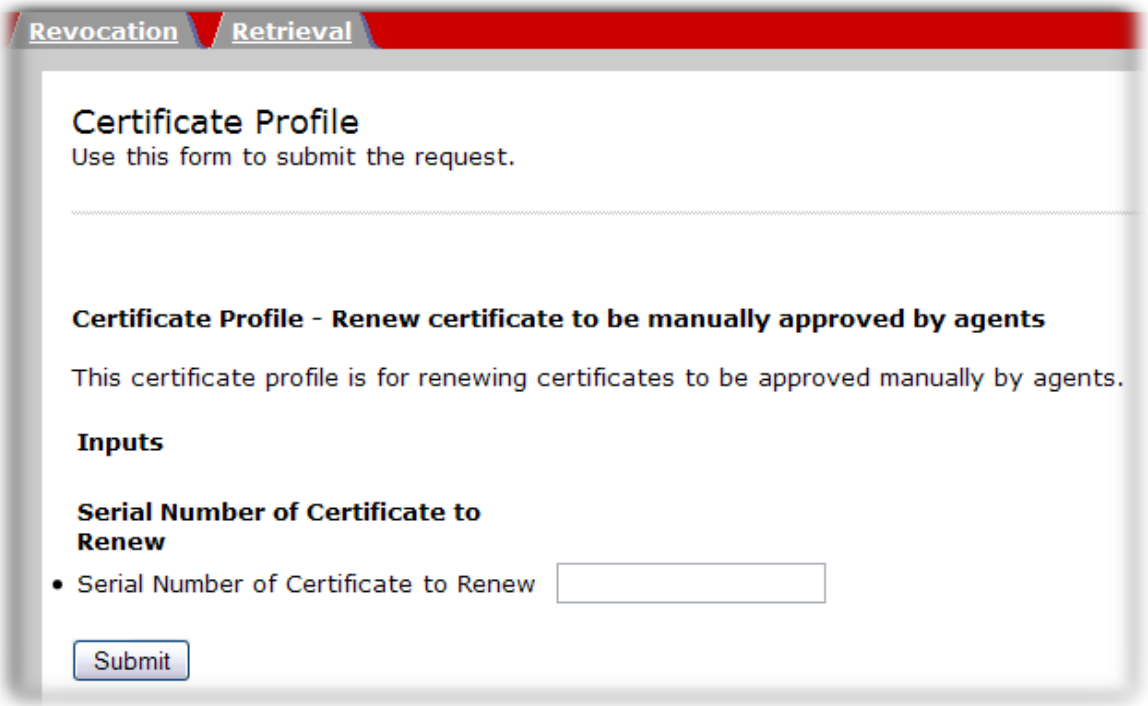
5.4.1.1.1. Agent-Approved or Directory-Based Renewals

Sometimes, a certificate renewal request has to be manually approved, either by a CA agent or by providing login information for the user directory.

1. Open the end-entities services page for the CA which issued the certificate (or its clone).

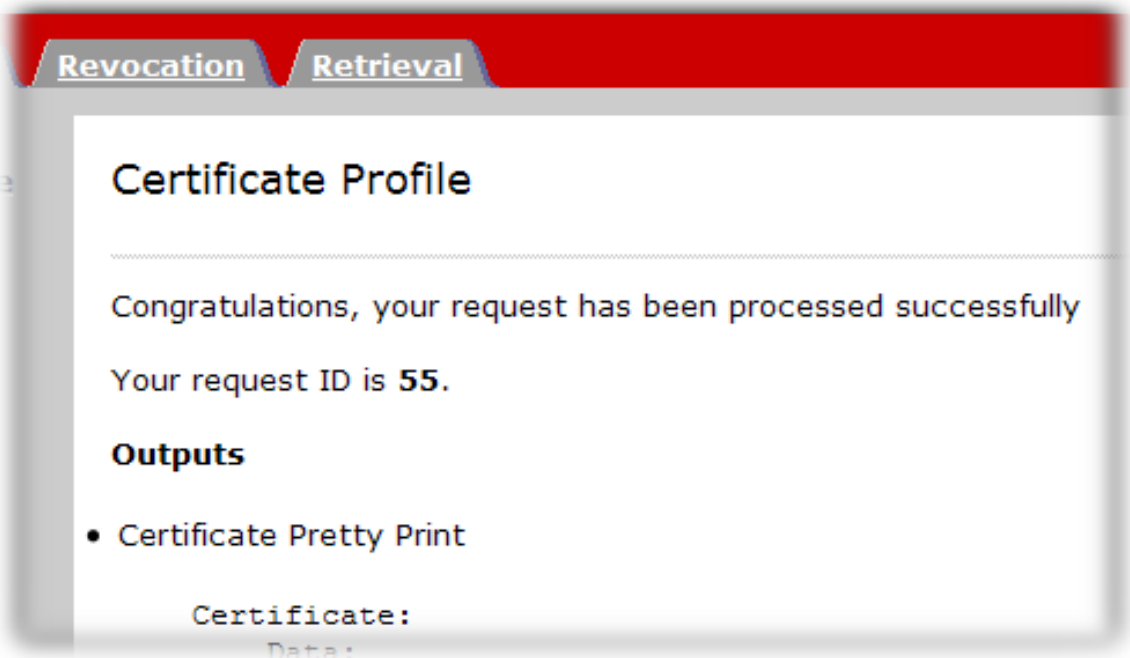
   ```
   https://server.example.com:8443/ca/ee/ca
   ```

2. Click the name of the renewal form to use.
3. Enter the serial number of the certificate to renew. This can be in decimal or hexadecimal form.



4. Click the renew button.

5. The request is submitted. For directory-based renewals, the renewed certificate is automatically returned. Otherwise, the renewal request will be approved by an agent.



5.4.1.1.2. Certificate-Based Renewal

Some user certificates are stored directly in your browser, so some renewal forms will simply check your browser certificate database for a certificate to renew. If a certificate can be renewed, then the CA automatically approved and reissued it.

IMPORTANT

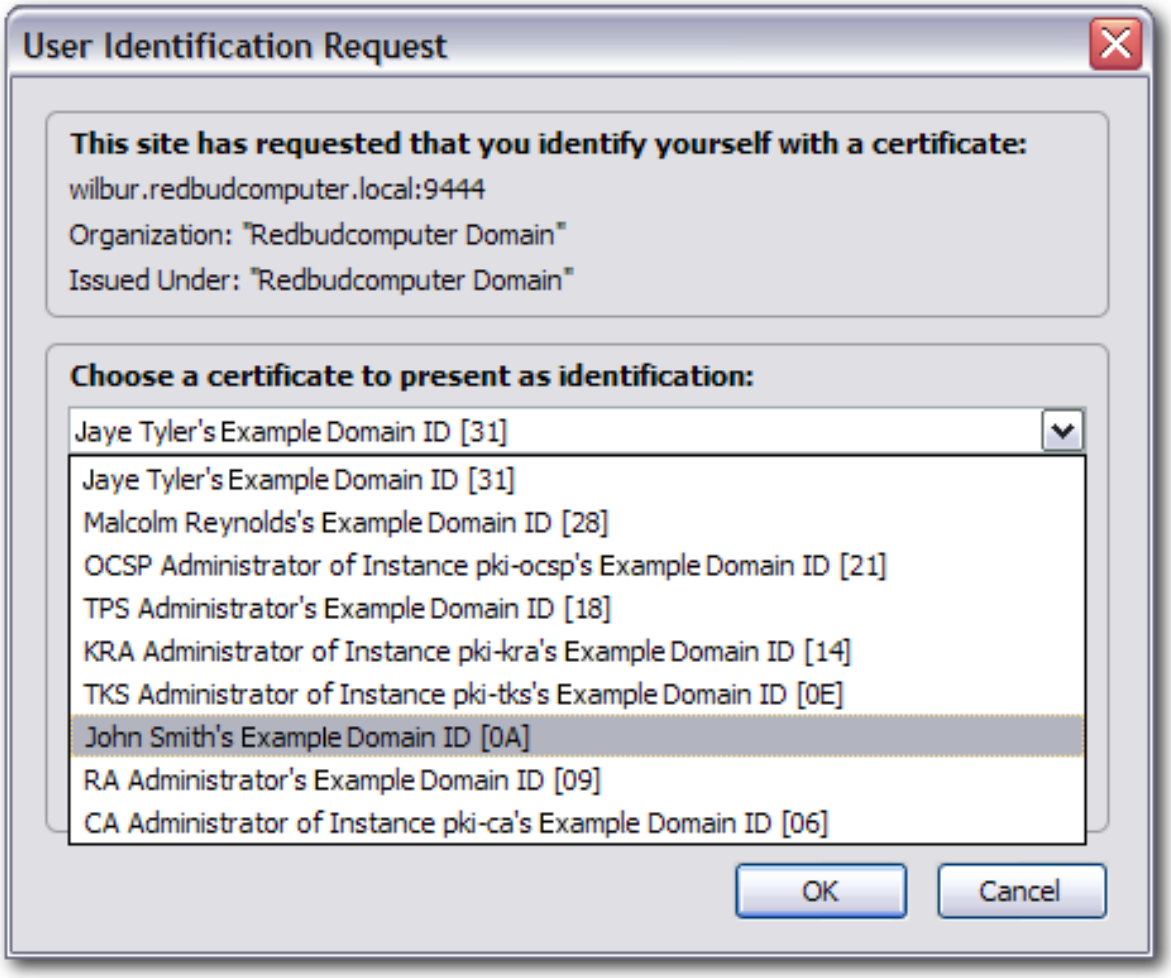
If the certificate which is being renewed has *already* expired, then it probably cannot be used for certificate-based renewal. The browser client may disallow any SSL client authentication with an expired certificate.

In that case, the certificate must be renewed using one of the other renewal methods.

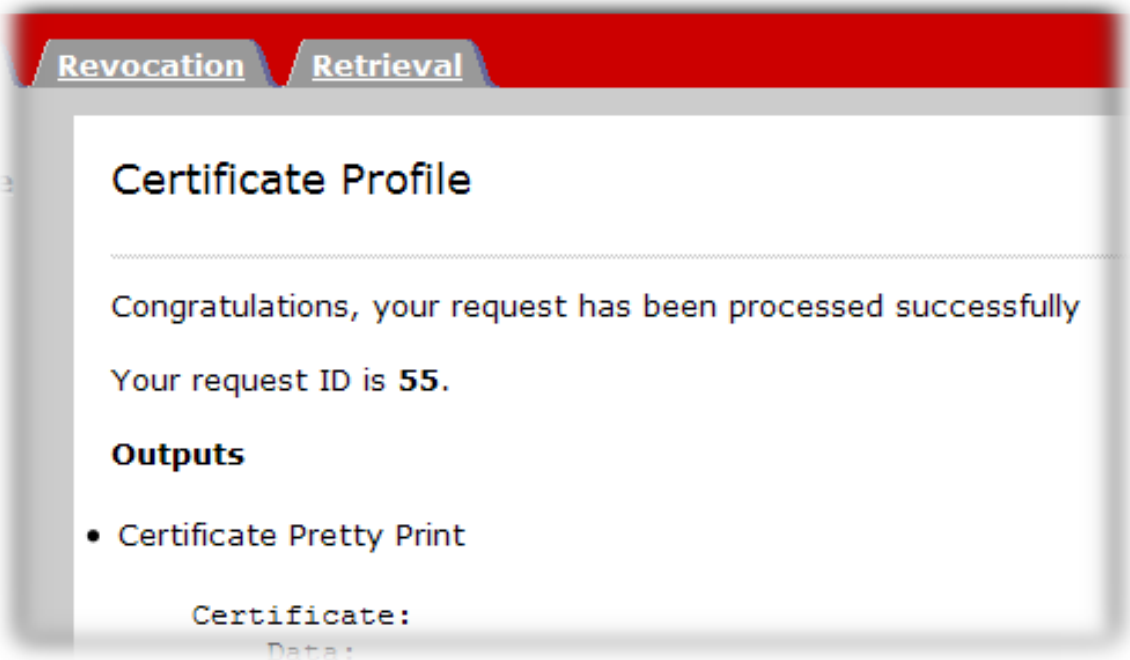
1. Open the end-entities services page for the CA which issued the certificate (or its clone).

   ```
   https://server.example.com:8443/ca/ee/ca
   ```

2. Click the name of the renewal form to use.
3. There is no input field, so click the **Renew** button.
4. When prompted, select the certificate to renew.



5. The request is submitted and the renewed certificate is automatically returned.



5.4.1.2. Renewal by generating CSR with same keys

Sometimes, the original CSR might not be available. The **certutil** tool allows one to regenerate a CSR with the same keys, provided that the key pair is in the NSS database. This can be achieved by doing the following:

1. Find the corresponding key id in the NSS db:

```
Certutil -d <nssdb dir> -K
```

2. Generate a CSR using a specific key:

```
Certutil -d <nssdb dir> -R -k <key id> -s <subject DN> -o <CSR output file>
```

Alternatively, instead of *keyid*, if a key is associated with a certificate in the NSS db, *nickname* could be used:

- Generate a CSR using an existing nickname:

```
Certutil -d <nssdb dir> -R -k <nickname> -s <subject DN> -o <CSR output file>
```

5.4.2. Renewal by Re-keying Certificates

Since renewal by *re-keying* is basically generating a new CSR with the same info as the old certificate, just follow any one of the methods described in Section 5.2, "Creating Certificate Signing Requests" . Be mindful to enter the same information as the old certificate.

5.5. SUBMITTING CERTIFICATE REQUESTS USING CMC

This section describes the procedure to enroll a certificate using Certificate Management over CMS (CMC).

For general information about configuration and the workflow of enrolling certificates using CMC, see:

- The *Configuration for CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.
- The *Enrolling with CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.
- CMCRequest(1) man page
- CMCResponse(1) man page

CMC enrollment is possible in various ways to meet the requirements for different scenarios. Section 5.5.2, "The CMC Enrollment Process" supplements the *Enrolling with CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* with more details. Additionally, the Section 5.5.3, "Practical CMC Enrollment Scenarios" section enables administrators to decide which mechanisms should be used in which scenario.

5.5.1. Using CMC Enrollment

CMC enrollment allows an enrollment client to use a CMCAuth plug-in for authentication, by which the certificate request is pre-signed with an agent certificate. The Certificate Manager automatically issues certificates when a valid request signed with the agent certificate is received.

NOTE

CMC enrollments are enabled by default. It should not be necessary to enable the CMC enrollment authentication plug-ins or profiles unless the configuration has been changed.

The CMCAuth authentication plug-in also provides CMC revocation for the client. CMC revocation allows the client to have the certificate request signed by the agent certificate, and then send such a request to the Certificate Manager. The Certificate Manager automatically revokes certificates when a valid request signed with the agent certificate is received. CMC revocation can be created with the **CMCRevoke** command line tool. For more information about **CMCRevoke**, see Section 7.2, "Performing a CMC Revocation".

A CMC request can be submitted through browser end-entities forms or using a tool such as **HttpClient** to post the request to the appropriate profile. The **CMCRequest** tool generates a signed certificate request which can then be submitted using the **HttpClient** tool or the browser end-entities forms to enroll and receive the certificate automatically and immediately.

The **CMCRequest** tool has a simple command syntax, with all the configuration given in the **.cfg** input file:

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

A single CMC enrollment can also be created using the **CMCEnroll** tool, with the following syntax:

```
CMCEnroll -d /agent's/certificate/directory -h password -n cert_nickname -r certrequest.file -p
certDB_passwd [-c "comment"]
```

These tools are described in more detail in the **CMCEnroll(1)** man page.

NOTE

Surround values that include spaces in quotation marks.

5.5.1.1. Testing CMCEnroll

1. Create a certificate request using the **certutil** tool.
2. Copy the PKCS #10 ASCII output to a text file.
3. Run the CMCEnroll utility.

   For example, if the input file called **request34.txt**, the agent certificate is stored in the browser databases, the certificate common name of the agent certificate is **CertificateManagerAgentsCert**, and the password for the certificate database is **secret**, the command is as follows:

   ```
   CMCEnroll -d ~jsmith/.mozilla/firefox/1234.jsmith -n "CertificateManagerAgentsCert" -r
   /export/requests/request34.txt -p secret
   ```

   The output of this command is stored in a file with the same filename with **.out** appended to the filename.
4. Submit the signed certificate through the end-entities page.

1. Open the end-entities page.

   ```
   https://server.example.com:8443/ca/ee/ca
   ```

2. Select the CMC enrollment form from the list of certificate profiles.
3. Paste the content of the output file into the **Certificate Request** text area of this form.
4. Remove **-----BEGIN NEW CERTIFICATE REQUEST-----** and **----END NEW CERTIFICATE REQUEST-----** from the pasted content.
5. Fill in the contact information, and submit the form.

5. The certificate is immediately processed and returned.
6. Use the agent page to search for the new certificate.

5.5.2. The CMC Enrollment Process

Use the following general procedure to request and issue a certificate using CMC:

1. Create a Certificate Signing Request (CSR) in one of the following formats:
   - PKCS #10 format
   - Certificate Request Message Format (CRMF) format

   For details about creating CSRs in these formats, see Section 5.2, “Creating Certificate Signing Requests”.

2. Import the admin certificate into the client NSS database. For example:
   - Execute the command below to extract the admin client certificate from the **.p12** file:

     ```
     $ openssl pkcs12 -in /root/.dogtag/instance/ca_admin_cert.p12 -clcerts -nodes -nokeys -
     out /root/.dogtag/instance/ca_admin_cert.crt
     ```

   - Validate and import the admin client certificate according to guidance in *Managing Certificate/Key Crypto Token* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*:

     ```
     $ PKICertImport -d . -n "CA Admin - Client Certificate" -t ",," -a -i
     /root/.dogtag/instance/ca_admin_cert.crt -u C
     ```

     IMPORTANT

     Make sure all intermediate certificates and the root CA certificate have been imported before importing the CA Admin client certificate.

   - Import the private keys associated with the certificates.

     ```
     $ pki -c password pkcs12-import --pkcs12-file /root/.dogtag/instance/ca_admin_cert.p12 -
     -pkcs12-password-file /root/.dogtag/instance/ca/pkcs12_password.conf
     ```

3. Create a configuration file for a CMC request, such as **/home/user_name/cmc-request.cfg**, with the following content:

```
# NSS database directory where CA agent certificate is stored
dbdir=/home/user_name/.dogtag/nssdb/

# NSS database password
password=password

# Token name (default is internal)
tokenname=internal

# Nickname for signing certificate
nickname=subsystem_admin

# Request format: pkcs10 or crmf
format=pkcs10

# Total number of PKCS10/CRMF requests
numRequests=1

# Path to the PKCS10/CRMF request
# The content must be in Base-64 encoded format.
# Multiple files are supported. They must be separated by space.
input=/home/user_name/file.csr

# Path for the CMC request
output=/home/user_name/cmc-request.bin
```

   For further details, see the CMCRequest(1) man page.

4. Create the CMC request:

```
$ CMCRequest /home/user_name/cmc-request.cfg
```

   If the command succeeds, the **CMCRequest** utility stored the CMC request in the file specified in the ***output*** parameter in the request configuration file.

5. Create a configuration file for **HttpClient**, such as **/home/user_name/cmc-submit.cfg**, which you use in a later step to submit the CMC request to the CA. Add the following content to the created file:

```
# PKI server host name
host=server.example.com

# PKI server port number
port=8443

# Use secure connection
secure=true

# Use client authentication
clientmode=true

# NSS database directory where the CA agent certificate is stored.
```

```
dbdir=/home/user_name/.dogtag/nssdb/

# NSS database password
password=password

# Token name (default: internal)
tokenname=internal

# Nickname of signing certificate
nickname=subsystem_admin

# Path for the CMC request
input=/home/user_name/cmc-request.bin

# Path for the CMC response
output=/home/user_name/cmc-response.bin
```

IMPORTANT

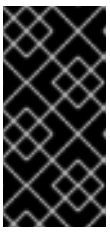
The nickname of the certificate specified in the ***nickname*** parameter must match the one previously used for the CMC request.

6. Depending on what type of certificate you request, add the following parameter to the configuration file created in the previous step:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=profile_name
```

For example, for a CA signing certificate:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCcaCert
```



IMPORTANT

When an agent submits the CMC request in the next step, the profile specified in this parameter must use the **CMCAuth** authentication plug-in. Whereas in user-initiated enrollments, the profile must use the **CMCUserSignedAuth** plug-in. For further details, see the Section 10.3, "CMC Authentication Plug-ins" .

7. Submit the CMC request to the CA:

```
$ HttpClient /home/user_name/cmc-submit.cfg
```

8. To convert the CMC response to a PKCS #7 certificate chain, pass the CMC response file to the ***-i*** parameter of the **CMCResponse** utility. For example:

```
$ CMCResponse -i /home/user_name/cmc-response.bin -o /home/user_name/cert_chain.crt
```

5.5.3. Practical CMC Enrollment Scenarios

This section describes frequent practical usage scenarios and their workflows to enable CA administrators to decide which CMC method to use in which situation.

For a general process of enrolling a certificate using CMC, see Section 5.5.2, "The CMC Enrollment Process".

5.5.3.1. Obtaining System and Server Certificates

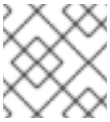
If a service, such as LDAP or a web server, requires a TLS server certificate, the administrator of this server creates a CSR based on the documentation of the service and sends it to the CA's agent for approval. Use the procedure described in Section 5.5.2, "The CMC Enrollment Process" for this process. Additionally, consider the following requirements:

Enrollment Profiles

The agent must either use one of the existing CMC profiles listed in Section 10.3, "CMC Authentication Plug-ins", or, alternatively, create a custom profile that uses the **CMCAuth** authentication mechanism.

CMC Signing Certificate

For system certificates, the CA agent must generate and sign the CMC request. For this, set the ***nickname*** parameter in the **CMCRequest** configuration file to the nickname of the CA agent.



NOTE

The CA agent must have access to its own private key.

HttpClient TLS Client Nickname

Use the same certificate for signing in the **CMCRequest** utility's configuration file as for TLS client authentication in the configuration file for **HttpClient**.

HttpClient *servlet* Parameter

The ***servlet*** in the configuration file passed to the **HttpClient** utility refers to the CMC servlet and the enrollment profile which handles the request.

Depending on what type of certificate you request, add one of the following entries to the configuration file created in the previous step:

- For a CA signing certificate:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCcaCert
```

- For a KRA transport certificate:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCkraTransportCert
```

- For a OCSP signing certificate:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCocspCert
```

- For a audit signing certificate:

```
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCauditSigningCert
```

- For a subsystem certificate:
  - For RSA certificates:

    ```
    servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCsubsystemCert
    ```

  - For ECC certificates:

    ```
    servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCECCsubsystemCert
    ```

- For a TLS server certificate:
  - For RSA certificates:

    ```
    servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCserverCert
    ```

  - For ECC certificates:

    ```
    servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCECCserverCert
    ```

- For an admin certificate:

  ```
  servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caFullCMCUserCert
  ```

Further details:

- When an agent pre-signs a CSR, the Proof of Identification is considered established because the agent examines the CSR for identification. No additional CMC-specific identification proof is required.
- PKCS #10 files already provide Proof of Possession information and no additional Proof of Possession (POP) is required.
- In agent pre-approved requests, the **PopLinkWittnessV2** feature must be disabled because the identification is checked by the agent.

5.5.3.2. Obtaining the First Signing Certificate for a User

There are two ways to approve a user's first signing certificate:

- An agent signs the CMC request. See Section 5.5.3.2.1, “Signing a CMC Request with an Agent Certificate”.
- Certificate enrollment is authenticated by using a Shared Secret. See Section 5.5.3.2.2, “Authenticating for Certificate Enrollment Using a Shared Secret”.

5.5.3.2.1. Signing a CMC Request with an Agent Certificate

The process for signing a CMC request with an agent certificate is the same as for system and server certificates described in Section 5.5.3.1, “Obtaining System and Server Certificates” . The only difference is that the user creates the CSR and sends it to a CA agent for approval.

5.5.3.2.2. Authenticating for Certificate Enrollment Using a Shared Secret

When a user wants to obtain the first signing certificate and the agent cannot approve the request as described in Section 5.5.3.2.1, "Signing a CMC Request with an Agent Certificate" , you can use a Shared Token. With this token, the user can obtain the first signing certificate. This certificate can then be used to sign other certificates of the user.

In this scenario, use the Shared Secret mechanism to obtain the first signing certificate of the user. Use the following information together with Section 5.5.2, "The CMC Enrollment Process":

1. Create a Shared Token either as the user or CA administrator. For details, see *The Shared Secret Workflow* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

   Note that:

   - If the user created the token, the user must send the token to the CA administrator.
   - If the CA administrator created the token, the administrator must share the password used to generate the token with the user. Use a secure way to transmit the password.

2. As the CA administrator, add the Shared Token to the user entry in LDAP. For details, see Section 10.4.2.1, "Adding a CMC Shared Secret to a User Entry for Certificate Enrollment" and the *Enabling the CMC Shared Secret Feature* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

3. Use the following parameters in the configuration file passed to the **CMCRequest** utility:

   - ***identification.enable***
   - ***witness.sharedSecret***
   - ***identityProofV2.enable***
   - ***identityProofV2.hashAlg***
   - ***identityProofV2.macAlg***
   - ***request.useSharedSecret***
   - ***request.privKeyId***

4. If required by the CA, additionally use the following parameters in the configuration file passed to the **CMCRequest** utility:

   - ***popLinkWitnessV2.enable***
   - ***popLinkWitnessV2.keyGenAlg***
   - ***popLinkWitnessV2.macAlg***

5.5.3.3. Obtaining an Encryption-only Certificate for a User

This section describes the workflow for obtaining an encryption-only certificate which is signed with an existing user signing certificate:

NOTE

If a user owns multiple certificates for different usages, where one is signing, the user must obtain the signing certificate first. Once the user owns a signing certificate, it can be used for Proof Of Origin without requiring to set up and rely on the CMC Shared Secret mechanism.

For details about obtaining a user's first signing certificate, see Section 5.5.3.2, “Obtaining the First Signing Certificate for a User”.

As a user:

1. Use the cryptographic token stored in a Network Security Services (NSS) database or on a smart card that contains the user's signing certificate and keys.
2. Generate the CSR in PKCS #10 or the CRMF format.

   NOTE

   Use the CRMF format, if key archival is required.

3. Generate the CMC request.

   Since this is an encryption-only certificate, the private key is not able to sign. Therefore, Proof Of Possession (POP) is not included. For this reason, the enrollment requires two steps: If the initial request is successful, results in a CMC status with the **EncryptedPOP** control. The user then uses the response and generates a CMC request that contains the **DecryptedPOP** control and submits it in the second step.

   a. For the first step, in addition to the default parameters, the user must set the following parameters in the configuration file passed to the **CMCRequest** utility:

      - ***identification.enable***
      - ***witness.sharedSecret***
      - ***identityProofV2.enable***
      - ***identityProofV2.hashAlg***
      - ***identityProofV2.macAlg***
      - ***popLinkWitnessV2.enable*** if required by the CA
      - ***popLinkWitnessV2.keyGenAlg*** if required by the CA
      - ***popLinkWitnessV2.macAlg*** if required by the CA
      - ***request.privKeyId***

      For details, see the CMCRequest(1) man page.

      The response contains:

      - A CMC encrypted POP control

- The **CMCStatusInfoV2** control with the **POP required** error
- The request ID

b. For the second step, in addition to the default parameters, the user must set the following parameters in the configuration file passed to the **CMCRequest** utility:

- ***decryptedPop.enable***
- ***encryptedPopResponseFile***
- ***decryptedPopRequestFile***
- ***request.privKeyId***

For details, see the CMCRequest(1) man page.

5.5.3.3.1. Example on Obtaining an Encryption-only certificate with Key Archival

To perform an enrollment with key archival, generate a CMC request that contains the user's encrypted private key in the CRMF request. The following procedure assumes that the user already owns a signing certificate. The nickname of this signing certificate is set in the configuration files in the procedure.

NOTE

The following procedure describes the two-trip issuance used with encryption-only keys, which cannot be used for signing. If you use a key which can sign certificates, pass the **-q POP_SUCCESS** option instead of **-q POP_NONE** to the **CRMFPopClient** utility for a single-trip issuance.

For instructions about using **CRMFPoPClient** with **POP_SUCCESS**, see Section 5.2.1.3.1, "Using **CRMFPopClient** to Create a CSR with Key Archival" and Section 5.2.1.3.2, "Using **CRMFPopClient** to Create a CSR for SharedSecret-based CMC" .

1. Search for the KRA transport certificate. For example:

```
$ pki cert-find --name KRA_transport_certificate_subject_CN
```

2. Use the serial number of the KRA transport certificate, which you retrieved in the previous step, to store the certificate in a file. For example, to store the certificate with the *12345* serial number in the **/home/user_name/kra.cert** file:

```
$ pki cert-show 12345 --output /home/user_name/kra.cert
```

3. Use the **CRMFPopClient** utility to:
   - Create a CSR with key archival:
     1. Change to the certificate database directory of the user or entity for which the certificate is being requested, for example:

```
$ cd /home/user_name/
```

2. Use the **CRMFPopClient** utility to create a CRMF request, where the RSA private key is wrapped by the KRA transport certificate. For example, to store the request in the **/home/user_name/crmf.req** file:

```
$ CRMFPopClient -d . -p token_password -n subject_DN -q POP_NONE \
  -b /home/user_name/kra.cert -w "AES/CBC/PKCS5Padding" \
  -v -o /home/user_name/crmf.req
```

Note the ID of the private key displayed by the command. The ID is required in a later step as value in the ***request.privKeyId*** parameter in the configuration file for the second trip.

4. Create a configuration file for the **CRMRequest** utility, such as **/home/*user_name*/cmc.cfg** with the following content:

```
#numRequests: Total number of PKCS10 requests or CRMF requests.
numRequests=1

#input: full path for the PKCS10 request or CRMF request,
#the content must be in Base-64 encoded format
input=/home/user_name/crmf.req

#output: full path for the CMC request in binary format
output=/home/user_name/cmc.req

#tokenname: name of token where agent signing cert can be found
#(default is internal)
tokenname=internal

#nickname: nickname for user certificate which will be used
#to sign the CMC full request.
nickname=signing_certificate

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
dbdir=/home/user_name/.dogtag/nssdb/

#password: password for cert9.db which stores the agent certificate
password=password

#format: request format, either pkcs10 or crmf
format=crmf
```

5. Create the CMC request:

```
$ CMCRequest /home/user_name/cmc.cfg
```

If the command succeeds, the **CMCRequest** utility stored the CMC request in the file specified in the ***output*** parameter in the request configuration file.

6. Create a configuration file for **HttpClient**, such as **/home/user_name/cmc-submit.cfg**, which you use in a later step to submit the CMC request to the CA. Add the following content to the created file:

```
#host: host name for the http server
host=server.example.com
```

```
#port: port number
port=8443

#secure: true for secure connection, false for nonsecure connection
secure=true

#input: full path for the enrollment request, the content must be in
#binary format
input=/home/user_name/cmc.req

#output: full path for the response in binary format
output=/home/user_name/cmc-response_round_1.bin

#tokenname: name of token where TLS client authentication cert can be found
#(default is internal)
#This parameter will be ignored if secure=false
tokenname=internal

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
#This parameter will be ignored if secure=false
dbdir=/home/user_name/.dogtag/nssdb/

#clientmode: true for client authentication, false for no client authentication
#This parameter will be ignored if secure=false
clientmode=true

#password: password for cert9.db
#This parameter will be ignored if secure=false and clientauth=false
password=password

#nickname: nickname for client certificate
#This parameter will be ignored if clientmode=false
nickname=signing_certificate

#servlet: servlet name
servlet=/ca/ee/ca/profileSubmitUserSignedCMCFull?profileId=caFullCMCUserSignedCert
```

7. Submit the CMC request to the CA:

```
$ HttpClient /home/user_name/cmc-submit.cfg
```

If the command succeeds, the **HTTPClient** utility stored the CMC response in the file specified in the ***output*** parameter in the configuration file.

8. Verify the response by passing the response file to the **CMCResponse** utility. For example:

```
$ CMCResponse -d /home/user_name/.dogtag/nssdb/ -i /home/user_name/cmc-
response_round_1.bin
```

If the first trip was successful, **CMCResponse** displays output similar to the following:

```
Certificates:
  Certificate:
    Data:
```

```
        Version:  v3
        Serial Number: 0x1
        Signature Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
        Issuer: CN=CA Signing Certificate,OU=pki-tomcat,O=unknown00262DFC6A5E Security
Domain
        Validity:
          Not Before: Wednesday, May 17, 2017 6:06:50 PM PDT America/Los_Angeles
          Not  After: Sunday, May 17, 2037 6:06:50 PM PDT America/Los_Angeles
        Subject: CN=CA Signing Certificate,OU=pki-tomcat,O=unknown00262DFC6A5E Security
Domain
...
Number of controls is 3
Control #0: CMC encrypted POP
  OID: {1 3 6 1 5 5 7 7 9}
   encryptedPOP decoded
Control #1: CMCStatusInfoV2
  OID: {1 3 6 1 5 5 7 7 25}
  BodyList: 1
  OtherInfo type: FAIL
   failInfo=POP required
Control #2: CMC ResponseInfo
  requestID: 15
```

9. For the second trip, create a configuration file for **DecryptedPOP**, such as **/home/user_name/cmc_DecryptedPOP.cfg**, which you use in a later step. Add the following content to the created file:

```
#numRequests: Total number of PKCS10 requests or CRMF requests.
numRequests=1

#input: full path for the PKCS10 request or CRMF request,
#the content must be in Base-64 encoded format
#this field is actually unused in 2nd trip
input=/home/user_name/crmf.req

#output: full path for the CMC request in binary format
#this field is actually unused in 2nd trip
output=/home/user_name/cmc2.req

#tokenname: name of token where agent signing cert can be found
#(default is internal)
tokenname=internal

#nickname: nickname for agent certificate which will be used
#to sign the CMC full request.
nickname=signing_certificate

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
dbdir=/home/user_name/.dogtag/nssdb/

#password: password for cert9.db which stores the agent
#certificate
password=password

#format: request format, either pkcs10 or crmf
format=crmf
```

```
decryptedPop.enable=true
encryptedPopResponseFile=/home/user_name/cmc-response_round_1.bin
request.privKeyId=-25aa0a8aad395ebac7e6a19c364f0dcb5350cfef
decryptedPopRequestFile=/home/user_name/cmc.DecryptedPOP.req
```

10. Create the **DecryptPOP** CMC request:

```
$ CMCRequest /home/user_name/cmc.DecryptedPOP.cfg
```

If the command succeeds, the **CMCRequest** utility stored the CMC request in the file specified in the ***decryptedPopRequestFile*** parameter in the request configuration file.

11. Create a configuration file for **HttpClient**, such as **/home/user_name/decrypted_POP_cmc-submit.cfg**, which you use in a later step to submit the **DecryptedPOP** CMC request to the CA. Add the following content to the created file:

```
#host: host name for the http server
host=server.example.com

#port: port number
port=8443

#secure: true for secure connection, false for nonsecure connection
secure=true

#input: full path for the enrollment request, the content must be in binary format
input=/home/user_name/cmc.DecryptedPOP.req

#output: full path for the response in binary format
output=/home/user_name/cmc-response_round_2.bin

#tokenname: name of token where TLS client authentication cert can be found (default is
internal)
#This parameter will be ignored if secure=false
tokenname=internal

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
#This parameter will be ignored if secure=false
dbdir=/home/user_name/.dogtag/nssdb/

#clientmode: true for client authentication, false for no client authentication
#This parameter will be ignored if secure=false
clientmode=true

#password: password for cert9.db
#This parameter will be ignored if secure=false and clientauth=false
password=password

#nickname: nickname for client certificate
#This parameter will be ignored if clientmode=false
nickname=singing_certificate

#servlet: servlet name
servlet=/ca/ee/ca/profileSubmitUserSignedCMCFull?profileId=caFullCMCUserCert
```

12. Submit the **DecryptedPOP** CMC request to the CA:

```
$ HttpClient /home/user_name/decrypted_POP_cmc-submit.cfg
```

If the command succeeds, the **HTTPClient** utility stored the CMC response in the file specified in the ***output*** parameter in the configuration file.

13. To convert the CMC response to a PKCS #7 certificate chain, pass the CMC response file to the ***-i*** parameter of the **CMCResponse** utility. For example:

```
$ CMCResponse -i /home/user_name/cmc-response_round_2.bin -o
/home/user_name/certs.p7
```

Alternatively, to display the individual certificates in PEM format, pass the **-v** to the utility.

If the second trip was successful, **CMCResponse** displays output similar to the following:

```
Certificates:
  Certificate:
    Data:
      Version:  v3
      Serial Number: 0x2D
      Signature Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
      Issuer: CN=CA Signing Certificate,OU=pki-tomcat,O=unknown00262DFC6A5E Security
Domain
      Validity:
        Not Before: Thursday, June 15, 2017 3:43:45 PM PDT America/Los_Angeles
        Not  After: Tuesday, December 12, 2017 3:43:45 PM PST America/Los_Angeles
      Subject: CN=user_name,UID=example,OU=keyArchivalExample
...
Number of controls is 1
Control #0: CMCStatusInfo
  OID: {1 3 6 1 5 5 7 7 1}
  BodyList: 1
  Status: SUCCESS
```

5.6. PERFORMING BULK ISSUANCE

There can be instances when an administrator needs to submit and generate a large number of certificates simultaneously. A combination of tools supplied with Certificate System can be used to post a file containing certificate requests to the CA. This example procedure uses the **PKCS10Client** command to generate the requests and the **sslget** command to send the requests to the CA.

1. Since this process is scripted, multiple variables need to be set to identify the CA (host, port) and the items used for authentication (the agent certificate and certificate database and password). For example, set these variables for the session by exporting them in the terminal:

```
export d=/var/tmp/testDir
export p=password
export f=/var/tmp/server.csr.txt
export nick="CA agent cert"
export cahost=1.2.3.4
export caport=8443
```

NOTE

The local system must have a valid security database with an agent's certificate in it. To set up the databases:

1. Export or download the agent user certificate and keys from the browser and save to a file, such as **agent.p12**.
2. If necessary, create a new directory for the security databases.

```
mkdir ${d}
```

3. If necessary, create new security databases.

```
certutil -N -d ${d}
```

4. Stop the Certificate System instance.

```
pki-server stop instance_name
```

5. Use **pk12util** to import the certificates.

```
# pk12util -i /tmp/agent.p12 -d ${d} -W p12filepassword
```

If the procedure is successful, the command prints the following output:

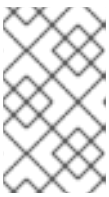
```
pk12util: PKCS12 IMPORT SUCCESSFUL
```

6. Start the Certificate System instance.

```
pki-server start instance_name
```

2. Two additional variables must be set. A variable that identify the CA profile to be used to process the requests, and a variable that is used to send a post statement to supply the information for the profile form.

```
export
post="cert_request_type=pkcs10&xmlOutput=true&profileId=caAgentServerCert&cert_request=
"
export url="/ca/ee/ca/profileSubmitSSLClient"
```



NOTE

This example submits the certificate requests to the **caAgentServerCert** profile (identified in the **profileId** element of the **post** statement. Any certificate profile can be used, including custom profiles.

3. Test the variable configuration.

```
echo ${d} ${p} ${f} ${nick} ${cahost} ${caport} ${post} ${url}
```

4. Generate the certificate requests using (for this example) **PKCS10Client**:

```
time for i in {1..10}; do /usr/bin/PKCS10Client -d ${d} -p ${p} -o ${f}.${i} -s
"cn=testms${i}.example.com"; cat ${f}.${i} >> ${f}; done

perl -pi -e 's/\r\n//;s/\+/%2B/g;s/\//%2F/g' ${f}

wc -l ${f}
```

5. Submit the bulk certificate request file created in step 4 to the CA profile interface using **sslget**. For example:

```
cat ${f} | while read thisreq; do /usr/bin/sslget -n "${nick}" -p ${p} -d ${d} -e ${post}${thisreq} -
v -r ${url} ${cahost}:${caport}; done
```

5.7. ENROLLING A CERTIFICATE ON A CISCO ROUTER

Simple Certificate Enrollment Protocol (SCEP), designed by Cisco, is a way for a router to communicate a certificate issuing authority, such as a CA, to enroll certificates for the router.

Normally, a router installer enters the CA's URL and a challenge password (also called a one-time PIN) into the router and issues a command to initiate the enrollment. The router then communicates with the CA over SCEP to generate, request, and retrieve the certificate. The router can also check the status of a pending request using SCEP.

5.7.1. Enabling SCEP Enrollments

For security reasons, SCEP enrollments are disabled by default in the CA. To allow routers to be enrolled, SCEP enrollments must be manually enabled for the CA.

1. Stop the CA server, so that you can edit the configuration files.

```
pki-server stop instance_name
```

2. Open the CA's **CS.cfg** file.

```
vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

3. Set the ***ca.scep.enable*** to true. If the parameter is not present, then add a line with the parameter.

```
ca.scep.enable=true
```

4. Restart the CA server.

```
pki-server start instance_name
```

5.7.2. Configuring Security Settings for SCEP

Several different parameters allow administrators to set specific security requirements for SCEP connections, such as not using the same certificate for enrollment authentication and regular certificate enrollments, or setting allowed encryption algorithms to prevent downgrading the connection strength.

These parameters are listed in Table 5.1, "Configuration Parameters for SCEP Security" .

Table 5.1. Configuration Parameters for SCEP Security

Parameter	Description
ca.scep.encryptionAlgorithm	Sets the default or preferred encryption algorithm.
ca.scep.allowedEncryptionAlgorithms	Sets a comma-separated list of allowed encryption algorithms.
ca.scep.hashAlgorithm	Sets the default or preferred hash algorithm.
ca.scep.allowedHashAlgorithms	Sets a comma-separated list of allowed hash algorithms.
ca.scep.nickname	Gives the nickname of the certificate to use for SCEP communication. The default is to use the CA's key pair and certificate unless this parameter is set.
ca.scep.nonceSizeLimit	Sets the maximum nonce size, in bytes, allowed for SCEP requests. The default is 16 bytes.

To set security settings for connections for SCEP enrollments:

1. Stop the CA server, so that you can edit the configuration files.

```
pki-server stop instance_name
```

2. Open the CA's **CS.cfg** file.

```
vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

3. Set the desired security parameters, as listed in Table 5.1, "Configuration Parameters for SCEP Security". If the parameter is not already present, then add it to the **CS.cfg** file.

```
ca.scep.encryptionAlgorithm=DES3
ca.scep.allowedEncryptionAlgorithms=DES3
ca.scep.hashAlgorithm=SHA1
ca.scep.allowedHashAlgorithms=SHA1,SHA256,SHA512
ca.scep.nickname=Server-Cert
ca.scep.nonceSizeLimit=20
```

4. Restart the CA server.

```
pki-server start instance_name
```

5.7.3. Configuring a Router for SCEP Enrollment

NOTE

Not all versions of router IOS have the relevant crypto features. Make sure that the firmware image has the Certification Authority Interoperability feature.
Certificate System SCEP support was tested on a Cisco 2611 router running IOS C2600 Software (C2600-JK9S-M), version 12.2(40), RELEASE SOFTWARE (fc1).

Before enrolling SCEP certificates on the router, make sure that the router is appropriately configured:

- The router must be configured with an IP address, DNS server, and routing information.
- The router's date/time must be correct.
- The router's hostname and dnsname must be configured.

See the router documentation for instructions on configuring the router hardware.

5.7.4. Generating the SCEP Certificate for a Router

The following procedure details how to generate the SCEP certificate for a router.

1. Pick a random PIN.
2. Add the PIN and the router's ID to the **flatfile.txt** file so that the router can authenticate directly against the CA. For example:

   ```
   vim /var/lib/pki/instance_name/ca/conf/flatfile.txt

   UID:172.16.24.238
   PWD:Uojs93wkfd0IS
   ```

   Be sure to insert an empty line after the **PWD** line.

   The router's IP address can be an IPv4 address or an IPv6 address.

   Using flat file authentication is described in Section 10.2.4, “Configuring Flat File Authentication”.

3. Log into the router's console. For this example, the router's name is **scep**:

   ```
   scep>
   ```

4. Enable privileged commands.

   ```
   scep> enable
   ```

5. Enter configuration mode.

   ```
   scep# conf t
   ```

6. Import the CA certificate for every CA in the certificate chain, starting with the root. For example, the following command sequence imports two CA certificates in the chain into the router:

   ```
   ```

```
scep(config)# crypto ca trusted-root1
scep(ca-root)# root CEP http://server.example.com:8080/ca/cgi-bin/pkiclient.exe
scep(ca-root)# crl optional
scep(ca-root)# exit
scep(config)# cry ca authenticate 1
scep(config)# crypto ca trusted-root0
scep(ca-root)# root CEP http://server.example.com:8080/ca/cgi-bin/pkiclient.exe
scep(ca-root)# crl optional
scep(ca-root)# exit
scep(config)# cry ca authenticate 0
```

7. Set up a CA identity, and enter the URL to access the SCEP enrollment profile. For example, for the CA:

```
scep(config)# crypto ca identity CA
scep(ca-identity)# enrollment url http://server.example.com:8080/ca/cgi-bin
scep(ca-identity)# crl optional
```

8. Get the CA's certificate.

```
scep(config)# crypto ca authenticate CA
Certificate has the following attributes:
Fingerprint: 145E3825 31998BA7 F001EA9A B4001F57
% Do you accept this certificate? [yes/no]: yes
```

9. Generate RSA key pair.

```
scep(config)# crypto key generate rsa
The name for the keys will be: scep.server.example.com
Choose the size of the key modulus in the range of 360 to 2048 for your
General Purpose Keys. Choosing a key modulus greater than 512 may take
a few minutes.

How many bits in the modulus [512]:
Generating RSA keys ...
[OK]
```

10. Lastly, generate the certificate on the router.

```
scep(config)# crypto ca enroll CA
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
password to the CA Administrator in order to revoke your certificate.
For security reasons your password will not be saved in the configuration.
Please make a note of it.

Password: secret
Re-enter password: secret

% The subject name in the certificate will be: scep.server.example.com
% Include the router serial number in the subject name? [yes/no]: yes
% The serial number in the certificate will be: 57DE391C
% Include an IP address in the subject name? [yes/no]: yes
```

```
% Interface: Ethernet0/0
% Request certificate from CA? [yes/no]: yes
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the fingerprint.

% Fingerprint:D89DB555 E64CC2F7 123725B4 3DBDF263

Jan 12 13:41:17.348: %CRYPTO-6-CERTRET: Certificate received from Certificate
```

11. Close configuration mode.

```
scep(config)# exit
```

12. To make sure that the router was properly enrolled, list all of the certificates stored on the router.

```
scep# show crypto ca certificates
Certificate
 Status: Available
 Certificate Serial Number: 0C
 Key Usage: General Purpose
 Issuer:
 CN = Certificate Authority
  O = Sfbay Red hat Domain 20070111d12
 Subject Name Contains:
 Name: scep.server.example.com
 IP Address: 10.14.1.94
 Serial Number: 57DE391C
 Validity Date:
 start date: 21:42:40 UTC Jan 12 2007
 end date: 21:49:50 UTC Dec 31 2008
 Associated Identity: CA

CA Certificate
 Status: Available
 Certificate Serial Number: 01
 Key Usage: Signature
 Issuer:
 CN = Certificate Authority
  O = Sfbay Red hat Domain 20070111d12
 Subject:
 CN = Certificate Authority
  O = Sfbay Red hat Domain 20070111d12
 Validity Date:
 start date: 21:49:50 UTC Jan 11 2007
 end date: 21:49:50 UTC Dec 31 2008
 Associated Identity: CA
```

5.7.5. Working with Subordinate CAs

Before a router can authenticate to a CA, every CA certificate in the CA's certificate chain must be imported into the router, starting with the root. For example, the following command sequence imports two CA certificates in the chain into the router:

```
scep(config)# crypto ca trusted-root1
scep(ca-root)# root CEP http://server.example.com:8080/ca/cgi-bin/pkiclient.exe
scep(ca-root)# crl optional
scep(ca-root)# exit
scep(config)# cry ca authenticate 1
scep(config)# crypto ca trusted-root0
scep(ca-root)# root CEP http://server.example.com:8080/ca/cgi-bin/pkiclient.exe
scep(ca-root)# crl optional
scep(ca-root)# exit
scep(config)# cry ca authenticate 0
```

If the CA certificates do not have the CRL distribution point extension set, turn off the CRL requirement by setting it to **optional**:

```
scep(ca-root)# crl optional
```

After that, set up the CA identity as described in Section 5.7.4, "Generating the SCEP Certificate for a Router".

5.7.6. Re-enrolling a Router

Before a router can be re-enrolled with new certificates, the existing configuration has to be removed.

1. Remove (zeroize) the existing keys.

```
scep(config)# crypto key zeroize rsa
% Keys to be removed are named scep.server.example.com.
Do you really want to remove these keys? [yes/no]: yes
```

2. Remove the CA identity.

```
scep(config)# no crypto ca identity CA
% Removing an identity will destroy all certificates received from
the related Certificate Authority.

Are you sure you want to do this? [yes/no]: yes
% Be sure to ask the CA administrator to revoke your certificates.

No enrollment sessions are currently active.
```

5.7.7. Enabling Debugging

The router provides additional debugging during SCEP operations by enabling the debug statements.

```
scep# debug crypto pki callbacks
Crypto PKI callbacks debugging is on

scep# debug crypto pki messages
Crypto PKI Msg debugging is on

scep# debug crypto pki transactions
Crypto PKI Trans debugging is on
```

```
scep#debug crypto verbose
verbose debug output debugging is on
```

5.7.8. Issuing ECC Certificates with SCEP

By default, an ECC CA does not support SCEP out of box. However, it is possible to work around it by using a designated RSA certificate to handle each of the following two areas:

- encryption/decryption cert - designate an RSA cert having encryption/decryption capability; (scepRSAcert in the following example)
- signature cert - get an RSA cert to use on the client side for signing purpose instead of self-signed; (signingCert cert in the following example)

For example, with scepRSAcert cert being the encrypt/decrypt cert, and signingCert being the signing cert:

```
sscep enroll -c ca.crt -e scepRSAcert.crt -k local.key -r local.csr -K sign.key -O sign.crt -E 3des -S
sha256 -l cert.crt -u 'http://example.example.com:8080/ca/cgi-bin/pkiclient.exe'
```

5.8. USING CERTIFICATE TRANSPARENCY

Certificate System provides a basic version of Certificate Transparency (CT) V1 support (rfc 6962). It has the capability of issuing certificates with embedded Signed Certificate Time stamps (SCTs) from any trusted log where each deployment site choses to have its root CA cert included. You can also configure the system to support multiple CT logs. A minimum of one trusted CT log is required for this feature to work.

IMPORTANT

It is the responsibility of the deployment site to establish its trust relationship with a trusted CT log server.

For more information on how to configure Certificate Transparency, see the *Configuring Certificate Transparency* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

5.8.1. Testing Certificate Transparency

As example on how to test a CT setup, the following procedure describes an actual test against Google CT test logs. A more comprehensive test procedure would involve setting up a TLS server and test for the inclusion of its certs from its specified CT logs. However, the following serves as a quick test that checks for inclusion of the SCT extension once a certificate has been issued.

The test procedure consists in generating and submitting a Certificate Signing Request (CSR), in order to verify its SCT extension using **openssl**. The test configuration in the **CS.cfg** file is as follows:

```
ca.certTransparency.mode=enabled
ca.certTransparency.log.1.enable=true
ca.certTransparency.log.1.pubKey=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEw8i8S7qiGEs9NXv
0ZJFh6uuOm<snip>
ca.certTransparency.log.1.url=http://ct.googleapis.com:80/testtube/
ca.certTransparency.log.1.version=1
```

```
ca.certTransparency.log.2.enable=true
ca.certTransparency.log.2.pubKey=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEKATl2B3SAbxyzG
OfNRB+AytNTG<snip>
ca.certTransparency.log.2.url=http://ct.googleapis.com:80/logs/crucible/
ca.certTransparency.log.2.version=1
ca.certTransparency.log.3.enable=false
ca.certTransparency.log.3.pubKey=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEiKfWtuoWCPMEzS
KySjMjXpo38W<snip>
ca.certTransparency.log.3.url=http://ct.googleapis.com:80/logs/solera2020/
ca.certTransparency.log.3.version=1
ca.certTransparency.log.num=3
```

1. First, generate a CSR, e.g:

```
# PKCS10Client -d . -p passwd -l 2048 -n "cn=user.test.domain.com,OU=user-
TEST,O=TestDomain" -o pkcs10-TLS.req
```

2. Next, submit the CSR to an enrollment profile depending on the CT mode defined by the **ca.certTransparency.mode** parameter in **CS.cfg**:
   - if the parameter is set to *enabled*, use any enrollment profile
   - if the parameter is set to *perProfile*, use one of the CT profiles: e.g. *caServerCertWithSCT*
3. Copy the issued b64 cert into a file, e.g. **.ct1.pem**.
4. Convert the pem to binary:

```
#  AtoB ct1.pem ct1.bin
```

5. Display the DER certificate content:

```
#  openssl x509 -noout -text -inform der -in ct1.bin
```

6. Observe that the SCT extension is present, e.g:

```
        CT Precertificate SCTs:
         Signed Certificate Timestamp:
           Version   : v1 (0x0)
           Log ID    : B0:CC:83:E5:A5:F9:7D:6B:AF:7C:09:CC:28:49:04:87:
                       2A:C7:E8:8B:13:2C:63:50:B7:C6:FD:26:E1:6C:6C:77
           Timestamp : Jun 11 23:07:14.146 2020 GMT
           Extensions: none
           Signature : ecdsa-with-SHA256
                       30:44:02:20:6E:E7:DC:D6:6B:A6:43:E3:BB:8E:1D:28:
                       63:C6:6B:03:43:4E:7A:90:0F:D6:2B:E8:ED:55:1D:5F:
                       86:0C:5A:CE:02:20:53:EB:75:FA:75:54:9C:9F:D3:7A:
                       D4:E7:C6:6C:9B:33:2A:75:D8:AB:DE:7D:B9:FA:2B:19:
                       56:22:BB:EF:19:AD
         Signed Certificate Timestamp:
           Version   : v1 (0x0)
           Log ID    : C3:BF:03:A7:E1:CA:88:41:C6:07:BA:E3:FF:42:70:FC:
                       A5:EC:45:B1:86:EB:BE:4E:2C:F3:FC:77:86:30:F5:F6
           Timestamp : Jun 11 23:07:14.516 2020 GMT
```

```
        Extensions: none
        Signature : ecdsa-with-SHA256
            30:44:02:20:4A:C9:4D:EF:64:02:A7:69:FF:34:4E:41:
            F4:87:E1:6D:67:B9:07:14:E6:01:47:C2:0A:72:88:7A:
            A9:C3:9C:90:02:20:31:26:15:75:60:1E:E2:C0:A3:C2:
            ED:CF:22:A0:3B:A4:10:86:D1:C1:A3:7F:68:CC:1A:DD:
            6A:5E:10:B2:F1:8F
```

Alternatively, verify the SCT by running an asn1 dump:

```
# openssl asn1parse -i -inform der -in ct1.bin
```

and observe the hex dump, e.g:

```
  740:d=4  hl=4 l= 258 cons:    SEQUENCE
  744:d=5  hl=2 l=  10 prim:     OBJECT            :CT Precertificate SCTs
  756:d=5  hl=3 l= 243 prim:     OCTET STRING      [HEX
DUMP]:0481F000EE007500B0CC83E5A5F97D6B<snip>
```

CHAPTER 6. USING AND CONFIGURING THE TOKEN MANAGEMENT SYSTEM: TPS AND TKS

This chapter provides procedures for using hardware security modules, also called *HSMs* or *tokens*, to generate and store Certificate System instance certificates and keys.

This chapter only contains administration procedures. For general information on the concepts behind the Token Management System, see the *Red Hat Certificate System Planning, Installation and Deployment Guide*.

6.1. TPS PROFILES

NOTE

See the *TPS Profiles* section of the *Red Hat Certificate System Planning, Installation and Deployment Guide* for general information.

Unlike CA enrollment profiles, which are defined and stored in individual files or in LDAP, TPS profiles (also known as token types) are defined in the TPS configuration file, **CS.cfg**.

TPS profile (token type) configuration parameters are set in the following format:

```
op.<explicit op>.<profile id>.<implicit op>.<key type>.*
```

In the above, *<explicit op>* and *<implicit op>* are one of the explicit and implicit operations discussed in the TPS Operations section below, and *<key type>* is the name given for each certificate type.

An example configuration parameter may look like the following example:

```
op.enroll.userKey.keyGen.encryption.*
```

6.2. TPS OPERATIONS

Explicit Operations

An *explicit operation* is an operation called by a user. Explicit operations include **enroll** (**op.enroll.***), **format** (**op.format.***), and **pinReset** (**op.pinReset.***).

Implicit Operations

An *implicit operation* is an operation that takes place due to the policy or status of a token at a time when an explicit operation is being processed. Implicit operations include **keyGen** (**op.enroll.userKey.keyGen.***), **renewal** (**op.enroll.userKey.renewal.***), **update.applet** (**op.enroll.userKey.update.applet.***), and key update (**op.enroll.userKey.update.symmetricKeys.***).

Some implicit operations are controlled per key type. These include **recovery**, **serverKeygen**, and **revocation**.

The following example of a TPS profile specifies user keys to be generated on the server side:

```
op.enroll.userKey.keyGen.encryption.serverKeygen.archive=true
op.enroll.userKey.keyGen.encryption.serverKeygen.drm.conn=kra1
op.enroll.userKey.keyGen.encryption.serverKeygen.enable=true
```

Additionally, the following example tells TPS that a token whose keys are compromised should revoke the certification with revocation reason **1** during the state transition:

```
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.revokeCert=true
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.revokeCert.reason=1
```

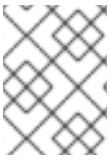
According to RFC 5280, possible revocation reasons and their codes are defined as follows:

Table 6.1. Revocation Reasons and Codes

Reason	Code
unspecified	0
keyCompromise	1
CACompromise	2
affiliationChanged	3
superseded	4
cessationOfOperation	5
certificateHold	6
removeFromCRL	8
privilegeWithdrawn	9
AACompromise	10

6.3. TOKEN POLICIES

This section provides a list of token policies that can be applied on a per token basis using the TPS UI. Ech section will show how each policy is reflected in the configuration.



NOTE

See the *Token Policies* section of the *Red Hat Certificate System Planning, Installation and Deployment Guide* for general information.

The policy is a collection of policies each separated by a semicolon ("**;**""). Each policy can be turned on or off with the keywords **YES** or **NO**. Each policy in the list below will be introduced with its default value - the action taken by TPS if the setting did not exist at all in the policy string.

RE_ENROLL=YES

This policy controls whether or not a token allows a reenroll operation. This allows an already enrolled token (with certificates) to be reenrolled and given new ones. If set to **NO**, the server will return an error if a reenrollment is attempted.

This policy does not require special configuration. The enrollment will proceed with the standard enrollment profile, which likely enrolled the token originally.

RENEW=NO;RENEW_KEEP_OLD_ENC_CERTS=YES

Renewal allows a token to have their profile generated certificates to be renewed in place on the token. If **RENEW** is set to **YES**, a simple enrollment from the Enterprise Security Client (ESC) will result in a renewal instead of a reenrollment as discussed above.

The **RENEW_KEEP_OLD_ENC_CERTS** setting determines if a renewal operation will retain the previous version of the encryption certificate. Retaining the previous certificate allows users to access data encrypted with the old certificate. Setting this option to **NO** will mean that anything encrypted with the old certificate will no longer be recoverable.

Configuration:

```
op.enroll.userKey.renewal.encryption.ca.conn=ca1
op.enroll.userKey.renewal.encryption.ca.profileId=caTokenUserEncryptionKeyRenewal
op.enroll.userKey.renewal.encryption.certAttrId=c2
op.enroll.userKey.renewal.encryption.certId=C2
op.enroll.userKey.renewal.encryption.enable=true
op.enroll.userKey.renewal.encryption.gracePeriod.after=30
op.enroll.userKey.renewal.encryption.gracePeriod.before=30
op.enroll.userKey.renewal.encryption.gracePeriod.enable=false
op.enroll.userKey.renewal.keyType.num=2
op.enroll.userKey.renewal.keyType.value.0=signing
op.enroll.userKey.renewal.keyType.value.1=encryption
op.enroll.userKey.renewal.signing.ca.conn=ca1
op.enroll.userKey.renewal.signing.ca.profileId=caTokenUserSigningKeyRenewal
op.enroll.userKey.renewal.signing.certAttrId=c1
op.enroll.userKey.renewal.signing.certId=C1
op.enroll.userKey.renewal.signing.enable=true
op.enroll.userKey.renewal.signing.gracePeriod.after=30
op.enroll.userKey.renewal.signing.gracePeriod.before=30
op.enroll.userKey.renewal.signing.gracePeriod.enable=false
```

This type of renewal configuration mirrors the basic **userKey** standard enrollment profile with a few added settings that are renewal specific. This parity is needed because we went to renew exactly the number and type of certs that were enrolled originally on to the token before renewal is to be put into play.

FORCE_FORMAT=NO

This policy causes every enrollment operation to prompt a format operation if enabled. This is a last-step option to allow tokens to be reset without a user having to return it to an administrator. If set to **YES**, every enrollment operation initiated by the user will cause a format to happen, esentially resetting the token to the formatted state.

No additional configuration is necessary. A simple format occurs given the same TPS profile used to perform a standard format operation.

PIN_RESET=NO

This policy determines if an already enrolled token can perform an explicit “pin reset” change using the ESC. This value must be set to **YES** or the attempted operation will be rejected with an error by the server.

Configuration:

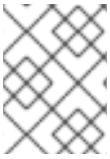
```
op.enroll.userKey.pinReset.enable=true
op.enroll.userKey.pinReset.pin.maxLen=10
op.enroll.userKey.pinReset.pin.maxRetries=127
op.enroll.userKey.pinReset.pin.minLen=4
```

In the above example, the settings for **minLen** and **maxLen** put constraints on the length of a chosen password, and the **maxRetries** setting sets the token to only allow a given number of retries before locking up.

TPS policies can be edited easily using the latest TPS user interface. Navigate to the token that needs a policy change and click **Edit**. This will bring up a dialog that will allow you to edit the field, which is a collection of semi colon separated policies strung together. Each supported policy must be set to ***<POLICYNAME>*=YES** or ***<POLICYNAME>*=NO** in order to be recognized by TPS.

6.4. TOKEN OPERATION AND POLICY PROCESSING

This section discusses major operations (both explicit and implicit) that involve a token. The list below will discuss each feature and its configuration.



NOTE

See the *Token Policies*section in the *Red Hat Certificate System Planning, Installation and Deployment Guide* for general information.

Format

The Format operation (user-initiated) takes a token in a completely blank state as supplied by the manufacturer, and loads a Coolkey applet on it.

Configuration example:

```
#specify that we want authentication for format. We almost always want this at true:
op.format.userKey.auth.enable=true
#specify the ldap authentication configuration, so TPS knows where to validate credentials:
op.format.userKey.auth.id=ldap1
#specify the connection the the CA
op.format.userKey.ca.conn=ca1
#specify id of the card manager applet on given token
op.format.userKey.cardmgr_instance=A0000000030000

#specify if we need to match the visa cuid to the nist sp800sp derivation algorithm KDD value.
Mostly will be false:
op.format.userKey.cuidMustMatchKDD=false

#enable ability to restrict key changoever to a specific range of key set:
op.format.userKey.enableBoundedGPKeyVersion=true
#enable the phone home url to write to the token:
op.format.userKey.issuerinfo.enable=true
```

```
#actual home url to write to token:
op.format.userKey.issuerinfo.value=http://server.example.com:8080/tps/phoneHome
#specify whether to request a login from the client. Mostly true, external reg may want this to be
false:
op.format.userKey.loginRequest.enable=true
#Actual range of desired keyset numbers:
op.format.userKey.maximumGPKeyVersion=FF
op.format.userKey.minimumGPKeyVersion=01
#Whether or not to revoke certs on the token after a format, and what the reason will be if so:
op.format.userKey.revokeCert=true
op.format.userKey.revokeCert.reason=0
#This will roll back the reflected keyyset version of the token in the tokendb. After a failed key
changeover operation. This is to keep the value in sync with reality in the tokendb. Always false,
since this version of TPS avoids this situation now:
op.format.userKey.rollbackKeyVersionOnPutKeyFailure=false

#specify connection to the TKS:
op.format.userKey.tks.conn=tks1
#where to get the actual applet file to write to the token:
op.format.userKey.update.applet.directory=/usr/share/pki/tps/applets
#Allows a completely blank token to be recognized by TPS. Mostly should be true:
op.format.userKey.update.applet.emptyToken.enable=true
#Always should be true, not supported:
op.format.userKey.update.applet.encryption=true
#Actual version of the applet file we want to upgrade to. This file will have a name something like:
1.4.54de7a99.ijc:
op.format.userKey.update.applet.requiredVersion=1.4.54de790f
#Symm key changeover:
op.format.userKey.update.symmetricKeys.enable=false
op.format.userKey.update.symmetricKeys.requiredVersion=1
#Make sure the token db is in sync with reality. Should always be true:
op.format.userKey.validateCardKeyInfoAgainstTokenDB=true
```

Enrollment

The basic enrollment operation takes a formatted token and places certs and keys onto the token in an effort to personalize the token. The following configuration example will explain how this can be controlled.

The example shows basic enrollment which does not deal with renewal and internal recovery. Settings not discussed here are either covered in the Format section, or not crucial.

```
op.enroll.userKey.auth.enable=true
op.enroll.userKey.auth.id=ldap1
op.enroll.userKey.cardmgr_instance=A0000000030000
op.enroll.userKey.cuidMustMatchKDD=false

op.enroll.userKey.enableBoundedGPKeyVersion=true
op.enroll.userKey.issuerinfo.enable=true
op.enroll.userKey.issuerinfo.value=http://server.example.com:8080/tps/phoneHome

#configure the encryption cert and keys  we want on the token:

#connection the the CA, which issues the certs:
op.enroll.userKey.keyGen.encryption.ca.conn=ca1
#Profile id we want the CA to use to issue our encrytion cert:
```

op.enroll.userKey.keyGen.encryption.ca.profileId=caTokenUserEncryptionKeyEnrollment

#These two cover the indexes of the certs written to the token. Each cert needs a unique index or “slot”. In our sample the enc cert will occupy slot 2 and the signing cert, shown later, will occupy slot 1. Avoid overlap with these numbers:
op.enroll.userKey.keyGen.encryption.certAttrId=c2
op.enroll.userKey.keyGen.encryption.certId=C2

op.enroll.userKey.keyGen.encryption.cuid_label=$cuid$
#specify size of generated private key:
op.enroll.userKey.keyGen.encryption.keySize=1024
op.enroll.userKey.keyGen.encryption.keyUsage=0
op.enroll.userKey.keyGen.encryption.keyUser=0
#specify pattern for what the label of the cert will look like when the cert nickname is displayed in browsers and mail clients:
op.enroll.userKey.keyGen.encryption.label=encryption key for $userid$
#specify if we want to overwrite certs on a re-enrollment operation. This is almost always the case:
op.enroll.userKey.keyGen.encryption.overwrite=true

#The next several settings specify the capabilities that the private key on the final token will inherit. For instance this will determine if the cert can be used for encryption or digital signatures. There are settings for both the private and public key.

op.enroll.userKey.keyGen.encryption.private.keyCapabilities.decrypt=true
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.derive=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.encrypt=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.private=true
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.sensitive=true
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.sign=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.signRecover=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.token=true
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.unwrap=true
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.verify=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.verifyRecover=false
op.enroll.userKey.keyGen.encryption.private.keyCapabilities.wrap=false
op.enroll.userKey.keyGen.encryption.privateKeyAttrId=k4
op.enroll.userKey.keyGen.encryption.privateKeyNumber=4
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.decrypt=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.derive=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.encrypt=true
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.private=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.sensitive=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.sign=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.signRecover=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.token=true
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.unwrap=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.verify=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.verifyRecover=false
op.enroll.userKey.keyGen.encryption.public.keyCapabilities.wrap=true

#The following index numbers correspond to the index or slot that the private and public keys occupy. The common formula we use is that the public key index will be 2 * cert id + 1, and the private key index, shown above will be 2 * cert id. In this example the cert id is 2, so the key ids will be 4 and 5 respectively. When composing these, be careful not to create conflicts. This applies to the signing key section below.

```
op.enroll.userKey.keyGen.encryption.publicKeyAttrId=k5
op.enroll.userKey.keyGen.encryption.publicKeyNumber=5

#specify if, when a certificate is slated for revocation, based on other rules, we want to check to
see if some other token is using this cert in a shared situation. If this is set to true, and this
situation is found the cert will not be revoked until the last token wants to revoke this cert:
op.enroll.userKey.keyGen.encryption.recovery.destroyed.holdRevocationUntilLastCredential=false


#specify, if we want server side keygen, if we want to have that generated key archived to the
drm. This is almost always the case, since we want the ability to later recover a cert and its
encryption private key back to a new token:
op.enroll.userKey.keyGen.encryption.serverKeygen.archive=true
#connection to drm to generate the key for us:
op.enroll.userKey.keyGen.encryption.serverKeygen.drm.conn=kra1
#specify server side keygen of the encryption private key. This most often will be desired:
op.enroll.userKey.keyGen.encryption.serverKeygen.enable=true

#This setting tells us how many certs we want to enroll for this TPS profile, in the case “userKey”.
Here we want 2 total certs. The next values then go on to index into the config what two types of
certs we want, signing and encryption:
op.enroll.userKey.keyGen.keyType.num=2
op.enroll.userKey.keyGen.keyType.value.0=signing
op.enroll.userKey.keyGen.keyType.value.1=encryption

#configure the signing cert and keys we want on the token the settings for these are similar to the
encryption settings already discussed, except the capability flags presented below, since this is a
signing key.

op.enroll.userKey.keyGen.signing.ca.conn=ca1
op.enroll.userKey.keyGen.signing.ca.profileId=caTokenUserSigningKeyEnrollment
op.enroll.userKey.keyGen.signing.certAttrId=c1
op.enroll.userKey.keyGen.signing.certId=C1
op.enroll.userKey.keyGen.signing.cuid_label=$cuid$
op.enroll.userKey.keyGen.signing.keySize=1024
op.enroll.userKey.keyGen.signing.keyUsage=0
op.enroll.userKey.keyGen.signing.keyUser=0
op.enroll.userKey.keyGen.signing.label=signing key for $userid$
op.enroll.userKey.keyGen.signing.overwrite=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.decrypt=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.derive=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.encrypt=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.private=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.sensitive=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.sign=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.signRecover=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.token=true
op.enroll.userKey.keyGen.signing.private.keyCapabilities.unwrap=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.verify=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.verifyRecover=false
op.enroll.userKey.keyGen.signing.private.keyCapabilities.wrap=false
op.enroll.userKey.keyGen.signing.privateKeyAttrId=k2
op.enroll.userKey.keyGen.signing.privateKeyNumber=2
op.enroll.userKey.keyGen.signing.public.keyCapabilities.decrypt=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.derive=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.encrypt=false
```

```
op.enroll.userKey.keyGen.signing.public.keyCapabilities.private=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.sensitive=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.sign=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.signRecover=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.token=true
op.enroll.userKey.keyGen.signing.public.keyCapabilities.unwrap=false
op.enroll.userKey.keyGen.signing.public.keyCapabilities.verify=true
op.enroll.userKey.keyGen.signing.public.keyCapabilities.verifyRecover=true
op.enroll.userKey.keyGen.signing.public.keyCapabilities.wrap=false
op.enroll.userKey.keyGen.signing.publicKeyAttrId=k3
op.enroll.userKey.keyGen.signing.publicKeyNumber=3
```

Pin Reset

The configuration for pin reset is discussed in Section 6.3, "Token Policies", because pin reset relies on a policy to determine if it is to be legally performed or not.

Renewal

The configuration for renewal is discussed in Section 6.3, "Token Policies", since renewal relies on a policy to determine if it is legal to perform or not upon an already enrolled token.

Recovery

Recovery is implicitly set into motion when the user of the TPS user interface marks a previously active token into an unfavorable state such as "lost" or "destroyed". Once this happens, the next enrollment of a new token by the same user will adhere to the following configuration to recover the certificates from the user's old token, to this new token.

The end result of this operation is that the user will have a new physical token that may contain the encryption certificates recovered from the old token, so that the user can continue to encrypt and decrypt data as needed. A new signing certificate is also usually placed on this token as shown in the sample config examples below.

The following is a list of supported states into which a token can be placed manually in the TPS user interface, as seen in the configuration:

- **tokendb._069=#** - **DAMAGED (1)**: Corresponds to **destroyed** in the recovery configuration. Used when a token has been physically damaged.
- **tokendb._070=#** - **PERM_LOST (2)**: Corresponds to **keyCompromise** in the recovery configuration. Used when a token has been lost permanently.
- **tokendb._071=#** - **SUSPENDED (3)**: Corresponds to **onHold** in the recovery configuration. Used when a token has been temporarily misplaced, but the user expects to find it again.
- **tokendb._072=#** - **TERMINATED (6)**: Corresponds to **terminated** in the recovery configuration. Used to take a token out of service forever for internal reasons.

Example recovery configuration:

```
#When a token is marked destroyed, don't revoke the certs on the token unless all other tokens do
not have the certs included:
op.enroll.userKey.keyGen.encryption.recovery.destroyed.holdRevocationUntilLastCredential=false

#specify if we even want to revoke certs a token is marked destroyed:
op.enroll.userKey.keyGen.encryption.recovery.destroyed.revokeCert=false
```

```
#if we want to revoke any certs here, specify the reason for revocation that will be sent to the CA:
op.enroll.userKey.keyGen.encryption.recovery.destroyed.revokeCert.reason=0
#speficy if we want to revoke expired certs when marking the token destroyed:
op.enroll.userKey.keyGen.encryption.recovery.destroyed.revokeExpiredCerts=false
```

Additional settings are used to specify what kind of supported static recovery should be used when performing a recovery operation to a new token (when the original token has been marked destroyed). The following schemes are supported:

- Recover Last (**RecoverLast**): Recover the latest encryption certificate to be placed on the token.
- Generate New Key and Recover Last (**GenerateNewKeyAndRecoverLast**): Same as Recover Last, but also generate a new encryption certificate and upload it to the token as well. The new token will then have two certificates.
- Generate New Key (**GenerateNewKey**): Generate a new encryption certificate and place it on the token. Do not recover any old certificates.

For example:

```
op.enroll.userKey.keyGen.encryption.recovery.destroyed.scheme=RecoverLast
```

The following configuration example determines how to recover tokens marked as permanently lost:

```
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.holdRevocationUntilLastCredential=f
alse
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.revokeCert=true
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.revokeCert.reason=1
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.revokeExpiredCerts=false
op.enroll.userKey.keyGen.encryption.recovery.keyCompromise.scheme=GenerateNewKey

# Section when a token is marked terminated.

op.enroll.userKey.keyGen.encryption.recovery.terminated.holdRevocationUntilLastCredential=false

op.enroll.userKey.keyGen.encryption.recovery.terminated.revokeCert=true
op.enroll.userKey.keyGen.encryption.recovery.terminated.revokeCert.reason=1
op.enroll.userKey.keyGen.encryption.recovery.terminated.revokeExpiredCerts=false
op.enroll.userKey.keyGen.encryption.recovery.terminated.scheme=GenerateNewKey

# This section details the recovery profile with respect to which certs and of what kind get
recovered on the token.

op.enroll.userKey.keyGen.recovery.destroyed.keyType.num=2
op.enroll.userKey.keyGen.recovery.destroyed.keyType.value.0=signing
op.enroll.userKey.keyGen.recovery.destroyed.keyType.value.1=encryption
```

Finally, the following example determines what the system will do about the signing certificate that was on the old token. In most cases, the **GenerateNewKey** recovery scheme should be used in order to avoid potentially having multiple copies of a signing private key available (for example, one that is recovered on a new token, and one on an old token that was permanently lost but found by somebody else).

```
op.enroll.userKey.keyGen.recovery.keyCompromise.keyType.value.0=signing
```

```
op.enroll.userKey.keyGen.recovery.keyCompromise.keyType.value.1=encryption
op.enroll.userKey.keyGen.recovery.onHold.keyType.num=2
op.enroll.userKey.keyGen.recovery.onHold.keyType.value.0=signing
op.enroll.userKey.keyGen.recovery.onHold.keyType.value.1=encryption

op.enroll.userKey.keyGen.signing.recovery.destroyed.holdRevocationUntilLastCredential=false
op.enroll.userKey.keyGen.signing.recovery.destroyed.revokeCert=true
op.enroll.userKey.keyGen.signing.recovery.destroyed.revokeCert.reason=0
op.enroll.userKey.keyGen.signing.recovery.destroyed.revokeExpiredCerts=false
op.enroll.userKey.keyGen.signing.recovery.destroyed.scheme=GenerateNewKey
op.enroll.userKey.keyGen.signing.recovery.keyCompromise.holdRevocationUntilLastCredential=fals

op.enroll.userKey.keyGen.signing.recovery.keyCompromise.revokeCert=true
op.enroll.userKey.keyGen.signing.recovery.keyCompromise.revokeCert.reason=1
op.enroll.userKey.keyGen.signing.recovery.keyCompromise.revokeExpiredCerts=false
op.enroll.userKey.keyGen.signing.recovery.keyCompromise.scheme=GenerateNewKey
op.enroll.userKey.keyGen.signing.recovery.onHold.holdRevocationUntilLastCredential=false
op.enroll.userKey.keyGen.signing.recovery.onHold.revokeCert=true

op.enroll.userKey.keyGen.signing.recovery.onHold.revokeCert.reason=6
op.enroll.userKey.keyGen.signing.recovery.onHold.revokeExpiredCerts=false
op.enroll.userKey.keyGen.signing.recovery.onHold.scheme=GenerateNewKey
op.enroll.userKey.keyGen.signing.recovery.terminated.holdRevocationUntilLastCredential=false
op.enroll.userKey.keyGen.signing.recovery.terminated.revokeCert=true
op.enroll.userKey.keyGen.signing.recovery.terminated.revokeCert.reason=1
op.enroll.userKey.keyGen.signing.recovery.terminated.revokeExpiredCerts=false
op.enroll.userKey.keyGen.signing.recovery.terminated.scheme=GenerateNewKey

# Configuration for the case when we mark a token “onHold” or temporarily lost

op.enroll.userKeyTemporary.keyGen.encryption.recovery.onHold.revokeCert=true
op.enroll.userKeyTemporary.keyGen.encryption.recovery.onHold.revokeCert.reason=0
op.enroll.userKeyTemporary.keyGen.encryption.recovery.onHold.scheme=RecoverLast
op.enroll.userKeyTemporary.keyGen.recovery.onHold.keyType.num=2
op.enroll.userKeyTemporary.keyGen.recovery.onHold.keyType.value.0=signing
op.enroll.userKeyTemporary.keyGen.recovery.onHold.keyType.value.1=encryption
op.enroll.userKeyTemporary.keyGen.signing.recovery.onHold.revokeCert=true
op.enroll.userKeyTemporary.keyGen.signing.recovery.onHold.revokeCert.reason=0
op.enroll.userKeyTemporary.keyGen.signing.recovery.onHold.scheme=GenerateNewKey
```

Applet Update

The following example shows how to configure a Coolkey applet update operation. This operation can be performed during format, enrollment, and PIN reset operations:

```
op.format.userKey.update.applet.directory=/usr/share/pki/tps/applets
op.format.userKey.update.applet.emptyToken.enable=true
op.format.userKey.update.applet.encryption=true
op.format.userKey.update.applet.requiredVersion=1.4.54de790f
```

Some of these options have already been demonstrated in the Format section. They provide information needed to determine if applet upgrade should be allowed, where to find the applet files, and the applet version to upgrade the token to. The version in the **requiredVersion** maps to a file name inside the **directory**.

Key Update

This operation, which can take place during format, enrollment, and PIN reset operations, allows the user to have their Global Platform key set version upgraded from the default supplied by the manufacturer.

TPS

The following options will instruct the TPS to upgrade the keyset from 1 to 2 during the next format operation requested on behalf of a given token. After this is done, the TKS must derive the three new keys that will be written to the token, Afterwards, the token must be used with the same TPS and TKS installation, otherwise it will become locked.

```
op.format.userKey.update.symmetricKeys.enable=true
op.format.userKey.update.symmetricKeys.requiredVersion=2
```

You can also specify a version lower than current to downgrade the keyset instead.

TKS

As mentioned above, the TKS must be configured to generate the new keys to write to the token. First, the new master key identifier, **02**, must be mapped to its PKCS #11 object nickname in the TKS **CS.cfg**, as shown in the following example:

```
tks.mk_mappings.#02#01=internal:new_master
tks.defKeySet.mk_mappings.#02#01=internal:new_master
```

The above will map a key set number to an actual master key which exists in the TKS NSS database.

Master keys are identified by IDs such as **01**. The TKS maps these IDs to PKCS #11 object nicknames specified in the **masterKeyId** part of the mapping. Therefore, the first number is updated as the master key version is updated, and the second number stays consistent.

When attempting to upgrade from version 1 to version 2, the mapping determines how to find the master key nickname which will be used to derive the 3 parts of the new key set.

The setting of **internal** in the above example references the name of the token where the master key resides. It could also be an external HSM module with a name such as **nethsm**. The strong **new_master** is an example of the master key nickname itself.

6.5. INTERNAL REGISTRATION

NOTE

See the *TPS Profiles* section of the *Red Hat Certificate System Planning, Installation and Deployment Guide* for general information.

In case of *Internal Registration*, the TPS profile (token type) is determined by the *Mapping Resolver*. In contrast with *External Registration*, authentication information is defined within the profile itself. For example:

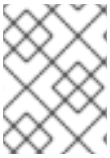
```
op.enroll.userKey.auth.enable=true
op.enroll.userKey.auth.id=ldap1
```

Another difference from External Registration is that the CA and KRA connector information is defined under each key type of each profile. For example:

```
op.enroll.userKey.keyGen.encryption.ca.conn=ca1
op.enroll.userKey.keyGen.encryption.serverKeygen.drm.conn=kra1
```

TKS connector information, however, is defined per profile:

```
op.enroll.userKey.tks.conn=tks1
```



NOTE

Switching registration types between Internal and External Registration means you have to format all previously registered tokens before you can continue using them.

6.6. EXTERNAL REGISTRATION

External Registration obtains the token type (TPS profile) from the authenticated user LDAP record. It also allows certificate/key recovery information to be specified in the same user record.

An External Registration TPS profile is similar to the Internal Registration profile discussed previously. It allows you to specify new certificate enrollments for both client-side and server-side key generation. Unlike Internal Registration, it allows you to choose specific certificate (and its matching keys) to be retrieved and loaded onto the token.

NOTE

Switching registration types between Internal and External Registration means you have to format all previously registered tokens before you can continue using them.

6.6.1. Enabling External Registration

External Registration can only be enabled globally for an entire TPS instance. The following example shows a set of global configuration parameters pertaining to External Registration:

```
externalReg.allowRecoverInvalidCert.enable=true
externalReg.authId=ldap1
externalReg.default.tokenType=externalRegAddToToken
externalReg.delegation.enable=true
externalReg.enable=true
externalReg.recover.byKeyID=false
externalReg.format.loginRequest.enable=true
externalReg.mappingResolver=keySetMappingResolver
```

6.6.2. Customizing User LDAP Record Attribute Names

Authentication parameters pertaining to External Registration are shown in the following example (with their default values):

```
auths.instance.ldap1.externalReg.certs.recoverAttributeName=certsToAdd
auths.instance.ldap1.externalReg.cuidAttributeName=tokenCUID
auths.instance.ldap1.externalReg.tokenTypeAttributeName=tokenType
```

The LDAP record attribute names can be customized here. Make sure that the actual attributes in the user's LDAP records match this configuration.

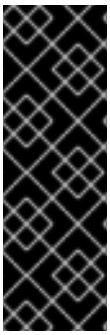
6.6.3. Configuring certsToAdd attributes

The **certsToAdd** attribute takes multiple values in the following form:

```
<cert serial # in decimal>,<CA connector ID>,<key ID>,<kra connector ID>
```

For example:

```
59,ca1,0,kra1
```



IMPORTANT

By default, key recovery searches for the key by certificate, which makes the *<key ID>* value irrelevant. However, the TPS can optionally be configured to search for the key using this attribute, and therefore it is typically simpler to set the value to 0. That value is invalid, which avoids the possibility of retrieving an unmatched key.

Recovering by key ID is not recommended, because the KRA can not verify if the certificate matches with the key in this situation.

When specifying the **certsToAdd** attribute with only certificate and CA information, the TPS assumes that the certificate in question is already on the token, and that it should be preserved. This concept is called *Key Retention*.

The following examples show relevant attributes in the user LDAP record:

```
tokenType: externalRegAddToToken
certstoadd: 59,ca1,0,kra1
certstoadd: 134,ca1,0,kra1
Certstoadd: 24,ca1
```

6.6.4. Token to User Matching Enforcement

Optionally, you can set the system up so that the token used for registration must match the token record card-unique ID (CUID) attribute in the user record. If this attribute (**tokencuid**) is missing from the record, CUID matching is not enforced.

```
Tokencuid: a10192030405028001c0
```

Another attribute about External Registration is that the Token Policies on each token are bypassed.

NOTE

For the certificate and keys to be “recovered” in External Registration, connector information for CA and KRA is specified in the user LDAP record. Any CA and/or KRA connector information specified in the TPS profile pertaining to the certificate/keys to be “recovered” is to be ignored.

```
certstoadd: 59,ca1,0,kra1
```

6.6.5. Delegation Support

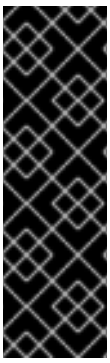
Delegation support is useful where a user has delegates who can act on their behalf (for example, an executive at a company has one or more delegates) in terms of authentication (logins), data encryption and decryption, or signing (with limitations).

An example scenario could be that each delegate has their own token which they use to act on behalf of the executive. This token contains a combination of the following certificates and keys (determined by TPS profiles):

- Authentication certificate/keys: The CN contains the name and unique ID of the delegate. The Subject Alternative Name (SAN) extension contains the Principal Name (UPN) of the executive.
- Encryption certificate: An exact copy of the executive's encryption certificate.
- Signing certificate: The CN contains the delegate's name and unique ID. The SAN contains the RFC822Name of the executive.

Use the following parameter to enable delegation support:

```
externalReg.delegation.enable=true
```



IMPORTANT

To work around a bug, manually set the ***op.enroll.delegateISEtoken.keyGen.encryption.ca.profileId*** parameter in the **/var/lib/pki/*instance_name*/tps/conf/CS.cfg** file to **caTokenUserDelegateAuthKeyEnrollment**:

```
op.enroll.delegateISEtoken.keyGen.encryption.ca.profileId=caTokenUserDelegateAuthKeyEnrollment
```

6.6.6. SAN and DN Patterns

The **auths.instance.*<authID>*.ldapStringAttributes** in the authentication instance configuration specifies which attributes will be retrieved during authentication. For example:

```
auths.instance.ldap1.ldapStringAttributes=mail,cn,uid,edipi,pcc,firstname,lastname,exec-edipi,exec-pcc,exec-mail,certsToAdd,tokenCUID,tokenType
```

Once retrieved from the user's LDAP record, the values of these attributes can be referenced and used to form the Subject Alternative Name (SAN) or Distinguished Name (DN) of the certificate in the format of **$auth.*<attribute name>*$**. For example:

```
op.enroll.delegateIEtoken.keyGen.authentication.SANpattern=$auth.exec-edipi$.$auth.exec-
pcc$@EXAMPLE.com
op.enroll.delegateIEtoken.keyGen.authentication.dnpattern=cn=$auth.firstname$.$auth.lastname$.$aut
h.edipi$,e=$auth.mail$,o=TMS Org
```

When patterns are used in TPS profiles for SAN and DN, it is important to ensure the CA enrollment profile specified in the TPS profile is set up correctly. For example:

On TPS, in profile delegateIEtoken

```
op.enroll.delegateIEtoken.keyGen.authentication.ca.profileId=caTokenUserDelegateAuthKeyEnrollm
ent
```

On CA, in enrollment profile caTokenUserDelegateAuthKeyEnrollment

The **subjectDNInputImpl** plug-in must be specified as one of the inputs in order to allow the DN to be specified by the TPS profile above:

```
input.i2.class_id=subjectDNInputImpl
input.i2.name=subjectDNInputImpl
```

Similarly, to allow the SAN to be specified by the above TPS profile, the **subjectAltNameExtInputImpl** plug-in must be specified:

```
input.i3.class_id=subjectAltNameExtInputImpl
input.i3.name=subjectAltNameExtInputImpl
```

The **subjAltExtpattern** must be specified as well:

```
policyset.set1.p6.default.params.subjAltExtPattern_0=
(UTF8String)1.3.6.1.4.1.311.20.2.3,$request.req_san_pattern_0$
```

In the above example, the OID **1.3.6.1.4.1.311.20.2.3** is the OID for the User Principal Name (UPN), and **request.req_san_pattern_0** is the first SAN pattern specified in the **delegateIEtoken** SAN pattern.

You can specify multiple SANs at the same time. On the TPS side, specify multiple SANs in the **SANpattern**, delimited by a comma ("**,**"). On the CA side, a corresponding amount of **subjAltExtPattern** needs to be defined in the following format:

```
policyset.<policy set id>.<policy id>.default.params.subjAltExtPattern_<ordered number>=
```

In the above, the *<ordered number>* starts with 0 and increases by one for each SAN pattern specified on the TPS side:

```
policyset.set1.p6.default.params.subjAltExtPattern_0=
policyset.set1.p6.default.params.subjAltExtPattern_1=
...
```

The following is a complete example:

Example 6.1. SANpattern and DNpattern configuration

The LDAP record contains the following information:

```
givenName: user1a
mail: user1a@example.org
firstname: user1a
edipi: 123456789
pcc: AA
exec-edipi: 999999999
exec-pcc: BB
exec-mail: user1b@EXAMPLE.com
tokenType: delegateISEtoken
certstoadd: 59,ca1,0,kra1
```

TPS External Registration profile **delegateIEtoken** contains:

- **SANpattern**:

```
op.enroll.delegateISEtoken.keyGen.authentication.SANpattern=$auth.exec-
edipi$.$auth.exec-pcc$@EXAMPLE.com
```

- **DNPattern**:

```
op.enroll.delegateISEtoken.keyGen.authentication.dnpattern=cn=$auth.firstname$.$auth.las
tname$.$auth.edipi$,e=$auth.mail$,o=TMS Org
```

CA **caTokenUserDelegateAuthKeyEnrollment** contains:

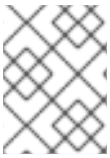
```
input.i2.class_id=subjectDNInputImpl
input.i2.name=subjectDNInputImpl
input.i3.class_id=subjectAltNameExtInputImpl
input.i3.name=subjectAltNameExtInputImpl

policyset.set1.p6.constraint.class_id=noConstraintImpl
policyset.set1.p6.constraint.name=No Constraint
policyset.set1.p6.default.class_id=subjectAltNameExtDefaultImpl
policyset.set1.p6.default.name=Subject Alternative Name Extension Default
policyset.set1.p6.default.params.subjAltExtGNEnable_0=true
policyset.set1.p6.default.params.subjAltExtPattern_0=
(UTF8String)1.3.6.1.4.1.311.20.2.3,$request.req_san_pattern_0$
policyset.set1.p6.default.params.subjAltExtType_0=OtherName
policyset.set1.p6.default.params.subjAltNameExtCritical=false
policyset.set1.p6.default.params.subjAltNameNumGNs=1
```

The resulting certificate then contains:

```
Subject: CN=user1a..123456789,E=user1a@example.org,O=TMS Org
Identifier: Subject Alternative Name - 2.5.29.17
Critical: no
Value:
  OtherName: (UTF8String)1.3.6.1.4.1.311.20.2.3,999999999.BB@EXAMPLE.com
```

6.7. MAPPING RESOLVER CONFIGURATION

The Token Processing System provides a single mapping resolver by default. The resolver is called **FilterMappingResolver**. This section will cover its configuration.



NOTE

See the *Mapping Resolver* section of the *Red Hat Certificate System Planning, Installation, and Deployment Guide* for general information about the Mapping Resolver.

6.7.1. Key Set Mapping Resolver

During External Registration, the key set must be resolved using the resolver before a user can authenticate.

The key set mapping resolver name is defined as follows:

```
externalReg.mappingResolver=<keySet mapping resolver name>
```

For example:

```
externalReg.mappingResolver=keySetMappingResolver
```

The following configuration example shows a full instance configuration:

```
mappingResolver.keySetMappingResolver.class_id=filterMappingResolverImpl
mappingResolver.keySetMappingResolver.mapping.0.filter.appletMajorVersion=0
mappingResolver.keySetMappingResolver.mapping.0.filter.appletMinorVersion=0
mappingResolver.keySetMappingResolver.mapping.0.filter.keySet=
mappingResolver.keySetMappingResolver.mapping.0.filter.tokenATR=
mappingResolver.keySetMappingResolver.mapping.0.filter.tokenCUID.end=a1000000000000000000
mappingResolver.keySetMappingResolver.mapping.0.filter.tokenCUID.start=a0000000000000000000

mappingResolver.keySetMappingResolver.mapping.0.target.keySet=defKeySet
mappingResolver.keySetMappingResolver.mapping.1.filter.appletMajorVersion=1
mappingResolver.keySetMappingResolver.mapping.1.filter.appletMinorVersion=1
mappingResolver.keySetMappingResolver.mapping.1.filter.keySet=
mappingResolver.keySetMappingResolver.mapping.1.filter.tokenATR=1234
mappingResolver.keySetMappingResolver.mapping.1.filter.tokenCUID.end=
mappingResolver.keySetMappingResolver.mapping.1.filter.tokenCUID.start=
mappingResolver.keySetMappingResolver.mapping.1.target.keySet=defKeySet
mappingResolver.keySetMappingResolver.mapping.2.filter.appletMajorVersion=
mappingResolver.keySetMappingResolver.mapping.2.filter.appletMinorVersion=
mappingResolver.keySetMappingResolver.mapping.2.filter.keySet=
mappingResolver.keySetMappingResolver.mapping.2.filter.tokenATR=
mappingResolver.keySetMappingResolver.mapping.2.filter.tokenCUID.end=
mappingResolver.keySetMappingResolver.mapping.2.filter.tokenCUID.start=
mappingResolver.keySetMappingResolver.mapping.2.target.keySet=jForte
mappingResolver.keySetMappingResolver.mapping.order=0,1,2
```

The above example defines three mappings named **0**, **1**, and **2**. They are ordered in ascending order using the **mappingResolver.keySetMappingResolver.mapping.order=0,1,2** line in the example. This order means the input parameters will be run against the mapping filter **0** first; only if they do not match

that filter, the next one in the mapping order will be tried. For example, if a token with the following characteristics is evaluated:

```
CUID=a0000000000000000011
appletMajorVersion=0
appletMinorVersion=0
```

Then it would pass mapping **0** and be assigned its target, which is configured to **defKeySet**, because the applet version matches and the CUID falls within the CUID start and end range for that mapping.

On the other hand, if a token has the following parameters:

```
CUID=b0000000000000000000
ATR=2222
appletMajorVersion=1
appletMinorVersion=1
```

In this case this token fails mapping **0** because it is outside the specified CUID range. It also fails mapping **1** because while the applet versions match, the ATR does not. The above token will be assigned to mapping **2** and its target, **jForte**.

Note how mapping **2** has no assignments for any of its filters. This causes the mapping to match all tokens, effectively making it a "default" value. Mappings like this must be specified last in the mapping order, because any other mappings after it will never be evaluated.

6.7.2. Token Type (TPS) Mapping Resolver

There are three default **tokenType** mapping resolvers defined in the Token Processing System: **formatProfileMappingResolver**, **enrollProfileMappingResolver**, and **pinResetProfileMappingResolver**. Compared to the External Registration case discussed in the previous section, in the Internal Registration case token types are actually calculated from the defined mapping resolver.

The token type mapping resolver names are defined as follows:

```
op.<op>.mappingResolver=<mapping resolver name>
```

For example:

```
op.enroll.mappingResolver=enrollProfileMappingResolver
```

The following configuration example describes the **enrollProfileMappingResolver**:

```
mappingResolver.enrollProfileMappingResolver.class_id=filterMappingResolverImpl
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.appletMajorVersion=1
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.appletMinorVersion=
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.tokenATR=
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.tokenCUID.end=b10000000000000000000
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.tokenCUID.start=b00000000000000000000
mappingResolver.enrollProfileMappingResolver.mapping.0.filter.tokenType=userKey
mappingResolver.enrollProfileMappingResolver.mapping.0.target.tokenType=userKey
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.appletMajorVersion=1
```

```
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.appletMinorVersion=
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.tokenATR=
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.tokenCUID.end=a0000000000000001000
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.tokenCUID.start=a0000000000000000000
mappingResolver.enrollProfileMappingResolver.mapping.1.filter.tokenType=soKey
mappingResolver.enrollProfileMappingResolver.mapping.1.target.tokenType=soKey
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.appletMajorVersion=
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.appletMinorVersion=
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.tokenATR=
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.tokenCUID.end=
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.tokenCUID.start=
mappingResolver.enrollProfileMappingResolver.mapping.2.filter.tokenType=
mappingResolver.enrollProfileMappingResolver.mapping.2.target.tokenType=userKey
mappingResolver.enrollProfileMappingResolver.mapping.order=1,0,2
```

Three mappings are defined for the **enrollProfileMappingResolver** in the above example. The mappings are named **0**, **1**, and **2**. The **mappingResolver.enrollProfileMappingResolver.mapping.order=1,0,2** line defines the order in which the mappings will be processed. If a token matches a mapping, no further mappings in the order will be evaluated; if it does not match a mapping, the next one in the order will be tried.

In case of a token with the following parameters:

```
CUID=a0000000000000000011
appletMajorVersion=1
appletMinorVersion=0
extension: tokenType=soKey
```

A token with this configuration will match the filters for mapping **1** because the applet version matches, the CUID fails within the specified start and end range, and the extension **tokenType** matches. Therefore, this token will be assigned the target for that mapping - **soKey**.

In another case, if the token has the following parameters:

```
CUID=b0000000000000000010
appletMajorVersion=1
appletMinorVersion=1
```

In this case, the token will fail mapping **1** because the CUID is outside the specified range. Then it will also fail mapping **0**, because the **tokenType** extension is missing. This token will then match mapping **2**, because it has no specified filters in order to match all tokens which did not match any of the previous filters.

6.8. AUTHENTICATION CONFIGURATION

The Token Processing System supports directory-based authentication using a user ID and password (**UidPwdDirAuthentication**) by default. Authentication instances are defined in the **CS.cfg** file using the following pattern:

```
auths.instance.<auths ID>.*
```

The *<auths ID>* is the authenticator name to be referenced by the TPS profiles for authentication preferences. For example:

```
op.enroll.userKey.auth.id=ldap1
```

The following configuration example shows a full definition of an authentication instance:

```
auths.impl.UidPwdDirAuth.class=com.netscape.cms.authentication.UidPwdDirAuthentication
auths.instance.ldap1.pluginName=UidPwdDirAuth
auths.instance.ldap1.authCredName=uid
auths.instance.ldap1.dnpattern=
auths.instance.ldap1.externalReg.certs.recoverAttributeName=certsToAdd
auths.instance.ldap1.externalReg.cuidAttributeName=tokenCUID
auths.instance.ldap1.externalReg.tokenTypeAttributeName=tokenType
auths.instance.ldap1.ldap.basedn=dc=sjc,dc=example,dc=com
auths.instance.ldap1.ldap.ldapauth.authtype=BasicAuth
auths.instance.ldap1.ldap.ldapauth.bindDN=
auths.instance.ldap1.ldap.ldapauth.bindPWPrompt=ldap1
auths.instance.ldap1.ldap.ldapauth.clientCertNickname=subsystemCert cert-pki-tomcat
auths.instance.ldap1.ldap.ldapconn.host=host1.EXAMPLE.com
auths.instance.ldap1.ldap.ldapconn.port=389
auths.instance.ldap1.ldap.ldapconn.secureConn=False
auths.instance.ldap1.ldap.ldapconn.version=3
auths.instance.ldap1.ldap.maxConns=15
auths.instance.ldap1.ldap.minConns=3
auths.instance.ldap1.ldapByteAttributes=
auths.instance.ldap1.ldapStringAttributes=mail,cn,uid,edipi,pcc,firstname,lastname,exec-edipi,exec-
pcc,exec-mail,certsToAdd,tokenCUID,tokenType
auths.instance.ldap1.ldapStringAttributes._000=###################################
auths.instance.ldap1.ldapStringAttributes._001=# For isExternalReg
auths.instance.ldap1.ldapStringAttributes._002=#   attributes will be available as
auths.instance.ldap1.ldapStringAttributes._003=#       $<attribute>$
auths.instance.ldap1.ldapStringAttributes._004=#   attributes example:
auths.instance.ldap1.ldapStringAttributes._005=#mail,cn,uid,edipi,pcc,firstname,lastname,exec-
edipi,exec-pcc,exec-mail,certsToAdd,tokenCUID,tokenType
auths.instance.ldap1.ldapStringAttributes._006=###################################
auths.instance.ldap1.pluginName=UidPwdDirAuth
auths.instance.ldap1.ui.description.en=This authenticates user against the LDAP directory.
auths.instance.ldap1.ui.id.PASSWORD.credMap.authCred=pwd
auths.instance.ldap1.ui.id.PASSWORD.credMap.msgCred.extlogin=PASSWORD
auths.instance.ldap1.ui.id.PASSWORD.credMap.msgCred.login=password
auths.instance.ldap1.ui.id.PASSWORD.description.en=LDAP Password
auths.instance.ldap1.ui.id.PASSWORD.name.en=LDAP Password
auths.instance.ldap1.ui.id.UID.credMap.authCred=uid
auths.instance.ldap1.ui.id.UID.credMap.msgCred.extlogin=UID
auths.instance.ldap1.ui.id.UID.credMap.msgCred.login=screen_name
auths.instance.ldap1.ui.id.UID.description.en=LDAP User ID
auths.instance.ldap1.ui.id.UID.name.en=LDAP User ID
auths.instance.ldap1.ui.retries=3
auths.instance.ldap1.ui.title.en=LDAP Authentication
```

TPS authentication instances are configured in a way similar to the CA's **UidPwdDirAuthentication** authentication instance, since both are handled by the same plug-in. However, the TPS requires several extra parameters on top of the CA configuration.

In case of common operations (for both Internal and External registration), profiles that call for this authentication method allow TPS to project how the UID and password will be labeled on the client side. This is controlled by the **auths.instance.ldap1.ui.id.UID.name.en=LDAP User ID** and **auths.instance.ldap1.ui.id.PASSWORD.name.en=LDAP Password** parameters in the above example; this configuration tells clients to display the UID/password pair as "LDAP User ID" and "LDAP Password". Both parameters can be customized.

The **credMap.authCred** entries configure how the internal authentication plug-in accepts information presented to it, and the **credMap.msgCred** entries configure how this information is passed to the TPS. These fields allow you to use customized plug-in implementations, and should be left at their default values unless you are using a custom authentication plug-in.

Parameters related to External Registration are discussed in Section 6.6, “External Registration”.

Similarly to CA authentication configuration, you can define multiple authentication instances for the same authentication implementation. This may be useful when the TPS serves multiple groups of users; you can direct each group to use its own TPS profile, each configured to use its own directory server authentication.

6.9. CONNECTORS

Connectors define how the TPS communicates with other subsystems - namely CA, KRA, and TKS. In general, these parameters are set up during TPS installation. The following is an example of connector configuration:

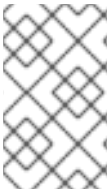
```
tps.connector.ca1.enable=true
tps.connector.ca1.host=host1.EXAMPLE.com
tps.connector.ca1.maxHttpConns=15
tps.connector.ca1.minHttpConns=1
tps.connector.ca1.nickName=subsystemCert cert-pki-tomcat
tps.connector.ca1.port=8443
tps.connector.ca1.timeout=30
tps.connector.ca1.uri.enrollment=/ca/ee/ca/profileSubmitSSLClient
tps.connector.ca1.uri.getcert=/ca/ee/ca/displayBySerial
tps.connector.ca1.uri.renewal=/ca/ee/ca/profileSubmitSSLClient
tps.connector.ca1.uri.revoke=/ca/ee/subsystem/ca/doRevoke
tps.connector.ca1.uri.unrevoke=/ca/ee/subsystem/ca/doUnrevoke
tps.connector.kra1.enable=true
tps.connector.kra1.host=host1.EXAMPLE.com
tps.connector.kra1.maxHttpConns=15
tps.connector.kra1.minHttpConns=1
tps.connector.kra1.nickName=subsystemCert cert-pki-tomcat
tps.connector.kra1.port=8443
tps.connector.kra1.timeout=30
tps.connector.kra1.uri.GenerateKeyPair=/kra/agent/kra/GenerateKeyPair
tps.connector.kra1.uri.TokenKeyRecovery=/kra/agent/kra/TokenKeyRecovery
tps.connector.tks1.enable=true
tps.connector.tks1.generateHostChallenge=true
tps.connector.tks1.host=host1.EXAMPLE.com
tps.connector.tks1.keySet=defKeySet
tps.connector.tks1.maxHttpConns=15
tps.connector.tks1.minHttpConns=1
tps.connector.tks1.nickName=subsystemCert cert-pki-tomcat
tps.connector.tks1.port=8443
tps.connector.tks1.serverKeygen=true
```

```
tps.connector.tks1.timeout=30
tps.connector.tks1.tksSharedSymKeyName=sharedSecret
tps.connector.tks1.uri.computeRandomData=/tks/agent/tks/computeRandomData
tps.connector.tks1.uri.computeSessionKey=/tks/agent/tks/computeSessionKey
tps.connector.tks1.uri.createKeySetData=/tks/agent/tks/createKeySetData
tps.connector.tks1.uri.encryptData=/tks/agent/tks/encryptData
```

TPS profiles refer to these connectors by their IDs. For example

```
op.enroll.userKey.keyGen.signing.ca.conn=ca1
```

Multiple connector of the same kind (for example, multiple CA connectors) can be defined. This may be useful when one TPS instance serves multiple backend Certificate System servers for different groups of tokens.



NOTE

Automatic failover for connectors in TPS is currently not supported. A manual failover procedure must be performed to point the TPS to alternate CA, KRA, or TKS, as long as they are clones of the original systems.

6.10. REVOCATION ROUTING CONFIGURATION

To configure revocation routing, you must first define a list of relevant CA connectors and add them to the connector list in the following format:

```
tps.connCAList=ca1,ca2
```

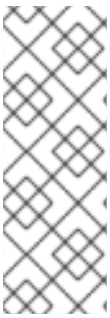
Additionally, you must add the CA signing certificate to the TPS **nssdb** and set up trust:

```
#cd <TPS instance directory>/alias
```

```
#certutil -d . -A -n <CA signing cert nickname> -t “CT,C,C” -i <CA signing cert b64 file name>
```

Finally, the nickname of the CA signing certificate must be added to the connector using the following option:

```
tps.connector.ca1.caNickname=caSigningCert cert-pki-tomcat CA
```



NOTE

During CA discovery, the TPS may automatically calculate the Authority Key Identifier of the CA and add it to the connector configuration. For example:

```
tps.connector.ca1.caSKI=i9wOnN0QZLkzkndAB1MKMcjbRP8=
```

This behavior is expected.

6.11. SETTING UP SERVER-SIDE KEY GENERATION

Server-side key generation means that keys are generated by a Key Recovery Authority (KRA), an

optional Certificate System subsystem. Generating keys by the KRA is necessary to allow recovery of keys on lost or damaged tokens, or key retrieval in the case of external registration. This section describes how to configure server-side key generation in TMS.

During TPS installation you are asked to specify whether you want to use key archival. If you confirm, setup will perform automatic basic configuration, specifically the following parameters:

TPS connector parameters for the KRA:

```
tps.connector.kra1.enable=true
tps.connector.kra1.host=host1.EXAMPLE.com
tps.connector.kra1.maxHttpConns=15
tps.connector.kra1.minHttpConns=1
tps.connector.kra1.nickName=subsystemCert cert-pki-tomcat
tps.connector.kra1.port=8443
tps.connector.kra1.timeout=30
tps.connector.kra1.uri.GenerateKeyPair=/kra/agent/kra/GenerateKeyPair
tps.connector.kra1.uri.TokenKeyRecovery=/kra/agent/kra/TokenKeyRecovery
```

TPS profile-specific parameters for server-side key generation:

```
op.enroll.userKey.keyGen.encryption.serverKeygen.archive=true
op.enroll.userKey.keyGen.encryption.serverKeygen.drm.conn=kra1
op.enroll.userKey.keyGen.encryption.serverKeygen.enable=true
```

Set the **serverKeygen.enable=true** option for **serverKeygen.archive** to take effect.

IMPORTANT

The LunaSA HSM does not support a smaller key size than 2048 bits for RSA encryption.

For example, to configure a key size of 2048 bits, set the following parameter in the **/var/lib/pki/*instance_name*/tps/conf/CS.cfg** file:

```
op.enroll.userKey.keyGen.encryption.keySize=2048
```

TKS configuration:

The following configures the nickname of the transport certificate used for communication between the TKS and KRA (via TPS):

```
tks.drm_transport_cert_nickname=transportCert cert-pki-tomcat KRA
```

The referenced transport certificate must also exist in the TKS instance security module. For example:

```
transportCert cert-pki-tomcat KRA                u,u,u
```

KRA configuration

Depending on the PKCS#11 token, parameters **kra.keygen.temporaryPairs**, **kra.keygen.sensitivePairs**, and **kra.keygen.extractablePairs** can be customized for key generation options. These parameters are all set to **false** by default.

The following values for these parameters have been tested with some of the security modules supported by Red Hat Certificate System:

NSS (when in FIPS mode):

```
kra.keygen.extractablePairs=true
```

nCipher nShield Connect 6000 (works by default without specifying):

For specifying RSA keys:

```
kra.keygen.temporaryPairs=true
```

(Do not specify any other parameters.)

For generating ECC keys:

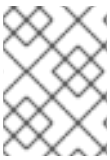
```
kra.keygen.temporaryPairs=true
kra.keygen.sensitivePairs=false
kra.keygen.extractablePairs=true
```

LunaSA CKE - Key Export Model (non-FIPS mode):

```
kra.keygen.temporaryPairs=true
kra.keygen.sensitivePairs=true
kra.keygen.extractablePairs=true
```

NOTE

Gemalto SafeNet LunaSA only supports PKI private key extraction in its CKE - Key Export model, and only in non-FIPS mode. The LunaSA Cloning model and the CKE model in FIPS mode do not support PKI private key extraction.



NOTE

When LunaSA CKE – Key Export Model is in FIPS mode, pki private keys cannot be extracted.

6.12. SETTING UP NEW KEY SETS

This section describes setting up an alternative to the default key set in the Token Processing System (TPS) and in the Token Key Service (TKS).

TKS configuration

The default key set is configured in the TKS using the following options in the **/var/lib/pki/*instance_name*/tks/conf/CS.cfg** file:

```
tks.defKeySet._000=##
tks.defKeySet._001=## Axalto default key set:
tks.defKeySet._002=##
tks.defKeySet._003=## tks.defKeySet.mk_mappings.#02#01=<tokenname>:<nickname>
tks.defKeySet._004=##
tks.defKeySet.auth_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.kek_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.mac_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.defKeySet.nistSP800-108KdfOnKeyVersion=00
tks.defKeySet.nistSP800-108KdfUseCuidAsKdd=false
```

The above configuration defines settings specific to a certain type or class of tokens that can be used in the TMS. The most important part are the 3 developer or (out of the box) session keys, which are used to create a secure channel before symmetric key handover takes place. A different type of key may have different default values for these keys.

The settings describing the **nistSP800** key diversification method control whether this method or the standard Visa method is used. Specifically, the value of the **tks.defKeySet.nistSP800-108KdfOnKeyVersion** option determines that the NIST version will be used. The **nistSP800-108KdfUseCuidAsKdd** option allows you to use the legacy key ID value of CUID during processing. The newer KDD value is most commonly used and therefore this option is disabled (**false**) by default. This allows you to configure a new key set to enable support for a new class of keys.

Example 6.2. Enabling Support for the jForte Class

To enable support for the **jForte** class, set:

```
tks.jForte._000=##
tks.jForte._001=## SAFLink's jForte default key set:
tks.jForte._002=##
tks.jForte._003=## tks.jForte.mk_mappings.#02#01=<tokenname>:<nickname>
tks.jForte._004=##
tks.jForte.auth_key=#30#31#32#33#34#35#36#37#38#39#3a#3b#3c#3d#3e#3f
tks.jForte.kek_key=#50#51#52#53#54#55#56#57#58#59#5a#5b#5c#5d#5e#5f
tks.jForte.mac_key=#40#41#42#43#44#45#46#47#48#49#4a#4b#4c#4d#4e#4f
tks.jForte.nistSP800-108KdfOnKeyVersion=00
tks.jForte.nistSP800-108KdfUseCuidAsKdd=false
```

Note the difference in the 3 static session keys compared to the previous example.

Certificate System supports the Secure Channel Protocol 03 (SCP03) for Giesecke & Devrient (G&D) Smart Cafe 6 smart cards. To enable SCP03 support for these smart cards in a TKS, set in the **/var/lib/pki/*instance_name*/tks/conf/CS.cfg** file:

```
tks.defKeySet.prot3.divers=emv
tks.defKeySet.prot3.diversVer1Keys=emv
tks.defKeySet.prot3.devKeyType=DES3
tks.defKeySet.prot3.masterKeyType=DES3
```

TPS configuration

The TPS must be configured to recognize the new key set when a supported client attempts to perform an operation on a token. The default **defKeySet** is used most often.

The primary method to determine the **keySet** in the TPS involves Section 6.7, "Mapping Resolver Configuration". See the linked section for a discussion of the exact settings needed to establish this resolver mechanism.

If the KeySet Mapping Resolver is not present, several fallback methods are available for the TPS to determine the correct **keySet**:

- You can add the **tps.connector.tks1.keySet=defKeySet** to the **CS.cfg** configuration file of the TPS.
- Certain clients can possibly be configured to explicitly pass the desired **keySet** value. However, the Enterprise Security Client does not have this ability at this point.
- When the TPS calculates the proper **keySet** based on the desired method, all requests to the TKS to help create secure channels pass the **keySet** value as well. The TKS can then use its own **keySet** configuration (described above) to determine how to proceed.

6.13. SETTING UP A NEW MASTER KEY

This section will describe the procedures and configuration required to set up a new master key in the Token Key Service (TKS). See the *Red Hat Certificate System Planning, Installation, and Deployment Guide* for background information.

Procedure 6.1. Creating a New Master Key

1. Obtain internal the PIN required to access the TKS security databases:

```
# cat /var/lib/pki/pki-tomcat/tks/conf/password.conf
internal=649713464822
internaldb=secret12
replicationdb=-752230707
```

2. Open the **alias**/ directory of the TKS instance:

```
# cd /var/lib/pki/pki-tomcat/alias
```

3. Generate a new master key using the **tkstool** utility. For example:

```
# tkstool -M -n new_master -d /var/lib/pki/pki-tomcat/alias -h <token_name>
Enter Password or Pin for "NSS Certificate DB":

Generating and storing the master key on the specified token . . .

Naming the master key "new_master" . . .

Computing and displaying KCV of the master key on the specified token . . .

new_master key KCV:  CA5E 1764
```

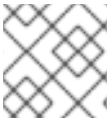
4. Verify that the keys have been properly added to the database:

```
# tkstool -L -d .
```

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

Enter Password or Pin for "NSS Certificate DB":
        <0> new_master
```

6.13.1. Generating and Transporting Wrapped Master Keys (Key Ceremony)

If a master key is going to be used on an external token or in multiple locations, then it must be *wrapped* so that it can be safely transported to the hardware tokens. The **tkstool** utility can be used to generate transport keys, which are then used to send the master key to the facility where the tokens are generated. The process of transferring wrapped master keys is commonly called a *Key Ceremony*.



NOTE

Transport keys can only be used with the master key they were generated with.

Procedure 6.2. Generating and Transporting Wrapped Master Keys

1. Obtain the internal PIN required to access the Token Key Service security databases:

```
# cat /var/lib/pki/pki-tomcat/tks/conf/password.conf

internal=649713464822
internaldb=secret12
replicationdb=-752230707
```

2. Open the TKS instance **alias**/ directory:

```
# cd /var/lib/pki/pki-tomcat/alias
```

3. Create a transport key named **transport**:

```
# tkstool -T -d . -n transport
```

NOTE

The **tkstool** utility prints out the key shares and KCV values for each of the three session keys generated. Save them to a file as they are necessary to regenerate the transport key in new databases later in this procedure, and to regenerate the key if lost.

4. When prompted, fill in the database password. Then, follow on-screen instructions to generate a random seed.

```
A random seed must be generated that will be used in the
creation of your key.  One of the easiest ways to create a
random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter
is full.  DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!
```

```
Continue typing until the progress meter is full:

|************************************************************|

Finished.


Type the word "proceed" and press enter
```

5. The next prompt will generate a series of session keys. Follow on-screen instructions until the final message:

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

6. Use the transport key to generate and wrap a master key and store it in a file named **file**:

```
# tkstool -W -d . -n new_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 "new_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)
```

7. Copy the wrapped master key over to the appropriate locations or facility.

8. If necessary, generate new security databases on the HSM or at the facility:

```
# tkstool -N -d <directory>
```

Alternatively, add the **-I** option to produce a key identical to the one generated originally in a the new database. Regenerating the transport key in this way requires that you input the session key share and KCV for each of the session keys generated earlier in this procedure.

```
# tkstool -I -d <directory> -n verify_transport
```

9. Use the transport key to unwrap the master key stored in the file. Provide the security database PIN when prompted:

```
# tkstool -U -d directory -n new_master -t verify_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 "new_master" . . .
Successfully unwrapped, stored, and named the master key!
```

10. Verify that the keys have been added to the database properly:

```
# 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
   <1> new_master
```

6.14. SETTING UP A TKS/TPS SHARED SYMMETRIC KEY

The shared symmetric key must be present in the **NSS** databases of both the TPS and TKS subsystems. This key is automatically generated when creating the a TPS subsystem. If both the TPS and TKS are installed within the same Tomcat instance, no additional setup is required as the TKS will automatically use the key created by TPS; however, if both subsystems are on separate instances, or even different physical hosts, you must follow the procedure described in this section to securely transport the key to the TKS.

Several possible methods are available to securely transport the shared key between the TPS and TKS:

- The authomatic method: This method works in cases where the subsystem certificates for the TPS are kept in the software NSS database.
- If the above method fails, a fallback manual method is available where the shared key is generated on the TPS using the **tkstool** utility, which can wrap the key from the TPS, allowing for secure transport without exposing the key in transit, and unwrap it into the TKS NSS database.

The following describes the general configuration for both the TPS and TKS, regardless of the method which will be used to import the key. Note that the automatic method will generate these configurations automatically.

TKS

```
tks.useNewSharedSecretNames=true
tps.0.host=dhcp-16-206.sjc.example.com
```

```
tps.0.nickname=TPS-<tps host name>-8443 sharedSecret
tps.0.port=8443
tps.0.userid=,TPS-<tps host name>-8443
tps.list=0
```

NOTE

The above list can be extended when one TKS is connecting to multiple TPS instances.

TPS

```
conn.tks1.tksSharedSymKeyName=TPS-<tps host name>-8443 sharedSecret
```

NOTE

The host name must be the same as the one configured on the TKS side.

6.14.1. Manually Generating and Transporting a Shared Symmetric Key

This section describes how to generate and transport a shared symmetric key manually. This method is useful in cases where automatic generation and transport fails, but should be avoided otherwise.

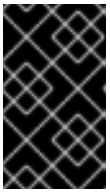
The manual method consists of two procedures. The first one is performed on the Token Key Service side, and the second one on the Token Processing System.

Procedure 6.3. Manual Shared Secret Key Method - TKS side

1. Install the Token Key Service on the first system. See the *Red Hat Certificate System Planning, Installation, and Deployment Guide* for installation instructions.
2. Stop the TKS service:

```
#pki-server stop pki-tomcat
```

3. Change into the **/var/lib/pki/pki-tomcat/alias** directory, and use **tkstool** to create the shared secret key on the TKS. Make sure to generate the shared key before you restart the new TKS instance.



IMPORTANT

The **tkstool** script will display information about the key during the key creation process. Make sure to note down this information, because it will be required later to import the key into the TPS.

```
#cd /var/lib/pki/pki-tomcat/alias
#tkstool -T -d /var/lib/pki/pki-tomcat/tks/alias -n TPS-<tps host name>-8443 sharedSecret
Generating the first session key share . . .
   first session key share:      792F AB89 8989 D902
                                 9429 6137 8632 7CC4
   first session key share KCV:  D1B6 14FD
```

```
Generating the second session key share . . .
   second session key share:      4CDF C8E0 B385 68EC
                                  380B 6D5E 1C19 3E5D
   second session key share KCV:  1EC7 8D4B
Generating the third session key share . . .
   third session key share:       CD32 3140 25B3 C789
                                  B54F 2C94 26C4 9752
   third session key share KCV:  73D6 8633
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .
   transport key KCV:  A8D0 97A2
Storing transport key on final specified token . . .
Naming transport key "sharedSecret" . . .
Successfully generated, stored, and named the transport key!
```

4. Configure the new key in the TKS:

```
tks.useNewSharedSecretNames=true
tps.0.host=dhcp-16-206.sjc.redhat.com
tps.0.nickname=TPS-<tps host name>-8443 sharedSecret
tps.0.port=8443
tps.0.userid=TPS-<tps host name>-8443 sharedSecret
tps.list=0
```

5. Start the TKS:

```
#pki-server start pki-tomcat
```

Procedure 6.4. Manual Shared Secret Key Method - TPS side

1. Install the Token Processing System on the second system. See the *Red Hat Certificate System 10 Planning, Installation, and Deployment Guide* for installation instructions.

2. Stop the TPS service:

```
#pki-server stop pki-tomcat
```

3. Change into the **/var/lib/pki/pki-tomcat/alias** directory, and use **tkstool** to import the shared key into the NSS software token:

```
#cd /var/lib/pki/pki-tomcat/alias
#tkstool -I -d . -n TPS-<tps host name>-8443 sharedSecret
```

   At this point, the script will prompt you for session key shares which were displayed to you when generating and wrapping the shared keys on the TKS side in the procedure above.

4. Configure the shared secret in the TPS:

```
conn.tks1.tksSharedSymKeyName=TPS-<tps host name>-8443 sharedSecret
```

5. Start the TPS service:

```
#pki-server start pki-tomcat
```

6.15. USING DIFFERENT APPLETS FOR DIFFERENT SCP VERSIONS

In Certificate System, the following parameter in the **/var/lib/*instance_name*/tps/conf/CS.cfg** file specifies which applet should be loaded for all Secure Channel Protocol (SCP) versions for each token operation:

```
op.operation.token_type.update.applet.requiredVersion=version
```

However, you can also set individual applets for specific SCP versions, by adding the following parameter:

```
op.operation.token_type.update.applet.requiredVersion.prot.protocol_version=version
```

Certificate System supports setting individual protocol versions for the following operations:

- **format**
- **enroll**
- **pinReset**

Example 6.3. Setting Protocol Versions for Enrollment Operations

To configure a specific applet for SCP03 and a different applet for all other protocols when performing enrollment operations for the **userKey** token:

1. Edit the **/var/lib/*instance_name*/tps/conf/CS.cfg** file:
   a. Set the ***op.enroll.userKey.update.applet.requiredVersion*** parameter to specify the applet used by default. For example:

      ```
      op.enroll.userKey.update.applet.requiredVersion=1.4.58768072
      ```

   b. Set the ***op.enroll.userKey.update.applet.requiredVersion.prot.3*** parameter to configure the applet Certificate System uses for the SCP03 protocol. For example:

      ```
      op.enroll.userKey.update.applet.requiredVersion.prot.3=1.5.558cdcff
      ```

2. Restart Certificate System:

   ```
   pki-server restart instance_name
   ```

For details about enabling SCP03 for Giesecke & Devrient (G&D) Smart Cafe 6 smart cards in a TKS, see Section 6.12, “Setting Up New Key Sets”.

CHAPTER 7. REVOKING CERTIFICATES AND ISSUING CRLS

The Certificate System provides methods for revoking certificates and for producing lists of revoked certificates, called certificate revocation lists (CRLs). This chapter describes the methods for revoking a certificate, describes CMC revocation, and provides details about CRLs and setting up CRLs.

7.1. ABOUT REVOKING CERTIFICATES

Certificates can be revoked by an end user (the original owner of the certificate) or by a Certificate Manager agent. End users can revoke certificates by using the revocation form provided in the end-entities page. Agents can revoke end-entity certificates by using the appropriate form in the agent services interface. Certificate-based (SSL/TLS client authentication) is required in both cases.

An end user can revoke only certificates that contain the same subject name as the certificate presented for authentication. After successful authentication, the server lists the certificates belonging to the end user. The end user can then select the certificate to be revoked or can revoke all certificates in the list. The end user can also specify additional details, such as the date of revocation and revocation reason for each certificate or for the list as a whole.

Agents can revoke certificates based on a range of serial numbers or based on subject name components. When the revocation request is submitted, agents receive a list of certificates from which they can pick the ones to be revoked. For instructions on how agents revoke end-entity certificates, see the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

When revocation requests are approved, the Certificate Manager marks the corresponding certificate records in its internal database as revoked, and, if configured to do so, removes the revoked certificates from the publishing directory. These changes are reflected in the next CRL which the CA issues.

Server and client applications that use public-key certificates as ID tokens need access to information about the validity of a certificate. Because one of the factors that determines the validity of a certificate is its revocation status, these applications need to know whether the certificate being validated has been revoked. The CA has a responsibility to do the following:

- Revoke the certificate if a revocation request is received by the CA and approved.
- Make the revoked certificate status available to parties or applications that need to verify its validity status.

Whenever a certificate is revoked, the Certificate Manager automatically updates the status of the certificate in its internal database, it marks the copy of the certificate in its internal database as revoked and removes the revoked certificate from the publishing directory, if the Certificate Manager is configured to remove the certificate from the database.

One of the standard methods for conveying the revocation status of certificates is by publishing a list of revoked certificates, known a certificate revocation list (CRL). A CRL is a publicly available list of certificates that have been revoked.

The Certificate Manager can be configured to generate CRLs. These CRLs can be created to conform to X.509 standards by enabling extension-specific modules in the CRL configuration. The server supports standard CRL extensions through its CRL issuing points framework; see Section 7.3.3, “Setting CRL Extensions” for more information on setting up CRL extensions for issuing points. The Certificate Manager can generate a CRL every time a certificate is revoked and at periodic intervals. If publishing is set up, the CRLs can be published to a file, an LDAP directory, or an OCSP responder.

A CRL is issued and digitally signed by the CA that issued the certificates listed in the CRL or by an entity that has been authorized by that CA to issue CRLs. The CA may use a single key pair to sign both

the certificates and CRLs it issues or two separate key pairs, one for signing certificates and another one for signing CRLs.

By default, the Certificate Manager uses a single key pair for signing the certificates it issues and CRLs it generates. To create another key pair for the Certificate Manager and use it exclusively for signing CRLs, see Section 7.3.4, “Setting a CA to Use a Different Certificate to Sign CRLs” .

CRLs are generated when issuing points are defined and configured and when CRL generation is enabled.

When CRLs are enabled, the server collects revocation information as certificates are revoked. The server attempts to match the revoked certificate against all issuing points that are set up. A given certificate can match none of the issuing points, one of the issuing points, several of the issuing points, or all of the issuing points. When a certificate that has been revoked matches an issuing point, the server stores the information about the certificate in the cache for that issuing point.

The cache is copied to the internal directory at the intervals set for copying the cache. When the interval for creating a CRL is reached, a CRL is created from the cache. If a delta CRL has been set up for this issuing point, a delta CRL is also created at this time. The full CRL contains all revoked certificate information since the Certificate Manager began collecting this information. The delta CRL contains all revoked certificate information since the last update of the full CRL.

The full CRLs are numbered sequentially, as are delta CRLs. A full CRL and a delta CRL can have the same number; in that case, the delta CRL has the same number as the *next* full CRL. For example, if the full CRL is the first CRL, it is CRL 1. The delta CRL is Delta CRL 2. The data combined in CRL 1 and Delta CRL 2 is equivalent to the next full CRL, which is CRL 2.

NOTE

When changes are made to the extensions for an issuing point, no delta CRL is created with the next full CRL for that issuing point. A delta CRL is created with the *second* full CRL that is created, and then all subsequent full CRLs.

The internal database stores only the latest CRL and delta CRL. As each new CRL is created, the old one is overwritten.

When CRLs are published, each update to the CRL and delta CRL is published to the locations specified in the publishing set up. The method of publishing determines how many CRLs are stored. For file publishing, each CRL that is published to a file using the number for the CRL, so no file is overwritten. For LDAP publishing, each CRL that is published replaces the old CRL in the attribute containing the CRL in the directory entry.

By default, CRLs do not contain information about revoked expired certificates. The server can include revoked expired certificates by enabling that option for the issuing point. If expired certificates are included, information about revoked certificates is not removed from the CRL when the certificate expires. If expired certificates are not included, information about revoked certificates is removed from the CRL when the certificate expires.

7.1.1. User-Initiated Revocation

When an end user submits a certificate revocation request, the first step in the revocation process is for the Certificate Manager to identify and authenticate the end user to verify that the user is attempting to revoke his own certificate, not a certificate belonging to someone else.

In SSL/TSL client authentication, the server expects the end user to present a certificate that has the

same subject name as the one to be revoked and uses that for authentication purposes. The server verifies the authenticity of a revocation request by mapping the subject name in the certificate presented for client authentication to certificates in its internal database. The server revokes the certificate only if the certificate maps successfully to one or more valid or expired certificates in its internal database.

After successful authentication, the server lists the valid or expired certificates that match the subject name of the certificate presented for client authentication. The user can then either select the certificates to be revoked or revoke all certificates in the list.

7.1.2. Reasons for Revoking a Certificate

A Certificate Manager can revoke any certificate it has issued. There are generally accepted reason codes for revoking a certificate that are often included in the CRL, such as the following:

- **0**. Unspecified; no particular reason is given.
- **1**. The private key associated with the certificate was compromised.
- **2**. The private key associated with the CA that issued the certificate was compromised.
- **3**. The owner of the certificate is no longer affiliated with the issuer of the certificate and either no longer has rights to the access gained with the certificate or no longer needs it.
- **4**. Another certificate replaces this one.
- **5**. The CA that issued the certificate has ceased to operate.
- **6**. The certificate is on hold pending further action. It is treated as revoked but may be taken off hold in the future so that the certificate is active and valid again.
- **8**. The certificate is going to be *removed* from the CRL because it was removed from hold. This only occurs in delta CRLs.
- **9**. The certificate is revoked because the privilege of the owner of the certificate has been withdrawn.

A certificate can be revoked by administrators, agents, and end entities. Agents and administrators with agent privileges can revoke certificates using the forms in the agent services page. End users can revoke certificates using the forms in the **Revocation** tab of the end-entity interface. End users can revoke only their own certificates, whereas agents and administrators can revoke any certificates issued by the server. End users are also required to authenticate to the server in order to revoke a certificate.

Whenever a certificate is revoked, the Certificate Manager updates the status of the certificate in its internal database. The server uses the entries in the internal database to track of all revoked certificates, and, when configured, it makes the CRLs public by publishing it to a central repository to notify other users that the certificates in the list are no longer valid.

7.1.3. CRL Issuing Points

Because CRLs can grow very large, there are several methods to minimize the overhead of retrieving and delivering large CRLs. One of these methods partitions the entire certificate space and associates a separate CRL with every partition. This partition is called a *CRL issuing point*, the location where a subset of all the revoked certificates is maintained. Partitioning can be based on whether the revoked certificate is a CA certificate, whether it was revoked for a specific reason, or whether it was issued using a specific profile. Each issuing point is identified by its name.

By default, the Certificate Manager generates and publishes a single CRL, the *master CRL*. An issuing point can generate CRLs for all certificates, for only CA signing certificates, or for all certificates including expired certificates.

Once the issuing points have been defined, they can be included in certificates so that an application that needs to check the revocation status of a certificate can access the CRL issuing points specified in the certificate instead of the master or main CRL. Since the CRL maintained at the issuing point is smaller than the master CRL, checking the revocation status is much faster.

CRL distribution points can be associated with certificates by setting the **CRLDistributionPoint** extension.

7.1.4. Delta CRLs

Delta CRLs can be issued for any defined issuing point. A delta CRL contains information about any certificates revoked since the last update to the full CRL. Delta CRLs for an issuing point are created by enabling the **DeltaCRLIndicator** extension.

7.1.5. Publishing CRLs

The Certificate Manager can publish the CRL to a file, an LDAP-compliant directory, or to an OCSP responder. Where and how frequently CRLs are published are configured in the Certificate Manager, as described in Chapter 9, *Publishing Certificates and CRLs*.

Because CRLs can be very large, publishing CRLs can take a very long time, and it is possible for the process to be interrupted. Special publishers can be configured to publish CRLs to a file over HTTP1.1, and, if the process is interrupted, the CA subsystem's web server can resume publishing at the point it was interrupted, instead of having to begin again. This is described in Section 9.8, "Setting up Resumable CRL Downloads".

7.1.6. Certificate Revocation Pages

The end-entities page of the Certificate Manager includes default HTML forms for revocation authenticated by an SSL/TLS client. The forms are accessible from the **Revocation** tab. You can see the form for such a revocation by clicking the **User Certificate** link.

To change the form appearance to suit organization's requirements, edit the **UserRevocation.html**, the form that allows the SSL/TSL client authenticated revocation of client or personal certificates. The file is in the **/var/lib/*instance_name*/webapps/*subsystem_type*/ee/*subsystem_type*** directory.

7.2. PERFORMING A CMC REVOCATION

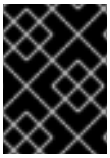
Similar to Certificate Management over CMS (CMC) enrollment, CMC revocation enables users to set up a revocation client, and sign the revocation request with either an agent certificate or a user certificate with a matching **subjectDN** attribute. Then the user can send the signed request to the Certificate Manager.

Alternatively, CMC revocation can also be authenticated using the Shared Secret Token mechanism. For details, see *Enabling the CMC Shared Secret Feature*.

Regardless of whether a user or agent signs the request or if a Shared Secret Token is used, the Certificate Manager automatically revokes the certificate when it receives a valid revocation request.

Certificate System provides the following utilities for CMC revocation requests:

- **CMCRequest**. For details, see Section 7.2.1, "Revoking a Certificate Using **CMCRequest**".
- **CMCRevoke**. For details, see Section 7.2.2, "Revoking a Certificate Using **CMCRevoke**".



IMPORTANT

Red Hat recommends using the **CMCRequest** utility to generate CMC revocation requests, because it provides more options than **CMCRevoke**.

7.2.1. Revoking a Certificate Using CMCRequest

To revoke a certificate using **CMCRequest**:

1. Create a configuration file for the CMC revocation request, such as **/home/user_name/cmc-request.cfg**, with the following content:

```
#numRequests: Total number of PKCS10 requests or CRMF requests.
numRequests=1

#output: full path for the CMC request in binary format
output=/home/user_name/cmc.revoke.userSigned.req

#tokenname: name of token where user signing cert can be found
#(default is internal)
tokenname=internal

#nickname: nickname for user signing certificate which will be used
#to sign the CMC full request.
nickname=signer_user_certificate

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
dbdir=/home/user_name/.dogtag/nssdb/

#password: password for cert9.db which stores the user signing
#certificate and keys
password=myPass

#format: request format, either pkcs10 or crmf.
format=pkcs10

## revocation parameters
revRequest.enable=true
revRequest.serial=45
revRequest.reason=unspecified
revRequest.comment=user test revocation
revRequest.issuer=issuer
revRequest.sharedSecret=shared_secret
```

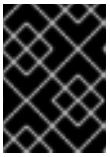
2. Create the CMC request:

```
# CMCRequest /home/user_name/cmc-request.cfg
```

If the command succeeds, the **CMCRequest** utility stores the CMC request in the file specified in the ***output*** parameter in the request configuration file.

3. Create a configuration file, such as /**home/user_name/cmc-submit.cfg**, which you use in a later step to submit the CMC revocation request to the CA. Add the following content to the created file:

```
#host: host name for the http server
host=>server.example.com

#port: port number
port=8443

#secure: true for secure connection, false for nonsecure connection
secure=true

#input: full path for the enrollment request, the content must be
#in binary format
input=/home/user_name/cmc.revoke.userSigned.req

#output: full path for the response in binary format
output=/home/user_name/cmc.revoke.userSigned.resp

#tokenname: name of token where SSL client authentication certificate
#can be found (default is internal)
#This parameter will be ignored if secure=false
tokenname=internal

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
#This parameter will be ignored if secure=false
dbdir=/home/user_name/.dogtag/nssdb/

#clientmode: true for client authentication, false for no client
#authentication. This parameter will be ignored if secure=false
clientmode=true

#password: password for cert9.db
#This parameter will be ignored if secure=false and clientauth=false
password=password

#nickname: nickname for client certificate
#This parameter will be ignored if clientmode=false
nickname=signer_user_certificate
```



IMPORTANT

If the CMC revocation request is signed, set the ***secure*** and ***clientmode*** parameters to **true** and, additionally, fill the ***nickname*** parameter.

4. Depending on who signed the request, the ***servlet*** parameter in the configuration file for **HttpClient** must be set accordingly:

   - If an agent signed the request, set:

     ```
     servlet=/ca/ee/ca/profileSubmitCMCFull
     ```

   - If a user signed the request, set:

```
servlet=/ca/ee/ca/profileSubmitSelfSignedCMCFull
```

5. Submit the CMC request:

```
# HttpClient /home/user_name/cmc-submit.cfg
```

For further details about revoking a certificate using **CMCRequest**, see the CMCRequest(1) man page.

7.2.2. Revoking a Certificate Using CMCRevoke

The CMC revocation utility, **CMCRevoke**, is used to sign a revocation request with an agent's certificate. This utility simply passes the required information – certificate serial number, issuer name, and revocation reason – to identify the certificate to revoke, and then the require information to identify the CA agent performing the revocation (certificate nickname and the database with the certificate).

The reason the certificate is being revoked can be any of the following (with the number being the value passed to the **CMCRevoke** utility):

- **0** – unspecified
- **1** – the key was compromised
- **2** – the CA key was compromised
- **3** – the employee's affiliation changed
- **4** – the certificate has been superseded
- **5** – cessation of operation
- **6** – the certificate is on hold

The available tool arguments are described in detail in the *Command-Line Tools Guide*.

7.2.2.1. Testing CMCRevoke

1. Create a CMC revocation request for an existing certificate.

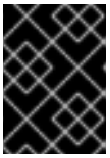
```
CMCRevoke -d/path/to/agent-cert-db -nnickname -iissuerName -sserialName -mreason -
ccomment
```

For example, if the directory containing the agent certificate is **~jsmith/.mozilla/firefox/**, the nickname of the certificate is **AgentCert**, and the serial number of the certificate is **22**, the command is as shown:

```
CMCRevoke -d"~jsmith/.mozilla/firefox/" -n"ManagerAgentCert" -i"cn=agentAuthMgr" -s22 -
m0 -c"test comment"
```

NOTE

Surround values that include spaces in quotation marks.



IMPORTANT

Do not have a space between the argument and its value. For example, giving a serial number of 26 is **-s26**, not **-s 26**.

2. Open the end-entities page.

```
https://server.example.com:8443/ca/ee/ca
```

3. Select the **Revocation** tab.
4. Select the **CMC Revoke** link on the menu.
5. Paste the output from the **CMCRevoke** into the text area.
6. Remove **-----BEGIN NEW CERTIFICATE REQUEST-----** and **----END NEW CERTIFICATE REQUEST-----** from the pasted content.
7. Click **Submit**.
8. The returned page should confirm that correct certificate has been revoked.

7.3. ISSUING CRLS

1. The Certificate Manager uses its CA signing certificate key to sign CRLs. To use a separate signing key pair for CRLs, set up a CRL signing key and change the Certificate Manager configuration to use this key to sign CRLs. See Section 7.3.4, "Setting a CA to Use a Different Certificate to Sign CRLs" for more information.
2. Set up CRL issuing points. An issuing point is already set up and enabled for a master CRL.

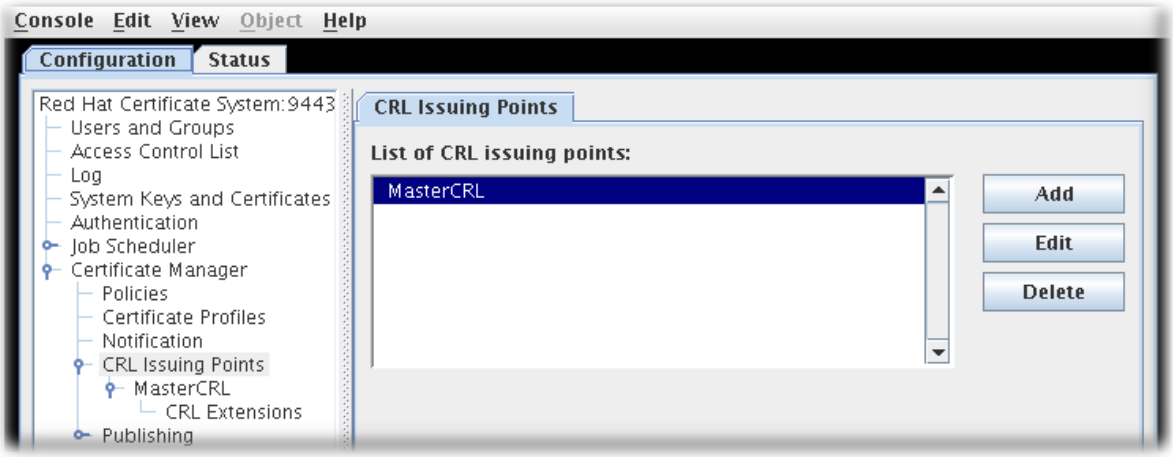


Figure 7.1. Default CRL Issuing Point

Additional issuing points for the CRLs can be created. See Section 7.3.1, "Configuring Issuing Points" for details.

There are five types of CRLs the issuing points can create, depending on the options set when configuring the issuing point to define what the CRL will list:

- *Master CRL* contains the list of revoked certificates from the entire CA.

- *ARL* is an Authority Revocation List containing only revoked CA certificates.
- *CRL with expired certificates* includes revoked certificates that have expired in the CRL.
- *CRL from certificate profiles* determines the revoked certificates to include based on the profiles used to create the certificates originally.
- *CRLs by reason code* determines the revoked certificates to include based on the revocation reason code.

3. Configure the CRLs for each issuing point. See Section 7.3.2, "Configuring CRLs for Each Issuing Point" for details.
4. Set up the CRL extensions which are configured for the issuing point. See Section 7.3.3, "Setting CRL Extensions" for details.
5. Set up the delta CRL for an issuing point by enabling extensions for that issuing point, **DeltaCRLIndicator** or **CRLNumber**.
6. Set up the **CRLDistributionPoint** extension to include information about the issuing point.
7. Set up publishing CRLs to files, an LDAP directory, or an OCSP responder. See Chapter 9, *Publishing Certificates and CRLs* for details about setting up publishing.

7.3.1. Configuring Issuing Points

Issuing points define which certificates are included in a new CRL. A master CRL issuing point is created by default for a master CRL containing a list of all revoked certificates for the Certificate Manager.

To create a new issuing point, do the following:

1. Open the Certificate System Console.

```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, expand **Certificate Manager** from the left navigation menu. Then select **CRL Issuing Points**.
3. To edit an issuing point, select the issuing point, and click **Edit**. The only parameters which can be edited are the name of the issuing point and whether the issuing point is enabled or disabled.

   To add an issuing point, click **Add**. The CRL Issuing Point Editor window opens.

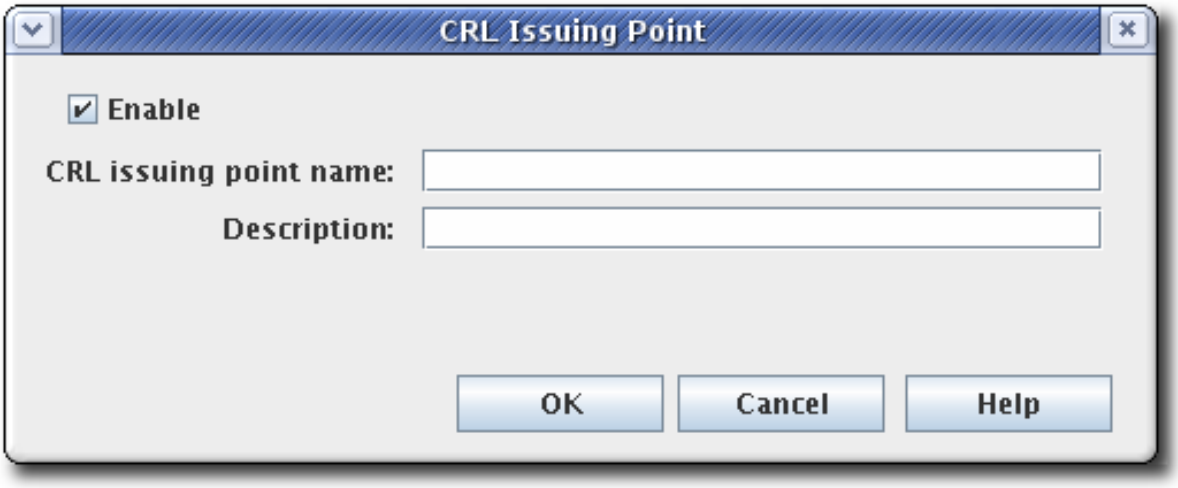
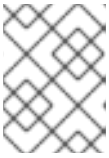


Figure 7.2. CRL Issuing Point Editor



NOTE

If some fields do not appear large enough to read the content, expand the window by dragging one of the corners.

Fill in the following fields:

- **Enable**. Enables the issuing point if selected; deselect to disable.
- **CRL Issuing Point name**. Gives the name for the issuing point; spaces are not allowed.
- **Description**. Describes the issuing point.

4. Click **OK**.

To view and configure a new issuing point, close the CA Console, then open the Console again. The new issuing point is listed below the **CRL Issuing Points** entry in the navigation tree.

Configure CRLs for the new issuing point, and set up any CRL extensions that will be used with the CRL. See Section 7.3.2, "Configuring CRLs for Each Issuing Point" for details on configuring an issuing point. See Section 7.3.3, "Setting CRL Extensions" for details on setting up the CRL extensions. All the CRLs created appear on the **Update Revocation List** page of the agent services pages.

NOTE

pkiconsole is being deprecated.

7.3.2. Configuring CRLs for Each Issuing Point

Information, such as the generation interval, the CRL version, CRL extensions, and the signing algorithm, can all be configured for the CRLs for the issuing point. The CRLs must be configured for each issuing point.

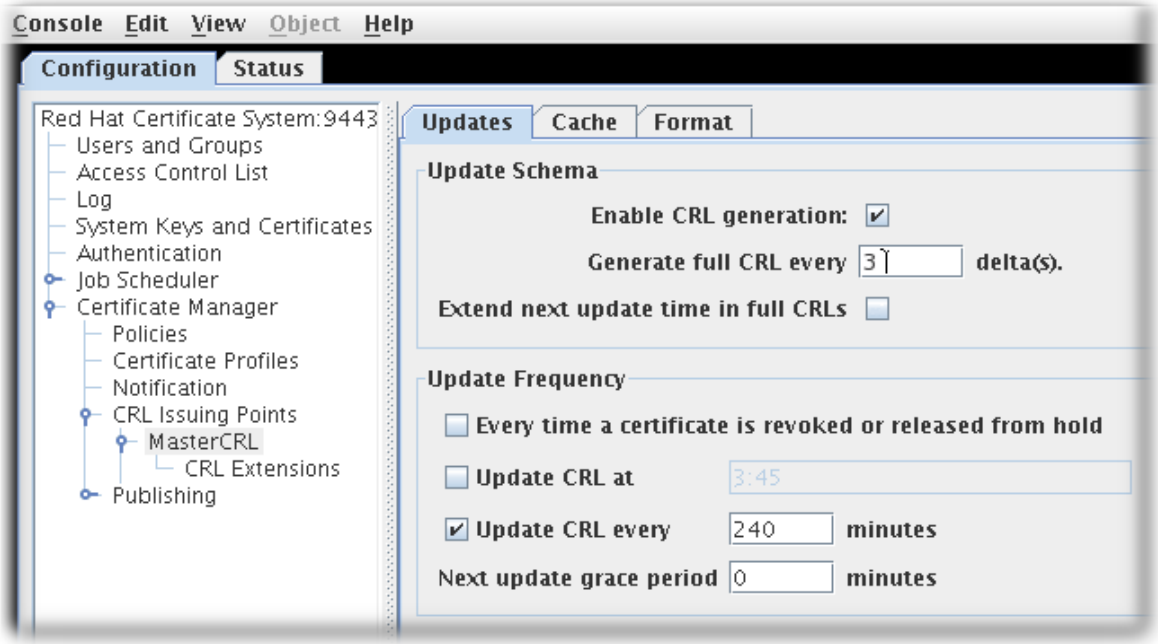
1. Open the CA console.

```
pkiconsole https://server.example.com:8443/ca
```

2. In the navigation tree, select **Certificate Manager**, and then select **CRL Issuing Points**.

3. Select the issuing point name below the **Issuing Points** entry.

4. Configure how and how often the CRLs are updated by supplying information in the **Update** tab for the issuing point. This tab has two sections, **Update Schema** and **Update Frequency**.



- The **Update Schema** section has the following options:

  - **Enable CRL generation**. This checkbox sets whether CRLs are generated for that issuing point.

  - **Generate full CRL every # delta(s)**. This field sets how frequently CRLs are created in relation to the number of changes.

  - **Extend next update time in full CRLs**. This provides an option to set the **nextUpdate** field in the generated CRLs. The ***nextUpdate*** parameter shows the date when the next CRL is issued, regardless of whether it is a full or delta CRL. When using a combination of full and delta CRLs, enabling ***Extend next update time in full CRLs*** will make the ***nextUpdate*** parameter in a full CRL show when the next *full* CRL will be issued. Otherwise, the ***nextUpdate*** parameter in the full CRL will show when the next *delta* CRL will be issued, since the delta will be the next CRL to be issued.

- The **Update Frequency** section sets the different intervals when the CRLs are generated and issued to the directory.

  - **Every time a certificate is revoked or released from hold**. This sets the Certificate Manager to generate the CRL every time it revokes a certificate. The Certificate Manager attempts to issue the CRL to the configured directory whenever it is generated. Generating a CRL can be time consuming if the CRL is large. Configuring the Certificate Manager to generate CRLs every time a certificate is revoked may engage the server for a considerable amount of time; during this time, the server will not be able to update the directory with any changes it receives.

    This setting is not recommended for a standard installation. This option should be selected to test revocation immediately, such as testing whether the server issues the CRL to a flat file.

- **Update the CRL at**. This field sets a daily time when the CRL should be updated. To specify multiple times, enter a comma-separate list of times, such as **01:50,04:55,06:55**. To enter a schedule for multiple days, enter a comma-separated list to set the times within the same day, and then a semicolon separated list to identify times for different days. For example, this sets revocation on Day 1 of the cycle at 1:50am, 4:55am, and 6:55am and then Day 2 at 2am, 5am, and 5pm:

  ```
  01:50,04:55,06:55;02:00,05:00,17:00
  ```

- **Update the CRL every**. This checkbox enables generating CRLs at the interval set in the field. For example, to issue CRLs every day, select the checkbox, and enter **1440** in this field.

- **Next update grace period**. If the Certificate Manager updates the CRL at a specific frequency, the server can be configured to have a grace period to the next update time to allow time to create the CRL and issue it. For example, if the server is configured to update the CRL every 20 minutes with a grace period of 2 minutes, and if the CRL is updated at 16:00, the CRL is updated again at 16:18.

IMPORTANT

Due to a known issue, when currently setting full and delta Certificate Revocation List schedules, the **Update CRL every time a certificate is revoked or released from hold** option also requires you to fill out the two **grace period** settings. Thus, in order to select this option you need to first select the **Update CRL every** option and enter a number for the **Next update grace period # minutes** box.

5. The **Cache** tab sets whether caching is enabled and the cache frequency.

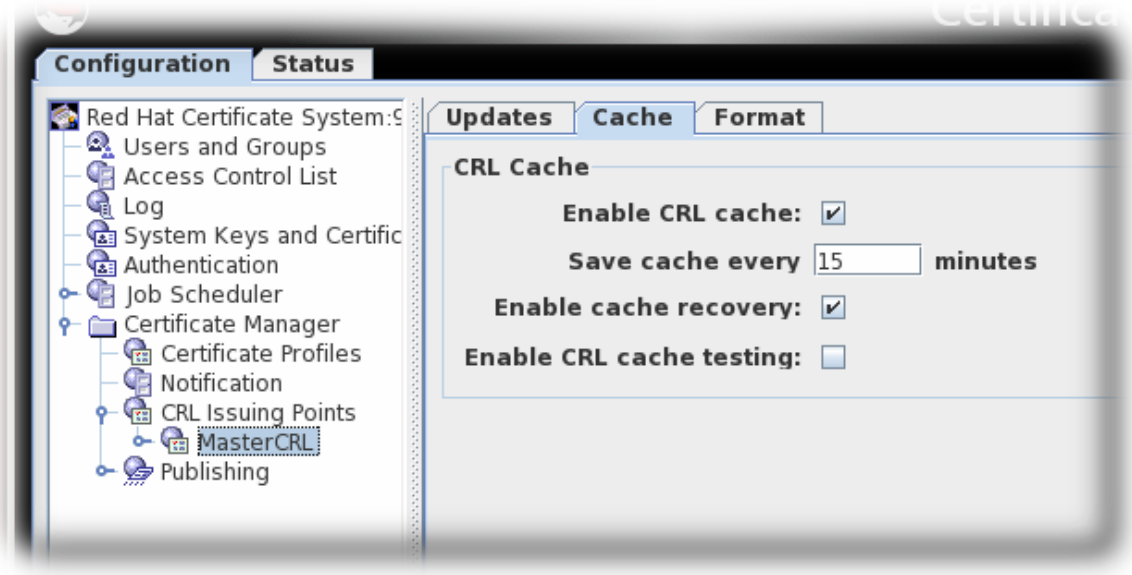


Figure 7.3. CRL Cache Tab

- **Enable CRL cache**. This checkbox enables the cache, which is used to create delta CRLs. If the cache is disabled, delta CRLs will not be created. For more information about the cache, see Section 7.1, “About Revoking Certificates”.

- **Update cache every**. This field sets how frequently the cache is written to the internal database. Set to **0** to have the cache written to the database every time a certificate is revoked.

- **Enable cache recovery**. This checkbox allows the cache to be restored.

- **Enable CRL cache testing**. This checkbox enables CRL performance testing for specific CRL issuing points. CRLs generated with this option should not be used in deployed CAs, as CRLs issued for testing purposed contain data generated solely for the purpose of performance testing.

6. The **Format** tab sets the formatting and contents of the CRLs that are created. There are two sections, **CRL Format** and **CRL Contents**.

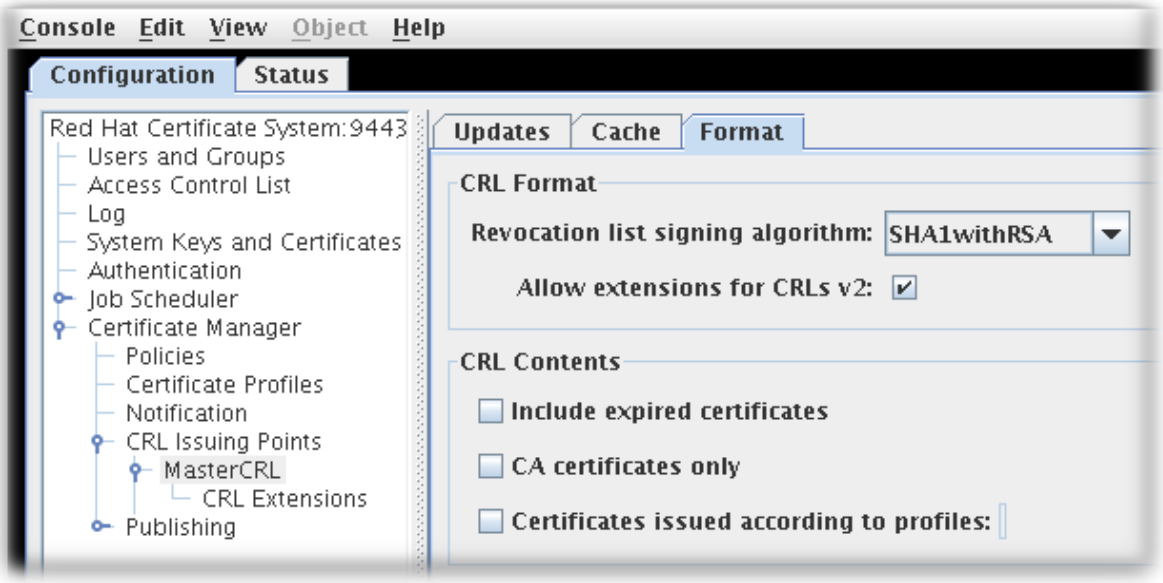
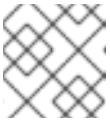


Figure 7.4. CRL Format Tab

- The **CRL Format** section has two options:

  - **Revocation list signing algorithm** is a drop down list of allowed ciphers to encrypt the CRL.

  - **Allow extensions for CRL v2** is a checkbox which enabled CRL v2 extensions for the issuing point. If this is enabled, set the required CRL extensions described in Section 7.3.3, “Setting CRL Extensions”.



NOTE

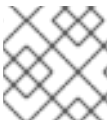
Extensions must be turned on to create delta CRLs.

- The **CRL Contents** section has three checkboxes which set what types of certificates to include in the CRL:

  - **Include expired certificates**. This includes revoked certificates that have expired. If this is enabled, information about revoked certificates remains in the CRL after the certificate expires. If this is not enabled, information about revoked certificates is removed when the certificate expires.

- **CA certificates only**. This includes only CA certificates in the CRL. Selecting this option creates an Authority Revocation List (ARL), which lists only revoked CA certificates.
- **Certificates issued according to profiles**. This only includes certificates that were issued according to the listed profiles; to specify multiple profiles, enter a comma-separated list.

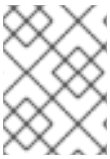
7. Click **Save**.
8. Extensions are allowed for this issuing point and can be configured. See Section 7.3.3, "Setting CRL Extensions" for details.



NOTE

pkiconsole is being deprecated.

7.3.3. Setting CRL Extensions



NOTE

Extensions only need configured for an issuing point if the **Allow extensions for CRLs v2** checkbox is selected for that issuing point.

When the issuing point is created, three extensions are automatically enabled: **CRLReason**, **InvalidityDate**, and **CRLNumber**. Other extensions are available but are disabled by default. These can be enabled and modified. For more information about the available CRL extensions, see Section B.4.2, "Standard X.509 v3 CRL Extensions Reference".

To configure CRL extensions, do the following:

1. Open the CA console.

```
pkiconsole https://server.example.com:8443/ca
```

2. In the navigation tree, select **Certificate Manager**, and then select **CRL Issuing Points**.
3. Select the issuing point name below the **Issuing Points** entry, and select the **CRL Extension** entry below the issuing point.

   The right pane shows the **CRL Extensions Management** tab, which lists configured extensions.

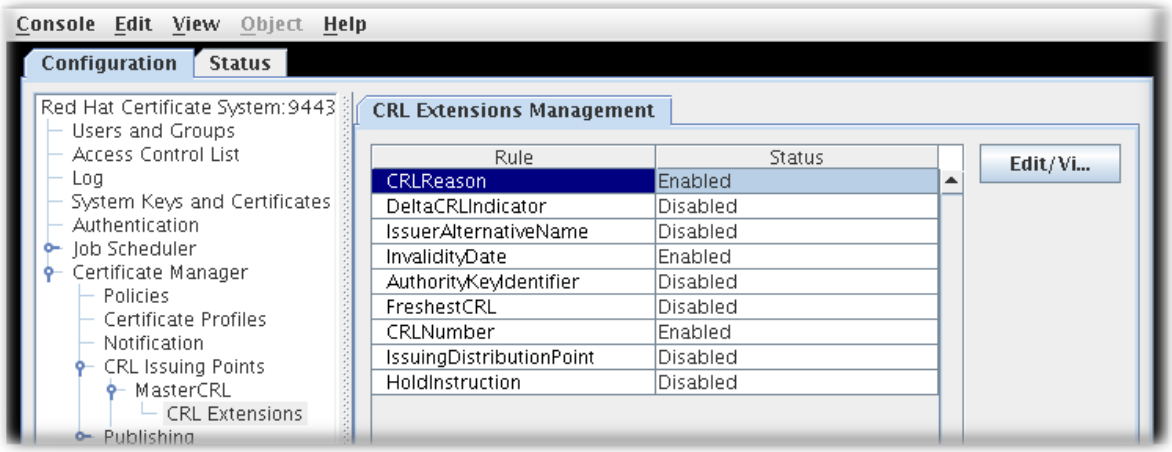


Figure 7.5. CRL Extensions

4. To modify a rule, select it, and click **Edit/View**.

5. Most extensions have two options, enabling them and setting whether they are critical. Some require more information. Supply all required values. See Section B.4.2, "Standard X.509 v3 CRL Extensions Reference" for complete information about each extension and the parameters for those extensions.

6. Click **OK**.

7. Click **Refresh** to see the updated status of all the rules.

NOTE

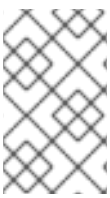
pkiconsole is being deprecated.

7.3.4. Setting a CA to Use a Different Certificate to Sign CRLs

For instruction on how to configure this feature by editing the **CS.cfg** file, see the *Setting a CA to Use a Different Certificate to Sign CRLs* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

7.3.5. Generating CRLs from Cache

By default, CRLs are generated from the CA's internal database. However, revocation information can be collected as the certificates are revoked and kept in memory. This revocation information can then be used to update CRLs from memory. Bypassing the database searches that are required to generate the CRL from the internal database significantly improves performance.



NOTE

Because of the performance enhancement from generating CRLs from cache, enable the ***enableCRLCache*** parameter in most environments. However, the ***Enable CRL cache testing*** parameter *should not* be enabled in a production environment.

7.3.5.1. Configuring CRL Generation from Cache in the Console

NOTE

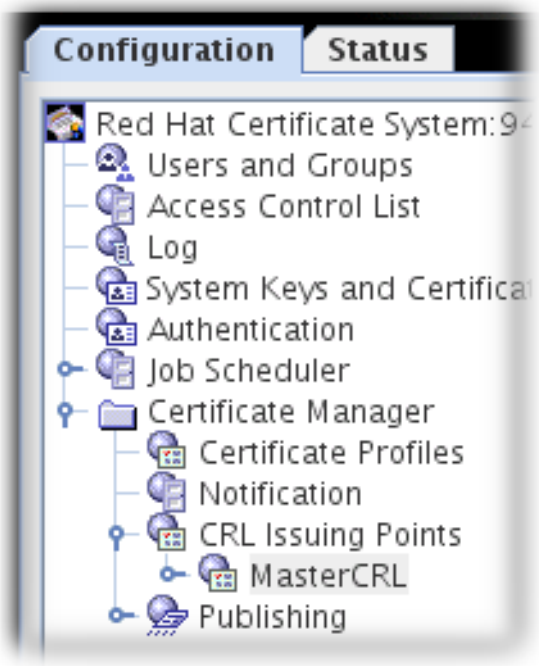
pkiconsole is being deprecated.

1. Open the console.

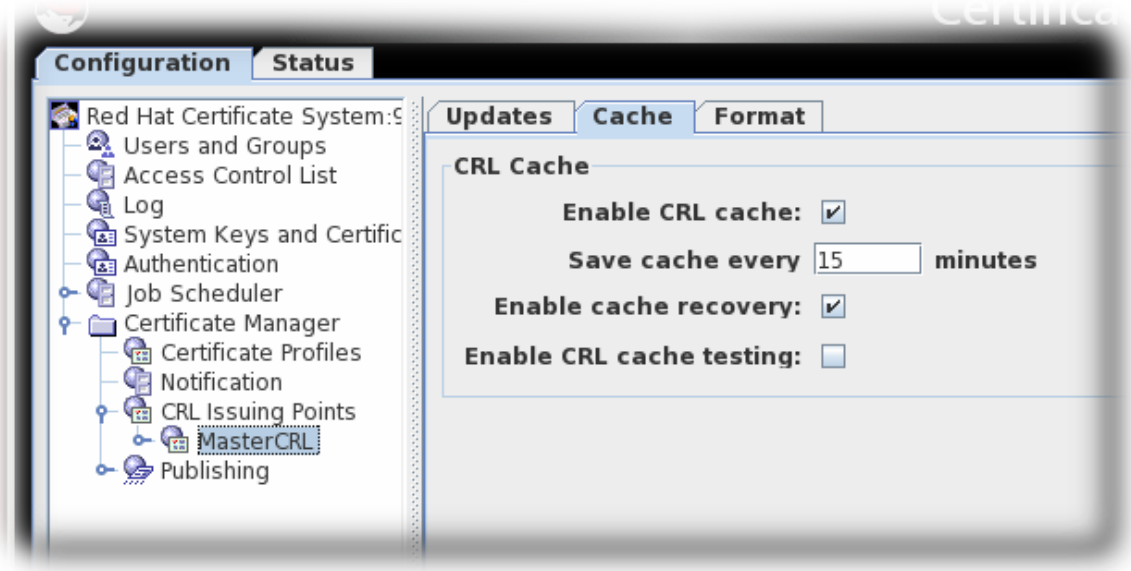
```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, expand the **Certificate Manager** folder and the **CRL Issuing Points** subfolder.

3. Select the **MasterCRL** node.



4. Select **Enable CRL cache**.



5. Save the changes.

7.3.5.2. Configuring CRL Generation from Cache in CS.cfg

For instruction on how to configure this feature by editing the **CS.cfg** file, see the *Configuring CRL Generation from Cache in CS.cfg* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

7.4. SETTING FULL AND DELTA CRL SCHEDULES

CRLs are generated periodically. Setting that period is touched on in the configuration in Section 7.3.2, "Configuring CRLs for Each Issuing Point".

CRLs are issued according to a time-based schedule. CRLs can be issued every single time a certificate is revoked, at a specific time of day, or once every so-many minutes.

Time-based CRL generation schedules apply to every CRL that is generated. There are two kinds of CRLs, full CRLs and delta CRLs. A full CRL has a record of every single revoked certificate, whereas delta CRLs contain only the certificates that have been revoked since the last CRL (delta or full) was generated.

By default, full CRLs are generated at every specified interval in the schedule. It is possible space out the time between generating full CRLs by generating interim *delta* CRLs. The generation interval is configured in the *CRL schema*, which sets the scheme for generating delta and full CRLs.

If the interval is set to 3, for example, then the first CRL generated will be both a full and delta CRL, then the next two generation updates are delta CRLs only, and then the fourth interval is both a full and delta CRL again. In other words, every third generation interval has both a full CRL and a delta CRL.

```
Interval   1, 2, 3, 4, 5, 6, 7 ...
Full CRL   1        4        7 ...
Delta CRL  1, 2, 3, 4, 5, 6, 7 ...
```

NOTE

For delta CRLs to be generated in addition to full CRLs, the CRL cache must be enabled.

7.4.1. Configuring CRL Update Intervals in the Console

NOTE

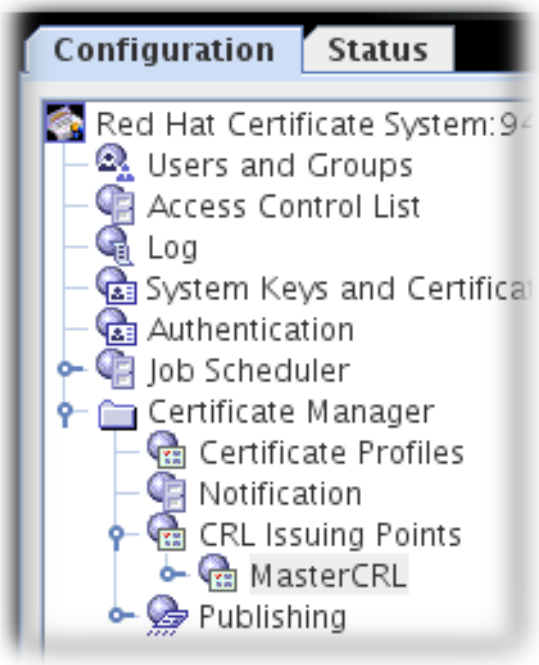
pkiconsole is being deprecated.

1. Open the console.

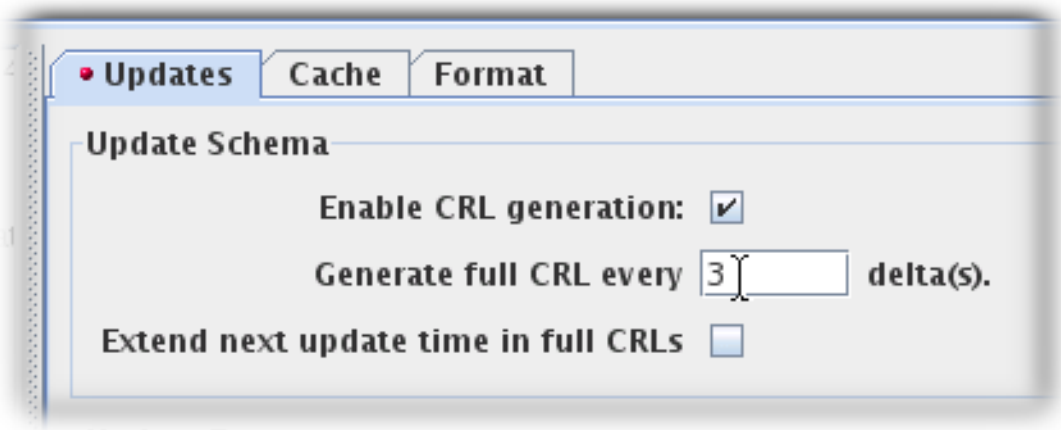
   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, expand the **Certificate Manager** folder and the **CRL Issuing Points** subfolder.

3. Select the **MasterCRL** node.



4. Enter the required interval in the **Generate full CRL every # delta(s)** field.

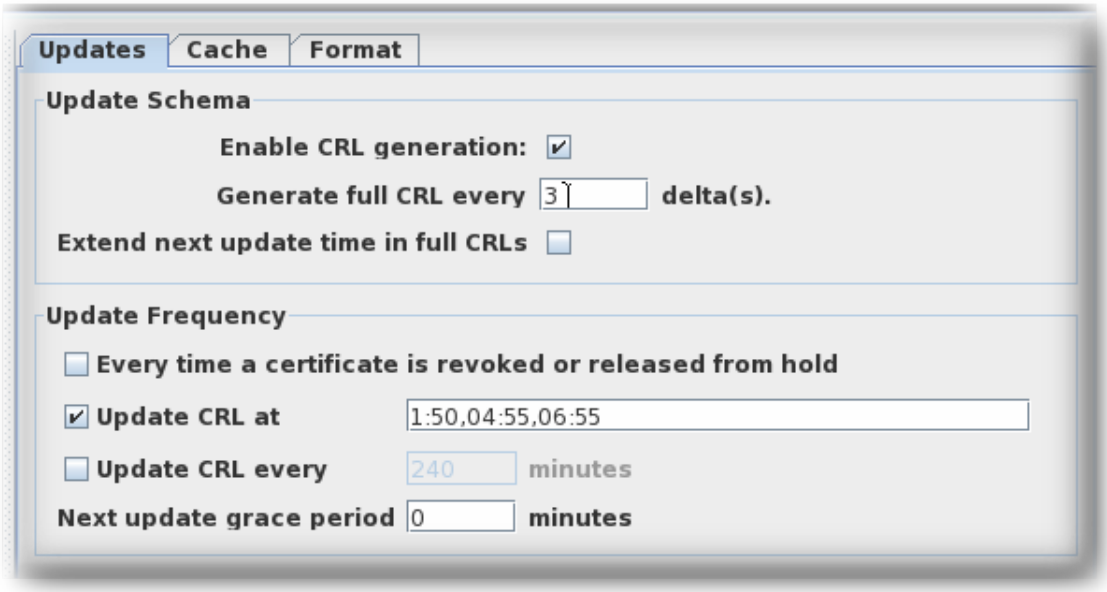


5. Set the update frequency, either by specifying the occasion of a certificate revocation, a cyclical interval or set times for the updates to occur:

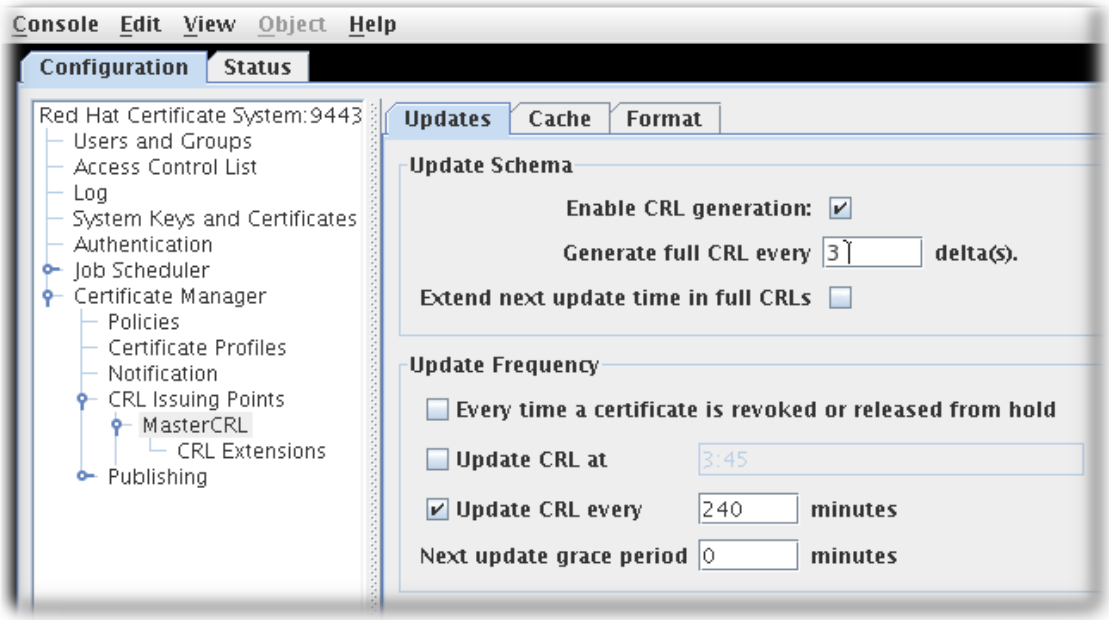
    - Select the **Update CRL every time a certificate is revoked or released from hold** checkbox. The **Update CRL every time a certificate is revoked or released from hold** option also requires you to fill out the two **Grace period** settings. This is a known issue, and the bug is being tracked in Red Hat Bugzilla.

    - Select the **Update CRL every time a certificate is revoked or released from hold** checkbox.

    - Select the **Update CRL at** checkbox and enter specific times separated by commas, such as **01:50,04:55,06:55**.



- Select **Update CRL every** checkbox and enter the required interval, such as **240**.



6. Save the changes.

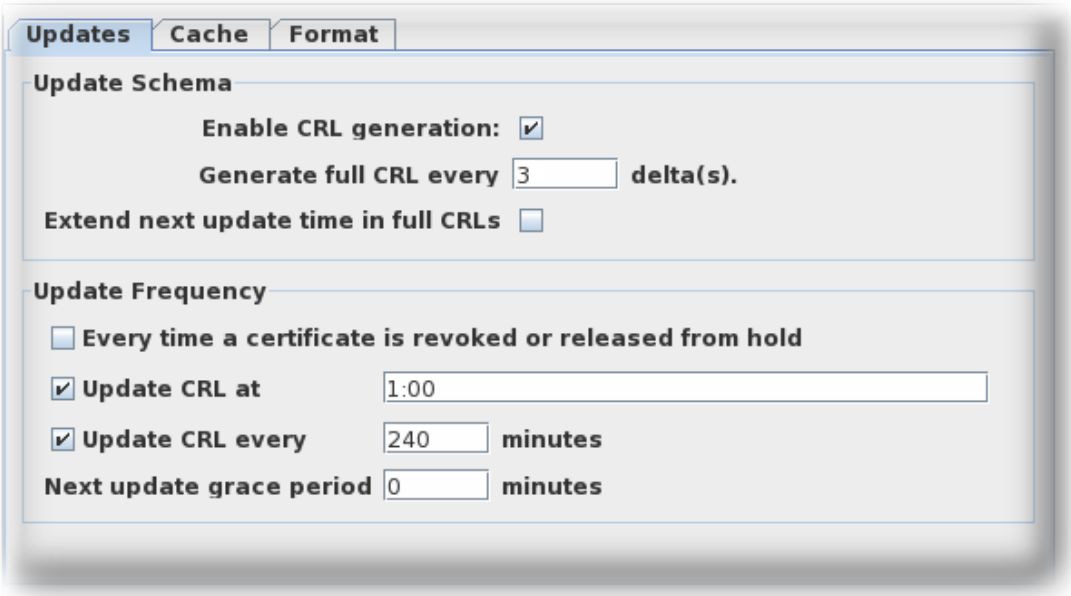
IMPORTANT

The **Update CRL every time a certificate is revoked or released from hold** option also requires you to fill out the two **grace period** settings. This is a known issue, and the bug is being tracked in Red Hat Bugzilla.

NOTE

Schedule drift can occur when updating CRLs by interval. Typically, drift occurs as a result of manual updates and CA restarts.

To prevent schedule drift, select the **Update CRL at** checkbox and enter a value. The interval updates will resynchronize with the **Update CRL at** value every 24 hours.



Only one **Update CRL at** value will be accepted when updating CRLs by interval.

7.4.2. Configuring Update Intervals for CRLs in CS.cfg

For instruction on how to configure this feature by editing the **CS.cfg** file, see the *Configuring Update Intervals for CRLs in CS.cfg* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

7.4.3. Configuring CRL Generation Schedules over Multiple Days

By default, CRL generation schedules cover 24 hours. Also, by default, when full and delta CRLs are enabled full CRLs occur at specific intervals in place of one or all delta CRLs, i.e., every third update.

To set CRL generation schedules across multiple days, the list of times uses commas to separate times within the same day and a semicolon to delimit days:

```
ca.crl.MasterCRL.dailyUpdates=01:00,03:00,18:00;02:00,05:00,17:00
```

This example updates CRLs on day one of the schedule at 01:00, 03:00, and 18:00, and on day two of the schedule at 02:00, 05:00, and 17:00. On day three the cycle starts again.

NOTE

The semicolon indicates a new day. Starting the list with a semicolon results in an initial day where no CRLs are generated. Likewise, ending the list with a semicolon adds a final day to the schedule where no CRLs are generated. Two semicolons together result in a day with no CRL generation.

To set full CRL updates independent of delta updates, the list of times accepts time values prepended with an asterisk to indicate when full CRL updates should occur:

```
ca.crl.MasterCRL.dailyUpdates=01:00,03:00,18:00,*23:00;02:00,05:00,21:00,*23:30
```

This example generates delta CRL updates on day one at 01:00, 03:00, and 18:00, with a full and delta CRL update at 23:00. On day two, delta CRLs are updated at 02:00, 05:00, and 21:00, with a full and delta CRL update at 23:30. On day three, the cycle starts again.

NOTE

The semicolon and asterisk syntax works in both the console and when manually editing the **CS.cfg** file.

7.5. ENABLING REVOCATION CHECKING

Revocation checking means that a Certificate System subsystem verifies that a certificate is both valid and not revoked when an agent or administrator attempts to access the instance's secure interfaces. This leverages a local OCSP service (either a CA's internal OCSP service or a separate OCSP responder) to check the revocation status of the certificate.

OCSP configuration is covered in Section 7.6, "Using the Online Certificate Status Protocol (OCSP) Responder".

See *Enabling Automatic Revocation Checking on the CA* in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

See *Enabling Certificate Revocation Checking for Subsystems* in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

7.6. USING THE ONLINE CERTIFICATE STATUS PROTOCOL (OCSP) RESPONDER

7.6.1. Setting up the OCSP Responder

If a CA within the security domain is selected when the Online Certificate Status Manager is configured, there is no extra step required to configure the OCSP service. The CA's CRL publishing is set up automatically, and its signing certificate is automatically added and trusted in the Online Certificate Status Manager's certificate database. However, if a non-security domain CA is selected, then the OCSP service must be manually configured after the Online Certificate Status Manager is configured.

NOTE

Not every CA within the security domain to which the OCSP Manager belongs is automatically trusted by the OCSP Manager when it is configured. Every CA in the certificate chain of the CA configured in the CA panel is trusted automatically by the OCSP Manager. Other CAs within the security domain but not in the certificate chain must be trusted manually.

To set up the Online Certificate Status Manager for a Certificate Manager outside the security domain:

1. Configure the CRLs for every CA that will publish to an OCSP responder.

2. Enable publishing, set up a publisher, and set publishing rules in every CA that the OCSP service will handle (Chapter 9, *Publishing Certificates and CRLs*). This is not necessary if the Certificate Managers publish to an LDAP directory and the Online Certificated Status Manager is set up to read from that directory.

3. The certificate profiles must be configured to include the Authority Information Access extension, pointing to the location at which the Certificate Manager listens for OCSP service requests (Section 7.6.4, "Enabling the Certificate Manager's Internal OCSP Service").

4. Configure the OCSP Responder.

    - Configure the Revocation Info store (Section 7.6.2.2, "Configure the Revocation Info Stores: Internal Database" and Section 7.6.2.3, "Configure the Revocation Info Stores: LDAP Directory").

    - Identify every publishing Certificate Manager to the OCSP responder (Section 7.6.2, "Identifying the CA to the OCSP Responder").

    - If necessary, configure the trust settings for the CA which signed the OCSP signing certificate (Section 17.7, "Changing the Trust Settings of a CA Certificate").

5. Restart both subsystems after configuring them.

6. Verify that the CA is properly connected to the OCSP responder (Section 7.6.2.1, "Verify Certificate Manager and Online Certificate Status Manager Connection").

7.6.2. Identifying the CA to the OCSP Responder

Before a CA is configured to publish CRLs to the Online Certificate Status Manager, the CA must be identified to the Online Certificate Status Manager by storing the CA signing certificate in the internal database of the Online Certificate Status Manager. The Certificate Manager signs CRLs with the key pair associated with this certificate; the Online Certificate Status Manager verifies the signature against the stored certificate.

NOTE

If a CA within the security domain is selected when the Online Certificate Status Manager is configured, there is no extra step required to configure the Online Certificate Status Manager to recognize the CA; the CA signing certificate is automatically added and trusted in the Online Certificate Status Manager's certificate database. However, if a non-security domain CA is selected, then the CA signing certificate must be manually added to the certificate database after the Online Certificate Status Manager is configured.

It is not necessary to import the certificate chain for a CA which will publish its CRL to the Online Certificate Status Manager. The only time a certificate chain is needed for the OCSP service is if the CA connects to the Online Certificate Status Manager through SSL/TLS authentication when it publishes its CRL. Otherwise, the Online Certificate Status Manager does not need to have the complete certificate chain.

However, the Online Certificate Status Manager must have the certificate which signed the CRL, either a CA signing certificate or a separate CRL signing certificate, in its certificate database. The OCSP service verifies the CRL by comparing the certificate which signed the CRL against the certificates in its

database, not against a certificate chain. If both a root CA and one of its subordinate CAs publish CRLs to the Online Certificate Status Manager, the Online Certificate Status Manager needs the CA signing certificate of both CAs.

To import the CA or CRL signing certificate which is used to sign the certificates the CA is publishing to the Online Certificate Status Manager, do the following:

1. Get the Certificate Manager's base-64 CA signing certificate from the end-entities page of the CA.
2. Open the Online Certificate Status Manager agent page. The URL has the format **https://***hostname:SSLport*/**ocsp**/**agent**/**ocsp**.
3. In the left frame, click **Add Certificate Authority**.
4. In the form, paste the encoded CA signing certificate inside the text area labeled **Base 64 encoded certificate (including the header and footer)**.
5. To verify that the certificate is added successfully, in the left frame, click **List Certificate Authorities**.

The resulting form should show information about the new CA. The **This Update**, **Next Update**, and **Requests Served Since Startup** fields should show a value of zero (0).

7.6.2.1. Verify Certificate Manager and Online Certificate Status Manager Connection

When the Certificate Manager is restarted, it tries to connect to the Online Certificate Status Manager's SSL/TLS port. To verify that the Certificate Manager did indeed communicate with the Online Certificate Status Manager, check the **This Update** and **Next Update** fields, which should be updated with the appropriate timestamps of the CA's last communication with the Online Certificate Status Manager. The **Requests Served Since Startup** field should still show a value of zero (0) since no client has tried to query the OCSP service for certificate revocation status.

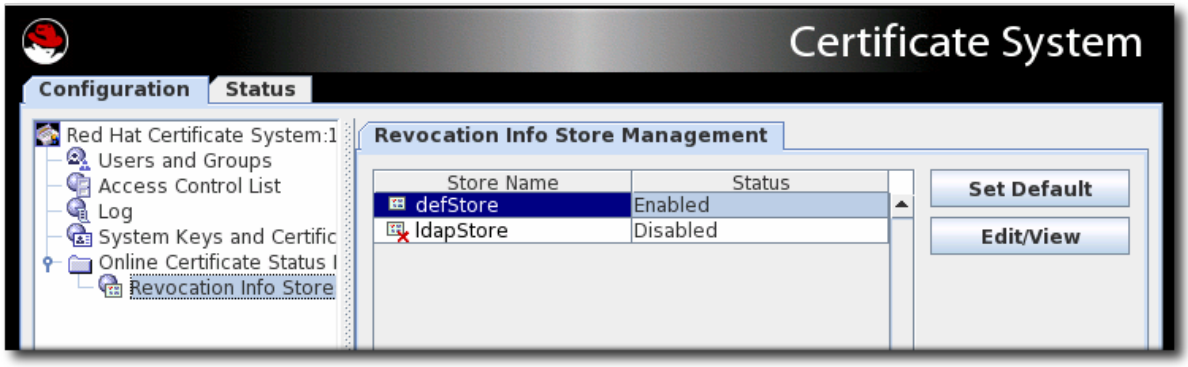
7.6.2.2. Configure the Revocation Info Stores: Internal Database

The Online Certificate Status Manager stores each Certificate Manager's CRL in its internal database and uses it as the CRL store for verifying the revocation status of certificates. To change the configuration that the Online Certificate Status Manager uses for storing the CRLs in its internal database:

1. Open the Online Certificate Status Manager Console.

```
pkiconsole https://server.example.com:8443/ocsp
```

2. In the **Configuration** tab, select **Online Certificate Status Manager**, and then select **Revocation Info Stores**.



The right pane shows the two repositories the Online Certificate Status Manager can use; by default, it uses the CRL in its internal database.

3. Select the **defStore**, and click **Edit/View**.

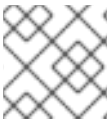
4. Edit the **defStore** values.

- **notFoundAsGood.** Sets the OCSP service to return an OCSP response of GOOD if the certificate in question cannot be found in any of the CRLs. If this is not selected, the response is UNKNOWN, which, when encountered by a client, results in an error message.

- **byName.** The OCSP Responder only supports the basic response type, which includes the ID of the OCSP Responder making the response. The ResponderID field within the basic

response type is determined by the value of the ***ocsp.store.defStore.byName*** parameter. If ***byName*** parameter is true or is missing, the OCSP authority signing certificate subject name is used as the ResponderID field of the OCSP response. If ***byName*** parameter is false, the OCSP authority signing certificate key hash will be the ResponderID field of the OCSP response.

- **includeNextUpdate.** Includes the timestamp of the next CRL update time.

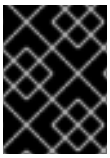


NOTE

pkiconsole is being deprecated.

7.6.2.3. Configure the Revocation Info Stores: LDAP Directory

Although the OCSP Manager stores the CA CRLs in its internal database by default, it can be configured to use a CRL published to an LDAP directory instead.



IMPORTANT

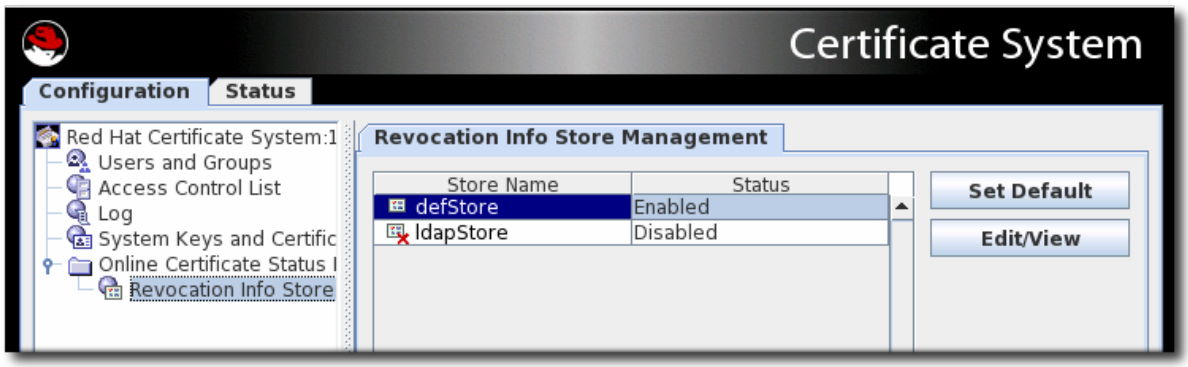
If the **ldapStore** method is enabled, the OCSP user interface does not check the certificate status.

To configure the Online Certificate Status Manager to use an LDAP directory:

1. Open the Online Certificate Status Manager Console.

```
pkiconsole https://server.example.com:8443/ocsp
```

2. In the **Configuration** tab, select **Online Certificate Status Manager**, and then select **Revocation Info Stores**.

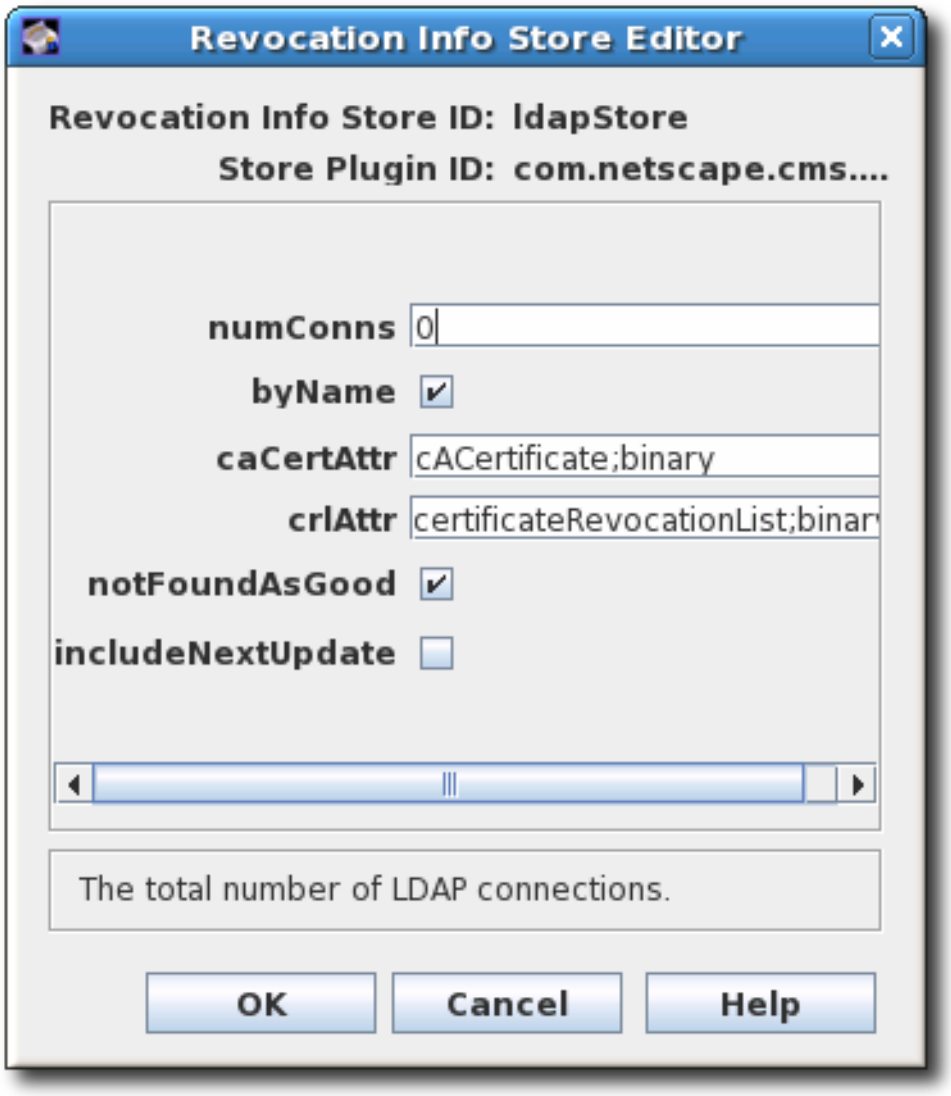


The right pane shows the two repositories the Online Certificate Status Manager can use; by default, it uses the CRL in its internal database.

3. To use the CRLs in LDAP directories, click **Set Default** to enable the **ldapStore** option.

4. Select **ldapStore**, and click **Edit/View**.

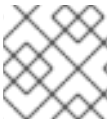
5. Set the **ldapStore** parameters.



- **numConns.** The total number of LDAP directories the OCSP service should check. By default, this is set to 0. Setting this value shows the corresponding number of **host**, **port**, **baseDN**, and **refreshInSec** fields.
- **host.** The fully-qualified DNS hostname of the LDAP directory.
- **port.** The non-SSL/TLS port of the LDAP directory.
- **baseDN.** The DN to start searching for the CRL. For example, **O=example.com**.
- **refreshInSec.** How often the connection is refreshed. The default is 86400 seconds (daily).
- **caCertAttr.** Leave the default value, **cACertificate;binary**, as it is. It is the attribute to which the Certificate Manager publishes its CA signing certificate.
- **crlAttr.** Leave the default value, **certificateRevocationList;binary**, as it is. It is the attribute to which the Certificate Manager publishes CRLs.
- **notFoundAsGood.** Sets the OCSP service to return an OCSP response of GOOD if the certificate in question cannot be found in any of the CRLs. If this is not selected, the response is UNKNOWN, which, when encountered by a client, results in an error message.
- **byName.** The OCSP Responder only supports the basic response type, which includes the ID of the OCSP Responder making the response. The ResponderID field within the basic response type is determined by the value of the ***ocsp.store.defStore.byName*** parameter.

If ***byName*** parameter is true or is missing, the OCSP authority signing certificate subject name is used as the ResponderID field of the OCSP response. If ***byName*** parameter is false, the OCSP authority signing certificate key hash will be the ResponderID field of the OCSP response.

- **includeNextUpdate.** The Online Certificate Status Manager can include the timestamp of the next CRL update time.



NOTE

pkiconsole is being deprecated.

7.6.2.4. Testing the OCSP Service Setup

Test whether the Certificate Manager can service OCSP requests properly by doing the following:

1. Turn on revocation checking in the browser or client.
2. Request a certificate from the CA that has been enabled for OCSP services.
3. Approve the request.
4. Download the certificate to the browser or client.
5. Make sure the CA is trusted by the browser or client.
6. Check the status of Certificate Manager's internal OCSP service.

   Open the CA agent services page, and select the **OCSP Services** link.
7. Test the independent Online Certificate Status Manager subsystem.

   Open the Online Certificate Status Manager agent services page, and click the **List Certificate Authorities** link.

   The page should show information about the Certificate Manager configured to publish CRLs to the Online Certificate Status Manager. The page also summarizes the Online Certificate Status Manager's activity since it was last started.
8. Revoke the certificate.
9. Verify the certificate in the browser or client. The server should return that the certificate has been revoked.
10. Check the Certificate Manager's OCSP-service status again to verify that these things happened:
    - The browser sent an OCSP query to the Certificate Manager.
    - The Certificate Manager sent an OCSP response to the browser.
    - The browser used that response to validate the certificate and returned its status, that the certificate could not be verified.
11. Check the independent OCSP service subsystem again to verify that these things happened:
    - The Certificate Manager published the CRL to the Online Certificate Status Manager.

- The browser sent an OCSP response to the Online Certificate Status Manager.
- The Online Certificate Status Manager sent an OCSP response to the browser.
- The browser used that response to validate the certificate and returned its status, that the certificate could not be verified.

7.6.3. Setting the Response for Bad Serial Numbers

OCSP responders check the revocation status and expiration date of a certificate before determining whether the certificate is valid; by default, the OCSP does not validate other information on the certificate.

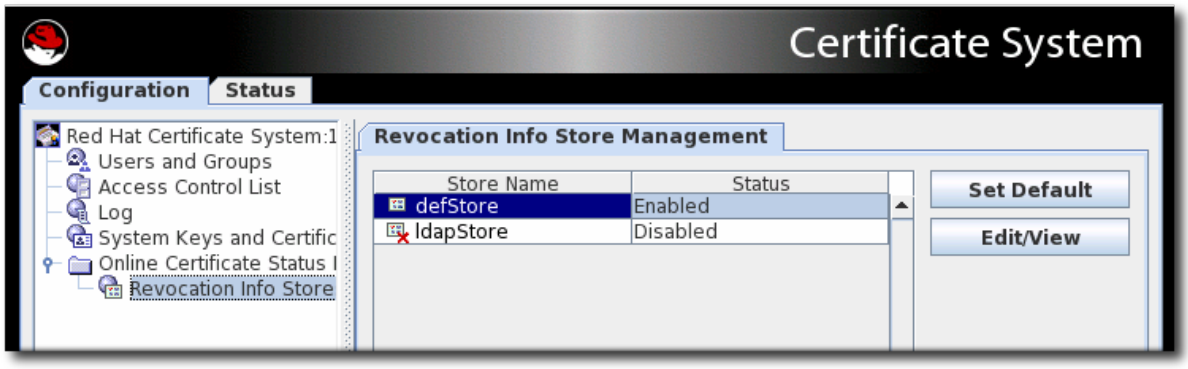
The ***notFoundAsGood*** parameter sets how the OCSP handles a certificate with an invalid serial number. This parameter is enabled by default, which means that if a certificate is present with a bad serial number but the certificate is otherwise valid, the OCSP returns a status of **GOOD** for the certificate.

To have the OCSP check and reject certificates based on bad serial numbers as well as revocation status, change the ***notFoundAsGood*** setting. In that case, the OCSP returns a status of **UNKNOWN** with a certificate with a bad serial number. The client interprets that as an error and can respond accordingly.

1. Open the Online Certificate Status Manager Console.

```
pkiconsole https://server.example.com:8443/ocsp
```

2. In the **Configuration** tab, select **Online Certificate Status Manager**, and then select **Revocation Info Stores**.



3. Select the **defStore**, and click **Edit/View**.
4. Edit the ***notFoundAsGood*** value. Selecting the checkbox means that the OCSP returns a value of **GOOD** even if the serial number on the certificate is bad. Unselecting the checkbox means that the OCSP sends a value of **UNKNOWN**, which the client can intrepret as an error.

5. Restart the OCSP Manager.

```
]# pki-server restart instance-name
```

NOTE

pkiconsole is being deprecated.

7.6.4. Enabling the Certificate Manager's Internal OCSP Service

The Certificate Manager has a built-in OCSP service, which can be used by OCSP-compliant clients to query the Certificate Manager directly about the revocation status of the certificate. When the Certificate Manager is installed, an OCSP signing certificate is issued and the OCSP service is turned on by default. This OCSP signing certificate is used to sign all responses to OCSP service requests. Since the internal OCSP service checks the status of certificates stored in the Certificate Manager's internal database, publishing does not have to be configured to use this service.

Clients can query the OCSP service through the non-SSL/TLS end-entity port of the Certificate Manager. When queried for the revocation status of a certificate, the Certificate Manager searches its internal database for the certificate, checks its status, and responds to the client. Since the Certificate Manager has real-time status of all certificates it has issued, this method of revocation checking is the most accurate.

Every CA's built-in OCSP service is turned on at installation. However, to use this service, the CA needs to issue certificates with the Authority Information Access extension.

1. Go to the CA's end-entities page. For example:

```
https://server.example.com:8443/ca/ee/ca
```

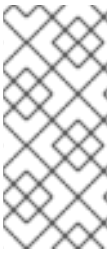
2. Find the CA signing certificate.

3. Look for the Authority Info Access extension in the certificate, and note the **Location URIName** value, such as **https://server.example.com:8443/ca/ocsp**.

4. Update the enrollment profiles to enable the Authority Information Access extension, and set the **Location** parameter to the Certificate Manager's URI. For information on editing the certificate profiles, see Section 3.2, “Setting up Certificate Profiles”.

5. Restart the CA instance.

```
]# pki-server restart instance-name
```



NOTE

To disable the Certificate Manager's internal OCSP service, edit the CA's **CS.cfg** file and change the value of the **ca.ocsp** parameter to **false**.

```
ca.ocsp=false
```

7.6.5. Submitting OCSP Requests Using the OCSPClient program

The **OCSPClient** program can be used for performing OCSP requests. For example:

```
]# OCSPClient -h server.example.com -p 8080 -d /etc/pki/pki-tomcat/alias -c "caSigningCert cert-pki-ca" --serial 2
CertID.serialNumber=2
CertStatus=Good
```

The **OCSPClient** command can be used with the following command-line options:

Table 7.1. Available OCSPClient Options

Option	Description
-d *database*	Security database location (default: current directory)
-h *hostname*	OCSP server hostname (default: example.com)
-p *port*	OCSP server port number (default: 8080)
-t *path*	OCSP service path (default: /ocsp/ee/ocsp)
-c *nickname*	CA certificate nickname (defaut: CA Signing Certificate)

Option	Description
-n *times*	Number of submissions (default: 1)
--serial *serial_number*	Serial number of certificate to be checked
--input *input_file*	Input file containing DER-encoded OCSP request
--output *output_file*	Output file to store DER-encoded OCSP response
-v, --verbose	Run in verbose mode
--help	Show help message

7.6.6. Submitting OCSP Requests Using the GET Method

OCSP requests which are smaller than 255 bytes can be submitted to the Online Certificate Status Manager using a GET method, as described in RFC 6960. To submit OCSP requests over GET:

1. Generate an OCSP request for the certificate the status of which is being queried. For example:

```
]# openssl ocsp -CAfile ca.pem -issuer issuer.pem -serial serial_number -reqout - | base64

MEIwQDA+MDwwOjAJBgUrDgMCGgUABBT4cyABkyiCIhU4JpmIBewdDnn8ZgQUbyBZ44kgy
35o7xW5BMzM8FTvyTwCAQE=
```

2. Paste the URL in the address bar of a web browser to return the status information. The browser must be able to handle OCSP requests.

```
https://server.example.com:8443/ocsp/ee/ocsp/MEIwQDA+MDwwOjAJBgUrDgMCGgUABBT4
cyABkyiCIhU4JpmIBewdDnn8ZgQUbyBZ44kgy35o7xW5BMzM8FTvyTwCAQE=
```

3. The OCSP Manager responds with the certificate status which the browser can interpret. The possible statuses are GOOD, REVOKED, and UNKNOWN.

Alternatively, run the OCSP from the command line by using a tool such as **curl** to send the request and **openssl** to parse the response. For example:

1. Generate an OCSP request for the certificate the status of which is being queried. For example:

```
]# openssl ocsp -CAfile ca.pem -issuer issuer.pem -serial serial_number -reqout - | base64

MEIwQDA+MDwwOjAJBgUrDgMCGgUABBT4cyABkyiCIhU4JpmIBewdDnn8ZgQUbyBZ44kgy
35o7xW5BMzM8FTvyTwCAQE=
```

2. Connect to the OCSP Manager using **curl** to send the OCSP request.

```
curl
https://server.example.com:8443/ocsp/ee/ocsp/MEIwQDA+MDwwOjAJBgUrDgMCGgUABBT4
cyABkyiCIhU4JpmIBewdDnn8ZgQUbyBZ44kgy35o7xW5BMzM8FTvyTwCAQE= >
```

```
ocspresp.der
```

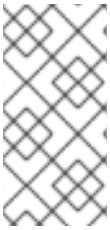
3. Parse the response using **openssl**:

```
openssl ocsp -respin ocspresp.der -resp_text
```

For certificates issued by a 7.1 CA with the Authority Information Access extension to be sent to the OCSP with the GET method, a redirect needs to be created to forward the requests to the appropriate URL, as described in Section 7.6.7, "Setting up a Redirect for Certificates Issued in Certificate System 7.1 and Earlier".

7.6.7. Setting up a Redirect for Certificates Issued in Certificate System 7.1 and Earlier

The location for the OCSP user pages, specified in the URL with the file root **/ocsp/ee/ocsp/**, is different in Certificate System 10 or Certificate System 8.1 than the location in Certificate System 7.1, which was simply **/ocsp/**. In order for certificates issued by a 7.1 or earlier CA with the Authority Information Access extension to be sent to the OCSP, create a redirect to forward the requests to the appropriate URL.



NOTE

Setting the redirect is only required to manage certificates issued by a 7.1 or earlier CA with the Authority Information Access extension. If the certificates are issued by a later version Certificate Manager or do not contain the Authority Information Access extension, then this configuration is not necessary.

1. Stop the OCSP Responder.

```
]# pki-server stop instance-name
```

2. Change to the OCSP's end user web applications directory. For example:

```
]# cd /var/lib/pki-ocsp/webapps/ocsp
```

3. Change to the **ROOT/WEB-INF/** directory in the **ROOT** folder of the OCSP's web applications directory. For example:

```
]# cd /var/lib/pki-ocsp/webapps/ocsp/ROOT/WEB-INF/
```

4. Create and open the **lib/** directory in the **ROOT** folder of the OCSP's web applications directory.

```
]# mkdir lib
]# cd lib/
```

5. Create a symlink that links back to the **/usr/share/java/pki/cms.jar** JAR file. For example:

```
]# ln -s /usr/share/java/pki/cms.jar cms.jar
```

6. Move up to the main web application directory. For example:

```
]# cd /var/lib/pki-ocsp/webapps/ocsp/
```

7. Rename the current instance (**ocsp**) directory. For example:

```
]# mv /var/lib/pki-ocsp/webapps/ocsp/ocsp /var/lib/pki-ocsp/webapps/ocsp/ocsp2
```

8. Change to the **WEB-INF/** directory in the original **ocsp/** directory. For example:

```
]# cd  /var/lib/pki-ocsp/webapps/ocsp/ocsp/WEB-INF
```

9. In the original **ocsp/WEB-INF/** directory, edit the **web.xml** file and add lines mapping between the **eeocspAddCRL** and **csadmin-wizard** servlets.

```
 <servlet-mapping>
   <servlet-name>  ocspOCSP  </servlet-name>
   <url-pattern>   /ee/ocsp/*  </url-pattern>
 </servlet-mapping>
```

10. Create and install the **web.xml** file in the **ROOT** directory. For example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>

 <display-name>Welcome to Tomcat</display-name>
 <description>
   Welcome to Tomcat
 </description>

 <servlet>
  <servlet-name>ocspProxy</servlet-name>
  <servlet-class>com.netscape.cms.servlet.base.ProxyServlet</servlet-class>
  <init-param>
   <param-name>destContext</param-name>
   <param-value>/ocsp2</param-value>
  </init-param>
  <init-param>
   <param-name>destServlet</param-name>
   <param-value>/ee/ocsp</param-value>
  </init-param>
 </servlet>

 <servlet>
  <servlet-name>ocspOther</servlet-name>
  <servlet-class>com.netscape.cms.servlet.base.ProxyServlet</servlet-class>
  <init-param>
   <param-name>destContext</param-name>
   <param-value>/ocsp2</param-value>
  </init-param>
  <init-param>
   <param-name>srcContext</param-name>
   <param-value>/ocsp</param-value>
  </init-param>
  <init-param>
   <param-name>destServlet</param-name>
```

```
      <param-value></param-value>
    </init-param>
    <init-param>
      <param-name>matchURIStrings</param-name>

<param-value>/ocsp/registry,/ocsp/acl,/ocsp/jobsScheduler,/ocsp/ug,/ocsp/server,/ocsp/log,
          /ocsp/auths,/ocsp/start,/ocsp/ocsp,/ocsp/services,/ocsp/agent,/ocsp/ee,
          /ocsp/admin</param-value>
    </init-param>
    <init-param>
      <param-name>destServletOnNoMatch</param-name>
      <param-value>/ee/ocsp</param-value>
    </init-param>
    <init-param>
      <param-name>appendPathInfoOnNoMatch</param-name>
      <param-value>/ocsp</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>ocspProxy</servlet-name>
    <url-pattern>/ocsp</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ocspOther</servlet-name>
    <url-pattern>/ocsp/*</url-pattern>
  </servlet-mapping>

</web-app>
```

11. Edit the **/var/lib/pki-ocsp/conf/context.xml** file, changing the following line:

```
<Context>
 to
<Context crossContext="true" >
```

12. Edit the **/var/lib/pki-ocsp/webapps/ocsp/ocsp2/services.template** file and change the following line:

```
result.recordSet[i].uri);
 to
result.recordSet[i].uri + "/");
```

13. Start the OCSP instance.

```
]# pki-server start instance-name
```

CHAPTER 8. MANAGING PKI ACME RESPONDER

This chapter describes how to manage *PKI ACME Responder*.

For information on how to set up PKI ACME Responder, see the *Setting up PKI ACME Responder* chapter in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

8.1. ENABLING/DISABLING ACME SERVICES

Users that belong to the *Administrators* group can enable or disable services in the ACME responder. The user can authenticate either with basic authentication or client certificate authentication.

- To enable or disable ACME services with basic authentication, specify the username and password:

```
$ pki -u <username> -p <password> acme-<enable/disable>
```

- To enable or disable ACME services with client certificate authentication, specify the certificate nickname and NSS database password:

```
$ pki -n <nickname> -c <password> acme-<enable/disable>
```

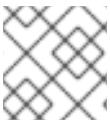
8.2. CHECKING THE STATUS OF PKI ACME RESPONDER

- To check the status of the ACME responder, run the following command:

```
$ pki acme-info
Status: Available
Terms of Service: https://www.example.com/acme/tos.pdf
Website: https://www.example.com
CAA Identities: example.com
External Account Required:false
```

If the services are disabled, the command will show the following result:

```
$ pki acme-info
Status: Unavailable
```



NOTE

The actual output depends on what is configured in the **metadata.conf** configuration file.

PART III. ADDITIONAL CONFIGURATION TO MANAGE CA SERVICES

CHAPTER 9. PUBLISHING CERTIFICATES AND CRLS

Red Hat Certificate System includes a customizable publishing framework for the Certificate Manager, enabling certificate authorities to publish certificates, certificate revocation lists (CRLs), and other certificate-related objects to any of the supported repositories: an LDAP-compliant directory, a flat file, and an online validation authority. This chapter explains how to configure a Certificate Manager to publish certificates and CRLs to a file, to a directory, and to the Online Certificate Status Manager.

The general process to configure publishing is as follows:

1. Configure publishing to a file, LDAP directory, or OCSP responder.

   There can be a single publisher or multiple publishers, depending on how many locations will be used. The locations can be split by certificates and CRLs or narrower definitions, such as certificate type. Rules determine which type to publish and to what location by being associated with the publisher.

2. Set rules to determine what certificates are published to the locations. Any rule which a certificate or CRL matches is activated, so the same certificate can be published to a file and to an LDAP directory by matching a file-based rule and matching a directory-based rule.

   Rules can be set for each object type: CA certificates, CRLs, user certificates, and cross-pair certificates. Disable all rules that will not be used.

3. Configure CRLs. CRLs must be configured before they can be published. See Chapter 7, *Revoking Certificates and Issuing CRLs*.

4. Enable publishing after setting up publishers, mappers, and rules. Once publishing is enabled, the server starts publishing immediately. If the publishers, mappers, and rules are not completely configured, publishing may not work correctly or at all.

9.1. ABOUT PUBLISHING

The Certificate System is capable of publishing certificates to a file or an LDAP directory and of publishing CRLs to a file, an LDAP directory, or to an OCSP responder.

For additional flexibility, specific types of certificates or CRLs can be published to a single format or all three. For example, CA certificates can be published only to a directory and not to a file, and user certificates can be published to both a file and a directory.

NOTE

An OCSP responder only provides information about CRLs; certificates are not published to an OCSP responder.

Different publishing locations can be set for certificates files and CRL files, as well as different publishing locations for different types of certificates files or different types of CRL files.

Similarly, different types of certificates and different types of CRLs can be published to different places in a directory. For example, certificates for users from the West Coast division of a company can be published in one branch of the directory, while certificates for users in the East Coast division can be published to another branch in the directory.

When publishing is enabled, every time a certificate or a CRL is issued, updated, or revoked, the publishing system is invoked. The certificate or CRL is evaluated by the rules to see if it matches the

type and predicate set in the rule. The type specifies if the object is a CRL, CA certificate, or any other certificate. The predicate sets more criteria for the type of object being evaluated. For example, it can specify user certificates, or it can specify West Coast user certificates. To use predicates, a value needs to be entered in the predicate field of the publishing rule, and a corresponding value (although formatted somewhat differently) needs to be contained in the certificate or certificate request to match. The value in the certificate or certificate request may be derived from information in the certificate, such as the type of certificate, or may be derived from a hidden value that is placed in the request form. If no predicate is set, all certificates of that type are considered to match. For example, all CRLs match the rule if **CRL** is set as the type.

Every rule that is matched publishes the certificate or CRL according to the method and location specified in that rule. A given certificate or CRL can match no rules, one rule, more than one rule, or all rules. The publishing system attempts to match every certificate and CRL issued against all rules.

When a rule is matched, the certificate or CRL is published according to the method and location specified in the publisher associated with that rule. For example, if a rule matches all certificates issued to users, and the rule has a publisher that publishes to a file in the location **/etc/CS/certificates**, the certificate is published as a file to that location. If another rule matches all certificates issued to users, and the rule has a publisher that publishes to the LDAP attribute **userCertificate;binary** attribute, the certificate is published to the directory specified when LDAP publishing was enabled in this attribute in the user's entry.

For rules that specify to publish to a file, a new file is created when either a certificate or a CRL is issued in the stipulated directory.

For rules that specify to publish to an LDAP directory, the certificate or CRL is published to the entry specified in the directory, in the attribute specified. The CA overwrites the values for any published certificate or CRL attribute with any subsequent certificate or CRL. Simply put, any existing certificate or CRL that is already published is replaced by the next certificate or CRL.

For rules that specify to publish to an Online Certificate Status Manager, a CRL is published to this manager. Certificates are not published to an Online Certificate Status Manager.

For LDAP publishing, the location of the user's entry needs to be determined. Mappers are used to determine the entry to which to publish. The mappers can contain an exact DN for the entry, some variable that associates information that can be gotten from the certificate to create the DN, or enough information to search the directory for a unique attribute or set of attributes in the entry to ascertain the correct DN for the entry.

When a certificate is revoked, the server uses the publishing rules to locate and delete the corresponding certificate from the LDAP directory or from the filesystem.

When a certificate expires, the server can remove that certificate from the configured directory. The server does not do this automatically; the server must be configured to run the appropriate job. For details, see Chapter 13, *Setting Automated Jobs*.

Setting up publishing involves configuring publishers, mappers, and rules.

9.1.1. Publishers

Publishers specify the location to which certificates and CRLs are published. When publishing to a file, publishers specify the filesystem publishing directory. When publishing to an LDAP directory, publishers specify the attribute in the directory that stores the certificate or CRL; a mapper is used to determine the DN of the entry. For every DN, a different formula is set for deriving that DN. The location of the LDAP directory is specified when LDAP publishing is enabled. When publishing a CRL to an OCSP responder, publishers specify the hostname and URI of the Online Certificate Status Manager.

9.1.2. Mappers

Mappers are only used in LDAP publishing. Mappers construct the DN for an entry based on information from the certificate or the certificate request. The server has information from the subject name of the certificate and the certificate request and needs to know how to use this information to create a DN for that entry. The mapper provides a formula for converting the information available either to a DN or to some unique information that can be searched in the directory to obtain a DN for the entry.

9.1.3. Rules

Rules for file, LDAP, and OCSP publishing tell the server whether and how a certificate or CRL is to be published. A rule first defines what is to be published, a certificate or CRL matching certain characteristics, by setting a type and predicate for the rule. A rule then specifies the publishing method and location by being associated with a publisher and, for LDAP publishing, with a mapper.

Rules can be as simple or complex as necessary for the PKI deployment and are flexible enough to accommodate different scenarios.

9.1.4. Publishing to Files

The server can publish certificates and CRLs to flat files, which can then be imported into any repository, such as a relational database. When the server is configured to publish certificates and CRLs to file, the published files are DER-encoded binary blobs, base-64 encoded text blobs, or both.

- For each certificate the server issues, it creates a file that contains the certificate in either DER-encoded or base-64 encoded format. Each file is named either **cert-***serial_number***.der** or **cert-***serial_number***.b64**. The *serial_number* is the serial number of the certificate contained in the file. For example, the filename for a DER-encoded certificate with the serial number **1234** is **cert-1234.der**.

- Every time the server generates a CRL, it creates a file that contains the new CRL in either DER-encoded or base-64 encoded format. Each file is named either *issuing_point_name*-*this_update***.der** or *issuing_point_name*-*this_update***.b64**, depending on the format. The *issuing_point_name* identifies the CRL issuing point which published the CRL, and *this_update* specifies the value derived from the time-dependent update value for the CRL contained in the file. For example, the filename for a DER-encoded CRL with the value **This Update: Friday January 28 15:36:00 PST 2020**, is **MasterCRL-20200128-153600.der**.

9.1.5. OCSP Publishing

There are two forms of Certificate System OCSP services, an internal service for the Certificate Manager and the Online Certificate Status Manager. The internal service checks the internal database of the Certificate Manager to report on the status of a certificate. The internal service is not set for publishing; it uses the certificates stored in its internal database to determine the status of a certificate. The Online Certificate Status Manager checks CRLs sent to it by Certificate Manager. A publisher is set for each location a CRL is sent and one rule for each type of CRL sent.

For detailed information on both OCSP services, see Section 7.6, “Using the Online Certificate Status Protocol (OCSP) Responder”.

9.1.6. LDAP Publishing

In *LDAP publishing*, the server publishes the certificates, CRLs, and other certificate-related objects to a directory using LDAP or LDAPS. The branch of the directory to which it publishes is called the *publishing directory*.

- For each certificate the server issues, it creates a blob that contains the certificate in its DER-encoded format in the specified attribute of the user's entry. The certificate is published as a DER encoded binary blob.

- Every time the server generates a CRL, it creates a blob that contains the new CRL in its DER-encoded format in the specified attribute of the entry for the CA.

The server can publish certificates and CRLs to an LDAP-compliant directory using the LDAP protocol or LDAP over SSL (LDAPS) protocol, and applications can retrieve the certificates and CRLs over HTTP. Support for retrieving certificates and CRLs over HTTP enables some browsers to import the latest CRL automatically from the directory that receives regular updates from the server. The browser can then use the CRL to check all certificates automatically to ensure that they have not been revoked.

For LDAP publishing to work, the user entry must be present in the LDAP directory.

If the server and publishing directory become out of sync for some reason, privileged users (administrators and agents) can also manually initiate the publishing process. For instructions, see Section 9.12.2, “Manually Updating the CRL in the Directory” .

9.2. CONFIGURING PUBLISHING TO A FILE

The general process to configure publishing involves setting up a publisher to publish the certificates or CRLs to the specific location. There can be a single publisher or multiple publishers, depending on how many locations will be used. The locations can be split by certificates and CRLs or finer definitions, such as certificate type. Rules determine which type to publish and to what location by being associated with the publisher.

Publishing to file simply publishes the CRLs or certificates to text files on a given host.

Publishers must be created and configured for each publishing location; publishers are not automatically created for publishing to a file. To publish all files to a single location, create one publisher. To publish to different locations, create a publisher for each location. A location can either contain an object type, like user certificates, or a subset of an object type, like West Coast user certificates.

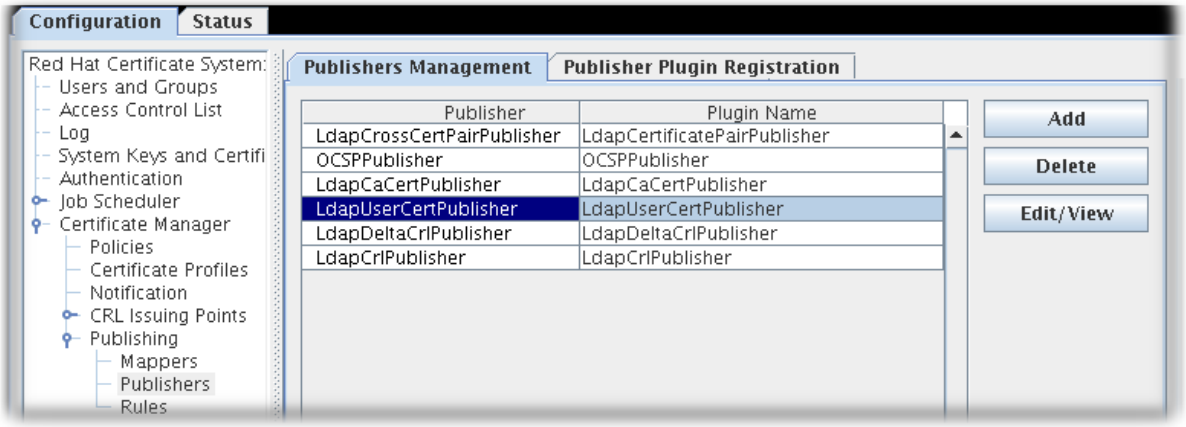
To create publishers for publishing to files:

1. Log into the Certificate Manager Console.

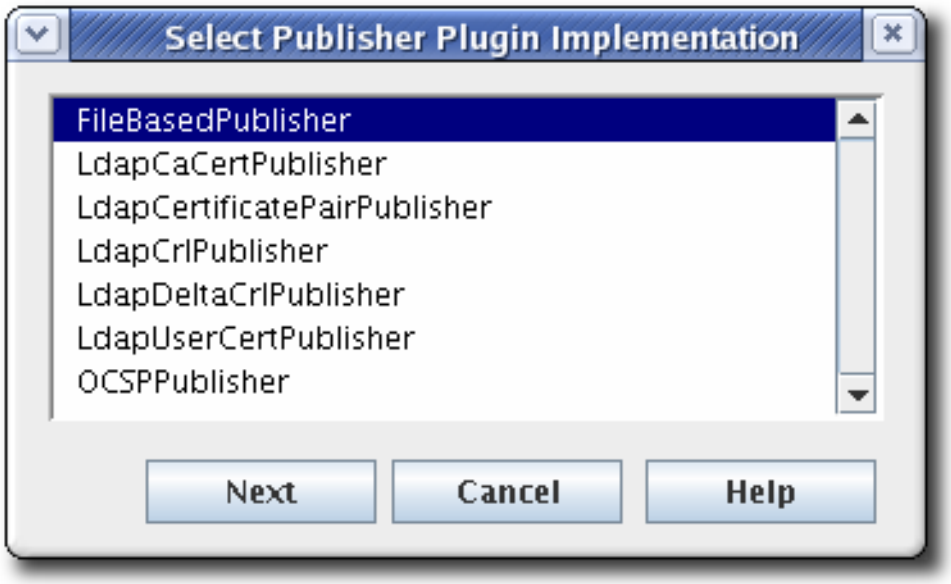
   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Publishers**.

   The **Publishers Management** tab, which lists configured publisher instances, opens on the right.

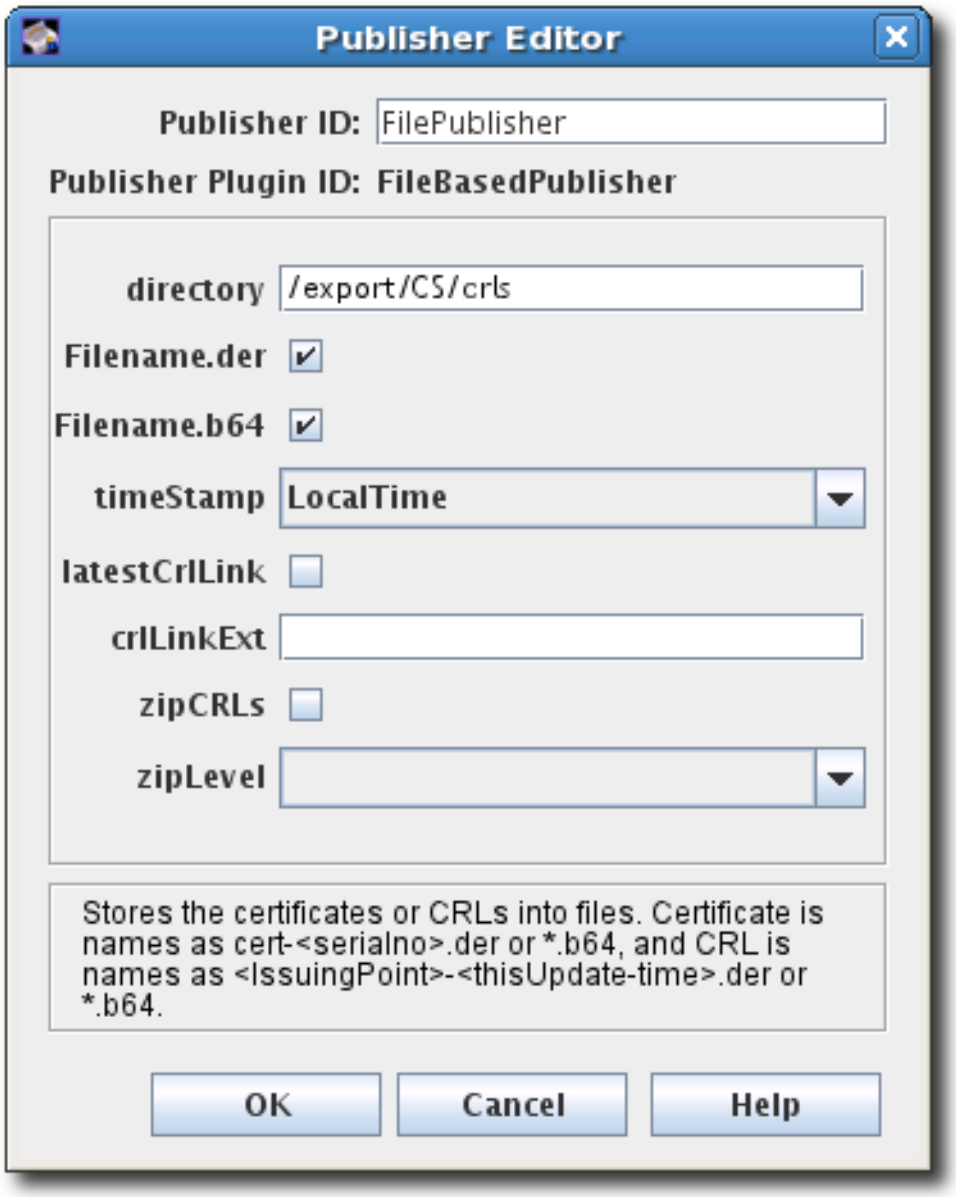


3. Click **Add** to open the **Select Publisher Plug-in Implementation** window, which lists registered publisher modules.



4. Select the **FileBasedPublisher** module, then open the editor window.

   This is the module that enables the Certificate Manager to publish certificates and CRLs to files.



5. Configure the information for publishing the certificate:

    - The publisher ID, an alphanumeric string with no spaces like **PublishCertsToFile**
    - The path to the directory in which the Certificate Manager should publish the files. The path can be an absolute path or can be relative to the Certificate System instance directory. For example, **/export/CS/certificates**.
    - The file type to publish, by selecting the checkboxes for DER-encoded files, base-64 encoded files, or both.
    - For CRLs, the format of the timestamp. Published certificates include serial numbers in their file names, while CRLs use timestamps.
    - For CRLs, whether to generate a link in the file to go to the latest CRL. If enabled, the link assumes that the name of the CRL issuing point to use with the extension will be supplied in the **crlLinkExt** field.
    - For CRLs, whether to compress (zip) CRLs and the compression level to use.

After configuring the publisher, configure the rules for the published certificates and CRLs, as described in Section 9.5, “Creating Rules”.

NOTE

pkiconsole is being deprecated.

9.3. CONFIGURING PUBLISHING TO AN OCSP

The general process to configure publishing involves setting up a publisher to publish the certificates or CRLs to the specific location. There can be a single publisher or multiple publishers, depending on how many locations will be used. The locations can be split by certificates and CRLs or finer definitions, such as certificate type. Rules determine which type to publish and to what location by being associated with the publisher.

Publishing to an OCSP Manager is a way to publish CRLs to a specific location for client verification.

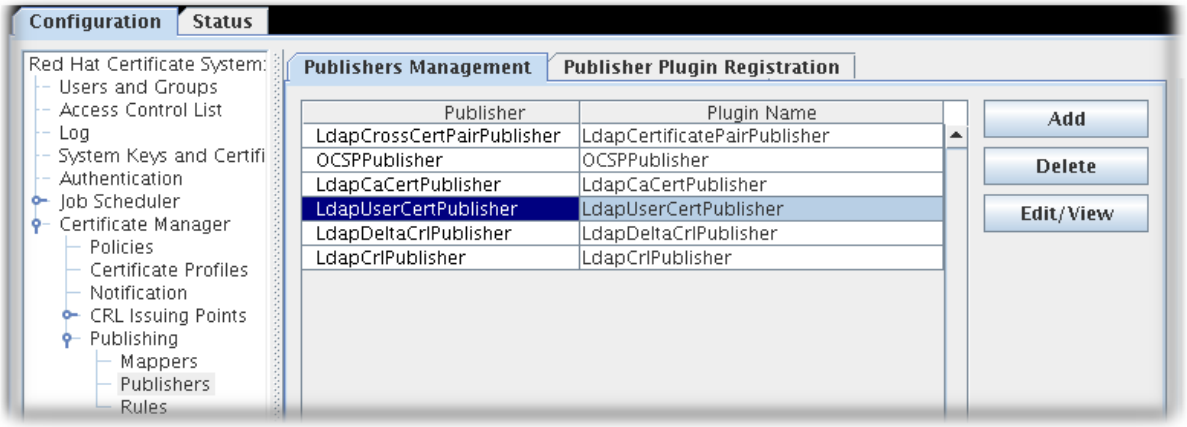
A publisher must be created and configured for each publishing location; publishers are not automatically created for publishing to the OCSP responder. Create a single publisher to publish everything to s single location, or create a publisher for every location to which CRLs will be published. Each location can contain a different kind of CRL.

9.3.1. Enabling Publishing to an OCSP with Client Authentication

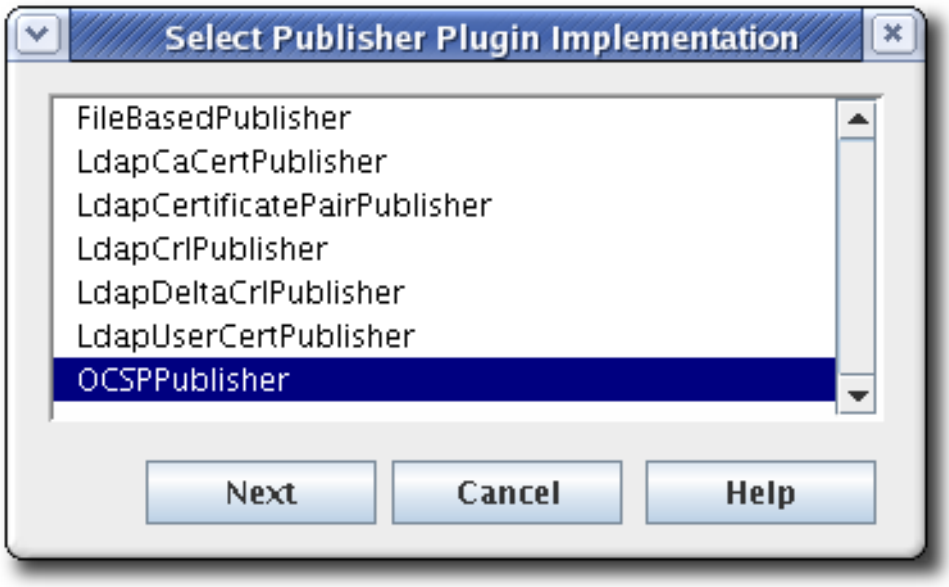
1. Log into the Certificate Manager Console.

```
pkiconsole https://server.example.com:8443/ca
```

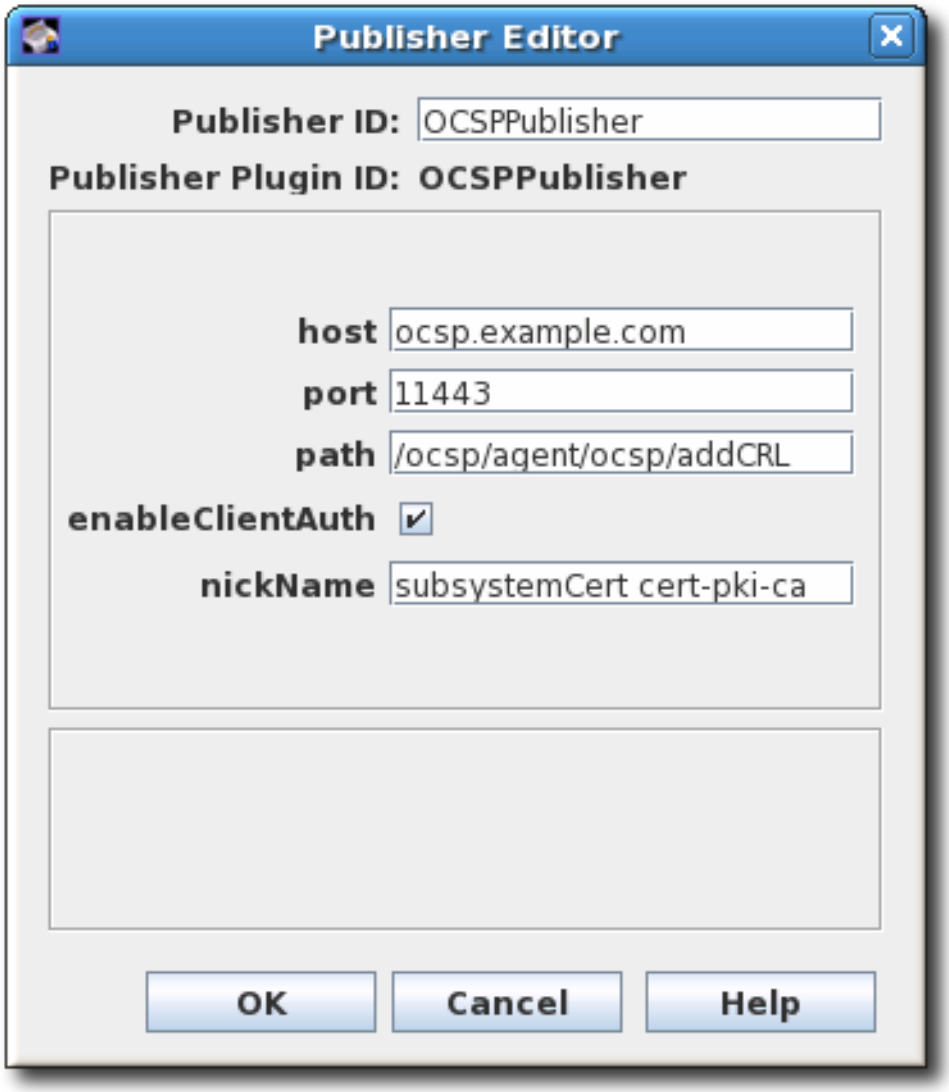
2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Publishers**.



3. Click **Add** to open the **Select Publisher Plug-in Implementation** window, which lists registered publisher modules.



4. Select the **OCSPPublisher** module, then open the editor window. This is the publisher module that enables the Certificate Manager to publish CRLs to the Online Certificate Status Manager.



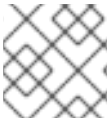
- The publisher ID must be an alphanumeric string with no spaces, like **PublishCertsToOCSP**.

- The **host** can be the fully-qualified domain name, such as **ocspResponder.example.com**, or an IPv4 or IPv6 address.
- The default path is the directory to send the CRL to, like **/ocsp/agent/ocsp/addCRL**.
- If client authentication is used (**enableClientAuth** is checked), then the **nickname** field gives the nickname of the certificate to use for authentication. This certificate must already exist in the OCSP security database; this will usually be the CA subsystem certificate.

5. Create a user entry for the CA on the OCSP Manager. The user is used to authenticate to the OCSP when sending a new CRL. There are two things required:
   - Name the OCSP user entry after the CA server, like **CA-***hostname-EEport*.
   - Use whatever certificate was specified in the publisher configuration as the user certificate in the OCSP user account. This is usually the CA's subsystem certificate.

   Setting up subsystem users is covered in Section 15.3.2.1, "Creating Users".

After configuring the publisher, configure the rules for the published certificates and CRLs, as described in Section 9.5, "Creating Rules".



NOTE

pkiconsole is being deprecated.

9.4. CONFIGURING PUBLISHING TO AN LDAP DIRECTORY

The general process to configure publishing involves setting up a publisher to publish the certificates or CRLs to the specific location. There can be a single publisher or multiple publishers, depending on how many locations will be used. The locations can be split by certificates and CRLs or finer definitions, such as certificate type. Rules determine which type to publish and to what location by being associated with the publisher.

Configuring LDAP publishing is similar to other publishing procedures, with additional steps to configure the directory:

1. Configure the Directory Server to which certificates will be published. Certain attributes have to be added to entries and bind identities and authentication methods have to be configured.
2. Configure a publisher for each type of object published: CA certificates, cross-pair certificates, CRLs, and user certificates. The publisher declares in which attribute to store the object. The attributes set by default are the X.500 standard attributes for storing each object type. This attribute can be changed in the publisher, but generally, it is not necessary to change the LDAP publishers.
3. Set up mappers to enable an entry's DN to be derived from the certificate's subject name. This generally does not need set for CA certificates, CRLs, and user certificates. There can be more than one mapper set for a type of certificate. This can be useful, for example, to publish certificates for two sets of users from different divisions of a company who are located in different parts of the directory tree. A mapper is created for each of the groups to specify a different branch of the tree.

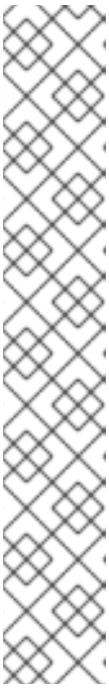
   For details about setting up mappers, see Section 9.4.3, "Creating Mappers".
4. Create rules to connect publishers to mappers, as described in Section 9.5, "Creating Rules".

5. Enable publishing, as described in Section 9.6, “Enabling Publishing” .

9.4.1. Configuring the LDAP Directory

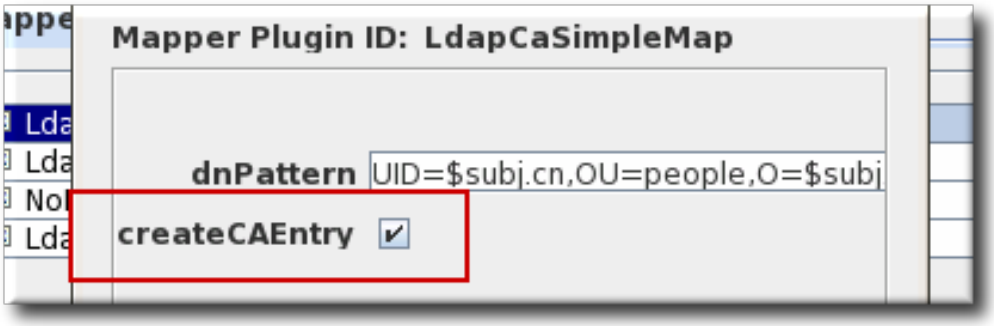
Before certificates and CRLs can be published, the Directory Server must be configured to work with the publishing system. This means that user entries must have attributes that allow them to receive certificate information, and entries must be created to represent the CRLs.

1. Set up the entry for the CA. For the Certificate Manager to publish its CA certificate and CRL, the directory must include an entry for the CA.



NOTE

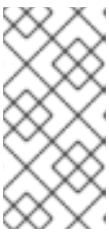
When LDAP publishing is configured, the Certificate Manager automatically creates or converts an entry for the CA in the directory. This option is set in both the CA and CRL mapper instances and enabled by default. If the directory restricts the Certificate Manager from creating entries in the directory, turn off this option in those mapper instances, and add an entry for the CA manually in the directory.



When adding the CA's entry to the directory, select the entry type based on the DN of the CA:

- If the CA's DN begins with the **cn** component, create a new **person** entry for the CA. Selecting a different type of entry may not allow the **cn** component to be specified.
- If the CA's DN begins with the **ou** component, create a new **organizationalunit** entry for the CA.

The entry does not have to be in the **pkiCA** or **certificationAuthority** object class. The Certificate Manager will convert this entry to the **pkiCA** or **certificationAuthority** object class automatically by publishing its CA's signing certificate.



NOTE

The **pkiCA** object class is defined in RFC 4523, while the **certificationAuthority** object class is defined in the (obsolete) RFC 2256. Either object class is acceptable, depending on the schema definitions used by the Directory Server. In some situations, both object classes can be used for the same CA entry.

For more information on creating directory entries, see the Red Hat Directory Server documentation.

2. Add the correct schema elements to the CA and user directory entries.

For a Certificate Manager to publish certificates and CRLs to a directory, it must be configured with specific attributes and object classes.

Object Type	Schema	Reason
End-entity certificate	userCertificate;binary (attribute)	This is the attribute to which the Certificate Manager publishes the certificate. This is a multi-valued attribute, and each value is a DER-encoded binary X.509 certificate. The LDAP object class named **inetOrgPerson** allows this attribute. The **strongAuthenticationUser** object class allows this attribute and can be combined with any other object class to allow certificates to be published to directory entries with other object classes. The Certificate Manager does not automatically add this object class to the schema table of the corresponding Directory Server. If the directory object that it finds does not allow the **userCertificate;binary** attribute, adding or removing the certificate fails.
CA certificate	caCertificate;binary (attribute)	This is the attribute to which the Certificate Manager publishes the certificate. The Certificate Manager publishes its own CA certificate to its own LDAP directory entry when the server starts. The entry corresponds to the Certificate Manager's issuer name. This is a required attribute of the **pkiCA** or **certificationAuthority** object class. The Certificate Manager adds this object class to the directory entry for the CA if it can find the CA's directory entry.

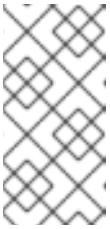
Object Type	Schema	Reason
CRL	certificateRevocationList;binary (attribute)	This is the attribute to which the Certificate Manager publishes the CRL. The Certificate Manager publishes the CRL to its own LDAP directory entry. The entry corresponds to the Certificate Manager's issuer name. This is an attribute of the **pkiCA** or **certificationAuthority** object class. The value of the attribute is the DER-encoded binary X.509 CRL. The CA's entry must already contain the **pkiCA** or **certificationAuthority** object class for the CRL to be published to the entry.
Delta CRL	deltaRevocationList;binary (attribute)	This is the attribute to which the Certificate Manager publishes the delta CRL. The Certificate Manager publishes the delta CRL to its own LDAP directory entry, separate from the full CRL. The delta CRL entry corresponds to the Certificate Manager's issuer name. This attribute belongs to the **deltaCRL** or **certificationAuthority-V2** object class. The value of the attribute is the DER-encoded binary X.509 delta CRL.

3. Set up a bind DN for the Certificate Manager to use to access the Directory Server.

   The Certificate Manager user must have read-write permissions to the directory to publish certificates and CRLs to the directory so that the Certificate Manager can modify the user entries with certificate-related information and the CA entry with CA's certificate and CRL related information.

   The bind DN entry can be either of the following:

   - An existing DN that has write access, such as the Directory Manager.
   - A new user which is granted write access. The entry can be identified by the Certificate Manager's DN, such as **cn=testCA, ou=Research Dept, o=Example Corporation, st=California, c=US**.



NOTE

Carefully consider what privileges are given to this user. This user can be restricted in what it can write to the directory by creating ACLs for the account. For instructions on giving write access to the Certificate Manager's entry, see the Directory Server documentation.

4. Set the directory authentication method for how the Certificate Manager authenticates to Directory Server. There are three options: basic authentication (simple username and password); SSL without client authentication (simple username and password); and SSL with client authentication (certificate-based).

   See the Red Hat Directory Server documentation for instructions on setting up these methods of communication with the server.

9.4.2. Configuring LDAP Publishers

The Certificate Manager creates, configures, and enables a set of publishers that are associated with LDAP publishing. The default publishers (for CA certificates, user certificates, CRLs, and cross-pair certificates) already conform to the X.500 standard attributes for storing certificates and CRLs and do not need to be changed.

Table 9.1. LDAP Publishers

Publisher	Description
LdapCaCertPublisher	Publishes CA certificates to the LDAP directory.
LdapCrlPublisher	Publishes CRLs to the LDAP directory.
LdapDeltaCrlPublisher	Publishes delta CRLs to the LDAP directory.
LdapUserCertPublisher	Publishes all types of end-entity certificates to the LDAP directory.
LdapCrossCertPairPublisher	Publishes cross-signed certificates to the LDAP directory.

9.4.3. Creating Mappers

Mappers are only used with LDAP publishing. Mappers define a relationship between a certificate's subject name and the DN of the directory entry to which the certificate is published. The Certificate Manager needs to derive the DN of the entry from the certificate or the certificate request so it can determine which entry to use. The mapper defines the relationship between the DN for the user entry and the subject name of the certificate or other input information so that the exact DN of the entry can be determined and found in the directory.

When it is configured, the Certificate Manager automatically creates a set of mappers defining the most common relationships. The default mappers are listed in Table 9.2, “Default Mappers”.

Table 9.2. Default Mappers

Mapper	Description
LdapUserCertMap	Locates the correct attribute of user entries in the directory in order to publish user certificates.
LdapCrlMap	Locates the correct attribute of the CA's entry in the directory in order to publish the CRL.
LdapCaCertMap	Locates the correct attribute of the CA's entry in the directory in order to publish the CA certificate.

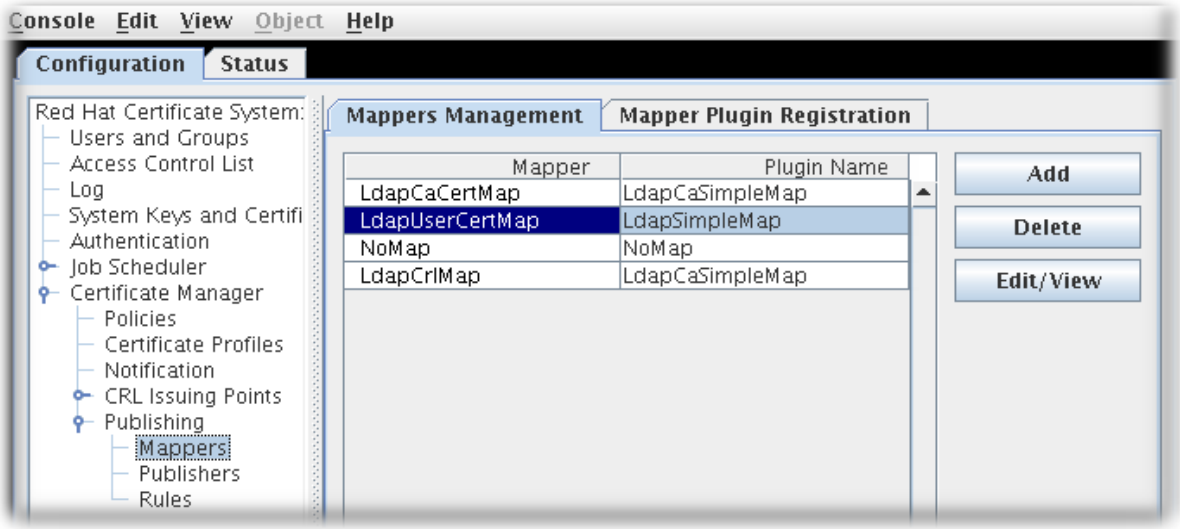
To use the default mappers, configure each of the macros by specifying the DN pattern and whether to create the CA entry in the directory. To use other mappers, create and configure an instance of the mapper. For more information, see Section C.2, "Mapper Plug-in Modules " .

1. Log into the Certificate Manager Console.

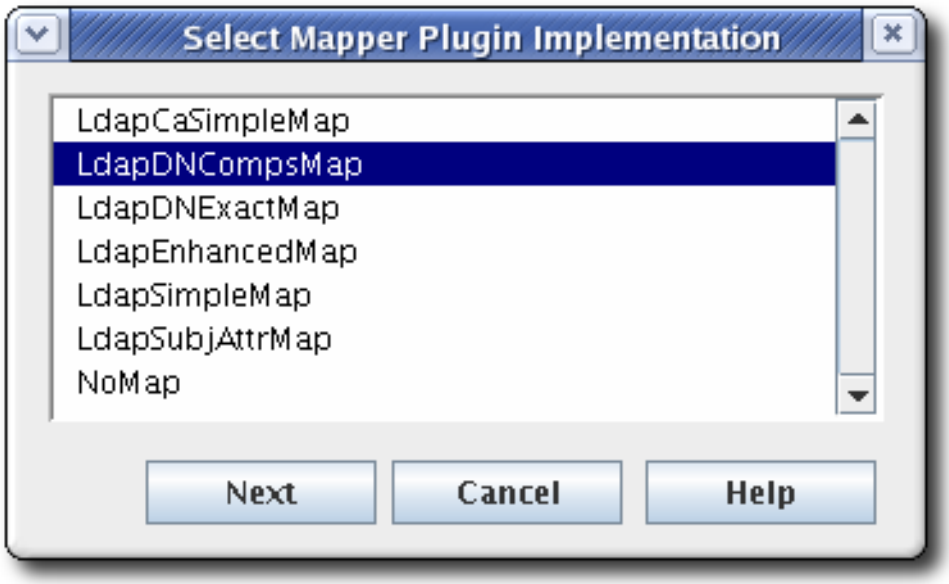
   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Mappers**.

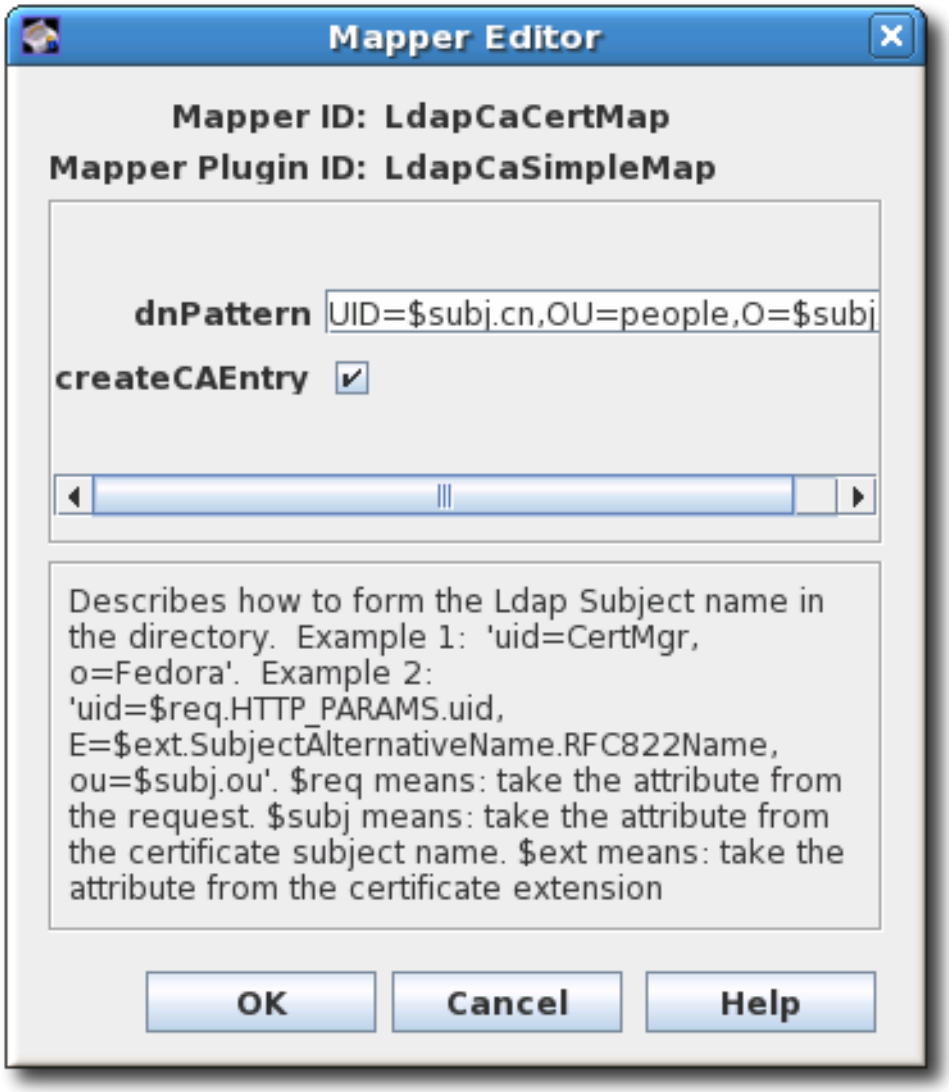
   The **Mappers Management** tab, which lists configured mappers, opens on the right.



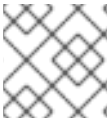
3. To create a new mapper instance, click **Add**. The **Select Mapper Plugin Implementation** window opens, which lists registered mapper modules. Select a module, and edit it. For complete information about these modules, see Section C.2, "Mapper Plug-in Modules " .



4. Edit the mapper instance, and click **OK**.



See Section C.2, "Mapper Plug-in Modules " for detailed information about each mapper.



NOTE

pkiconsole is being deprecated.

9.4.4. Completing Configuration: Rules and Enabling

After configuring the mappers for LDAP publishing, configure the rules for the published certificates and CRLs, as described in Section 9.5, "Creating Rules".

Once the configuration is complete, enable publishing, as described in Section 9.6, "Enabling Publishing".

9.5. CREATING RULES

Rules determine what certificate object is published in what location. Rules work independently, not in tandem. A certificate or CRL that is being published is matched against every rule. Any rule which it matches is activated. In this way, the same certificate or CRL can be published to a file, to an Online Certificate Status Manager, and to an LDAP directory by matching a file-based rule, an OCSP rule, and matching a directory-based rule.

Rules can be set for each object type: CA certificates, CRLs, user certificates, and cross-pair certificates. The rules can be more detailed for different kinds of certificates or different kinds of CRLs.

The rule first determines if the object matches by matching the type and predicate set up in the rule with the object. Where matching objects are published is determined by the publisher and mapper associated with the rule.

Rules are created for each type of certificate the Certificate Manager issues.

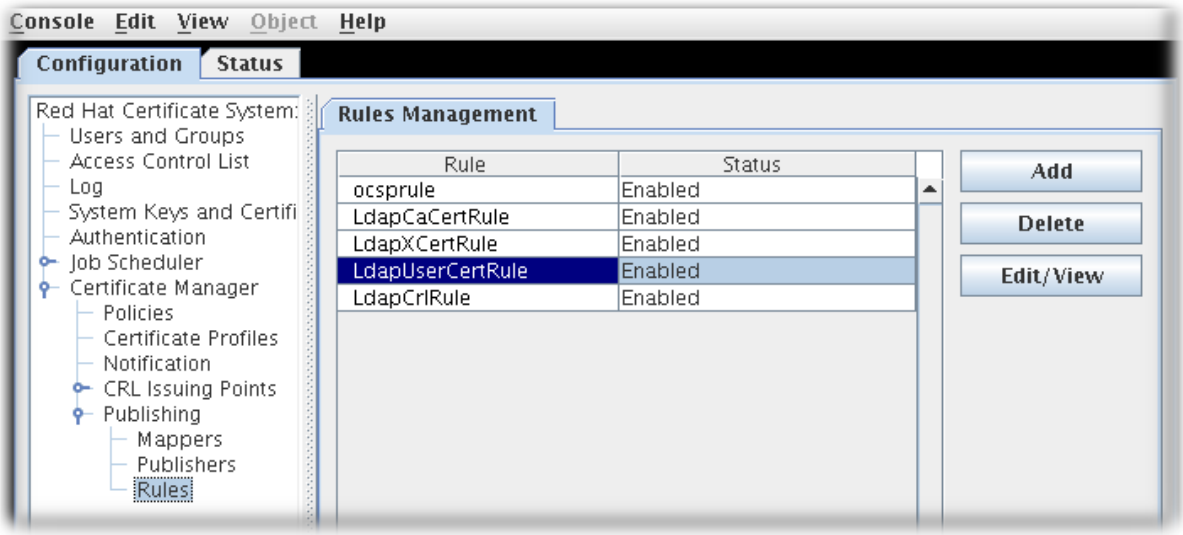
Modify publishing rules by doing the following:

1. Log into the Certificate Manager Console.

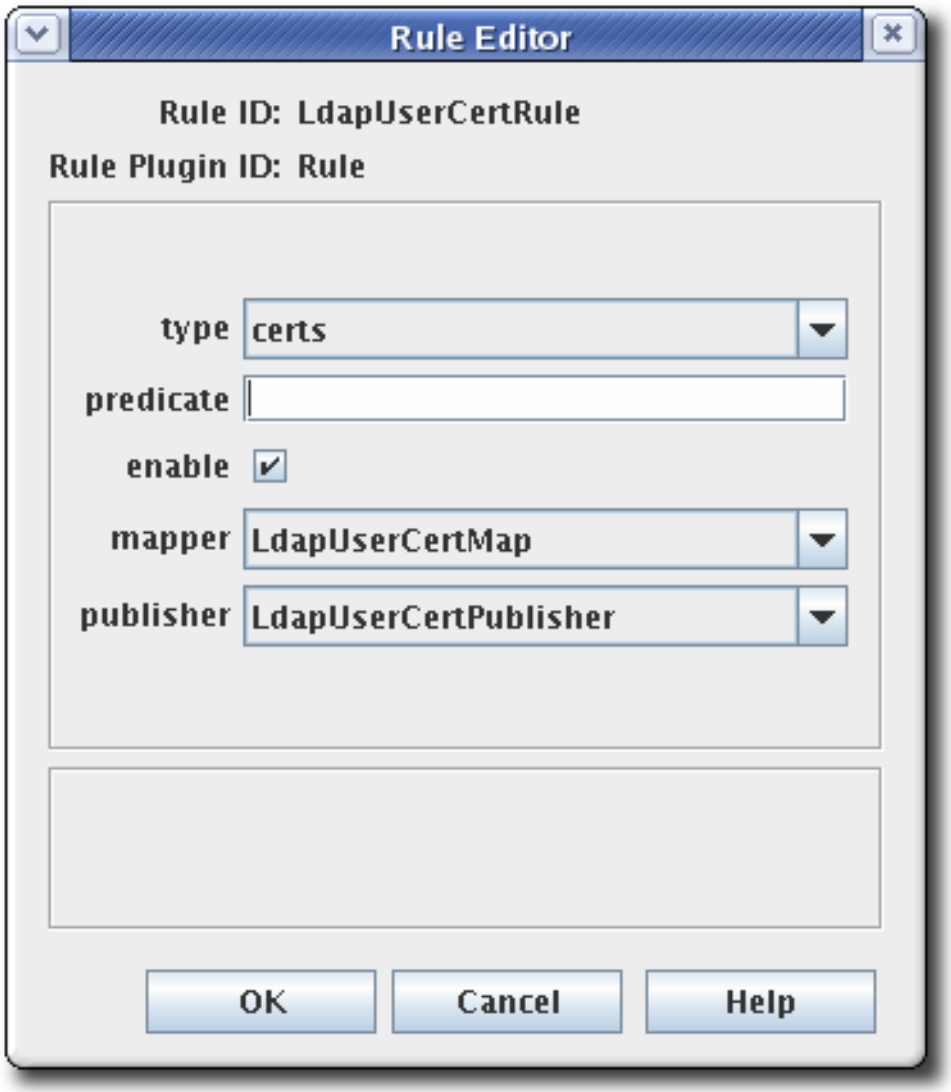
```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Rules**.

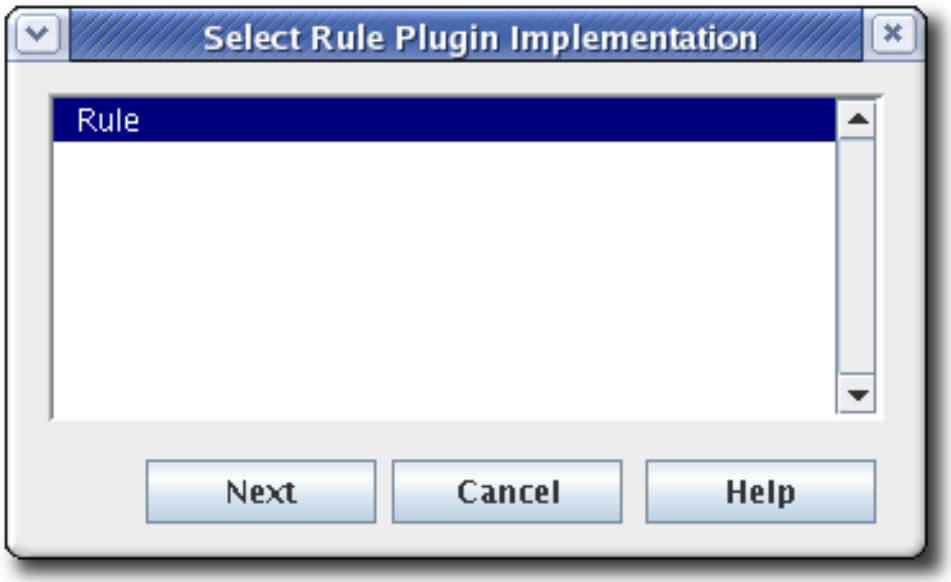
   The **Rules Management** tab, which lists configured rules, opens on the right.



3. To edit an existing rule, select that rule from the list, and click **Edit**. This opens the **Rule Editor** window.

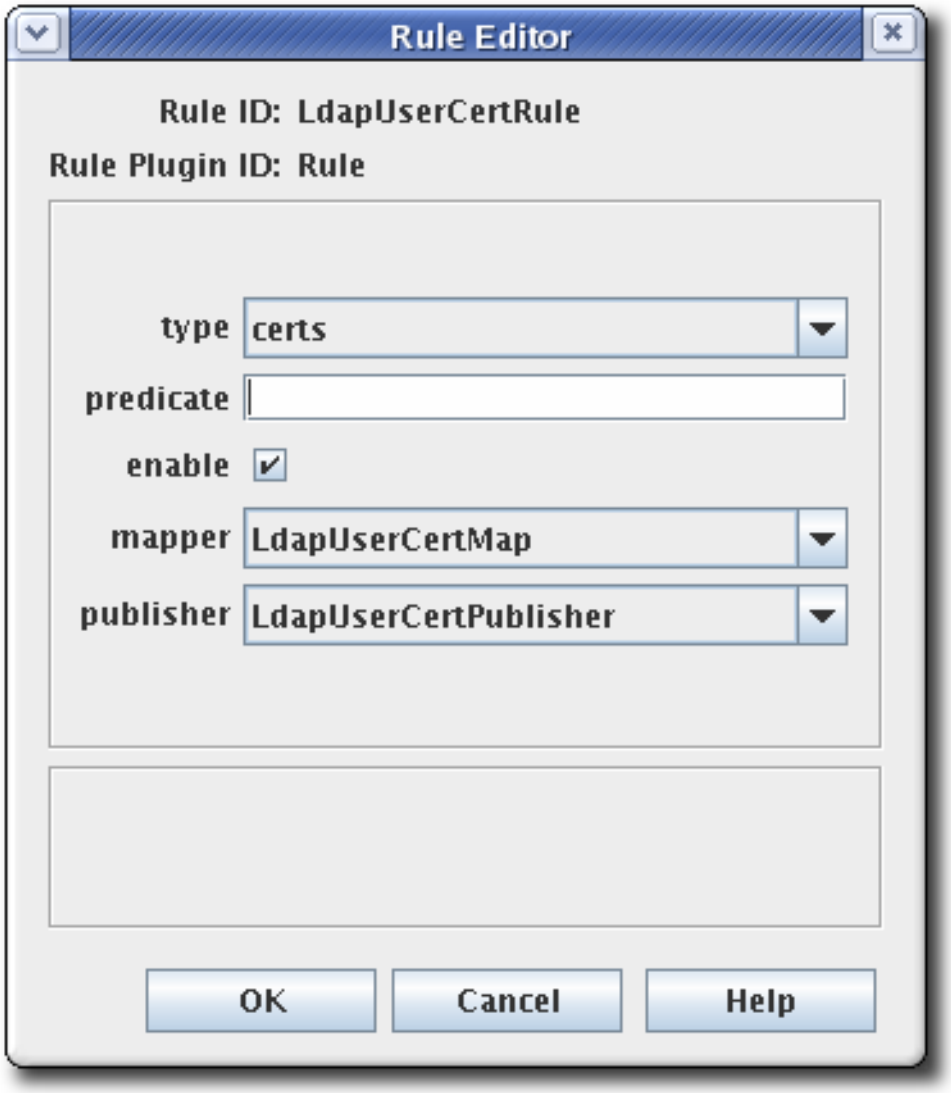


4. To create a rule, click **Add**. This opens the **Select Rule Plug-in Implementation** window.



Select the **Rule** module. This is the only default module. If any custom modules have been been registered, they are also available.

5. Edit the rule.

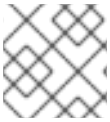


- **type**. This is the type of certificate for which the rule applies. For a CA signing certificate, the value is **cacert**. For a cross-signed certificate, the value is **xcert**. For all other types of certificates, the value is **certs**. For CRLs, specify **crl**.
- **predicate**. This sets the predicate value for the type of certificate or CRL issuing point to which this rule applies. The predicate values for CRL issuing points, delta CRLs, and certificates are listed in Table 9.3, "Predicate Expressions".
- **enable**.
- **mapper**. Mappers are not necessary when publishing to a file; they are only needed for LDAP publishing. If this rule is associated with a publisher that publishes to an LDAP directory, select an appropriate mapper here. Leave blank for all other forms of publishing.
- **publisher**. Sets the publisher to associate with the rule.

Table 9.3, "Predicate Expressions" lists the predicates that can be used to identify CRL issuing points and delta CRLs and certificate profiles.

Table 9.3. Predicate Expressions

Predicate Type	Predicate
CRL Issuing Point	**issuingPointId==***Issuing_Point_Instance_ID* **&& isDeltaCRL==[true|false]** To publish only the master CRL, set **isDeltaCRL==false**. To publish only the delta CRL, set **isDeltaCRL==true**. To publish both, set a rule for the master CRL and another rule for the delta CRL.
Certificate Profile	**profileId==***profile_name* To publish certificates based on the profile used to issue them, set **profileId==** to a profile name, such as **caServerCert**.



NOTE

pkiconsole is being deprecated.

9.6. ENABLING PUBLISHING

Publishing can be enabled for only files, only LDAP, or both. Publishing should be enabled after setting up publishers, rules, and mappers. Once enabled, the server attempts to begin publishing. If publishing was not configured correctly before being enabled, publishing may exhibit undesirable behavior or may fail.

NOTE

Configure CRLs. CRLs must be configured before they can be published. See Chapter 7, *Revoking Certificates and Issuing CRLs*.

1. Log into the Certificate Manager Console.

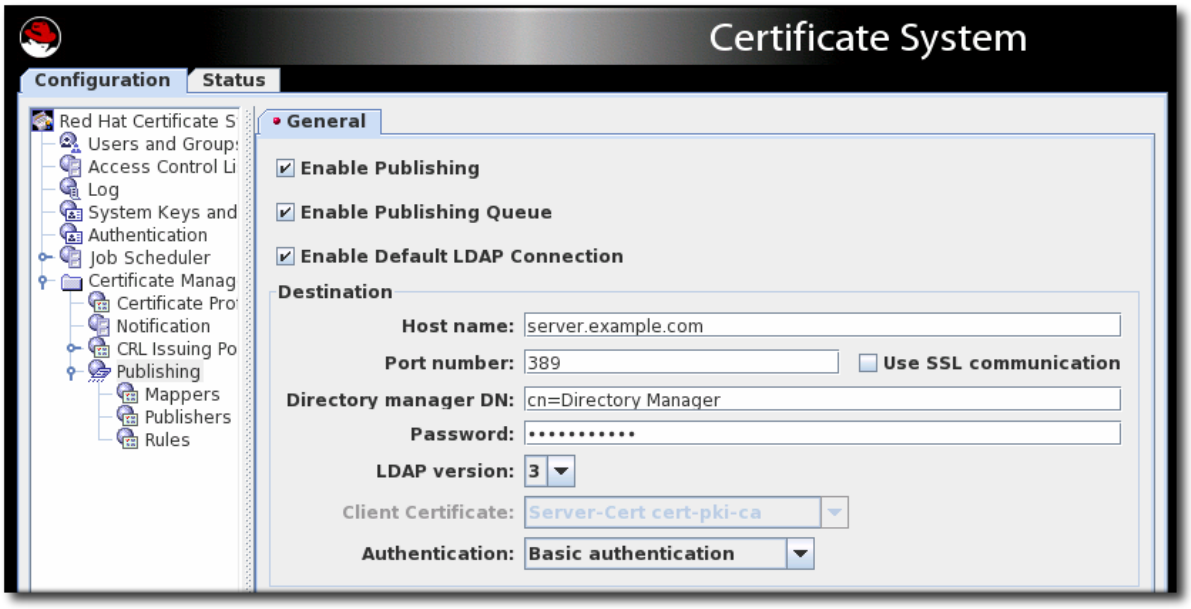
```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**.

   The right pane shows the details for publishing to an LDAP-compliant directory.

3. To enable publishing to a file only, select **Enable Publishing**.

4. To enable LDAP publishing, select both **Enable Publishing** and **Enable Default LDAP Connection**.



In the **Destination** section, set the information for the Directory Server instance.

- **Host name**. If the Directory Server is configured for SSL client authenticated communication, the name must match the **cn** component in the subject DN of the Directory Server's SSL server certificate.

  The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address.

- **Port number**.

- **Directory Manager DN**. This is the distinguished name (DN) of the directory entry that has Directory Manager privileges. The Certificate Manager uses this DN to access the directory tree and to publish to the directory. The access control set up for this DN determines whether the Certificate Manager can perform publishing. It is possible to create another DN that has limited read-write permissions for only those attributes that the publishing system actually needs to write.

- **Password.** This is the password which the CA uses to bind to the LDAP directory to which the certificate or CRL is published. The Certificate Manager saves this password in its **password.conf** file. For example:

  ```
  CA LDAP Publishing:password
  ```

  NOTE

  The parameter name which identifies the publishing password (**CA LDAP Publishing**) is set in the Certificate Manager's **CS.cfg** file in the **ca.publish.ldappublish.ldap.ldapauth.bindPWPrompt** parameter, and it can be edited.

- **Client certificate**. This sets the certificate the Certificate Manager uses for SSL client authentication to the publishing directory. By default, the Certificate Manager uses its SSL server certificate.

- **LDAP version**. Select LDAP version 3.

- **Authentication**. The way the Certificate Manager authenticates to the Directory Server. The choices are **Basic authentication** and **SSL client authentication**.

  If the Directory Server is configured for basic authentication or for SSL communication without client authentication, select **Basic authentication** and specify values for the Directory manager DN and password.

  If the Directory Server is configured for SSL communication with client authentication, select **SSL client authentication** and the **Use SSL communication** option, and identify the certificate that the Certificate Manager must use for SSL client authentication to the directory.

The server attempts to connect to the Directory Server. If the information is incorrect, the server displays an error message.

NOTE

pkiconsole is being deprecated.

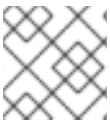
9.7. ENABLING A PUBLISHING QUEUE

Part of the enrollment process includes publishing the issued certificate to any directories or files. This, essentially, closes out the initial certificate request. However, publishing a certificate to an external network can significantly slow down the issuance process — which leaves the request open.

To avoid this situation, administrators can enable a *publishing queue*. The publishing queue separates the publishing operation (which may involve an external LDAP directory) from the request and enrollment operations, which uses a separate request queue. The request queue is updated immediately to show that the enrollment process is complete, while the publishing queue sends the information at the pace of the network traffic.

The publishing queue sets a defined, limited number of threads that publish generated certificates, rather than opening a new thread for each approved certificate.

The publishing queue is disabled by default. It can be enabled in the CA Console, along with enabling publishing.



NOTE

pkiconsole is being deprecated.

NOTE

While the publishing queue is disabled by default, the queue is automatically enabled if LDAP publishing is enabled *in the Console*. Otherwise, the queue can be enabled manually.

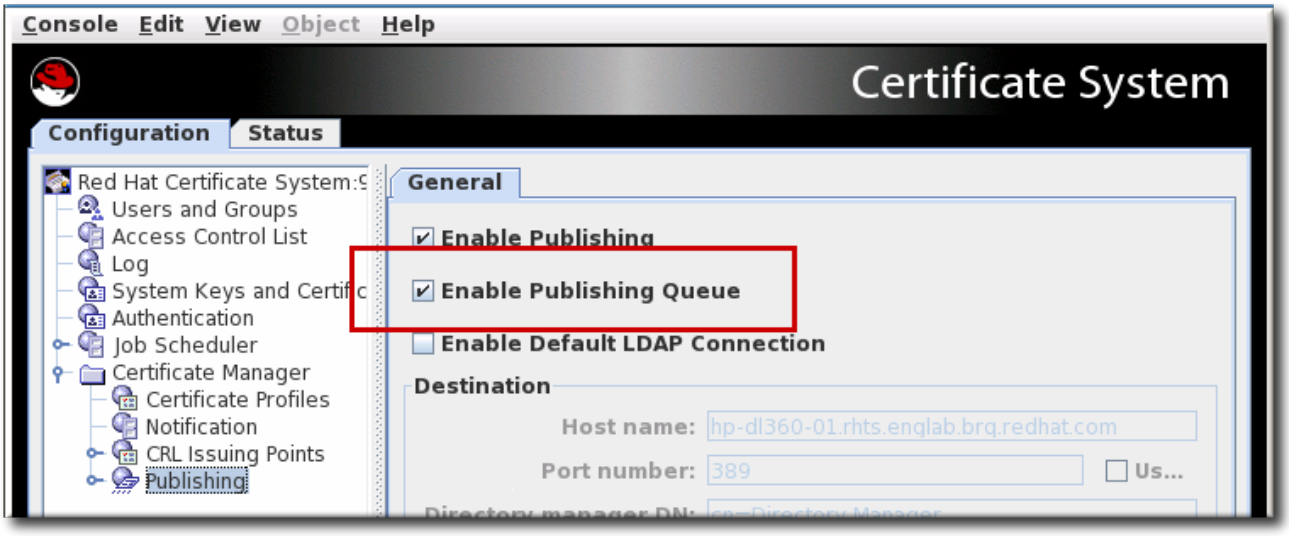


Figure 9.1. Enabling the Publishing Queue

NOTE

Enabling the publishing queue by editing the **CS.cfg** file allows administrators to set other options for publishing, like the number of threads to use for publishing operations and the queue page size.

For instruction on how to configure this feature by editing the **CS.cfg** file, see the *Enabling and Configuring a Publishing Queue* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

9.8. SETTING UP RESUMABLE CRL DOWNLOADS

Certificate System provides option for interrupted CRL downloads to be resumed smoothly. This is done by publishing the CRLs as a plain file over HTTP. This method of downloading CRLs gives flexibility in retrieving CRLs and lowers overall network congestion.

9.8.1. Retrieving CRLs Using wget

Because CRLs can be published as a text file over HTTP, they can be manually retrieved from the CA using a tool such as **wget**. The **wget** command can be used to retrieve any published CRL. For example, to retrieve a full CRL which is newer than the previous full CRL:

```
[root@server ~]# wget --no-check-certificate -d
https://server.example.com:8443/ca/ee/ca/crl/MasterCRL.bin
```

The relevant parameters for **wget** are summarized in Table 9.4, “wget Options to Use for Retrieving CRLs”.

Table 9.4. wget Options to Use for Retrieving CRLs

Argument	Description
no argument	Retrieves the full CRL.

Argument	Description
-N	Retrieves the CRL that is newer than the local copy (delta CRL).
-c	Retrieves a partially-downloaded file.
--no-check-certificate	Skips SSL for the connection, so it is not necessary to configure SSL between the host and client.
-d	Prints debug information.

9.9. PUBLISHING CROSS-PAIR CERTIFICATES

The cross-pair certificates can be published as a **crossCertificatePair** entry to an LDAP directory or to a file; this is enabled by default. If this has been disabled, it can be re-enabled through the Certificate Manager Console by doing the following:

1. Open the CA console.

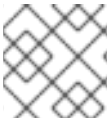
```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, select the **Certificate Manager** link in the left pane, then the **Publishing** link.
3. Click the **Rules** link under **Publishing**. This opens the **Rules Management** pane on the right.
4. If the rule exists and has been disabled, select the **enable** checkbox. If the rule has been deleted, then click **Add** and create a new rule.
    1. Select **xcerts** from the **type** drop-down menu.
    2. Make sure the **enable** checkbox is selected.
    3. Select **LdapCaCertMap** from the **mapper** drop-down menu.
    4. Select **LdapCrossCertPairPublisher** from the **publisher** drop-down menu.

The mapper and publisher specified in the publishing rule are both listed under **Mapper** and **Publisher** under the **Publishing** link in the left navigation window of the CA Console. The mapper, **LdapCaCertMap**, by default designates that the **crossCertificatePair** be stored to the **LdapCaSimpleMap** LDAP entry. The publisher, **LDAPCrossPairPublisher**, by default sets the attribute to store the cross-pair certificate in the CA entry to **crossCertificatePair;binary**.

For more information on using cross-pair certificates, see Section 17.5, “Using Cross-Pair Certificates”.

For more information on creating cross-pair certificate profiles, see the *Configuring Cross-Pair profiles* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .



NOTE

pkiconsole is being deprecated.

9.10. TESTING PUBLISHING TO FILES

To verify that the Certificate Manager is publishing certificates and CRLs correctly to file:

1. Open the CA's end-entities page, and request a certificate.
2. Approve the request through the agent services page, if required.
3. Retrieve the certificate from the end-entities page, and download the certificate into the browser.
4. Check whether the server generated the DER-encoded file containing the certificate.

   Open the directory to which the binary blob of the certificate is supposed to be published. The certificate file should be named **cert-***serial_number***.der**.
5. Convert the DER-encoded certificate to its base 64-encoded format using the Binary to ASCII tool. For more information on this tool, refer to the **BtoA(1)** man page.

   ```
   BtoA input_file output_file
   ```

   input_file sets the path to the file that contains the DER-encoded certificate, and *output_file* sets the path to the file to write the base-64 encoded certificate.
6. Open the ASCII file; the base-64 encoded certificate is similar to the one shown:

   ```
   -----BEGIN CERTIFICATE-----
   MMIIBtgYJYIZIAYb4QgIFoIIBpzCCAZ8wggGbMIIBRaADAgEAAgEBMA0GCSqGSIb3DQEBB
   AUAMFcxC
   AJBgNVBAYTAlVTMSwwKgYDVQQKEyNOZXRzY2FwZSBDb21tdW5pY2F0aWhfyyuougjgjjg
   mkgjkgmjg
   fjfgjjjgfyjfyj9ucyBDb3Jwb3JhdGlvbjpMEaMBgGA1UECxMRSXNzdWluZyhgdfhbfdpffjphotoo
   gdhkBBdXRob3JpdHkwHhcNOTYxMTA4MDkwNzM0WhcNOTgxMTA4MDkwNzMM0WjBXMQ
   swCQYDVQQGEwJ
   VUzEsMCoGA1UEChMjTmV0c2NhcGUgQ29tbXVuaWNhdGlvbnMgQ29ycG9yY2F0aW9ucyB
   Db3Jwb3Jhd
   GlvbjpMEaMBgGA1UECxMRSXNzdWluZyBBdXRob3JpdHkwHh
   -----END CERTIFICATE-----
   ```
7. Convert the base 64-encoded certificate to a readable form using the Pretty Print Certificate tool. For more information on this tool, refer to the **PrettyPrintCert(1)** man page.

   ```
   PrettyPrintCert input_file [output_file]
   ```

   input_file sets the path to the ASCII file that contains the base-64 encoded certificate, and *output_file*, optionally, sets the path to the file to write the certificate. If an output file is not set, the certificate information is written to the standard output.
8. Compare the output with the certificate issued; check the serial number in the certificate with the one used in the filename.

   If everything matches, the Certificate Manager is configured correctly to publish certificates to file.
9. Revoke the certificate.

10. Check whether the server generated the DER-encoded file containing the CRL.

    Open the directory to which the server is to publish the CRL as a binary blob. The CRL file should have a name in the form **crl-***this_update***.der**. *this_update* specifies the value derived from the time-dependent **This Update** variable of the CRL.

11. Convert the DER-encoded CRL to its base 64-encoded format using the Binary to ASCII tool.

    ```
    BtoA input_file output_file
    ```

12. Convert the base 64-encoded CRL to readable form using the Pretty Print CRL tool.

    ```
    PrettyPrintCrl input_file [output_file]
    ```

13. Compare the output.

9.11. VIEWING CERTIFICATES AND CRLS PUBLISHED TO FILE

Certificates and CRLs can be published to two types of files: base-64 encoded or DER-encoded. The content of these files can be viewed by converting the files to pretty-print format using the **dumpasn1** tool or the **PrettyPrintCert** or **PrettyPrintCrl** tool.

To view the content in a base-64 encoded file:

1. Convert the base-64 file to binary. For example:

   ```
   AtoB /tmp/example.b64 /tmp/example.bin
   ```

2. Use the **PrettyPrintCert** or **PrettyPrintCrl** tool to convert the binary file to pretty-print format. For example:

   ```
   PrettyPrintCert example.bin example.cert
   ```

To view the content of a DER-encoded file, simply run the **dumpasn1**, **PrettyPrintCert**, or **PrettyPrintCrl** tool with the DER-encoded file. For example:

```
PrettyPrintCrl example.der example.crl
```

9.12. UPDATING CERTIFICATES AND CRLS IN A DIRECTORY

The Certificate Manager and the publishing directory can become out of sync if certificates are issued or revoked while the Directory Server is down. Certificates that were issued or revoked need to be published or unpublished manually when the Directory Server comes back up.

To find certificates that are out of sync with the directory - valid certificates that are not in the directory and revoked or expired certificates that are still in the directory - the Certificate Manager keeps a record of whether a certificate in its internal database has been published to the directory. If the Certificate Manager and the publishing directory become out of sync, use the **Update Directory** option in the Certificate Manager agent services page to synchronize the publishing directory with the internal database.

The following choices are available for synchronizing the directory with the internal database:

- Search the internal database for certificates that are out of sync and publish or unpublish.
- Publish certificates that were issued while the Directory Server was down. Similarly, unpublish certificates that were revoked or that expired while Directory Server was down.
- Publish or unpublish a range of certificates based on serial numbers, from serial number *xx* to serial number *yy*.

A Certificate Manager's publishing directory can be manually updated by a Certificate Manager agent only.

9.12.1. Manually Updating Certificates in the Directory

The **Update Directory Server** form in the Certificate Manager agent services page can be used to update the directory manually with certificate-related information. This form initiates a combination of the following operations:

- Update the directory with certificates.
- Remove expired certificates from the directory.

  Removing expired certificates from the publishing directory can be automated by scheduling an automated job. For details, see Chapter 13, *Setting Automated Jobs*.
- Remove revoked certificates from the directory.

Manually update the directory with changes by doing the following:

1. Open the Certificate Manager agent services page.
2. Select the **Update Directory Server** link.
3. Select the appropriate options, and click **Update Directory**.

   The Certificate Manager starts updating the directory with the certificate information in its internal database. If the changes are substantial, updating the directory can take considerable time. During this period, any changes made through the Certificate Manager, including any certificates issued or any certificates revoked, may not be included in the update. If any certificates are issued or revoked while the directory is updated, update the directory again to reflect those changes.

When the directory update is complete, the Certificate Manager displays a status report. If the process is interrupted, the server logs an error message.

If the Certificate Manager is installed as a root CA, the CA signing certificate may get published using the publishing rule set up for user certificates when using the agent interface to update the directory with valid certificates. This may return an object class violation error or other errors in the mapper. Selecting the appropriate serial number range to exclude the CA signing certificate can avoid this problem. The CA signing certificate is the first certificate a root CA issues.

- Modify the default publishing rule for user certificates by changing the value of the **predicate** parameter to **profileId!=caCACert**.
- Use the **LdapCaCertPublisher** publisher plug-in module to add another rule, with the predicate parameter set to **profileId=caCACert**, for publishing subordinate CA certificates.

9.12.2. Manually Updating the CRL in the Directory

The **Certificate Revocation List** form in the Certificate Manager agent services page manually updates the directory with CRL-related information.

Manually update the CRL information by doing the following:

1. Open the Certificate Manager agent services page.
2. Select **Update Revocation List**.
3. Click **Update**.

The Certificate Manager starts updating the directory with the CRL in its internal database. If the CRL is large, updating the directory takes considerable time. During this period, any changes made to the CRL may not be included in the update.

When the directory is updated, the Certificate Manager displays a status report. If the process is interrupted, the server logs an error message.

9.13. REGISTERING CUSTOM MAPPER AND PUBLISHER PLUG-IN MODULES

New mapper or publisher plug-in modules can be registered in a Certificate Manager's publishing framework. Unwanted mapper or publisher plug-in modules can be deleted. Before deleting a module, delete all the rules that are based on this module.

1. Create the custom job class. For this example, the custom publisher plug-in is called **MyPublisher.java**.
2. Compile the new class.

```
javac -d . -classpath $CLASSPATH MyPublisher.java
```

3. Create a directory in the CA's **WEB-INF** web directory to hold the custom classes, so that the CA can access them.

```
mkdir /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes
```

4. Copy the new plug-in files into the new **classes** directory, and set the owner to the Certificate System system user (**pkiuser**).

```
cp -pr com /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes

chown -R pkiuser:pkiuser /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes
```

5. Register the plug-in.
    1. Log into the Certificate Manager Console.

```
pkiconsole https://server.example.com:8443/ca
```

    2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**.

3. To register a mapper module, select **Mappers**, and then select the **Mapper Plugin Registration** tab.

   To register a publisher module, select **Publishers**, and then select the **Publisher Plug-in Registration** tab.

4. To register a plug-in, click **Register**.

5. Set the plug-in name and plug-in class name. The class name is, the path to the implementing Java class. If this class is part of a package, include the package name. For example, to register a class named **customMapper** in a package named **com.customplugins**, the name is **com.customplugins.customMapper**.

NOTE

pkiconsole is being deprecated.

CHAPTER 10. AUTHENTICATION FOR ENROLLING CERTIFICATES

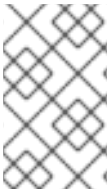
This chapter covers how to enroll end entity certificates, how to create and manage server certificates, the authentication methods available in the Certificate System to use when enrolling end entity certificates, and how to set up those authentication methods.

Enrollment is the process of issuing certificates to an end entity. The process is creating and submitting the request, authenticating the user requesting it, and then approving the request and issuing the certificate.

The method used to authenticate the end entity determines the entire enrollment process. There are three ways that the Certificate System can authenticate an entity:

- In *agent-approved* enrollment, end-entity requests are sent to an agent for approval. The agent approves the certificate request.
- In *automatic* enrollment, end-entity requests are authenticated using a plug-in, and then the certificate request is processed; an agent is not involved in the enrollment process.
- In *CMC enrollment*, a third party application can create a request that is signed by an agent and then automatically processed.

A Certificate Manager is initially configured for agent-approved enrollment and for CMC authentication. Automated enrollment is enabled by configuring one of the authentication plug-in modules. More than one authentication method can be configured in a single instance of a subsystem.



NOTE

An email can be automatically sent to an end entity when the certificate is issued for any authentication method by configuring automated notifications. See Chapter 12, *Using Automated Notifications* for more information on notifications.

10.1. CONFIGURING AGENT-APPROVED ENROLLMENT

The Certificate Manager is initially configured for agent-approved enrollment. An end entity makes a request which is sent to the agent queue for an agent's approval. An agent can modify request, change the status of the request, reject the request, or approve the request. Once the request is approved, the signed request is sent to the Certificate Manager for processing. The Certificate Manager processes the request and issues the certificate.

The agent-approved enrollment method is not configurable. If a Certificate Manager is not configured for any other enrollment method, the server automatically sends all certificate-related requests to a queue where they await agent approval. This ensures that all requests that lack authentication credentials are sent to the request queue for agent approval.

To use agent-approved enrollment, leave the authentication method blank in the profile's **.cfg** file. For example:

```
auth.instance_id=
```

10.2. AUTOMATED ENROLLMENT

In automated enrollment, an end-entity enrollment request is processed as soon as the user successfully authenticates by the method set in the authentication plug-in module; no agent approval is necessary. The following authentication plug-in modules are provided:

- *Directory-based enrollment.* End entities are authenticated against an LDAP directory using their user ID and password or their DN and password. See Section 10.2.1, "Setting up Directory-Based Authentication".

- *PIN-based enrollment.* End entities are authenticated against an LDAP directory using their user ID, password, and a PIN set in their directory entry. See Section 10.2.2, "Setting up PIN-Based Enrollment".

- *Certificate-based authentication*. Entities of some kind — both end users and other entities, like servers or tokens — are authenticated to the CA using a certificate issued by the CA which proves their identity. This is most commonly used for renewal, where the original certificate is presented to authenticate the renewal process. See Section 10.2.3, "Using Certificate-Based Authentication".

- *AgentCertAuth*. This method automatically approves a certificate request if the entity submitting the request is authenticated as a subsystem agent. A user authenticates as an agent by presenting an agent certificate. If the presented certificate is recognized by the subsystem as an agent certificate, then the CA automatically processes the certificate request.

  This form of automatic authentication can be associated with the certificate profile for enrolling for server certificates.

  This plug-in is enabled by default and has no parameters.

- *Flat file-based enrollment* . Used exclusively for router (SCEP) enrollments, a text file is used which contains a list of IP addresses, hostnames, or other identifier and a password, which is usually a random PIN. A router authenticates to the CA using its ID and PIN, and then the CA compares the presented credentials to the list of identities in the text file. See Section 10.2.4, "Configuring Flat File Authentication".

10.2.1. Setting up Directory-Based Authentication

The **UidPwdDirAuth** and the **UdnPwdDirAuth** plug-in modules implement directory-based authentication. End users enroll for a certificate by providing their user IDs or DN and password to authenticate to an LDAP directory.

1. Create an instance of either the **UidPwdDirAuth** or **UdnPwdDirAuth** authentication plug-in module and configure the instance.

   1. Open the CA Console.

      ```
      pkiconsole https://server.example.com:8443/ca
      ```

   2. In the **Configuration** tab, select **Authentication** in the navigation tree.

      The right pane shows the **Authentication Instance** tab, which lists the currently configured authentication instances.

      NOTE

      The **UidPwdDirAuth** plug-in is enabled by default.

3. Click **Add**.

   The **Select Authentication Plug-in Implementation** window appears.

4. Select **UidPwdDirAuth** for user ID and password authentication, or select **UdnPwdDirAuth** for DN and password authentication.

5. Fill in the following fields in the **Authentication Instance Editor** window:

   - **Authentication Instance ID.** Accept the default instance name, or enter a new name.
   - **dnpattern.** Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN.
   - **ldapStringAttributes.** Specifies the list of LDAP string attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes are copied from the authentication directory into the authentication token and used by the certificate profile to generate the subject name. Entering values for this parameter is optional.
   - **ldapByteAttributes.** Specifies the list of LDAP byte (binary) attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token for use by other modules, such as adding additional information to users' certificates.

     Entering values for this parameter is optional.
   - **ldap.ldapconn.host.** Specifies the fully-qualified DNS hostname of the authentication directory.
   - **ldap.ldapconn.port.** Specifies the TCP/IP port on which the authentication directory listens to requests; if the **ldap.ldapconn.secureConn.** checkbox is selected, this should be the SSL port number.
   - **ldap.ldapconn.secureConn.** Specifies the type, SSL or non-SSL, of the port on which the authentication directory listens to requests from the Certificate System. Select if this is an SSL port.
   - **ldap.ldapconn.version.** Specifies the LDAP protocol version, either **2** or **3**. The default is **3**, since all Directory Servers later than version 3.x are LDAPv3.
   - **ldap.basedn.** Specifies the base DN for searching the authentication directory. The server uses the value of the **uid** field from the HTTP input (what a user enters in the enrollment form) and the base DN to construct an LDAP search filter.
   - **ldap.minConns.** Specifies the minimum number of connections permitted to the authentication directory. The permissible values are **1** to **3**.
   - **ldap.maxConns.** Specifies the maximum number of connections permitted to the authentication directory. The permissible values are **3** to **10**.

6. Click **OK**. The authentication instance is set up and enabled.

2. Set the certificate profiles to use to enroll users by setting policies for specific certificates. Customize the enrollment forms by configuring the inputs in the certificate profiles, and include inputs for the information needed by the plug-in to authenticate the user. If the default inputs

do not contain all of the information that needs to be collected, submit a request created with a third-party tool.

For information on configuring the profiles, see Section 3.7.2, "Inserting LDAP Directory Attribute Values and Other Information into the Subject Alt Name".

NOTE

pkiconsole is being deprecated.

Setting up Bound LDAP Connection

Some environments require disallowing an anonymous bind for the LDAP server that is used for authentication. To create a bound connection between a CA and the LDAP server, you need to make the following configuration changes:

- Set up directory-based authentication according to the following example in **CS.cfg**:

```
auths.instance.UserDirEnrollment.ldap.ldapBoundConn=true
auths.instance.UserDirEnrollment.ldap.ldapauth.authtype=BasicAuth
auths.instance.UserDirEnrollment.ldap.ldapauth.bindDN=cn=Directory Manager
auths.instance.UserDirEnrollment.ldap.ldapauth.bindPWPrompt=externalLDAP
externalLDAP.authPrefix=auths.instance.UserDirEnrollment
cms.passwordlist=internaldb,replicationdb,externalLDAP
```

  where **bindPWPrompt** is the tag or prompt that is used in the **password.conf** file; it is also the name used under the **cms.passwordlist** and **authPrefix** options.

- Add the tag or prompt from **CS.cfg** with its password in **password.conf**:

```
externalLDAP=your_password
```

Setting up External Authorization

A directory-based authentication plug-in can also be configured to evaluate the group membership of the user for authentication. To set up the plug-in this way, the following options has to be configured in **CS.cfg**:

- **groupsEnable** is a boolean option that enables retrieval of groups. The default value is **false**.
- **groupsBasedn** is the base DN of groups. It needs to be specified when it differs from the default **basedn**.
- **groups** is the DN component for groups. The default value is **ou=groups**.
- **groupObjectClass** is one of the following group object classes: **groupofuniquenames**, **groupofnames**. The default value is **groupofuniquenames**.
- **groupUseridName** is the name of the user ID attribute in the group object member attribute. The default value is **cn**.
- **useridName** is the name of the user ID DN component. The default value is **uid**.
- **searchGroupUserByUserdn** is a boolean option that determines whether to search the group object member attribute for the **userdn** or **${groupUserIdName}=${uid}** attributes. The default value is **true**.

For example:

```
auths.instance.UserDirEnrollment.pluginName=UidPwdDirAuth
auths.instance.UserDirEnrollment.ldap.basedn=cn=users,cn=accounts,dc=local
auths.instance.UserDirEnrollment.ldap.groupObjectClass=groupofnames
auths.instance.UserDirEnrollment.ldap.groups=cn=groups
auths.instance.UserDirEnrollment.ldap.groupsBasedn=cn=accounts,dc=local
auths.instance.UserDirEnrollment.ldap.groupsEnable=true
auths.instance.UserDirEnrollment.ldap.ldapconn.host=local
auths.instance.UserDirEnrollment.ldap.ldapconn.port=636
auths.instance.UserDirEnrollment.ldap.ldapconn.secureConn=true
```

Finally, you have to modify the /***instance_path***/**ca**/**profiles**/**ca**/***profile_id***.**cfg** file to configure the profile to use the **UserDirEnrollment** auth instance defined in **CS.cfg**, and if appropriate, provide an ACL for authorization based on groups. For example:

```
auth.instance_id=UserDirEnrollment
auths.acl=group="cn=devlab-access,ou=engineering,dc=example,dc=com"
```

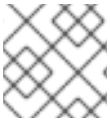
10.2.2. Setting up PIN-Based Enrollment

PIN-based authentication involves setting up PINs for each user in the LDAP directory, distributing those PINs to the users, and then having the users provide the PIN along with their user ID and password when filling out a certificate request. Users are then authenticated both against an LDAP directory using their user ID and password and against the PIN in their LDAP entry. When the user successfully authenticates, the request is automatically processed, and a new certificate is issued.

The Certificate System provides a tool, **setpin**, that adds the necessary schema for PINs to the Directory Server and generates the PINs for each user.

The PIN tool performs the following functions:

- Adds the necessary schema for PINs to the LDAP directory.
- Adds a PIN manager user who has read-write permissions to the PINs that are set up.
- Sets up ACIs to allow for PIN removal once the PIN has been used, giving read-write permissions for PINs to the PIN manager, and preventing users from creating or changing PINs.
- Creates PINs in each user entry.



NOTE

This tool is documented in the *Certificate System Command-Line Tools Guide*.

1. Use the PIN tool to add schema needed for PINs, add PINs to the user entries, and then distribute the PINs to users.
    1. Open the **/usr/share/pki/native-tools/** directory.
    2. Open the **setpin.conf** file in a text editor.
    3. Follow the instructions outlined in the file and make the appropriate changes.

Usually, the parameters which need updated are the Directory Server's host name, Directory Manager's bind password, and PIN manager's password.

4. Run the **setpin** command with its **optfile** option pointing to the **setpin.conf** file.

```
setpin optfile=/usr/share/pki/native-tools/setpin.conf
```

The tool modifies the schema with a new attribute (by default, **pin**) and a new object class (by default, **pinPerson**), creates a **pinmanager** user, and sets the ACI to allow only the **pinmanager** user to modify the **pin** attribute.

5. To generate PINs for specific user entries or to provide user-defined PINs, create an input file with the DNs of those entries listed. For ezample:

```
dn:uid=bjensen,ou=people,dc=example,dc=com
dn:uid=jsmith,ou=people,dc=example,dc=com
dn:jtyler,ou=people,dc=example,dc=com
...
```

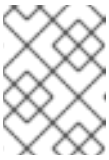
For information on constructing an input file, see the PIN generator chapter in the *Certificate System Command-Line Tools Guide*.

6. Disable setup mode for the **setpin** command. Either comment out the **setup** line or change the value to no.

```
vim /usr/share/pki/native-tools/setpin.conf

setup=no
```

Setup mode creates the required uers and object classes, but the tool will not generate PINs while in setup mode.

7. Run the **setpin** command to create PINs in the directory.



NOTE

Test-run the tool first without the **write** option to generate a list of PINs without actually changing the directory.

For example:

```
setpin host=yourhost port=9446 length=11 input=infile output=outfile write
"binddn=cn=pinmanager,o=example.com" bindpw="password" basedn=o=example.com
"filter=(uid=u*)" hash=sha256
```

WARNING

Do not set the **hash** argument to **none**. Running the **setpin** command with **hash=none** results in the pin being stored in the user LDAP entry as plain text.

8. Use the output file for delivering PINs to users after completing setting up the required authentication method.

   After confirming that the PIN-based enrollment works, deliver the PINs to users so they can use them during enrollment. To protect the privacy of PINs, use a secure, out-of-band delivery method.

2. Set the policies for specific certificates in the certificate profiles to enroll users. See Chapter 3, *Making Rules for Issuing Certificates (Certificate Profiles)* for information about certificate profile policies.

3. Create and configure an instance of the **UidPwdPinDirAuth** authentication plug-in.

   1. Open the CA Console.

      ```
      pkiconsole https://server.example.com:8443/ca
      ```

   2. In the **Configuration** tab, select **Authentication** in the navigation tree.

      The right pane shows the **Authentication Instance** tab, which lists the currently configured authentication instances.

   3. Click **Add**.

      The **Select Authentication Plug-in Implementation** window appears.

   4. Select the **UidPwdPinDirAuth** plug-in module.

   5. Fill in the following fields in the **Authentication Instance Editor** window:

      - **Authentication Instance ID.** Accept the default instance name or enter a new name.
      - **removePin.** Sets whether to remove PINs from the authentication directory after end users successfully authenticate. Removing PINs from the directory restricts users from enrolling more than once, and thus prevents them from getting more than one certificate.
      - **pinAttr.** Specifies the authentication directory attribute for PINs. The **PIN Generator** utility sets the attribute to the value of the **objectclass** parameter in the **setpin.conf** file; the default value for this parameter is **pin**.
      - **dnpattern.** Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN.
      - **ldapStringAttributes.** Specifies the list of LDAP string attributes that should be considered *authentic* for the end entity. Entering values for this parameter is optional.

- **ldapByteAttributes.** Specifies the list of LDAP byte (binary) attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token for use by other modules, such as adding additional information to users' certificates.

  Entering values for this parameter is optional.

- **ldap.ldapconn.host.** Specifies the fully-qualified DNS host name of the authentication directory.

- **ldap.ldapconn.port.** Specifies the TCP/IP port on which the authentication directory listens to requests from the Certificate System.

- **ldap.ldapconn.secureConn.** Specifies the type, SSL or non-SSL, of the port on which the authentication directory listens to requests. Select if this is an SSL port.

- **ldap.ldapconn.version.** Specifies the LDAP protocol version, either **2** or **3**. By default, this is **3**, since all Directory Server versions later than 3.x are LDAPv3.

- **ldap.ldapAuthentication.bindDN.** Specifies the user entry as whom to bind when removing PINs from the authentication directory. Specify this parameter only if the **removePin** checkbox is selected. It is recommended that a separate user entry that has permission to modify only the PIN attribute in the directory be created and used. For example, do not use the Directory Manager's entry because it has privileges to modify the entire directory content.

- **password.** Gives the password associated with the DN specified by the **ldap.ldapauthbindDN** parameter. When saving changes, the server stores the password in the single sign-on password cache and uses it for subsequent start ups. This parameter needs set only if the **removePin** checkbox is selected.

- **ldap.ldapAuthentication.clientCertNickname.** Specifies the nickname of the certificate to use for SSL client authentication to the authentication directory to remove PINs. Make sure that the certificate is valid and has been signed by a CA that is trusted in the authentication directory's certificate database and that the authentication directory's **certmap.conf** file has been configured to map the certificate correctly to a DN in the directory. This is needed for PIN removal only.

- **ldap.ldapAuthentication.authtype.** Specifies the authentication type, basic authentication or SSL client authentication, required in order to remove PINs from the authentication directory.

  - **BasicAuth** specifies basic authentication. With this option, enter the correct values for **ldap.ldapAuthentication.bindDN** and **password** parameters; the server uses the DN from the **ldap.ldapAuthentication.bindDN** attribute to bind to the directory.

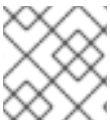
  - **SslClientAuth** specifies SSL client authentication. With this option, set the value of the **ldap.ldapconn.secureConn** parameter to **true** and the value of the **ldap.ldapAuthentication.clientCertNickname** parameter to the nickname of the certificate to use for SSL client authentication.

- **ldap.basedn.** Specifies the base DN for searching the authentication directory; the server uses the value of the **uid** field from the HTTP input (what a user enters in the enrollment form) and the base DN to construct an LDAP search filter.

- **ldap.minConns.** Specifies the minimum number of connections permitted to the authentication directory. The permissible values are **1** to **3**.
- **ldap.maxConns.** Specifies the maximum number of connections permitted to the authentication directory. The permissible values are **3** to **10**.

6. Click **OK**.

4. Customize the enrollment forms by configuring the inputs in the certificate profiles. Include the information that will be needed by the plug-in to authenticate the user. If the default inputs do not contain all of the information that needs to be collected, submit a request created with a third-party tool.



NOTE

pkiconsole is being deprecated.

10.2.3. Using Certificate-Based Authentication

Certificate-based authentication is when a certificate is presented that verifies the identity of the requester and automatically validates and authenticates the request being submitted. This is most commonly used for renewal processes, when the original certificate is presented by the user, server, and application and that certificate is used to authenticate the request.

There are other circumstances when it may be useful to use certificate-based authentication for initially requesting a certificate. For example, tokens may be bulk-loaded with generic certificates which are then used to authenticate the users when they enroll for their user certificates or, alternatively, users can be issued signing certificates which they then use to authenticate their requests for encryption certificates.

The certificate-based authentication module, **SSLclientCertAuth**, is enabled by default, and this authentication method can be referenced in any custom certificate profile.

10.2.4. Configuring Flat File Authentication

A router certificate is enrolled and authenticated using a randomly-generated PIN. The CA uses the **flatFileAuth** authentication module to process a text file which contains the router's authentication credentials.

10.2.4.1. Configuring the flatFileAuth Module

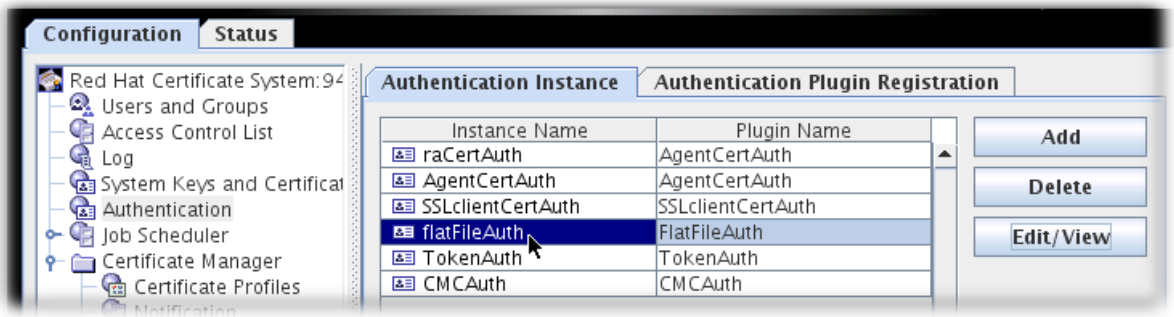
Flat file authentication is already configured for SCEP enrollments, but the location of the flat file and its authentication parameters can be edited.

1. Open the CA Console.

```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, select **Authentication** in the navigation tree.

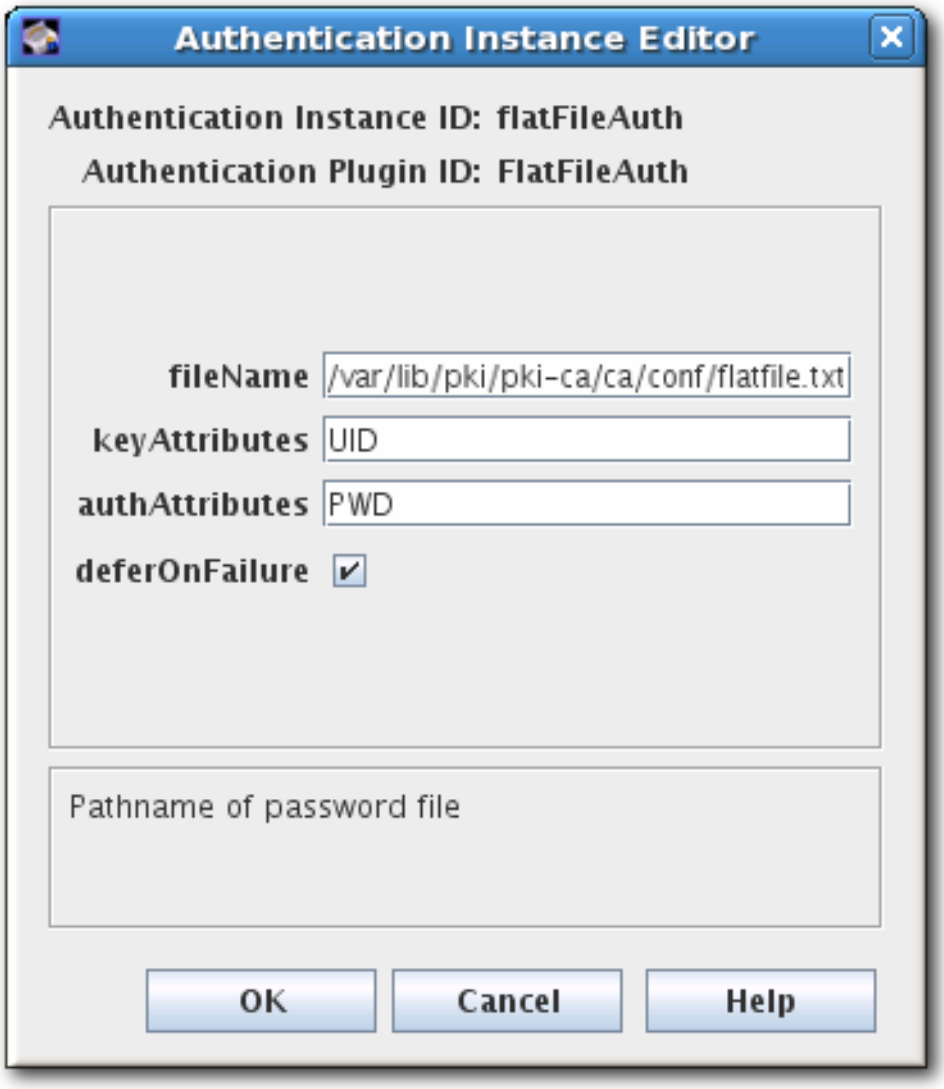
3. Select the **flatFileAuth** authentication module.



4. Click **Edit/View**.

5. To change the file location and name, reset the **fileName** field.

   To change the authentication name parameter, reset the **keyAttributes** value to another value submitted in the SCEP enrollment form, like CN. It is also possible to use multiple name parameters by separating them by commas, like **UID,CN**. To change the password parameter name, reset the **authAttributes** field.



6. Save the edits.

NOTE

pkiconsole is being deprecated.

10.2.4.2. Editing flatfile.txt

The same **flatfile.txt** file is used to authenticate every SCEP enrollment. This file must be manually updated every time a new PIN is issued to a router.

By default, this file is in **/var/lib/pki/pki-ca/ca/conf/** and specifies two parameters per authentication entry, the UID of the site (usually its IP address, either IPv4 or IPv6) and the PIN issued by the router.

```
UID:192.168.123.123
PIN:HU89dj
```

Each entry must be followed by a blank line. For example:

```
UID:192.168.123.123
PIN:HU89dj

UID:12.255.80.13
PIN:fiowIO89

UID:0.100.0.100
PIN:GRIOjisf
```

If the authentication entries are not separated by an empty line, then when the router attempts to authenticate to the CA, it will fail. For example:

```
... flatfile.txt entry ...
UID:192.168.123.123
PIN:HU89dj
UID:12.255.80.13
PIN:fiowIO89

... error log entry ...
[13/Jun/2020:13:03:09][http-9180-Processor24]: FlatFileAuth: authenticating user: finding user from
key: 192.168.123.123
[13/Jun/2020:13:03:09][http-9180-Processor24]: FlatFileAuth: User not found in password file.
```

10.3. CMC AUTHENTICATION PLUG-INS

CMC enrollment allows an enrollment client to use a CMC Authentication plug-in for authentication, by which the certificate request is either pre-signed with an agent certificate or a user certificate, depending on the plug-in. The Certificate Manager automatically issues certificates when a CMC request signed with a valid certificate is received.

The CMC authentication plug-ins also provide CMC revocation for the client. CMC revocation allows the client to have the certificate request either signed by the agent certificate, or a verifiable user that owns the certificate, and then send such a request to the Certificate Manager. The Certificate Manager automatically revokes certificates when a CMC revocation request signed with a valid certificate is received.

Certificate System provides the following CMC authentication plug-ins:

CMCAuth

Use this plug-in when a CA agent signs CMC requests.

To use the **CMCAuth** plug-in, set the following in the enrollment profile:

```
auth.instance_id=CMCAuth
```

By default, the following enrollment profiles use the **CMCAuth** plug-in:

- For system certificates:
  - **caCMCauditSigningCert**
  - **caCMCcaCert**
  - **caCMCECserverCert**
  - **caCMCECsubsystemCert**
  - **caCMCECUserCert**
  - **caCMCkraStorageCert**
  - **caCMCkraTransportCert**
  - **caCMCocspCert**
  - **caCMCserverCert**
  - **caCMCsubsystemCert**
- For user certificates:
  - **caCMCUserCert**
  - **caECFullCMCUserCert**
  - **caFullCMCUserCert**

CMCUserSignedAuth

Use this plug-in when users submit signed or SharedSecret-based CMC requests.

To use the **CMCUserSignedAuth** plug-in, set the following in the enrollment profile:

```
auth.instance_id=CMCUserSignedAuth
```

A user-signed CMC request must be signed by the user's certificate which contains the same **subjectDN** attribute as the requested certificate. You can only use a user-signed CMC request if the user already obtained a signing certificate which can be used to prove the user's identity for other certificates.

A SharedSecret-based CMC request means that the request was signed by the private key of the request itself. In this case, the CMC request must use the Shared Secret mechanism for authentication. A SharedSecret-based CMC request is typically used to obtain the user's first signing certificate, which is later used to obtain other certificates. For further details, see Section 10.4, “CMC SharedSecret Authentication”.

By default, the following enrollment profiles use the **CMCUserSignedAuth** plug-in:

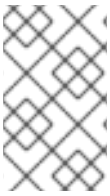
- **caFullCMCUserSignedCert**
- **caECFullCMCUserSignedCert**
- **caFullCMCSharedTokenCert**
- **caECFullCMCSharedTokenCert**

10.4. CMC SHAREDSECRET AUTHENTICATION

Use the Shared Secret feature to enable users to send unsigned CMC requests to the server. For example, this is necessary if a user wants to obtain the first signing certificate. This signing certificate can later be used to sign other certificates of this user.

10.4.1. Creating a Shared Secret Token

The *The Shared Secret Workflow* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* describes the workflow when using a Shared Secret Token. Depending on the situation, either an end entity user or an administrator creates the Shared Secret Token.



NOTE

To use the shared secret token, Certificate System must use an RSA issuance protection certificate. For details, see Enabling the CMC Shared Secret Feature section located in RHCS Planning, Installation, and Deployment Guide.

To create a Shared Secret Token, enter:

```
# CMCSharedToken -d /home/user_name/.dogtag/ -p NSS_password \
    -s "CMC_enrollment_password" -o /home/user_name/CMC_shared_token.b64 \
    -n "issuance_protection_certificate_nickname"
```

If you use an HSM, additionally pass the **-h *token_name*** option to the command to set the HSM security token name.

For further details about the **CMCSharedToken** utility, see the CMCSharedToken(8) man page.

NOTE

The generated token is encrypted and only the user who generated knows the password. If a CA administrator generates the token for a user, the administrator must provide the password to the user using a secure way.

After creating the Shared Token, an administrator must add the token to a user or certificate record. For details, see Section 10.4.2, “Setting a CMC Shared Secret”.

10.4.2. Setting a CMC Shared Secret

Depending on the planned action, an administrator must store a Shared Secret Token after generating it in the LDAP entry of the user or certificate.

For details about the workflow and when to use a Shared Secret, see the *The Shared Secret Workflow* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

10.4.2.1. Adding a CMC Shared Secret to a User Entry for Certificate Enrollment

To use the Shared Secret Token for certificate enrollment, store it as an administrator in the LDAP entry of the user:

```
# ldapmodify -D "cn=Directory Manager" -W -p 389 -h server.example.com -x

dn: uid=user_name,ou=People,dc=example,dc=com
changetype: modify
replace: shrTok
shrTok: base64-encoded_token
```

10.4.2.2. Adding a CMC Shared Secret to a Certificate for Certificate Revocations

To use the Shared Secret Token for certificate revocations, store it as an administrator in the LDAP entry of the certificate to be revoked:

```
# ldapmodify -D "cn=Directory Manager" -W -p 389 -h server.example.com -x

dn: cn=certificate_id,ou=certificateRepository,ou=ca,o=pki-tomcat-CA
changetype: modify
replace: shrTok
shrTok: base64-encoded_token
```

10.5. TESTING ENROLLMENT

For information on testing enrollment through the profiles, see Chapter 3, *Making Rules for Issuing Certificates (Certificate Profiles)*. To test whether end users can successfully enroll for a certificate using the authentication method set:

1. Open the end-entities page.

   ```
   https://server.example.com:8443/ca/ee/ca
   ```

2. In the **Enrollment** tab, open the customized enrollment form.

3. Fill in the values, and submit the request.

4. Enter the password to the key database when prompted.

5. When the correct password is entered, the client generates the key pair.

   Do not interrupt the key-generation process. Upon completion of the key generation, the request is submitted to the server to issue the certificate. The server subjects the request to the certificate profile and issues the certificate only if the request meets all the requirements.

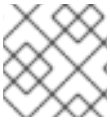
   When the certificate is issued, install the certificate in the browser.

6. Verify that the certificate is installed in the browser's certificate database.

7. If PIN-based directory authentication was configured with PIN removal, re-enroll for another certificate using the same PIN. The request should be rejected.

10.6. REGISTERING CUSTOM AUTHENTICATION PLUG-INS

Custom authentication plug-in modules can be registered through the CA Console. Authentication plug-in modules can also be deleted through the CA Console. Before deleting a module, delete instances that are based on that module.



NOTE

For writing custom plug-ins, refer to the Authentication Plug-in Tutorial.

1. Create the custom authentication class. For this example, the custom authentication plug-in is called **UidPwdDirAuthenticationTestms.java**.
2. Compile the new class.

```
javac -d . -classpath $CLASSPATH UidPwdDirAuthenticationTestms.java
```

3. Create a directory in the CA's **WEB-INF** web directory to hold the custom classes, so that the CA can access them for the enrollment forms.

```
mkdir /usr/share/pki/ca/webapps/ca/WEB-INF/classes
```

4. Copy the new plug-in files into the new **classes** directory, and set the owner to the Certificate System system user (**pkiuser**).

```
cp -pr com /usr/share/pki/ca/webapps/ca/WEB-INF/classes

chown -R pkiuser:pkiuser /usr/share/pki/ca/webapps/ca/WEB-INF/classes
```

5. Log into the console.

```
pkiconsole https://server.example.com:8443/ca
```

6. Register the plug-in.
    1. In the **Configuration** tab, click **Authentication** in the navigation tree.
    2. In the right pane, click the **Authentication Plug-in Registration** tab.

       The tab lists modules that are already registered.
    3. To register a plug-in, click **Register**.

       The **Register Authentication Plug-in Implementation** window appears.
    4. Specify which module to register by filling in the two fields:
        - **Plugin name.** The name for the module.
        - **Class name.** The full name of the class for this module. This is the path to the

implementing Java™ class. If this class is part of a package, include the package name. For example, to register a class named **customAuth** in a package named **com.customplugins**, the class name is **com.customplugins.customAuth**.

7. After registering the module, add the module as an active authentication instance.

    1. In the **Configuration** tab, click **Authentication** in the navigation tree.
    2. In the right pane, click the **Authentication Instance** tab.
    3. Click **Add**.
    4. Select the custom module, **UidPwdDirAuthenticationTestms.java**, from the list to add the module. Fill in the appropriate configuration for the module.

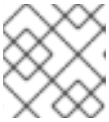
NOTE

pkiconsole is being deprecated.

8. Create a new end-entity enrollment form to use the new authentication module.

```
cd /var/lib/pki/pki-tomcat/ca/profiles/ca

cp -p caDirUserCert.cfg caDirUserCertTestms.cfg

vi caDirUserCertTestms.cfg

desc=Test ms - This certificate profile is for enrolling user certificates with directory-based
authentication.
visible=true
enable=true
enableBy=admin
name=Test ms - Directory-Authenticated User Dual-Use Certificate Enrollment
auth.instance_id=testms
...
```

9. Add the new profile to the CA's **CS.cfg** file.



NOTE

Back up the **CS.cfg** file before editing it.

```
vim /var/lib/pki/instance-name/ca/conf/CS.cfg

profile.list=caUserCert,caDualCert,caSignedLogCert,caTPSCert,caRARouterCert,caRouterCer
t,caServerCert,caOtherCert,caCACert,caInstallCACert,caRACert,caOCSPCert,caTransportCe
rt,caDirUserCert,caAgentServerCert,caAgentFileSigning,caCMCUserCert,caFullCMCUserCert
,caSimpleCMCUserCert,caTokenDeviceKeyEnrollment,caTokenUserEncryptionKeyEnrollment,
caTokenUserSigningKeyEnrollment,caTempTokenDeviceKeyEnrollment,caTempTokenUserEn
cryptionKeyEnrollment,caTempTokenUserSigningKeyEnrollment,caAdminCert,caInternalAuthS
erverCert,caInternalAuthTransportCert,caInternalAuthKRAstorageCert,caInternalAuthSubsyste
mCert,caInternalAuthOCSPCert,DomainController,caDirUserCertTestms
...
```

```
profile.caDirUserCertTestms.class_id=caEnrollImpl
profile.caDirUserCertTestms.config=/var/lib/pki/pki-tomcat/ca/profiles/ca/caDirUserCertTestms.cfg
```

10. Restart the CA.

```
pki-server restart instance_name
```

10.7. MANUALLY REVIEWING THE CERTIFICATE STATUS USING THE COMMAND LINE

To review certificate requests, ensure that you are authenticated as an agent with proper permissions to approve certificate requests. For details about configuring the **pki** command-line interface, see Section 2.5.1.1, “pki CLI Initialization”.

To review the requests:

1. Display the list of pending certificate requests:

```
$ pki agent_authentication_parameters ca-cert-request-find --status pending
```

This command lists all pending certificate requests.

2. Download a particular certificate request:

```
$ pki agent_authentication_parameters ca-cert-request-review id --file request.xml
```

3. Open the ***request.xml*** file in an editor or a separate terminal, and review the contents of the request to ensure it is legitimate. Then answer the prompt: if the request is valid, answer "**approve** and press **Enter**. If the request is invalid, answer **reject** and press **Enter**. Organizations can subscribe semantic differences to **reject** and **cancel**; both result in no certificate being issued.

10.8. MANUALLY REVIEWING THE CERTIFICATE STATUS USING THE WEB INTERFACE

1. Open the following URL in a web browser:

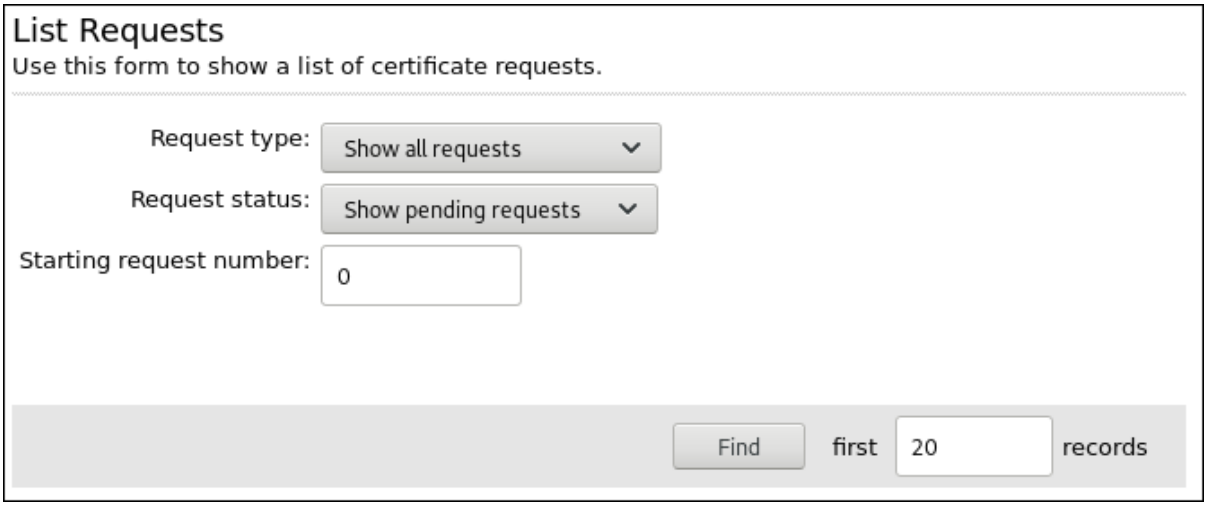
```
https://server_host_name:8443/ca/agent/ca
```

2. Authenticate as an agent. For information about authenticating as a user and configuring your browser, see Section 2.4.1, “Browser Initialization”.

3. On the sidebar on the left, click the **List requests** link.

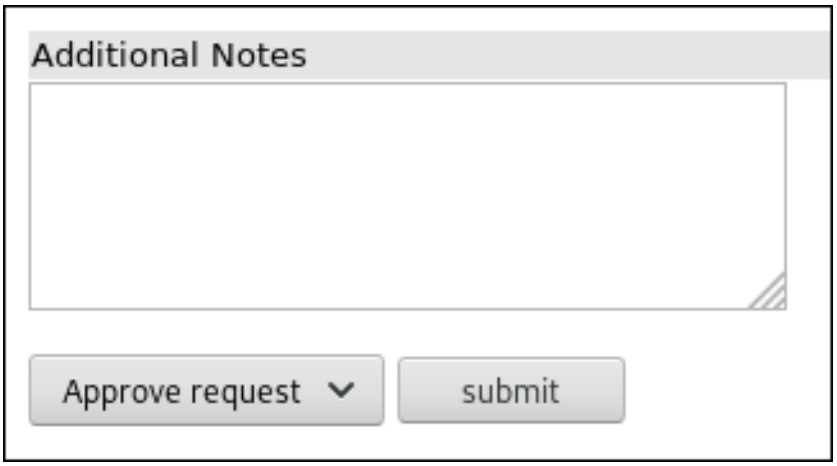
4. Filter the requests by selecting **Show all requests** for **Request type** and **Show pending requests** for **Request status**.

5. Click **Find** in the lower right corner.



6. The results page lists all pending requests waiting for review. Click on the request number to review a request.

7. Review the request information and ensure that it is a legitimate request. If necessary, modify the policy information to correct any mistakes or make any desired changes to the certificate, such as changing the **not valid after** field. Optionally, leave an additional note.



The drop down menu includes several review status updates. Select **Approve request** to approve the request or **Reject request** to deny it, and click **Submit**. Organizations can subscribe semantic differences to **Reject request** and **Cancel Request**; both result in no certificate being issued.

CHAPTER 11. AUTHORIZATION FOR ENROLLING CERTIFICATES (ACCESS EVALUATORS)

This chapter describes the authorization mechanism using access evaluators.

11.1. AUTHORIZATION MECHANISM

In addition to the *authentication* mechanism, each enrollment profile can be configured to have its own *authorization* mechanism. The authorization mechanism is executed only after a successful authentication.

The authorization mechanism is provided by the Access Evaluator plug-in framework. Access evaluators are pluggable classes that are used for evaluating access control instructions (ACI) entries. The mechanism provides an *evaluate* method that takes a predefined list of arguments (that is, **type**, **op**, **value**), evaluates an expression such as **group='Certificate Manager Agents'** and returns a boolean depending on the result of evaluation.

11.2. DEFAULT EVALUATORS

Red Hat Certificate System provides four default evaluators. The following entries are listed by default in the **CS.cfg** file:

```
accessEvaluator.impl.group.class=com.netscape.cms.evaluators.GroupAccessEvaluator
accessEvaluator.impl.ipaddress.class=com.netscape.cms.evaluators.IPAddressAccessEvaluator
accessEvaluator.impl.user.class=com.netscape.cms.evaluators.UserAccessEvaluator
accessEvaluator.impl.user_origreq.class=com.netscape.cms.evaluators.UserOrigReqAccessEvaluator
```

The **group** access evaluator evaluates the group membership properties of a user. For example, in the following enrollment profile entry, only the CA agents are allowed to go through enrollment with that profile:

```
authz.acl=group="Certificate Manager Agents"
```

The **ipaddress** access evaluator evaluates the IP address of the requesting subject. For example, in the following enrollment profile entry, only the host bearing the specified IP address can go through enrollment with that profile:

```
authz.acl=ipaddress="a.b.c.d.e.f"
```

The **user** access evaluator evaluates the user ID for exact match. For example, in the following enrollment profile entry, only the user matching the listed user is allowed to go through enrollment with that profile:

```
authz.acl=user="bob"
```

The **user_origreq** access evaluator evaluates the authenticated user against a previous matching request for equality. This special evaluator is designed specifically for renewal purpose to make sure the user requesting the renewal is the same user that owns the original request. For example, in the following renewal enrollment profile entry, the UID of the authenticated user must match the UID of the user requesting the renewal:

```
authz.acl=user_origreq="auth_token.uid"
```

New evaluators can be written in the current framework and can be registered through the CS console. The default evaluators can be used as templates to expand and customize into more targeted plug-ins.

CHAPTER 12. USING AUTOMATED NOTIFICATIONS

The Certificate System can be configured to send automatic email notifications to end users when certificates are issued or revoked or to an agent when a new request has arrived in the agent request queue. This chapter describes automated notifications and details how to enable, configure, and customize the notification email messages that are sent.

NOTE

Because of the types of notifications that can be sent, only Certificate Managers have the ability to be configured for notifications; this option is not available on the other subsystems.

12.1. ABOUT AUTOMATED NOTIFICATIONS FOR THE CA

Automated notifications are email messages sent when a specified event occurs. The system uses listeners that monitor the system to determine when a particular event has occurred; when the event happens, then the system is triggered to send an email to the configured recipient. Each type of notification uses a template, either in plain text or HTML, to construct the notification message. The template contains text and tokens that are expanded to fill in the correct information for a particular event. The messages can be customized by changing the text and tokens contained in the templates. The HTML templates can also be customized for different appearances and formatting.

12.1.1. Types of Automated Notifications

There are three types of automated notifications:

- *Certificate Issued*.

  A notification message is automatically sent to users who have been issued certificates. A rejection message is sent to a user if the user's certificate request is rejected.

- *Certificate Revocation*.

  A notification message is automatically sent to users when the user certificate is revoked.

- *Request in Queue*.

  A notification message is automatically sent to one or more agents when a request enters the agent request queue, using the email addresses set for the agent. This notification type sends an email every time a message enters the queue. For more information about the request in queue job, see Section 13.1.2.2, “requestInQueueNotifier (RequestInQueueJob)”.

  There is also a job that sends a notification to agents about the status of the queue, which includes a summary of the queue status at certain intervals.

12.1.2. Determining End-Entity Email Addresses

The notification system determines the email address of an end entity by checking first the certificate request or revocation request, then the subject name of the certificate, and last the Subject Alternative Name extension of the certificate, if the certificate contains this extension. If an email address cannot be found, the notification is sent to the email address specified in the **Sender's Email Address** field of the **Notification** panel.

12.2. SETTING UP AUTOMATED NOTIFICATIONS FOR THE CA

12.2.1. Setting up Automated Notifications in the Console

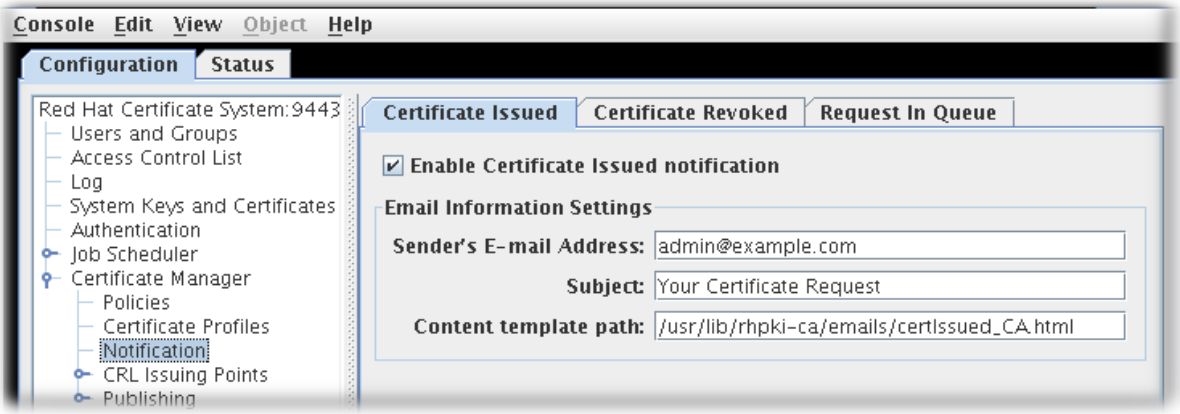
1. Open the Certificate Manager Console.

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. Open the **Configuration** tab.

3. Open the **Certificate Manager** heading in the navigation tree on the left. Then select **Notification**.

   The **Notification** tabs appear in the right side of the window.



4. Notifications can be sent for three kinds of events: newly-issued certificates, revoked certificates, and new certificate requests. To send a notification for any event, select the tab, check the **Enable** checkbox, and specify information in the following fields:

   - **Sender's E-mail Address**. Type the sender's full email address of the user who is notified of any delivery problems.
   - **Recipient's E-mail Address**. These are the email addresses of the agents who will check the queue. To list more than one recipient, separate the email addresses with commas. *For new requests in queue only.*
   - **Subject**. Type the subject title for the notification.
   - **Content template path**. Type the path, including the filename, to the directory that contains the template to use to construct the message content.

5. Click **Save**.

NOTE

Make sure the mail server is set up correctly. See Section 12.4, “Configuring a Mail Server for Certificate System Notifications”.

6. Customize the notification message templates. See Section 12.3, “Customizing Notification Messages” for more information.

7. Test the configuration. See Section 12.2.3, "Testing Configuration".

NOTE

pkiconsole is being deprecated.

12.2.2. Configuring Specific Notifications by Editing the CS.cfg File

1. Stop the CA subsystem.

```
pki-server stop instance_name
```

2. Open the **CS.cfg** file for that instance. This file is in the instance's **conf/** directory.

3. Edit all of the configuration parameters for the notification type being enabled.

   For certificate issuing notifications, there are four parameters:

```
ca.notification.certIssued.emailSubject
ca.notification.certIssued.emailTemplate
ca.notification.certIssued.enabled
ca.notification.certIssued.senderEmail
```

   For certificate revocation notifications, there are four parameters:

```
ca.notification.certRevoked.emailSubject
ca.notification.certRevoked.emailTemplate
ca.notification.certRevoked.enabled
ca.notification.certRevoked.senderEmail
```

   For certificate request notifications, there are five parameters:

```
ca.notification.requestInQ.emailSubject
ca.notification.requestInQ.emailTemplate
ca.notification.requestInQ.enabled
ca.notification.requestInQ.recipientEmail
ca.notification.requestInQ.senderEmail
```

   The parameters for the notification messages are explained in Section 12.2, "Setting up Automated Notifications for the CA".

4. Save the file.

5. Restart the CA instance.

```
pki-server start instance_name
```

6. If a job has been created to send automated messages, check that the mail server is correctly configured. See Section 12.4, "Configuring a Mail Server for Certificate System Notifications" .

7. The messages that are sent automatically can be customized; see Section 12.3, "Customizing Notification Messages" for more information.

12.2.3. Testing Configuration

To test whether the subsystem sends email notifications as configured, do the following:

1. Change the email address in the notification configuration for the request in queue notification to an accessible agent or administrator email address.

2. Open the end-entities page, and request a certificate using the agent-approved enrollment form.

   When the request gets queued for agent approval, a request-in-queue email notification should be sent. Check the message to see if it contains the configured information.

3. Log into the agent interface, and approve the request.

   When the server issues a certificate, the user receive a certificate-issued email notification to the address listed in the request. Check the message to see if it has the correct information.

4. Log into the agent interface, and revoke the certificate.

   The user email account should contain an email message reading that the certificate has been revoked. Check the message to see if it has the correct information.

12.3. CUSTOMIZING NOTIFICATION MESSAGES

The email notifications are constructed using a template for each type of message. This allows messages to be informative, easily reproducible, and easily customizable. The CA uses templates for its notification messages. Separate templates exist for HTML and plain text messages.

12.3.1. Customizing CA Notification Messages

Each type of CA notification message has an HTML template and a plain text template associated with it. Messages are constructed from text, tokens, and, for the HTML templates, HTML markup. *Tokens* are variables, identified by a dollar sign (**$**), in the message that are replaced by the current value when the message is constructed. See Table 12.3, “Notification Variables” for a list of available tokens.

The contents of any message type can be modified by changing the text and tokens in the message template. The appearance of the HTML messages can be changed by modifying the HTML commands in the HTML message template.

The default text version of the certificate-issuance-notification message is as follows:

```
Your certificate request has been processed successfully.
SubjectDN= $SubjectDN
IssuerDN= $IssuerDN
notAfter= $NotAfter
notBefore= $NotBefore
Serial Number= 0x$HexSerialNumber
To get your certificate, please follow this URL:
https://$HttpHost:$HttpPort/displayBySerial?op=displayBySerial&
 serialNumber=$SerialNumber
Please contact your admin if there is any problem.
And, of course, this is just a \$SAMPLE\$ email notification form.
```

This template can be customized as desired, by rearranging, adding, or removing tokens and text, as shown:

```
THE EXAMPLE COMPANY CERTIFICATE ISSUANCE CENTER
Your certificate has been issued!
You can pick up your new certificate at the following website:
https://$HttpHost:$HttpPort/displayBySerial?op=displayBySerial&
 serialNumber=$SerialNumber
This certificate has been issued with the following information:
Serial Number= 0x$HexSerialNumber
Name of Certificate Holder = $SubjectDN
Name of Issuer = $IssuerDN
Certificate Expiration Date = $NotAfter
Certificate Validity Date = $NotBefore
Contact IT by calling X1234, or going to the IT website http://IT
 if you have any problems.
```

Notification message templates are located in the **/var/lib/pki/*instance_name*/ca/emails** directory.

The name and location of these messages can be changed; make the appropriate changes when configuring the notification. All template names can be changed except for the certificate rejected templates; these names must remain the same. The templates associated with certificate issuance and certificate rejection must be located in the same directory and must use the same extension.

Table 12.1, "Notification Templates" lists the default template files provided for creating notification messages. Table 12.2, "Job Notification Email Templates" lists the default template files provided for creating job summary messages.

Table 12.1. Notification Templates

Filename	Description
certIssued_CA	Template for plain text notification emails to end entities when certificates are issued.
certIssued_CA.html	Template for HTML-based notification emails to end entities when certificates are issued.
certRequestRejected.html	Template for HTML-based notification emails to end entities when certificate requests are rejected.
certRequestRevoked_CA	Template for plain text notification emails to end entities when a certificate is revoked.
certRequestRevoked_CA.html	Template for HTML-based notification emails to end entities when a certificate is revoked.
reqInQueue_CA	Template for plain text notification emails to agents when a request enters the queue.
reqInQueue_CA.html	Template for HTML-based notification emails to agents when a request enters the queue.

Table 12.2. Job Notification Email Templates

Filename	Description
rnJob1.txt	Template for formulating the message content sent to end entities to inform them that their certificates are about to expire and that the certificates should be renewed or replaced before they expire.
rnJob1Summary.txt	Template for constructing the summary report to be sent to agents and administrators. Uses the **rnJob1Item.txt** template to format items in the message.
rnJob1Item.txt	Template for formatting the items included in the summary report.
riq1Item.html	Template for formatting the items included in the summary table, which is constructed using the **riq1Summary.html** template.
riq1Summary.html	Template for formulating the report or table that summarizes how many requests are pending in the agent queue of a Certificate Manager.
publishCerts	Template for the report or table that summarizes the certificates to be published to the directory. Uses the **publishCertsItem.html** template to format the items in the table.
publishCertsItem.html	Template for formatting the items included in the summary table.
ExpiredUnpublishJob	Template for the report or table that summarizes removal of expired certificates from the directory. Uses the **ExpiredUnpublishJobItem** template to format the items in the table.
ExpiredUnpublishJobItem	Template for formatting the items included in the summary table.

Table 12.3, "Notification Variables" lists and defines the variables that can be used in the notification message templates.

Table 12.3. Notification Variables

Token	Description

Token	Description
$CertType	Specifies the type of certificate; these can be any of the following: • TLS client (**client**) • TLS server (**server**) • CA signing certificate (**ca**) • other (**other**).
$ExecutionTime	Gives the time the job was run.
$HexSerialNumber	Gives the serial number of the certificate that was issued in hexadecimal format.
$HttpHost	Gives the fully qualified host name of the Certificate Manager to which end entities should connect to retrieve their certificates.
$HttpPort	Gives the Certificate Manager's end-entities (non-TLS) port number.
$InstanceID	Gives the ID of the subsystem that sent the notification.
$IssuerDN	Gives the DN of the CA that issued the certificate.
$NotAfter	Gives the end date of the validity period.
$NotBefore	Gives the beginning date of the validity period.
$RecipientEmail	Gives the email address of the recipient.
$RequestId	Gives the request ID.
$RequestorEmail	Gives the email address of the requester.
$RequestType	Gives the type of request that was made.
$RevocationDate	Gives the date the certificate was revoked.
$SenderEmail	Gives the email address of the sender; this is the same as the one specified in the **Sender's E-mail Address** field in the notification configuration.

Token	Description
$SerialNumber	Gives the serial number of the certificate that has been issued; the serial number is displayed as a hexadecimal value in the resulting message.
$Status	Gives the request status.
$SubjectDN	Gives the DN of the certificate subject.
$SummaryItemList	Lists the items in the summary notification. Each item corresponds to a certificate the job detects for renewal or removal from the publishing directory.
$SummaryTotalFailure	Gives the total number of items in the summary report that failed.
$SummaryTotalNum	Gives the total number of certificate requests that are pending in the queue or the total number of certificates to be renewed or removed from the directory in the summary report.
$SummaryTotalSuccess	Shows how many of the total number of items in the summary report succeeded.

12.4. CONFIGURING A MAIL SERVER FOR CERTIFICATE SYSTEM NOTIFICATIONS

The notifications and jobs features use the mail server configured in the Certificate System CA instances to send notification messages.

Before you start setting up the mail server, ensure the following parameters are specified in the **CS.cfg** configuration file:

```
smtp.host=localhost
smtp.port=25
```

Set up a mail server by doing the following:

1. Open the CA subsystem administrative console. For example:

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab, highlight the instance name at the top, and select the **SMTP** tab.

3. Supply the server name and port number of the mail server.

   The server name is the fully qualified DNS hostname of the machine on which the mail server is installed, such as **mail.example.com**. By default, the hostname of the mail server is **localhost** instead of the actual hostname.

The default port number on which the SMTP mail server listens is **25**.

4. Click **Save**.

NOTE

pkiconsole is being deprecated.

12.5. CREATING CUSTOM NOTIFICATIONS FOR THE CA

It can be possible to create custom notification functions to handle other PKI operations, such as token enrollments, by editing existing email notifications plug-ins for the Certificate System CA. Before attempting to create or use custom notification plug-ins, contact Red Hat support services.

CHAPTER 13. SETTING AUTOMATED JOBS

The Certificate System provides a customizable Job Scheduler that supports various mechanisms for scheduling **cron** jobs. This chapter explains how to configure Certificate System to use specific job plug-in modules for accomplishing jobs.

13.1. ABOUT AUTOMATED JOBS

The Certificate Manager Console includes a *Job Scheduler* option that can execute specific jobs at specified times. The Job Scheduler is similar to a traditional Unix **cron** daemon; it takes registered **cron** jobs and executes them at a pre-configured date and time. If configured, the scheduler checks at specified intervals for jobs waiting to be executed; if the specified execution time has arrived, the scheduler initiates the job automatically.

Jobs are implemented as Java™ classes, which are then registered with Certificate System as plug-in modules. One implementation of a job module can be used to configure multiple instances of the job. Each instance must have a unique name (an alphanumeric string with no spaces) and can contain different input parameter values to apply to different jobs.

13.1.1. Setting up Automated Jobs

The automated jobs feature is set up by doing the following:

- Enabling and configuring the Job Scheduler; see Section 13.2, "Setting up the Job Scheduler" for more information.
- Enabling and configuring the job modules and setting preferences for those job modules; see Section 13.3, "Setting up Specific Jobs" for more information.
- Customizing the email notification messages sent with these jobs by changing the templates associated with the types of notification. The message contents are composed of both plain text messages and HTML messages; the appearance is modified by changing the HTML templates. See Section 12.3.1, "Customizing CA Notification Messages" for more information.

13.1.2. Types of Automated Jobs

The types of automated jobs are **RenewalNotificationJob**, **RequestInQueueJob**, **PublishCertsJob**, and **UnpublishExpiredJob**. One instance of each job type is created when Certificate System is deployed.

13.1.2.1. certRenewalNotifier (RenewalNotificationJob)

The **certRenewalNotifier** job checks for certificates that are about to expire in the internal database. When it finds one, it automatically emails the certificate's owner and continues sending email reminders for a configured period of time or until the certificate is replaced. The job collects a summary of all renewal notifications and mails the summary to the configured agents or administrators.

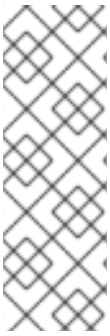
The job determines the email address to send the notification using an email resolver. By default, the email address is found in the certificate itself or in the certificate's associated enrollment request.

13.1.2.2. requestInQueueNotifier (RequestInQueueJob)

The **requestInQueueNotifier** job checks the status of the request queue at pre-configured time intervals. If any deferred enrollment requests are waiting in the queue, the job constructs an email message summarizing its findings and sends it to the specified agents.

13.1.2.3. publishCerts (PublishCertsJob)

The **publishCerts** job checks for any new certificates that have been added to the publishing directory that have not yet been published. When these new certificates are added, they are automatically published to an LDAP directory or file by the **publishCerts** job.



NOTE

Most of the time, publishers immediately publish any certificates that are created matching their rules to the appropriate publishing directory.

If a certificate is successfully published when it is created, then the **publishCerts** job will not re-publish the certificate. Therefore, the new certificate will not be listed in the job summary report, since the summary only lists certificates published by the **publishCerts** job.

13.1.2.4. unpublishExpiredCerts (UnpublishExpiredJob)

Expired certificates are not automatically removed from the publishing directory. If a Certificate Manager is configured to publish certificates to an LDAP directory, over time the directory will contain expired certificates.

The **unpublishExpiredCerts** job checks for certificates that have expired and are still marked as **published** in the internal database at the configured time interval. The job connects to the publishing directory and deletes those certificates; it then marks those certificates as **unpublished** in the internal database. The job collects a summary of expired certificates that it deleted and mails the summary to the agents or administrators specified by the configuration.

NOTE

This job automates removing expired certificates from the directory. Expired certificates can also be removed manually; for more information on this, see Section 9.12, “Updating Certificates and CRLs in a Directory”.

13.2. SETTING UP THE JOB SCHEDULER

The Certificate Manager can execute a job only if the Job Scheduler is enabled. The job settings, such as enabling the job schedule, setting the frequency, and enabling the job modules, can be done through the Certificate System CA Console or through editing the **CS.cfg** file.

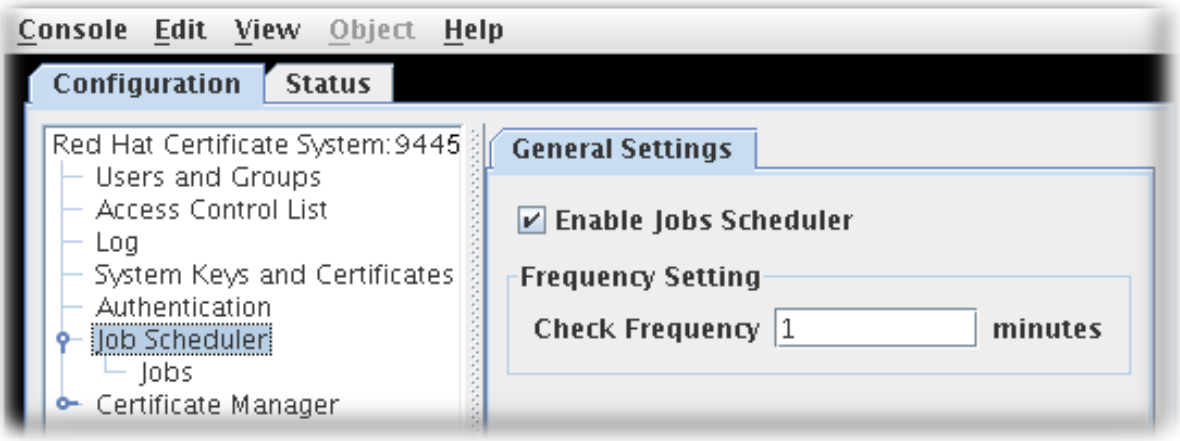
To turn the Job Scheduler on:

1. Open the Certificate Manager Console.

   ```
   pkiconsole https://server.example.com:8443/ca
   ```

2. In the **Configuration** tab navigation tree, click **Job Scheduler**.

   This opens the **General Settings** tab, which shows whether the Job Scheduler is currently enabled.



3. Click the **Enable Jobs Schedule** checkbox to enable or disable the Job Scheduler.

   Disabling the Job Scheduler turns off all the jobs.

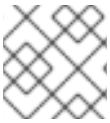
4. Set the frequency which the scheduler checks for jobs in the **Check Frequency** field.

   The frequency is how often the Job Scheduler daemon thread wakes up and calls the configured jobs that meet the **cron** specification. By default, it is set to one minute.

NOTE

The window for entering this information may be too small to see the input. Drag the corners of the Certificate Manager Console to enlarge the entire window.

5. Click **Save**.



NOTE

pkiconsole is being deprecated.

13.3. SETTING UP SPECIFIC JOBS

Automated jobs can be configured through the Certificate Manager Console or by editing the configuration file directory. It is recommended that these changes be made through the Certificate Manager Console.

13.3.1. Configuring Specific Jobs Using the Certificate Manager Console

NOTE

pkiconsole is being deprecated.

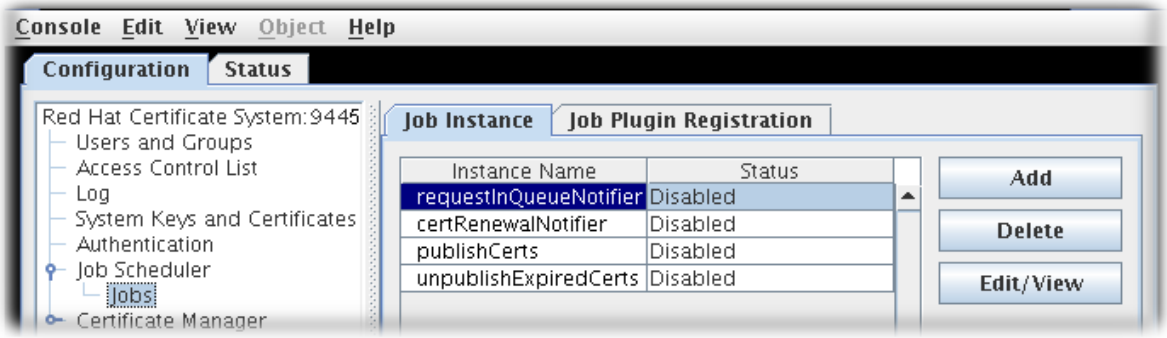
To enable and configure an automated job using the Certificate Manager Console:

1. Open the Certificate Manager Console.

```
pkiconsole https://server.example.com:8443/ca
```

2. Confirm that the Jobs Scheduler is enabled. See Section 13.2, "Setting up the Job Scheduler" for more information.

3. In the **Configuration** tab, select **Job Scheduler** from the navigation tree. Then select **Jobs** to open the **Job Instance** tab.



Select the job instance from the list, and click **Edit/View**.

The **Job Instance Editor** opens, showing the current job configuration.

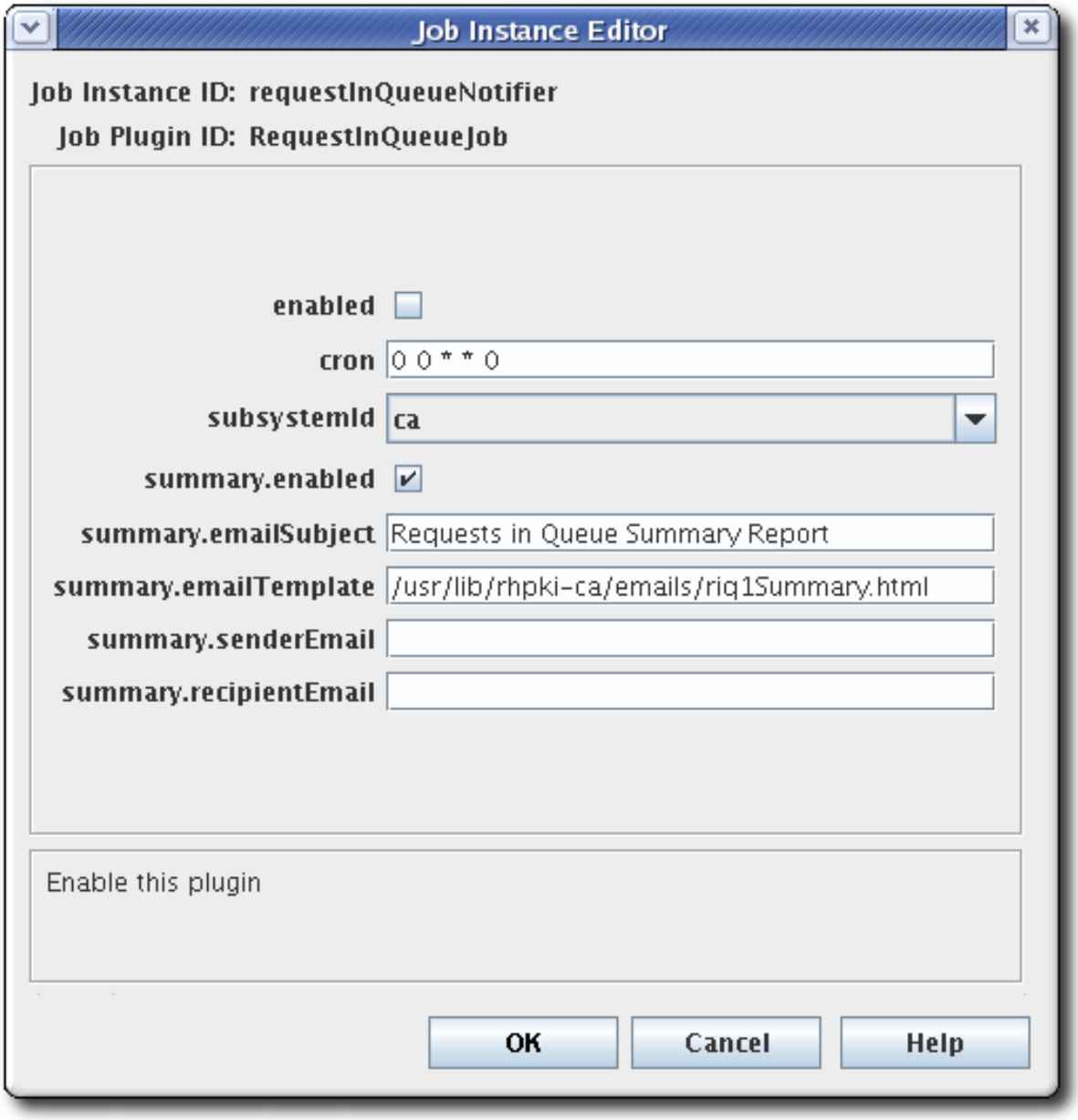


Figure 13.1. Job Configuration

4. Select **enabled** to turn on the job.
5. Set the configuration settings by specifying them in the fields for this dialog.
    - For **certRenewalNotifier**, see Section 13.3.3, "Configuration Parameters of certRenewalNotifier".
    - For **requestInQueueNotifier**, see Section 13.3.4, "Configuration Parameters of requestInQueueNotifier".
    - For **publishCerts**, see Section 13.3.5, "Configuration Parameters of publishCerts".
    - For **unpublishExpiredCerts**, see Section 13.3.6, "Configuration Parameters of unpublishExpiredCerts".
    - For more information about setting the **cron** time frequencies, see Section 13.3.7, "Frequency Settings for Automated Jobs".
6. Click **OK**.

7. Click **Refresh** to view any changes in the main window.

8. If the job is configured to send automatic messages, check that a mail server is set up correctly. See Section 12.4, "Configuring a Mail Server for Certificate System Notifications".

9. Customize the email message text and appearance.

13.3.2. Configuring Jobs by Editing the Configuration File

1. Ensure that the Jobs Scheduler is enabled and configured; see Section 13.2, "Setting up the Job Scheduler".

2. Stop the CA subsystem instance.

```
pki-server stop instance_name
```

3. Open the **CS.cfg** file for that server instance in a text editor.

4. Edit all of the configuration parameters for the job module being configured.

   - To configure the **certRenewalNotifier** job, edit all parameters that begin with **jobsScheduler.job.certRenewalNotifier**; see Section 13.3.3, "Configuration Parameters of certRenewalNotifier".

   - To configure the **requestInQueueNotifier** job, edit all parameters that begin with **jobsScheduler.job.requestInQueueNotifier**; see Section 13.3.4, "Configuration Parameters of requestInQueueNotifier".

   - To configure the **publishCerts** job, edit all parameters that begin with **jobsScheduler.job.publishCerts**; see Section 13.3.5, "Configuration Parameters of publishCerts".

   - To configure the **unpublishExpiredCerts** job, edit all parameters that begin with **jobsScheduler.job.unpublishExpiredCerts**; see Section 13.3.6, "Configuration Parameters of unpublishExpiredCerts".

5. Save the file.

6. Restart the server instance.

```
pki-server start instance_name
```

7. If the job will send automated messages, check that the mail server is set up correctly. See Section 12.4, "Configuring a Mail Server for Certificate System Notifications".

8. Customize the automatic job messages.

13.3.3. Configuration Parameters of certRenewalNotifier

Table 13.1, "certRenewalNotifier Parameters" gives details for each of these parameters that can be configured for the **certRenewalNotifier** job, either in the **CS.cfg** file or in the Certificate Manager Console.

Table 13.1. certRenewalNotifier Parameters

Parameter	Description
enabled	Specifies whether the job is enabled or disabled. The value **true** enables the job;**false** disables it.
cron	Sets the schedule when this job should be run. This sets the time at which the Job Scheduler daemon thread checks the certificates for sending renewal notifications. These settings must follow the conventions in Section 13.3.7, "Frequency Settings for Automated Jobs". For example: `0 3 * * 1-5` The job in the example is run Monday through Friday at 3:00 pm.
notifyTriggerOffset	Sets how long (in days) before the certificate expiration date the first notification will be sent.
notifyEndOffset	Sets how long (in days) after the certificate expires that notifications will continue to be sent if the certificate is not replaced.
senderEmail	Sets the sender of the notification messages, who will be notified of any delivery problems.
emailSubject	Sets the text of the subject line of the notification message.
emailTemplate	Sets the path, including the filename, to the directory that contains the template to use to create the message content.
summary.enabled	Sets whether a summary report of renewal notifications should be compiled and sent. The value **true** enables sending the summary;**false** disables it. If enabled, set the remaining summary parameters; these are required by the server to send the summary report.
summary.recipientEmail	Specifies the recipients of the summary message. These can be agents who need to know the status of user certificates or other users. Set more than one recipient by separating each email address with a comma.
summary.senderEmail	Specifies the email address of the sender of the summary message.
summary.emailSubject	Gives the subject line of the summary message.

Parameter	Description
summary.itemTemplate	Gives the path, including the filename, to the directory that contains the template to use to create the content and format of each item to be collected for the summary report.
summary.emailTemplate	Gives the path, including the filename, to the directory that contains the template to use to create the summary report email notification.

13.3.4. Configuration Parameters of requestInQueueNotifier

Table 13.2, "requestInQueueNotifier Parameters" gives details for each of these parameters that can be configured for the **requestInQueueNotifier** job, either in the **CS.cfg** file or in the Certificate Manager Console.

Table 13.2. requestInQueueNotifier Parameters

Parameter	Description
enabled	Sets whether the job is enabled (**true**) or disabled (**false**).
cron	Sets the time schedule for when the job should run. This is the time at which the Job Scheduler daemon thread checks the queue for pending requests. This setting must follow the conventions in Section 13.3.7, "Frequency Settings for Automated Jobs". For example: `0 0 * * 0`
subsystemid	Specifies the subsystem which is running the job. The only possible value is **ca**, for the Certificate Manager.
summary.enabled	Specifies whether a summary of the job accomplished should be compiled and sent. The value **true** enables the summary reports; **false** disables them. If enabled, set the remaining summary parameters; these are required by the server to send the summary report.
summary.emailSubject	Sets the subject line of the summary message.
summary.emailTemplate	Specifies the path, including the filename, to the directory containing the template to use to create the summary report.
summary.senderEmail	Specifies the sender of the notification message, who will be notified of any delivery problems.

Parameter	Description
summary.recipientEmail	Specifies the recipients of the summary message. These can be agents who need to process pending requests or other users. More than one recipient can be listed by separating each email address with a comma.

13.3.5. Configuration Parameters of publishCerts

Table 13.3, "publishCerts Parameters" gives details for each of these parameters that can be configured for the **publishCerts** job, either in the **CS.cfg** file or in the Certificate Manager Console.

Table 13.3. publishCerts Parameters

Parameter	Description
enabled	Sets whether the job is enabled. The value **true** is enabled; **false** is disabled.
cron	Sets the time schedule for when the job runs. This is the time the Job Scheduler daemon thread checks the certificates to removing expired certificates from the publishing directory. This setting must follow the conventions in Section 13.3.7, "Frequency Settings for Automated Jobs". For example: `0 0 * * 6`
summary.enabled	Specifies whether a summary of the certificates published by the job should be compiled and sent. The value **true** enables the summaries;**false** disables them. If enabled, set the remaining summary parameters; these are required by the server to send the summary report.
summary.emailSubject	Gives the subject line of the summary message.
summary.emailTemplate	Specifies the path, including the filename, to the directory containing the template to use to create the summary report.
summary.itemTemplate	Specifies the path, including the filename, to the directory containing the template to use to create the content and format of each item collected for the summary report.
summary.senderEmail	Specifies the sender of the summary message, who will be notified of any delivery problems.

Parameter	Description
summary.recipientEmail	Specifies the recipients of the summary message. These can be agents who need to know the status of user certificates or other users. More than one recipient can be set by separating each email address with a comma.

13.3.6. Configuration Parameters of unpublishExpiredCerts

Table 13.4, "unpublishExpiredCerts Parameters" gives details for each of these parameters that can be configured for the **unpublishedExpiresCerts** job, either in the **CS.cfg** file or in the Certificate Manager Console.

Table 13.4. unpublishExpiredCerts Parameters

Parameter	Description
enabled	Sets whether the job is enabled. The value **true** is enabled; **false** is disabled.
cron	Sets the time schedule for when the job runs. This is the time the Job Scheduler daemon thread checks the certificates to removing expired certificates from the publishing directory. This setting must follow the conventions in Section 13.3.7, "Frequency Settings for Automated Jobs". For example: `0 0 * * 6`
summary.enabled	Specifies whether a summary of the certificates published by the job should be compiled and sent. The value **true** enables the summaries;**false** disables them. If enabled, set the remaining summary parameters; these are required by the server to send the summary report.
summary.emailSubject	Gives the subject line of the summary message.
summary.emailTemplate	Specifies the path, including the filename, to the directory containing the template to use to create the summary report.
summary.itemTemplate	Specifies the path, including the filename, to the directory containing the template to use to create the content and format of each item collected for the summary report.
summary.senderEmail	Specifies the sender of the summary message, who will be notified of any delivery problems.

Parameter	Description
summary.recipientEmail	Specifies the recipients of the summary message. These can be agents who need to know the status of user certificates or other users. More than one recipient can be set by separating each email address with a comma.

13.3.7. Frequency Settings for Automated Jobs

The Job Scheduler uses a variation of the Unix **crontab** entry format to specify dates and times for checking the job queue and executing jobs. As shown in Table 13.5, "Time Values for Scheduling Jobs" and Figure 13.1, "Job Configuration", the time entry format consists of five fields. (The sixth field specified for the Unix **crontab** is not used by the Job Scheduler.) Values are separated by spaces or tabs.

Each field can contain either a single integer or a pair of integers separated by a hyphen (**-**) to indicate an inclusive range. To specify all legal values, a field can contain an asterisk rather than an integer. Day fields can contain a comma-separated list of values. The syntax of this expression is

> *Minute Hour Day_of_month Month_of_year Day_of_week*

Table 13.5. Time Values for Scheduling Jobs

Field	Value
Minute	0-59
Hour	0-23
Day of month	1-31
Month of year	1-12
Day of week	0-6 (where 0=Sunday)

For example, the following time entry specifies every hour at 15 minutes (1:15, 2:15, 3:15, and so on):

```
15 * * * *
```

The following example sets a job to run at noon on April 12:

```
0 12 12 4 *
```

The day-of-month and day-of-week options can contain a comma-separated list of values to specify more than one day. If both day fields are specified, the specification is inclusive; that is, the day of the month is not required to fall on the day of the week to be valid. For example, the following entry specifies a job execution time of midnight on the first and fifteenth of every month *and* on every Monday:

```
0 0 1,15 * 1
```

To specify one day type without the other, use an asterisk in the other day field. For example, the following entry runs the job at 3:15 a.m. every weekday morning:

```
15 3 * * 1-5
```

13.4. REGISTERING A JOB MODULE

Custom job plug-ins can be registered through the Certificate Manager Console. Registering a new module involves specifying the name of the module and the full name of the Java™ class that implements the module.

To register a new job module:

1. Create the custom job class. For this example, the custom job plug-in is called **MyJob.java**.
2. Compile the new class.

   ```
   javac -d . -classpath $CLASSPATH MyJob.java
   ```

3. Create a directory in the CA's **WEB-INF** web directory to hold the custom classes, so that the CA can access them.

   ```
   mkdir /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes
   ```

4. Copy the new plug-in files into the new **classes** directory, and set the owner to the Certificate System system user (**pkiuser**).

   ```
   cp -pr com /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes

   chown -R pkiuser:pkiuser /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF/classes
   ```

5. Register the plug-in.
   1. Log into the Certificate Manager Console.

      ```
      pkiconsole https://server.example.com:8443/ca
      ```

   2. In the **Configuration** tab, select **Job Scheduler** in the left navigation tree. Select **Jobs**.

      The Job Instance tab opens, which lists any currently configured jobs. Select the **Job Plugin Registration** tab.

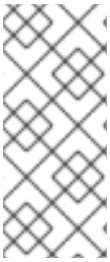
   3. Click **Register** to add the new module.

   4. In the **Register Job Scheduler Plugin Implementation** window, supply the following information:

      - **Plugin name.** Type a name for the plug-in module.
      - **Class name.** Type the full name of the class for this module; this is the path to the implementing Java™ class. If this class is part of a package, include the package name.

For example, to register a class named **customJob** that is in a package named **com.customplugins**, type **com.customplugins.customJob**.

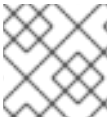
5. Click **OK**.



NOTE

It is also possible to delete job modules, but this is not recommended.

If it is necessary to delete a module, open the **Job Plugin Registration** tab as when registering a new module, select the module to delete, and click **Delete**. When prompted, confirm the deletion.



NOTE

pkiconsole is being deprecated.

PART IV. MANAGING THE SUBSYSTEM INSTANCES

CHAPTER 14. BASIC SUBSYSTEM MANAGEMENT

This chapter discusses the Certificate System administrative console, the configuration files, and other basic administrative tasks such as starting and stopping the server, managing logs, changing port assignments, and changing the internal database.

14.1. PKI INSTANCES

This version of the Certificate System continues to support *separate PKI instances* for all subsystems.

Separate PKI instances

- run as a single Java-based Apache Tomcat instance,
- contain a single PKI subsystem (CA, KRA, OCSP, TKS, or TPS), and
- must utilize unique ports if co-located on the same physical machine or virtual machine (VM).

Additionally, this version of the Certificate System introduces the notion of a *shared PKI instance*.

Shared PKI instances

- run as a single Java-based Apache Tomcat instance,
- can contain a single PKI subsystem that is identical to a separate PKI instance,
- can contain any combination of up to one of each type of PKI subsystem:
  - CA
  - TKS
  - CA, KRA
  - CA, OCSP
  - TKS, TPS
  - CA, KRA, TKS, TPS
  - CA, KRA, OCSP, TKS, TPS
  - and so on.
- allow all of their subsystems contained within that instance to share the same ports, and
- must utilize unique ports if more than one is co-located on the same physical machine or VM.

14.2. PKI INSTANCE EXECUTION MANAGEMENT

The act of starting, stopping, restarting, or obtaining the status of a PKI instance is known as execution management. Each PKI instance, separate or shared, is started, stopped, restarted, and has its status obtained separately. This section describes the execution management for any PKI instance.

14.2.1. Starting, Stopping, and Restarting a PKI Instance

A PKI instance is started, stopped, and restarted like other system programs, using **systemd**.

1. Log in to the server machine as **root**.
2. Run the **systemctl** command, specifying the action and the instance name:

   ```
   systemctl start|stop|restart pki-tomcatd@instance_name.service
   ```

   For example:

   ```
   systemctl restart pki-tomcatd@pki-tomcat.service
   ```

3. Alternatively, you can use the **pki-server** alias:

   ```
   pki-server start|stop|restart instance_name
   ```

   For example:

   ```
   pki-server restart pki-tomcat
   ```

14.2.2. Restarting a PKI Instance after a Machine Restart

If a computer running one or more PKI instances is shut down unexpectedly, more services than just the PKI instances must be restarted, in the proper order, for the subsystem to be available both through the HTML services page and the administrative console.

1. If the Directory Server instance used by the subsystem is installed on the local machine, restart the Administration Server and the Directory Server processes.

   ```
   systemctl start dirsrv-admin.service
   systemctl start dirsrv@instance_name.service
   ```

2. Start the Certificate System subsystem instances.

   ```
   pki-server start instance_name
   ```

14.2.3. Checking the PKI Instance Status

The **systemctl** command can be used to check the status of a process, showing whether it is running or stopped. For example:

```
systemctl -l status pki-tomcatd@pki-tomcat.service
pki-tomcatd@pki-tomcat.service - PKI Tomcat Server pki-tomcat
   Loaded: loaded (/lib/systemd/system/pki-tomcatd@.service; enabled)
   Active: inactive (dead) since Fri 2015-11-20 19:04:11 MST; 12s ago
  Process: 8728 ExecStop=/usr/libexec/tomcat/server stop (code=exited, status=0/SUCCESS)
  Process: 8465 ExecStart=/usr/libexec/tomcat/server start (code=exited, status=143)
  Process: 8316 ExecStartPre=/usr/bin/pkidaemon start tomcat %i (code=exited, status=0/SUCCESS)
 Main PID: 8465 (code=exited, status=143)
```

```
Nov 20 19:04:10 pki.example.com server[8728]: options used: -Dcatalina.base=/var/lib/pki/pki-tomcat
-Dcatalina.home=/usr/share/tomcat -Djava.endorsed.dirs= -Djava.io.tmpdir=/var/lib/pki/pki-
tomcat/temp -Djava.util.logging.config.file=/var/lib/pki/pki-tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
Nov 20 19:04:10 pki.example.com server[8728]: arguments used: stop
Nov 20 19:04:11 pki.example.com server[8465]: Nov 20, 2015 7:04:11 PM
org.apache.catalina.core.StandardServer await
Nov 20 19:04:11 pki.example.com server[8465]: INFO: A valid shutdown command was received via
the shutdown port. Stopping the Server instance.
Nov 20 19:04:11 pki.example.com server[8465]: PKIListener:
org.apache.catalina.core.StandardServer[before_stop]
Nov 20 19:04:11 pki.example.com server[8465]: PKIListener:
org.apache.catalina.core.StandardServer[stop]
Nov 20 19:04:11 pki.example.com server[8465]: PKIListener:
org.apache.catalina.core.StandardServer[configure_stop]
Nov 20 19:04:11 pki.example.com server[8465]: Nov 20, 2015 7:04:11 PM
org.apache.coyote.AbstractProtocol pause
Nov 20 19:04:11 pki.example.com server[8465]: INFO: Pausing ProtocolHandler ["http-bio-8080"]
Nov 20 19:04:11 pki.example.com systemd[1]: Stopped PKI Tomcat Server pki-tomcat.
```

If the instance is running, the status check returns information similar to the following example:

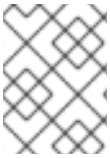
```
systemctl -l status pki-tomcatd@pki-tomcat.service
pki-tomcatd@pki-tomcat.service - PKI Tomcat Server pki-tomcat
   Loaded: loaded (/lib/systemd/system/pki-tomcatd@.service; enabled)
   Active: active (running) since Fri 2015-11-20 19:09:09 MST; 3s ago
  Process: 8728 ExecStop=/usr/libexec/tomcat/server stop (code=exited, status=0/SUCCESS)
  Process: 9154 ExecStartPre=/usr/bin/pkidaemon start tomcat %i (code=exited, status=0/SUCCESS)
 Main PID: 9293 (java)
   CGroup: /system.slice/system-pki\x2dtomcatd.slice/pki-tomcatd@pki-tomcat.service
           ├─9293 java -DRESTEASY_LIB=/usr/share/java/resteasy-base -
Djava.library.path=/usr/lib64/nuxwdog-jni -classpath
/usr/share/tomcat/bin/bootstrap.jar:/usr/share/tomcat/bin/tomcat-juli.jar:/usr/share/java/commons-
daemon.jar -Dcatalina.base=/var/lib/pki/pki-tomcat -Dcatalina.home=/usr/share/tomcat -
Djava.endorsed.dirs= -Djava.io.tmpdir=/var/lib/pki/pki-tomcat/temp -
Djava.util.logging.config.file=/var/lib/pki/pki-tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.security.manager -
Djava.security.policy==/var/lib/pki/pki-tomcat/conf/catalina.policy
org.apache.catalina.startup.Bootstrap start

Nov 20 19:09:10 pki.example.com server[9293]: Nov 20, 2015 7:09:10 PM
org.apache.catalina.core.StandardService startInternal
Nov 20 19:09:10 pki.example.com server[9293]: INFO: Starting service Catalina
Nov 20 19:09:10 pki.example.com server[9293]: Nov 20, 2015 7:09:10 PM
org.apache.catalina.core.StandardEngine startInternal
Nov 20 19:09:10 pki.example.com server[9293]: INFO: Starting Servlet Engine: Apache
Tomcat/7.0.54
Nov 20 19:09:10 pki.example.com server[9293]: Nov 20, 2015 7:09:10 PM
org.apache.catalina.startup.HostConfig deployDescriptor
Nov 20 19:09:10 pki.example.com server[9293]: INFO: Deploying configuration descriptor /etc/pki/pki-
tomcat/Catalina/localhost/ROOT.xml
Nov 20 19:09:12 pki.example.com server[9293]: Nov 20, 2015 7:09:12 PM
org.apache.catalina.startup.HostConfig deployDescriptor
Nov 20 19:09:12 pki.example.com server[9293]: INFO: Deployment of configuration descriptor
/etc/pki/pki-tomcat/Catalina/localhost/ROOT.xml has finished in 2,071 ms
Nov 20 19:09:12 pki.example.com server[9293]: Nov 20, 2015 7:09:12 PM
```

```
org.apache.catalina.startup.HostConfig deployDescriptor
Nov 20 19:09:12 pki.example.com server[9293]: INFO: Deploying configuration descriptor /etc/pki/pki-tomcat/Catalina/localhost/pki#admin.xml
```

14.2.4. Configuring a PKI Instance to Automatically Start Upon Reboot

The **systemctl** command can be used to automatically start instances upon reboot. For example, the following commands automatically start the Red Hat Administration Server, Directory Server, and a CA upon reboot:

```
# systemctl enable dirsrv-admin.service
# systemctl enable dirsrv.target
# systemctl enable pki-tomcatd@pki-tomcat.service
```



NOTE

The default PKI instance installation and configuration using the **pkispawn** command automatically enables the instance to start upon reboot.

To disable this behavior (that is, to prevent PKI instances from automatically starting upon reboot), issue the following commands:

```
# systemctl disable pki-tomcatd@pki-tomcat.service
# systemctl disable dirsrv.target
# systemctl disable dirsrv-admin.service
```

14.2.5. Setting sudo Permissions for Certificate System Services

For both simplicity of administration and security, the Certificate System and Directory Server processes can be configured so that PKI administrators (instead of only root) can start and stop the services.

A recommended option when setting up subsystems is to use a **pkiadmin** system group. (Details are in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .) All of the operating system users which will be Certificate System administrators are then added to this group. If this **pkiadmin** system group exists, then it can be granted sudo access to perform certain tasks.

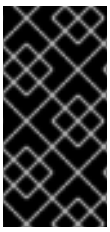
1. Edit the /**etc**/**sudoers** file; on Red Hat Enterprise Linux 8, this can be done using the **visudo** command:

   ```
   # visudo
   ```

2. Depending on what is installed on the machine, add a line for the Directory Server, the Administration Server, PKI management tools, and each PKI subsystem instance, granting **sudo** rights to the **pkiadmin** group:

   ```
   # For Directory Server services
   %pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv.target
   %pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv-admin.service

   # For PKI instance management
   %pkiadmin ALL = PASSWD: /usr/sbin/pkispawn *
   ```

```
%pkiadmin ALL = PASSWD: /usr/sbin/pkidestroy *

# For PKI instance services
%pkiadmin ALL = PASSWD: /usr/bin/systemctl * pki-tomcatd@instance_name.service
```



IMPORTANT

Make sure to set sudo permissions for every Certificate System, Directory Server, and Administration Server on the machine — and *only* for those instances on the machine. There could be multiple instances of the same subsystem type on a machine or no instance of a subsystem type. It depends on the deployment.

14.3. OPENING SUBSYSTEM CONSOLES AND SERVICES

Each subsystem has different interfaces for different user types to access. All subsystems have some kind of web services page for agents, administrators, or end users (or all three), with the exception of the TKS. Additionally, the CA, KRA, OCSP, and TKS all have a Java-based Console, which must be installed on a server, to perform administrative tasks to manage the subsystem itself.

The appearance and, to a limited extent, functionality of the subsystem's web-based services pages can be customized to better integrate with an organization's existing websites. See *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

14.3.1. Finding the Subsystem Web Services Pages

The CA, KRA, OCSP, TKS, and TPS subsystems have web services pages for agents, regular users, and administrators. These menu of web services can be accessed by opening the URL to the subsystem host over the subsystem's secure end user's port. For example, for the CA:

```
https://server.example.com:8443/ca/services
```

The main web services page for each subsystem has a list of available services pages; these are summarized in Table 14.1, “Default Web Services Pages” . To access any service specifically, access the appropriate port and append the appropriate directory to the URL. For example, to access the CA's end entities (regular users) web services:

```
https://server.example.com:8443/ca/ee/ca
```

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the services pages. For example:

```
https://1.2.3.4:8443/ca/services
https://[00:00:00:00:123:456:789:00:]:8443/ca/services
```

Some subsystem interfaces require client authentication to access them, usually interfaces associated with agent or administrator roles. Other interfaces, even those that run over secure (SSL connections) do not require client authentication. Some of these interfaces (such as end entities services) can be configured to require client authentication, but others cannot be configured to support client authentication. These differences are noted in Table 14.1, “Default Web Services Pages” .

NOTE

Anyone can access the end user pages for a subsystem, but accessing agent or admin web services pages requires that an agent or administrator certificate be issued and installed in the web browser, or authentication to the web services fails.

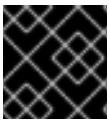
Table 14.1. Default Web Services Pages

Used for SSL	Used for Client Authentication[a]	Web Services	Web Service Location
Certificate Manager			
No		End Entities	ca/ee/ca/
Yes	No	End Entities	ca/ee/ca
Yes	Yes	Agents	ca/agent/ca
Yes	No	Services	ca/services
Yes	No	Console	pkiconsole https://*host:port*/ca
Key Recovery Authority			
Yes	Yes	Agents	kra/agent/kra
Yes	No	Services	kra/services
Yes	No	Console	pkiconsole https://*host:port*/kra
Online Certificate Status Manager			
Yes	Yes	Agents	ocsp/agent/ocsp
Yes	No	Services	ocsp/services
Yes	No	Console	pkiconsole https://*host:port*/ocsp
Token Key Service			
Yes	No	Services	tks/services

Used for SSL	Used for Client Authentication[a]	Web Services	Web Service Location
Yes	No	Console	pkiconsole https://*host:port*/tks
Token Processing System			
Yes		Services	index.cgi

[a] Services with a client authentication value of **No** can be reconfigured to require client authentication. Services which do not have either a **Yes** or **No** value cannot be configured to use client authentication.

14.3.2. Starting the Certificate System Administrative Console



IMPORTANT

pkiconsole is being deprecated.

The Console is opened by connecting to the subsystem instance over its SSL port using the **pkiconsole** command. This command has the format:

```
pkiconsole https://server.example.com:admin_port/subsystem_type
```

The *subsystem_type* can be **ca**, **kra**, **ocsp**, or **tks**. For example, this opens the KRA console:

```
pkiconsole https://server.example.com:8443/kra
```

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the console. For example:

```
pkiconsole https://1.2.3.4:8443/ca
pkiconsole https://[00:00:00:00:123:456:789:00:]:8443/ca
```

14.3.3. Enabling SSL for the Java Administrative Console

Certificate-based authentication to the Certificate System Console can be enabled so that administrators must authenticate using a client certificate before logging into the Certificate System Console. Store the administrators' certificates before enabling certificate-based authentication.

To enable SSL in the Console, configure both the client and the server.

IMPORTANT

If a CA is configured for client authentication over the admin port and that CA is a security domain manager, then **no new PKI subsystems can be configured that use that CA for its security domain**. New PKI instances register themselves to the security domain CA over the admin port but without using client authentication. If the CA requires client authentication, then the registration attempt fails.

First, set up the Certificate System server to use SSL client authentication:

1. Store the certificates for any administrator using this system. The certificate should be either from the CA itself or from whichever CA signed the certificate for the subsystem.
    1. Open the subsystem console.
    2. Select the **Users and Groups** option on the left.
    3. In the **Users** tab, select the administrative user, and click **Manage Certificates**.
    4. Click **Import**.
    5. Paste in the base-64 encoded SSL client certificate, such as the administrator certificate stored in the web browser.

    Make sure the client certificate is good for SSL client authentication; otherwise, the server will not accept the client certificate and will post an error message in the error log in the **/var/log**/*instanceID*/**system**:

    ```
    failure (14290): Error receiving connection
    SEC_ERROR_INADEQUATE_CERT_TYPE - Certificate type not approved for application.)
    ```

2. Stop the subsystem.

    ```
    pki-server stop instance_name
    ```

3. Open the instance configuration directory, **/var/lib/pki/*instance_name*/*subsystem_type*/conf**.
4. Open the file **CS.cfg**.
5. Change the value of the **authType** parameter from **pwd** to **sslclientauth**:

    ```
    authType=sslclientauth
    ```

6. Save the file.
7. Open the **server.xml** file.
8. Change the **clientAuth="false"** attribute to **clientAuth="want"** in the admin interface connector section:

    ```
    <Connector port="8443" maxHttpHeaderSize="8192"
        maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true"
        acceptCount="100" scheme="https" secure="true"
    ```

```
        clientAuth="want" sslProtocol="SSL"
.....
        serverCertFile="/var/lib/pki/pki-tomcat/conf/serverCertNick.conf"
        passwordFile="/var/lib/pki/pki-tomcat/conf/password.conf"
        passwordClass="org.apache.tomcat.util.net.jss.PlainPasswordFile"
        certdbDir="/var/lib/pki/pki-tomcat/alias"/>
```

The **want** value means that client authentication is preferred, but not required. This allows client authentication through interfaces that can easily use it (like the Console) while still allowing clients which do not easily support client authentication (other subsystems within the security domain) to connect using regular connections.

9. Start the subsystem.

```
pki-server start instance_name
```

After setting up the server, then configure the client to use SSL client authentication.

The Console must have access to the administrator certificate and keys used for SSL client authentication to the server. The Console's default certificate and key databases are stored in the **.redhat-idm-console** directory.

To provide access to the administrator certificate and keys, either export them from the administrator's browser into a **.p12** file and then import it by using **pk12util**, or copy the browser's certificate and key databases into the **.redhat-idm-console** directory. (This procedure assumes that the certificates are exported from the browser into a **.p12** file.)

1. Export the administrator user certificate and keys from the browser into a file, such as **admin.p12**.
2. Open the user's console directory.

```
/user-directory/.redhat-idm-console
```

3. If necessary, create new security databases.

```
certutil -N -d .
```

4. Stop the Certificate System instance.

```
pki-server stop instance_name
```

5. Use **pk12util** to import the certificates.

```
# pk12util -i /tmp/admin.p12 -d /user-directory/.redhat-idm-console -W [p12filepassword]
```

If the procedure is successful, the command prints the following:

```
pk12util: PKCS12 IMPORT SUCCESSFUL
```

6. Export the 64-bit blob of the issuing CA certificate from the browser and save it to a file like **ca.crt**.

7. Import the CA certificate from the base 64-blob associated with the admin user cert.

```
certutil -A -d . -n ca -t CT,C,C -i ./ca.crt
```

8. Start the Certificate System instance.

```
pki-server start instance_name
```

9. Start the Console; now, it prompts for a certificate.

14.4. RUNNING SUBSYSTEMS UNDER A JAVA SECURITY MANAGER

Java services have the option of having a Security Manager which defines unsafe and safe operations for applications to perform. When the subsystems are installed, they have the Security Manager enabled automatically, meaning each Tomcat instance starts with the Security Manager running.

14.4.1. About the Security Manager Policy Files

When the five Java subsystems (the CA, OCSP, KRA, TKS, and TPS) run within the Java Security Manager, they use a combination of three sets of policies:

- The **catalina.policy** file from the default Tomcat policy located in the **/usr/share/tomcat/conf** directory; this is updated whenever the general Tomcat files are updated.
- A **pki.policy** file, in the **/var/lib/pki/*instance_name*/*subsystem_type*/conf** directory, that is supplied with the subsystem instance.
- A **custom.policy** file, in the **/var/lib/pki/*instance_name*/*subsystem_type*/conf** directory, that contains user-defined security policies.

These three files are concatenated together whenever the Tomcat service starts to create a revised **catalina.policy** file, also in the **/var/lib/pki/*instance_name*/*subsystem_type*/conf** directory, which is used for the instance.

The default **pki.policy** file contains permissions that grant unrestricted access to the Tomcat, LDAP, and symkey services used by the PKI subsystems. For example:

```
// These permissions apply to Tomcat java as utilized by PKI instances
grant codeBase "file:/usr/share/java/tomcat/-" {
    permission java.security.AllPermission;
};
```

The **custom.policy** file is empty by default; administrators can write policies in that file which will be used in addition to the given PKI policies and Tomcat policies.

14.4.2. Starting a Subsystem Instance without the Java Security Manager

All Java subsystems configured under a PKI Tomcat instance are automatically run under a Java Security Manager (unless the instance was created by overriding **pki_security_manager=true** under the [Tomcat] section in the **/etc/pki/default.cfg** file). However, it is possible to start or restart an instance and run it *without* starting the Java Security Manager, as shown below.

Procedure 14.1. Starting an Instance Without the Java Security Manager

1. Stop the instance.

```
# pki-server stop instance_name
```

2. Edit the /**etc**/**sysconfig**/***instance_name*** file and turn off the security manager:

```
SECURITY_MANAGER="false"
```

3. Start the instance.

```
# pki-server start instance_name
```

14.5. CONFIGURING THE LDAP DATABASE

The Certificate System performs certificate- and key-management functions in response to the requests it receives. These functions include the following:

- Storing and retrieving certificate requests
- Storing and retrieving certificate records
- Storing CRLs
- Storing ACLs
- Storing privileged user and role information
- Storing and retrieving end users' encryption private key records

To fulfill these functions, the Certificate System is incorporated with a Red Hat Directory Server, referred to as the *internal database* or *local database*. The Directory Server is referenced as part of the Certificate System configuration; when the Certificate System subsystem is configured, a new database is created within the Directory Server. This database is used as an embedded database exclusively by the Certificate System instance and can be managed using directory management tools that come with the Directory Server.

The Certificate System instance database is listed with the other Directory Server databases in the *serverRoot*/**slapd-***DS_name*/**db**/ directory. These databases are named by the value determined by the value of the **pki_ds_database** variable under the specified subsystem section within the /**etc**/**pki**/**default.cfg** file (*CS_instance_name-CA*, *CS_instance_name-KRA*, *CS_instance_name-OCSP*, *CS_instance_name-TKS*, and *CS_instance_name-TPS* by default), which is the default format given during the instance configuration. For example, for a Certificate Manager named **ca1**, the database name would be **ca1-CA**. Similarly, the database name is determined by the value of the **pki_ds_base_dn** variable under the specified subsystem section within the /**etc**/**pki**/**default.cfg** file ((o=CS_instance_name-CA, o=CS_instance_name-KRA, o=CS_instance_name-OCSP, o=CS_instance_name-TKS, or o=CS_instance_name-TPS by default), and is also set during the configuration.

The subsystems use the database for storing different objects. A Certificate Manager stores all the data, certificate requests, certificates, CRLs, and related information, while a KRA only stores key records and related data.

WARNING

The internal database schema are configured to store only Certificate System data. Do not make any changes to it or configure the Certificate System to use any other LDAP directory. Doing so can result in data loss.

Additionally, do not use the internal LDAP database for any other purpose.

14.5.1. Changing the Internal Database Configuration

To change the Directory Server instance that a subsystem instance uses as its internal database:

1. Log into the subsystem administrative console.

```
pkiconsole https://server.example.com:admin_port/subsystem_type
```

2. In the **Configuration** tab, select the **Internal Database** tab.
3. Change the Directory Server instance by changing the hostname, port, and bind DN fields.

   The hostname is the fully qualified hostname of the machine on which the Directory Server is installed, such as **certificates.example.com**. The Certificate System uses this name to access the directory.

   By default, the hostname of the Directory Server instance used as the internal database is shown as **localhost** instead of the actual hostname. This is done to insulate the internal database from being visible outside the system since a server on **localhost** can only be accessed from the local machine. Thus, the default configuration minimizes the risk of someone connecting to this Directory Server instance from outside the local machine.

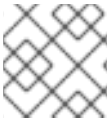
   The hostname can be changed to something other than **localhost** if the visibility of the internal database can be limited to a local subnet. For example, if the Certificate System and Directory Server are installed on separate machines for load balancing, specify the hostname of the machine in which the Directory Server is installed.

   The port number is the TCP/IP port used for non-SSL communications with the Directory Server.

   The DN should be the Directory Manager DN. The Certificate System subsystem uses this DN when it accesses the directory tree to communicate with the directory.

4. Click **Save**.

   The configuration is modified. If the changes require restarting the server, a prompt appears with that message. In that case, restart the server.



NOTE

pkiconsole is being deprecated.

14.5.2. Using a Certificate Issued by Certificate System in Directory Server

To use an encrypted connection to Directory Server when you installed Certificate System, it was necessary to either use a certificate issued by an external Certificate Authority (CA) or a self-signed certificate. However, after setting up the Certificate System CA, administrators often want to replace this certificate with one issued by Certificate System.

To replace the TLS certificate used by Directory Server with a certificate issued by Certificate System:

1. On the Directory Server host:

   a. Stop the Directory Server instance:

   ```
   # systemctl stop dirsrv@instance_name
   ```

   b. Generate a Certificate Signing Request (CSR).

   For example, to generate a CSR which uses 2048 bit RSA encryption, and to store it in the ~/**ds.csr** file:

   ```
   # PKCS10Client -d /etc/dirsrv/slapd-instance_name/ -p password -a rsa -l 2048 -o
   ~/ds.csr -n "CN=$HOSTNAME"
   PKCS10Client: Debug: got token.
   PKCS10Client: Debug: thread token set.
   PKCS10Client: token Internal Key Storage Token logged in...
   PKCS10Client: key pair generated.
   PKCS10Client: CertificationRequest created.
   PKCS10Client: b64encode completes.
   Keypair private key id: -3387b397ebe254b91c5d6c06dc36618d2ea8b7e6

   -----BEGIN CERTIFICATE REQUEST-----
   ...
   -----END CERTIFICATE REQUEST-----
   PKCS10Client: done. Request written to file: ~/ds.csr
   ```

   c. Start the Directory Server instance to enable the CA to process the request:

   ```
   # systemctl start dirsrv@instance_name
   ```

   d. Submit the CSR to the Certificate System's CA. For example:

   ```
   # pki -d /etc/dirsrv/slapd-instance_name/ ca-cert-request-submit --profile caServerCert --
   csr-file ~/ds.csr
   ----------------------------
   Submitted certificate request
   ----------------------------
     Request ID: 13
     Type: enrollment
     Request Status: pending
     Operation Result: success
   ```

2. On the Certificate System host:

a. Import the CA agent certificate into a Network Security Services (NSS) database to sign the CMC full request:

   i. Create a new directory. For example:

   ```
   # mkdir ~/certs_db/
   ```

   ii. Initialize the database in the newly created directory:

   ```
   # certutil -N -d ~/certs_db/
   ```

   iii. Display the serial number of the CA signing certificate:

   ```
   # pki -p 8080 ca-cert-find --name "CA Signing Certificate"
   ---------------
   1 entries found
   ---------------
   Serial Number: 0x87bbe2d
   ...
   ```

   iv. Use the serial number from the previous step to download the CA signing certificate into the ~/**certs_db**/**CA.pem** file:

   ```
   # pki -p 8080 ca-cert-show 0x87bbe2d --output ~/certs_db/CA.pem
   ```

   v. Import the CA signing certificate into the NSS database:

   ```
   # pki -d ~/certs_db/ -c password client-cert-import "CA Certificate" --ca-cert
   ~/certs_db/CA.pem
   ```

   vi. Import the agent certificate:

   ```
   # pk12util -d ~/certs_db/ -i ~/.dogtag/instance_name/ca_admin_cert.p12
   Enter Password or Pin for "NSS FIPS 140-2 Certificate DB": password
   Enter password for PKCS12 file: password
   pk12util: PKCS12 IMPORT SUCCESSFUL
   ```

b. Create the Certificate Management over CMS (CMC) request:

   i. Create a configuration file, such as ~/**sslserver-cmc-request.cfg**, with the following content:

   ```
   # NSS database directory where the CA agent certificate is stored.
   dbdir=~/certs_db/

   # NSS database password.
   password=password

   # Token name (default is internal).
   tokenname=internal

   # Nickname for CA agent certificate.
   nickname=caadmin
   ```

```
# Request format: pkcs10 or crmf.
format=pkcs10

# Total number of PKCS10/CRMF requests.
numRequests=1

# Path to the PKCS10/CRMF request.
# The content must be in Base-64 encoded format.
# Multiple files are supported. They must be separated by space.
input=~/ds.csr

# Path for the CMC request.
output=~/sslserver-cmc-request.bin
```

ii. Create the CMC request:

```
# CMCRequest ~/sslserver-cmc-request.cfg
...
The CMC enrollment request in base-64 encoded format:
...
The CMC enrollment request in binary format is stored in ~/sslserver-cmc-request.bin
```

c. Submit the CMC request:

i. Create a configuration file, such as **~/sslserver-cmc-submit.cfg**, with the following content:

```
# PKI server host name.
host=server.example.com

# PKI server port number.
port=8443

# Use secure connection.
secure=true

# Use client authentication.
clientmode=true

# NSS database directory where the CA agent certificate is stored.
dbdir=~/certs_db/

# NSS database password.
password=password

# Token name (default: internal).
tokenname=internal

# Nickname of CA agent certificate.
nickname=caadmin

# CMC servlet path
servlet=/ca/ee/ca/profileSubmitCMCFull?profileId=caCMCserverCert
```

```
# Path for the CMC request.
input=~/sslserver-cmc-request.bin

# Path for the CMC response.
output=~/sslserver-cmc-response.bin
```

ii. Submit the request:

```
# HttpClient sslserver-cmc-submit.cfg
...
The response in binary format is stored in
~/sslserver-cmc-response.bin
```

iii. Optionally, verify the result:

```
# CMCResponse -d ~/certs_db/ -i ~/sslserver-cmc-response.bin
...
Number of controls is 1
Control #0: CMCStatusInfoV2
  OID: {1 3 6 1 5 5 7 7 25}
  BodyList: 1
  Status: SUCCESS
```

d. Display the serial number of the Directory Server certificate:

```
# pki -p 8080 ca-cert-find --name "DS Certificate"
---------------
1 entries found
---------------
Serial Number: 0xc3eeb0c
...
```

e. Use the serial number from the previous step to download the certificate:

```
# pki -p 8080 ca-cert-show 0xc3eeb0c --output ~/ds.crt
```

f. Copy the certificate for Directory Server and the CA certificate to the Directory Server host. For example:

```
# scp ~/ds.crt ~/certs_db/CA.pem ds.example.com:~/
```

g. Stop Certificate System:

```
# pki-server stop instance_name
```

3. On the Directory Server host:

a. Stop the Directory Server instance:

```
# systemctl stop dirsrv@instance_name
```

b. Replace the certificates. For details, see the corresponding sections in the *Red Hat Directory Server Administration Guide*:

i. Remove the old certificate and CA certificate. See *Removing a Certificate* .

ii. Install the CA certificate issued by Certificate System. See *Installing a CA Certificate* .

iii. Install the certificate for Directory Server issued by Certificate System. See *Installing a Server Certificate*.

c. Start the Directory Server instance:

```
# systemctl start dirsrv@instance_name
```

4. Start Certificate System:

```
# pki-server stop instance_name
```

5. Optionally, configure certificate-based authentication. For details, see Section 14.5.3, "Enabling SSL/TLS Client Authentication with the Internal Database".

14.5.3. Enabling SSL/TLS Client Authentication with the Internal Database

Client authentication allows one entity to authenticate to another entity by presenting a certificate. This method of authentication is used by Certificate System agents to log into agent services pages, for example.

To use an SSL/TLS connection between a Certificate System instance and the LDAP directory instance that it uses as its internal database, client authentication must be enabled to allow the Certificate System instance to authenticate and bind to the LDAP directory.

There are two parts to setting up client authentication. The first is configuring the LDAP directory, such as setting up SSL/TLS and setting ACIs to control the Certificate System instance access. The second is creating a user on the Certificate System instance which it will use to bind to the LDAP directory and setting up its certificate.

To configure LDAPS for a PKI instance, see the pkispawn(8) man page (Example: Installing a PKI subsystem with a secure LDAP connection).

14.5.4. Restricting Access to the Internal Database

The Red Hat Directory Server Console displays an entry or icon for the Directory Server instance that the Certificate System uses as its internal database.

Unlike the Certificate System Console, in which access is restricted to users with Certificate System administrator privileges, the Directory Server Console can be accessed by any user. The user can open the Directory Server Console for the internal database and change to the data stored there, such as deleting users from the Certificate System administrators group or adding his own entry to the group.

Access can be restricted to the internal database to only those users who know the Directory Manager DN and password. This password can be changed by modifying the single sign-on password cache.

1. Log into the Directory Server Console.

2. Select the Certificate System internal database entry, and click **Open**.

3. Select the **Configuration** tab.
4. In the navigation tree, expand **Plug-ins**, and select **Pass-Through Authentication**.
5. In the right pane, deselect the **Enable plugin** checkbox.
6. Click **Save**.

   The server prompts to restart the server.
7. Click the **Tasks** tab, and click **Restart the Directory Server**.
8. Close the Directory Server Console.
9. When the server is restarted, open the Directory Server Console for the internal database instance.

   The **Login to Directory** dialog box appears; the **Distinguished Name** field displays the Directory Manager DN; enter the password.

   The Directory Server Console for the internal database opens only if the correct password is entered.

14.6. VIEWING SECURITY DOMAIN CONFIGURATION

A *security domain* is a registry of PKI services. PKI services, such as CAs, register information about themselves in these domains so users of PKI services can find other services by inspecting the registry. The security domain service in Certificate System manages both the registration of PKI services for Certificate System subsystems and a set of shared trust policies.

The security domain manages the trust relationships between subsystems automatically, so if a TPS, TKS, and KRA are within the same security domain, they can communicate securely.

NOTE

The security domain is used during subsystem configuration. When a subsystem is being set up, it can check the security domain registry to see available instances. If it needs to create a trusted relationship with another instance — like a TPS which uses a TKS and KRA for its operations — then the security domain is used to create a TPS agent user on the selected TKS and KRA instances.

The registry provides a complete view of all PKI services provided by the subsystems within that domain. Each Certificate System subsystem must be either a host or a member of a security domain.

Only a CA can host and manage a security domain. Each CA has its own LDAP entry, and the security domain is an organizational group underneath that CA entry:

```
ou=Security Domain,dc=example,dc=com
```

Then there is a list of each subsystem type beneath the security domain organizational group, with a special object class (**pkiSecurityGroup**) to identify the group type:

```
cn=KRAList,ou=Security Domain,dc=example,dc=com
objectClass: top
objectClass: pkiSecurityGroup
```

```
cn: KRAList
```

Each subsystem instance is then stored as a member of that group, with a special **pkiSubsystem** object class to identify the entry type:

```
dn: cn=server.example.com:8443,cn=KRAList,ou=Security Domain,dc=example,dc=com
objectClass: top
objectClass: pkiSubsystem
cn: kra.example.com:8443
host: server.example.com
SecurePort: 8443
SecureAgentPort: 8443
SecureAdminPort: 8443
UnSecurePort: 8080
DomainManager: false
Clone: false
SubsystemName: KRA server.example.com 8443
```

14.7. MANAGING THE SELINUX POLICIES FOR SUBSYSTEMS

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering. For more information about SELinux, see the *Using SELinux* guide for Red Hat Enterprise Linux 8.

14.7.1. About SELinux

Basically, SELinux identifies *objects* on a system, which can be files, directories, users, processes, sockets, or any other thing on a Linux host. These objects correspond to the Linux API objects. Each object is then mapped to a *security context*, which defines the type of object it is and how it is allowed to function on the Linux server.

System processes run within SELinux domains. Each domain has a set of rules that defines how the SELinux domain interacts with other SELinux objects on the system. This set of rules, then, determines which resources a process may access and what operations it may perform on those resources.

For Certificate System, each subsystem type runs within a specific domain for that subsystem type. Every instance of that subsystem type belongs to the same SELinux domain, regardless of how many instances are on the system For example, if there are three CAs installed on a server, all three belong to the **http_port_t** SELinux domain.

The rules and definitions for all the subsystems comprise the overall Certificate System SELinux policy. Certificate System SELinux policies are already configured when the subsystems are installed, and all SELinux policies are updated every time a subsystem is added with **pkispawn** or removed with **pkidestroy**.

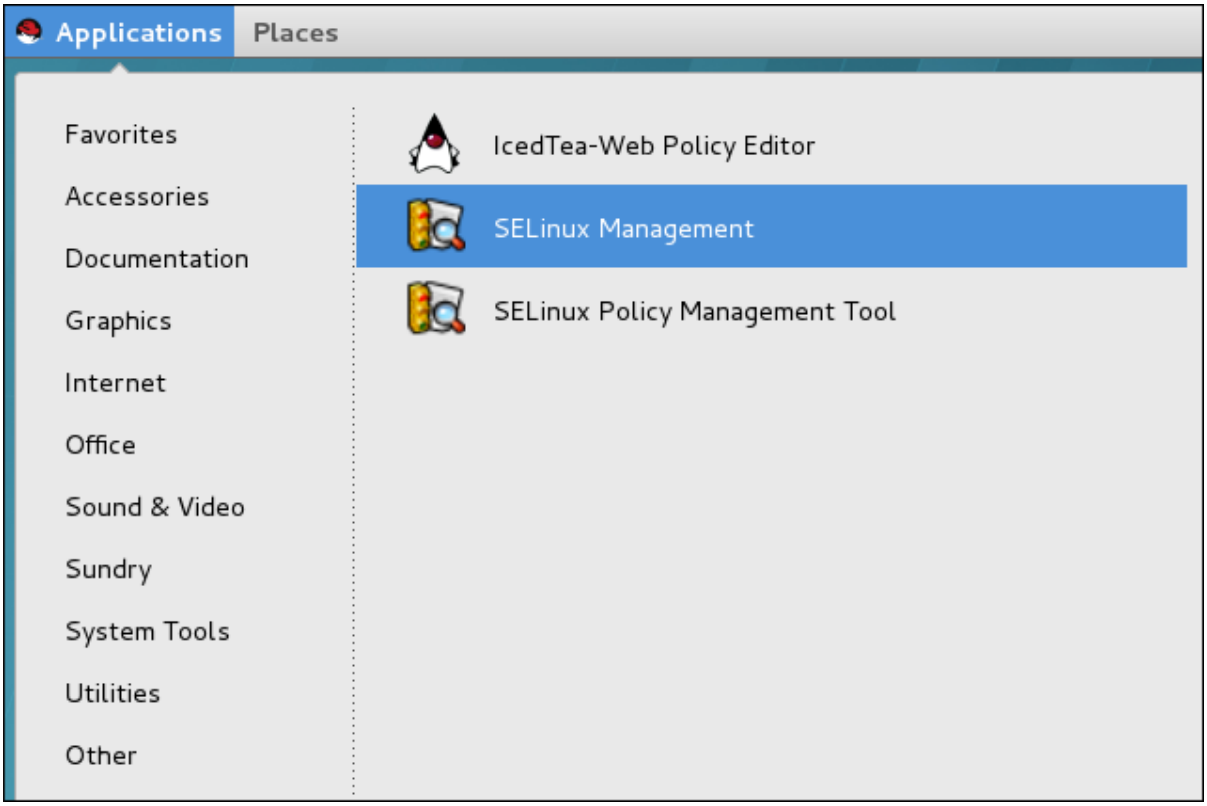
The Certificate System subsystems run with SELinux set in enforcing mode, meaning that Certificate System operations can be successfully performed even when all SELinux rules are required to be followed.

By default, the Certificate System subsystems run confined by SELinux policies.

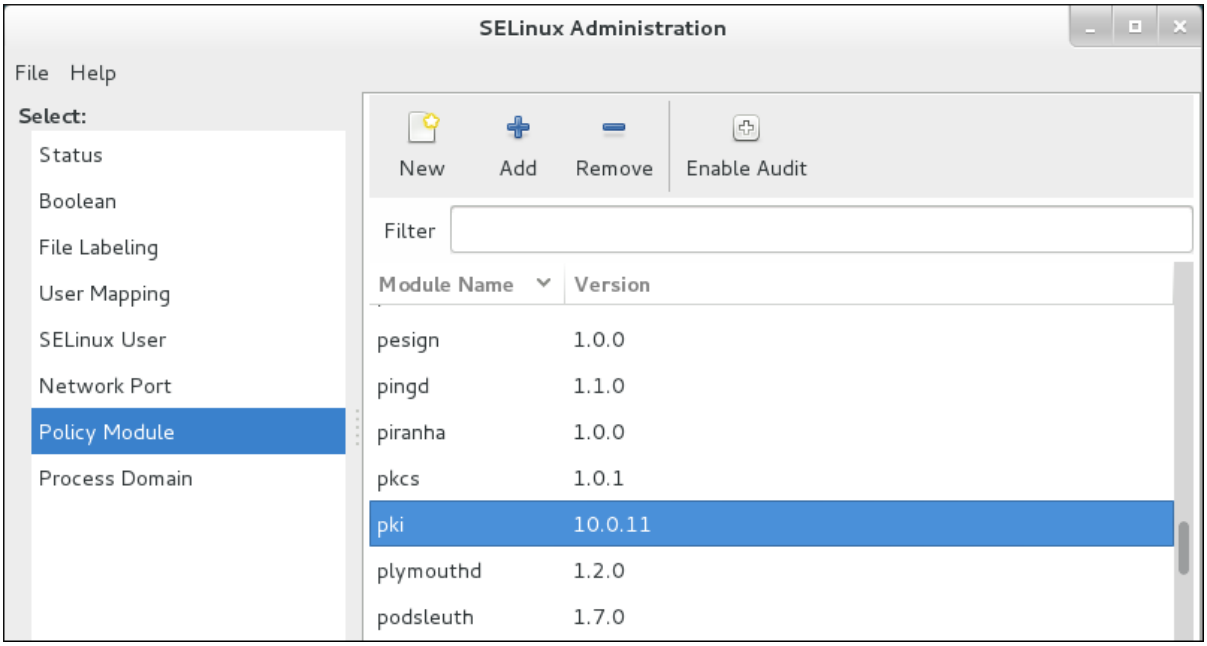
14.7.2. Viewing SELinux Policies for Subsystems

All Certificate System policies are are part of the system SELinux policy. The configured policies can be viewed using the SELinux Administration GUI, which you can get by installing the policycoreutils-gui package.

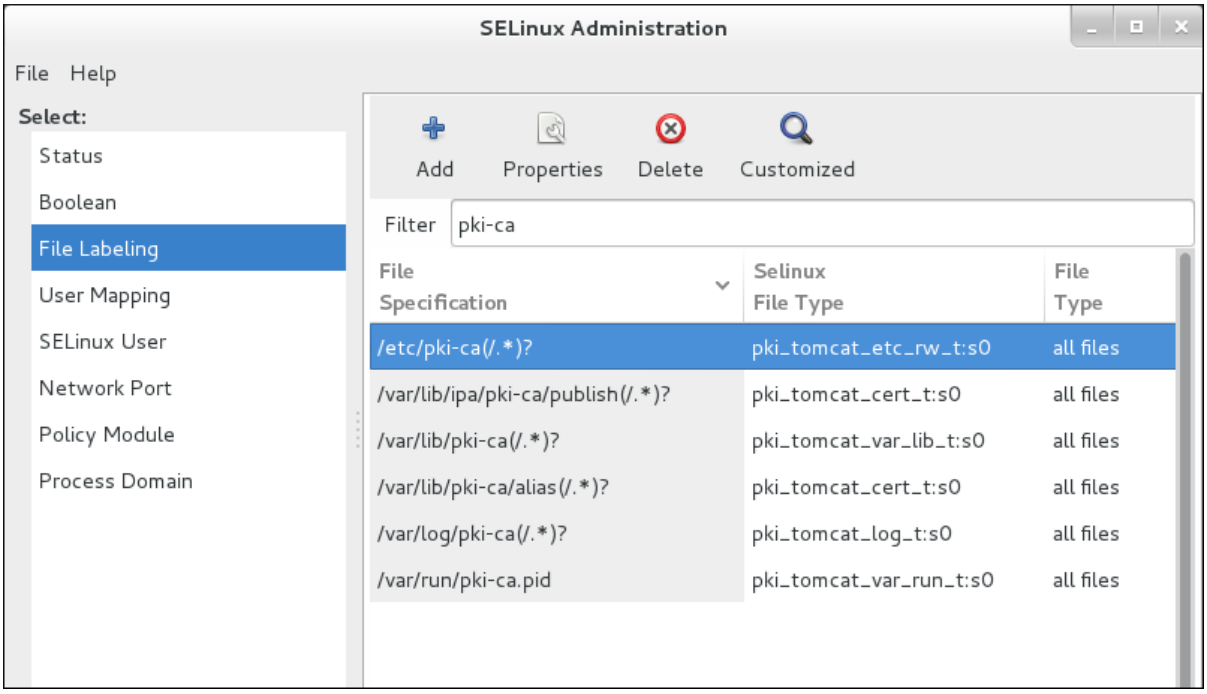
1. Either run the **system-config-selinux** command or open the utility by accessing **Applications → Other → SELinux Management** for the main system menu.



2. To check the version of the Certificate System SELinux policy installed, click the **Policy Module** section in the left bar.



3. To view the policies set on the individual files and processes, click the **File Labeling** section. To view the policies for the port assignments for the subsystems, click the **Network Port** section.



14.7.3. Relabeling nCipher netHSM Contexts

The nCipher netHSM software does not come with its own SELinux policy, so the Certificate System contains a default netHSM policy, shown in Example 14.1, "netHSM SELinux Policy".

Example 14.1. netHSM SELinux Policy

```
# default labeling for nCipher
/opt/nfast/scripts/init.d/(.*)  gen_context(system_u:object_r:initrc_exec_t,s0)
/opt/nfast/sbin/init.d-ncipher  gen_context(system_u:object_r:initrc_exec_t,s0)
/opt/nfast(/.*)?                gen_context(system_u:object_r:pki_common_t, s0)
/dev/nfast(/.*)?                gen_context(system_u:object_r:pki_common_dev_t,0)
```

Other rules allow the **pki_*_t** domain to talk to files that are labeled **pki_common_t** and **pki_common_dev_t**.

If any of the nCipher configuration is changed (even if it is in the default directory, **/opt/nfast**), run the **restorecon** to make sure all files are properly labeled:

```
restorecon -R /dev/nfast
restorecon -R /opt/nfast
```

If the nCipher software is installed in a different location or if a different HSM is used, the default Certificate System HSM policy needs to be relabelled using **semanage**.

14.8. BACKING UP AND RESTORING CERTIFICATE SYSTEM

Certificate System does not include backup and restore tools. However, the Certificate System components can still be archived and restored manually, which can be necessary for deployments where information cannot be accessed if certificate or key information is lost. Three major parts of Certificate System need to be backed up routinely in case of data loss or hardware failure:

- *Internal database.* Subsystems use an LDAP database to store their data. The Directory Server provides its own backup scripts and procedures.
- *Security databases.* The security databases store the certificate and key material. If these are stored on an HSM, then consult the HSM vendor documentation for information on how to back up the data. If the information is stored in the default directories in the instance **alias** directory, then it is backed up with the instance directory. To back it up separately, use a utility such as **tar** or **zip**.
- *Instance directory.* The instance directory contains all configuration files, security databases, and other instance files. This can be backed up using a utility such as **tar** or **zip**.

14.8.1. Backing up and Restoring the LDAP Internal Database

The Red Hat Directory Server documentation contains more detailed information on backing up and restoring the databases.

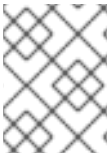
14.8.1.1. Backing up the LDAP Internal Database

Two pairs of subcommands of the **dsctl** command are available to back up the Directory Server instance. Each back-up subcommand has a counterpart to restore the files it generated:

- The **db2ldif** subcommand creates a LDIF file you can restore using the **ldif2db** subcommand.
- The **db2bak** subcommand creates a backup file you can restore using the **bak2db** subcommand.

14.8.1.1.1. Backing up using db2ldif

Running the **db2ldif** subcommand backs up a single subsystem database.



NOTE

As the **db2ldif** subcommand runs with the *dirsrv* user, it doesn't have permissions to write under the /**root**/ directory, so you need to provide a path where it can write.

Back up each Directory Server database used by PKI subsystems. You can use the **pki-server ca-db-config-show** command to check the database name for a given subsystem. For example, to back up the main database, **userRoot**:

1. Stop the instance:

```
# dsctl instance_name stop
```

2. Export the database into an LDIF file:

```
# dsctl instance_name db2ldif userroot /tmp/example.ldif
OK group dirsrv exists
OK user dirsrv exists
ldiffile: /tmp/example.ldif
[18/Jul/2018:10:46:03.353656777 +0200] - INFO - ldbm_instance_config_cachememsize_set
- force a minimal value 512000
[18/Jul/2018:10:46:03.383101305 +0200] - INFO - ldbm_back_ldbm2ldif - export userroot:
Processed 160 entries (100%).
```

```
[18/Jul/2018:10:46:03.391553963 +0200] - INFO - dblayer_pre_close - All database threads
now stopped
db2ldif successful
```

3. Start the instance:

```
# dsctl instance_name start
```

To restore the LDIF file using the **ldif2db** subcommand, see Section 14.8.1.2.1, “Restoring using ldif2db”.

14.8.1.1.2. Backing up using db2bak

Running the **db2bak** subcommand backs up all Certificate System subsystem databases for that Directory Server (and any other databases maintained by that Directory Server instance). For example:

For example:

1. Stop the instance:

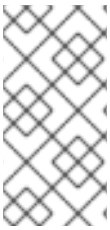
```
# dsctl instance_name stop
```

2. Backup the database:

```
# dsctl instance_name db2bak
OK group dirsrv exists
OK user dirsrv exists
[18/Jul/2018:14:02:37.358958713 +0200] - INFO - ldbm_instance_config_cachememsize_set
- force a minimal value 512000
...
db2bak successful
```

3. Start the instance:

```
# dsctl instance_name start
```



NOTE

As the **db2bak** subcommand runs with the *dirsrv* user, the target directory must be writeable by *dirsrv*. Running the subcommand without any argument creates the backup in the **/var/lib/dirsrv/slapd-*<instance_name>*/bak** folder where **db2bak** has the proper write permissions.

To restore the LDIF file using **bak2db**, see Section 14.8.1.2.2, “Restoring using bak2db”.

14.8.1.2. Restoring the LDAP Internal Database

Depending on how you backed up the Directory Server instance, use **ldif2db** or **bak2db** with the corresponding file(s) to restore the database.

NOTE

Make sure you stop the instance before restoring databases.

14.8.1.2.1. Restoring using ldif2db

If you created a LDIF file with **db2ldif**, stop the Directory Server instance and import the files using the **ldif2db** subcommand. You can specify a single database to restore from the backup. For example, for the main database, **userRoot**:

1. Stop the Directory Server instance:

```
# dsctl instance_name stop
```

2. Import the data from the LDIF file:

```
# dsctl instance_name ldif2db userroot /tmp/example.ldif
OK group dirsrv exists
OK user dirsrv exists
[17/Jul/2018:13:42:42.015554231 +0200] - INFO - ldbm_instance_config_cachememsize_set
- force a minimal value 512000
...
[17/Jul/2018:13:42:44.302630629 +0200] - INFO - import_main_offline - import userroot:
Import complete.  Processed 160 entries in 2 seconds. (80.00 entries/sec)
ldif2db successful
```

3. Start the Directory Server instance:

```
# dsctl instance_name start
```

14.8.1.2.2. Restoring using bak2db

If you created a backup file with **db2bak**, stop the Directory Server and import the file using the **bak2db** subcommand. For example:

1. Stop the Directory Server instance:

```
# dsctl instance_name stop
```

2. Restore the databases:

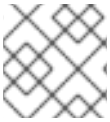
```
# dsctl instance_name bak2db /var/lib/dirsrv/slapd-instance_name/bak/instance_name-
time_stamp/
OK group dirsrv exists
OK user dirsrv exists
[20/Jul/2018:15:52:24.932598675 +0200] - INFO - ldbm_instance_config_cachememsize_set
- force a minimal value 512000
...
bak2db successful
```

3. Start the Directory Server instance:

```
# dsctl instance_name start
```

14.8.2. Backing up and Restoring the Instance Directory

The instance directory has all of the configuration information for the subsystem instance, so backing up the instance directory preserves the configuration information not contained in the internal database.



NOTE

Stop the subsystem instance before backing up the instance or the security databases.

1. Stop the subsystem instance.

```
pki-server stop instance_name
```

2. Save the directory to a compressed file:

```
# cd /var/lib/pki/
# tar -chvf /export/archives/pki/instance_name.tar instance_name/
```

For example:

```
# cd /var/lib/pki/
# tar -chvf /tmp/test.tar pki-tomcat/ca/
pki-tomcat/ca/
pki-tomcat/ca/registry/
pki-tomcat/ca/registry/ca/
.....
```

3. Restart the subsystem instance.

```
pki-server start instance_name
```

You can use the Certificate System backup files, both the **alias** database backups and the full instance directory backups, to replace the current directories if the data is corrupted or the hardware is damaged. To restore the data, uncompress the archive file using the **unzip** or **tar** tools, and copy the archive over the existing files.

To restore the instance directory:

1. Uncompress the archive:

```
cd /export/archives/pki/
tar -xvf instance_name.tar
```

For example:

```
# cd /tmp/
# tar -xvf test.tar
pki-tomcat/ca/
pki-tomcat/ca/registry/
pki-tomcat/ca/registry/ca/
pki-tomcat/ca/registry/ca/default.cfg
.....
```

2. Stop the subsystem instance if it is not already stopped.

```
pki-server stop instance_name
```

3. Copy the archived files to restore the instance directory:

```
cp -r /export/archives/pki/instance_name /var/lib/pki/instance_name
```

For example:

```
# cp -r /tmp/pki-tomcat/ca/ /var/lib/pki/pki-tomcat/ca/
```

4. Make sure the ownership and group permissions of the restored files are set to the **pkiuser**:

```
# chown -R pkiuser:pkiuser /var/lib/pki/pki-tomcat/ca/
```

5. Restart the subsystem instance.

```
pki-server start instance_name
```

14.9. RUNNING SELF-TESTS

The Certificate System has the added functionality to allow self-tests of the server. The self-tests are run at start up and can also be run on demand. The startup self-tests run when the server starts and keep the server from starting if a critical self-test fails. The on-demand self-tests are run by clicking the self-tests button in the subsystem console.

14.9.1. Running Self-Tests

The on-demand self-test for the CA, OCSP, KRA, or TKS subsystems are run from the console. The on-demand self-tests for the TPS system are run from the web services page.

14.9.1.1. Running Self-Tests from the Console

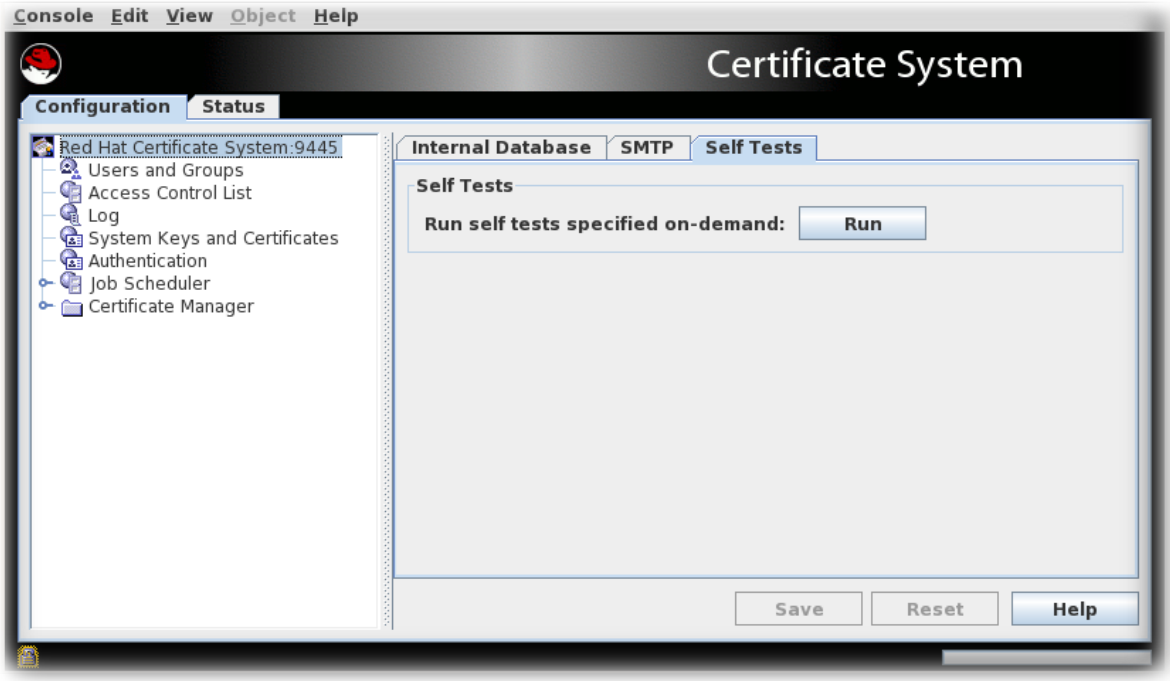
NOTE

pkiconsole is being deprecated.

1. Log into the Console.

```
pkiconsole https://server.example.com:admin_port/subsystem_type
```

2. Select the subsystem name at the top of the left pane.



3. Select the **Self Tests** tab.
4. Click **Run**.

   The self-tests that are configured for the subsystem will run. If any critical self-tests fail, the server will stop.
5. The **On-Demand Self Tests Results** window appears, showing the logged events for this run of the self-tests.

14.9.1.2. Running TPS Self-Tests

To run TPS self-tests from the command-line interface (CLI):

- **pki tps-selftest-find**
- **pki tps-selftest-run**
- **pki tps-selftest-show**

14.9.2. Self-Test Logging

A separate log, **selftest.log**, is added to the log directory that contains reports for both the start up self-tests and the on-demand self-tests. This log is configured by changing the setting for the log in the **CS.cfg** file. See the *Modifying Self-Test Configuration* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* for details.

14.9.3. Configuring POSIX System ACLs

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured. For more details on ACLs, see the corresponding chapter in the Red Hat Enterprise Linux System Administration Guide.

14.9.3.1. Setting POSIX System ACLs for the CA, KRA, OCSP, TKS, and TPS

Modern file systems like ext4 and XFS enable ACLs by default, and are most likely used on modern Red Hat Enterprise Linux installations.

1. Stop the instance.

```
pki-server stop instance_name
```

2. Set the group readability to the pkiadmin group for the instance's directories and files.

```
# setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki/instance_name
```

3. Apply execute (x) ACL permissions on all directories:

```
# find -L /var/lib/pki/instance_name -type d -exec setfacl -L -n -m
g:pkiadmin:rx,d:g:pkiadmin:rx {} \;
```

4. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/pki/instance_name/logs/signedAudit
```

5. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki/instance_name/logs/signedAudit
```

6. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:

```
# find -L /var/lib/pki/instance_name/logs/signedAudit -type d -exec setfacl -L -n -m
g:pkiaudit:rx,d:g:pkiaudit:rx {} \;
```

7. Start the instance.

```
pki-server start instance_name
```

8. Confirm that the file access controls were properly applied by using the **getfacl** command to show the current ACL settings:

```
# getfacl /var/lib/pki/instance_name
/var/lib/pki/instance_name/subsystem_type/logs/signedAudit/
getfacl: Removing leading '/' from absolute path names
# file: var/lib/pki/instance_name
# owner: pkiuser
# group: pkiuser
user::rwx
group::rwx
group:pkiadmin:r-x
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
```

```
default:group:pkiadmin:r-x
default:mask::rwx
default:other::r-x

# file: var/lib/pki/instance_name/logs/signedAudit
# owner: pkiuser
# group: pkiaudit
user::rwx
group::rwx
group:pkiaudit:r-x
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:pkiaudit:r-x
default:mask::rwx
default:other::---
```

CHAPTER 15. MANAGING CERTIFICATE SYSTEM USERS AND GROUPS

This chapter explains how to set up authorization for access to the administrative, agent services, and end-entities pages.

15.1. ABOUT AUTHORIZATION

Authorization is the process of allowing access to certain tasks associated with the Certificate System. Access can be limited to allow certain tasks to certain areas of the subsystem for certain users or groups and different tasks to different users and groups.

Users are specific to the subsystem in which they are created. Each subsystem has its own set of users independent of any other subsystem installed. The users are placed in groups, which can be predefined or user-created. Privileges are assigned to a group through *access control lists* (ACLs). There are ACLs associated with areas in the administrative console, agent services interface, and end-entities page that perform an authorization check before allowing an operation to proceed. *Access control instructions* (ACIs) in each of the ACLs are created that specifically allow or deny possible operations for that ACL to specified users, groups, or IP addresses.

The ACLs contain a default set of ACIs for the default groups that are created. These ACIs can be modified to change the privileges of predefined groups or to assign privileges to newly-created groups.

Authorization goes through the following process:

1. The users authenticate to the interface using either the Certificate System user ID and password or a certificate.

2. The server authenticates the user either by matching the user ID and password with the one stored in the database or by checking the certificate against one stored in the database. With certificate-based authentication, the server also checks that the certificate is valid and finds the group membership of the user by associating the DN of the certificate with a user and checking the user entry. With password-based authentication, the server checks the password against the user ID and then finds the group membership of the user by associating that user ID with the user ID contained in the group.

3. When the user tries to perform an operation, the authorization mechanism compares the user ID of the user, the group in which the user belongs, or the IP address of the user to the ACLs set for that user, group, or IP address. If an ACL exists that allows that operation, then the operation proceeds.

15.2. DEFAULT GROUPS

A user's privileges are determined by the group (role) membership of the user. There are three groups (roles) that a user can be assigned to:

- *Administrators.* This group is given full access to all of the tasks available in the administrative interface.

- *Agents.* This group is given full access to all of the tasks available in the agent services interface.

- *Auditors.* This group is given access to view the signed audit logs. This group does not have any other privileges.

There is a fourth role that is exclusively created for communication between subsystems. Administrators should never assign a real user to such a role:

- *Enterprise administrators.* Each subsystem instance is automatically assigned a subsystem-specific role as an enterprise administrator when it is joined to a security domain during configuration. These roles automatically provide trusted relationships among subsystems in the security domain, so that each subsystem can efficiently carry out interactions with other subsystems.

15.2.1. Administrators

Administrators have permissions to perform all administrative tasks. A user is designated or identified as being an administrator by being added to the **Administrators** group for the group. Every member of that group has administrative privileges for that instance of Certificate System.

At least one administrator must be defined for each Certificate System instance, but there is no limit to the number of administrators an instance can have. The first administrator entry is created when the instance is configured.

Administrators are authenticated with a simple bind using their Certificate System user ID and password.

Table 15.1. Security Domain User Roles

Role	Description
Security Domain Administrators	• Add and modify users in the security domain's user and group database. • Manage the shared trust policies. • Manage the access controls on the domain services. By default, the CA administrator of the CA hosting the domain is assigned as the security domain administrator.
Enterprise CA Administrators	• Automatically approve any sub-CA, server, and subsystem certificate from any CA in the domain. • Register and unregister CA subsystem information in the security domain.
Enterprise KRA Administrators	• Automatically approve any transport, storage, server, and subsystem certificate from any CA in the domain. • Register and unregister KRA subsystem information in the security domain. • Push KRA connector information to any CA.

Role	Description
Enterprise OCSP Administrators	• Automatically approve any OCSP, server, and subsystem certificate from any CA in the domain. • Register and unregister OCSP subsystem information in the security domain. • Push CRL publishing information to any CA.
Enterprise TKS Administrators	• Automatically approve any server and subsystem certificate from any CA in the domain. • Register and unregister TKS subsystem information in the security domain.
Enterprise TPS Administrators	• Automatically approve any server and subsystem certificate from any CA in the domain. • Register and unregister TPS subsystem information in the security domain.

As necessary, the security domain administrator can manage access controls on the security domain and on the individual subsystems. For example, the security domain administrator can restrict access so that only finance department KRA administrators can set up finance department KRAs.

Enterprise subsystem administrators are given enough privileges to perform operations on the subsystems in the domain. For example, an enterprise CA administrator has the privileges to have sub-CA certificates approved automatically during configuration. Alternatively, a security domain administrator can restrict this right if necessary.

15.2.2. Auditors

An auditor can view the signed audit logs and is created to audit the operation of the system. The auditor cannot administer the server in any way.

An auditor is created by adding a user to the **Auditors** group and storing the auditor's certificate in the user entry. The auditor's certificate is used to encrypt the private key of the key pair used to sign the audit log.

The **Auditors** group is set when the subsystem is configured. No auditors are assigned to this group during configuration.

Auditors are authenticated into the administrative console with a simple bind using their UID and password. Once authenticated, auditors can only view the audit logs. They cannot edit other parts of the system.

15.2.3. Agents

Agents are users who have been assigned end-entity certificate and key-management privileges. Agents can access the agent services interface.

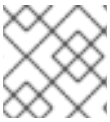
Agents are created by assigning a user to the appropriate subsystem agent group and identifying certificates that the agents must use for SSL client authentication to the subsystem for it to service requests from the agents. Each subsystem has its own agent group:

- The Certificate Manager Agents group.
- The Key Recovery Authority Agents group.
- The Online Certificate Status Manager Agents group.
- The Token Key Service Agents group.
- The Token Processing System Agents group.

Each Certificate System subsystem has its own agents with roles defined by the subsystem. Each subsystem must have at least one agent, but there is no limit to the number of agents a subsystem can have.

Certificate System identifies and authenticates a user with agent privileges by checking the user's SSL client certificate in its internal database.

15.2.4. Enterprise Groups



NOTE

No real user should ever be assigned to this group.

During subsystem configuration, every subsystem instance is joined to a security domain. Each subsystem instance is automatically assigned a subsystem-specific role as an enterprise administrator. These roles automatically provide trusted relationships among subsystems in the security domain, so that each subsystem can efficiently carry out interactions with other subsystems. For example, this allows OCSPs to push CRL publishing publishing information to all CAs in the domain, KRAs to push KRA connector information, and CAs to approve certificates generated within the CA automatically.

Enterprise subsystem administrators are given enough privileges to perform operations on the subsystems in the domain. Each subsystem has its own security domain role:

- Enterprise CA Administrators
- Enterprise KRA Administrators
- Enterprise OCSP Administrators
- Enterprise TKS Administrators
- Enterprise TPS Administrators

Additionally, there is a Security Domain Administrators group for the CA instance which manages the security domain, access control, users, and trust relationships within the domain.

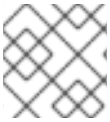
Each subsystem administrator authenticates to the other subsystems using SSL client authentication with the subsystem certificate issued during configuration by the security domain CA.

15.3. MANAGING USERS AND GROUPS FOR A CA, OCSP, KRA, OR TKS

Many of the operations that users can perform are dictated by the groups that they belong to; for instance, agents for the CA manage certificates and profiles, while administrators manage CA server configuration.

Four subsystems – the CA, OCSP, KRA, and TKS – use the Java administrative console to manage groups and users. The TPS has web-based admin services, and users and groups are configured through its web service page.

15.3.1. Managing Groups



NOTE

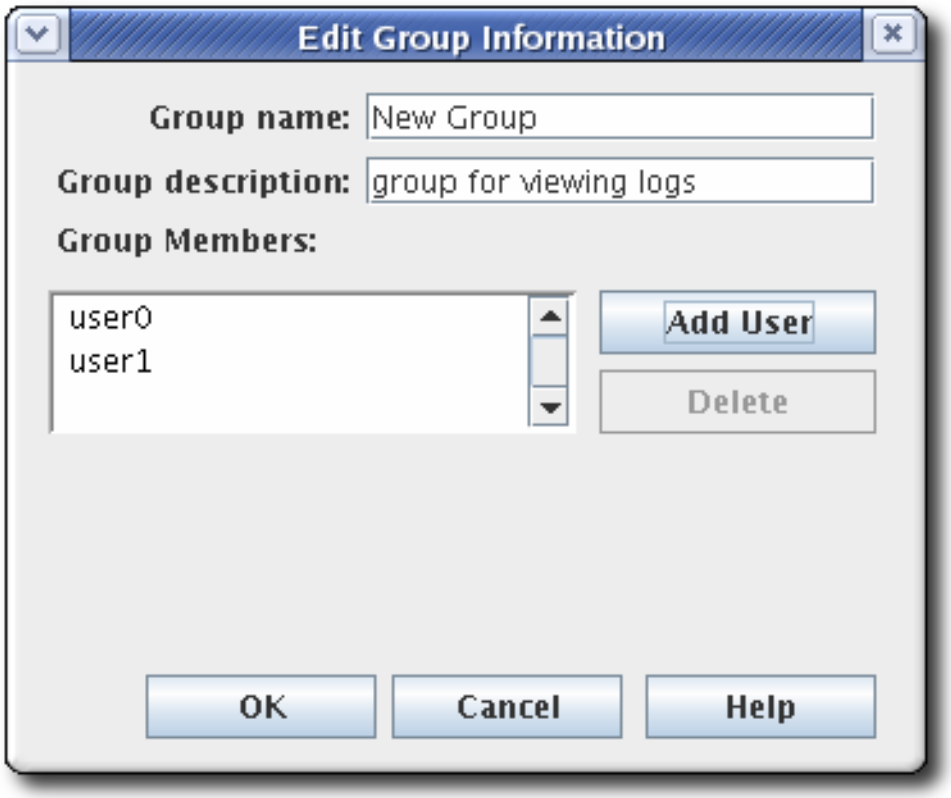
pkiconsole is being deprecated.

15.3.1.1. Creating a New Group

1. Log into the administrative console.

```
pkiconsole https://server.example.com:8443/subsystem_type
```

2. Select **Users and Groups** from the navigation menu on the left.
3. Select the **Groups** tab.
4. Click **Edit**, and fill in the group information.



It is only possible to add users who already exist in the internal database.

5. Edit the ACLs to grant the group privileges. See Section 15.5.4, "Editing ACLs" for more information. If no ACIs are added to the ACLs for the group, the group will have no access permissions to any part of Certificate System.

15.3.1.2. Changing Members in a Group

Members can be added or deleted from all groups. The group for administrators must have at least one user entry.

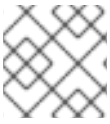
1. Log into the administrative console.
2. Select **Users and Groups** from the navigation tree on the left.
3. Click the **Groups** tab.
4. Select the group from the list of names, and click **Edit**.
5. Make the appropriate changes.
   - To change the group description, type a new description in the **Group description** field.
   - To remove a user from the group, select the user, and click **Delete**.
   - To add users, click **Add User**. Select the users to add from the dialog box, and click **OK**.

15.3.2. Managing Users (Administrators, Agents, and Auditors)

The users for each subsystem are maintained separately. Just because a person is an administrator in one subsystem does not mean that person has any rights (or even a user entry) for another subsystem. Users can be configured and, with their user certificates, trusted as agents, administrators, or auditors for a subsystem.

15.3.2.1. Creating Users

After you installed Certificate System, only the user created during the setup exists. This section describes how to create additional users.



NOTE

For security reasons, create individual accounts for Certificate System users.

15.3.2.1.1. Creating Users Using the Command Line

To create a user using the command line:

1. Add a user account. For example, to add the **example** user to the CA:

```
# pki -d ~/.dogtag/pki-instance_name/ca/alias/ -c password -n caadmin \
    ca-user-add example --fullName "Example User"
----------------------
Added user "example"
```

```
----------------------
 User ID: example
 Full name: Example User
```

This command uses the **caadmin** user to add a new account.

2. Optionally, add a user to a group. For example, to add the **example** user to the **Certificate Manager Agents** group:

```
# pki -d ~/.dogtag/pki-instance_name/ -p password -n "caadmin" \
    user-add-membership example Certificate Manager Agents
```

3. Create a certificate request:

   - If a Key Recovery Authority (KRA) exists in your Certificate System environment:

```
# CRMFPopClient -d ~/.dogtag/pki-instance_name/ -p password \
    -n "user_name" -q POP_SUCCESS -b kra.transport -w "AES/CBC/PKCS5Padding" \
    -v -o ~/user_name.req
```

     This command stores the Certificate Signing Request (CSR) in the **CRMF** format in the ***~/user_name.req*** file.

   - If no Key Recovery Authority (KRA) exists in your Certificate System environment:

     - Create a NSS database directory:

```
# export pkiinstance=ca1
# echo ${pkiinstance}
# export agentdir=~/.dogtag/${pkiinstance}/agent1.dir
# echo ${agentdir}
# pki -d ${agentdir}/ -C ${somepwdfile} client-init
```

     - Store the CSR in a PKCS-#10 formatted file specified by the **-o** option, **-d** for the path to an initialized NSS database directory, **-P** option for a password file, **-p** for a password, and **-n** for a subject DN:

```
# PKCS10Client -d ${agentdir}/ -P ${somepwdfile} -n "cn=agent1,uid=agent1" -o
${agentdir}/agent1.csr
PKCS10Client: Certificate request written into /.dogtag/ca1/agent1.dir/agent1.csr
PKCS10Client: PKCS#10 request key id written into
/.dogtag/ca1/agent1.dir/agent1.csr.keyId
```

4. Create an enrollment request:

   a. Create the ***~/cmc.role_crmf.cfg*** file with the following content:

```
#numRequests: Total number of PKCS10 requests or CRMF requests.
numRequests=1

#input: full path for the PKCS10 request or CRMF request,
#the content must be in Base-64 encoded format
#Multiple files are supported. They must be separated by space.
input=~/user_name.req
```

```
#output: full path for the CMC request in binary format
output=~/cmc.role_crmf.req

#tokenname: name of token where agent signing cert can be found (default is internal)
tokenname=internal

#nickname: nickname for agent certificate which will be used
#to sign the CMC full request.
nickname=PKI Administrator for Example.com

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
dbdir=~/.dogtag/pki-instance_name/

#password: password for cert9.db which stores the agent
#certificate
password=password

#format: request format, either pkcs10 or crmf
format=crmf
```

Set the parameters based on your environment and the CSR format used in the previous step.

b. Pass the previously created configuration file to the **CMCRequest** utility to create the CMC request:

```
# CMCRequest ~/cmc.role_crmf.cfg
```

5. Submit a Certificate Management over CMS (CMC) request:

a. Create the ***~/HttpClient_role_crmf.cfg*** file with the following content:

```
# #host: host name for the http server
host=server.example.com

#port: port number
port=8443

#secure: true for secure connection, false for nonsecure connection
secure=true

#input: full path for the enrollment request, the content must be in binary format
input=~/cmc.role_crmf.req

#output: full path for the response in binary format
output=~/cmc.role_crmf.resp

#tokenname: name of token where SSL client authentication cert can be found (default is
internal)
#This parameter will be ignored if secure=false
tokenname=internal

#dbdir: directory for cert9.db, key4.db and pkcs11.txt
#This parameter will be ignored if secure=false
```

```
dbdir=~/.dogtag/pki-instance_name/

#clientmode: true for client authentication, false for no client authentication
#This parameter will be ignored if secure=false
clientmode=true

#password: password for cert9.db
#This parameter will be ignored if secure=false and clientauth=false
password=password

#nickname: nickname for client certificate
#This parameter will be ignored if clientmode=false
nickname=PKI Administrator for Example.com

#servlet: servlet name
servlet=/ca/ee/ca/profileSubmitCMCFull
```

Set the parameters based on your environment.

b. Submit the request to the CA:

```
# HttpClient ~/HttpClient_role_crmf.cfg
Total number of bytes read = 3776
after SSLSocket created, thread token is Internal Key Storage Token
client cert is not null
handshake happened
writing to socket
Total number of bytes read = 2523
MIIJ1wYJKoZIhvcNAQcCoIIJyDCCCcQCAQMxDzANBglghkgBZQMEAgEFADAxBggr
...
The response in data format is stored in ~/cmc.role_crmf.resp
```

c. Verify the result:

```
# CMCResponse ~/cmc.role_crmf.resp
Certificates:
   Certificate:
      Data:
         Version:  v3
         Serial Number: 0xE
         Signature Algorithm: SHA256withRSA - 1.2.840.113549.1.1.11
         Issuer: CN=CA Signing Certificate,OU=pki-instance_name Security Domain
         Validity:
            Not Before: Friday, July 21, 2017 12:06:50 PM PDT America/Los_Angeles
            Not  After: Wednesday, January 17, 2018 12:06:50 PM PST
America/Los_Angeles
         Subject: CN=user_name
...
Number of controls is 1
Control #0: CMCStatusInfoV2
  OID: {1 3 6 1 5 5 7 7 25}
  BodyList: 1
  Status: SUCCESS
```

6. Optionally, to import the certificate as the user to its own **~/.dogtag/pki-instance_name/** database:

```
# certutil -d ~/.dogtag/pki-instance_name/ -A -t "u,u,u" -n "user_name certificate" -i
~/cmc.role_crmf.resp
```

7. Add the certificate to the user record:

   a. List certificates issued for the user to discover the certificate's serial number. For example, to list certificates that contain the **example** user name in the certificate's subject:

```
pki -d ~/.dogtag/pki-instance_name/ -c password -n caadmin ca-user-cert-find example
-----------------
1 entries matched
-----------------
  Cert ID: 2;6;CN=CA Signing Certificate,O=EXAMPLE;CN=PKI
Administrator,E=example@example.com,O=EXAMPLE
  Version: 2
  Serial Number: 0x6
  Issuer: CN=CA Signing Certificate,O=EXAMPLE
  Subject: CN=PKI Administrator,E=example@example.com,O=EXAMPLE
----------------------------
Number of entries returned 1
```

   The serial number of the certificate is required in the next step.

   b. Add the certificate using its serial number from the certificate repository to the user account in the Certificate System database. For example, for a CA user:

```
pki -c password -n caadmin ca-user-cert-add example --serial 0x6
```

15.3.2.1.2. Creating Users Using the Console

NOTE

pkiconsole is being deprecated.

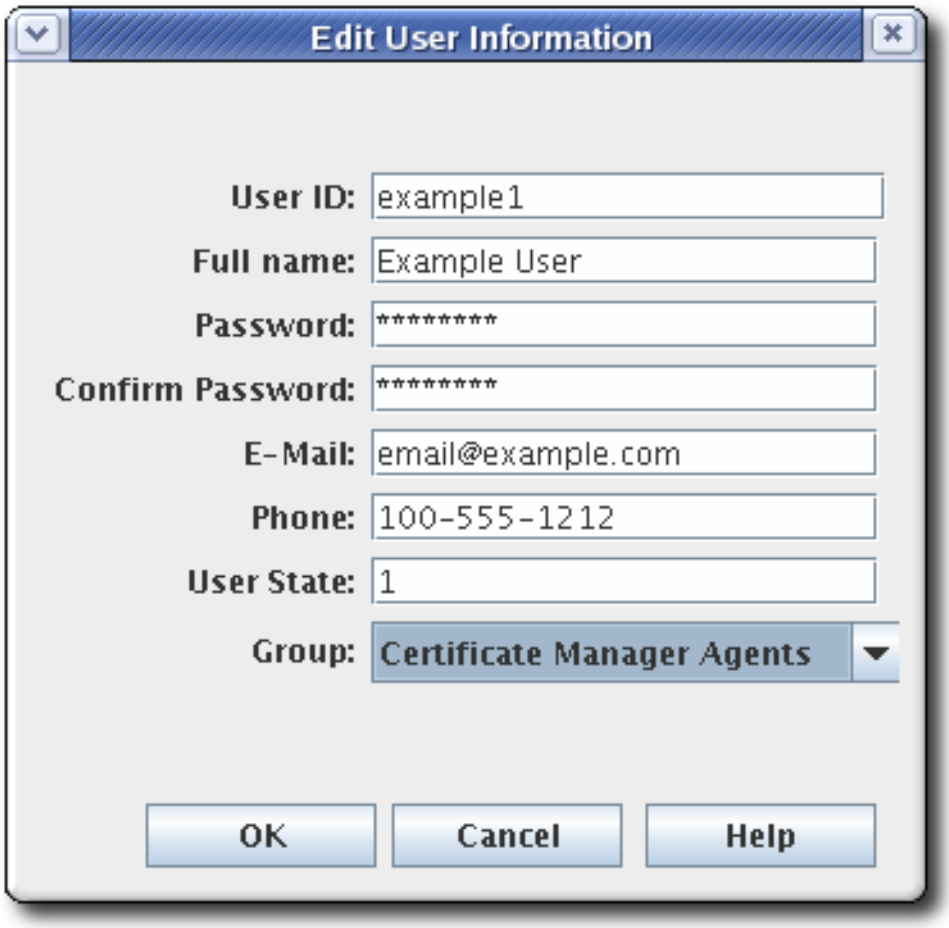
To create a user using the PKI Console:

1. Log into the administrative console.

```
pkiconsole https://server.example.com:8443/subsystem_type
```

2. In the **Configuration** tab, select **Users and Groups**. Click **Add**.

3. Fill in the information in the **Edit User Information** dialog.



Most of the information is standard user information, such as the user's name, email address, and password. This window also contains a field called **User State**, which can contain any string, which is used to add additional information about the user; most basically, this field can show whether this is an active user.

4. Select the group to which the user will belong. The user's group membership determines what privileges the user has. Assign agents, administrators, and auditors to the appropriate subsystem group.

5. Store the user's certificate.

    1. Request a user certificate through the CA end-entities service page.

    2. If auto-enrollment is not configured for the user profile, then approve the certificate request.

    3. Retrieve the certificate using the URL provided in the notification email, and copy the base-64 encoded certificate to a local file or to the clipboard.

    4. Select the new user entry, and click **Certificates**.

    5. Click **Import**, and paste in the base-64 encoded certificate.

15.3.2.2. Changing a Certificate System User's Certificate

1. Log into the administrative console.

2. Select **Users and Groups**.

3. Select the user to edit from the list of user IDs, and click **Certificates**.

4. Click **Import** to add the new certificate.

5. In the **Import Certificate** window, paste the new certificate in the text area. Include the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** marker lines.

15.3.2.3. Renewing Administrator, Agent, and Auditor User Certificates

There are two methods of renewing a certificate. *Regenerating* the certificate takes its original key and its original profile and request, and recreates an identical key with a new validity period and expiration date. *Re-keying* a certificate resubmits the initial certificate request to the original profile, but generates a new key pair. Administrator certificates can be renewed by being re-keyed.

Each subsystem has a bootstrap user that was created at the time the subsystem was created. A new certificate can be requested for this user before their original one expires, using one of the default renewal profiles.

Certificates for administrative users can be renewed directly in the end user enrollment forms, using the serial number of the original certificate.

1. Renew the admin user certificates in the CA's end users forms, as described in Section 5.4.1.1.2, "Certificate-Based Renewal". This must be the same CA as first issued the certificate (or a clone of it).

   Agent certificates can be renewed by using the certificate-based renewal form in the end entities page. **Self-renew user SSL client certificate**. This form recognizes and updates the certificate stored in the browser's certificate store directly.

   NOTE

   It is also possible to renew the certificate using **certutil**, as described in Section 17.3.3, "Renewing Certificates Using certutil" . Rather than using the certificate stored in a browser to initiate renewal, **certutil** uses an input file with the original key.

2. Add the renewed user certificate to the user entry in the internal LDAP database.

   1. Open the console for the subsystem.

      ```
      pkiconsole https://server.example.com:admin_port/subsystem_type
      ```

   2. Configuration | Users and Groups | Users | admin | Certificates | Import

   3. In the **Configuration** tab, select **Users and Groups**.

   4. In the **Users** tab, double-click the user entry with the renewed certificate, and click **Certificates**.

   5. Click **Import**, and paste in the base-64 encoded certificate.

   NOTE

   pkiconsole is being deprecated.

This can also be done by using **ldapmodify** to add the renewed certification directly to the user entry in the internal LDAP database, by replacing the ***userCertificate*** attribute in the user entry, such as **uid=admin,ou=people,dc=***subsystem-base-DN*.

15.3.2.4. Renewing an Expired Administrator, Agent, and Auditor User Certificate

When a valid user certificate has already expired, you can no longer use the web service page nor the **pki** command-line tool requiring authentication. In such a scenario, you can use the **pki-server cert-fix** command to renew an expired certificate.

Before you proceed, make sure:

- You have a valid CA certificate.
- You have root privileges.

Procedure 15.1. Renewing an Expired Administrator, Agent, and Auditor User Certificate

1. Disable self test.
   - Either run the following command:

     ```
     # pki-server selftest-disable -i PKI_instance
     ```

   - Or remove the following line from CA's **CS.cfg** file and restart the CA subsystem:

     ```
     selftests.container.order.startup=CAPresence:critical, SystemCertsVerification:critical
     ```

2. Check the expired certificates in the client's NSS database and find the certificate's serial number (certificate ID).
   a. List the user certificates:

      ```
      # certutil -L -d /root/nssdb/
      ```

   b. Get the expired certificate serial number, which you want to renew:

      ```
      # certutil -L -d /root/nssdb/ -n Expired_cert | grep Serial
          Serial Number: 16 (0x10)
      ```

3. Renew the certificate. The local LDAP server requires the LDAP Directory Manager's password.

   ```
   # pki-server cert-fix --ldap-url ldap://host389 --agent-uid caadmin -i PKI_instance -p
   PKI_https_port --extra-cert 16
   ```

4. Re-eanable self test.
   - Either run the following command:

     ```
     # pki-server selftest-enable -i PKI_instance
     ```

   - Or add the following line to CA's **CS.cfg** file and restart the CA subsystem:

     ```
     ```

```
selftests.container.order.startup=CAPresence:critical, SystemCertsVerification:critical
```

To verify that you have succeeded in the certificate renewal, you can display sufficient information about the certificate by running:

```
# pki ca-cert-find
```

To see full details of the specific certificate including attributes, extensions, public key modulus, hashes, and more, you can also run:

```
# pki ca-cert-show 16 --pretty
```

15.3.2.5. Deleting a Certificate System User

Users can be deleted from the internal database. Deleting a user from the internal database deletes that user from all groups to which the user belongs. To remove the user from specific groups, modify the group membership.

Delete a privileged user from the internal database by doing the following:

1. Log into the administrative console.
2. Select **Users and Groups** from the navigation menu on the left.
3. Select the user from the list of user IDs, and click **Delete**.
4. Confirm the delete when prompted.

15.4. CREATING AND MANAGING USERS FOR A TPS

There are three defined *roles* for TPS users, which function as groups for the TPS:

- *Agents*, who perform actual token management operations, such setting the token status and changing token policies
- *Administrators*, who manage users for the TPS subsystem and have limited control over tokens
- *Operators*, who have no management control but are able to view and list tokens, certificates, and activities performed through the TPS

Additional groups cannot be added for the TPS.

All of the TPS subsystem users are authenticated against an LDAP directory database that contains their certificate (because accessing the TPS's web services requires certificate-based authentication), and the authentication process checks the TPS group entries – **ou=TUS Agents**, **ou=TUS Administrators**, and **ou=TUS Operators** – to see to which roles the user belongs, using Apache's **mod_tokendb** module.

Users for the TPS are added and managed through the Web UI or the CLI. The Web UI is accessible at **https://*server.example.com*:8443/tps/ui/**.

To use the Web UI or the CLI, the TPS administrator has to authenticate using a user certificate.

15.4.1. Listing and Searching for Users

15.4.1.1. From the Web UI

To list users from the Web UI:

1. Click the **Accounts** tab.
2. Click the **Users** menu item. The list of users appears on the page.
3. To search for certain users, write the keyword in the search field and press **Enter**. To list all users again, remove the keyword and press **Enter**.

15.4.1.2. From the Command Line

To list users from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-find
```

To view user details from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-show username
```

15.4.2. Adding Users

15.4.2.1. From the Web UI

To add a user from the Web UI:

1. Click the **Accounts** tab.
2. Click the **Users** menu item.
3. Click the **Add** button on the **Users** page.
4. Fill in the user ID, full name, and TPS profile.
5. Click the **Save** button.

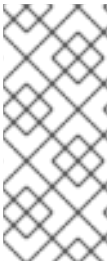
15.4.2.1.1. From the Command Line

To add a user from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-add username --
fullName full_name
```

15.4.3. Setting Profiles for Users

A TPS profile is much like a CA profile; it defines rules for processing different types of tokens. The profile is assigned automatically to a token based on some characteristic of the token, like the CUID. Users can only see tokens for the profiles which are assigned to them.



NOTE

A user can only see entries relating to the profile configured for it, including both token operations and tokens themselves. For an administrator to be able to search and manage all tokens configured in the TPS, the administrator user entry should be set to **All profiles**. Setting specific profiles for users is a simple way to control access for operators and agents to specific users or token types.

Token profiles are sets of policies and configurations that are applied to a token. Token profiles are mapped to tokens automatically based on some kind of attribute in the token itself, such as a CCUID range. Token profiles are created as other certificate profiles in the CA profile directory and are then added to the TPS configuration file, **CS.cfg**, to map the CA's token profile to the token type. Configuring token mapping is covered in Section 6.7, “Mapping Resolver Configuration”.

To manage user profiles from the Web UI:

1. Click the **Accounts** tab.
2. Click the **Users** menu item.
3. Click the user name of the user you want to modify.
4. Click the **Edit** link.
5. In the **TPS Profile** field, enter the profile names separated by commas, or enter **All Profiles**.
6. Click the **Save** button.

15.4.4. Managing User Roles

A role is just a group within the TPS. Each role can view different tabs of the TPS services pages. The group is editable, so it is possible to add and remove role assignments for a user.

A user can belong to more than one role or group. The bootstrap user, for example, belongs to all three groups.

15.4.4.1. From the Web UI

To manage group members from the Web UI:

1. Click the **Accounts** tab.
2. Click the **Groups** menu item.
3. Click the name of the group that you want to change, for example TPS Agents.
4. To add a user to this group:
    a. Click the **Add** button.
    b. Enter the user ID.
    c. Click the **Add** button.
5. To remove a user from this group:

a. Select the check box next to the user.

b. Click the **Remove** button.

c. Click the **OK** button.

15.4.4.2. From the Command Line

To list groups from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-group-find
```

To list group members from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-group-member-find
group_name
```

To add a user to a group from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-group-member-add
group_name user_name
```

To delete a user from a group from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-group-member-del
group_name user_name
```

15.4.5. Managing User Certificates

User certificates can be managed from the CLI:

- To list user certificates, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-cert-find
user_name
```

- To add a certificate to a user:

1. Obtain a user certificate for the new user. Requesting and submitting certificates is explained in Chapter 5, *Requesting, Enrolling, and Managing Certificates*.

IMPORTANT

A TPS administrator must have a signing certificate. The recommended profile to use is Manual User Signing and Encryption Certificates Enrollment.

2. Run the following command:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-cert-add
user_name --serial cert_serial_number
```

- To remove a certificate from a user, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-cert-del
user_name cert_id
```

15.4.6. Renewing TPS Agent and Administrator Certificates

Regenerating the certificate takes its original key and its original profile and request, and recreates an identical key with a new validity period and expiration date.

The TPS has a bootstrap user that was created at the time the subsystem was created. A new certificate can be requested for this user when their original one expires, using one of the default renewal profiles.

Certificates for administrative users can be renewed directly in the end user enrollment forms, using the serial number of the original certificate.

1. Renew the user certificates through the CA's end users forms, as described in Section 5.4.1.1.2, "Certificate-Based Renewal". This must be the same CA as first issued the certificate (or a clone of it).

   Agent certificates can be renewed by using the certificate-based renewal form in the end entities page, **Self-renew user SSL client certificate**. This form recognizes and updates the certificate stored in the browser's certificate store directly.

   NOTE

   It is also possible to renew the certificate using **certutil**, as described in Section 17.3.3, "Renewing Certificates Using certutil" . Rather than using the certificate stored in a browser to initiate renewal, **certutil** uses an input file with the original key.

2. Add the new certificate to the user and remove the old certificate as described in Section 15.4.5, "Managing User Certificates".

15.4.7. Deleting Users

WARNING

It is possible to delete the *last* user account, and the operation cannot be undone. Be very careful about the user which is selected to be deleted.

To delete users from the Web UI:

1. Click the **Accounts** tab.
2. Click the **Users** menu item.
3. Select the check box next to the users to be deleted.
4. Click the **Remove** button.

5. Click the **OK** button.

To delete a user from the CLI, run:

```
pki -d client_db_dir -c client_db_password -n admin_cert_nickname tps-user-del user_name
```

15.5. CONFIGURING ACCESS CONTROL FOR USERS

Authorization is the mechanism that checks whether a user is allowed to perform an operation. Authorization points are defined in certain groups of operations that require an authorization check.

15.5.1. About Access Control

Access control lists (ACLs) are the mechanisms that specify the authorization to server operations. An ACL exists for each set of operations where an authorization check occurs. Additional operations can be added to a ACL.

The ACL contains *access control instructions* (ACIs) which specifically allow or deny operations, such as read or modify. The ACI also contains an evaluator expression. The default implementation of ACLs specifies only users, groups, and IP addresses as possible evaluator types. Each ACI in an ACL specifies whether access is allowed or denied, what the specific operator is being allowed or denied, and which users, groups, or IP addresses are being allowed or denied to perform the operation.

The privileges of Certificate System users are changed by changing the access control lists (ACL) that are associated with the group in which the user is a member, for the users themselves, or for the IP address of the user. New groups are assigned access control by adding that group to the access control lists. For example, a new group for administrators who are only authorized to view logs, **LogAdmins**, can be added to the ACLs relevant to logs to allow read or modify access to this group. If this group is not added to any other ACLs, members of this group only have access to the logs.

The access for a user, group, or IP address is changed by editing the ACI entries in the ACLs. In the ACL interface, each ACI is shown on a line of its own. In this interface window, the ACI has the following syntax:

```
allow|deny (operation) user|group|IP="name"
```

NOTE

The IP address can be an IPv4 or IPv6 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*.

For example, the following is an ACI that allows administrators to perform read operations:

```
allow (read) group="Administrators"
```

An ACI can have more than one operation or action configured. The operations are separated with a comma with no space on either side. For example:

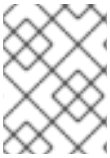
```
allow (read,modify) group="Administrators"
```

An ACI can have more than one group, user, or IP address by separating them with two pipe symbols (||) with a space on either side. For example:

```
allow (read) group="Administrators" || group="Auditors"
```

The administrative console can create or modify ACIs. The interface sets whether to allow or deny the operation in the **Allow and Deny** field, sets which operations are possible in the **Operations** field, and then lists the groups, users, or IP addresses being granted or denied access in the **Syntax** field.

An ACI can either allow or deny an operation for the specified group, user ID, or IP address. Generally, ACIs do not need to be created to deny access. If there are no allow ACIs that include a user ID, group, or IP address, then the group, user ID, or IP address is denied access.



NOTE

If a user is not explicitly allowed access to any of the operations for a resource, then this user is considered denied; he does not specifically need to be denied access.

For example, user JohnB is a member of the **Administrators** group. If an ACL has only the following ACL, JohnB is denied any access since he does not match any of the allow ACIs:

```
Allow (read,modify) group="Auditors" || user="BrianC"
```

There usually is no need to include a deny statement. Some situations can arise, however, when it is useful to specify one. For example, **JohnB**, a member of the **Administrators** group, has just been fired. It may be necessary to deny access specifically to **JohnB** if the user cannot be deleted immediately. Another situation is that a user, **BrianC**, is an administrator, but he should not have the ability to change some resource. Since the **Administrators** group must access this resource, **BrianC** can be specifically denied access by creating an ACI that denies this user access.

The allowed rights are the operations which the ACI controls, either by allowing or denying permission to perform the operation. The actions that can be set for an ACL vary depending on the ACL and subsystem. Two common operations that can be defined are read and modify.

The syntax field of the ACI editor sets the evaluator for the expression. The evaluator can specify group, name, and IP address (both IPv4 and IPv6 addresses). These are specified along with the name of the entity set as equals (**=**) or does not equal (**!=**).

The syntax to include a group in the ACL is **group="groupname"**. The syntax to exclude a group is **group!="groupname"**, which allows any group except for the group named. For example:

```
group="Administrators" || group!="Auditors"
```

It is also possible to use regular expressions to specify the group, such as using wildcard characters like an asterisk (*). For example:

```
group="* Managers"
```

For more information on supported regular expression patterns, see https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html.

The syntax to include a user in the ACL is **user="userID"**. The syntax to exclude the user is **user!="userID"**, which allows any user ID except for the user ID named. For example:

```
user="BobC" || user!="JaneK"
```

To specify all users, provide the value **anybody**. For example:

```
user="anybody"
```

It is also possible to use regular expressions to specify the user names, such as using wildcard characters like an asterisk (*). For example:

```
user="*johnson"
```

For more information on supported regular expression patterns, see https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html.

The syntax to include an IP address in the ACL is **ipaddress="ipaddress"**. The syntax to exclude an ID address from the ACL is **ipaddress!="ipaddress"**. An IP address is specified using its numeric value; DNS values are not permitted. For example:

```
ipaddress="12.33.45.99"
ipaddress!="23.99.09.88"
```

The IP address can be an IPv4 address, as shown above, or IPv6 address. An IPv4 address has the format *n.n.n.n* or *n.n.n.n,m.m.m.m* with the netmask. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example:

```
ipaddress="0:0:0:0:0:0:13.1.68.3"
```

It is also possible to use regular expressions to specify the IP address, such as using wildcard characters like an asterisk (*). For example:

```
ipaddress="12.33.45.*"
```

For more information on supported regular expression patterns, see https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html.

It is possible to create a string with more than one value by separating each value with two pipe characters (||) with a space on either side. For example:

```
user="BobC" || group="Auditors" || group="Administrators"
```

15.5.2. Changing the Access Control Settings for the Subsystem

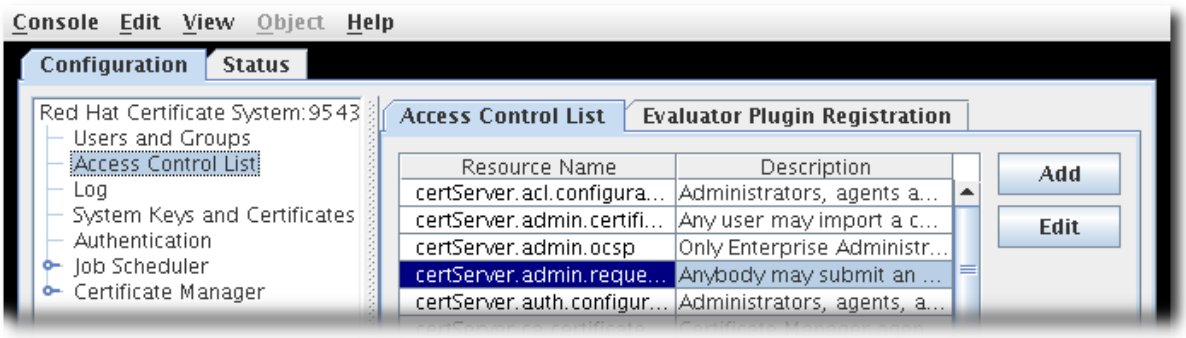
For instruction on how to configure this feature by editing the **CS.cfg** file, see the *Changing the Access Control Settings for the Subsystem* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

15.5.3. Adding ACLs

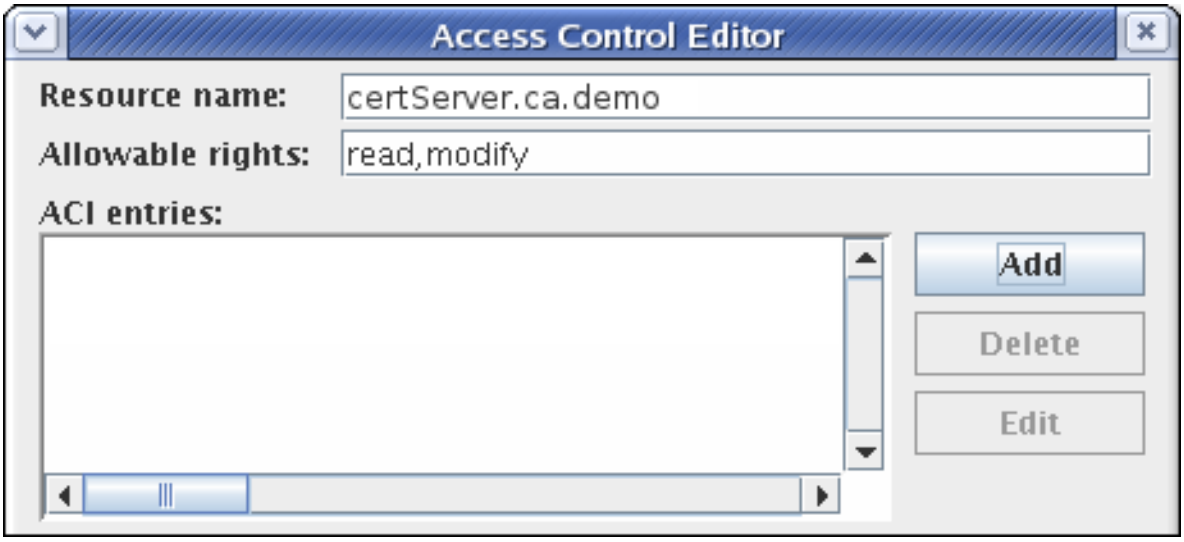
ACLs are stored in the internal database and can only be modified in the administrative console.

To add a new ACL:

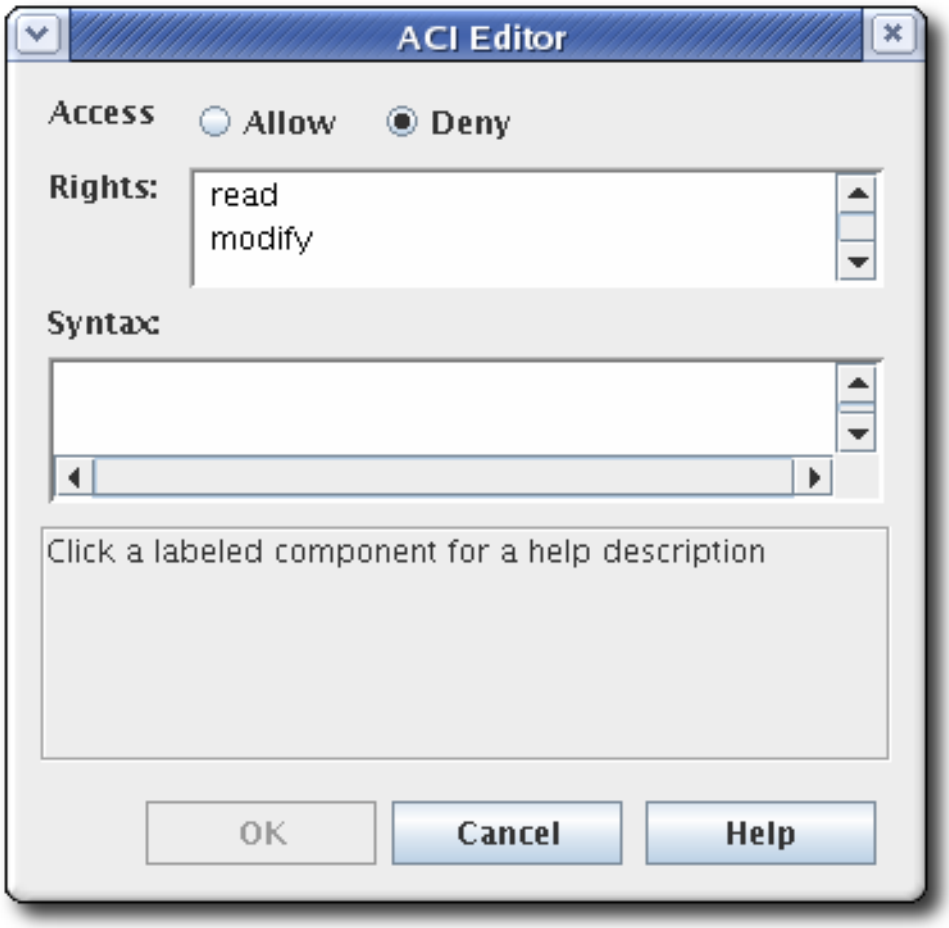
1. Log into the administrative console.
2. Select **Access Control List**.



3. Click **Add** to open the **Access Control Editor**.
4. Fill the **Resource name** and **Available rights** fields.



5. To add an access control instruction (ACI), click **Add**, and supply the ACI information.



a. Select the allow or deny radio button from the **Access** field to allow or deny the operation to the groups, users, or IP addresses specified. For more information about allowing or denying access, see Section 15.5.1, "About Access Control".

b. Set the rights. The available options are **read** and **modify**. To select both, hold the **Ctrl** or **Shift** button while selecting the entries.

c. Specify the user, group, or IP address that will be granted or denied access in the **Syntax** field. See Section 15.5.1, "About Access Control" for details on syntax.

6. Click **OK** to return to the **Access Control Editor** window.

7. Click **OK** to store the ACI.

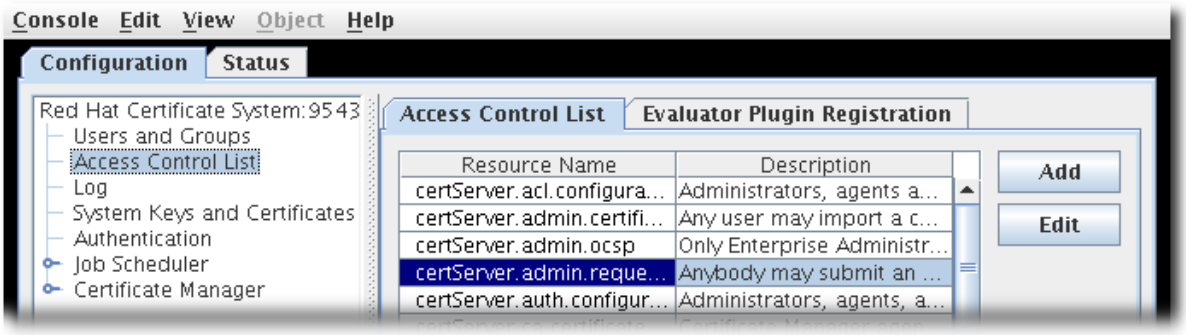
15.5.4. Editing ACLs

ACLs are stored in the internal database and can only be modified in the administrative console.

To edit the existing ACLs:

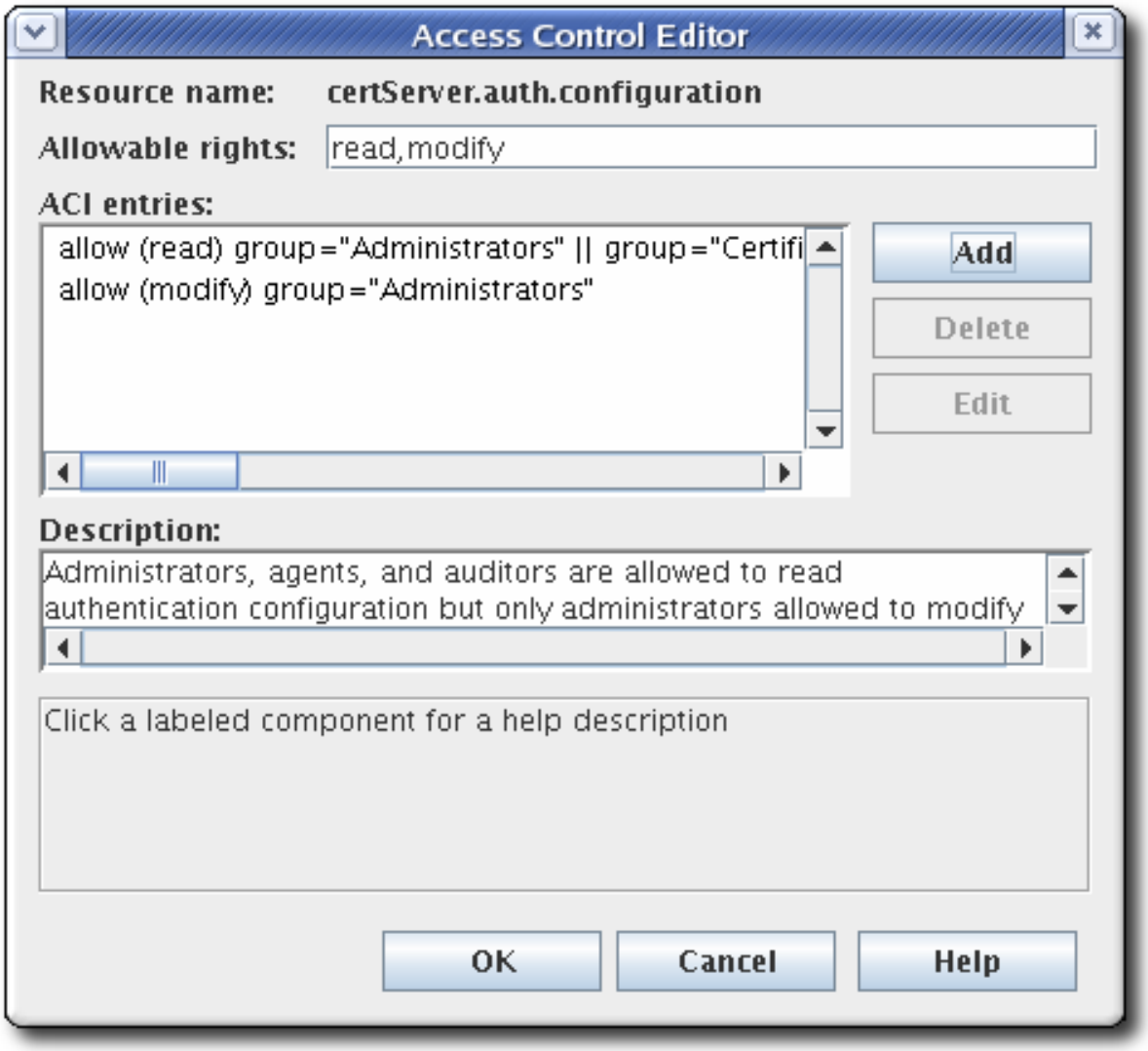
1. Log into the administrative console.

2. Select **Access Control List** in the left navigation menu.



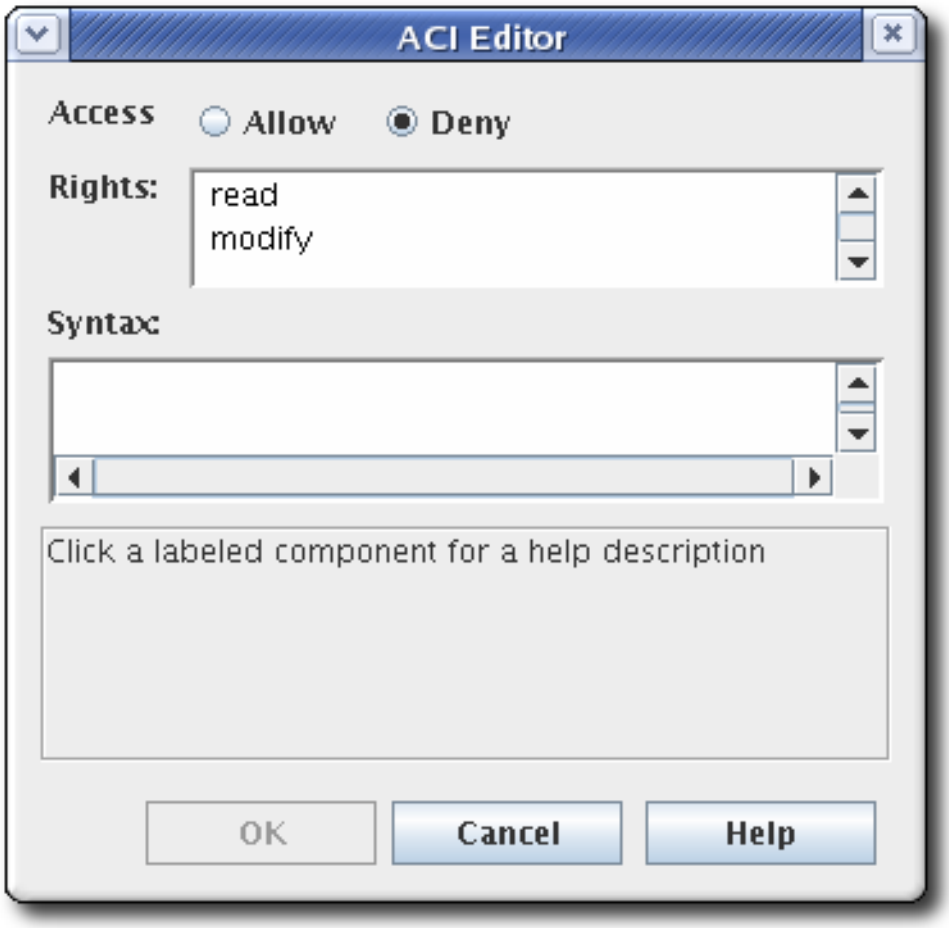
3. Select the ACL to edit from the list, and click **Edit**.

   The ACL opens in the **Access Control Editor** window.



4. To add an ACI, click **Add**, and supply the ACI information.

   To edit an ACI, select the ACI from the list in the **ACI entries** text area of the **ACL Editor** window. Click **Edit**.



1. Select the allow or deny radio button from the **Access** field to allow or deny the operation to the groups, users, or IP addresses specified. For more information about allowing or denying access, see Section 15.5.1, "About Access Control".

2. Set the rights for the access control. The options are **read** and **modify**. To set both, use the **Ctrl** or **Shift** buttons.

3. Specify the user, group, or IP address that will be granted or denied access in the **Syntax** field. See Section 15.5.1, "About Access Control" for details on syntax.

CHAPTER 16. CONFIGURING SUBSYSTEM LOGS

The Certificate System subsystems create log files that record events related to activities, such as administration, communications using any of the protocols the server supports, and various other processes employed by the subsystems. While a subsystem instance is running, it keeps a log of information and error messages on all the components it manages. Additionally, the Apache and Tomcat web servers generate error and access logs.

Each subsystem instance maintains its own log files for installation, audit, and other logged functions.

Log plug-in modules are listeners which are implemented as Java™ classes and are registered in the configuration framework.

All the log files and rotated log files, except for audit logs, are located in whatever directory was specified in **pki_subsystem_log_path** when the instance was created with **pkispawn**. Regular audit logs are located in the log directory with other types of logs, while signed audit logs are written to **/var/log/pki/*instance_name*/*subsystem_name*/signedAudit**. The default location for logs can be changed by modifying the configuration.

16.1. ABOUT CERTIFICATE SYSTEM LOGS

Certificate System subsystems keep several different kinds of logs, which provide specific information depending on the type of subsystem, types of services, and individual log settings. The kinds of logs that can be kept for an instance depend on the kind of subsystem that it is.

16.1.1. Signed Audit Logs

The Certificate System maintains audit logs for all events, such as requesting, issuing and revoking certificates and publishing CRLs. These logs are then signed. This allows authorized access or activity to be detected. An outside auditor can then audit the system if required. The assigned auditor user account is the only account which can view the signed audit logs. This user's certificate is used to sign and encrypt the logs. Audit logging is configured to specify the events that are logged.

Signed audit logs are written to **/var/log/pki/instance_name/subsystem_name/signedAudit**. However, the default location for logs can be changed by modifying the configuration.

For more information, see Section 16.3.2, “Using Signed Audit Logs” .

16.1.2. Debug Logs

Debug logs, which are enabled by default, are maintained for all subsystems, with varying degrees and types of information.

Debug logs contain very specific information for every operation performed by the subsystem, including plug-ins and servlets which are run, connection information, and server request and response messages.

The general types of services which are recorded to the debug log are briefly discussed in Section 16.2.1.1, “Services That Are Logged” . These services include authorization requests, processing certificate requests, certificate status checks, and archiving and recovering keys, and access to web services.

The debug logs for the CA, OCSP, KRA, and TKS record detailed information about the processes for the subsystem. Each log entry has the following format:

> [*date:time*] [*processor*]: *servlet*: *message*

-

The *message* can be a return message from the subsystem or contain values submitted to the subsystem.

For example, the TKS records this message for connecting to an LDAP server:

```
[10/Jun/2020:05:14:51][main]: Established LDAP connection using basic authentication to host
localhost port 389 as cn=Directory Manager
```

The *processor* is **main**, and the *message* is the message from the server about the LDAP connection, and there is no servlet.

The CA, on the other hand, records information about certificate operations as well as subsystem connections:

```
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestowner$
value=KRA-server.example.com-8443
```

In this case, the *processor* is the HTTP protocol over the CA's agent port, while it specifies the servlet for handling profiles and contains a *message* giving a profile parameter (the subsystem owner of a request) and its value (that the KRA initiated the request).

Example 16.1. CA Certificate Request Log Messages

```
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.profileapprovedby$ value=admin
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.cert_request$
value=MIIBozCCAZ8wggEFAgQqTfoHMIHHgAECpQ4wDDEKMAgGA1UEAxMBeKaBnzANBgkqhki
G9w0BAQEFAAOB...
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profile$
value=true
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.cert_request_type$ value=crmf
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.requestversion$ value=1.0.0
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_locale$
value=en
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.requestowner$ value=KRA-server.example.com-8443
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.dbstatus$
value=NOT_UPDATED
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.subject$
value=uid=jsmith, e=jsmith@example.com
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.requeststatus$ value=begin
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.user$ value=uid=KRA-server.example.com-
8443,ou=People,dc=example,dc=com
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_key$
value=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDreuEsBWq9WuZ2MaBwtNYxvkLP^
M
HcN0cusY7gxLzB+XwQ/VsWEoObGldg6WwJPOcBdvLiKKfC605wFdynbEgKs0fChV^M
k9HYDhmJ8hX6+PaquiHJSVNhsv5tOshZkCfMBbyxwrKd8yZ5G5I+2gE9PUznxJaM^M
HTmlOqm4HwFxzy0RRQIDAQAB
```

```
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.authmgrinstname$ value=raCertAuth
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.uid$ value=KRA-server.example.com-8443
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.userid$ value=KRA-server.example.com-8443
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.requestor_name$ value=
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profileid$
value=caUserCert
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.userdn$ value=uid=KRA-server.example.com-
4747,ou=People,dc=example,dc=com
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestid$
value=20
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.auth_token.authtime$ value=1212782378071
[06/Jun/2020:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.req_x509info$
value=MIICIKADAgECAgEAMA0GCSqGSIb3DQEBBQUAMEAxHjAcBgNVBAoTFVJlZGJ1ZGNv^M

bXB1dGVyIERvbWFpbjEeMBwGA1UEAxMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4X^M
DTA4MDYwNjE5NTkzOFoXDTA4MTIwMzE5NTkzOFowOzEhMB8GCSqGSIb3DQEJARYS^M
anNtaXRoQGV4YW1wbGUuY29tMRYwFAYKCZImiZPyLGQBARMGanNtaXRoMIGfMA0G^M
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDreuEsBWq9WuZ2MaBwtNYxvkLPHcN0cusY^M
7gxLzB+XwQ/VsWEoObGldg6WwJPOcBdvLiKKfC605wFdynbEgKs0fChVk9HYDhmJ^M
8hX6+PaquiHJSVNhsv5tOshZkCfMBbyxwrKd8yZ5G5I+2gE9PUznxJaMHTmlOqm4^M
HwFxzy0RRQIDAQABo4HFMIHCMB8GA1UdIwQYMBaAFG8gWeOJIMt+aO8VuQTMzPBU^M
78k8MEoGCCsGAQUFBwEBBD4wPDA6BggrBgEFBQcwAYYuaHR0cDovL3Rlc3Q0LnJl^M
ZGJ1ZGNvbXB1dGVyLmxvY2FsOjkwODAvY2Evb2NzcDAOBgNVHQ8BAf8EBAMCBeAw^M
HQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMEMCQGA1UdEQQdMBuBGSRyZXF1^
M
ZXN0LnJlcXVlc3Rvcl9lbWFpbCQ=
```

Likewise, the OCSP shows OCSP request information:

```
[07/Jul/2020:06:25:40][http-11180-Processor25]: OCSPServlet: OCSP Request:
[07/Jul/2020:06:25:40][http-11180-Processor25]: OCSPServlet:
MEUwQwIBADA+MDwwOjAJBgUrDgMCGgUABBSEWjCarLE6/BiSiENSsV9kHjqB3QQU
```

16.1.2.1. Installation Logs

All subsystems keep an install log.

Every time a subsystem is created either through the initial installation or creating additional instances with **pkispawn**, an installation file with the complete debug output from the installation, including any errors and, if the installation is successful, the URL and PIN to the configuration interface for the instance. The file is created in the **/var/log/pki/** directory for the instance with a name in the form **pki-*subsystem_name*-spawn.*timestamp*.log**.

Each line in the install log follows a step in the installation process.

Example 16.2. CA Install Log

```
```

```
...
2015-07-22 20:43:13 pkispawn    : INFO     ... finalizing
'pki.server.deployment.scriptlets.finalization'
2015-07-22 20:43:13 pkispawn    : INFO     ..... cp -p /etc/sysconfig/pki/tomcat/pki-
tomcat/ca/deployment.cfg /var/log/pki/pki-
tomcat/ca/archive/spawn_deployment.cfg.20150722204136
2015-07-22 20:43:13 pkispawn    : DEBUG    ..... chmod 660 /var/log/pki/pki-
tomcat/ca/archive/spawn_deployment.cfg.20150722204136
2015-07-22 20:43:13 pkispawn    : DEBUG    ..... chown 26445:26445 /var/log/pki/pki-
tomcat/ca/archive/spawn_deployment.cfg.20150722204136
2015-07-22 20:43:13 pkispawn    : INFO     ..... generating manifest file called
'/etc/sysconfig/pki/tomcat/pki-tomcat/ca/manifest'
2015-07-22 20:43:13 pkispawn    : INFO     ..... cp -p /etc/sysconfig/pki/tomcat/pki-
tomcat/ca/manifest /var/log/pki/pki-tomcat/ca/archive/spawn_manifest.20150722204136
2015-07-22 20:43:13 pkispawn    : DEBUG    ..... chmod 660 /var/log/pki/pki-
tomcat/ca/archive/spawn_manifest.20150722204136
2015-07-22 20:43:13 pkispawn    : DEBUG    ..... chown 26445:26445 /var/log/pki/pki-
tomcat/ca/archive/spawn_manifest.20150722204136
2015-07-22 20:43:13 pkispawn    : INFO     ..... executing 'systemctl enable pki-tomcatd.target'
2015-07-22 20:43:13 pkispawn    : INFO     ..... executing 'systemctl daemon-reload'
2015-07-22 20:43:13 pkispawn    : INFO     ..... executing 'systemctl restart pki-tomcatd@pki-
tomcat.service'
2015-07-22 20:43:14 pkispawn    : INFO     END spawning subsystem 'CA' of instance 'pki-tomcat'
2015-07-22 20:43:14 pkispawn    : DEBUG
```

16.1.2.2. Tomcat Error and Access Logs

The CA, KRA, OCSP, TKS, and TPS subsystems use a Tomcat web server instance for their agent and end-entities' interfaces.

Error and access logs are created by the Tomcat web server, which are installed with the Certificate System and provide HTTP services. The error log contains the HTTP error messages the server has encountered. The access log lists access activity through the HTTP interface.

Logs created by Tomcat:

- admin.*timestamp*
- catalina.*timestamp*
- catalina.out
- host-manager.*timestamp*
- localhost.*timestamp*
- localhost_access_log.*timestamp*
- manager.*timestamp*

These logs are not available or configurable within the Certificate System; they are only configurable within Apache or Tomcat. See the Apache documentation for information about configuring these logs.

16.1.2.3. Self-Tests Log

The self-tests log records information obtained during the self-tests run when the server starts or when the self-tests are manually run. The tests can be viewed by opening this log. This log is not configurable through the Console, it can only be configured by changing settings in the **CS.cfg** file. For instruction on how to configure logs by editing the **CS.cfg** file, see the *Enabling the Publishing Queue* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

The information about logs in this section does not pertain to this log. See Section 14.9, "Running Self-Tests" for more information about self-tests.

16.2. MANAGING LOGS

The Certificate System subsystem log files record events related to operations within that specific subsystem instance. For each subsystem, different logs are kept for issues such as installation, access, and web servers.

All subsystems have similar log configuration, options, and administrative paths.

16.2.1. An Overview of Log Settings

The way that logs are configured can affect Certificate System performance. For example, log file rotation keeps logs from becoming too large, which slows down subsystem performance. This section explains the different kinds of logs recorded by Certificate System subsystems and covers important concepts such as log file rotation, buffered logging, and available log levels.

16.2.1.1. Services That Are Logged

All major components and protocols of Certificate System log messages to log files. Table 16.1, "Services Logged" lists services that are logged by default. To view messages logged by a specific service, customize log settings accordingly. For details, see Section 16.3.1, "Viewing Logs in the Console" .

Table 16.1. Services Logged

Service	Description
ACLs	Logs events related to access control lists.
Administration	Logs events related to administration activities, such as HTTPS communication between the Console and the instance.
All	Logs events related to all the services.
Authentication	Logs events related to activity with the authentication module.
Certificate Authority	Logs events related to the Certificate Manager.
Database	Logs events related to activity with the internal database.
HTTP	Logs events related to the HTTP activity of the server. Note that HTTP events are actually logged to the errors log belonging to the Apache server incorporated with the Certificate System to provide HTTP services.

Service	Description
Key Recovery Authority	Logs events related to the KRA.
LDAP	Logs events related to activity with the LDAP directory, which is used for publishing certificates and CRLs.
OCSP	Logs events related to OCSP, such as OCSP status GET requests.
Others	Logs events related to other activities, such as command-line utilities and other processes.
Request Queue	Logs events related to the request queue activity.
User and Group	Logs events related to users and groups of the instance.

16.2.1.2. Log Levels (Message Categories)

The different events logged by Certificate System services are determined by the log levels, which makes identifying and filtering events simpler. The different Certificate System log levels are listed in Table 16.2, "Log Levels and Corresponding Log Messages" .

Log levels are represented by numbers indicating how detailed the level of logging to be performed by the server should be.

A higher priority level means less detail because only events of high priority are logged.

Table 16.2. Log Levels and Corresponding Log Messages

Log level	Message category	Description
0-1	Tracing	These messages contain finer-grained debugging information. This level should not be used regularly because it may impact the performance.
2-5	Debugging	These messages contain debugging information. This level is not recommended for regular use because it generates too much information.
6-10	Informational	These messages provide general information about the state of the Certificate System, including status messages such as *Certificate System initialization complete* and *Request for operation succeeded*.
11-15	Warning	These messages are warnings only and do not indicate any failure in the normal operation of the server.

Log level	Message category	Description
> 15	Failure	These messages indicate errors and failures that prevent the server from operating normally, including failures to perform a certificate service operation (*User authentication failed* or *Certificate revoked*) and unexpected situations that can cause irrevocable errors (*The server cannot send back the request it processed for a client through the same channel the request came from the client*). Setting the level above 15 will minimize the logs, as only failures will be recorded.

Log levels can be used to filter log entries based on the severity of an event. The default log level is 10.

Log data can be extensive, especially at lower (more verbose) logging levels. Make sure that the host machine has sufficient disk space for all the log files. It is also important to define the logging level, log rotation, and server-backup policies appropriately so that all the log files are backed up and the host system does not get overloaded; otherwise, information can be lost.

16.2.1.3. Buffered and Unbuffered Logging

The Java subsystems support buffered logging for all types of logs. The server can be configured for either buffered or unbuffered logging.

If buffered logging is configured, the server creates buffers for the corresponding logs and holds the messages in the buffers for as long as possible. The server flushes out the messages to the log files only when one of the following conditions occurs:

- The buffer gets full. The buffer is full when the buffer size is equal to or greater than the value specified by the **bufferSize** configuration parameter. The default value for this parameter is 512 KB.
- The flush interval for the buffer is reached. The flush interval is reached when the time interval since the last buffer flush is equal to or greater than the value specified by the **flushInterval** configuration parameter. The default value for this parameter is 5 seconds.
- When current logs are read from Console. The server retrieves the latest log when it is queried for current logs.

If the server is configured for unbuffered logging, the server flushes out messages as they are generated to the log files. Because the server performs an I/O operation (writing to the log file) each time a message is generated, configuring the server for unbuffered logging decreases performance.

Setting log parameters is described in Section 16.2.2, “Configuring Logs in the Console” .

16.2.1.4. Log File Rotation

The subsystem logs have an optional log setting that allows them to be rotated and start a new log file instead of letting log files grow indefinitely. Log files are rotated when either of the following occur:

- The size limit for the corresponding file is reached. The size of the corresponding log file is equal to or greater than the value specified by the **maxFileSize** configuration parameter. The default value for this parameter is 100 KB.

- The age limit for the corresponding file is reached. The corresponding log file is equal to or older than the interval specified by the **rolloverInterval** configuration parameter. The default value for this parameter is 2592000 seconds (every thirty days).

NOTE

Setting both these parameters to 0 effectively disables the log file rotation.

When a log file is rotated, the old file is named using the name of the file with an appended time stamp. The appended time stamp is an integer that indicates the date and time the corresponding active log file was rotated. The date and time have the forms YYYYMMDD (year, month, day) and HHMMSS (hour, minute, second).

Log files, especially the audit log file, contain critical information. These files should be periodically archived to some backup medium by copying the entire **log** directory to an archive medium.

NOTE

The Certificate System does not provide any tool or utility for archiving log files.

The Certificate System provides a command-line utility, **signtool**, that signs log files before archiving them as a means of tamper detection. For details, see Section 16.2.4.5, "Signing Log Files".

Signing log files is an alternative to the signed audit logs feature. Signed audit logs create audit logs that are automatically signed with a subsystem signing certificate. See Section 16.2.4.3, "Configuring a Signed Audit Log in the Console" for details about signed audit logs.

Rotated log files are not deleted.

16.2.2. Configuring Logs in the Console

Logs can be configured through both the subsystem Console and through the subsystem's **CS.cfg** file. Specialized logs, such as signed audit logs and custom logs, can also be created through the Console or configuration file.

Audit logs can be configured through the subsystem Console for the CA, OCSP, TKS, and KRA subsystems. TPS logs are only configured through the configuration file.

1. In the navigation tree of the **Configuration** tab, select **Log**.
2. The **Log Event Listener Management** tab lists the currently configured listeners.

   To create a new log instance, click **Add**, and select a module plug-in from the list in the **Select Log Event Listener Plug-in Implementation** window.
3. Set or modify the fields in the **Log Event Listener Editor** window. The different parameters are listed in Table 16.3, "Log Event Listener Fields".

Table 16.3. Log Event Listener Fields

Field	Description

Field	Description
Log Event Listener ID	Gives the unique name that identifies the listener. The names can have any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-), but it cannot contain other characters or spaces.
type	Gives the type of log file. **transaction** records audit logs.
enabled	Sets whether the log is active. Only enabled logs actually record events. The value is either **true** or **false**.
level	Sets the log level in the text field. The level must be manually entered in the field; there is no selection menu. The choices are **Debug**, **Information**, **Warning**, **Failure**, **Misconfiguration**, **Catastrophe**, and **Security**. For more information, see Section 16.2.1.2, "Log Levels (Message Categories)".
fileName	Gives the full path, including the file name, to the log file. The subsystem user should have read/write permission to the file.
bufferSize	Sets the buffer size in kilobytes (KB) for the log. Once the buffer reaches this size, the contents of the buffer are flushed out and copied to the log file. The default size is 512 KB. For more information on buffered logging, see Section 16.2.1.3, "Buffered and Unbuffered Logging".
flushInterval	Sets the amount of time before the contents of the buffer are flushed out and added to the log file. The default interval is 5 seconds.
maxFileSize	Sets the size, in kilobytes (KB), a log file can become before it is rotated. Once it reaches this size, the file is copied to a rotated file, and the log file is started new. For more information on log file rotation, see Section 16.2.1.4, "Log File Rotation". The default size is 2000 KB.
rolloverInterval	Sets the frequency for the server to rotate the active log file. The available options are hourly, daily, weekly, monthly, and yearly. The default is monthly. For more information, see Section 16.2.1.4, "Log File Rotation".

16.2.3. Configuring Logs in the CS.cfg File

For instruction on how to configure logs by editing the **CS.cfg** file, see the *Configuring Logs in the CS.cfg File* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

16.2.4. Managing Audit Logs

The audit log contains records for events that have been set up as recordable events. If the **logSigning** attribute is set to **true**, the audit log is signed with a log signing certificate belonging to the server. This certificate can be used by auditors to verify that the log has not been tampered with.

By default, regular audit logs are located in the **/var/log/pki/*instance_name*/*subsystem_name*/** directory with other types of logs, while signed audit logs are written to **/var/log/pki/*instance_name*/*subsystem_name*/signedAudit/**. The default location for logs can be changed by modifying the configuration.

The signed audit log creates a log recording system events, and the events are selected from a list of potential events. When enabled, signed audit logs record a verbose set of messages about the selected event activity.

Signed audit logs are configured by default when the instance is first created, but it is possible to configure signed audits logs after installation. (See Section 16.2.4.2, "Enabling Signed Audit Logging after Installation".) It is also possible to edit the configuration or change the signing certificates after configuration, as covered in Section 16.2.4.3, "Configuring a Signed Audit Log in the Console" .

16.2.4.1. A List of Audit Events

For a list of audit events in Certificate System, see Appendix E, *Audit Events*.

16.2.4.2. Enabling Signed Audit Logging after Installation

Signed audit logs can be enabled by default when an instance is first created by using the **pki_audit_group** deployment parameter with the **pkispawn** command. If, however, signed audit logs were not configured when an instance was created, they can be enabled afterwards by reassigning ownership of the audit log directory to the auditor system users group, such as **pkiaudit**.

1. Stop the instance:

```
# pki-server stop instance_name
```

2. Set the group ownership of the signed audit log directory to the PKI auditors operating system group, such as **pkiaudit**. This allows the users in the PKI auditors group to have the required read access to the **signedAudit** directory to verify the signatures on the log files. No user (except for the Certificate System user account, **pkiuser**) should have write access to the log files in this directory.

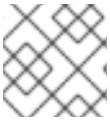
```
chgrp -R pkiaudit /var/log/pki/instance_name/subsystem_name/signedAudit
```

3. Restart the instance:

```
# pki-server start instance_name
```

16.2.4.3. Configuring a Signed Audit Log in the Console

Signed audit logs are configured by default when the instance is first created, but it is possible to edit the configuration or change the signing certificates after configuration.



NOTE

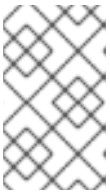
Provide enough space in the file system for the signed audit logs, since they can be large.

A log is set to a signed audit log by setting the **logSigning** parameter to **enable** and providing the nickname of the certificate used to sign the log. A special log signing certificate is created when the subsystems are first configured.

Only a user with auditor privileges can access and view a signed audit log. Auditors can use the **AuditVerify** tool to verify that signed audit logs have not been tampered with.

The signed audit log is created and enabled when the subsystem is configured, but it needs additional configuration to begin creating and signing audit logs.

1. Open the Console.



NOTE

To create or configure the audit log by editing the **CS.cfg** file, see the *Configuring Logs in the CS.cfg File* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

2. In the navigation tree of the **Configuration** tab, select **Log**.
3. In the **Log Event Listener Management** tab, select the **SignedAudit** entry.
4. Click **Edit/View**.
5. There are three fields which must be reset in the **Log Event Listener Editor** window.
    - Fill in the **signedAuditCertNickname**. This is the nickname of the certificate used to sign audit logs. An audit signing certificate is created when the subsystem is configured; it has a nickname like **auditSigningCert cert-*instance_name subsystem_name***.

NOTE

To get the audit signing certificate nickname, list the certificates in the subsystem's certificate database using **certutil**. For example:

```
certutil -L -d /var/lib/pki-tomcat/alias

Certificate Authority - Example Domain    CT,c,
subsystemCert cert-pki-tomcat             u,u,u
Server-Cert cert-pki-tomcat               u,u,u
auditSigningCert cert-pki-tomcat CA       u,u,Pu
```

    - Set the **logSigning** field to **true** to enable signed logging.
    - Set any **events** which are logged to the audit log. Appendix E, *Audit Events* lists the loggable events. Log events are separated by commas with no spaces.
6. Set any other settings for the log, such as the file name, the log level, the file size, or the rotation schedule.

NOTE

By default, regular audit logs are located in the **/var/log/pki/*instance_name*/*subsystem_name*/** directory with other types of logs, while signed audit logs are written to **/var/log/pki/*instance_name*/*subsystem_name*/signedAudit/**. The default location for logs can be changed by modifying the configuration.

7. Save the log configuration.

After enabling signed audit logging, assign auditor users by creating the user and assigning that entry to the auditor group. Members of the auditor group are the only users who can view and verify the signed audit log. See Section 15.3.2.1, "Creating Users" for details about setting up auditors.

Auditors can verify logs by using the **AuditVerify** tool. See the **AuditVerify(1)** man page for details about using this tool.

16.2.4.4. Handling Audit Logging Failures

There are events that could cause the audit logging function to fail, so events cannot be written to the log. For example, audit logging can fail when the file system containing the audit log file is full or when the file permissions for the log file are accidentally changed. If audit logging fails, the Certificate System instance shuts down in the following manner.

- Servlets are disabled and will not process new requests.
- All pending and new requests are killed.
- The subsystem is shut down.

When this happens, administrators and auditors should work together with the operating system administrator to resolve the disk space or file permission issues. When the IT problem is resolved, the auditor should make sure that the last audit log entries are signed. If not, they should be preserved by manual signing (Section 16.2.4.5, "Signing Log Files"), archived, and removed to prevent audit verification failures in the future. When this is completed, the administrators can restart the Certificate System.

16.2.4.5. Signing Log Files

The Certificate System can digitally sign log files before they are archived or distributed for audit purposes. This feature allows files to be checked for tampering.

This is an alternative to the signed audit logs feature. The signed audit log feature creates audit logs that are automatically signed; this tool manually signs archived logs. See Section 16.2.4.3, "Configuring a Signed Audit Log in the Console" for details about signed audit logs.

For signing log files, use a command-line utility called the Signing Tool (**signtool**). For details about this utility, see http://www.mozilla.org/projects/security/pki/nss/tools/.

The utility uses information in the certificate, key, and security module databases of the subsystem instance.

As a user with auditor privilegesuse the **signtool** command to sign the log directories:

```
signtool -d secdb_dir -k cert_nickname -Z output input
```

- *secdb_dir* specifies the path to the directory that contains the certificate, key, and security module databases for the CA.
- *cert_nickname* specifies the nickname of the certificate to use for signing.
- *output* specifies the name of the JAR file (a signed zip file).
- *input* specifies the path to the directory that contains the log files.

16.2.4.6. Filtering Audit Events

In Certificate System administrators can set filters to configure which audit events will be logged in the audit file based on the event attributes.

The format of the filters is the same as for LDAP filters. However, Certificate System only supports the following filters:

Table 16.4. Supported Audit Event Filters

Type	Format	Example
Presence	(*attribute*=*)	(ReqID=*)
Equality	(*attribute=value*)	(Outcome=Failure)
Substring	(*attribute=initial*any*...*any*final*)	(SubjectID=*admin*)
AND operation	(&(*filter_1*)(*filter_2*)...(*filter_n*))	(&(SubjectID=admin)(Outcome=Failure))
OR operation	(|(*filter_1*)(*filter_2*)...(*filter_n*))	(|(SubjectID=admin)(Outcome=Failure))
NOT operation	(!(*filter*))	(!(SubjectID=admin))

For further details on LDAP filters, see the *Using Compound Search Filters* in the *Red Hat Directory Server Administration Guide*.

Example 16.3. Filtering Audit Events

To log only failed events for profile certificate requests and events for processed certificates requests that have the ***InfoName*** field set to **rejectReadon** or **cancelReason**:

1. Edit the **/var/lib/pki/*instance_name*/*subsystem_type*/conf/CS.cfg** file and set the following parameters:

```
log.instance.SignedAudit.filters.PROFILE_CERT_REQUEST=(Outcome=Failure)
log.instance.SignedAudit.filters.CERT_REQUEST_PROCESSED=(|
(InfoName=rejectReason)(InfoName=cancelReason))
```

2. Restart Certificate System:

```
# pki-server restart instance_name
```

16.2.5. Managing Log Modules

The types of logs that are allowed and their behaviors are configured through *log module* plug-ins. New logging modules can be created and used to make custom logs.

New log plug-in modules can be registered through the Console. Registering a new module involves specifying the name of the module and the full name of the Java™ class that implements the log interface.

Before registering a plug-in module, put the Java™ class for the module in the **classes** directory; the implementation must be on the class path.

To register a log plug-in module with a subsystem instance:

1. Create the custom job class. For this example, the custom log plug-in is called **MyLog.java**.
2. Compile the new class into the lib directory of the instance.

```
javac -d . /var/lib/pki/pki-tomcat/lib -classpath $CLASSPATH MyLog.java
```

3. Create a directory in the CA's **WEB-INF** web directory to hold the custom classes, so that the CA can access them.

```
mkdir /var/lib/pki/pki-tomcat/webapps/ca/WEB-INF/classes
```

4. Set the owner to the Certificate System system user (**pkiuser**).

```
chown -R pkiuser:pkiuser /var/lib/pki/pki-tomcat/lib
```

5. Register the plug-in.
    1. Log into the Console.
    2. In the **Configuration** tab, select **Logs** from the navigation tree. Then select the **Log Event Listener Plug-in Registration** tab.
    3. Click **Register**.

       The **Register Log Event Listener Plug-in Implementation** window appears.
    4. Give the name for the plug-in module and the Java™ class name.

       The Java™ class name is the full path to the implementing Java™ class. If this class is part of a package, include the package name. For example, registering a class named **customLog** in a package named **com.customplugins**, the class name would be **com.customplugins.customLog**.
    5. Click **OK**.

Unwanted log plug-in modules can be deleted through the Console. Before deleting a module, delete all the listeners based on this module; see Section 16.2.1.4, “Log File Rotation” .

16.3. USING LOGS

16.3.1. Viewing Logs in the Console

To troubleshoot the subsystem, check the error or informational messages that the server has logged. Examining the log files can also monitor many aspects of the server's operation. Some log files can be viewed through the Console. However, the audit log is only accessible by users with the Auditor role,

using a method detailed in Section 16.3.2, “Using Signed Audit Logs” .

To view the contents of a log file:

1. Log into the Console.
2. Select the **Status** tab.
3. Under **Logs**, select the log to view.
4. Set the viewing preferences in the **Display Options** section.
    - **Entries** – The maximum number of entries to be displayed. When this limit is reached, the Certificate System returns any entries that match the search request. Zero (0) means no messages are returned. If the field is blank, the server returns every matching entry, regardless of the number found.
    - **Source** – Select the Certificate System component or service for which log messages are to be displayed. Choosing **All** means messages logged by all components that log to this file are displayed.
    - **Level** – Select a message category that represents the log level for filtering messages.
    - **Filename** – Select the log file to view.
5. Click **Refresh**.
6. To view a full entry, double-click it, or select the entry, and click **View**.

16.3.2. Using Signed Audit Logs

This section explains how a user in the Auditor group displays and verifies signed audit logs.

16.3.2.1. Listing Audit Logs

As a user with auditor privileges, use the the **pki *subsystem*-audit-file-find** command to list existing audit log files on the server.

For example, to list the audit log files on the CA hosted on **server.example.com**:

```
# pki -h server.example.com -p 8443 -n auditor ca-audit-file-find
-----------------
3 entries matched
-----------------
 File name: ca_audit.20170331225716
 Size: 2883

 File name: ca_audit.20170401001030
 Size: 189

 File name: ca_audit
 Size: 6705
----------------------------
Number of entries returned 3
----------------------------
```

The command uses the client certificate with the *auditor* nickname stored in the **~/.dogtag/nssdb/** directory for authenticating to the CA. For further details about the parameters used in the command and alternative authentication methods, see the pki(1) man page.

16.3.2.2. Downloading Audit Logs

As a user with auditor privileges, use the **pki *subsystem*-audit-file-retrieve** command to download a specific audit log from the server.

For example, to download an audit log file from the CA hosted on **server.example.com**:

1. Optionally, list the available log files on the CA. See Section 16.3.2.1, "Listing Audit Logs".
2. Download the log file. For example, to download the **ca_audit** file:

   ```
   # pki -U https://server.example.com:8443 -n auditor ca-audit-file-retrieve ca_audit
   ```

   The command uses the client certificate with the *auditor* nickname stored in the **~/.dogtag/nssdb/** directory for authenticating to the CA. For further details about the parameters used in the command and alternative authentication methods, see the pki(1) man page.

After downloading a log file, you can search for specific log entries, for example, using the **grep** utility:

```
# grep "\[AuditEvent=ACCESS_SESSION_ESTABLISH\]" log_file
```

16.3.2.3. Verifying Signed Audit Logs

If audit log signing is enabled, users with auditor privileges can verify the logs:

1. Initialize the NSS database and import the CA certificate. For details, see Section 2.5.1.1, "pki CLI Initialization" and the *Importing a certificate into an NSS Database* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.
2. If the audit signing certificate does not exist in the PKI client database, import it:
   a. Search the audit signing certificate for the subsystem logs you want to verify. For example:

   ```
   # pki ca-cert-find --name "CA Audit Signing Certificate"
   ---------------
   1 entries found
   ---------------
    Serial Number: 0x5
    Subject DN: CN=CA Audit Signing Certificate,O=EXAMPLE
    Status: VALID
    Type: X.509 version 3
    Key Algorithm: PKCS #1 RSA with 2048-bit key
    Not Valid Before: Fri Jul 08 03:56:08 CEST 2016
    Not Valid After: Thu Jun 28 03:56:08 CEST 2018
    Issued On: Fri Jul 08 03:56:08 CEST 2016
    Issued By: system
   ----------------------------
   Number of entries returned 1
   ----------------------------
   ```

b. Import the audit signing certificate into the PKI client:

```
# pki client-cert-import "CA Audit Signing Certificate" --serial 0x5 --trust ",,P"
----------------------------------------------------
Imported certificate "CA Audit Signing Certificate"
----------------------------------------------------
```

3. Download the audit logs. See Section 16.3.2.2, "Downloading Audit Logs".

4. Verify the audit logs.

a. Create a text file that contains a list of the audit log files you want to verify in chronological order. For example:

```
# cat > ~/audit.txt << EOF
ca_audit.20170331225716
ca_audit.20170401001030
ca_audit
EOF
```

b. Use the **AuditVerify** utility to verify the signatures. For example:

```
# AuditVerify -d ~/.dogtag/nssdb/ -n "CA Audit Signing Certificate" \
  -a ~/audit.txt
Verification process complete.
Valid signatures: 10
Invalid signatures: 0
```

For further details about using **AuditVerify**, see the AuditVerify(1) man page.

16.3.3. Displaying Operating System-level Audit Logs

NOTE

To see Operating System-level audit logs using the instructions below, the **auditd** logging framework must be configured per the *Enabling OS-level Audit Logs* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

To display operating system-level access logs, use the **ausearch** utility as root or as a privileged user with the **sudo** utility.

16.3.3.1. Displaying Audit Log Deletion Events

Since these events are keyed (with **rhcs_audit_deletion**), use the ***-k*** parameter to find events matching that key:

```
# ausearch -k rhcs_audit_deletion
```

16.3.3.2. Displaying Access to the NSS Database for Secret and Private Keys

Since these events are keyed (with **rhcs_audit_nssdb**), use the ***-k*** parameter to find events matching that key:

```
# ausearch -k rhcs_audit_nssdb
```

16.3.3.3. Displaying Time Change Events

Since these events are keyed (with **rhcs_audit_time_change**), use the ***-k*** parameter to find events matching that key:

```
# ausearch -k rhcs_audit_time_change
```

16.3.3.4. Displaying Package Update Events

Since these events are a typed message (of type **SOFTWARE_UPDATE**), use the ***-m*** parameter to find events matching that type:

```
# ausearch -m SOFTWARE_UPDATE
```

16.3.3.5. Displaying Changes to the PKI Configuration

Since these events are keyed (with **rhcs_audit_config**), use the ***-k*** parameter to find events matching that key:

```
# ausearch -k rhcs_audit_config
```

16.3.4. Smart Card Error Codes

Smart cards can report certain error codes to the TPS; these are recorded in the TPS's debug log file, depending on the cause for the message.

Table 16.5. Smart Card Error Codes

Return Code	Description
General Error Codes	
6400	No specific diagnosis
6700	Wrong length in Lc
6982	Security status not satisfied
6985	Conditions of use not satisfied
6a86	Incorrect P1 P2
6d00	Invalid instruction
6e00	Invalid class

Return Code	Description
Install Load Errors	
6581	Memory Failure
6a80	Incorrect parameters in data field
6a84	Not enough memory space
6a88	Referenced data not found
Delete Errors	
6200	Application has been logically deleted
6581	Memory failure
6985	Referenced data cannot be deleted
6a88	Referenced data not found
6a82	Application not found
6a80	Incorrect values in command data
Get Data Errors	
6a88	Referenced data not found
Get Status Errors	
6310	More data available
6a88	Referenced data not found
6a80	Incorrect values in command data
Load Errors	
6581	Memory failure
6a84	Not enough memory space
6a86	Incorrect P1/P2
6985	Conditions of use not satisfied

CHAPTER 17. MANAGING SUBSYSTEM CERTIFICATES

This chapter gives an overview of using certificates: what types and formats are available, how to request and create them through the HTML end-entity forms and through the Certificate System Console, and how to install certificates in the Certificate System and on different clients. Additionally, there is information on managing certificates through the Console and configuring the servers to use them.

17.1. REQUIRED SUBSYSTEM CERTIFICATES

Each subsystem has a defined set of certificates which must be issued to the subsystem instance for it to perform its operations. There are certain details of the certificate contents that are set during the Certificate Manager configuration, with different considerations for constraints, settings, and attributes depending on the types of certificates; planning the formats of certificates is covered in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

17.1.1. Certificate Manager Certificates

When a Certificate Manager is installed, the keys and requests for the CA signing certificate, SSL server certificate, and OCSP signing certificate are generated. The certificates are created before the configuration can be completed.

The CA certificate request is either submitted as a self-signing request to the CA, which then issues the certificate and finishes creating the self-signed root CA, or is submitted to a third-party public CA or another Certificate System CA. When the external CA returns the certificate, the certificate is installed, and installation of the subordinate CA is completed.

- Section 17.1.1.1, “CA Signing Key Pair and Certificate”
- Section 17.1.1.2, “OCSP Signing Key Pair and Certificate”
- Section 17.1.1.3, “Subsystem Certificate”
- Section 17.1.1.4, “SSL Server Key Pair and Certificate”
- Section 17.1.1.5, “Audit Log Signing Key Pair and Certificate”

17.1.1.1. CA Signing Key Pair and Certificate

Every Certificate Manager has a CA signing certificate with a public key corresponding to the private key the Certificate Manager uses to sign the certificates and CRLs it issues. This certificate is created and installed when the Certificate Manager is installed. The default nickname for the certificate is **caSigningCert cert-***instance_ID CA*, where *instance_ID* identifies the Certificate Manager instance. The default validity period for the certificate is five years.

The subject name of the CA signing certificate reflects the name of the CA that was set during installation. All certificates signed or issued by the Certificate Manager include this name to identify the issuer of the certificate.

The Certificate Manager's status as a root or subordinate CA is determined by whether its CA signing certificate is self-signed or is signed by another CA, which affects the subject name on the certificates.

- If the Certificate Manager is a root CA, its CA signing certificate is self-signed, meaning the subject name and issuer name of the certificate are the same.

- If the Certificate Manager is a subordinate CA, its CA signing certificate is signed by another CA, usually the one that is a level above in the CA hierarchy (which may or may not be a root CA). The root CA's signing certificate must be imported into individual clients and servers before the Certificate Manager can be used to issue certificates to them.

NOTE

The CA name *cannot* be changed or all previously-issued certificates are invalidated. Similarly, reissuing a CA signing certificate with a new key pair invalidates all certificates that were signed by the old key pair.

17.1.1.2. OCSP Signing Key Pair and Certificate

The subject name of the OCSP signing certificate is in the form **cn=OCSP cert-***instance_ID CA*, and it contains extensions, such as **OCSPSigning** and **OCSPNoCheck**, required for signing OCSP responses.

The default nickname for the OCSP signing certificate is **ocspSigningCert cert-***instance_ID*, where *instance_ID CA* identifies the Certificate Manager instance.

The OCSP private key, corresponding to the OCSP signing certificate's public key, is used by the Certificate Manager to sign the OCSP responses to the OCSP-compliant clients when queried about certificate revocation status.

17.1.1.3. Subsystem Certificate

Every member of the security domain is issued a server certificate to use for communications among other domain members, which is separate from the server SSL certificate. This certificate is signed by the security domain CA; for the security domain CA itself, its subsystem certificate is signed by itself.

The default nickname for the certificate is **subsystemCert cert-***instance_ID*.

17.1.1.4. SSL Server Key Pair and Certificate

Every Certificate Manager has at least one SSL server certificate that was first generated when the Certificate Manager was installed. The default nickname for the certificate is **Server-Cert cert-***instance_ID*, where *instance_ID* identifies the Certificate Manager instance.

By default, the Certificate Manager uses a single SSL server certificate for authentication. However, additional server certificates can be requested to use for different operations, such as configuring the Certificate Manager to use separate server certificates for authenticating to the end-entity services interface and agent services interface.

If the Certificate Manager is configured for SSL-enabled communication with a publishing directory, it uses its SSL server certificate for client authentication to the publishing directory by default. The Certificate Manager can also be configured to use a different certificate for SSL client authentication.

17.1.1.5. Audit Log Signing Key Pair and Certificate

The CA keeps a secure audit log of all events which occurred on the server. To guarantee that the audit log has not been tampered with, the log file is signed by a special log signing certificate.

The audit log signing certificate is issued when the server is first configured.

NOTE

While other certificates can use ECC keys, the audit signing certificate must *always* use an RSA key.

17.1.2. Online Certificate Status Manager Certificates

When the Online Certificate Status Manager is first configured, the keys for all required certificates are created, and the certificate requests for the OCSP signing, SSL server, audit log signing, and subsystem certificates are made. These certificate requests are submitted to a CA (either a Certificate System CA or a third-party CA) and must be installed in the Online Certificate Status Manager database to complete the configuration process.

- Section 17.1.2.2, "SSL Server Key Pair and Certificate"
- Section 17.1.2.3, "Subsystem Certificate"
- Section 17.1.2.4, "Audit Log Signing Key Pair and Certificate"
- Section 17.1.2.5, "Recognizing Online Certificate Status Manager Certificates"

17.1.2.1. OCSP Signing Key Pair and Certificate

Every Online Certificate Status Manager has a certificate, the OCSP signing certificate, which has a public key corresponding to the private key the Online Certificate Status Manager uses to sign OCSP responses. The Online Certificate Status Manager's signature provides persistent proof that the Online Certificate Status Manager has processed the request. This certificate is generated when the Online Certificate Status Manager is configured. The default nickname for the certificate is **ocspSigningCert cert-***instance_ID*, where *instance_ID OSCP* is the Online Certificate Status Manager instance name.

17.1.2.2. SSL Server Key Pair and Certificate

Every Online Certificate Status Manager has at least one SSL server certificate which was generated when the Online Certificate Status Manager was configured. The default nickname for the certificate is **Server-Cert cert-***instance_ID*, where *instance_ID* identifies the Online Certificate Status Manager instance name.

The Online Certificate Status Manager uses its server certificate for server-side authentication for the Online Certificate Status Manager agent services page.

The Online Certificate Status Manager uses a single server certificate for authentication purposes. Additional server certificates can be installed and used for different purposes.

17.1.2.3. Subsystem Certificate

Every member of the security domain is issued a server certificate to use for communications among other domain members, which is separate from the server SSL certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-***instance_ID*.

17.1.2.4. Audit Log Signing Key Pair and Certificate

The OCSP keeps a secure audit log of all events which occurred on the server. To guarantee that the audit log has not been tampered with, the log file is signed by a special log signing certificate.

The audit log signing certificate is issued when the server is first configured.

NOTE

While other certificates can use ECC keys, the audit signing certificate must *always* use an RSA key.

17.1.2.5. Recognizing Online Certificate Status Manager Certificates

Depending on the CA which signed the Online Certificate Status Manager's SSL server certificate, it may be necessary to get the certificate and issuing CA recognized by the Certificate Manager.

- If the Online Certificate Status Manager's server certificate is signed by the CA that is publishing CRLs, then nothing needs to be done.
- If the Online Certificate Status Manager's server certificate is signed by the same root CA that signed the subordinate Certificate Manager's certificates, then the root CA must be marked as a trusted CA in the subordinate Certificate Manager's certificate database.
- If the Online Certificate Status Manager's SSL server certificate is signed by a different root CA, then the root CA certificate must be imported into the subordinate Certificate Manager's certificate database and marked as a trusted CA.

If the Online Certificate Status Manager's server certificate is signed by a CA within the selected security domain, the certificate chain is imported and marked when the Online Certificate Status Manager is configured. No other configuration is required. However, if the server certificate is signed by an external CA, the certificate chain has to be imported for the configuration to be completed.

NOTE

Not every CA within the security domain is automatically trusted by the OCSP Manager when it is configured. Every CA in the certificate chain of the CA configured in the CA panel is, however, trusted automatically by the OCSP Manager. Other CAs within the security domain but not in the certificate chain must be added manually.

17.1.3. Key Recovery Authority Certificates

The KRA uses the following key pairs and certificates:

- Section 17.1.3.1, “Transport Key Pair and Certificate”
- Section 17.1.3.2, “Storage Key Pair”
- Section 17.1.3.3, “SSL Server Certificate”
- Section 17.1.3.4, “Subsystem Certificate”
- Section 17.1.3.5, “Audit Log Signing Key Pair and Certificate”

17.1.3.1. Transport Key Pair and Certificate

Every KRA has a transport certificate. The public key of the key pair that is used to generate the transport certificate is used by the client software to encrypt an end entity's private encryption key before it is sent to the KRA for archival; only those clients capable of generating dual-key pairs use the transport certificate.

17.1.3.2. Storage Key Pair

Every KRA has a storage key pair. The KRA uses the public component of this key pair to encrypt (or wrap) private encryption keys when archiving the keys. It uses the private component to decrypt (or unwrap) the archived key during recovery. For more information on how this key pair is used, see Chapter 4, *Setting up Key Archival and Recovery*.

Keys encrypted with the storage key can be retrieved only by authorized key recovery agents.

17.1.3.3. SSL Server Certificate

Every Certificate System KRA has at least one SSL server certificate. The first SSL server certificate is generated when the KRA is configured. The default nickname for the certificate is **Server-Cert cert-***instance_ID*, where *instance_id* identifies the KRA instance is installed.

The KRA's SSL server certificate was issued by the CA to which the certificate request was submitted, which can be a Certificate System CA or a third-party CA. To view the issuer name, open the certificate details in the **System Keys and Certificates** option in the KRA Console.

The KRA uses its SSL server certificate for server-side authentication to the KRA agent services interface. By default, the Key Recovery Authority uses a single SSL server certificate for authentication. However, additional SSL server certificates can be requested and installed for the KRA.

17.1.3.4. Subsystem Certificate

Every member of the security domain is issued a server certificate to use for communications among other domain members, which is separate from the server SSL certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-***instance_ID*.

17.1.3.5. Audit Log Signing Key Pair and Certificate

The KRA keeps a secure audit log of all events which occurred on the server. To guarantee that the audit log has not been tampered with, the log file is signed by a special log signing certificate.

The audit log signing certificate is issued when the server is first configured.

NOTE

While other certificates can use ECC keys, the audit signing certificate must *always* use an RSA key.

17.1.4. TKS Certificates

The TKS has three certificates. The SSL server and subsystem certificates are used for standard operations. An additional signing certificate is used to protect audit logs.

- Section 17.1.4.1, “SSL Server Certificate”
- Section 17.1.4.2, “Subsystem Certificate”
- Section 17.1.4.3, “Audit Log Signing Key Pair and Certificate”

17.1.4.1. SSL Server Certificate

Every Certificate System TKS has at least one SSL server certificate. The first SSL server certificate is generated when the TKS is configured. The default nickname for the certificate is **Server-Cert cert-***instance_ID*.

17.1.4.2. Subsystem Certificate

Every member of the security domain is issued a server certificate to use for communications among other domain members, which is separate from the server SSL certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-***instance_ID*.

17.1.4.3. Audit Log Signing Key Pair and Certificate

The TKS keeps a secure audit log of all events which occurred on the server. To guarantee that the audit log has not been tampered with, the log file is signed by a special log signing certificate.

The audit log signing certificate is issued when the server is first configured.

NOTE

While other certificates can use ECC keys, the audit signing certificate must *always* use an RSA key.

17.1.5. TPS Certificates

The TPS only uses three certificates: a server certificate, subsystem certificate, and audit log signing certificate.

- Section 17.1.5.1, "SSL Server Certificate"
- Section 17.1.5.2, "Subsystem Certificate"
- Section 17.1.5.3, "Audit Log Signing Key Pair and Certificate"

17.1.5.1. SSL Server Certificate

Every Certificate System TPS has at least one SSL server certificate. The first SSL server certificate is generated when the TPS is configured. The default nickname for the certificate is **Server-Cert cert-***instance_ID*.

17.1.5.2. Subsystem Certificate

Every member of the security domain is issued a server certificate to use for communications among other domain members, which is separate from the server SSL certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-***instance_ID*.

17.1.5.3. Audit Log Signing Key Pair and Certificate

The TPS keeps a secure audit log of all events which occurred on the server. To guarantee that the audit log has not been tampered with, the log file is signed by a special log signing certificate.

The audit log signing certificate is issued when the server is first configured.

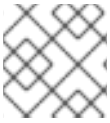
17.1.6. About Subsystem Certificate Key Types

When you create a new instance, you can specify the key type and key size in the configuration file passed to the **pkispawn** utility.

Example 17.1. Key Type-related Configuration Parameters for a CA

The following are key type-related parameters including example values. You can set these parameters in the configuration file which you pass to **pkispawn** when creating a new CA.

```
pki_ocsp_signing_key_algorithm=SHA256withRSA
pki_ocsp_signing_key_size=2048
pki_ocsp_signing_key_type=rsa

pki_ca_signing_key_algorithm=SHA256withRSA
pki_ca_signing_key_size=2048
pki_ca_signing_key_type=rsa

pki_sslserver_key_algorithm=SHA256withRSA
pki_sslserver_key_size=2048
pki_sslserver_key_type=rsa

pki_subsystem_key_algorithm=SHA256withRSA
pki_subsystem_key_size=2048
pki_subsystem_key_type=rsa

pki_admin_keysize=2048
pki_admin_key_size=2048
pki_admin_key_type=rsa

pki_audit_signing_key_algorithm=SHA256withRSA
pki_audit_signing_key_size=2048
pki_audit_signing_key_type=rsa
```



NOTE

The values in the example are for a CA. Other subsystems require different parameters.

For further details, see:

- The *Understanding the **pkispawn** Utility* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.
- The pki_default.cfg(5) man page for descriptions of the parameters and examples.

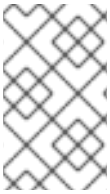
17.1.7. Using an HSM to Store Subsystem Certificates

By default, keys and certificates are stored in locally-managed databases, **key4.db** and **cert9.db**, respectively, in the **/var/lib/pki/*instance_name*/alias** directory. However, Red Hat Certificate System also supports hardware security modules (HSM), external devices which can store keys and certificates

in a centralized place on the network. Using an HSM can make some functions, like cloning, easier because the keys and certificates for the instance are readily accessible.

When an HSM is used to store certificates, then the HSM name is prepended to the certificate nickname, and the full name is used in the subsystem configuration, such as the **server.xml** file. For example:

```
serverCert="nethsm:Server-Cert cert-instance_ID
```



NOTE

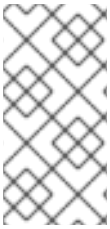
A single HSM can be used to store certificates and keys for mulitple subsystem instances, which may be installed on multiple hosts. When an HSM is used, any certificate nickname for a subsystem must be unique for every subsystem instance managed on the HSM.

Certificate System supports two types of HSM, nCipher netHSM and Chrysalis LunaSA.

17.2. REQUESTING CERTIFICATES THROUGH THE CONSOLE

The Certificate Setup Wizard for the CA, OCSP, KRA, and TKS automates the certificate enrollment process for subsystem certificates. The Console can create, submit, and install certificate requests and certificates for any of the certificates used by that subsystem. These certificates can be a server certificate or subsystem-specific certificate, such as a CA signing certificate or KRA transport certificate.

17.2.1. Requesting Signing Certificates



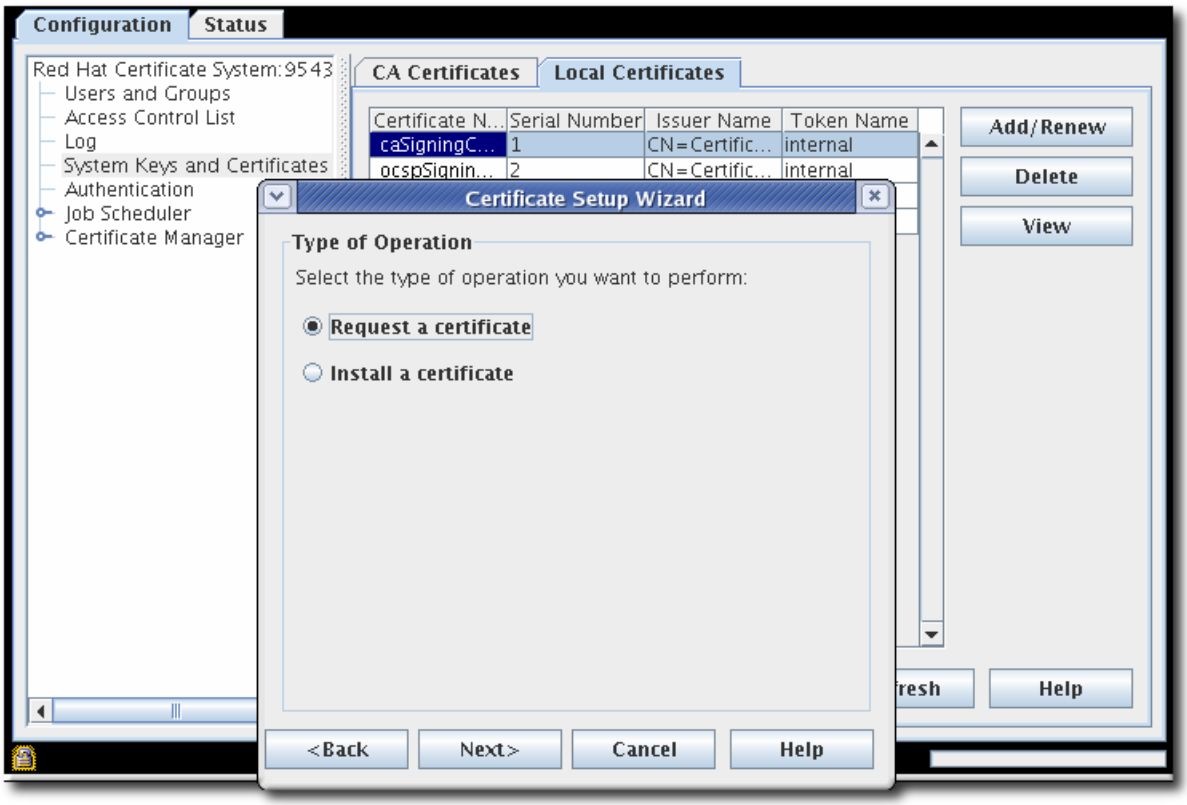
NOTE

It is important that the user generate and submit the client request from the computer that will be used later to access the subsystem because part of the request process generates a private key on the local machine. If location independence is required, use a hardware token, such as a smart card, to store the key pair and the certificate.

1. Open the subsystem console. For example:

```
pkiconsole https://server.example.com:8443/ca
```

2. In the **Configuration** tab, select **System Keys and Certificates** in the navigation tree.
3. In the right panel, select the **Local Certificates** tab.
4. Click **Add/Renew**.



5. Select the **Request a certificate** radio button.

6. Choose the signing certificate type to request.

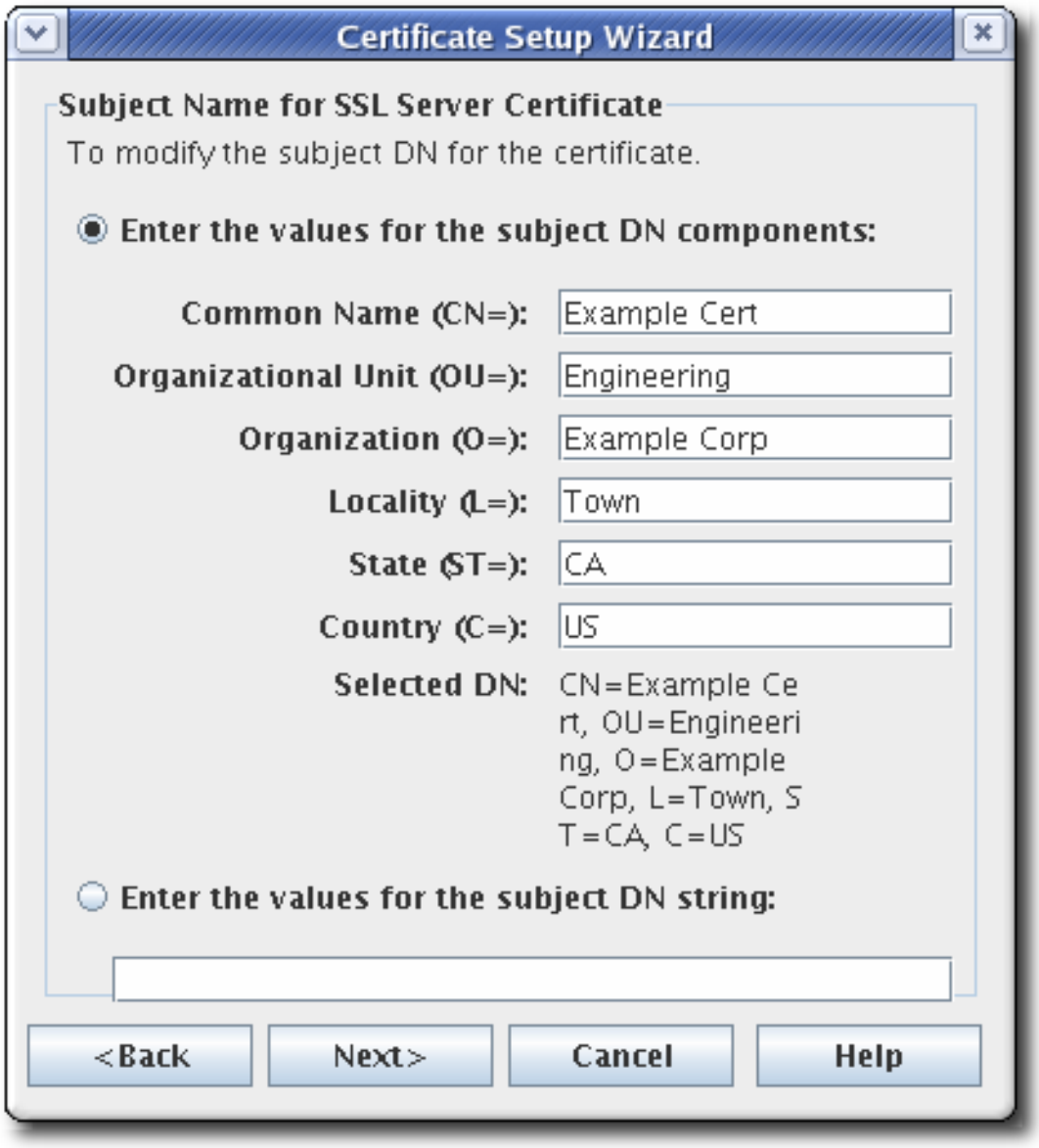
7. Select which type of CA will sign the request, either a root CA or a subordinate CA.

8. Set the key-pair information and set the location to generate the keys (the token), which can be either the internal security database directory or one of the listed external tokens.

   To create a new certificate, you must create a new key pair. Using an existing key pair will simply renew an existing certificate.

9. Select the message digest algorithm.

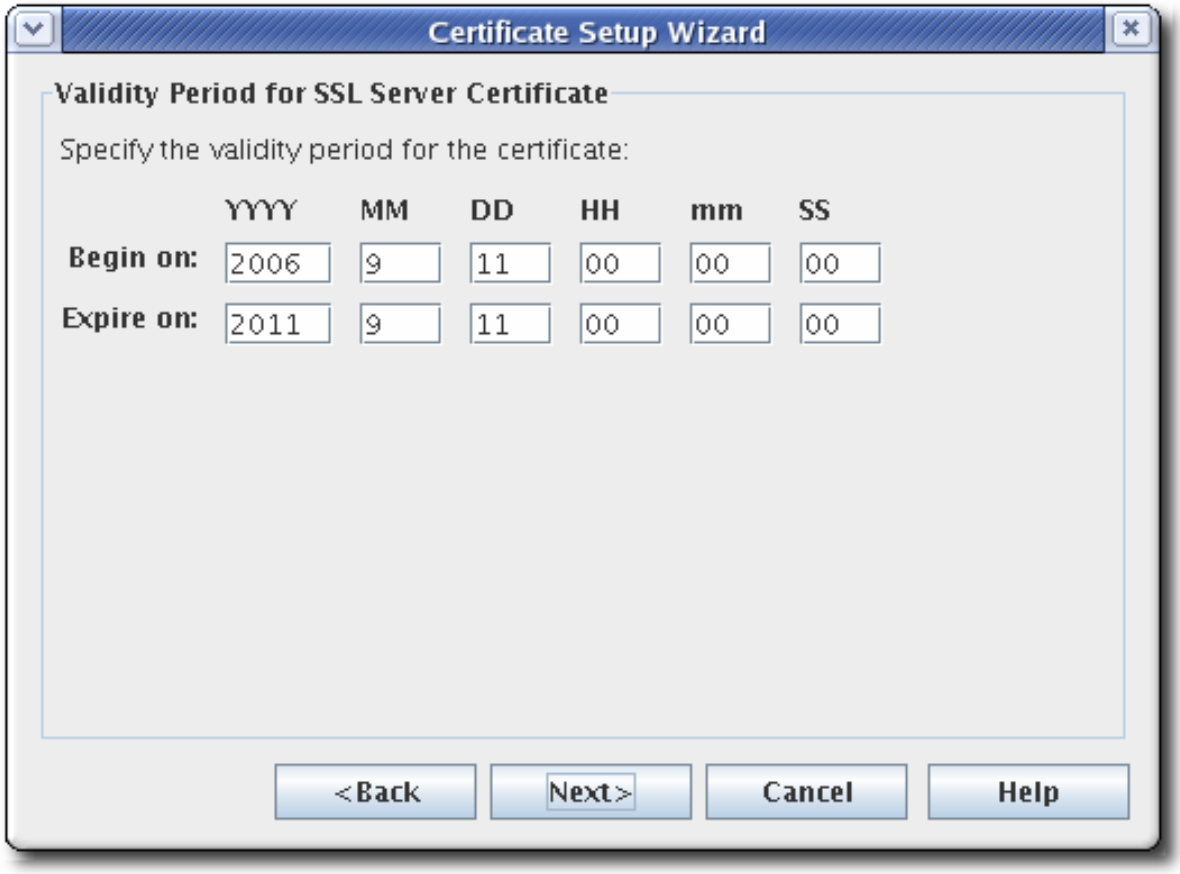
10. Give the subject name. Either enter values for individual DN attributes to build the subject DN or enter the full string.



The certificate request forms support all UTF-8 characters for the common name, organizational unit, and requester name fields.

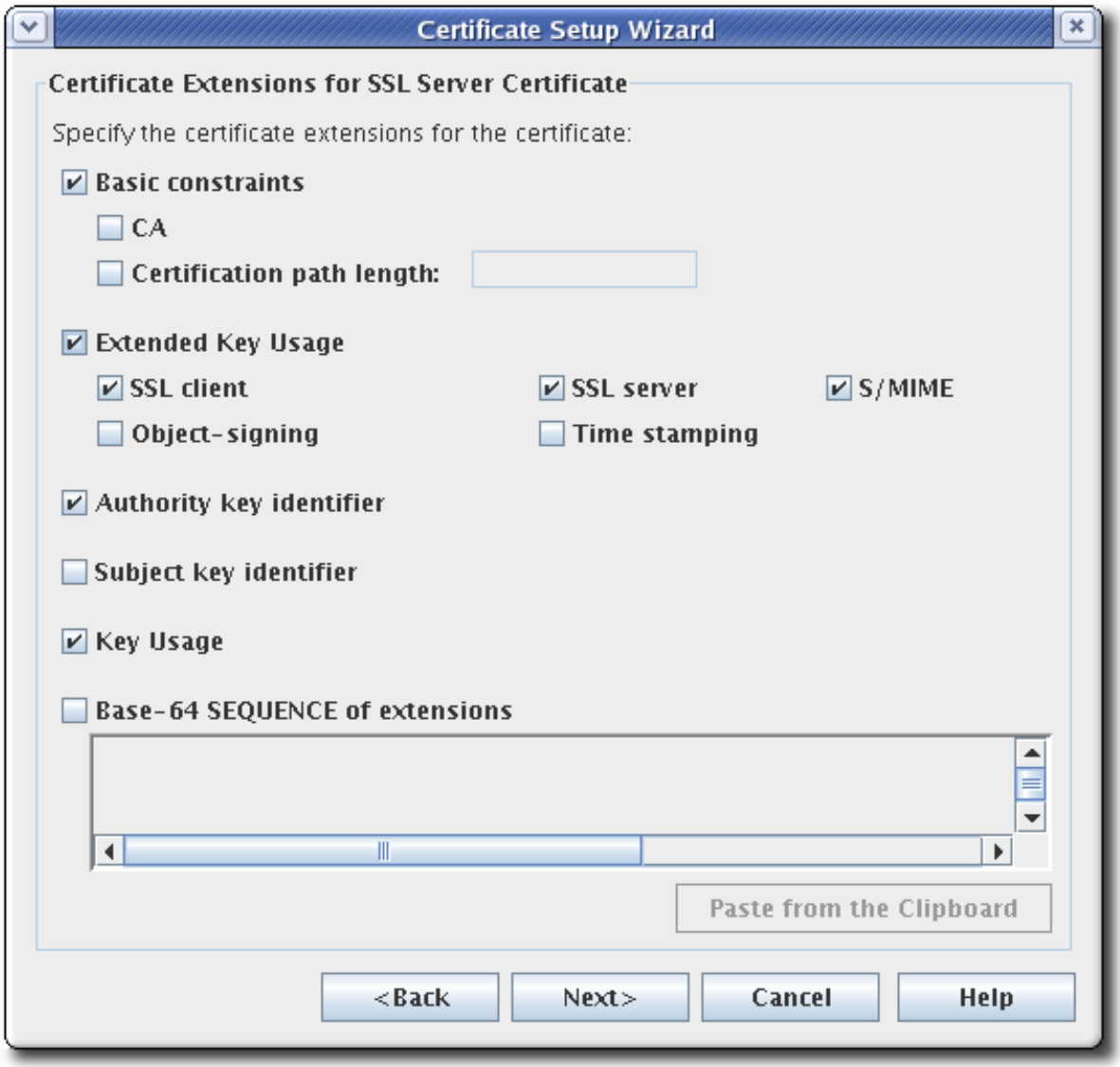
This support does not include supporting internationalized domain names.

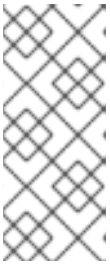
11. Specify the start and end dates of the validity period for the certificate and the time at which the validity period will start and end on those dates.



The default validity period is five years.

12. Set the standard extensions for the certificate. The required extensions are chosen by default. To change the default choices, read the guidelines explained in Appendix B, *Defaults, Constraints, and Extensions for Certificates and CRLs*.





NOTE

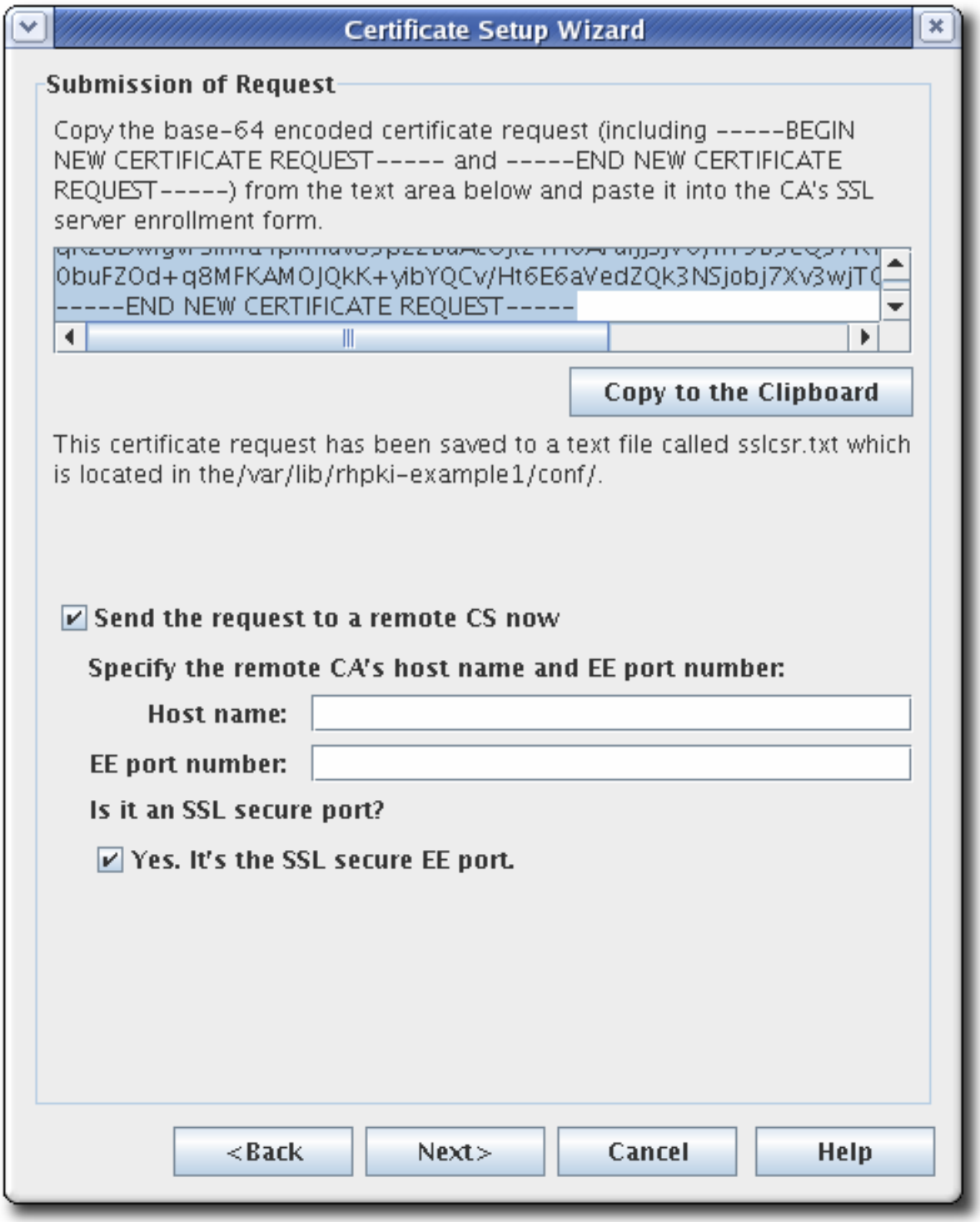
Certificate extensions are required to set up a CA hierarchy. Subordinate CAs must have certificates that include the extension identifying them as either a subordinate SSL CA (which allows them to issue certificates for SSL) or a subordinate email CA (which allows them to issue certificates for secure email). Disabling certificate extensions means that CA hierarchies cannot be set up.

- Basic Constraints. The associated fields are CA setting and a numeric setting for the certification path length.
- Extended Key Usage.
- Authority Key Identifier.
- Subject Key Identifier.
- Key Usage. The digital signature (bit 0), non-repudiation (bit 1), key certificate sign (bit 5), and CRL sign (bit 6) bits are set by default. The extension is marked critical as recommended by the PKIX standard and RFC 2459. See RFC 2459 for a description of the Key Usage extension.

- Base-64 SEQUENCE of extensions. This is for custom extensions. Paste the extension in MIME 64 DER-encoded format into the text field.

To add multiple extensions, use the **ExtJoiner** program. For information on using the tools, see the *Certificate System Command-Line Tools Guide*.

13. The wizard generates the key pairs and displays the certificate signing request.



The request is in base-64 encoded PKCS #10 format and is bounded by the marker lines **-----BEGIN NEW CERTIFICATE REQUEST-----** and **-----END NEW CERTIFICATE REQUEST-----**. For example:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
```

```
MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBC6SAwHgYDVQQKExdOZXRzY2F
wZSBDb21tdW5pY2
F0aW9ucczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFo
XDTk5MDIyMzE5MDA
wMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQLEw
ZQZW9wbGUxFz
AVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0eTEjMCEGC
SqGSIb3Dbndg
JARYUc3Vwcml5Yhvfggsvwryw4y7214vAOBgNVHQ8BAf8EBAMCBLAwFAYJYIZIAYb4QgEB
AQHBAQDAgCAM
A0GCSqGSIb3DQEBBAUAA4GBAFi9FzyJlLmS+kzsue0kTXawbwamGdYql2w4hIBgdR+jWeL
mD4CP4x
-----END NEW CERTIFICATE REQUEST-----
```

The wizard also copies the certificate request to a text file it creates in the configuration directory, which is located in **/var/lib/pki/*instance_name*/*subsystem_type*/conf/**. The name of the text file depends on the type of certificate requested. The possible text files are listed in Table 17.1, “Files Created for Certificate Signing Requests” .

Table 17.1. Files Created for Certificate Signing Requests

Filename	Certificate Signing Request
cacsr.txt	CA signing certificate
ocspcsr.txt	Certificate Manager OCSP signing certificate
ocspcsr.txt	OCSP signing certificate

Do not modify the certificate request before sending it to the CA. The request can either be submitted automatically through the wizard or copied to the clipboard and manually submitted to the CA through its end-entities page.

NOTE

The wizard's auto-submission feature can submit requests to a remote Certificate Manager only. It cannot be used for submitting the request to a third-party CA. To submit it to a third-party CA, use the certificate request file.

14. Retrieve the certificate.

    1. Open the Certificate Manager end-entities page.

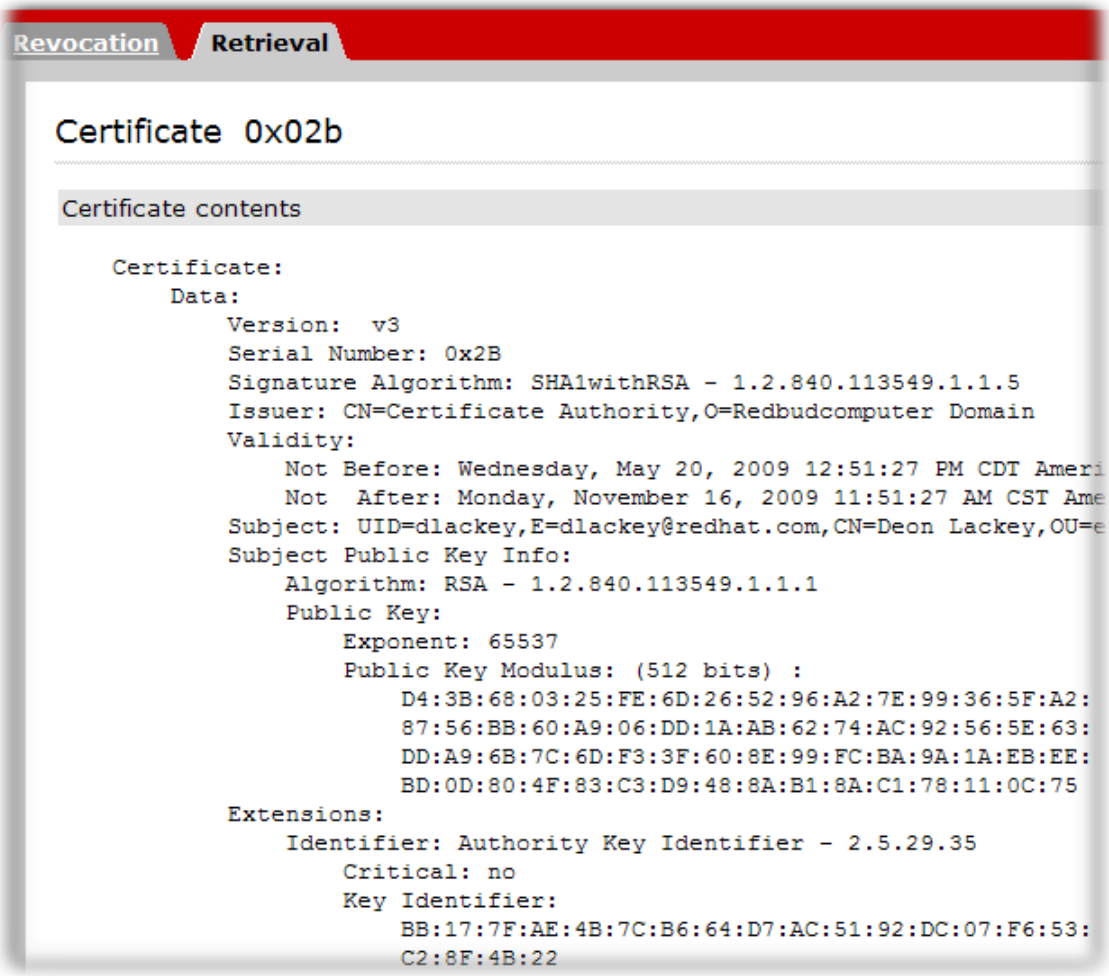
       ```
       https://server.example.com:8443/ca/ee/ca
       ```

    2. Click the **Retrieval** tab.

    3. Fill in the request ID number that was created when the certificate request was submitted, and click **Submit**.

    4. The next page shows the status of the certificate request. If the status is **complete**, then there is a link to the certificate. Click the **Issued certificate** link.

5. The new certificate information is shown in pretty-print format, in base-64 encoded format, and in PKCS #7 format.



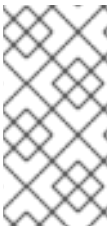
6. Copy the base-64 encoded certificate, including the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** marker lines, to a text file. Save the text file, and use it to store a

copy of the certificate in a subsystem's internal database. See Section 15.3.2.1, "Creating Users".

NOTE

pkiconsole is being deprecated.

17.2.2. Requesting Other Certificates



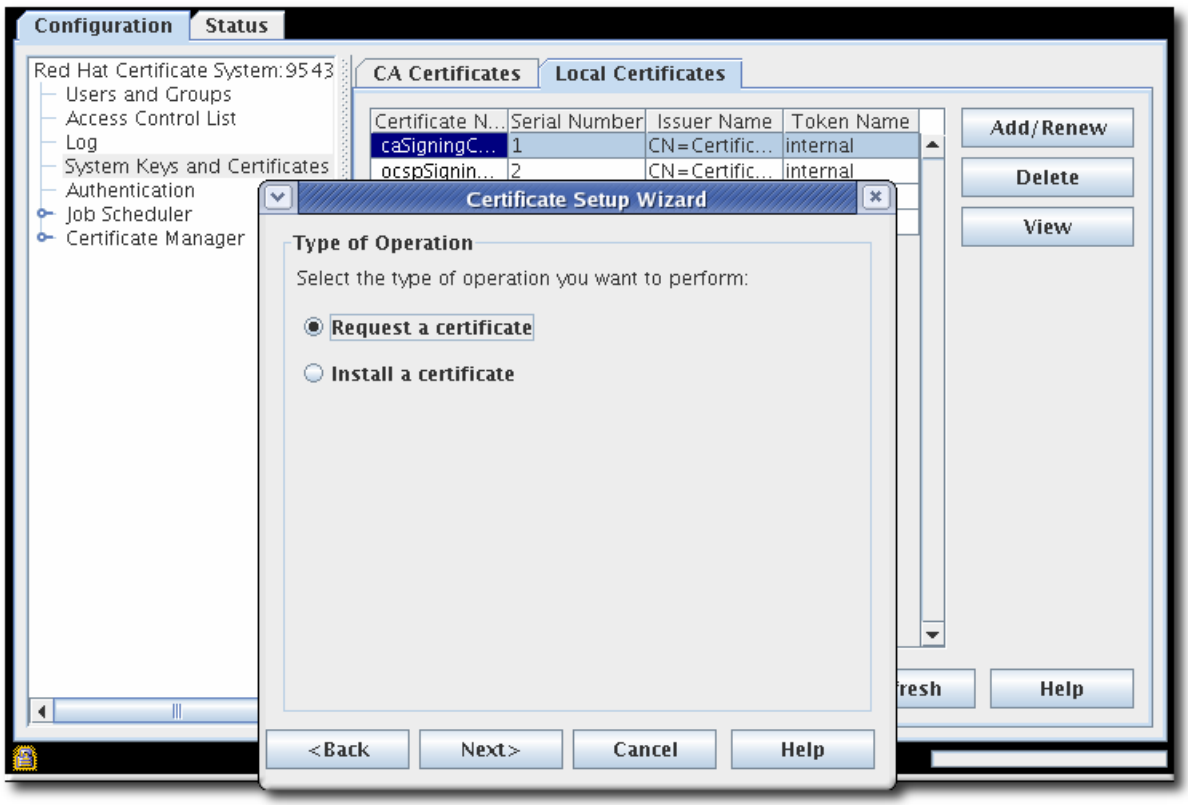
NOTE

It is important that the user generate and submit the client request from the computer that will be used later to access the subsystem because part of the request process generates a private key on the local machine. If location independence is required, use a hardware token, such as a smart card, to store the key pair and the certificate.

1. Open the subsystem console. For example:

```
pkiconsole https://server.example.com:8443/ca
```

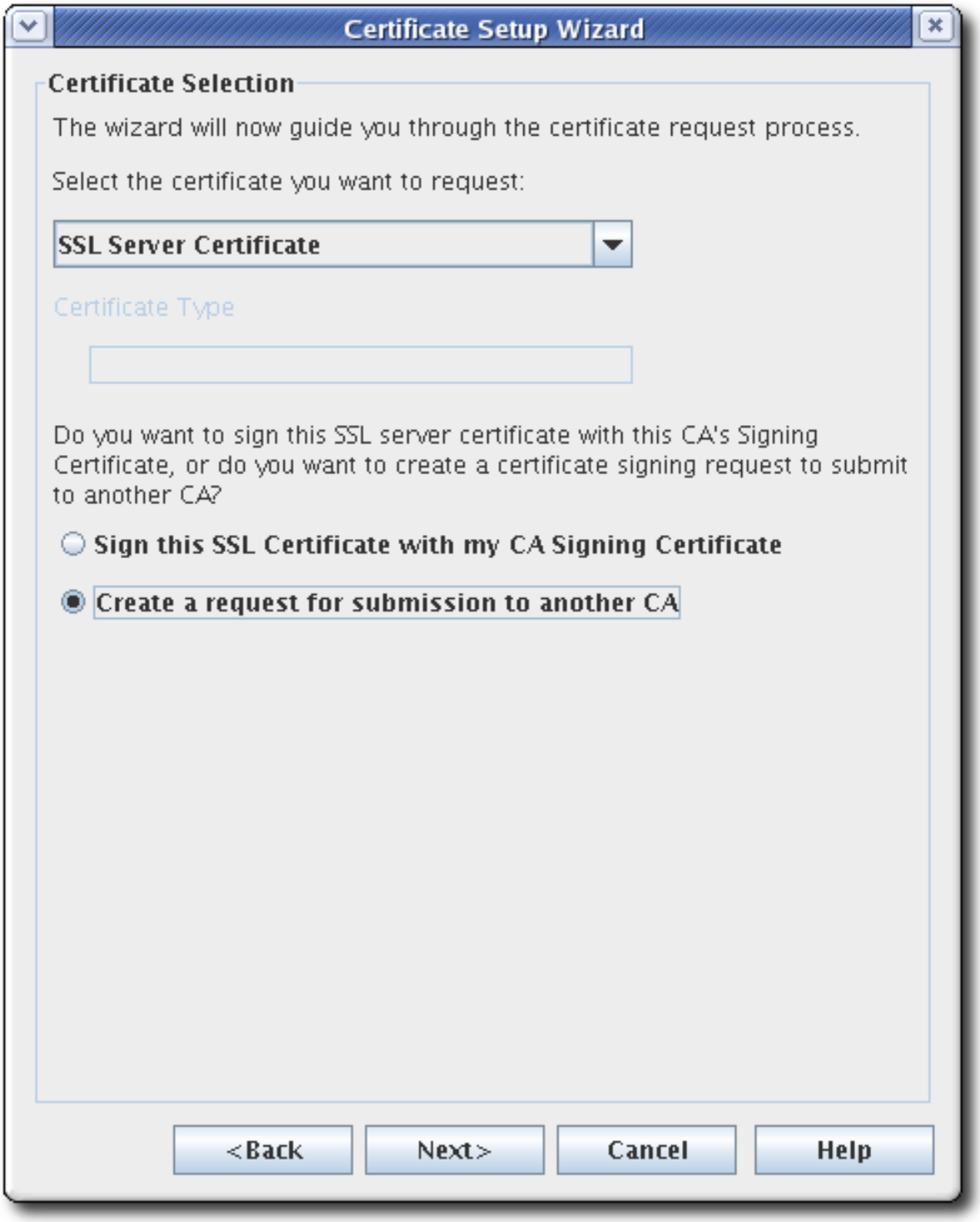
2. In the **Configuration** tab, select **System Keys and Certificates** in the navigation tree.
3. In the right panel, select the **Local Certificates** tab.
4. Click **Add/Renew**.



5. Select the **Request a certificate** radio button.
6. Choose the certificate type to request. The types of certificates that can be requested varies depending on the subsystem.

NOTE

If selecting to create an "other" certificate, the **Certificate Type** field becomes active. Fill in the type of certificate to create, either **caCrlSigning** for the CRL signing certificate, **caSignedLogCert** for an audit log signing certificate, or **client** for an SSL client certificate.

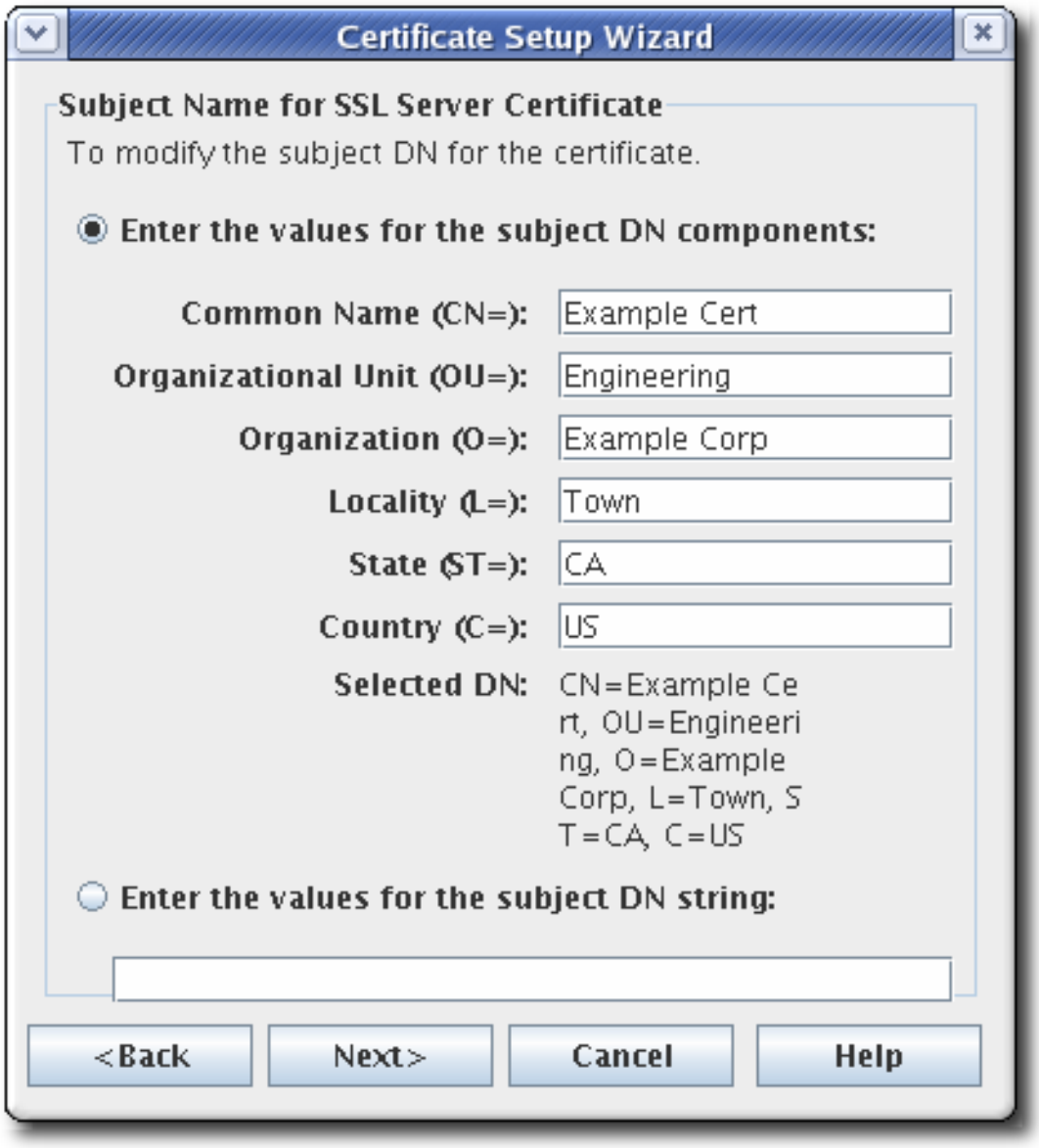


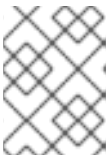
7. Select which type of CA will sign the request. The options are to use the local CA signing certificate or to create a request to submit to another CA.

8. Set the key-pair information and set the location to generate the keys (the token), which can be either the internal security database directory or one of the listed external tokens.

To create a new certificate, you must create a new key pair. Using an existing key pair will simply renew an existing certificate.

9. Give the subject name. Either enter values for individual DN attributes to build the subject DN or enter the full string.





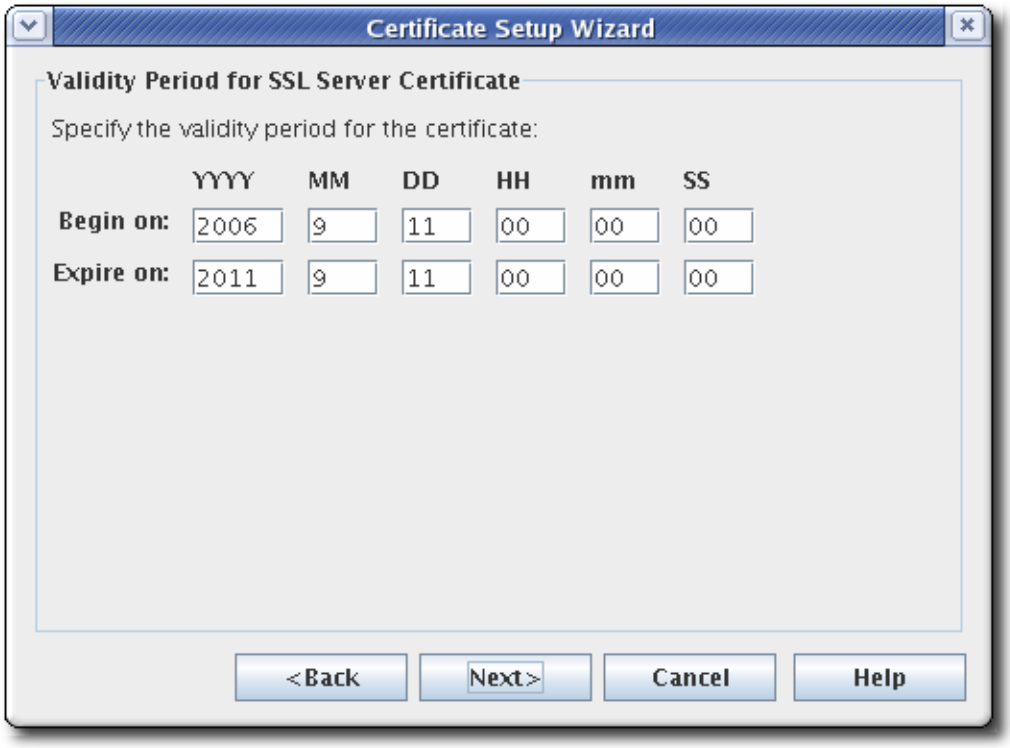
NOTE

For an SSL server certificate, the common name must be the fully-qualified host name of the Certificate System in the format *machine_name.domain.domain*.

The CA certificate request forms support all UTF-8 characters for the common name, organizational unit, and requester name fields.

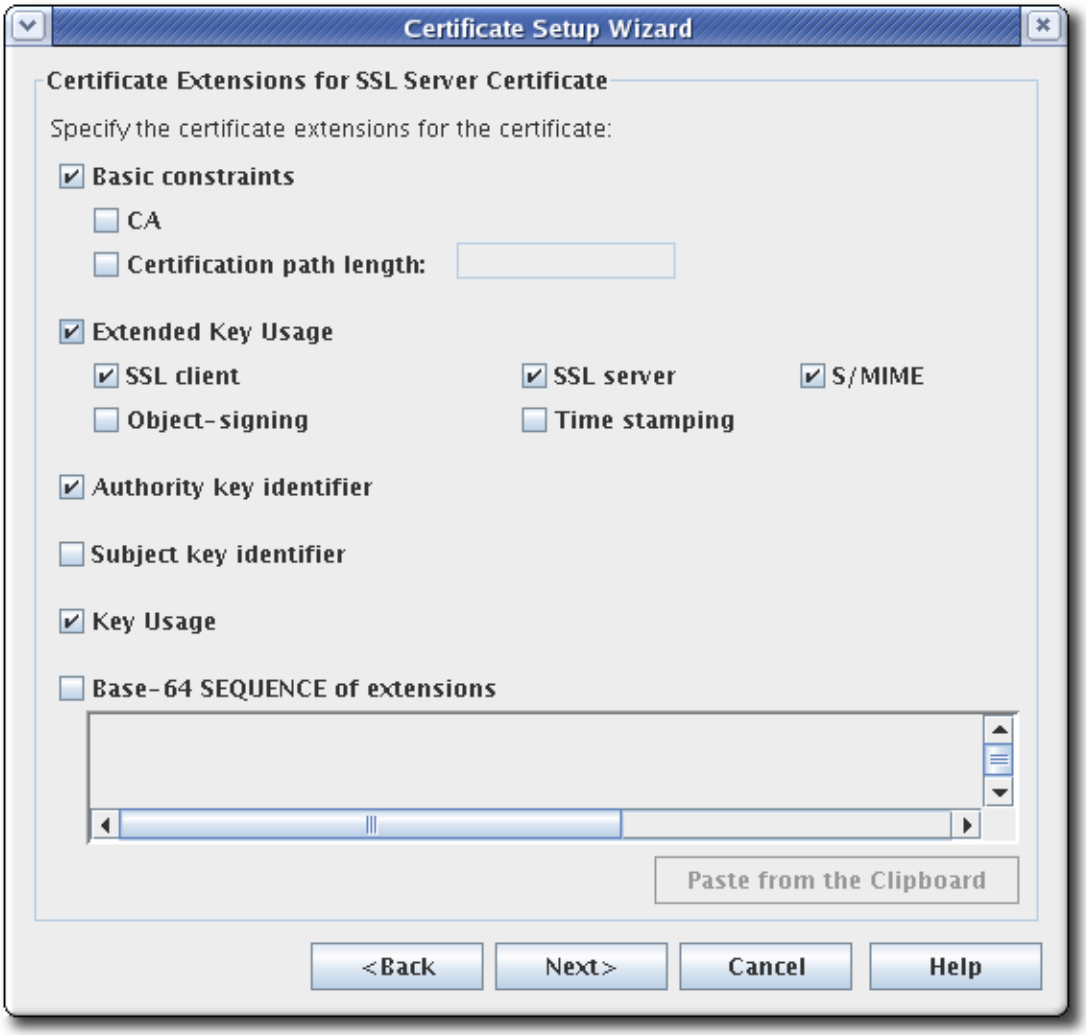
This support does not include supporting internationalized domain names.

10. Specify the start and end dates of the validity period for the certificate and the time at which the validity period will start and end on those dates.



The default validity period is five years.

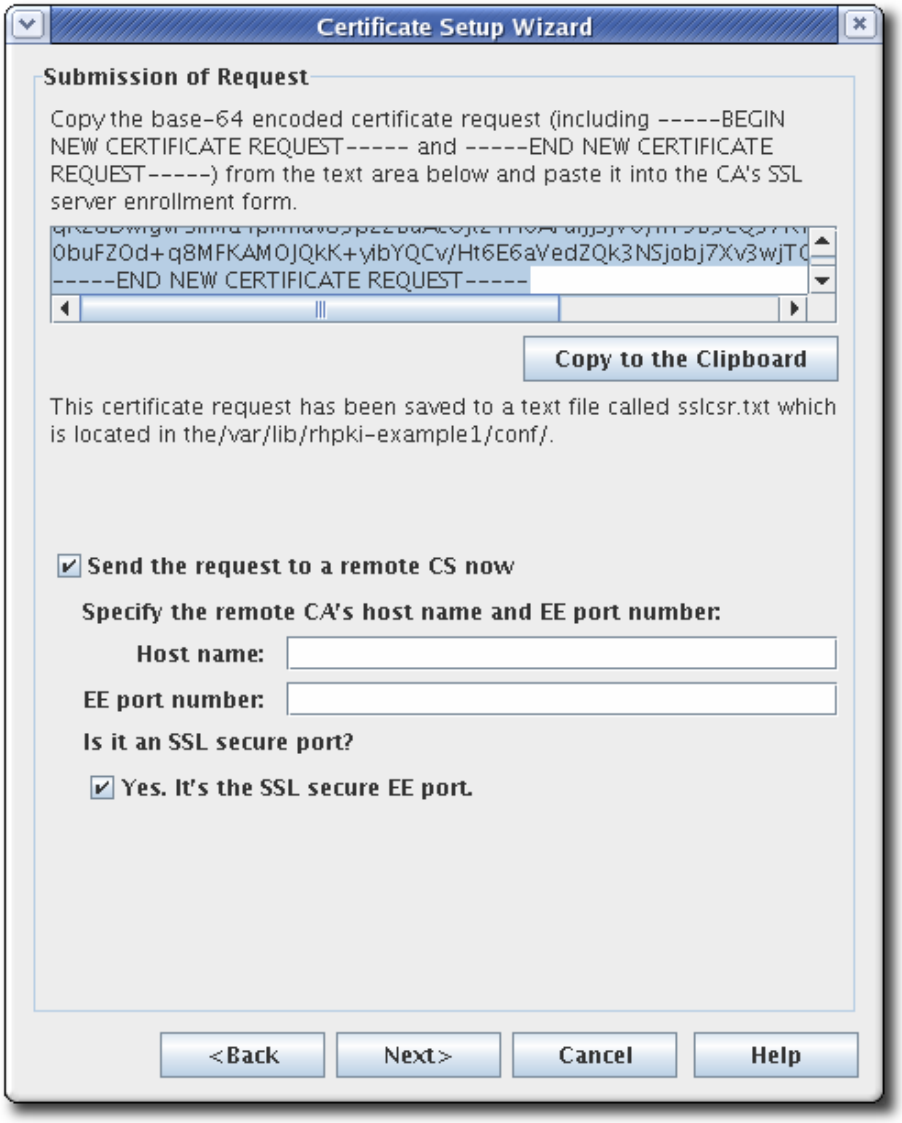
11. Set the standard extensions for the certificate. The required extensions are chosen by default. To change the default choices, read the guidelines explained in Appendix B, *Defaults, Constraints, and Extensions for Certificates and CRLs*.



- Extended Key Usage.
- Authority Key Identifier.
- Subject Key Identifier.
- Key Usage. The digital signature (bit 0), non-repudiation (bit 1), key certificate sign (bit 5), and CRL sign (bit 6) bits are set by default. The extension is marked critical as recommended by the PKIX standard and RFC 2459. See RFC 2459 for a description of the Key Usage extension.
- Base-64 SEQUENCE of extensions. This is for custom extensions. Paste the extension in MIME 64 DER-encoded format into the text field.

To add multiple extensions, use the **ExtJoiner** program. For information on using the tools, see the *Certificate System Command-Line Tools Guide*.

12. The wizard generates the key pairs and displays the certificate signing request.



The request is in base-64 encoded PKCS #10 format and is bounded by the marker lines **-----BEGIN NEW CERTIFICATE REQUEST-----** and **-----END NEW CERTIFICATE REQUEST-----**. For example:

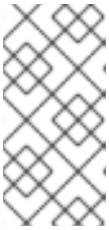
```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBC6SAwHgYDVQQKExdOZXRzY2F
wZSBDb21tdW5pY2
F0aW9uczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFo
XDTk5MDIyMzE5MDA
wMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQLEw
ZQZW9wbGUxFz
AVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0eTEjMCEGC
SqGSIb3Dbndg
JARYUc3Vwcml5Yhvfggsvwryw4y7214vAOBgNVHQ8BAf8EBAMCBLAwFAYJYIZIAYb4QgEB
AQHBAQDAgCAM
A0GCSqGSIb3DQEBBAUAA4GBAFi9FzyJlLmS+kzsue0kTXawbwamGdYql2w4hIBgdR+jWeL
mD4CP4x
-----END NEW CERTIFICATE REQUEST-----
```

The wizard also copies the certificate request to a text file it creates in the configuration directory, which is located in **/var/lib/pki/*instance_name*/*subsystem_type*/conf/**. The name of the text file depends on the type of certificate requested. The possible text files are listed in Table 17.2, “Files Created for Certificate Signing Requests” .

Table 17.2. Files Created for Certificate Signing Requests

Filename	Certificate Signing Request
kracsr.txt	KRA transport certificate
sslcsr.txt	SSL server certificate
othercsr.txt	Other certificates, such as Certificate Manager CRL signing certificate or SSL client certificate

Do not modify the certificate request before sending it to the CA. The request can either be submitted automatically through the wizard or copied to the clipboard and manually submitted to the CA through its end-entities page.



NOTE

The wizard's auto-submission feature can submit requests to a remote Certificate Manager only. It cannot be used for submitting the request to a third-party CA. To submit the request to a third-party CA, use one of the certificate request files.

13. Retrieve the certificate.

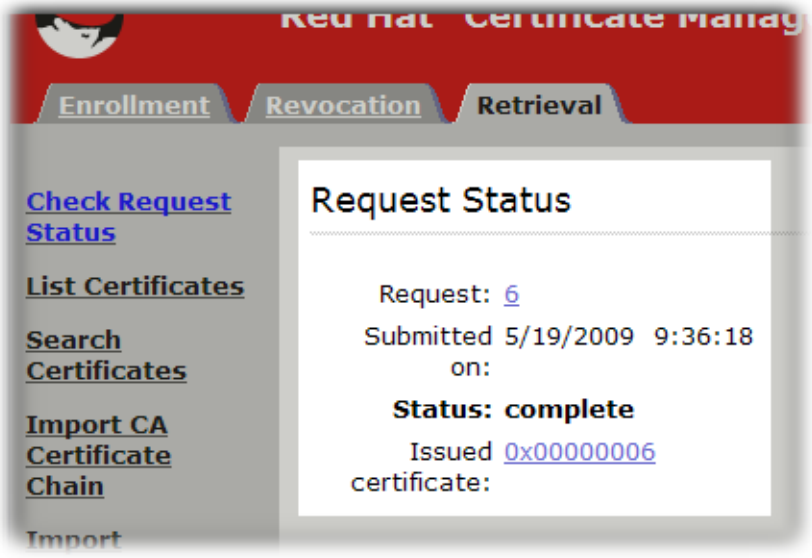
    1. Open the Certificate Manager end-entities page.

       ```
       https://server.example.com:8443/ca/ee/ca
       ```

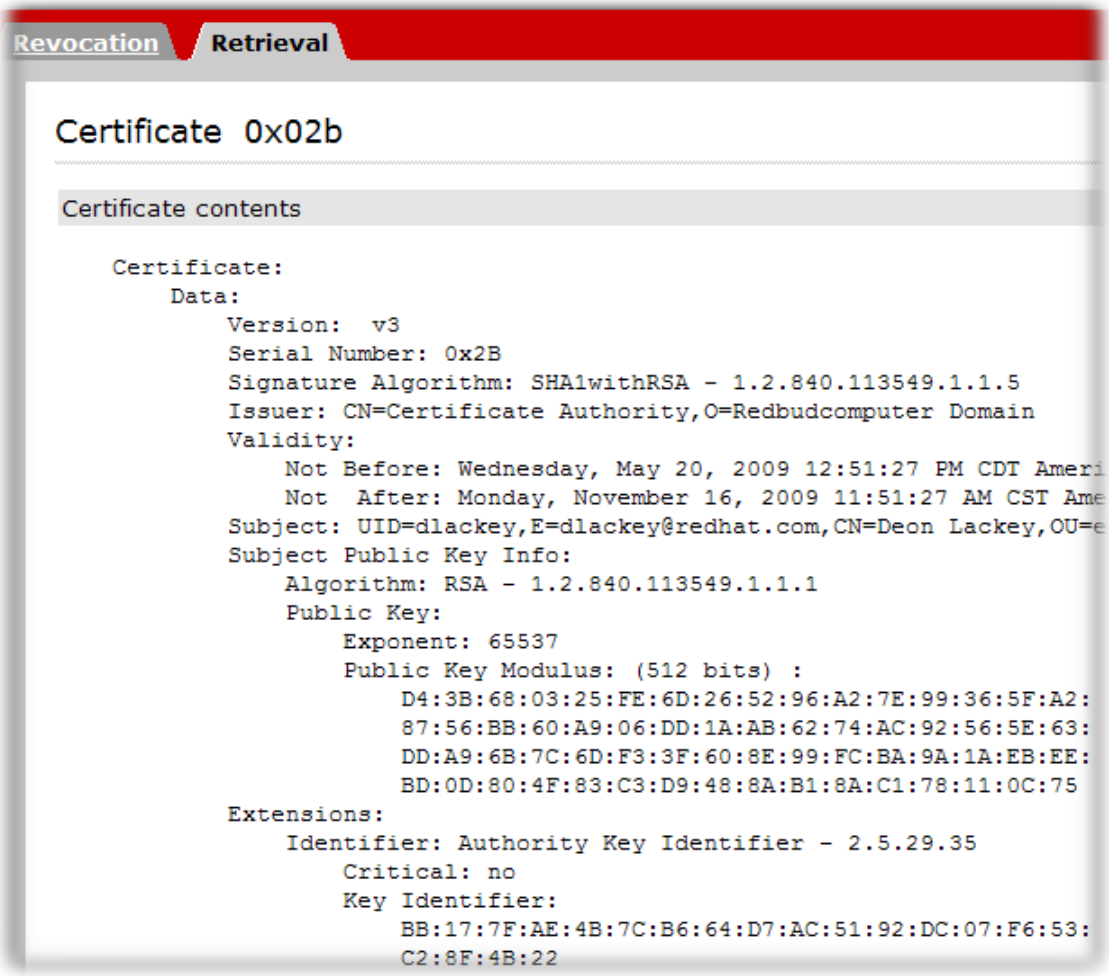
    2. Click the **Retrieval** tab.

    3. Fill in the request ID number that was created when the certificate request was submitted, and click **Submit**.

    4. The next page shows the status of the certificate request. If the status is **complete**, then there is a link to the certificate. Click the **Issued certificate** link.



5. The new certificate information is shown in pretty-print format, in base-64 encoded format, and in PKCS #7 format.



6. Copy the base-64 encoded certificate, including the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** marker lines, to a text file. Save the text file, and use it to store a copy of the certificate in a subsystem's internal database. See Section 15.3.2.1, "Creating Users".

17.3. RENEWING SUBSYSTEM CERTIFICATES

There are two methods of renewing a certificate. *Regenerating* the certificate takes its original key and its original profile and request, and recreates an identical key with a new validity period and expiration date. *Re-keying* a certificate resubmits the initial certificate request to the original profile, but generates a new key pair. Administrator certificates can be renewed by being re-keyed.

17.3.1. Re-keying Certificates in the End-Entities Forms

Subsystem certificates can be renewed directly in the end user enrollment forms, using the serial number of the original certificate.

1. Renew the certificates in the CA's end-entities forms, as described in Section 5.4, "Renewing Certificates". This requires the serial number of the subsystem certificate being renewed.

2. Import the certificate into the subsystem's database, as described in Section 17.6.1, "Installing Certificates in the Certificate System Database". The certificate can be imported using **certutil** or the console. For example:

```
certutil -A -n "ServerCert cert-example" -t u,u,u -d /var/lib/pki/instance_name/alias -a -i
/tmp/example.cert
```

17.3.2. Renewing Certificates in the Console

The Java subsystems can renew any of their subsystem certificates through their administrative console. The process is exactly the same as requesting new subsystem certificates (Section 17.2, "Requesting Certificates through the Console"), with one crucial difference: renewal uses an *existing key pair* rather than generating a new one.

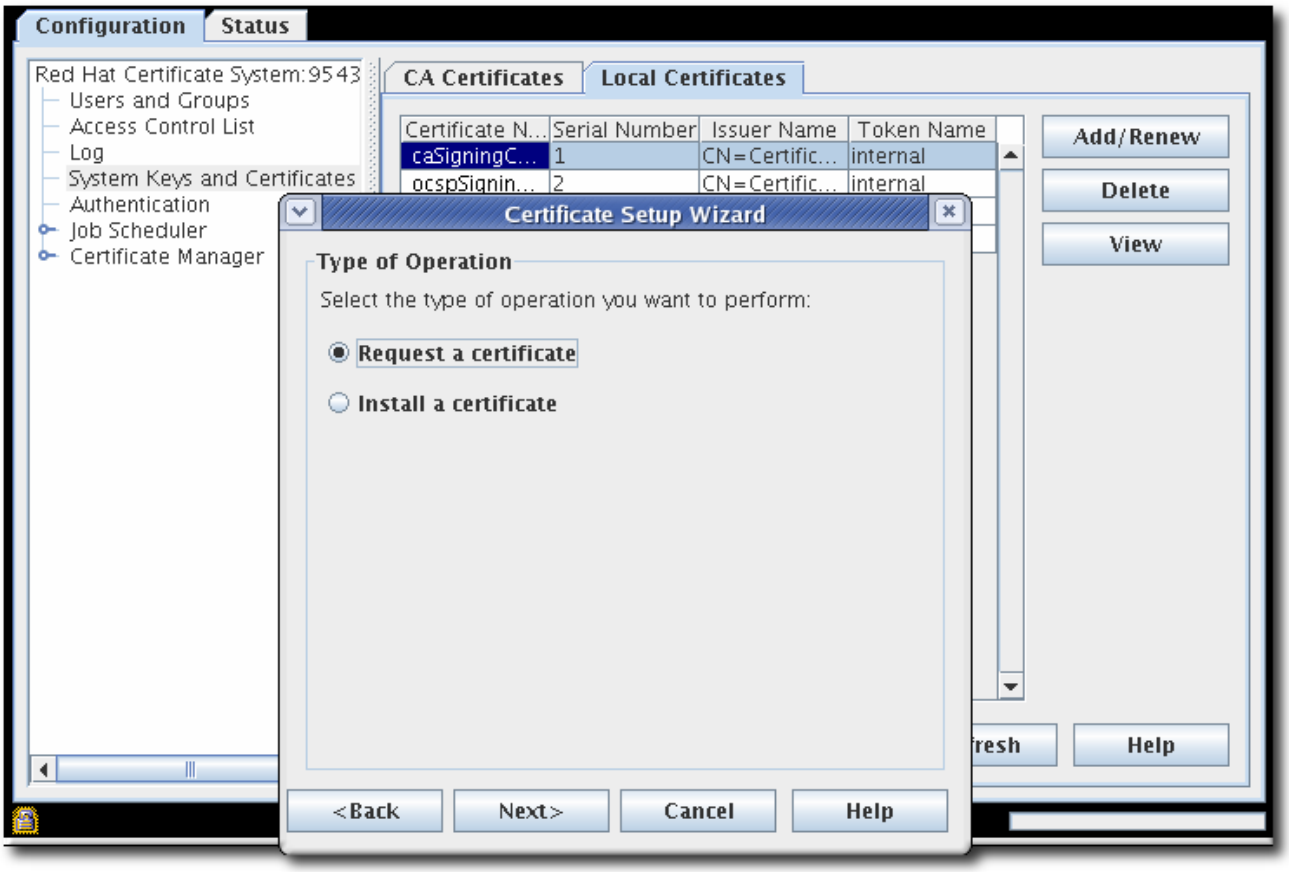


Figure 17.1. Renewing Subsystem Certificate

After renewing a certificate, then delete the original certificate from the database (Section 17.6.3, "Deleting Certificates from the Database").

17.3.3. Renewing Certificates Using certutil

certutil can be used to generate a certificate request using an existing key pair in the certificate database. The new certificate request can then be submitted through the regular profile pages for the CA to issue a renewed certificate.

NOTE

Encryption and signing certificates are created in a single step. However, the renewal process only renews one certificate at a time.

To renew both certificates in a certificate pair, each one has to be renewed individually.

1. Get the password for the token database.

```
cat /var/lib/pki/instance_name/conf/password.conf

internal=263163888660
```

2. Open the certificate database directory of the instance whose certificate is being renewed.

```
cd /var/lib/pki/instance_name/alias
```

3. List the key and nickname for the certificate being renewed. In order to renew a certificate, the key pairs used to generate and the subject name given to the new certificate must be the same as the one in the old certificate.

```
# certutil -K -d .

certutil: Checking token "NSS Certificate DB" in slot "NSS User Private Key and Certificate
Services"
Enter Password or Pin for "NSS Certificate DB":
< 0> rsa      69481646e38a6154dc105960aa24ccf61309d37d   caSigningCert cert-pki-tomcat
CA
```

4. Copy the **alias** directory as a backup, then delete the original certificate from the certificate database. For example:

```
certutil -D -n "ServerCert cert-example"  -d .
```

5. Run the **certutil** command with the options set to the values in the existing certificate.

```
certutil -d . -R -n "NSS Certificate DB:cert-pki-tomcat CA" -s "cn=CA Authority,o=Example
Domain" -a -o example.req2.txt
```

   The difference between generating a new certificate and key pair and renewing the certificate is the value of the **-n** option. To generate an entirely new request and key pair, then **-k** sets the key type and is used with **-g**, which sets the bit length. For a renewal request, the **-n** option uses the certificate nickname to access the existing key pair stored in the security database.

   For further details about the parameters, see the certutil(1) man page.

6. Submit the certificate request and then retrieve it and install it, as described in Section 5.3, “Requesting and Receiving Certificates”.

17.3.4. Renewing System Certificates

Certificate System does not automatically renew system certificates online while the PKI server is running. However, if a system certificate expires, Certificate System will fail to start.

To renew system certificates:

1. If the system certificate is expired:

   a. Create a temporary certificate:

```
# pki-server cert-create sslserver --temp
```

b. Import the temporary certificate into Certificate System's Network Security Services (NSS) database:

```
# pki-server cert-import sslserver
```

c. Start Certificate System:

```
# pki-server start instance_name
```

2. Display the certificates and note the ID of the expired system certificate:

```
# pki-server cert-find
```

3. Create the new permanent certificate:

```
# pki-server cert-create certificate_ID
```

4. Stop Certificate System:

```
# pki-server stop instance_name
```

5. Import the new certificate to replace the expired certificate:

```
# pki-server cert-import certificate_ID
```

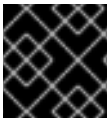
6. Start Certificate System:

```
# pki-server start instance_name
```

17.4. CHANGING THE NAMES OF SUBSYSTEM CERTIFICATES

One alternative to renewing certificates is replacing them with new certificates, meaning that a new certificate is generated with new keys. Generally, a new certificate can be added to the database and the old one deleted, a simple one-to-one swap. This is possible because the individual subsystem servers identify certificates based on their nickname; as long as the certificate nickname remains the same, the server can find the required certificate even if other factors – like the subject name, serial number, or key – are different.

However, in some situations, the new certificate may have a new certificate nickname, as well. In that case, the certificate nickname needs to be updated in all of the required settings in the subsystem's **CS.cfg** configuration file.



IMPORTANT

Always restart a subsystem after editing the **CS.cfg** file.

These tables list all of the configuration parameters for each of the subsystem's certificates:

- Table 17.3, “CA Certificate Nickname Parameters”
- Table 17.4, “KRA Certificate Nickname Parameters”
- Table 17.5, “OCSP Certificate Nickname Parameters”
- Table 17.6, “TKS Certificate Nickname Parameters”
- Table 17.7, “TPS Nickname Parameters in CS.cfg”

Table 17.3. CA Certificate Nickname Parameters

CA Signing Certificate	• ca.cert.signing.nickname • ca.signing.cacertnickname • ca.signing.certnickname • ca.signing.nickname • cloning.signing.nickname
OCSP Signing Certificate	• ca.ocsp_signing.cacertnickname • ca.ocsp_signing.certnickname • ca.cert.ocsp_signing.nickname • ca.ocsp_signing.nickname • cloning.ocsp_signing.nickname
Subsystem Certificate	• ca.cert.subsystem.nickname • ca.subsystem.nickname • cloning.subsystem.nickname • pkiremove.cert.subsystem.nickname
Server Certificate	• ca.sslserver.nickname • ca.cert.sslserver.nickname
Audit Signing Certificate	• ca.audit_signing.nickname • ca.cert.audit_signing.nickname • cloning.audit_signing.nickname

Table 17.4. KRA Certificate Nickname Parameters

Transport Certificate	• cloning.transport.nickname • kra.cert.transport.nickname • kra.transport.nickname • tks.kra_transport_cert_nickname Note that this parameter is in the TKS configuration file. This needs changed in the TKS configuration if the *KRA* transport certificate nickname changes, even if the TKS certificates all stay the same.
Storage Certificate	• cloning.storage.nickname • kra.storage.nickname • kra.cert.storage.nickname
Server Certificate	• kra.cert.sslserver.nickname • kra.sslserver.nickname
Subsystem Certificate	• cloning.subsystem.nickname • kra.cert.subsystem.nickname • kra.subsystem.nickname • pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	• cloning.audit_signing.nickname • kra.cert.audit_signing.nickname • kra.audit_signing.nickname

Table 17.5. OCSP Certificate Nickname Parameters

OCSP Signing Certificate	• cloning.signing.nickname • ocsp.signing.certnickname • ocsp.signing.cacertnickname • ocsp.signing.nickname

Server Certificate	• ocsp.cert.sslserver.nickname • ocsp.sslserver.nickname
Subsystem Certificate	• cloning.subsystem.nickname • ocsp.subsystem.nickname • ocsp.cert.subsystem.nickname • pkiremove.cert.subsystem
Audit Log Signing Certificate	• cloning.audit_signing.nickname • ocsp.audit_signing.nickname • ocsp.cert.audit_signing.nickname

Table 17.6. TKS Certificate Nickname Parameters

KRA Transport Certificate[a]	• tks.kra_transport_cert_nickname
Server Certificate	• tks.cert.sslserver.nickname • tks.sslserver.nickname
Subsystem Certificate	• cloning.subsystem.nickname • tks.cert.subsystem.nickname • tks.subsystem.nickname • pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	• cloning.audit_signing.nickname • tks.audit_signing.nickname • tks.cert.audit_signing.nickname

[a] This needs changed in the TKS configuration if the *KRA* transport certificate nickname changes, even if the TKS certificates all stay the same.

Table 17.7. TPS Nickname Parameters in CS.cfg

Server Certificate	• tps.cert.sslserver.nickname
Subsystem Certificate	• tps.cert.subsystem.nickname • selftests.plugin.TPSValidity.nickname • selftests.plugin.TPSPresence.nickname • pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	• tps.cert.audit_signing.nickname

17.5. USING CROSS-PAIR CERTIFICATES

In the late 1990s, as the US government began enhancing its public key infrastructure, it became apparent that branches of government with their own, separate PKI deployments still needed to be able to recognize and trust each others certificates *as if* the certificates were issued from their own CA. (The method of getting certificates trusted outside a network for external clients to use is a serious, not easily resolved issue for any PKI administrator.)

The US government devised a standard for issuing *cross-pair certificates* called the Federal Bridge Certificate Authority. These certificates are also called *bridge certificates*, for obvious reasons. Bridge or cross-pair certificates are CA signing certificate that are framed as dual certificate pairs, similar to encryption and signing certificate pairs for users, only each certificate in the pair is issued by a different CA. Both partner CAs store the other CA signing certificate in its database, so all of the certificates issued within the other PKI are trusted and recognized.

Bridging certificates honors certificates issued by a CA that is not chained to the root CA in its own PKI. By establishing a trust between the Certificate System CA and another CA through a cross-pair CA certificate, the cross-pair certificate can be downloaded and used to trust the certificates issued by the other CA, just as downloading and installing a single CA certificate trusts all certificates issued by the CA.

The Certificate System can issue, import, and publish cross-pair CA certificates. A special profile must be created for issuing cross-pair certificates, and then the certificates can be requested and installed for the CA using the Certificate Wizard for the CA subsystem.

For more information on creating cross-pair certificate profiles, see the *Configuring Cross-Pair profiles* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* .

For more information on publishing cross-pair certificates, see Section 9.9, “Publishing Cross-Pair Certificates”.

17.5.1. Installing Cross-Pair Certificates

Both cross-pair certificates can be imported into the Certificate System databases using the **certutil** tool or by selecting the **Cross-Pair Certificates** option from the Certificate Setup Wizard, as described in Section 17.6.1, “Installing Certificates in the Certificate System Database” .

When both certificates have been imported into the database, a **crossCertificatePair** entry is formed and stored in the database. The original individual cross-pair CA certificates are deleted once the **crossCertificatePair** entry is created.

17.5.2. Searching for Cross-Pair Certificates

Both CAs in bridge certificates can store or publish the cross-pair certificates as a **crossCertificatePair** entry in an LDAP database. The Certificate Manager's internal database can be searched for the **crossCertificatePair** entry with **ldapsearch**.

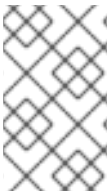
```
/usr/lib[64]/mozldap/ldapsearch -D "cn=directory manager" -w secret -p 389 -h server.example.com -
b "o=server.example.com-pki-ca" -s sub "(crossCertificatePair=*)"
```

17.6. MANAGING THE CERTIFICATE DATABASE

Each Certificate System instance has a certificate database, which is maintained in its internal token. This database contains certificates belonging to the subsystem installed in the Certificate System instance and various CA certificates the subsystems use for validating the certificates they receive.

Even if an external token is used to generate and store key pairs, Certificate System always maintains its list of trusted and untrusted CA certificates in its internal token.

This section explains how to view the contents of the certificate database, delete unwanted certificates, and change the trust settings of CA certificates installed in the database using the Certificate System window. For information on adding certificates to the database, see Section 17.6.1, “Installing Certificates in the Certificate System Database”.



NOTE

The Certificate System command-line utility **certutil** can be used to manage the certificate database by editing trust settings and adding and deleting certificates. For details about this tool, see http://www.mozilla.org/projects/security/pki/nss/tools/.

Administrators should periodically check the contents of the certificate database to make sure that it does not include any unwanted CA certificates. For example, if the database includes CA certificates that should not ever be trusted within the PKI setup, delete them.

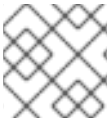
17.6.1. Installing Certificates in the Certificate System Database

If new server certificates are issued for a subsystem, they must be installed in that subsystem database. Additionally, user and agent certificates must be installed in the subsystem databases. If the certificates are issued by an external CA, then usually the corresponding CA certificate or certificate chain needs to be installed.

Certificates can be installed in the subsystem certificate database through the Console's Certificate Setup Wizard or using the **certutil** utility.

- Section 17.6.1.1, “Installing Certificates through the Console”
- Section 17.6.1.2, “Installing Certificates Using certutil”
- Section 17.6.1.3, “About CA Certificate Chains”

17.6.1.1. Installing Certificates through the Console



NOTE

pkiconsole is being deprecated.

The Certificate Setup Wizard can install or import the following certificates into either an internal or external token used by the Certificate System instance:

- Any of the certificates used by a Certificate System subsystem
- Any trusted CA certificates from external CAs or other Certificate System CAs
- Certificate chains

A certificate chain includes a collection of certificates: the subject certificate, the trusted root CA certificate, and any intermediate CA certificates needed to link the subject certificate to the trusted root. However, the certificate chain the wizard imports must include only CA certificates; none of the certificates can be a user certificate.

In a certificate chain, each certificate in the chain is encoded as a separate DER-encoded object. When the wizard imports a certificate chain, it imports these objects one after the other, all the way up the chain to the last certificate, which may or may not be the root CA certificate. If any of the certificates in the chain are already installed in the local certificate database, the wizard replaces the existing certificates with the ones in the chain. If the chain includes intermediate CA certificates, the wizard adds them to the certificate database as *untrusted* CA certificates.

The subsystem console uses the same wizard to install certificates and certificate chains. To install certificates in the local security database, do the following:

1. Open the console.

   ```
   pkiconsole https://server.example.com:secure_port/subsystem_type
   ```

2. In the **Configuration** tab, select **System Keys and Certificates** from the left navigation tree.
3. There are two tabs where certificates can be installed, depending on the subsystem type and the type of certificate.

   - The **CA Certificates** tab is for installing CA certificates and certificate chains. For Certificate Managers, this tab is used for third-party CA certificates or other Certificate System CA certificates; all of the local CA certificates are installed in the **Local Certificates** tab. For all other subsystems, all CA certificates and chains are installed through this tab.
   - The **Local Certificates** tab is where all server certificates, subsystem certificates, and local certificates such as OCSP signing or KRA transport are installed.

   Select the appropriate tab.
4. To install a certificate in the **Local Certificates** tab, click **Add/Renew**. To install a certificate in the **CA Certificates** tab, click **Add**. Both will open the Certificate Setup Wizard.

   1. When the wizard opens, select the **Install a certificate** radio button, and click **Next**.

2. Select the type of certificate to install. The options for the drop-down menu are the same options available for creating a certificate, depending on the type of subsystem, with the additional option to install a cross-pair certificate.

3. Paste in the certificate body, including the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----**, into the text area, or specify the absolute file location; this must be a local file.

   The certificate will look like the following:

   ```
   -----BEGIN CERTIFICATE-----
   MIICKzCCAZSgAwIBAgIBAzANgkqkiG9w0BAQQFADA3MQswCQYDVQQGEw
   JVUzERMA8GA1UEChMITmV0c2NhcGUxFTATBgNVBAsTDFN1cHJpeWEncy
   BDQTAeFw05NzEwMTgwMTM2MjVaFw05OTEwMTgwMTM2MjVaMEgxCzAJBg
   NVBAYTAlVTMREwDwYDVQQKEwhOZXRzY2FwZTENMAsGA1UECxMEUHawcz
   EXMBUGA1UEAxMOU3Vwcml5YSBTaGV0dHkwgZ8wDQYJKoZIhdfNAQEBBQ
   ADgY0AMIGJAoGBAMr6eZiPGfjX3uRJgEjmKiqG7SdATYzBcABu1AVyd7
   chRFOGD3wNktbf6hRo6EAmM5R1Askzf8AW7LiQZBcrXpc0k4du+2j6xJ
   u2MPm8WKuMOTuvzpo+SGXeImHVChEqooCwfdiZywyZNmgaMa2MS6pUkf
   QVAgMBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAwIAgD
   -----END CERTIFICATE-----
   ```

5. The wizard displays the certificate details. Review the fingerprint to make sure this is the correct certificate, or use the **Back** button to go back and submit a different one. Give a nickname for the certificate.

   The wizard installs the certificate.

6. Any CA that signed the certificate must be trusted by the subsystem. Make sure that this CA's certificate exists in the subsystem's certificate database (internal or external) and that it is trusted.

   If the CA certificate is not listed, add the certificate to the certificate database as a trusted CA. If the CA's certificate is listed but untrusted, change the trust setting to trusted, as shown in Section 17.7, “Changing the Trust Settings of a CA Certificate” .

   When installing a certificate issued by a CA that is not stored in the Certificate System certificate database, add that CA's certificate chain to the database. To add the CA chain to the database, copy the CA chain to a text file, start the wizard again, and install the CA chain.

17.6.1.2. Installing Certificates Using certutil

To install subsystem certificates in the Certificate System instance's security databases using **certutil**, do the following:

1. Open the subsystem's security database directory.

   ```
   cd /var/lib/pki/instance_name/alias
   ```

2. Run the **certutil** command with the **-A** to add the certificate and **-i** pointing to the file containing the certificate issued by the CA.

   ```
   certutil -A -n cert-name -t trustargs
     -d . -a -i certificate_file
   ```

NOTE

If the Certificate System instance's certificates and keys are stored on an HSM, then specify the token name using the **-h** option.

For example:

```
certutil -A -n "ServerCert cert-instance_name" -t u,u,u -d . -a -i /tmp/example.cert
```

For information about using the **certutil** command, see http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html.

17.6.1.3. About CA Certificate Chains

Any client or server software that supports certificates maintains a collection of trusted CA certificates in its certificate database. These CA certificates determine which other certificates the software can validate. In the simplest case, the software can validate only certificates issued by one of the CAs for which it has a certificate. It is also possible for a trusted CA certificate to be part of a chain of CA certificates, each issued by the CA above it in a certificate hierarchy.

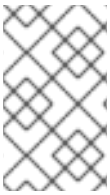
The first certificate in the chain is processed in a context-specific manner, which varies according to how it is being imported. For Mozilla Firefox, this handling depends upon the MIME content type used on the object being downloaded. For Red Hat servers, it depends upon the options selected in the server administration interface.

Subsequent certificates are all treated the same. If the certificates contain the SSL-CA bit in the Netscape Certificate Type certificate extension and do not already exist in the local certificate database, they are added as untrusted CAs. They can be used for certificate chain validation as long as there is a trusted CA somewhere in the chain.

17.6.2. Viewing Database Content

The certificates stored in the subsystem certificates database, **cert9.db**, can be viewed through the subsystem administrative console. Alternatively, the certificates can be listed using the **certutil** utility. **certutil** must be used to view the TPS certificates because the TPS subsystem does not use an administrative console.

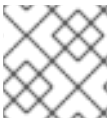
- Section 17.6.2.1, “Viewing Database Content through the Console”
- Section 17.6.2.2, “Viewing Database Content Using certutil”



NOTE

The certificates listed in the **cert9.db** database are the subsystem certificates used for subsystem operations. User certificates are stored with the user entries in the LDAP internal database.

17.6.2.1. Viewing Database Content through the Console



NOTE

pkiconsole is being deprecated.

To view the contents of the database through the administrative console, do the following:

1. Open the subsystem console.

```
pkiconsole https://server.example.com:secure_port/subsystem_type
```

2. In the **Configuration** tab, select **System Keys and Certificates** from the left navigation tree.

3. There are two tabs, **CA Certificates** and **Local Certificates**, which list different kinds of certificates.

   - **CA Certificates** lists CA certificates for which the corresponding private key material is not available, such as certificates issued by third-party CAs such as Entrust or Verisign or external Certificate System Certificate Managers.

   - **Local Certificates** lists certificates kept by the Certificate System subsystem instance, such as the KRA transport certificate or OCSP signing certificate.

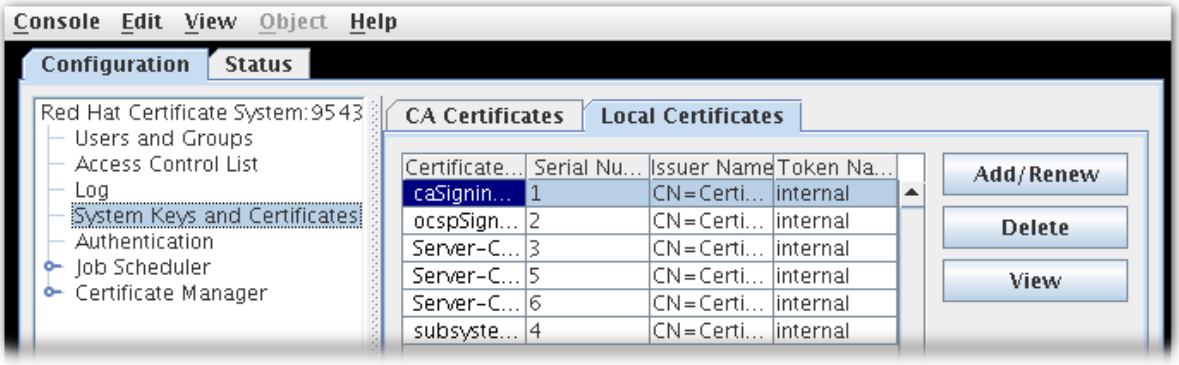


Figure 17.2. Certificate Database Tab

4. The **Certificate Database Management** table lists the all of the certificates installed on the subsystem. The following information is supplied for each certificate:

   - **Certificate Name**

   - **Serial Number**

   - **Issuer Names**, the common name (**cn**) of the issuer of this certificate.

   - **Token Name**, the name of the cryptographic token holding the certificate; for certificate stored in the database, this is **internal**.

To view more detailed information about the certificate, select the certificate, and click **View**. This opens a window which shows the serial number, validity period, subject name, issuer name, and certificate fingerprint of the certificate.

17.6.2.2. Viewing Database Content Using certutil

To view the certificates in the subsystem database using **certutil**, open the instance's certificate database directory, and run the **certutil** with the **-L** option. For example:

```
cd /var/lib/pki/instance_name/alias

certutil -L -d .
```

```
Certificate Authority - Example Domain    CT,c,
subsystemCert cert-instance name          u,u,u
Server-Cert cert-instance_name            u,u,u
```

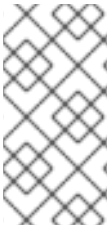
To view the keys stored in the subsystem databases using **certutil**, run the **certutil** with the **-K** option. For example:

```
cd /var/lib/pki/instance_name/alias

certutil -K -d .

Enter Password or Pin for "NSS Certificate DB":
<0> subsystemCert cert-instance_name
<1>
<2> Server-Cert cert-instance_name
```

For information about using the **certutil** command, see http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html.

17.6.3. Deleting Certificates from the Database

Removing unwanted certificates reduces the size of the certificate database.

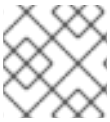


NOTE

When deleting CA certificates from the certificate database, be careful not to delete the *intermediate CA certificates*, which help a subsystem chain up to the trusted CA certificate. If in doubt, leave the certificates in the database as *untrusted* CA certificates; see Section 17.7, “Changing the Trust Settings of a CA Certificate” .

- Section 17.6.3.1, “Deleting Certificates through the Console”
- Section 17.6.3.2, “Deleting Certificates Using certutil”

17.6.3.1. Deleting Certificates through the Console



NOTE

pkiconsole is being deprecated.

To delete a certificate through the Console, do the following:

1. Open the subsystem console.

   ```
   pkiconsole https://server.example.com:secure_port/subsystem_type
   ```

2. In the **Configuration** tab, select **System Keys and Certificates** from the left navigation tree.
3. Select the certificate to delete, and click **Delete**.
4. When prompted, confirm the delete.

17.6.3.2. Deleting Certificates Using certutil

To delete a certificate from the database using **certutil**:

1. Open the instance's certificate databases directory.

```
/var/lib/pki/instance_name/alias
```

2. List the certificates in the database by running the **certutil** with the **-L** option. For example:

```
certutil -L -d .

Certificate Authority - Example Domain    CT,c,
subsystemCert cert-instance_name          u,u,u
Server-Cert cert-instance_name            u,u,u
```

3. Delete the certificate by running the **certutil** with the **-D** option.

```
certutil -D -d . -n certificate_nickname
```

For example:

```
certutil -D -d . -n "ServerCert cert-instance_name"
```

4. List the certificates again to confirm that the certificate was removed.

```
certutil -L -d .

Certificate Authority - Example Domain    CT,c,
subsystemCert cert-instance_name          u,u,u
```

For information about using the **certutil** command, see http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html.

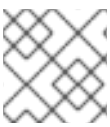
17.7. CHANGING THE TRUST SETTINGS OF A CA CERTIFICATE

Certificate System subsystems use the CA certificates in their certificate databases to validate certificates received during an SSL-enabled communication.

It can be necessary to change the trust settings on a CA stored in the certificate database, temporarily or permanently. For example, if there is a problem with access or compromised certificates, marking the CA certificate as untrusted prevents entities with certificates signed by that CA from authenticating to the Certificate System. When the problem is resolved, the CA can be marked as trusted again.

To untrust a CA permanently, consider removing its certificate from the trust database. For instructions, see Section 17.6.3, “Deleting Certificates from the Database” .

17.7.1. Changing Trust Settings through the Console



NOTE

pkiconsole is being deprecated.

To change the trust setting of a CA certificate, do the following:

1. Open the subsystem console.

```
pkiconsole https://server.example.com:secure_port/subsystem_type
```

2. In the **Configuration** tab, **System Keys and Certificates** from the left navigation tree.
3. Select the **CA certificates** tab.
4. Select the CA certificate to modify, and click **Edit**.
5. A prompt opens which reads **The Certificate chain is (un)trusted, are you sure you want to (un)trust it?**

   Clicking **yes** changes the trust setting of the certificate chain; pressing **no** preserves the original trust relationship.

17.7.2. Changing Trust Settings Using certutil

To change the trust setting of a certificate using **certutil**, do the following:

1. Open the instance's certificate databases directory.

```
cd /var/lib/pki/instance_name/alias
```

2. List the certificates in the database by running the **certutil** with the **-L** option. For example:

```
certutil -L -d .

Certificate Authority - Example Domain    CT,c,
subsystemCert cert-instance_name          u,u,u
Server-Cert cert-instance_name            u,u,u
```

3. Change the trust settings for the certificate by running the **certutil** with the **-M** option.

```
certutil -M -n cert_nickname -t trust -d .
```

   For example:

```
certutil -M -n "Certificate Authority - Example Domain" -t TCu,TCu,TCu -d .
```

4. List the certificates again to confirm that the certificate trust was changed.

```
certutil -L -d .

Certificate Authority - Example Domain    CTu,CTu,CTu
subsystemCert cert-instance_name          u,u,u
Server-Cert cert-instance_name            u,u,u
```

For information about using the **certutil** command, see http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html.

17.8. MANAGING TOKENS USED BY THE SUBSYSTEMS

Certificate System managers two groups of tokens: tokens used by the subsystems to perform PKI tasks and tokens issued through the subsystem. These management tasks refer specifically to tokens that are used by the subsystems.

For information on managing smart card tokens, see Chapter 6, *Using and Configuring the Token Management System: TPS and TKS*.

17.8.1. Detecting Tokens

To see if a token can be detected by Certificate System to be installed or configured, use the **TokenInfo** utility.

```
TokenInfo /var/lib/pki/instance_name/alias
Database Path: /var/lib/pki/instance_name/alias
Found external module 'NSS Internal PKCS #11 Module'
```

This utility will return all tokens which can be detected by the Certificate System, not only tokens which are installed in the Certificate System.

17.8.2. Viewing Tokens

To view a list of the tokens currently installed for a Certificate System instance, use the **modutil** utility.

1. Open the instance **alias** directory. For example:

   ```
   cd /var/lib/pki/instance_name/alias
   ```

2. Show the information about the installed PKCS #11 modules installed as well as information on the corresponding tokens using the **modutil** tool.

   ```
   modutil -dbdir . -nocertdb -list
   ```

17.8.3. Changing a Token's Password

The token, internal or external, that stores the key pairs and certificates for the subsystems is protected (encrypted) by a password. To decrypt the key pairs or to gain access to them, enter the token password. This password is set when the token is first accessed, usually during Certificate System installation.

It is good security practice to change the password that protects the server's keys and certificates periodically. Changing the password minimizes the risk of someone finding out the password. To change a token's password, use the **certutil** command-line utility.

For information about **certutil**, see http://www.mozilla.org/projects/security/pki/nss/tools/.

The single sign-on password cache stores token passwords in the **password.conf** file. This file must be manually updated every time the token password is changed. For more information on managing passwords through the **password.conf** file, see *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

CHAPTER 18. SETTING TIME AND DATE IN RED HAT ENTERPRISE LINUX 7

The section contains how to set time and date in Red Hat Enterprise Linux 7:

The system time is always kept in *Coordinated Universal Time* (UTC) and converted in applications to local time as needed. *Local time* is the actual time in your current time zone, taking into account *daylight saving time* (DST).

The **timedatectl** utility is distributed as part of the **systemd** system and service manager and allows you to review and change the configuration of the system clock.

CHANGING THE CURRENT TIME

```
timedatectl set-time HH:MM:SS
```

Replace *HH* with an hour, *MM* with a minute, and *SS* with a second, all typed in two-digit form.

CHANGING THE CURRENT DATE

```
timedatectl set-time YYYY-MM-DD
```

Replace *YYYY* with a four-digit year, *MM* with a two-digit month, and *DD* with a two-digit day of the month.

The time change is audited by the operating system. For more information see the *Auditing Time Change Events* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

CHAPTER 19. DETERMINING CERTIFICATE SYSTEM PRODUCT VERSION

The Red Hat Certificate System product version is stored in the **/usr/share/pki/CS_SERVER_VERSION** file. To display the version:

```
# cat /usr/share/pki/CS_SERVER_VERSION
Red Hat Certificate System 10.0 (Batch Update 1)
```

To find the product version of a running server, access the following URLs from your browser:

- **http://*host_name*:*port_number*/ca/admin/ca/getStatus**
- **http://*host_name*:*port_number*/kra/admin/kra/getStatus**
- **http://*host_name*:*port_number*/ocsp/admin/ocsp/getStatus**
- **http://*host_name*:*port_number*/tks/admin/tks/getStatus**
- **http://*host_name*:*port_number*/tps/admin/tps/getStatus**

NOTE

Note that each component is a separate package and thus could have a separate version number. The above will show the version number for each currently running component.

CHAPTER 20. UPDATING RED HAT CERTIFICATE SYSTEM

To update Certificate System and the operating system it is running on, use the **yum update** command. This downloads, verifies, and installs updates for Certificate System as well as operating system packages. For further information on updating Certificate System and validating that the update was successful, see the see the *Updating Certificate System Packages* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide*.

CHAPTER 21. TROUBLESHOOTING

This chapter covers some of the more common usage problems that are encountered when installing Certificate System.

Q: The init script returned an OK status, but my CA instance does not respond. Why?

A: This should not happen. Usually (but not always), this indicates a listener problem with the CA, but it can have many different causes. Check in the **catalina.out**, **system**, and **debug** log files for the instance to see what errors have occurred. This lists a couple of common errors.

One situation is when there is a PID for the CA, indicating the process is running, but that no listeners have been opened for the server. This would return Java invocation class errors in the **catalina.out** file:

```
Oct 29, 2010 4:15:44 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-9080
java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:615)
        at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:243)
        at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:408)
Caused by: java.lang.UnsatisfiedLinkError: jss4
```

This could mean that you have the wrong version of JSS or NSS. The process requires **libnss3.so** in the path. Check this with this command:

```
ldd /usr/lib64/libjss4.so
```

If **libnss3.so** is not found, try unsetting the **LD_LIBRARY_PATH** variable and restart the CA.

```
unset LD_LIBRARY_PATH
pki-server restart instance_name
```

Q: I can't open the pkiconsole and I'm seeing Java exceptions in stdout.

A: This probably means that you have the wrong JRE installed or the wrong JRE set as the default. Run **alternatives --config java** to see what JRE is selected. Red Hat Certificate System requires OpenJDK 1.8.

Q: I tried to run pkiconsole, and I got Socket exceptions in stdout. Why?

A: This means that there is a port problem. Either there are incorrect SSL settings for the administrative port (meaning there is bad configuration in the **server.xml**) or the wrong port was given to access the admin interface.

Port errors will look like the following:

```
NSS Cipher Supported '0xff04'
```

```
java.io.IOException: SocketException cannot read on socket
        at org.mozilla.jss.ssl.SSLSocket.read(SSLSocket.java:1006)
        at org.mozilla.jss.ssl.SSLInputStream.read(SSLInputStream.java:70)
        at
com.netscape.admin.certsrv.misc.HttpInputStream.fill(HttpInputStream.java:303)
        at
com.netscape.admin.certsrv.misc.HttpInputStream.readLine(HttpInputStream.java:224)
        at
com.netscape.admin.certsrv.connection.JSSConnection.readHeader(JSSConnection.java:439)
        at
com.netscape.admin.certsrv.connection.JSSConnection.initReadResponse(JSSConnection.java:4
30)
        at
com.netscape.admin.certsrv.connection.JSSConnection.sendRequest(JSSConnection.java:344)

        at
com.netscape.admin.certsrv.connection.AdminConnection.processRequest(AdminConnection.java
:714)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:62
3)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:59
0)
        at
com.netscape.admin.certsrv.connection.AdminConnection.authType(AdminConnection.java:323)

        at
com.netscape.admin.certsrv.CMSServerInfo.getAuthType(CMSServerInfo.java:113)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:499)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:548)
        at com.netscape.admin.certsrv.Console.main(Console.java:1655)
```

Q: I tried to enroll for a certificate, and I got the error "request is not submitted...Subject Name Not Found"?

A: This most often occurs with a custom LDAP directory authentication profile and it shows that the directory operation failed. Particularly, it failed because it could not construct a working DN. The error will be in the CA's **debug** log. For example, this profile used a custom attribute (***MYATTRIBUTE***) that the directory didn't recognize:

```
[14/Feb/2011:15:52:25][http-1244-Processor24]: BasicProfile: populate() policy
setid =userCertSet
[14/Feb/2011:15:52:25][http-1244-Processor24]: AuthTokenSubjectNameDefault:
populate start
[14/Feb/2011:15:52:25][http-1244-Processor24]: AuthTokenSubjectNameDefault:
java.io.IOException: Unknown AVA keyword 'MYATTRIBUTE'.
[14/Feb/2011:15:52:25][http-1244-Processor24]: ProfileSubmitServlet: populate
Subject Name Not Found
[14/Feb/2011:15:52:25][http-1244-Processor24]: CMSServlet: curDate=Mon Feb 14
15:52:25 PST 2011 id=caProfileSubmit time=13
```

Any custom components – attributes, object classes, and unregistered OIDs – which are used in the subject DN can cause a failure. For most cases, the X.509 attributes defined in RHC 2253 should be used in subject DNs instead of custom attributes.

Q: Why are my enrolled certificates not being published?

A: This usually indicates that the CA is misconfigured. The main place to look for errors is the **debug** log, which can indicate where the misconfiguration is. For example, this has a problem with the mappers:

```
[31/Jul/2010:11:18:29][Thread-29]: LdapSimpleMap: cert subject
dn:UID=me,E=me@example.com,CN=yes
[31/Jul/2010:11:18:29][Thread-29]: Error mapping:
mapper=com.netscape.cms.publish.mappers.LdapSimpleMap@258fdcd0 error=Cannot
find a match in the LDAP server for certificate. netscape.ldap.LDAPException:
error result (32); matchedDN = ou=people,c=test; No such object
```

Check the publishing configuration in the CA's **CS.cfg** file or in the **Publishing** tab of the CA console. In this example, the problem was in the mapping parameter, which must point to an *existing* LDAP suffix:

```
ca.publish.mapper.instance.LdapUserCertMap.dnPattern=UID=$subj.UID,dc=publish
```

Q: How do I open the pkiconsole utility from a remote host?

A: In certain situations, administrators want to open the **pkiconsole** on the Certificate System server from a remote host. For that, administrators can use a Virtual Network Computing (VNC) connection:

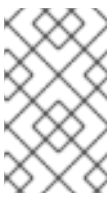
1. Setup a VNC server, for example, on the Red Hat Certificate System server. For details about remote desktop access, see the relevant section in the RHEL 8 documentation.

IMPORTANT

The **pkiconsole** utility cannot run on a server with Federal Information Processing Standard (FIPS) mode enabled. Use a different host with Red Hat Enterprise Linux to run the VNC server, if FIPS mode is enabled on your Certificate System server. Note that this utility will be deprecated.

2. Open the **pkiconsole** utility in the VNC window. For example:

```
# pkiconsole https://server.example.com:8443/ca
```



NOTE

VNC viewers are available for different kind of operating systems. However, Red Hat supports only VNC viewers installed on Red Hat Enterprise Linux from the integrated repositories.

Q: What do I do when the LDAP server is not responding?

A: If the Red Hat Directory Server instance used for the internal database is not running, a connectivity issue occurred, or a TLS connection failure occurred, then you cannot connect to the

```
[02/Apr/2019:15:55:41][authorityMonitor]: authorityMonitor: failed to get LDAPConnection.
Retrying in 1 second.
[02/Apr/2019:15:55:42][authorityMonitor]: In LdapBoundConnFactory::getConn()
[02/Apr/2019:15:55:42][authorityMonitor]: masterConn is null.
[02/Apr/2019:15:55:42][authorityMonitor]: makeConnection: errorIfDown true
[02/Apr/2019:15:55:42][authorityMonitor]: TCP Keep-Alive: true
java.net.ConnectException: Connection refused (Connection refused)
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
[02/Apr/2019:15:55:42][authorityMonitor]: Can't create master connection in
LdapBoundConnFactory::getConn!
    Could not connect to LDAP server host example911.redhat.com port 389 Error
netscape.ldap.LDAPException:
        Unable to create socket: java.net.ConnectException: Connection refused (Connection
refused) (-1)
```

After fixing the underlying network problem, such as a cable was unplugged, the Red Hat Directory Server was stopped, significant packet loss occurred, or ensuring that the TLS connection can be recreated, stop and then start the Certificate System instance in question:

```
# systemctl stop pki-tomcatd-nuxwdog@instance_name.service
```

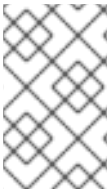
```
# systemctl start pki-tomcatd-nuxwdog@instance_name.service
```

CHAPTER 22. SUBSYSTEM CONTROL AND MAINTENANCE

This chapter provides information on how to control (start, stop, restart, and status check) a Red Hat Certificate System subsystem, as well as general maintenance (health check) recommendation.

22.1. STARTING, STOPPING, RESTARTING, AND OBTAINING STATUS

Red Hat Certificate System subsystem instances can be stopped and started using the **systemctl** utility on Red Hat Enterprise Linux 8.



NOTE

You can also use the **pki-server** alias to start and stop instances: **pki-server <command> <instance>** is an alias to **systemctl <command> pki-tomcatd@<instance>.service.**.

To start an instance:

```
# systemctl start unit_file@instance_name.service
```

```
# pki-server start instance_name
```

To stop an instance:

```
# systemctl stop unit_file@instance_name.service
```

```
# pki-server stop instance_name
```

To restart an instance:

```
# systemctl restart unit_file@instance_name.service
```

```
# pki-server restart instance_name
```

To display the status of an instance:

```
# systemctl status unit_file@instance_name.service
```

unit_file has one of the following values:

- **pki-tomcat**: With watchdog disabled
- **pki-tomcat-nuxwdog**: With watchdog enabled

22.2. SUBSYSTEM HEALTH CHECK

It is important for administrators to periodically monitor possible failures, such as the following:

- Audit failure caused by a full disk
- Signing failure caused by HSM connection issue

- LDAP server connection issues
- And so on

Self-tests can also be run by demand as described in Section 14.9, "Running Self-Tests".

22.2.1. Healthcheck in PKI

PKI Healthcheck is a command-line tool that helps find issues that may impact the health of your Certificate System environment. If needed, this tool can report to the Healthcheck tool present in Red Hat Identity Management.

22.2.1.1. PKI Healthcheck Test Modules

PKI Healthcheck consists of independent modules which test for:

- *Certificate sync between CS.cfg and NSS database*

  Checks whether the system certificates in **CS.cfg** (located in **/var/lib/pki/<instance>/<subsystem>/conf/CS.cfg**) and NSS database (located in **/var/lib/pki/<instance>/alias/**) match. Else, the Certificate Authority (CA) fails to start.

- *System certificate expiry*

  Checks the expiry status of the installed system certificates (See System Certificates for more information).

- *System certificate trust flags*

  Checks whether the installed system certificates carry the correct Trust flags (See System Certificates for more information).

- *Subsystem connectivity check*

  Checks whether a subsystem is running and able to respond to requests.

- *Subsystem clones connectivity and data check*

  Checks simple connectivity and data sanity for a set of clones configured within a given CS subsystem. A given CA subsystem's security domain is consulted to identify clones that have been set. The check then proceeds to reach out to each clone and verify data sanity where applicable.

22.2.1.2. PKI Healthcheck Configuration

The PKI Healthcheck tool configuration is stored at **/etc/pki/healthcheck.conf**. It looks like the following:

```
[global]
    plugin_timeout=300
    cert_expiration_days=30

    # Dogtag specific section
    [dogtag]
    instance_name=pki-tomcat
```

22.2.1.3. Running PKI Healthcheck

- To perform a health check, run the **pki-healthcheck** command.
- You can also execute a specific check. For example:

```
# pki-healthcheck --source pki.server.healthcheck.meta.csconfig --check
DogtagCertsConfigCheck
```

For more information on the possible options, see the man page: **man pki-healthcheck**.

22.2.1.4. Healthcheck Output Formats

Healthcheck generates the following outputs, which you can set using the **--output-type**:

- By default, machine-readable output in JSON format (**json**).
- Alternatively, human-readable output (**human**).

You can specify a alternative file destination with the **--output-file** option.

22.2.1.5. Healthcheck Results

The report consists of a message describing what was run and the status. Each Healthcheck module returns one of the following results:

SUCCESS
: configured as expected, the check executed and found no issue

WARNING
: not an error, but worth keeping an eye on or evaluating (e.g. a certificate will expire soon)

ERROR
: not configured as expected, something is wrong but your server is probably still working (e.g. a clone conflict)

CRITICAL
: not configured as expected, with a high possibility for impact (e.g. a service is not started, certificates are expired, etc.)

If the status is not successful, the message may include additional information or recommandations, which can be used by the admin to correct the issue (e.g. a file has the wrong permissions, expected X and got Y).

PART V. REFERENCES

APPENDIX A. CERTIFICATE PROFILE INPUT AND OUTPUT REFERENCE

Profile inputs and outputs define the expected input parameters in the certificate request and the output format of the enrollment result. Like many other components in Red Hat Certificate System, profile inputs and outputs are implemented as JAVA plug-ins to offer customization and flexibility. This appendix provides reference for the default input and output plug-ins.

A.1. INPUT REFERENCE

An input puts certain fields on the enrollment page associated with a particular certificate profile. The inputs set for a certificate profile are used to generate the enrollment page dynamically with the appropriate fields; these input fields collect necessary information for the profile to generate the final certificate.

A.1.1. Certificate Request Input

The Certificate Request input is used for enrollments in which a certificate request is pasted into the enrollment form. It allows the request format to be set from a drop-down list and provides an input field to paste the request.

This input puts the following fields in the enrollment form:

- **Certificate Request Type**. This drop-down menu lets the user specify the certificate request type. The choices are PKCS #10 or CRMF. Certificate Management Messages over Cryptographic Message Syntax (CMC) enrollment is supported with both PKCS #10 and CRMF.
- **Certificate Request**. This is the text area in which to paste the request.

Example A.1.

```
caAdminCert.cfg:input.i1.class_id=certReqInputImpl
```

A.1.2. CMC Certificate Request Input

The CMC Certificate Request input is used for enrollments using a Certificate Message over CMS (CMC) certificate request is submitted in the request form. The request type must be either PKCS #10 or CRMF, and the only field is the **Certificate Request** text area in which to paste the request.

Example A.2.

```
caCMCUserCert.cfg:input.i1.class_id=cmcCertReqInputImpl
```

A.1.3. Dual Key Generation Input

The Dual Key Generation input is for enrollments in which dual key pairs will be generated, and thus two certificates issued, one for signing and one for encryption.

This input puts the following fields into the enrollment form:

- **Key Generation Request Type**. This field is a read-only field displaying **crmf** as the request type.
- **Key Generation Request**. This field sets the selection for the key size in the key generation request for both encryption and signing certificates.

Example A.3.

```
caDualCert.cfg:input.i1.class_id=dualKeyGenInputImpl
```

A.1.4. File-Signing Input

The File-Signing input sets the fields to sign a file to show it has not been tampered with.

This input creates the following fields:

- **Key Generation Request Type**. This field is a read-only field displaying **crmf** as the request type.
- **Key Generation Request**. This input adds a drop-down menu to select the key size to use in the key generation request.
- **URL Of File Being Signed**. This gives the location of the file which is to be signed.
- **Text Being Signed**. This gives the filename.

Example A.4.

```
caAgentFileSigning.cfg:input.i2.class_id=fileSigningInputImpl
```

A.1.5. Image Input

The Image input sets the field to sign an image file. The only field which this input creates is **Image URL**, which gives the location of the image which is to be signed.

A.1.6. Key Generation Input

The Key Generation input is used for enrollments in which a single key pair will be generated, generally user-based certificate enrollments.

This input puts the following fields into the enrollment form:

- **Key Generation Request Type**. This field is a read-only field displaying **crmf** as the request type.
- **Key Generation Request**. This input adds a drop-down menu to select the key size to use in the key generation request.

Example A.5.

```
caDualCert.cfg:input.i1.class_id=keyGenInputImpl
```

A.1.7. nsHKeyCertRequest (Token Key) Input

The Token Key input is used to enroll keys for hardware tokens for agents to use later for certificate-based authentication.

This input puts the following fields into the enrollment form:

- **Token Key CUID**. This field gives the CUID (contextually unique user ID) for the token device.
- **Token Key User Public Key**. This field must contain the token user's public key.

Example A.6.

```
caTempTokenDeviceKeyEnrollment.cfg:input.i1.class_id=nsHKeyCertReqInputImpl
```

A.1.8. nsNKeyCertRequest (Token User Key) Input

The Token User Key input is used to enroll keys for the user of a hardware token, for agents to use the token later for certificate-based authentication. This input puts the following fields into the enrollment form:

- **Token Key User UID**. This field gives the UID for the LDAP entry of the user of the token device.
- **Token Key User Public Key**. This field must contain the token user's public key.

Example A.7.

```
caTempTokenUserEncryptionKeyEnrollment.cfg:input.i1.class_id=nsNKeyCertReqInputImpl
```

A.1.9. Serial Number Renewal Input

The Serial Number Renewal Input is used to set the serial number of an existing certificate so that the CA can pull the original certificate entry and use the information to regenerate the certificate. The input inserts a **Serial Number** field into the enrollment form.

This is the only input that needs to be used with a renewal form; all the other information is supplied by the certificate entry.

Example A.8.

```
caTokenUserEncryptionKeyRenewal.cfg:input.i1.class_id=serialNumRenewInputImpl
```

A.1.10. Subject DN Input

The Subject DN input allows the user to input the specific DN to set as the certificate subject name, and the input inserts a single **Subject Name** field into the enrollment form.

Example A.9.

```
caAdminCert.cfg:input.i3.class_id=subjectDNInputImpl
```

A.1.11. Subject Name Input

The Subject Name input is used for enrollment when DN parameters need to be collected from the user. The parameters are used to formulate the subject name in the certificate. This input puts the following fields into the enrollment form:

- **UID** (the LDAP directory user ID)
- **Email**
- **Common Name** (the name of the user)
- **Organizational Unit** (the organizational unit (***ou***) to which the user belongs)
- **Organization** (the organization name)
- **Country** (the country where the user is located)

Example A.10.

```
caDualCert.cfg:input.i2.class_id=subjectNameInputImpl
```

A.1.12. Submitter Information Input

The Submitter Information input collects the certificate requester's information such as name, email, and phone.

This input puts the following fields into the enrollment form:

- Requester Name
- Requester Email
- Requester Phone

Example A.11.

```
caAdminCert.cfg:input.i2.class_id=submitterInfoInputImpl
```

A.1.13. Generic Input

The Generic Input allows admins to specify any number of input fields to be used with extension plug-ins that handle patterns. For example, the **ccm** and **GUID** parameters are used in the patterned Subject Alternative Name Extension Default plug-in:

Example A.12.

```
input.i3.class_id=genericInputImpl
input.i3.params.gi_display_name0=ccm
input.i3.params.gi_param_enable0=true
input.i3.params.gi_param_name0=ccm
input.i3.params.gi_display_name1=GUID
input.i3.params.gi_param_enable1=true
input.i3.params.gi_param_name1=GUID
input.i3.params.gi_num=2
…
policyset.set1.p6.default.class_id=subjectAltNameExtDefaultImpl
policyset.set1.p6.default.name=Subject Alternative Name Extension Default
policyset.set1.p6.default.params.subjAltExtGNEnable_0=true
policyset.set1.p6.default.params.subjAltExtGNEnable_1=true
policyset.set1.p6.default.params.subjAltExtPattern_0=$request.ccm$
policyset.set1.p6.default.params.subjAltExtType_0=DNSName
policyset.set1.p6.default.params.subjAltExtPattern_1=
(Any)1.3.6.1.4.1.311.25.1,0410$request.GUID$
policyset.set1.p6.default.params.subjAltExtType_1=OtherName
policyset.set1.p6.default.params.subjAltNameExtCritical=false
policyset.set1.p6.default.params.subjAltNameNumGNs=2
```

A.1.14. Subject Alternative Name Extension Input

The Subject Alternative Name Extension Input is used along with the Subject Alternative Name Extension Default plug-in. It allows admins to enable the numbered parameters in URI with the pattern **req_san_pattern_#** into the input and therefore the **SubjectAltNameExt** extension. For example, URI containing:

```
...&req_san_pattern_0=host0.Example.com&req_san_pattern_1=host1.Example.com
```

injects **host0.Example.com** and **host1.Example.com** into the **SubjectAltNameExt** extension from the profile below.

Example A.13.

```
input.i3.class_id=subjectAltNameExtInputImpl
input.i3.name=subjectAltNameExtInputImpl
…
policyset.serverCertSet.9.constraint.class_id=noConstraintImpl
policyset.serverCertSet.9.constraint.name=No Constraint
policyset.serverCertSet.9.default.class_id=subjectAltNameExtDefaultImpl
policyset.serverCertSet.9.default.name=Subject Alternative Name Extension Default
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_0=true
policyset.serverCertSet.9.default.params.subjAltExtPattern_0=$request.req_san_pattern_0$
policyset.serverCertSet.9.default.params.subjAltExtType_0=DNSName
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_1=true
policyset.serverCertSet.9.default.params.subjAltExtPattern_1=$request.req_san_pattern_1$
```

```
policyset.serverCertSet.9.default.params.subjAltExtType_1=DNSName
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_2=false
policyset.serverCertSet.9.default.params.subjAltExtPattern_2=$request.req_san_pattern_2$
policyset.serverCertSet.9.default.params.subjAltExtType_2=DNSName
policyset.serverCertSet.9.default.params.subjAltNameExtCritical=false
policyset.serverCertSet.9.default.params.subjAltNameNumGNs=3
```

A.2. OUTPUT REFERENCE

An output is the response to the end user of a successful enrollment.

A.2.1. Certificate Output

This output displays the certificate in pretty-print format. This output cannot be configured or changed. It does not display anything other than the certificate in pretty-print format.

This output needs to be specified for any automated enrollment. Once a user successfully authenticates using the automated enrollment method, the certificate is automatically generated, and this output page is returned to the user. In an agent-approved enrollment, the user can get the certificate, once it is issued, by providing the request ID in the end-entities page.

Example A.14.

```
caAdminCert.cfg:output.o1.class_id=certOutputImpl
```

A.2.2. PKCS #7 Output

This output returns the certificate and the certificate chain in PKCS #7 format. PKCS #7 format is the Cryptographic Message Syntax Standard, which is used for signing. This output cannot be configured or changed.

Example A.15.

```
caAgentFileSigning.cfg:output.o1.class_id=pkcs7OutputImpl
```

A.2.3. nsNSKeyOutput

This class implements the output plug-in that returns the DER encoded certificates for token keys.

A.2.4. CMMF Output

This output returns the certificate in Certificate Management Messages Formats (CMMF). CMMF govern communication between different parts of a PKI and is used for requesting certificates and requesting certificate revocation.

APPENDIX B. DEFAULTS, CONSTRAINTS, AND EXTENSIONS FOR CERTIFICATES AND CRLS

This appendix explains both the standard certificate extensions defined by X.509 v3 and the extensions defined by Netscape that were used in versions of products released before X.509 v3 was finalized. It provides recommendations for extensions to use with specific kinds of certificates, including PKIX Part 1 recommendations.

IMPORTANT

This appendix is a reference for defaults, constraints, and certificate and CRL extensions that are used or are configurable in Red Hat Certificate System. For a complete reference and explanation of certificate and CRL extensions, see RFC 3280.

This appendix contains the following sections:

- Section B.1, "Defaults Reference"
- Section B.2, "Constraints Reference"
- Section B.3, "Standard X.509 v3 Certificate Extension Reference"
- Section B.4, "CRL Extensions"

B.1. DEFAULTS REFERENCE

Defaults are used to define the contents of a certificate. This section lists and defines the predefined defaults.

B.1.1. Authority Info Access Extension Default

This default attaches the Authority Info Access extension. This extension specifies how an application validating a certificate can access information, such as online validation services and CA policy data, about the CA that has issued the certificate. This extension should not be used to point directly to the CRL location maintained by a CA; the CRL Distribution Points extension, Section B.1.7, "CRL Distribution Points Extension Default", provides references to CRL locations.

For general information about this extension, see Section B.3.1, "authorityInfoAccess".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

This default can define up to five locations, with parameters for each location. The parameters are marked with an *n* in the table to show with which location the parameter is associated.

Table B.1. Authority Info Access Extension Default Configuration Parameters

Parameter	Description

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
Method_*n*	Specifies the access method for retrieving additional information about the CA that has issued the certificate in which the extension appears. This is one of the following values: • ocsp (1.3.6.1.5.5.7.48.1). • calssuers (1.3.6.1.5.5.7.48.2) • renewal (2.16.840.1.113730.16.1)
LocationType_*n*	Specifies the general name type for the location that contains additional information about the CA that has issued the certificate. This is one of the following types: • DirectoryName • DNSName • EDIPartyName • IPAddress • OID • RFC822Name • URIName

Parameter	Description
Location_*n*	Specifies the address or location to get additional information about the CA that has issued the certificate. • 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, 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 **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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*. • For **OID**, 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*. • For **RFC822Name**, the value must be a valid Internet mail address. • For **URIName**, the value must be a non-relative universal resource identifier (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://ocspResponder.example.com:8000*. Certificate System allows both IPv4 and IPv6 IP addresses.
Enable_*n*	Specifies whether this location is enabled. Select **true** to mark this as set; select **false** to disable it.

B.1.2. Authority Key Identifier Extension Default

This default attaches the Authority Key Identifier extension to the certificate. The extension identifies the public key that corresponds to the private key used by a CA to sign certificates. This default has no parameters. If used, this extension is included in the certificate with the public key information.

This default takes the following constraint:

- No Constraints; see Section B.2.8, "No Constraint" .

For general information about this extension, see Section B.3.2, "authorityKeyIdentifier".

B.1.3. Authentication Token Subject Name Default

This profile default populates subject names based on the attribute values in the authentication token (AuthToken) object.

This default plug-in works with the directory-based authentication manager. The Directory-Based User Dual-Use Certificate Enrollment certificate profile has two input parameters, UID and password. The directory-based authentication manager checks if the given UID and password are correct.

In addition, the directory-based authentication manager formulates the subject name of the issuing certificate. It forms the subject name by using the user's DN value from *AuthToken*.

This default is responsible for reading the subject name from the *AuthToken* and placing it in the certificate request so that the final certificate contains the subject name.

The following constraints can be defined with this default:

- No Constraints; see Section B.2.8, "No Constraint" .

B.1.4. Basic Constraints Extension Default

This default attaches the Basic Constraint extension to the certificate. The extension identifies whether the Certificate Manager is a CA. The extension is also used during the certificate chain verification process to identify CA certificates and to apply certificate chain-path length constraints.

For general information about this extension, see Section B.3.3, "basicConstraints".

The following constraints can be defined with this default:

- Basic Constraints Extension Constraint; see Section B.2.1, "Basic Constraints Extension Constraint".
- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.2. Basic Constraints Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.

Parameter	Description
IsCA	Specifies whether the certificate subject is a CA. With **true**, the server checks the **PathLen** parameter and sets the specified path length in the certificate. With **false**, the server treats the certificate subject as a non-CA and ignores the value specified for the **PathLen** parameter.
PathLen	Specifies the path length, the maximum number of CA certificates that may be chained below (subordinate to) the subordinate CA certificate being issued. The path length affects the number of CA certificates to be used during certificate validation. The chain starts with the end-entity certificate being validated and moves up. The **maxPathLen** parameter has no effect if the extension is set in end-entity certificates. The permissible values are **0** or *n*. The value should be less than the path length specified in the Basic Constraints extension of the CA signing certificate. **0** specifies that no subordinate CA certificates are allowed below the subordinate CA certificate; only an end-entity certificate may follow in the path. *n* must be an integer greater than zero. It specifies the maximum number of subordinate CA certificates allowed below the subordinate CA certificate. If the field is blank, the path length defaults to a value that is determined by the path length set in the Basic Constraints extension in the issuer's certificate. If the issuer's path length is unlimited, the path length in the subordinate CA certificate will also be unlimited. If the issuer's path length is an integer greater than zero, the path length in the subordinate CA certificate will be set to a value that is one less than the issuer's path length; for example, if the issuer's path length is 4, the path length in the subordinate CA certificate will be set to 3.

B.1.5. CA Validity Default

This default adds an option to a CA certificate enrollment or renewal profile to bypass the CA's signing certificate's expiration constraint. This means that the issued CA certificate can have an expiration date that is later than the issuing CA signing certificate expiration date.

The following constraints can be defined with this default:

- Validity Constraint; see Section B.2.14, “Validity Constraint”.
- No Constraints; see Section B.2.8, “No Constraint”.

Table B.3. CA Validity Default Parameters

Parameter	Description
bypassCAnotafterrange	Sets the default value for whether a requesting CA can request a certificate whose validity period extends past the issuing CA's validity period.
range	Specifies the absolute validity period for this certificate, in the number of days.
startTime	Sets when the validity period begins, based on the current time.

B.1.6. Certificate Policies Extension Default

This default attaches the Certificate Policy Mappings extension into the certificate template. This extension defines one or more policies, indicating the policy under which the certificate has been issued and the purposes for which the certificate may be used. This default defines up to five policies, but this can be value can be changed.

For general information about this extension, see Section B.3.4, "certificatePoliciesExt"

Table B.4. Certificate Policies Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
numCertPolicies	Specifies the number of policies that can be defined. The default is **5**.
enable	Select **true** to enable the policy; select **false** to disable the policy.
policyId	Specifies the OID identifier for the policy.
cpsURI.enable	The extension can include a URI to the issuer's Certificate Practice Statement. Select **true** to enable URI; select **false** to disable URI.
CPSURI.value	This value is a pointer to a Certification Practice Statement (CPS) published by the CA. The pointer is in the form of a URI.
usernotice.enable	The extension can include a URI to the issuer's Certificate Practice Statement or can embed issuer information, such as a user notice in text form. Select **true** to enable user notices; select **false** to disable the user notices.

Parameter	Description
usernotice.noticeReference.noticeNumbers	This optional user notice parameter is a sequence of numbers that points to messages stored elsewhere.
usernotice.noticeReference.organization	This optional user notice parameter specifies the name of the company.
usernotice.explicitText.value	This optional user notice parameter contains the message within the certificate.

B.1.7. CRL Distribution Points Extension Default

This default attaches the CRL Distribution Points extension to the certificate. This extension identifies locations from which an application that is validating the certificate can obtain the CRL information to verify the revocation status of the certificate.

For general information about this extension, see Section B.3.5, "CRLDistributionPoints".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

This default defines up to five locations, with parameters for each location. The parameters are marked with an *n* in the table to show with which location the parameter is associated.

Table B.5. CRL Distribution Points Extension Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
Type_*n*	Specifies the type of CRL distribution point. The permissible values are **DirectoryName**, **URIName**, or **RelativeToIssuer**. The type must correspond to the value in the **Name** field.

Parameter	Description
Name_*n*	Specifies the name of the CRL distribution point, the name can be in any of the following formats: • An X.500 directory name in the RFC 2253 syntax. The name looks similar to the subject name in a certificate, like *cn=CA Central, ou=Research Dept, o=Example Corporation, c=US*. • A URIName, such as *http://testCA.example.com:80*. • An RDN which specifies a location relative to the CRL issuer. In this case, the value of the **Type** attribute must be **RelativeToIssuer**.
Reasons_*n*	Specifies revocation reasons covered by the CRL maintained at the distribution point. Provide a comma-separated list of the following constants: • unused • keyCompromise • cACompromise • affiliationChanged • superseded • cessationOfOperation • certificateHold
IssuerType_*n*	Specifies the naming type of the issuer that has signed the CRL maintained at the distribution point. The issuer name can be in any of the following formats: • RFC822Name • DirectoryName • DNSName • EDIPartyName • URIName • IPAddress • OIDName • OtherName
IssuerName_*n*	

Parameter	Description
	Specifies the name format of the CRL issuer that CRL. The permissible values are as follows: • 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, 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*. Certificate System supports both IPv4 and IPv6 addresses. • 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, 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**. **KerberosName** has the format *Realm|NameType|NameStrings*, such as **realm1|0|userID1,userID2**. **OtherName** must have the format *(type)oid,string*. For example, **(IA5String)1.2.3.4,MyExample**. The value for this parameter must correspond to the value in the **issuerName** field.

Parameter	Description

B.1.8. Extended Key Usage Extension Default

This default attaches the Extended Key Usage extension to the certificate.

For general information about this extension, see Section B.3.6, "extKeyUsage".

The extension identifies the purposes, in addition to the basic purposes indicated in the Key Usage extension, for which the certified public key may be used. For example, if the key usage extension identifies a signing key, the Extended Key Usage extension can narrow the usage of the key for only signing OCSP responses or only Java™ applets.

Table B.6. PKIX Usage Definitions for the Extended Key Usage Extension

Usage	OID
Server authentication	1.3.6.1.5.5.7.3.1
Client authentication	1.3.6.1.5.5.7.3.2
Code signing	1.3.6.1.5.5.7.3.3
Email	1.3.6.1.5.5.7.3.4
IPsec end system	1.3.6.1.5.5.7.3.5
IPsec tunnel	1.3.6.1.5.5.7.3.6
IPsec user	1.3.6.1.5.5.7.3.7
Timestamping	1.3.6.1.5.5.7.3.8

Windows 2000 can encrypt files on the hard disk, a feature known as encrypted file system (EFS), using certificates that contain the Extended Key Usage extension with the following two OIDs:

1.3.6.1.4.1.311.10.3.4 (EFS certificate)

1.3.6.1.4.1.311.10.3.4.1 (EFS recovery certificate)

The EFS recovery certificate is used by a recovery agent when a user loses the private key and the data encrypted with that key needs to be used. Certificate System supports these two OIDs and allows certificates to be issued containing the Extended Key Usage extension with these OIDs.

Normal user certificates should be created with only the EFS OID, not the recovery OID.

The following constraints can be defined with this default:

- Extended Key Usage Constraint; see Section B.2.3, "Extended Key Usage Extension Constraint".
- Extension Constraint; see Section B.2.4, "Extension Constraint" .

- No Constraints; see Section B.2.8, "No Constraint".

Table B.7. Extended Key Usage Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
OIDs	Specifies the OID that identifies a key-usage purpose. The permissible values are a unique, valid OID specified in the dot-separated numeric component notation. For example, *2.16.840.1.113730.1.99*. Depending on the key-usage purposes, the OIDs can be designated by PKIX (listed in Table B.6, "PKIX Usage Definitions for the Extended Key Usage Extension") or custom OIDs. Custom OIDs must be in the registered subtree of IDs reserved for the company's use. Although it is possible to use custom OIDs for evaluating and testing the Certificate System, in a production environment, comply with the ISO rules for defining OIDs and for registering subtrees of IDs.

B.1.9. Freshest CRL Extension Default

This default attaches the Freshest CRL extension to the certificate.

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

This default defines five locations with parameters for each location. The parameters are marked with an *n* in the table to show with which location the parameter is associated.

Table B.8. Freshest CRL Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
PointEnable_*n*	Select **true** to enable this point; select **false** to disable this point.
PointType_*n*	Specifies the type of issuing point, either **DirectoryName** or **URIName**.

Parameter	Description
PointName_*n*	• If **pointType** is set to **directoryName**, the value must be an X.500 name, similar to the subject name in a certificate. For example, *cn=CACentral,ou=Research Dept,o=Example Corporation,c=US*. • If **pointType** is set to **URIName**, the name must be a URI, an absolute pathname that specifies the host. For example, *http://testCA.example.com/get/crls/here/*.
PointIssuerName_*n*	Specifies the name of the issuer that has signed the CRL. The name can be in any of the following formats: • 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, 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*. Certificate System supports both IPv4 and IPv6 addresses. • 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, 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,

Parameter	Description
	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**. **KerberosName** has the format *Realm|NameType|NameStrings*, such as **realm1|0|userID1,userID2**. **OtherName** must have the format *(type)oid,string*. For example, **(IA5String)1.2.3.4,MyExample**. The name value must comply with the format specified in **PointType_**.
PointType_*n*	Specifies the general name type of the CRL issuer that signed the CRL. The permissible values are as follows: • RFC822Name • DirectoryName • DNSName • EDIPartyName • URIName • IPAddress • OIDName • OtherName The value for this parameter must correspond to the value in the **PointIssuerName** field.

B.1.10. Generic Extension Default

This extension allows for the creation of a generic extension with user determined data. The default ensures the generic extension is populated correctly.

Table B.9. Generic Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
genericExtOID	Specifies the extensions OID identifier.
genericExtData	The binary data contained within the extension.

B.1.11. Inhibit Any-Policy Extension Default

The inhibit any-policy extension can be used for certificates issued to CAs. The inhibit any-policy indicates that the special anyPolicy OID, with the value { 2 5 29 32 0 }, is not considered an explicit match for other certificate policies.

Table B.10. Inhibit Any-Policy Extension Default Configuration Parameters

Parameter	Description
Critical	*This policy must be marked as critical.* Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
SkipCerts	This parameter indicate the number of additional certificates that may appear in the path before any-policy is no longer allowed. A value of **1** indicates that any-policy may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.

B.1.12. Issuer Alternative Name Extension Default

This default attaches the Issuer Alternative Name extension to the certificate. The Issuer Alternative Name extension is used to associate Internet-style identities with the certificate issuer.

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

This default defines five locations with parameters for each location. The parameters are marked with an *n* in the table to show with which location the parameter is associated.

Table B.11. Issuer Alternative Name Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.

Parameter	Description
issuerAltExtType	This sets the type of name extension to be used, which can be one of the following: • RFC822Name • DirectoryName • DNSName • EDIPartyName • URIName • IPAddress • OIDName
issuerAltExtPattern	Specifies the request attribute value to include in the extension. The attribute value must conform to any of the supported general name types. The permissible value is a request attribute included in the certificate request. If the server finds the attribute in the request, it sets the attribute value in the extension and adds the extension to certificates. If multiple attributes are specified and none of the attributes are present in the request, the server does not add the Issuer Alternative Name extension to certificates. If no suitable attributes can be used from the request to form the issuerAlternativeName, then literal string can be used without any token expression. For example, *Certificate Authority*.

B.1.13. Key Usage Extension Default

This default attaches the Key Usage extension to the certificate. The extension specifies the purposes for which the key contained in a certificate should be used, such as data signing, key encryption, or data encryption, which restricts the usage of a key pair to predetermined purposes.

For general information about this extension, see Section B.3.8, "keyUsage".

The following constraints can be defined with this default:

- Key Usage Constraint; see Section B.2.6, "Key Usage Extension Constraint".
- Extension Constraint; see Section B.2.4, "Extension Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

Table B.12. Key Usage Extension Default Configuration Parameters

Parameter	Description

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
digitalSignature	Specifies whether to allow signing SSL client certificates and S/MIME signing certificates. Select **true** to set.
nonRepudiation	Specifies whether to use for S/MIME signing certificates. Select **true** to set. **WARNING** Using this bit is controversial. Carefully consider the legal consequences of its use before setting it for any certificate.
keyEncipherment	Specifies whether the public key in the subject is used to encipher private or secret keys. This is set for SSL server certificates and S/MIME encryption certificates. Select **true** to set.
dataEncipherment	Specifies whether to set the extension when the subject's public key is used to encipher user data as opposed to key material. Select **true** to set.
keyAgreement	Specifies whether to set the extension whenever the subject's public key is used for key agreement. Select **true** to set.
keyCertsign	Specifies whether the public key is used to verify the signature of other certificates. This setting is used for CA certificates. Select **true** to set the option.
cRLSign	Specifies whether to set the extension for CA signing certificates that sign CRLs. Select **true** to set.
encipherOnly	Specifies whether to set the extension if the public key is only for encrypting data while performing key agreement. If this bit is set, **keyAgreement** should also be set. Select **true** to set.

Parameter	Description
decipherOnly	Specifies whether to set the extension if the public key is only for decrypting data while performing key agreement. If this bit is set, **keyAgreement** should also be set. Select **true** to set.

B.1.14. Name Constraints Extension Default

This default attaches a Name Constraints extension to the certificate. The extension is used in CA certificates to indicate a name space within which the subject names or subject alternative names in subsequent certificates in a certificate chain should be located.

For general information about this extension, see Section B.3.9, "nameConstraints".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

This default defines up to five locations for both the permitted subtree and the excluded subtree and sets parameters for each location. The parameters are marked with an *n* in the table to show with which location the parameter is associated.

Table B.13. Name Constraints Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
PermittedSubtrees*n*.min	Specifies the minimum number of permitted subtrees. • **-1** specifies that the field should not be set in the extension. • **0** specifies that the minimum number of subtrees is zero. • *n* must be an integer that is greater than zero. It sets the minimum required number of subtrees.

Parameter	Description
PermittedSubtreesmax_*n*	Specifies the maximum number of permitted subtrees. • **-1** specifies that the field should not be set in the extension. • **0** specifies that the maximum number of subtrees is zero. • *n* must be an integer that is greater than zero. It sets the maximum number of subtrees allowed.
PermittedSubtreeNameChoice_*n*	Specifies the general name type for the permitted subtree to include in the extension. The permissible values are as follows: • RFC822Name • DirectoryName • DNSName • EDIPartyName • URIName • IPAddress • OIDName • OtherName

Parameter	Description
PermittedSubtreeNameValue_*n*	Specifies the general name value for the permitted subtree to include in the extension. • 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, 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*. Certificate System supports both IPv4 and IPv6 addresses. • For **IPAddress**, the value must be a valid IP address conforming to Classless Inter-Domain Routing (CIDR) notation. An IPv4 address must be in the *n.n.n.n* format, or *n.n.n.n/m* with a netmask - for example, *10.34.3.133* or *110.34.3.133/24*. IPv6 addresses must also conform to CIDR notation; examples with netmasks include *2620:52:0:2203:527b:9dff:fe56:4495/64* or *2001:db8::/64*. • 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**. **KerberosName** has the format *Realm|NameType|NameStrings*, such as **realm1|0|userID1,userID2**. **OtherName** must have the format *(type)oid,string*. For example, **(IA5String)1.2.3.4,MyExample**.
PermittedSubtreeEnable_*n*	Select **true** to enable this permitted subtree entry.

Parameter	Description
ExcludedSubtrees*n*.min	Specifies the minimum number of excluded subtrees. • **-1** specifies that the field should not be set in the extension. • **0** specifies that the minimum number of subtrees is zero. • *n* must be an integer that is greater than zero. This sets the minimum number of required subtrees.
ExcludedSubtreeMax_*n*	Specifies the maximum number of excluded subtrees. • **-1** specifies that the field should not be set in the extension. • **0** specifies that the maximum number of subtrees is zero. • *n* must be an integer that is greater than zero. This sets the maximum number of allowed subtrees.
ExcludedSubtreeNameChoice_*n*	Specifies the general name type for the excluded subtree to include in the extension. The permissible values are as follows: • RFC822Name • DirectoryName • DNSName • EDIPartyName • URIName • IPAddress • OIDName • OtherName

Parameter	Description
ExcludedSubtreeNameValue_*n*	Specifies the general name value for the permitted subtree to include in the extension. • For **RFC822Name**, the value must be a valid Internet mail address. For example, *testCA@example.com*. • For **DirectoryName**, the value must be an X.500 name, similar to the subject name in a certificate. For example, *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*. Certificate System supports both IPv4 and IPv6 addresses. • For **IPAddress**, the value must be a valid IP address conforming to Classless Inter-Domain Routing (CIDR) notation. An IPv4 address must be in the *n.n.n.n* format, or *n.n.n.n/m* with a netmask - for example, *10.34.3.133* or *110.34.3.133/24*. IPv6 addresses must also conform to CIDR notation; examples with netmasks include *2620:52:0:2203:527b:9dff:fe56:4495/64* or *2001:db8::/64*. • 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*. • For **OtherName**, the values are names with any other format. This supports **PrintableString**, **IA5String**, **UTF8String**, **BMPString**, **Any**, and **KerberosName**. **KerberosName** has the format *Realm|NameType|NameStrings*, such as **realm1|0|userID1,userID2**. **OtherName** must have the format *(type)oid,string*. For example, **(IA5String)1.2.3.4,MyExample**.
ExcludedSubtreeEnable_*n*	Select **true** to enable this excluded subtree entry.

B.1.15. Netscape Certificate Type Extension Default

WARNING

This extension is obsolete. Use the Key Usage or Extended Key Usage certificate extensions instead.

This default attaches a Netscape Certificate Type extension to the certificate. The extension identifies the certificate type, such as CA certificate, server SSL certificate, client SSL certificate, or S/MIME certificate. This restricts the usage of a certificate to predetermined purposes.

B.1.16. Netscape Comment Extension Default

WARNING

This extension is obsolete.

This default attaches a Netscape Comment extension to the certificate. The extension can be used to include textual comments in certificates. Applications that are capable of interpreting the comment display it when the certificate is used or viewed.

For general information about this extension, see Section B.4.3.2, "netscape-comment".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.14. Netscape Comment Extension Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
CommentContent	Specifies the content of the comment to appear in the certificate.

B.1.17. No Default Extension

This default can be used to set constraints when no defaults are being used. This default has no settings and sets no defaults but does allow all of the constraints available to be set.

B.1.18. OCSP No Check Extension Default

This default attaches an OCSP No Check extension to the certificate. The extension, which should be used in OCSP responder certificates only, indicates how OCSP-compliant applications can verify the revocation status of the certificate an authorized OCSP responder uses to sign OCSP responses.

For general information about this extension, see Section B.3.10, "OCSPNocheck".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

Table B.15. OCSP No Check Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.

B.1.19. Policy Constraints Extension Default

This default attaches a Policy Constraints extension to the certificate. The extension, which can be used in CA certificates only, constrains path validation in two ways: either to prohibit policy mapping or to require that each certificate in a path contain an acceptable policy identifier. The default can specify both **ReqExplicitPolicy** and **InhibitPolicyMapping**. PKIX standard requires that, if present in the certificate, the extension must never consist of a null sequence. At least one of the two specified fields must be present.

For general information about this extension, see Section B.3.11, "policyConstraints".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

Table B.16. Policy Constraints Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.

Parameter	Description
reqExplicitPolicy	Specifies the total number of certificates permitted in the path before an explicit policy is required. This is the number of CA certificates that can be chained below the subordinate CA certificate before an acceptable policy is required. • **-1** specifies that the field should not be set in the extension. • **0** specifies that no subordinate CA certificates are permitted in the path before an explicit policy is required. • *n* must be an integer that is greater than zero. It specifies the maximum number of subordinate CA certificates allowed in the path before an explicit policy is required. This number affects the number of CA certificates to be used during certificate validation. The chain starts with the end-entity certificate being validated and moving up the chain. The parameter has no effect if the extension is set in end-entity certificates.
inhibitPolicyMapping	Specifies the total number of certificates permitted in the path before policy mapping is no longer permitted. • **-1** specifies that the field should not be set in the extension. • **0** specifies that no subordinate CA certificates are permitted in the path before policy mapping is no longer permitted. • *n* must be an integer that is greater than zero. It specifies at the maximum number of subordinate CA certificates allowed in the path before policy mapping is no longer permitted. For example, a value of 1 indicates that policy mapping may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.

B.1.20. Policy Mappers Extension Default

This default attaches a Policy Mappings extension to the certificate. The extension lists pairs of OIDs, each pair identifying two policy statements of two CAs. The pairing indicates that the corresponding policies of one CA are equivalent to policies of another CA. The extension may be useful in the context of cross-certification. If supported, the extension is included in CA certificates only. The default maps policy statements of one CA to that of another by pairing the OIDs assigned to their policy statements

Each pair is defined by two parameters, **issuerDomainPolicy** and **subjectDomainPolicy**. The pairing indicates that the issuing CA considers the **issuerDomainPolicy** equivalent to the

subjectDomainPolicy of the subject CA. The issuing CA's users may accept an **issuerDomainPolicy** for certain applications. The policy mapping tells these users which policies associated with the subject CA are equivalent to the policy they accept.

For general information about this extension, see Section B.3.12, "policyMappings".

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

Table B.17. Policy Mappings Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
IssuerDomainPolicy_*n*	Specifies the OID assigned to the policy statement of the issuing CA to map with the policy statement of another CA. For example, *1.2.3.4.5*.
SubjectDomainPolicy_*n*	Specifies the OID assigned to the policy statement of the subject CA that corresponds to the policy statement of the issuing CA. For example, *6.7.8.9.10*.

B.1.21. Private Key Usage Period Extension Default

The Private Key Usage Period extension allows the certificate issuer to specify a different validity period for the private key than for the certificate itself. This extension is intended for use with digital signature keys.

Table B.18. Private key Usage Period Configuration Parameters

Parameter	Description
Critical	This extension should always be non-critical.
puStartTime	This parameters sets the start time. The default value is **0**, which starts the validity period from the time the extension is activated.
puDurationDays	This parameters sets the duration of the usage period. The default value is **365**, which sets the validity period to 365 days from the time the extension is activated.

B.1.22. Signing Algorithm Default

This default attaches a signing algorithm in the certificate request. This default presents an agent with the possible algorithms that can be used for signing the certificate.

The following constraints can be defined with this default:

- Signing Algorithm Constraint; see Section B.2.10, "Signing Algorithm Constraint".
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.19. Signing Algorithm Default Configuration Parameters

Parameter	Description
signingAlg	Specify the default signing algorithm to be used to create this certificate. An agent can override this value by specifying one of the values contained in the **signingAlgsAllowed** parameter.
signingAlgsAllowed	Specify the signing algorithms that can be used for signing this certificate. The algorithms can be any or all of the following: • MD2withRSA • MD5withRSA • SHA256withRSA • SHA512withRSA

B.1.23. Subject Alternative Name Extension Default

This default attaches a Subject Alternative Name extension to the certificate. The extension binds additional identities, such as an email address, a DNS name, an IP address (both IPv4 and IPv6), or a URI, to the subject of the certificate. The standard requires that if the certificate subject field contains an empty sequence, then the Subject Alternative name extension must contain the subject's alternative name and that the extension be marked critical.

For any of the directory-based authentication methods, the Certificate System can retrieve values for any string and byte attributes and set them in the certificate request. These attributes are set by entering them in the **ldapStringAttributes** and **ldapByteAttributes** fields defined in the automated enrollment modules.

If authenticated attributes – meaning attributes stored in an LDAP database – need to be part of this extension, use values from the **$request.*X*$** token.

There is an additional attribute to insert a universally unique identifier (UUID) into the subject alt name. This option generates a random number for version 4 UUID; the pattern is defined by referencing the server which will generate the number in an additional **subjAltExtSource** parameter.

A basic Subject Alternative Name Extension default is configured in the example.

Example B.1. Default Subject Alternative Name Extension Configuration

```
policyset.serverCertSet.9.constraint.name=No Constraint
policyset.serverCertSet.9.default.class_id=subjectAltNameExtDefaultImpl
policyset.serverCertSet.9.default.name=Subject Alternative Name Extension Default
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_0=true
```

```
policyset.serverCertSet.9.default.params.subjAltExtPattern_0=$request.requestor_email$
policyset.serverCertSet.9.default.params.subjAltExtType_0=RFC822Name
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_1=true
policyset.serverCertSet.9.default.params.subjAltExtPattern_1=$request.SAN1$
policyset.serverCertSet.9.default.params.subjAltExtType_1=DNSName
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_2=true
policyset.serverCertSet.9.default.params.subjAltExtPattern_2=http://www.server.example.com
policyset.serverCertSet.9.default.params.subjAltExtType_2=URIName
policyset.serverCertSet.9.default.params.subjAltExtType_3=OtherName
policyset.serverCertSet.9.default.params.subjAltExtPattern_3=(IA5String)1.2.3.4,$server.source$
policyset.serverCertSet.9.default.params.subjAltExtSource_3=UUID4
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_3=true
policyset.serverCertSet.9.default.params.subjAltExtType_4=RFC822Name
policyset.serverCertSet.9.default.params.subjAltExtGNEnable_4=false
policyset.serverCertSet.9.default.params.subjAltExtPattern_4=
policyset.serverCertSet.9.default.params.subjAltNameExtCritical=false
policyset.serverCertSet.9.default.params.subjAltNameNumGNs=5
```

The Subject Alternative Name extension default checks the certificate request for the profile attributes. If the request contains an attribute, the profile reads its value and sets it in the extension. It is also possible for the Subject Alternative Name extension default to insert attribute values from an LDAP directory, if LDAP-based authentication is configured. The extension added to the certificates contain all the configured attributes.

The variables that can be used with the Subject Alternative Name extension default are listed in Table B.20, “Variables to Insert Values in the Subject Alternative Name” .

Table B.20. Variables to Insert Values in the Subject Alternative Name

Policy Set Token	Description
$request.auth_token.cn$	The LDAP common name (***cn***) attribute of the user who requested the certificate.
$request.auth_token.mail$	The value of the LDAP email (***mail***) attribute of the user who requested the certificate.
$request.auth_token.tokenCertSubject$	The certificate subject name.
$request.auth_token.uid$	The LDAP user ID (***uid***) attribute of the user who requested the certificate.
$request.auth_token.user$	
$request.auth_token.userDN$	The user DN of the user who requested the certificate.
$request.auth_token.userid$	The value of the user ID attribute for the user who requested the certificate.

Policy Set Token	Description
$request.uid$	The value of the user ID attribute for the user who requested the certificate.
$request.profileRemoteAddr$	The IP address of the user making the request. This can be an IPv4 or an IPv6 address, depending on the client. 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*.
$request.profileRemoteHost$	The hostname or IP address of the user's machine. The hostname can be the fully-qualified domain name and the protocol, such as **http://server.example.com**. 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*.
$request.requestor_email$	The email address of the person who submitted the request.
$request.requestowner$	The person who submitted the request.
$request.subject$	The subject name DN of the entity to which the certificate is issued. For example, *uid=jsmith, e=jsmith@example.com*.
$request.tokencuid$	The card unique ID (CUID) of the smart card token used for requesting the enrollment.
$request.upn$	The Microsoft UPN. This has the format *(UTF8String)1.3.6.1.4.1.311.20.2.3,$request.upn$*.
$server.source$	Instructs the server to generate a version 4 UUID (random number) component in the subject name. This always has the format *(IA5String)1.2.3.4,$server.source$*.

Multiple attributes can be set for a single extension. The **subjAltNameNumGNs** parameter controls how many of the listed attributes are required to be added to the certificate. This parameter must be added to custom profiles and may need to be modified in default profiles to include as many attributes as required. In Example B.1, "Default Subject Alternative Name Extension Configuration" , the **subjAltNameNumGNs** is set to **5** to insert the **RFC822Name**, **DNSName**, **URIName**, **OtherName**, and **RFC822Name** names (generic names **_0**, **_1**, **_2**, **_3**, and **_4**).

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.21. Subject Alternative Name Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
Pattern	Specifies the request attribute value to include in the extension. The attribute value must conform to any of the supported general name types. If the server finds the attribute in the request, it sets the attribute value in the extension and adds the extension to certificates. If multiple attributes are specified and none of the attributes are present in the request, the server does not add the Subject Alternative Name extension to certificates. The permissible value is a request attribute included in the certificate request. For example, *$request.requestor_email$*.
Type	Specifies the general name type for the request attribute. • Select **RFC822Name** if the request-attribute value is an email address in the **local-part@domain** format. For example, *jdoe@example.com* • Select **DirectoryName** if the request-attribute value is an X.500 directory name, similar to the subject name in a certificate. For example, *cn=Jane Doe, ou=Sales Dept, o=Example Corporation, c=US*. • Select **DNSName** if the request-attribute value is a DNS name. For example, *corpDirectory.example.com*. • Select **EDIPartyName** if the request-attribute value is an EDI party name. For example, *Example Corporation*. • Select **URIName** if the request-attribute value is a non-relative URI that includes both a scheme, such as **http**, and a fully qualified domain name or IP address of the

Parameter	Description
	host. For example, *http://hr.example.com*. rtificate System supports both IPv4 and IPv6 addresses. • Select **IPAddress** if the request-attribute value is a valid IP address specified in dot-separated numeric component notation. For example, *128.21.39.40*. 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*. • Select **OIDName** if the request-attribute value is a unique, valid OID specified in the dot-separated numeric component notation. For example, *1.2.3.4.55.6.5.99*. • Select **OtherName** for names with any other format. This supports **PrintableString**, **IA5String**, **UTF8String**, **BMPString**, **Any**, and **KerberosName**. **KerberosName** has the format *Realm|NameType|NameStrings*, such as **realm1|0|userID1,userID2**. **OtherName** must have the format *(type)oid,string*. For example, **(IA5String)1.2.3.4,MyExample**.
Source	Specifies an identification source or protocol to use to generate an ID. The only supported source is UUID4, which generates a random number to create the UUID.
Number of Components (NumGNs)	Specifies the number of name components that must be included in the subject alternative name.

B.1.24. Subject Directory Attributes Extension Default

This default attaches a Subject Directory Attributes extension to the certificate. The Subject Directory Attributes extension conveys any desired directory attribute values for the subject of the certificate.

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.22. Subject Directory Attributes Extension Default Configuration Parameters

Parameter	Description
Critical	Select **true** to mark this extension critical; select **false** to mark the extension noncritical.
Name	The attribute name; this can be any LDAP directory attribute, such as ***cn*** or ***mail***.
Pattern	Specifies the request attribute value to include in the extension. The attribute value must conform to the allowed values of the attribute. If the server finds the attribute, it sets the attribute value in the extension and adds the extension to certificates. If multiple attributes are specified and none of the attributes are present in the request, the server does not add the Subject Directory Attributes extension to certificates. For example, *$request.requestor_email$*.
Enable	Sets whether that attribute is able to be added to the certificate. Select **true** to enable the attribute.

B.1.25. Subject Info Access Extension Default

Implements an enrollment default policy that populates a Subject Information Access extension in the certificate template. This extension indicates how to access information and services for the subject of the certificate in which the extension appears.

Parameter	Description
Critical	This extension is supposed to be non-critical.
subjInfoAccessNumADs	The number of information access sections included with the certificate.
subjInfoAccessADMethod_n	OID of the access method.
subjInfoAccessADMethod_n	Type of access method. • URIName • Directory name • DNS Name • EID Party Name • IP Address • OID Name • RFC822Name

Parameter	Description
subjInfoAccessADLocation_n	Location based on the type subjInfoAccessADMethod_n i.e., a URL for URI Name.
subjInfoAccessADEnable_n	Select **true** to enable this extension; select **false** to disable this extension.

B.1.26. Subject Key Identifier Extension Default

This default attaches a Subject Key Identifier extension to the certificate. The extension identifies certificates that contain a particular public key, which identifies a certificate from among several that have the same subject name.

For general information about this extension, see Section B.3.16, "subjectKeyIdentifier".

If enabled, the profile adds a Subject Key Identifier Extension to an enrollment request if the extension does not already exist. If the extension exists in the request, such as a CRMF request, the default replaces the extension. After an agent approves the manual enrollment request, the profile accepts any Subject Key Identifier Extension that is already there.

This default has no parameters. If used, this extension is included in the certificate with the public key information.

The following constraints can be defined with this default:

- Extension Constraint; see Section B.2.4, "Extension Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

B.1.27. Subject Name Default

This default attaches a server-side configurable subject name to the certificate request. A static subject name is used as the subject name in the certificate.

The following constraints can be defined with this default:

- Subject Name Constraint; see Section B.2.11, "Subject Name Constraint" .
- Unique Subject Name Constraint; see Section B.2.13, "Unique Subject Name Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

Table B.23. Subject Name Default Configuration Parameters

Parameter	Description
Name	Specify the subject name for this certificate.

If you need to get a certificate subject name that uses the DNPATTERN value from the UidPwdDirAuth plugin, then configure the profile to use the Subject Name Default plugin and substitute the ***Name*** parameter with the "Subject Name" from the *AuthToken* as shown below.

```
policyset.userCertSet.1.default.class_id=subjectNameDefaultImpl
policyset.userCertSet.1.default.name=Subject Name Default
policyset.userCertSet.1.default.params.name=$request.auth_token.tokenCertSubject$
```

B.1.28. User Key Default

This default attaches a user-supplied key into the certificate request. This is a required default. Keys are part of the enrollment request.

The following constraints can be defined with this default:

- Key Constraint; see Section B.2.5, "Key Constraint".
- No Constraints; see Section B.2.8, "No Constraint" .

B.1.29. User Signing Algorithm Default

This default implements an enrollment default profile that populates a user-supplied signing algorithm in the certificate request. If included in the certificate profile, this allows a user to choose a signing algorithm for the certificate, subject to the constraint set.

No inputs are provided to add signing algorithm choices to the enrollment form, but it is possible to submit a request that contains this information.

The following constraints can be defined with this default:

- Signing Algorithm Constraint; see Section B.2.10, "Signing Algorithm Constraint".
- No Constraints; see Section B.2.8, "No Constraint" .

B.1.30. User Subject Name Default

This default attaches a user-supplied subject name to the certificate request. If included in the certificate profile, it allows a user to supply a subject name for the certificate, subject to the constraints set. This extension preserves the subject name that is specified in the original certificate request when the certificate is issued.

The following constraints can be defined with this default:

- Subject Name Constraint; see Section B.2.11, "Subject Name Constraint" .
- Unique Subject Name Constraint; see Section B.2.13, "Unique Subject Name Constraint" .
- No Constraints; see Section B.2.8, "No Constraint" .

B.1.31. User Validity Default

This default attaches a user-supplied validity to the certificate request. If included in the certificate profile, it allows a user to supply the validity period, subject to the constraints set. This default profile preserves that user-defined validity period in the original certificate request when the certificate is issued.

No inputs are provided to add user-supplied validity date to the enrollment form, but it is possible to submit a request that contains this information.

The following constraints can be defined with this default:

- Validity Constraint; see Section B.2.14, "Validity Constraint".
- No Constraints; see Section B.2.8, "No Constraint".

B.1.32. User Supplied Extension Default

The User Supplied Extension Default class populates a certificate with any certificate extension defined by the user in the certificate request. This requires users to submit certificate requests which meet certain standards or give certain information because the profile can require specific extensions before enrolling a certificate.

WARNING

Be exceptionally cautious about setting this extension default, since it allows users to specify an extension in the certificate request. If this default is used, then Red Hat strongly recommends using a constraint corresponding to the extension to minimize any possible abuse of the User Supplied Extension Default.

The user-defined extension is validated against whatever constraint is set, so it is possible to restrict the kind of extension (through the Extension Constraint) or to set rules for the key and other basic constraints, such as whether this is a CA certificate.

NOTE

If this extension is set on a profile with a corresponding OID (Extension Constraint), then any certificate request processed through that profile *must* carry the specified extension or the request is rejected.

If a certificate profile was enabled with the User Supplied Extension Default *before* the errata RHSA 2008:0500, then this profile must be edited to support user supplied extensions in certificate requests. Apply the **userExtensionDefaultImpl** default, as shown in the example. The given OID is for the Basic Constraints Extension Constraint.

```
policyset.set1.p6.default.class_id=userExtensionDefaultImpl
policyset.set1.p6.default.name=User Supplied Extension Default
policyset.set1.p6.default.params.userExtOID=2.5.29.19
```

The CA handles an enrollment with the User Supplied Extension Default in one of three ways:

- If the OID of the extension is specified in both the certificate request and the default, then the extension is validated by the constraints and applied to the certificate.

- If an OID of an extension is given in the request but is not specified in the User Supplied Extension Default in the profile, then the user-specified extension is ignored, and the certificate is successfully enrolled without that extension.
- If this extension is set on a profile with a corresponding OID (Extension Constraint), then any certificate request processed through that profile *must* carry the specified extension or the request is rejected.

A certificate *request* that contains the user-defined extensions must be submitted to the profile. The certificate enrollment forms, however, do not have any input fields for users to add user-supplied extensions. Submitting a certificate request without supplying the extension fails.

Example B.2, "User Supplied Extension Default for the Extended Key Usage Extension" adds the User Supplied Extension Default to a profile with the Extended Key Usage Constraint. The OID specified in the ***userExtOID*** parameter is for the Extended Key Usage Extension.

Example B.2. User Supplied Extension Default for the Extended Key Usage Extension

```
policyset.set1.2.constraint.class_id=extendedKeyUsageExtConstraintImpl
policyset.set1.2.constraint.name=Extended Key Usage Extension
policyset.set1.2.constraint.params.exKeyUsageCritical=false
policyset.set1.2.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.2,1.3.6.1.5.5.7.3.4
policyset.set1.2.default.class_id=userExtensionDefaultImpl
policyset.set1.2.default.name=User Supplied Extension Default
policyset.set1.2.default.params.userExtOID=2.5.29.37
```

In Example B.2, "User Supplied Extension Default for the Extended Key Usage Extension" , although the User Supplied Extension Default allows a user to specify the Extended Key Usage Extension (2.5.29.37), the constraint limits the user request to only the SSL client authentication (1.3.6.1.5.5.7.3.2) and email protection (1.3.6.1.5.5.7.3.4) uses.

Editing profiles is described in Section 3.2, "Setting up Certificate Profiles".

Example B.3. Multiple User Supplied Extensions in CSR

The RHCS enrollment profile framework allows to define multiple User Supplied Extensions in the same profile. For example, a combination of the following can be specified.

- For Extended Key Usage Extension:

```
policyset.serverCertSet.2.constraint.class_id=extendedKeyUsageExtConstraintImpl
policyset.serverCertSet.2.constraint.name=Extended Key Usage Extension
policyset.serverCertSet.2.constraint.params.exKeyUsageCritical=false
policyset.serverCertSet.2.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.2,1.3.6.1.5.5.
7.3.4
policyset.serverCertSet.2.default.class_id=userExtensionDefaultImpl
policyset.serverCertSet.2.default.name=User Supplied Extension Default
policyset.serverCertSet.2.default.params.userExtOID=2.5.29.37
```

- For Key Usage Extension:

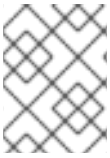
  By using the following format, you can apply a policy which parameter of the extension:

  - *Must* exist in the CSR: **value = "true"**

- *Must not* exist in the CSR: **value = "false"**
- Is *optional*: **value = "-"**

For example:

```
policyset.serverCertSet.13.constraint.class_id=keyUsageExtConstraintImpl
policyset.serverCertSet.13.constraint.name=Key Usage Extension Constraint
policyset.serverCertSet.13.constraint.params.keyUsageCritical=-
policyset.serverCertSet.13.constraint.params.keyUsageCrlSign=false
policyset.serverCertSet.13.constraint.params.keyUsageDataEncipherment=-
policyset.serverCertSet.13.constraint.params.keyUsageDecipherOnly=-
policyset.serverCertSet.13.constraint.params.keyUsageDigitalSignature=-
policyset.serverCertSet.13.constraint.params.keyUsageEncipherOnly=-
policyset.serverCertSet.13.constraint.params.keyUsageKeyAgreement=true
policyset.serverCertSet.13.constraint.params.keyUsageKeyCertSign=-
policyset.serverCertSet.13.constraint.params.keyUsageKeyEncipherment=-
policyset.serverCertSet.13.constraint.params.keyUsageNonRepudiation=-
policyset.serverCertSet.13.default.class_id=userExtensionDefaultImpl
policyset.serverCertSet.13.default.name=User Supplied Key Usage Extension
policyset.serverCertSet.13.default.params.userExtOID=2.5.29.15
```



NOTE

For an example on how to create a CSR with user-defined extensions attributes, see Section 5.2.1.1.2, “Using **certutil** to Create a CSR With User-defined Extensions” .

B.1.33. Validity Default

This default attaches a server-side configurable validity period into the certificate request.

The following constraints can be defined with this default:

- Validity Constraint; see Section B.2.14, “Validity Constraint”.
- No Constraints; see Section B.2.8, “No Constraint”.

Table B.24. Validity Default Configuration Parameters

Parameter	Description
range	Specifies the validity period for this certificate.
startTime	Sets when the validity period begins, based on the current time.

B.2. CONSTRAINTS REFERENCE

Constraints are used to define the allowable contents of a certificate and the values associated with that content. This section lists the predefined constraints with complete definitions of each.

B.2.1. Basic Constraints Extension Constraint

The Basic Constraints extension constraint checks if the basic constraint in the certificate request satisfies the criteria set in this constraint.

Table B.25. Basic Constraints Extension Constraint Configuration Parameters

Parameter	Description
basicConstraintsCritical	Specifies whether the extension can be marked critical or noncritical. Select **true** to mark this extension critical; select **false** to prevent this extension from being marked critical. Selecting a hyphen **-**, implies no criticality preference.
basicConstraintsIsCA	Specifies whether the certificate subject is a CA. Select **true** to require a value of **true** for this parameter (is a CA); select **false** to disallow a value of **true** for this parameter; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
basicConstraintsMinPathLen	Specifies the minimum allowable path length, the maximum number of CA certificates that may be chained below (subordinate to) the subordinate CA certificate being issued. The path length affects the number of CA certificates used during certificate validation. The chain starts with the end-entity certificate being validated and moves up. This parameter has no effect if the extension is set in end-entity certificates. The permissible values are **0** or **n**. The value must be less than the path length specified in the Basic Constraints extension of the CA signing certificate. **0** specifies that no subordinate CA certificates are allowed below the subordinate CA certificate being issued; only an end-entity certificate may follow in the path. *n* must be an integer greater than zero. This is the minimun number of subordinate CA certificates allowed below the subordinate CA certificate being used.

Parameter	Description
basicConstraintsMaxPathLen	Specifies the maximum allowable path length, the maximum number of CA certificates that may be chained below (subordinate to) the subordinate CA certificate being issued. The path length affects the number of CA certificates used during certificate validation. The chain starts with the end-entity certificate being validated and moves up. This parameter has no effect if the extension is set in end-entity certificates. The permissible values are **0** or *n*. The value must be greater than the path length specified in the Basic Constraints extension of the CA signing certificate. **0** specifies that no subordinate CA certificates are allowed below the subordinate CA certificate being issued; only an end-entity certificate may follow in the path. *n* must be an integer greater than zero. This is the maximum number of subordinate CA certificates allowed below the subordinate CA certificate being used. If the field is blank, the path length defaults to a value determined by the path length set on the Basic Constraints extension in the issuer's certificate. If the issuer's path length is unlimited, the path length in the subordinate CA certificate is also unlimited. If the issuer's path length is an integer greater than zero, the path length in the subordinate CA certificate is set to a value one less than the issuer's path length; for example, if the issuer's path length is 4, the path length in the subordinate CA certificate is set to 3.

B.2.2. CA Validity Constraint

The CA Validity constraint checks if the validity period in the certificate template is within the CA's validity period. If the validity period of the certificate is out outside the CA certificate's validity period, the constraint is rejected.

B.2.3. Extended Key Usage Extension Constraint

The Extended Key Usage extension constraint checks if the Extended Key Usage extension on the certificate satisfies the criteria set in this constraint.

Table B.26. Extended Key Usage Extension Constraint Configuration Parameters

Parameter	Description
exKeyUsageCritical	When set to **true**, the extension can be marked as critical. When set to **false**, the extension can be marked noncritical.

Parameter	Description
exKeyUsageOIDs	Specifies the allowable OIDs that identifies a key-usage purpose. Multiple OIDs can be added in a comma-separated list.

B.2.4. Extension Constraint

This constraint implements the general extension constraint. It checks if the extension is present.

Table B.27. Extension Constraint

Parameter	Description
extCritical	Specifies whether the extension can be marked critical or noncritical. Select **true** to mark the extension critical; select **false** to mark it noncritical. Select **-** to enforce no preference.
extOID	The OID of an extension that must be present in the cert to pass the constraint.

B.2.5. Key Constraint

This constraint checks the size of the key for RSA keys, and the name of the elliptic curve for EC keys. When used with RSA keys the ***KeyParameters*** parameter contains a comma-separated list of legal key sizes, and with EC Keys the ***KeyParameters*** parameter contains a comma-separated list of available ECC curves.

Table B.28. Key Constraint Configuration Parameters

Parameter	Description
keyType	Gives a key type; this is set to **-** by default and uses an RSA key system. The choices are rsa and ec. If the key type is specified and not identified by the system, the constraint will be rejected.
KeyParameters	Defines the specific key parameters. The parameters which are set for the key differe, depending on the value of the ***keyType*** parameter (meaning, depending on the key type). • With RSA keys, the ***KeyParameters*** parameter contains a comma-separated list of legal key sizes. • With ECC keys, the ***KeyParameters*** parameter contains a comma-separated list of available ECC curves.

B.2.6. Key Usage Extension Constraint

The Key Usage extension constraint checks if the key usage constraint in the certificate request satisfies the criteria set in this constraint.

Table B.29. Key Usage Extension Constraint Configuration Parameters

Parameter	Description
keyUsageCritical	Select **true** to mark this extension critical; select **false** to mark it noncritical. Select**-** for no preference.
keyUsageDigitalSignature	Specifies whether to sign SSL client certificates and S/MIME signing certificates. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
kleyUsageNonRepudiation	Specifies whether to set S/MIME signing certificates. Select **true** to mark this as set; select**false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter. **WARNING** Using this bit is controversial. Carefully consider the legal consequences of its use before setting it for any certificate.
keyEncipherment	Specifies whether to set the extension for SSL server certificates and S/MIME encryption certificates. Select **true** to mark this as set; select**false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
keyUsageDataEncipherment	Specifies whether to set the extension when the subject's public key is used to encrypt user data, instead of key material. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.

Parameter	Description
keyUsageKeyAgreement	Specifies whether to set the extension whenever the subject's public key is used for key agreement. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
keyUsageCertsign	Specifies whether the extension applies for all CA signing certificates. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
keyUsageCRLSign	Specifies whether to set the extension for CA signing certificates that are used to sign CRLs. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
keyUsageEncipherOnly	Specifies whether to set the extension if the public key is to be used only for encrypting data. If this bit is set, **keyUsageKeyAgreement** should also be set. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.
keyUsageDecipherOnly	Specifies whether to set the extension if the public key is to be used only for deciphering data. If this bit is set, **keyUsageKeyAgreement** should also be set. Select **true** to mark this as set; select **false** to keep this from being set; select a hyphen, **-**, to indicate no constraints are placed for this parameter.

B.2.7. Netscape Certificate Type Extension Constraint

WARNING

This constraint is obsolete. Instead of using the Netscape Certificate Type extension constraint, use the Key Usage extension or Extended Key Usage extension.

The Netscape Certificate Type extension constraint checks if the Netscape Certificate Type extension in the certificate request satisfies the criteria set in this constraint.

B.2.8. No Constraint

This constraint implements no constraint. When chosen along with a default, there are not constraints placed on that default.

B.2.9. Renewal Grace Period Constraint

The Renewal Grace Period Constraint sets rules on when a user can renew a certificate based on its expiration date. For example, users cannot renew a certificate until a certain time before it expires or if it goes past a certain time after its expiration date.

One important thing to remember when using this constraint is that this constraint is set on the *original enrollment profile*, not the renewal profile. The rules for the renewal grace period are part of the original certificate and are carried over and applied for any subsequent renewals.

This constraint is only available with the No Default extension.

Table B.30. Renewal Grace Period Constraint Configuration Parameters

Parameter	Description
renewal.graceAfter	Sets the period, in days, *after* the certificate expires that it can be submitted for renewal. If the certificate has been expired longer that that time, then the renewal request is rejected. If no value is given, there is no limit.
renewal.graceBefore	Sets the period, in days, *before* the certificate expires that it can be submitted for renewal. If the certificate is not that close to its expiration date, then the renewal request is rejected. If no value is given, there is no limit.

B.2.10. Signing Algorithm Constraint

The Signing Algorithm constraint checks if the signing algorithm in the certificate request satisfies the criteria set in this constraint.

Table B.31. Signing Algorithms Constraint Configuration Parameters

Parameter	Description

Parameter	Description
signingAlgsAllowed	Sets the signing algorithms that can be specified to sign the certificate. The algorithms can be any or all of the following: • MD2withRSA • MD5withRSA • SHA256withRSA • SHA512withRSA • SHA256withEC • SHA384withEC • SHA512withEC

B.2.11. Subject Name Constraint

The Subject Name constraint checks if the subject name in the certificate request satisfies the criteria.

Table B.32. Subject Name Constraint Configuration Parameters

Parameter	Description
Pattern	Specifies a regular expression or other string to build the subject DN.

Subject Names and Regular Expressions

The regular expression for the Subject Name Constraint is matched by the Java facility for matching regular expressions. The format for these regular expressions are listed in https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html. This allows wildcards such as asterisks (*****) to search for any number of the characters and periods (**.**) to search for any type character.

For example, if the pattern of the subject name constraint is set to **uid=.***, the certificate profile framework checks if the subject name in the certificate request matches the pattern. A subject name like **uid=user, o=Example, c=US** satisfies the pattern **uid=.***. The subject name **cn=user, o=example,c=US** does not satisfy the pattern. **uid=.*** means the subject name must begin with the ***uid*** attribute; the period-asterisk (**.***) wildcards allow any type and number of characters to follow ***uid***.

It is possible to require internal patterns, such as **.*ou=Engineering.***, which requires the **ou=Engineering** attribute with any kind of string before and after it. This matches **cn=jdoe,ou=internal,ou=west coast,ou=engineering,o="Example Corp",st=NC** as well as **uid=bjensen,ou=engineering,dc=example,dc=com**.

Lastly, it is also possible to allow requests that are either one string or another by setting a pipe sign (|) between the options. For example, to permit subject names that contain either **ou=engineering,ou=people** or **ou=engineering,o="Example Corp"**, the pattern is **.*ou=engineering,ou=people.*** | **.*ou=engineering,o="Example Corp".***.

NOTE

For constructing a pattern which uses a special character, such as a period (**.**), escape the character with a back slash (\). For example, to search for the string **o="Example Inc."**, set the pattern to **o="Example Inc\."**.

Subject Names and the UID or CN in the Certificate Request

The pattern that is used to build the subject DN can also be based on the CN or UID of the person requesting the certificate. The Subject Name Constraint sets the patter of the CN (or UID) to recognize in the DN of the certificate request, and then the Subject Name Default builds on that CN to create the subject DN of the certificate, using a predefined directory tree.

For example, to use the CN of the certificate request:

```
policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policyset.serverCertSet.1.constraint.name=Subject Name Constraint
policyset.serverCertSet.1.constraint.params.pattern=CN=[^,]+,.+
policyset.serverCertSet.1.constraint.params.accept=true
policyset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policyset.serverCertSet.1.default.name=Subject Name Default
policyset.serverCertSet.1.default.params.name=CN=$request.req_subject_name.cn$,DC=example,
DC=com
```

B.2.12. Unique Key Constraint

This constraint checks that the public key is unique.

Table B.33. Unique Key Constraints Parameters

Parameter	Description
allowSameKeyRenewal	A request is considered a renewal and is accepted if this parameter is set to **true**, if a public key is not unique, *and* if the subject DN matches an existing certificate. However, if the public key is a duplicate and does not match an existing Subject DN, the request is rejected. When the parameter is set to **false**, a duplicate public key request will be rejected.

B.2.13. Unique Subject Name Constraint

The Unique Subject Name constraint restricts the server from issuing multiple certificates with the same subject names. When a certificate request is submitted, the server automatically checks the nickname against other issued certificate nicknames. This constraint can be applied to certificate enrollment and renewal through the end-entities' page.

Certificates cannot have the same subject name unless one certificate is expired or revoked (and not on hold). So, active certificates cannot share a subject name, with one exception: if certificates have different key usage bits, then they can share the same subject name, because they have different uses.

Table B.34. Unique Subject Name Constraint Configuration Parameters

Parameter	Description
enableKeyUsageExtensionChecking	Optional setting which allows certificates to have the same subject name as long as their key usage settings are different. This is either **true** or **false**. The default is **true**, which allows duplicate subject names.

B.2.14. Validity Constraint

The Validity constraint checks if the validity period in the certificate request satisfies the criteria.

The parameters provided must be sensible values. For instance, a ***notBefore*** parameter that provides a time which has already passed will not be accepted, and a ***notAfter*** parameter that provides a time earlier than the ***notBefore*** time will not be accepted.

Table B.35. Validity Constraint Configuration Parameters

Parameter	Description
range	The range of the validity period. This is an integer which sets the number of days. The difference (in days) between the ***notBefore*** time and the ***notAfter*** time must be less than the range value, or this constraint will be rejected.
notBeforeCheck	Verifies that the range is not within the grace period. When the ***NotBeforeCheck*** Boolean parameter is set to true, the system will check the ***notBefore*** time is not greater than the current time plus the ***notBeforeGracePeriod*** value. If the ***notBeforeTime*** is not between the current time and the ***notBeforeGracePeriod*** value, this constraint will be rejected.
notBeforeGracePeriod	The grace period (in seconds) after the ***notBefore*** time. If the ***notBeforeTime*** is not between the current time and the ***notBeforeGracePeriod*** value, this constraint will be rejected. This constraint is only checked if the ***notBeforeCheck*** parameter has been set to true.
notAfterCheck	Verfies whether the given time is not after the expiration period. When the ***notAfterCheck*** Boolean parameter is set to true, the system will check the ***notAfter*** time is not greater than the current time. If the current time exceeds the ***notAfter*** time, this constraint will be rejected.

B.3. STANDARD X.509 V3 CERTIFICATE EXTENSION REFERENCE

An X.509 v3 certificate contains an extension field that permits any number of additional fields to be added to the certificate. Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates. Older Netscape servers, such as Red Hat Directory Server and Red Hat Certificate System, that were developed before PKIX part 1 standards were defined require Netscape-specific extensions.

The following is an example of the section of a certificate containing X.509 v3 extensions. The Certificate System can display certificates in readable pretty-print format, as shown here. As in this example, certificate extensions appear in sequence and only one instance of a particular extension may appear per certificate; for example, a certificate may contain only one subject key identifier extension. Certificates that support these extensions have the version **0x2** (which corresponds to version 3).

Example B.4. Sample Pretty-Print Certificate Extensions

```
Data:
  Version:  v3
  Serial Number: 0x1
  Signature Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
  Issuer: CN=Certificate Manager,OU=netscape,O=ExampleCorp,L=MV,ST=CA,C=US
  Validity:
    Not Before: Friday, February 21, 2005 12:00:00 AM PST America/Los_Angeles
    Not  After: Monday, February 21, 2007 12:00:00 AM PST America/Los_Angeles
  Subject: CN=Certificate Manager,OU=netscape,O=ExampleCorp,L=MV,ST=CA,C=US
  Subject Public Key Info:
    Algorithm: RSA - 1.2.840.113549.1.1.1
    Public Key:
      Exponent: 65537
      Public Key Modulus: (2048 bits) :
        E4:71:2A:CE:E4:24:DC:C4:AB:DF:A3:2E:80:42:0B:D9:
        CF:90:BE:88:4A:5C:C5:B3:73:BF:49:4D:77:31:8A:88:
        15:A7:56:5F:E4:93:68:83:00:BB:4F:C0:47:03:67:F1:
        30:79:43:08:1C:28:A8:97:70:40:CA:64:FA:9E:42:DF:
        35:3D:0E:75:C6:B9:F2:47:0B:D5:CE:24:DD:0A:F7:84:
        4E:FA:16:29:3B:91:D3:EE:24:E9:AF:F6:A1:49:E1:96:
        70:DE:6F:B2:BE:3A:07:1A:0B:FD:FE:2F:75:FD:F9:FC:
        63:69:36:B6:5B:09:C6:84:92:17:9C:3E:64:C3:C4:C9
  Extensions:
    Identifier: Netscape Certificate Type - 2.16.840.1.113730.1.1
      Critical: no
      Certificate Usage:
        SSL CA
        Secure Email CA
        ObjectSigning CA
    Identifier: Basic Constraints - 2.5.29.19
      Critical: yes
      Is CA: yes
      Path Length Constraint: UNLIMITED
    Identifier: Subject Key Identifier - 2.5.29.14
      Critical: no
      Key Identifier:
        3B:46:83:85:27:BC:F5:9D:8E:63:E3:BE:79:EF:AF:79:
        9C:37:85:84
    Identifier: Authority Key Identifier - 2.5.29.35
      Critical: no
      Key Identifier:
        3B:46:83:85:27:BC:F5:9D:8E:63:E3:BE:79:EF:AF:79:
```

```
      9C:37:85:84
    Identifier: Key Usage: - 2.5.29.15
      Critical: yes
      Key Usage:
        Digital Signature
        Key CertSign
        Crl Sign
  Signature:
    Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
    Signature:
      AA:96:65:3D:10:FA:C7:0B:74:38:2D:93:54:32:C0:5B:
      2F:18:93:E9:7C:32:E6:A4:4F:4E:38:93:61:83:3A:6A:
      A2:11:91:C2:D2:A3:48:07:6C:07:54:A8:B8:42:0E:B4:
      E4:AE:42:B4:B5:36:24:46:4F:83:61:64:13:69:03:DF:
      41:88:0B:CB:39:57:8C:6B:9F:52:7E:26:F9:24:5E:E7:
      BC:FB:FD:93:13:AF:24:3A:8F:DB:E3:DC:C9:F9:1F:67:
      A8:BD:0B:95:84:9D:EB:FC:02:95:A0:49:2C:05:D4:B0:
      35:EA:A6:80:30:20:FF:B1:85:C8:4B:74:D9:DC:BB:50
```

An *object identifier* (OID) is a string of numbers identifying a unique object, such as a certificate extension or a company's certificate practice statement. The Certificate System comes with a set of extension-specific profile plug-in modules which enable X.509 certificate extensions to be added to the certificates the server issues. Some of the extensions contain fields for specifying OIDs.

The PKIX standard recommends that all objects, such as extensions and statements, that are used in certificates be included in the form of an OID. This promotes interoperability between organizations on a shared network. If certificates will be issued that will be used on shared networks, register the OID prefixes with the appropriate registration authority.

OIDs are controlled by the International Standards Organization (ISO) registration authority. In some cases, this authority is delegated by ISO to regional registration authorities. In the United States, the American National Standards Institute (ANSI) manages this registration.

Using an OID registered to another organization or failing to register an OID may carry legal consequences, depending the situation. Registration may be subject to fees. For more information, contact the appropriate registration authority.

To define or assign OIDs for custom objects, know the company's *arc*, an OID for a private enterprise. If the company does not have an arc, it needs to get one. The http://www.alvestrand.no/objectid/ has more information on registering and using OIDs.

For example, the Netscape-defined OID for an extension named **Netscape Certificate Comment** is 2.16.840.1.113730.1.13. The OID assigned to this extension is hierarchical and includes the former Netscape company arc, **2.16.840.1**. The OID definition entry is http://www.alvestrand.no/objectid/2.16.840.1.113730.1.13.html.

If an OID extension exists in a certificate and is marked critical, the application validating the certificate must be able to interpret the extension, including any optional qualifiers, or it must reject the certificate. Since it is unlikely that all applications will be able to interpret a company's custom extensions embedded in the form of OIDs, the PKIX standard recommends that the extension be always marked noncritical.

This section summarizes the extension types defined as part of the Internet X.509 version 3 standard and indicates which types are recommended by the PKIX working group.

This reference summarizes important information about each certificate. For complete details, see both the X.509 v3 standard, available from the ITU, and *Internet X.509 Public Key Infrastructure - Certificate and CRL Profile (RFC 3280)*, available at RFC 3280. The descriptions of extensions reference the RFC and section number of the standard draft that discusses the extension; the object identifier (OID) for each extension is also provided.

Each extension in a certificate can be designated as critical or noncritical. A certificate-using system, such as a web browser, must reject the certificate if it encounters a critical extension it does not recognize; however, a noncritical extension can be ignored if it is not recognized.

B.3.1. authorityInfoAccess

The Authority Information Access extension indicates how and where to access information about the issuer of the certificate. The extension contains an **accessMethod** and an **accessLocation** field. **accessMethod** specifies by OID the type and format of information about the issuer named in **accessLocation**.

PKIX Part 1 defines one **accessMethod** (**id-ad-calssuers**) to get a list of CAs that have issued certificates higher in the CA chain than the issuer of the certificate using the extension. The **accessLocation** field then typically contains a URL indicating the location and protocol (LDAP, HTTP, or FTP) used to retrieve the list.

The Online Certificate Status Protocol (RFC 2560), available at RFC 2560, defines an *accessMethod* (**id-ad-ocsp**) for using OCSP to verify certificates. The **accessLocation** field then contains a URL indicating the location and protocol used to access an OCSP responder that can validate the certificate.

OID

1.3.6.1.5.5.7.1.1

Criticality

This extension must be noncritical.

B.3.2. authorityKeyIdentifier

The Authority Key Identifier extension identifies the public key corresponding to the private key used to sign a certificate. This extension is useful when an issuer has multiple signing keys, such as when a CA certificate is renewed.

The extension consists of one or both of the following:

- An explicit key identifier, set in the **keyIdentifier** field
- An issuer, set in the **authorityCertIssuer** field, and serial number, set in the **authorityCertSerialNumber** field, identifying a certificate

If the **keyIdentifier** field exists, it is used to select the certificate with a matching **subjectKeyIdentifier** extension. If the **authorityCertIssuer** and **authorityCertSerialNumber** fields are present, then they are used to identify the correct certificate by **issuer** and **serialNumber**.

If this extension is not present, then the issuer name alone is used to identify the issuer certificate.

PKIX Part 1 requires this extension for all certificates except self-signed root CA certificates. Where a key identifier has not been established, PKIX recommends that the **authorityCertIssuer** and **authorityCertSerialNumber** fields be specified. These fields permit construction of a complete

certificate chain by matching the **SubjectName** and **CertificateSerialNumber** fields in the issuer's certificate against the **authortiyCertIssuer** and **authorityCertSerialNumber** in the Authority Key Identifier extension of the subject certificate.

OID

2.5.29.35

Criticality

This extension is always noncritical and is always evaluated.

B.3.3. basicConstraints

This extension is used during the certificate chain verification process to identify CA certificates and to apply certificate chain path length constraints. The **cA** component should be set to **true** for all CA certificates. PKIX recommends that this extension should not appear in end-entity certificates.

If the **pathLenConstraint** component is present, its value must be greater than the number of CA certificates that have been processed so far, starting with the end-entity certificate and moving up the chain. If **pathLenConstraint** is omitted, then all of the higher level CA certificates in the chain must not include this component when the extension is present.

OID

2.5.29.19

Criticality

PKIX Part 1 requires that this extension be marked critical. This extension is evaluated regardless of its criticality.

B.3.4. certificatePoliciesExt

The Certificate Policies extension defines one or more policies, each of which consists of an OID and optional qualifiers. The extension can include a URI to the issuer's Certificate Practice Statement or can embed issuer information, such as a user notice in text form. This information can be used by certificate-enabled applications.

If this extension is present, PKIX Part 1 recommends that policies be identified with an OID only, or, if necessary, only certain recommended qualifiers.

OID

2.5.29.32

Criticality

This extension may be critical or noncritical.

B.3.5. CRLDistributionPoints

This extension defines how CRL information is obtained. It should be used if the system is configured to use CRL issuing points.

If the extension contains a **DistributionPointName** with a type set to URI, the URI is assumed to be a pointer to the current CRL for the specified revocation reasons and will be issued by the named **cRLIssuer**. The expected values for the URI are those defined for the Subject Alternative Name

extension. If the **distributionPoint** omits reasons, the CRL must include revocations for all reasons. If the **distributionPoint** omits **cRLIssuer**, the CRL must be issued by the CA that issued the certificate.

PKIX recommends that this extension be supported by CAs and applications.

OID

2.5.29.31

Criticality

PKIX recommends that this extension be marked noncritical and that it be supported for all certificates.

B.3.6. extKeyUsage

The Extended Key Usage extension indicates the purposes for which the certified public key may be used. These purposes may be in addition to or in place of the basic purposes indicated in the Key Usage extension.

The Extended Key Usage extension must include **OCSP Signing** in an OCSP responder's certificate unless the CA signing key that signed the certificates validated by the responder is also the OCSP signing key. The OCSP responder's certificate must be issued directly by the CA that signs certificates the responder will validate.

The Key Usage, Extended Key Usage, and Basic Constraints extensions act together to define the purposes for which the certificate is intended to be used. Applications can use these extensions to disallow the use of a certificate in inappropriate contexts.

Table B.36, "PKIX Extended Key Usage Extension Uses" lists the uses defined by PKIX for this extension, and Table B.37, "Private Extended Key Usage Extension Uses" lists uses privately defined by Netscape.

OID

2.5.29.37

Criticality

If this extension is marked critical, the certificate must be used for one of the indicated purposes only. If it is not marked critical, it is treated as an advisory field that may be used to identify keys but does not restrict the use of the certificate to the indicated purposes.

Table B.36. PKIX Extended Key Usage Extension Uses

Use	OID
Server authentication	1.3.6.1.5.5.7.3.1
Client authentication	1.3.6.1.5.5.7.3.2
Code signing	1.3.6.1.5.5.7.3.3
Email	1.3.6.1.5.5.7.3.4
Timestamping	1.3.6.1.5.5.7.3.8

Use	OID
OCSP Signing	1.3.6.1.5.5.7.3.9[a]
[a] OCSP Signing is not defined in PKIX Part 1, but in RFC 2560, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*.	

Table B.37. Private Extended Key Usage Extension Uses

Use	OID
Certificate trust list signing	1.3.6.1.4.1.311.10.3.1
Microsoft Server Gated Crypto (SGC)	1.3.6.1.4.1.311.10.3.3
Microsoft Encrypted File System	1.3.6.1.4.1.311.10.3.4
Netscape SGC	2.16.840.1.113730.4.1

B.3.7. issuerAltName Extension

The Issuer Alternative Name extension is used to associate Internet-style identities with the certificate issuer. Names must use the forms defined for the Subject Alternative Name extension.

OID

2.5.29.18

Criticality

PKIX Part 1 recommends that this extension be marked noncritical.

B.3.8. keyUsage

The Key Usage extension defines the purpose of the key contained in the certificate. The Key Usage, Extended Key Usage, and Basic Constraints extensions act together to specify the purposes for which a certificate can be used.

If this extension is included at all, set the bits as follows:

- **digitalSignature** (**0**) for SSL client certificates, S/MIME signing certificates, and object-signing certificates.
- **nonRepudiation** (**1**) for some S/MIME signing certificates and object-signing certificates.

WARNING

Use of this bit is controversial. Carefully consider the legal consequences of its use before setting it for any certificate.

- **keyEncipherment** (**2**) for SSL server certificates and S/MIME encryption certificates.
- **dataEncipherment** (**3**) when the subject's public key is used to encrypt user data instead of key material.
- **keyAgreement** (**4**) when the subject's public key is used for key agreement.
- **keyCertSign** (**5**) for all CA signing certificates.
- **cRLSign** (**6**) for CA signing certificates that are used to sign CRLs.
- **encipherOnly** (**7**) if the public key is used only for enciphering data. If this bit is set, **keyAgreement** should also be set.
- **decipherOnly** (**8**) if the public key is used only for deciphering data. If this bit is set, **keyAgreement** should also be set.

Table B.38, "Certificate Uses and Corresponding Key Usage Bits" summarizes the guidelines for typical certificate uses.

If the **keyUsage** extension is present and marked critical, then it is used to enforce the usage of the certificate and key. The extension is used to limit the usage of a key; if the extension is not present or not critical, all types of usage are allowed.

If the **keyUsage** extension is present, critical or not, it is used to select from multiple certificates for a given operation. For example, it is used to distinguish separate signing and encryption certificates for users who have separate certificates and key pairs for operations.

OID

2.5.29.15

Criticality

This extension may be critical or noncritical. PKIX Part 1 recommends that it should be marked critical if it is used.

Table B.38. Certificate Uses and Corresponding Key Usage Bits

Purpose of Certificate	Required Key Usage Bit
CA Signing	• keyCertSign • cRLSign

Purpose of Certificate	Required Key Usage Bit
SSL Client	digitalSignature
SSL Server	keyEncipherment
S/MIME Signing	digitalSignature
S/MIME Encryption	keyEncipherment
Certificate Signing	keyCertSign
Object Signing	digitalSignature

B.3.9. nameConstraints

This extension, which can used in CA certificates only, defines a name space within which all subject names in subsequent certificates in a certification path must be located.

OID

2.5.29.30

Criticality

PKIX Part 1 requires that this extension be marked critical.

B.3.10. OCSPNocheck

The extension is meant to be included in an OCSP signing certificate. The extension tells an OCSP client that the signing certificate can be trusted without querying the OCSP responder (since the reply would again be signed by the OCSP responder, and the client would again request the validity status of the signing certificate). This extension is null-valued; its meaning is determined by its presence or absence.

Since the presence of this extension in a certificate will cause OCSP clients to trust responses signed with that certificate, use of this extension should be managed carefully. If the OCSP signing key is compromised, the entire process of validating certificates in the PKI will be compromised for the duration of the validity period of the certificate. Therefore, certificates using **OCSPNocheck** should be issued with short lifetimes and be renewed frequently.

OID

1.3.6.1.5.5.7.48.4

Criticality

This extension should be noncritical.

B.3.11. policyConstraints

This extension, which is for CA certificates only, constrains path validation in two ways. It can be used to prohibit policy mapping or to require that each certificate in a path contain an acceptable policy identifier.

PKIX requires that, if present, this extension must never consist of a null sequence. At least one of the two available fields must be present.

OID

2.5.29.36

Criticality

This extension may be critical or noncritical.

B.3.12. policyMappings

The Policy Mappings extension is used in CA certificates only. It lists one or more pairs of OIDs used to indicate that the corresponding policies of one CA are equivalent to policies of another CA. It may be useful in the context of cross-pair certificates.

This extension may be supported by CAs and applications.

OID

2.5.29.33

Criticality

This extension must be noncritical.

B.3.13. privateKeyUsagePeriod

The Private Key Usage Period extension allows the certificate issuer to specify a different validity period for the private key than for the certificate itself. This extension is intended for use with digital signature keys.

NOTE

PKIX Part 1 recommends against the use of this extension. CAs conforming to PKIX Part 1 *must not* generate certificates with this extension.

OID

2.5.29.16

B.3.14. subjectAltName

The Subject Alternative Name extension includes one or more alternative (non-X.500) names for the identity bound by the CA to the certified public key. It may be used in addition to the certificate's subject name or as a replacement for it. Defined name forms include Internet electronic mail address (SMTP, as defined in RFC-822), DNS name, IP address (both IPv4 and IPv6), and uniform resource identifier (URI).

PKIX requires this extension for entities identified by name forms other than the X.500 distinguished name (DN) used in the subject field. PKIX Part 1 describes additional rules for the relationship between this extension and the subject field.

Email addresses may be provided in the Subject Alternative Name extension, the certificate subject name field, or both. If the email address is part of the subject name, it must be in the form of the **EmailAddress** attribute defined by PKCS #9. Software that supports S/MIME must be able to read an

email address from either the Subject Alternative Name extension or from the subject name field.

OID

2.5.29.17

Criticality

If the certificate's subject field is empty, this extension must be marked critical.

B.3.15. subjectDirectoryAttributes

The Subject Directory Attributes extension conveys any desired directory attribute values for the subject of the certificate. It is not recommended as an essential part of the proposed PKIX standard but may be used in local environments.

OID

2.5.29.9

Criticality

PKIX Part 1 requires that this extension be marked noncritical.

B.3.16. subjectKeyIdentifier

The Subject Key Identifier extension identifies the public key certified by this certificate. This extension provides a way of distinguishing public keys if more than one is available for a given subject name.

The value of this extension should be calculated by performing a SHA-1 hash of the certificate's DER-encoded **subjectPublicKey**, as recommended by PKIX. The Subject Key Identifier extension is used in conjunction with the Authority Key Identifier extension for CA certificates. If the CA certificate has a Subject Key Identifier extension, the key identifier in the Authority Key Identifier extension of the certificate being verified should match the key identifier of the CA's Subject Key Identifier extension. It is not necessary for the verifier to recompute the key identifier in this case.

PKIX Part 1 requires this extension for all CA certificates and recommends it for all other certificates.

OID

2.5.29.14

Criticality

This extension is always noncritical.

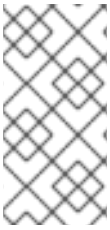
B.4. CRL EXTENSIONS

B.4.1. About CRL Extensions

Since its initial publication, the X.509 standard for CRL formats has been amended to include additional information within a CRL. This information is added through CRL extensions.

The extensions defined by ANSI X9 and ISO/IEC/ITU for X.509 CRLs [X.509] [X9.55] allow additional attributes to be associated with CRLs. The *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, available at RFC 5280, recommends a set of extensions to be used in CRLs. These extensions are called *standard CRL extensions*.

The standard also allows custom extensions to be created and included in CRLs. These extensions are called *private*, *proprietary*, or *custom* CRL extensions and carry information unique to an organization or business. Applications may not able to validate CRLs that contain private critical extensions, so it is not recommended that custom extensions be used in a general context.



NOTE

Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER) standards are specified in the CCITT Recommendations X.208 and X.209. For a quick summary of ASN.1 and DER, see *A Layman's Guide to a Subset of ASN.1, BER, and DER* , which is available at RSA Laboratories' web site, http://www.rsa.com.

B.4.1.1. Structure of CRL Extensions

A CRL extension consists of the following parts:

- The object identifier (OID) for the extension. This identifier uniquely identifies the extension. It also determines the ASN.1 type of value in the value field and how the value is interpreted. When an extension appears in a CRL, the OID appears as the extension ID field (**extnID**) and the corresponding ASN.1 encoded structure appears as the value of the octet string (**extnValue**); examples are shown in Example B.4, “Sample Pretty-Print Certificate Extensions” .
- A flag or Boolean field called **critical**.

  The **true** or **false** value assigned to this field indicates whether the extension is critical or noncritical to the CRL.

  - If the extension is critical and the CRL is sent to an application that does not understand the extension based on the extension's ID, the application must reject the CRL.
  - If the extension is not critical and the CRL is sent to an application that does not understand the extension based on the extension's ID, the application can ignore the extension and accept the CRL.
- An octet string containing the DER encoding of the value of the extension.

The application receiving the CRL checks the extension ID to determine if it can recognize the ID. If it can, it uses the extension ID to determine the type of value used.

B.4.1.2. Sample CRL and CRL Entry Extensions

The following is an example of an X.509 CRL version 2 extension. The Certificate System can display CRLs in readable pretty-print format, as shown here. As shown in the example, CRL extensions appear in sequence and only one instance of a particular extension may appear per CRL; for example, a CRL may contain only one Authority Key Identifier extension. However, CRL-entry extensions appear in appropriate entries in the CRL.

```
Certificate Revocation List:
  Data:
    Version:  v2
    Signature Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
    Issuer: CN=Certificate Authority,O=Example Domain
    This Update: Wednesday, July 29, 2009 8:59:48 AM GMT-08:00
    Next Update: Friday, July 31, 2009 8:59:48 AM GMT-08:00
    Revoked Certificates: 1-3 of 3
```

```
        Serial Number: 0x11
        Revocation Date: Thursday, July 23, 2009 10:07:15 AM GMT-08:00
        Extensions:
          Identifier: Revocation Reason - 2.5.29.21
            Critical: no
            Reason: Privilege_Withdrawn
        Serial Number: 0x1A
        Revocation Date: Wednesday, July 29, 2009 8:50:11 AM GMT-08:00
        Extensions:
          Identifier: Revocation Reason - 2.5.29.21
            Critical: no
            Reason: Certificate_Hold
          Identifier: Invalidity Date - 2.5.29.24
            Critical: no
            Invalidity Date: Sun Jul 26 23:00:00 GMT-08:00 2009
        Serial Number: 0x19
        Revocation Date: Wednesday, July 29, 2009 8:50:49 AM GMT-08:00
        Extensions:
          Identifier: Revocation Reason - 2.5.29.21
            Critical: no
            Reason: Key_Compromise
          Identifier: Invalidity Date - 2.5.29.24
            Critical: no
            Invalidity Date: Fri Jul 24 23:00:00 GMT-08:00 2009
    Extensions:
      Identifier: Authority Info Access: - 1.3.6.1.5.5.7.1.1
        Critical: no
        Access Description:
          Method #0: ocsp
          Location #0: URIName: http://example.com:9180/ca/ocsp
      Identifier: Issuer Alternative Name - 2.5.29.18
        Critical: no
        Issuer Names:
          DNSName: example.com
      Identifier: Authority Key Identifier - 2.5.29.35
        Critical: no
        Key Identifier:
          50:52:0C:AA:22:AC:8A:71:E3:91:0C:C5:77:21:46:9C:
          0F:F8:30:60
      Identifier: Freshest CRL - 2.5.29.46
        Critical: no
        Number of Points: 1
        Point 0
          Distribution Point: [URIName: http://server.example.com:8443/ca/ee/ca/getCRL?
op=getDeltaCRL&crlIssuingPoint=MasterCRL]
      Identifier: CRL Number - 2.5.29.20
        Critical: no
        Number: 39
      Identifier: Issuing Distribution Point - 2.5.29.28
        Critical: yes
        Distribution Point:
          Full Name:
            URIName: http://example.com:9180/ca/ee/ca/getCRL?
op=getCRL&crlIssuingPoint=MasterCRL
        Only Contains User Certificates: no
        Only Contains CA Certificates: no
```

```
        Indirect CRL: no
    Signature:
      Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
      Signature:
        47:D2:CD:C9:E5:F5:9D:56:0A:97:31:F5:D5:F2:51:EB:
        1F:CF:FA:9E:63:D4:80:13:85:E5:D8:27:F0:69:67:B5:
        89:4F:59:5E:69:E4:39:93:61:F2:E3:83:51:0B:68:26:
        CD:99:C4:A2:6C:2B:06:43:35:36:38:07:34:E4:93:80:
        99:2F:79:FB:76:E8:3D:4C:15:5A:79:4E:E5:3F:7E:FC:
        D8:78:0D:1D:59:A0:4C:14:42:B7:22:92:89:38:3A:4C:
        4A:3A:06:DE:13:74:0E:E9:63:74:D0:2F:46:A1:03:37:
        92:F0:93:D9:AA:F8:13:C5:06:25:02:B0:FD:3B:41:E7:
        62:6F:67:A3:9F:F5:FA:03:41:DA:8D:FD:EA:2F:E3:2B:
        3E:F8:E9:CC:3B:9F:E4:ED:73:F2:9E:B9:54:14:C1:34:
        68:A7:33:8F:AF:38:85:82:40:A2:06:97:3C:B4:88:43:
        7B:AF:5D:87:C4:47:63:4A:11:65:E3:75:55:4D:98:97:
        C2:2E:62:08:A4:04:35:5A:FE:0A:5A:6E:F1:DE:8E:15:
        27:1E:0F:87:33:14:16:2E:57:F7:DC:77:BE:D2:75:AB:
        A9:7C:42:1F:84:6D:40:EC:E7:ED:84:F8:14:16:28:33:
        FD:11:CD:C5:FC:49:B7:7B:39:57:B3:E6:36:E5:CD:B6
```

A *delta CRL* is a subset of the CRL which contains only the changes since the last CRL was published. Any CRL which contains the delta CRL indicator extension is a delta CRL.

```
ertificate Revocation List:
  Data:
    Version:  v2
    Signature Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
    Issuer: CN=Certificate Authority,O=SjcRedhat Domain
    This Update: Wednesday, July 29, 2009 9:02:28 AM GMT-08:00
    Next Update: Thursday, July 30, 2009 9:02:28 AM GMT-08:00
    Revoked Certificates:
      Serial Number: 0x1A
      Revocation Date: Wednesday, July 29, 2009 9:00:48 AM GMT-08:00
      Extensions:
        Identifier: Revocation Reason - 2.5.29.21
          Critical: no
          Reason: Remove_from_CRL
      Serial Number: 0x17
      Revocation Date: Wednesday, July 29, 2009 9:02:16 AM GMT-08:00
      Extensions:
        Identifier: Revocation Reason - 2.5.29.21
          Critical: no
          Reason: Certificate_Hold
        Identifier: Invalidity Date - 2.5.29.24
          Critical: no
          Invalidity Date: Mon Jul 27 23:00:00 GMT-08:00 2009
  Extensions:
    Identifier: Authority Info Access: - 1.3.6.1.5.5.7.1.1
      Critical: no
      Access Description:
        Method #0: ocsp
        Location #0: URIName: http://server.example.com:8443/ca/ocsp
    Identifier: Delta CRL Indicator - 2.5.29.27
      Critical: yes
      Base CRL Number: 39
```

```
        Identifier: Issuer Alternative Name - 2.5.29.18
            Critical: no
            Issuer Names:
                DNSName: a-f8.sjc.redhat.com
        Identifier: Authority Key Identifier - 2.5.29.35
            Critical: no
            Key Identifier:
                50:52:0C:AA:22:AC:8A:71:E3:91:0C:C5:77:21:46:9C:
                0F:F8:30:60
        Identifier: CRL Number - 2.5.29.20
            Critical: no
            Number: 41
        Identifier: Issuing Distribution Point - 2.5.29.28
            Critical: yes
            Distribution Point:
                Full Name:
                    URIName: http://server.example.com:8443/ca/ee/ca/getCRL?
op=getCRL&crlIssuingPoint=MasterCRL
            Only Contains User Certificates: no
            Only Contains CA Certificates: no
            Indirect CRL: no
    Signature:
        Algorithm: SHA1withRSA - 1.2.840.113549.1.1.5
        Signature:
            68:28:DA:90:D5:39:CB:6D:BE:42:04:77:C9:E4:09:60:
            C1:97:A6:99:AB:A0:5B:A2:F3:8B:5E:4E:D6:05:70:B0:
            87:1F:D7:0E:4B:C6:B2:DE:8B:92:D8:7C:3B:36:1C:79:
            96:2A:64:E6:7A:25:1D:E7:40:62:48:7A:24:C9:9D:11:
            A6:7F:BB:6B:03:A0:9C:1D:BC:1C:EE:9A:4B:A6:48:2C:
            3B:5E:2B:B1:70:3C:C3:42:96:28:26:AB:82:18:F2:E9:
            F2:55:48:A8:7E:7F:FE:D4:3D:0B:EA:A2:2F:4E:E6:C3:
            C3:C1:6A:E5:C6:85:5B:42:B1:70:2A:C6:E1:D9:0C:AF:
            DA:01:22:FF:80:6E:2E:A7:E5:34:DC:AF:E6:C2:B5:B3:
            1B:FC:28:36:8A:91:4A:22:E7:03:A5:ED:4E:62:0C:D9:
            7F:81:BB:80:99:B8:61:2A:02:C6:9C:41:2E:01:82:21:
            80:82:69:52:BD:B2:AA:DB:0F:80:0A:7E:2A:F3:15:32:
            69:D2:40:0D:39:59:93:75:A2:ED:24:70:FB:EE:19:C0:
            BE:A2:14:36:D0:AC:E8:E2:EE:23:83:DD:BC:DF:38:1A:
            9E:37:AF:E3:50:D9:47:9D:22:7C:36:35:BF:13:2C:16:
            A2:79:CF:05:41:88:8E:B6:A2:4E:B3:48:6D:69:C6:38
```

B.4.2. Standard X.509 v3 CRL Extensions Reference

In addition to certificate extensions, the X.509 proposed standard defines extensions to CRLs, which provide methods for associating additional attributes with Internet CRLs. These are one of two kinds: extensions to the CRL itself and extensions to individual certificate entries in the CRL.

- Section B.4.2.1, “Extensions for CRLs”
- Section B.4.2.2, “CRL Entry Extensions”

B.4.2.1. Extensions for CRLs

The following CRL descriptions are defined as part of the Internet X.509 v3 Public Key Infrastructure proposed standard.

- Section B.4.2.1.1, "authorityInfoAccess"
- Section B.4.2.1.2, "authorityKeyIdentifier"
- Section B.4.2.1.3, "CRLNumber"
- Section B.4.2.1.4, "deltaCRLIndicator"
- Section B.4.2.1.5, "FreshestCRL"
- Section B.4.2.1.6, "issuerAltName"
- Section B.4.2.1.7, "issuingDistributionPoint"
- Section B.4.2.1.5, "FreshestCRL"

B.4.2.1.1. authorityInfoAccess

The Authority Information Access extension identifies how delta CRL information is obtained. The freshestCRL extension is placed in the full CRL to indicate where to find the latest delta CRL.

OID

1.3.6.1.5.5.7.1.1

Criticality

PKIX requires that this extension must not be critical.

Parameters

Table B.39. Authority Infomation Access Configuration Parameters

Parameter	Description
enable	Specifies whether the rule is enabled or disabled. The default is to have this extension disabled.
critical	Sets whether the extension is marked as critical; the default is noncritical.
numberOfAccessDescriptions	Indicates the number of access descriptions, from 0 to any positive integer; the default is 0. When setting this parameter to an integer other than 0, set the number, and then click OK to close the window. Re-open the edit window for the rule, and the fields to set the points will be present.
accessMethod*n*	The only accepted value for this parameter is calssuers. The calssuers method is used when the information available lists certificates that can be used to verify the signature on the CRL. No other method should be used when the AIA extension is included in a CRL.

Parameter	Description
accessLocationType*n*	Specifies the type of access location for the *n* access description. The options are either **DirectoryName** or **URI**.
accessLocation*n*	If **accessLocationType** is set to **DirectoryName**, the value must be a string in the form of an X.500 name, similar to the subject name in a certificate. For example, *CN=CACentral,OU=Research Dept,O=Example Corporation,C=US*. If **accessLocationType** is set to **URI**, the name must be a URI; the URI must be an absolute pathname and must specify the host. For example, *http://testCA.example.com/get/crls/here/*.

B.4.2.1.2. authorityKeyIdentifier

The Authority Key Identifier extension for a CRL identifies the public key corresponding to the private key used to sign the CRL. For details, see the discussion under certificate extensions at Section B.3.2, "authorityKeyIdentifier".

The PKIX standard recommends that the CA must include this extension in all CRLs it issues because a CA's public key can change, for example, when the key gets updated, or the CA may have multiple signing keys because of multiple concurrent key pairs or key changeover. In these cases, the CA ends up with more than one key pair. When verifying a signature on a certificate, other applications need to know which key was used in the signature.

OID

2.5.29.35

Parameters

Table B.40. AuthorityKeyIdentifierExt Configuration Parameters

Parameter	Description
enable	Specifies whether the rule is enabled or disabled. The default is to have this extension disabled.
critical	Sets whether the extension is marked as critical; the default is noncritical.

B.4.2.1.3. CRLNumber

The CRLNumber extension specifies a sequential number for each CRL issued by a CA. It allows users to easily determine when a particular CRL supersedes another CRL. PKIX requires that all CRLs have this extension.

OID

2.5.29.20

Criticality

This extension must not be critical.

Parameters

Table B.41. CRLNumber Configuration Parameters

Parameter	Description
enable	Specifies whether the rule is enabled, which is the default.
critical	Sets whether the extension is marked as critical; the default is noncritical.

B.4.2.1.4. deltaCRLIndicator

The deltaCRLIndicator extension generates a delta CRL, a list only of certificates that have been revoked since the last CRL; it also includes a reference to the base CRL. This updates the local database while ignoring unchanged information already in the local database. This can significantly improve processing time for applications that store revocation information in a format other than the CRL structure.

OID

2.5.29.27

Criticality

PKIX requires that this extension be critical if it exists.

Parameters

Table B.42. DeltaCRL Configuration Parameters

Parameter	Description
enable	Sets whether the rule is enabled. By default, it is disabled.
critical	Sets whether the extension is critical or noncritical. By default, this is critical.

B.4.2.1.5. FreshestCRL

The freshestCRL extension identifies how delta CRL information is obtained. The freshestCRL extension is placed in the full CRL to indicate where to find the latest delta CRL.

OID

2.5.29.46

Criticality

PKIX requires that this extension must be noncritical.

Parameters

Table B.43. FreshestCRL Configuration Parameters

Parameter	Description
enable	Sets whether the extension rule is enabled. By default, this is disabled.
critical	Marks the extension as critical or noncritical. The default is noncritical.
numPoints	Indicates the number of issuing points for the delta CRL, from **0** to any positive integer; the default is **0**. When setting this to an integer other than 0, set the number, and then click **OK** to close the window. Re-open the edit window for the rule, and the fields to set these points will be present.
pointType*n*	Specifies the type of issuing point for the *n* issuing point. For each number specified in **numPoints**, there is an equal number of **pointType** parameters. The options are either **DirectoryName** or **URIName**.
pointName*n*	If **pointType** is set to **directoryName**, the value must be a string in the form of an X.500 name, similar to the subject name in a certificate. For example, *CN=CACentral,OU=Research Dept,O=Example Corporation,C=US*. If **pointType** is set to **URIName**, the name must be a URI; the URI must be an absolute pathname and must specify the host. For example, *http://testCA.example.com/get/crls/here/*.

B.4.2.1.6. issuerAltName

The Issuer Alternative Name extension allows additional identities to be associated with the issuer of the CRL, like binding attributes such as a mail address, a DNS name, an IP address (both IPv4 and IPv6), and a uniform resource indicator (URI), with the issuer of the CRL. For details, see the discussion under certificate extensions at Section B.3.7, “issuerAltName Extension”.

OID

2.5.29.18

Parameters

Table B.44. IssuerAlternativeName Configuration Parameters

Parameter	Description
enable	Sets whether the extension rule is enabled; by default, this is disabled.
critical	Sets whether the extension is critical; by default, this is noncritical.
numNames	Sets the total number of alternative names or identities permitted in the extension. Each name has a set of configuration parameters, **nameType** and **name**, which must have appropriate values or the rule returns an error. Change the total number of identities by changing the value specified in this field; there is no limit on the total number of identities that can be included in the extension. Each set of configuration parameters is distinguished by an integer derived from the value of this field. For example, if the **numNames** parameter is set to **2**, the derived integers are **0** and **1**.

Parameter	Description
nameType*n*	Specifies the general-name type; this can be any of the following: • **rfc822Name** if the name is an Internet mail address. • **directoryName** if the name is an X.500 directory name. • **dNSName** if the name is a DNS name. • **ediPartyName** if the name is a EDI party name. • **URL** if the name is a URI (default). • **iPAddress** if the name is an 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*. • **OID** if the name is an object identifier. • **otherName** if the name is in any other name form; this supports **PrintableString**, **IA5String**, **UTF8String**, **BMPString**, **Any**, and **KerberosName**.
name*n*	Specifies the general-name value; the allowed values depend on the name type specified in the **nameType** field. • For **rfc822Name**, the value must be a valid Internet mail address in the *local-part@domain* format. • For **directoryName**, the value must be a string X.500 name, similar to the subject name in a certificate. For example, *CN=CACentral,OU=Research Dept,O=Example Corporation,C=US*. • For **dNSName**, the value must be a valid domain name in the DNS format. For example, *testCA.example.com*. • For **ediPartyName**, the name must be an IA5String. For example, *Example Corporation*.

Parameter	Description
	• For **URL**, the value must be a non-relative URL. For example, *http://testCA.example.com*. • For **iPAddress**, the value must be a valid IP address specified in dot-separated numeric component notation. It can be the IP address or the IP address including the netmask. 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 IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. 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:255.255.255.0*, and *FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000*. • For **OID**, the value must be a unique, valid OID specified in the dot-separated numeric component notation. For example, *1.2.3.4.55.6.5.99*. Although custom OIDs can be used to evaluate and test the server, in a production environment, comply with the ISO rules for defining OIDs and for registering subtrees of IDs. • For **otherName**, the names can be 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/pki/pki-tomcat/ca/othername.txt*. **KerberosName** has the format *Realm|NameType|NameStrings*, such as *realm1|0|userID1,userID2*. The name must be the absolute path to the file that contains the general name in its base-64 encoded format. For example, */var/lib/pki/pki-tomcat/ca/extn/ian/othername.txt*.

B.4.2.1.7. issuingDistributionPoint

The Issuing Distribution Point CRL extension identifies the CRL distribution point for a particular CRL and indicates what kinds of revocation it covers, such as revocation of end-entity certificates only, CA certificates only, or revoked certificates that have a limited set of reason codes.

PKIX Part I does not require this extension.

OID

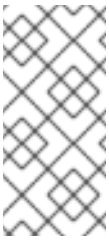
2.5.29.28

Criticality

PKIX requires that this extension be critical if it exists.

Parameters

Table B.45. IssuingDistributionPoint Configuration Parameters

Parameter	Description
enable	Sets whether the extension is enabled; the default is disabled.
critical	Marks the extension as critical, the default, or noncritical.
pointType	Specifies the type of the issuing distribution point from the following: • **directoryName** specifies that the type is an X.500 directory name. • **URI** specifies that the type is a uniform resource indicator. • **RelativeToIssuer** specifies that the type is a relative distinguished name (RDN), which represents a single node of a DN, such as **ou=Engineering**.
pointName	Gives the name of the issuing distribution point. The name of the distribution point depends on the value specified for the **pointType** parameter. • For **directoryName**, the name must be an X.500 name. For example, *cn=CRLCentral,ou=Research Dept,o=Example Corporation,c=US*. • For **URIName**, the name must be a URI that is an absolute pathname and specifies the host. For example, *http://testCA.example.com/get/crls/here/*.  NOTE The CRL may be stored in the directory entry corresponding to the CRL issuing point, which may be different than the directory entry of the CA.

Parameter	Description
onlySomeReasons	Specifies the reason codes associated with the distribution point. Permissible values are a combination of reason codes (**unspecified**, **keyCompromise**, **cACompromise**, **affiliationChanged**, **superseded**, **cessationOfOperation**, **certificateHold**, and **removeFromCRL**) separated by commas. Leave the field blank if the distribution point contains revoked certificates with all reason codes (default).
onlyContainsCACerts	Specifies that the distribution point contains user certificates only if set. By default, this is not set, which means the distribution point contains all types of certificates.
indirectCRL	Specifies that the distribution point contains an indirect CRL; by default, this is not selected.

B.4.2.2. CRL Entry Extensions

The sections that follow lists the CRL entry extension types that are defined as part of the Internet X.509 v3 Public Key Infrastructure proposed standard. All of these extensions are noncritical.

B.4.2.2.1. certificateIssuer

The Certificate Issuer extension identifies the certificate issuer associated with an entry in an indirect CRL.

This extension is used only with indirect CRLs, which are not supported by the Certificate System.

OID

2.5.29.29

B.4.2.2.2. invalidityDate

The Invalidity Date extension provides the date on which the private key was compromised or that the certificate otherwise became invalid.

OID

2.5.29.24

Parameters

Table B.46. InvalidityDate Configuration Parameters

Parameter	Description
enable	Sets whether the extension rule is enabled or disabled. By default, this is enabled.
critical	Marks the extension as critical or noncritical; by default, this is noncritical.

B.4.2.2.3. CRLReason

The Reason Code extension identifies the reason for certificate revocation.

OID

2.5.29.21

Parameters

Table B.47. CRLReason Configuration Parameters

Parameter	Description
enable	Sets whether the extension rule is enabled or disabled. By default, this is enabled.
critical	Marks the extension as critical or noncritical. By default, this is noncritical.

B.4.3. Netscape-Defined Certificate Extensions Reference

Netscape defined certain certificate extensions for its products. Some of the extensions are now obsolete, and others have been superseded by the extensions defined in the X.509 proposed standard. All Netscape extensions should be tagged as noncritical, so that their presence in a certificate does not make that certificate incompatible with other clients.

B.4.3.1. netscape-cert-type

The Netscape Certificate Type extension can be used to limit the purposes for which a certificate can be used. It has been replaced by the X.509 v3 extensions Section B.3.6, “extKeyUsage” and Section B.3.3, “basicConstraints”.

If the extension exists in a certificate, it limits the certificate to the uses specified in it. If the extension is not present, the certificate can be used for all applications, except for object signing.

The value is a bit-string, where the individual bit positions, when set, certify the certificate for particular uses as follows:

- bit 0: SSL Client certificate
- bit 1: SSL Server certificate

- bit 2: S/MIME certificate
- bit 3: Object Signing certificate
- bit 4: reserved
- bit 5: SSL CA certificate
- bit 6: S/MIME CA certificate
- bit 7: Object Signing CA certificate

OID

2.16.840.1.113730.1.1

B.4.3.2. netscape-comment

The value of this extension is an IA5String. It is a comment that can be displayed to the user when the certificate is viewed.

OID

2.16.840.1.113730.13

APPENDIX C. PUBLISHING MODULE REFERENCE

Several publisher, mapper, and rule modules are configured by default with the Certificate Manager.

- Section C.1, "Publisher Plug-in Modules"
- Section C.2, "Mapper Plug-in Modules "
- Section C.3, "Rule Instances"

C.1. PUBLISHER PLUG-IN MODULES

This section describes the publisher modules provided for the Certificate Manager. The modules are used by the Certificate Manager to enable and configure specific publisher instances.

- Section C.1.1, "FileBasedPublisher"
- Section C.1.2, "LdapCaCertPublisher"
- Section C.1.3, "LdapUserCertPublisher"
- Section C.1.4, "LdapCrlPublisher"
- Section C.1.5, "LdapDeltaCrlPublisher"
- Section C.1.6, "LdapCertificatePairPublisher"
- Section C.1.7, "OCSPPublisher"

C.1.1. FileBasedPublisher

The **FileBasedPublisher** plug-in module configures a Certificate Manager to publish certificates and CRLs to file. This plug-in can publish base-64 encoded files, DER-encoded files, or both, depending on the checkboxes selected when the publisher is configured. The certificate and CRL content can be viewed by converting the files using the **PrettyPrintCert** and **PrettyPrintCRL** tools. For details on viewing the content in base-64 and DER-encoded certificates and CRLs, see Section 9.11, "Viewing Certificates and CRLs Published to File".

By default, the Certificate Manager does not create an instance of the **FileBasedPublisher** module.

Table C.1. FileBasedPublisher Configuration Parameters

Parameter	Description
Publisher ID	Specifies a name for the publisher, an alphanumeric string with no spaces. For example, **PublishCertsToFile**.
directory	Specifies the complete path to the directory to which the Certificate Manager creates the files; the path can be an absolute path or can be relative to the Certificate System instance directory. For example, **/export/CS/certificates**.

C.1.2. LdapCaCertPublisher

The **LdapCaCertPublisher** plug-in module configures a Certificate Manager to publish or unpublish a CA certificate to the **caCertificate;binary** attribute of the CA's directory entry.

The module converts the object class of the CA's entry to **pkiCA** or **certificationAuthority**, if it is not used already. Similarly, it also removes the **pkiCA** or **certificationAuthority** object class when unpublishing if the CA has no other certificates.

During installation, the Certificate Manager automatically creates an instance of the **LdapCaCertPublisher** module for publishing the CA certificate to the directory.

Table C.2. LdapCaCertPublisher Configuration Parameters

Parameter	Description
caCertAttr	Specifies the LDAP directory attribute to publish the CA certificate. This must be **caCertificate;binary**.
caObjectClass	Specifies the object class for the CA's entry in the directory. This must be **pkiCA** or **certificationAuthority**.

C.1.3. LdapUserCertPublisher

The **LdapUserCertPublisher** plug-in module configures a Certificate Manager to publish or unpublish a user certificate to the **userCertificate;binary** attribute of the user's directory entry.

This module is used to publish any end-entity certificate to an LDAP directory. Types of end-entity certificates include SSL client, S/MIME, SSL server, and OCSP responder.

During installation, the Certificate Manager automatically creates an instance of the **LdapUserCertPublisher** module for publishing end-entity certificates to the directory.

Table C.3. LdapUserCertPublisher Configuration Parameters

Parameter	Description
certAttr	Specifies the directory attribute of the mapped entry to which the Certificate Manager should publish the certificate. This must be **userCertificate;binary**.

C.1.4. LdapCrlPublisher

The **LdapCrlPublisher** plug-in module configures a Certificate Manager to publish or unpublish the CRL to the **certificateRevocationList;binary** attribute of a directory entry.

During installation, the Certificate Manager automatically creates an instance of the **LdapCrlPublisher** module for publishing CRLs to the directory.

Table C.4. LdapCrlPublisher Configuration Parameters

Parameter	Description
crlAttr	Specifies the directory attribute of the mapped entry to which the Certificate Manager should publish the CRL. This must be **certificateRevocationList;binary**.

C.1.5. LdapDeltaCrlPublisher

The **LdapDeltaCrlPublisher** plug-in module configures a Certificate Manager to publish or unpublish a delta CRL to the **deltaRevocationList** attribute of a directory entry.

During installation, the Certificate Manager automatically creates an instance of the **LdapDeltaCrlPublisher** module for publishing CRLs to the directory.

Table C.5. LdapDeltaCrlPublisher Configuration Parameters

Parameter	Description
crlAttr	Specifies the directory attribute of the mapped entry to which the Certificate Manager should publish the delta CRL. This must be **deltaRevocationList;binary**.

C.1.6. LdapCertificatePairPublisher

The **LdapCertificatePairPublisher** plug-in module configures a Certificate Manager to publish or unpublish a cross-signed certificate to the **crossCertPair;binary** attribute of the CA's directory entry.

The module also converts the object class of the CA's entry to a **pkiCA** or **certificationAuthority**, if it is not used already. Similarly, it also removes the **pkiCA** or **certificationAuthority** object class when unpublishing if the CA has no other certificates.

During installation, the Certificate Manager automatically creates an instance of the **LdapCertificatePairPublisher** module named **LdapCrossCertPairPublisher** for publishing the cross-signed certificates to the directory.

Table C.6. LdapCertificatePairPublisher Parameters

Parameter	Description
crossCertPairAttr	Specifies the LDAP directory attribute to publish the CA certificate. This must be **crossCertificatePair;binary**.
caObjectClass	Specifies the object class for the CA's entry in the directory. This must be **pkiCA** or **certificationAuthority**.

C.1.7. OCSPPublisher

The **OCSPPublisher** plug-in module configures a Certificate Manager to publish its CRLs to an Online Certificate Status Manager.

The Certificate Manager does not create any instances of the **OCSPPublisher** module at installation.

Table C.7. OCSPPublisher Parameters

Parameter	Description
host	Specifies the fully qualified hostname of the Online Certificate Status Manager.
port	Specifies the port number on which the Online Certificate Status Manager is listening to the Certificate Manager. This is the Online Certificate Status Manager's SSL port number.
path	Specifies the path for publishing the CRL. This must be the default path, **/ocsp/agent/ocsp/addCRL**.
enableClientAuth	Sets whether to use client (certificate-based) authentication to access the OCSP service.
nickname	Gives the nickname of the certificate in the OCSP service's database to use for client authentication. This is only used if the **enableClientAuth** option is set to true.

C.2. MAPPER PLUG-IN MODULES

This section describes the mapper plug-in modules provided for the Certificate Manager. These modules configure a Certificate Manager to enable and configure specific mapper instances.

The available mapper plug-in modules include the following:

- Section C.2.1, “LdapCaSimpleMap”
- Section C.2.2, “LdapDNExactMap”
- Section C.2.3, “LdapSimpleMap”
- Section C.2.4, “LdapSubjAttrMap”
- Section C.2.5, “LdapDNCompsMap”

C.2.1. LdapCaSimpleMap

The **LdapCaSimpleMap** plug-in module configures a Certificate Manager to create an entry for the CA in an LDAP directory automatically and then map the CA's certificate to the directory entry by formulating the entry's DN from components specified in the certificate request, certificate subject name, certificate extension, and attribute variable assertion (AVA) constants. For more information on AVAs, check the directory documentation.

The CA certificate mapper specifies whether to create an entry for the CA, to map the certificate to an existing entry, or to do both.

If a CA entry already exists in the publishing directory and the value assigned to the **dnPattern** parameter of this mapper is changed, but the **uid** and **o** attributes are the same, the mapper fails to create the second CA entry. For example, if the directory already has a CA entry for **uid=CA,ou=Marketing,o=example.com** and a mapper is configured to create another CA entry with **uid=CA,ou=Engineering,o=example.com**, the operation fails.

The operation may fail because the directory has the *UID Uniqueness* plug-in set to a specific base DN. This setting prevents the directory from having two entries with the same UID under that base DN. In this example, it prevents the directory from having two entries under **o=example.com** with the same UID, **CA**.

If the mapper fails to create a second CA entry, check the base DN to which the UID Uniqueness plug-in is set, and check if an entry with the same UID already exists in the directory. If necessary, adjust the mapper setting, remove the old CA entry, comment out the plug-in, or create the entry manually.

During installation, the Certificate Manager automatically creates two instances of the CA certificate mapper module. The mappers are named as follows:

- **LdapCrlMap** for CRLs (see Section C.2.1.2, "LdapCrlMap")
- **LdapCaCertMap** for CA certificates (see Section C.2.1.1, "LdapCaCertMap").

Table C.8. LdapCaSimpleMap Configuration Parameters

Parameter	Description
createCAEntry	Creates a CA's entry, if selected (default). If selected, the Certificate Manager first attempts to create an entry for the CA in the directory. If the Certificate Manager succeeds in creating the entry, it then attempts to publish the CA's certificate to the entry. If this is not selected, the entry must already be present in order to publish to it.

Parameter	Description
dnPattern	Specifies the DN pattern the Certificate Manager should use to construct to search for the CA's entry in the publishing directory. The value of **dnPattern** can be a list of AVAs separated by commas. An AVA can be a variable, such as **cn=$subj.cn**, that the Certificate Manager can derive from the certificate subject name or a constant, such as **o=Example Corporation**. If the CA certificate does not have the **cn** component in its subject name, adjust the CA certificate mapping DN pattern to reflect the DN of the entry in the directory where the CA certificate is to be published. For example, if the CA certificate subject DN is **o=Example Corporation** and the CA's entry in the directory is **cn=Certificate Authority, o=Example Corporation**, the pattern is **cn=Certificate Authority, o=$subj.o**. ● Example 1: **uid=CertMgr, o=Example Corporation** ● Example 2: **cn=$subj.cn,ou=$subj.ou,o=$subj.o,c=US** ● Example 3: **uid=$req.HTTP_PARAMS.uid, e=$ext.SubjectAlternativeName.RFC822Name,ou=$subj.ou** In the above examples, **$req** takes the attribute from the certificate request, **$subj** takes the attribute from the certificate subject name, and **$ext** takes the attribute from the certificate extension.

C.2.1.1. LdapCaCertMap

The **LdapCaCertMap** mapper is an instance of the **LdapCaSimpleMap** module. The Certificate Manager automatically creates this mapper during installation.

This mapper creates an entry for the CA in the directory and maps the CA certificate to the CA's entry in the directory.

By default, the mapper is configured to create an entry for the CA in the directory, The default DN pattern for locating the CA's entry is as follows:

```
uid=$subj.cn,ou=people,o=$subj.o
```

C.2.1.2. LdapCrlMap

The **LdapCrlMap** mapper is an instance of the **LdapCaSimpleMap** module. The Certificate Manager automatically creates this mapper during installation.

This mapper creates an entry for the CA in the directory and maps the CRL to the CA's entry in the directory.

By default, the mapper is configured to create an entry for the CA in the directory. The default DN pattern for locating the CA's entry is as follows:

```
uid=$subj.cn,ou=people,o=$subj.o
```

C.2.2. LdapDNExactMap

The **LdapDNExactMap** plug-in module configures a Certificate Manager to map a certificate to an LDAP directory entry by searching for the LDAP entry DN that matches the certificate subject name. To use this mapper, each certificate subject name must exactly match a DN in a directory entry. For example, if the certificate subject name is **uid=jdoe, o=Example Corporation, c=US**, when searching the directory for the entry, the Certificate Manager only searches for an entry with the DN **uid=jdoe, o=Example Corporation, c=US**.

If no matching entries are found, the server returns an error and does not publish the certificate.

This mapper does not require any values for any parameters because it obtains all values from the certificate.

C.2.3. LdapSimpleMap

The **LdapSimpleMap** plug-in module configures a Certificate Manager to map a certificate to an LDAP directory entry by deriving the entry's DN from components specified in the certificate request, certificate's subject name, certificate extension, and attribute variable assertion (AVA) constants. For more information on AVAs, see the directory documentation.

By default, the Certificate Manager uses mapper rules that are based on the simple mapper. During installation, the Certificate Manager automatically creates an instance of the simple mapper module, named **LdapUserCertMap**. The default mapper maps various types of end-entity certificates to their corresponding directory entries.

The simple mapper requires one parameter, **dnPattern**. The value of **dnPattern** can be a list of AVAs separated by commas. An AVA can be a variable, such as **uid=$subj.UID**, or a constant, such as **o=Example Corporation**.

- Example 1: **uid=CertMgr, o=Example Corporation**
- Example 2: **cn=$subj.cn,ou=$subj.ou,o=$subj.o,c=US**
- Example 3: uid=**$req.HTTP_PARAMS.uid, e=$ext.SubjectAlternativeName.RFC822Name,ou=$subj.ou**

In the examples, **$req** takes the attribute from the certificate request, **$subj** takes the attribute from the certificate subject name, and **$ext** takes the attribute from the certificate extension.

C.2.4. LdapSubjAttrMap

The **LdapSubjAttrMap** plug-in module configures a Certificate Manager to map a certificate to an LDAP directory entry using a configurable LDAP attribute. To use this mapper, the directory entries must include the specified LDAP attribute.

This mapper requires the exact pattern of the subject DN because the Certificate Manager searches the directory for the attribute with a value that exactly matches the entire subject DN. For example, if the specified LDAP attribute is **certSubjectDN** and the certificate subject name is **uid=jdoe, o=Example**

Corporation, c=US, the Certificate Manager searches the directory for entries that have the attribute **certSubjectDN=uid=jdoe, o=Example Corporation, c=US**.

If no matching entries are found, the server returns an error and writes it to the log.

Table C.9, "LdapSubjAttrMap Parameters" describes these parameters.

Table C.9. LdapSubjAttrMap Parameters

Parameter	Description
certSubjNameAttr	Specifies the name of the LDAP attribute that contains a certificate subject name as its value. The default is **certSubjectName**, but this can be configured to any LDAP attribute.
searchBase	Specifies the base DN for starting the attribute search. The permissible value is a valid DN of an LDAP entry, such as **o=example.com, c=US**.

C.2.5. LdapDNCompsMap

The **LdapDNCompsMap** plug-in module implements the DN components mapper. This mapper maps a certificate to an LDAP directory entry by constructing the entry's DN from components, such as **cn**, **ou**, **o**, and **c**, specified in the certificate subject name, and then uses it as the search DN to locate the entry in the directory. The mapper locates the following entries:

- The CA's entry in the directory for publishing the CA certificate and the CRL.
- End-entity entries in the directory for publishing end-entity certificates.

The mapper takes DN components to build the search DN. The mapper also takes an optional root search DN. The server uses the DN components to form an LDAP entry to begin a subtree search and the filter components to form a search filter for the subtree. If none of the DN components are configured, the server uses the base DN for the subtree. If the base DN is null and none of the DN components match, an error is returned. If none of the DN components and filter components match, an error is returned. If the filter components are null, a base search is performed.

Both the **DNComps** and **filterComps** parameters accept valid DN components or attributes separated by commas. The parameters do not accept multiple entries of an attribute; for example, **filterComps** can be set to **cn,ou** but not to **cn,ou2,ou1**. To create a filter with multiple instances of the same attribute, such as if directory entries contain multiple **ou** s, modify the source code for the **LdapDNCompsMap** module.

The following components are commonly used in DNs:

- **uid** represents the user ID of a user in the directory.
- **cn** represents the common name of a user in the directory.
- **ou** represents an organizational unit in the directory.
- **o** represents an organization in the directory.
- **l** represents a locality (city).

- **st** represents a state.
- **c** represents a country.

For example, the following DN represents the user named Jane Doe who works for the Sales department at Example Corporation, which is located in Mountain View, California, United States:

```
cn=Jane Doe, ou=Sales, o=Example Corporation, l=Mountain View, st=California, c=US
```

The Certificate Manager can use some or all of these components (**cn**, **ou**, **o**, **l**, **st**, and **c**) to build a DN for searching the directory. When creating a mapper rule, these components can be specified for the server to use to build a DN; that is, components to match attributes in the directory. This is set through the **dnComps** parameter.

For example, the components **cn**, **ou**, **o**, and **c** are set as values for the **dnComps** parameter. To locate Jane Doe's entry in the directory, the Certificate Manager constructs the following DN by reading the DN attribute values from the certificate, and uses the DN as the base for searching the directory:

```
cn=Jane Doe, ou=Sales, o=Example Corporation, c=US
```

- A subject name does not need to have all of the components specified in the **dnComps** parameter. The server ignores any components that are not part of the subject name, such as **l** and **st** in this example.
- Unspecified components are not used to build the DN. In the example, if the **ou** component is not included, the server uses this DN as the base for searching the directory:

  ```
  cn=Jane Doe, o=Example Corporation, c=US
  ```

For the **dnComps** parameter, enter those DN components that the Certificate Manager can use to form the LDAP DN exactly. In certain situations, however, the subject name in a certificate may match more than one entry in the directory. Then, the Certificate Manager might not get a single, distinct matching entry from the DN. For example, the subject name **cn=Jane Doe, ou=Sales, o=Example Corporation, c=US** might match two users with the name Jane Doe in the directory. If that occurs, the Certificate Manager needs additional criteria to determine which entry corresponds to the subject of the certificate.

To specify the components the Certificate Manager must use to distinguish between different entries in the directory, use the **filterComps** parameter; for details, see Table C.10, "LdapDNCompsMap Configuration Parameters". For example, if **cn**, **ou**, **o**, and **c** are values for the **dnComps** parameter, enter **l** for the **filterComps** parameter only if the **l** attribute can be used to distinguish between entries with identical **cn**, **ou**, **o**, and **c** values.

If the two Jane Doe entries are distinguished by the value of the **uid** attribute - one entry's **uid** is **janedoe1**, and the other entry's **uid** is **janedoe2** - the subject names of certificates can be set to include the **uid** component.

NOTE

The **e**, **l**, and **st** components are not included in the standard set of certificate request forms provided for end entities. These components can be added to the forms, or the issuing agents can be required to insert these components when editing the subject name in the certificate issuance forms.

C.2.5.1. Configuration Parameters of LdapDNCompsMap

With this configuration, a Certificate Manager maps its certificates with the ones in the LDAP directory by using the **dnComps** values to form a DN and the **filterComps** values to form a search filter for the subtree.

- If the formed DN is null, the server uses the **baseDN** value for the subtree. If both the formed DN and base DN are null, the server logs an error.
- If the filter is null, the server uses the **baseDN** value for the search. If both the filter and base DN are null, the server logs an error.

Table C.10, "LdapDNCompsMap Configuration Parameters" describes these parameters.

Table C.10. LdapDNCompsMap Configuration Parameters

Parameter	Description
baseDN	Specifies the DN to start searching for an entry in the publishing directory. If the **dnComps** field is blank, the server uses the base DN value to start its search in the directory.
dnComps	Specifies where in the publishing directory the Certificate Manager should start searching for an LDAP entry that matches the CA's or the end entity's information. For example, if **dnComps** uses the **o** and **c** attributes of the DN, the server starts the search from the **o=***org*, **c=***country* entry in the directory, where *org* and *country* are replaced with values from the DN in the certificate. If the **dnComps** field is empty, the server checks the **baseDN** field and searches the directory tree specified by that DN for entries matching the filter specified by **filterComps** parameter values. The permissible values are valid DN components or attributes separated by commas.

Parameter	Description
filterComps	Specifies components the Certificate Manager should use to filter entries from the search result. The server uses the **filterComps** values to form an LDAP search filter for the subtree. The server constructs the filter by gathering values for these attributes from the certificate subject name; it uses the filter to search for and match entries in the LDAP directory. If the server finds more than one entry in the directory that matches the information gathered from the certificate, the search is successful, and the server optionally performs a verification. For example, if **filterComps** is set to use the email and user ID attributes (**filterComps=e,uid**), the server searches the directory for an entry whose values for email and user ID match the information gathered from the certificate. The permissible values are valid directory attributes in the certificate DN separated by commas. The attribute names for the filters need to be attribute names from the certificate, not from ones in the LDAP directory. For example, most certificates have an **e** attribute for the user's email address; LDAP calls that attribute **mail**.

C.3. RULE INSTANCES

This section discusses the rule instances that have been set.

C.3.1. LdapCaCertRule

The **LdapCaCertRule** can be used to publish CA certificates to an LDAP directory.

Table C.11. LdapCaCert Rule Configuration Parameters

Parameter	Value	Description
type	**cacert**	Specifies the type of certificate that will be published.
predicate		Specifies a predicate for the publisher.
enable	**yes**	Enables the rule.
mapper	**LdapCaCertMap**	Specifies the mapper used with the rule. See Section C.2.1.1, "LdapCaCertMap" for details on the mapper.

Parameter	Value	Description
publisher	**LdapCaCertPublisher**	Specifies the publisher used with the rule. See Section C.1.2, "LdapCaCertPublisher" for details on the publisher.

C.3.2. LdapXCertRule

The **LdapXCertRule** is used to publish cross-pair certificates to an LDAP directory.

Table C.12. LdapXCert Rule Configuration Parameters

Parameter	Value	Description
type	**xcert**	Specifies the type of certificate that will be published.
predicate		Specifies a predicate for the publisher.
enable	**yes**	Enables the rule.
mapper	**LdapCaCertMap**	Specifies the mapper used with the rule. See Section C.2.1.1, "LdapCaCertMap" for details on the mapper.
publisher	**LdapCrossCertPairPublisher**	Specifies the publisher used with the rule. See Section C.1.6, "LdapCertificatePairPublisher" for details on this publisher.

C.3.3. LdapUserCertRule

The **LdapUserCertRule** is used to publish user certificates to an LDAP directory.

Table C.13. LdapUserCert Rule Configuration Parameters

Parameter	Value	Description
type	**certs**	Specifies the type of certificate that will be published.
predicate		Specifies a predicate for the publisher.
enable	**yes**	Enables the rule.

Parameter	Value	Description
mapper	**LdapUserCertMap**	Specifies the mapper used with the rule. See Section C.2.3, "LdapSimpleMap" for details on the mapper.
publisher	**LdapUserCertPublisher**	Specifies the publisher used with the rule. See Section C.1.3, "LdapUserCertPublisher" for details on the publisher.

C.3.4. LdapCRLRule

The **LdapCRLRule** is used to publish CRLs to an LDAP directory.

Table C.14. LdapCRL Rule Configuration Parameters

Parameter	Value	Description
type	**crl**	Specifies the type of certificate that will be published.
predicate		Specifies a predicate for the publisher.
enable	**yes**	Enables the rule.
mapper	**LdapCrlMap**	Specifies the mapper used with the rule. See Section C.2.1.2, "LdapCrlMap" for details on the mapper.
publisher	**LdapCrlPublisher**	Specifies the publisher used with the rule. See Section C.1.4, "LdapCrlPublisher" for details on the publisher.

APPENDIX D. ACL REFERENCE

This section describes what each resource controls, lists the possible operations describing the outcome of those operations, and provides the default ACIs for each ACL resource defined. Each subsystem contains only those ACLs that are relevant to that subsystem.

D.1. ABOUT ACL CONFIGURATION FILES

Access control is the method to set rules on who can access part of a server and the operations that user can perform. The four subsystems which depend on the LDAP directory service and use a Java console — the CA, KRA, OCSP, and TKS — all implement LDAP-style access control to access their resources. These access control lists (ACL) are located in the **/var/lib/pki/*instance_name*/conf/*subsystem*/acl.ldif** file.

NOTE

This section provides only a very brief overview of access control concepts. Access control is described in much more detail in the *Managing Access Control* chapter in the *Red Hat Directory Server Administration Guide* .

The Certificate System ACL files are LDIF files that are loaded by the internal database. The individual ACLs are defined as ***resourceACLS*** attributes which identify the area of the subsystem being protected and then a list of all of the specific access controls being set.

```
resourceACLS: class_name:all rights: allow|deny (rights) type=target description
```

Each rule which allows or denies access to a resource is called an *access control instruction* (ACI). (The sum of all of the ACIs for a resource is an access control list.) Before defining the actual ACI, the ACL attribute is first applied to a specific plug-in class used by the Certificate System subsystem. This focuses each ACL to a specific function performed by the subsystem, providing both more security for the instance and better control over applying ACLs.

Example D.1. Default ACL to List Certificate Profiles

```
resourceACLS: certServer.ca.profiles:list:allow (list) group="Certificate Manager
Agents":Certificate Manager agents may list profiles
```

Because each subsystem (CA, KRA, OCSP, and TKS) has different resources for its operations, each subsystem instance has its own **acl.ldif** file and its own defined ACLs.

Each ACI defines what access or behavior can be done (the *right*) and who the ACI applies to (the *target*). The basic format of an ACI is, then:

```
allow|deny (rights) user|group
```

Rights are types of operations that the ACI allows a user to perform. For LDAP ACIs, there is a relatively limited list of rights to directory entries, like search, read, write, and delete. The Certificate System uses additional rights that cover common PKI tasks, like revoke, submit, and assign.

If an operation is not explicitly allowed in an ACI, then it is implicitly denied. If an operation is explicitly denied in one ACI, then it trumps any ACI which explicitly allows it. Deny rules are always superior to allow rules to provide additional security.

Each ACI has to apply to specific users or groups. This is set using a couple of common conditions, usually **user=** or **group=**, though there are other options, like **ipaddress=** which defines client-based access rather than entry-based access. If there is more than one condition, the conditions can be composed using the double pipe (||) operator, signifying logical disjunction ("or"), and the double ampersand (&&) operator, signifying logical conjunction ("and"). For example, **group="group1" || "group2"**.

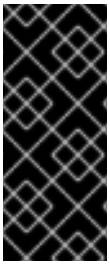
Each area of the ***resourceACLS*** attribute value is defined in Table D.1, "Sections of the ACL Attribute Value".

Table D.1. Sections of the ACL Attribute Value

Value	Description
class_name	The plug-in class to which the ACI is applied.
all operations	The list of every operation covered in the ACI definition. There can be multiple operations in a single ACI and multiple ACIs in a single ***resourceACLS*** attribute.
allow|deny	Whether the action is being allowed for the target user or group or denied to the target user or group.
(*operations*)	The operations being allowed or denied.
type=target	The target to identify who this applies to. This is commonly a user (such as **user=***"name"*) or a group (**group=***"group"*). If there is more than one condition, the conditions can be composed using the double pipe (||) operator (logical "or") and the double ampersand (&&) operator (logical "and"). For example, **group="group1" || "group2"**.
description	A description of what the ACL is doing.

D.2. COMMON ACLS

This section covers the default access control configuration that is common for all four subsystem types. These access control rules manage access to basic and common configuration settings, such as logging and adding users and groups.



IMPORTANT

These ACLs are common in that the same ACLs occur in each subsystem instance's **acl.ldif** file. These are not *shared* ACLs in the sense that the configuration files or settings are held in common by all subsystem instances. As with all other instance configuration, these ACLs are maintained independently of other subsystem instances, in the instance-specific **acl.ldif** file.

D.2.1. certServer.acl.configuration

Controls operations to the ACL configuration. The default configuration is:

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators"
```

Table D.2. certServer.acl.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View ACL resources and list ACL resources, ACL listing evaluators, and ACL evaluator types.	Allow	Administrators Agents Auditors
modify	Add, delete, and update ACL evaluators.	Allow	Administrators

D.2.2. certServer.admin.certificate

Controls which users can import a certificate through a Certificate Manager. By default, this operation is allowed to everyone. The default configuration is:

```
allow (import) user="anybody"
```

NOTE

This entry is associated with the CA administration web interface which is used to configure the instance. This ACL is only available during instance configuration and is unavailable after the CA is running.

Table D.3. certServer.admin.certificate ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups

Operations	Description	Allow/Deny Access	Targeted Users/Groups
import	Import a CA administrator certificate, and retrieve certificates by serial number.	Allow	Anyone

D.2.3. certServer.auth.configuration

Controls operations on the authentication configuration.

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators
```

Table D.4. certServer.auth.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View authentication plug-ins, authentication type, configured authentication manager plug-ins, and authentication instances. List authentication manager plug-ins and authentication manager instances.	Allow	Administrators Agents Auditors
modify	Add or delete authentication plug-ins and authentication instances. Modify authentication instances.	Allow	Administrators

D.2.4. certServer.clone.configuration

Controls who can read and modify the configuration information used in cloning. The default setting is:

```
allow (modify,read) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" || group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators"
```

Table D.5. certServer.clone.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View original instance configuration.	Allow	Enterprise Administrators
modify	Modify original instance configuration.	Allow	Enterprise Administrators

D.2.5. certServer.general.configuration

Controls access to the general configuration of the subsystem instance, including who can view and edit the CA's settings.

```
allow (read) group="Administrators" || group="Auditors" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents";allow (modify) group="Administrators"
```

Table D.6. certServer.general.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the operating environment, LDAP configuration, SMTP configuration, server statistics, encryption, token names, subject name of certificates, certificate nicknames, all subsystems loaded by the server, CA certificates, and all certificates for management.	Allow	Administrators Agents Auditors

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Modify the settings for the LDAP database, SMTP, and encryption. Issue import certificates, install certificates, trust and untrust CA certificates, import cross-pair certificates, and delete certificates. Perform server restart and stop operations. Log in all tokens and check token status. Run self-tests on demand. Get certificate information. Process the certificate subject name. Validate the certificate subject name, certificate key length, and certificate extension.	Allow	Administrators

D.2.6. certServer.log.configuration

Controls access to the log configuration for the Certificate Manager, including changing the log settings.

```
allow (read) group="Administrators" || group="Auditors" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents";allow (modify) group="Administrators"
```

Table D.7. certServer.log.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View log plug-in information, log plug-in configuration, and log instance configuration. List log plug-ins and log instances (excluding NTEventLog).	Allow	Administrators Agents Auditors
modify	Add and delete log plug-ins and log instances. Modify log instances, including log rollover parameters and log level.	Allow	Administrators

D.2.7. certServer.log.configuration.fileName

Restricts access to change the file name of a log for the instance.

```
allow (read) group="Administrators" || group="Auditors" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents";deny (modify) user=anybody
```

Table D.8. certServer.log.configuration.fileName ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the value of the **fileName** parameter for a log instance.	Allow	Administrators Agents Auditors
modify	Change the value of the **fileName** parameter for a log instance.	Deny	Anyone

D.2.8. certServer.log.content.system

Controls who can view the instance's logs.

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors"
```

Table D.9. certServer.log.content.system ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View log content. List all logs.	Allow	Administrators Agents Auditors

D.2.9. certServer.log.content.signedAudit

Controls who has access to the signed audit logs. The default setting is:

```
allow (read) group="Auditors"
```

Table D.10. certServer.log.content.signedAudit ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View log content. List logs.	Allow	Auditors

D.2.10. certServer.registry.configuration

Controls access to the administration registry, the file that is used to register plug-in modules. Currently, this is only used to register certificate profile plug-ins.

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators"
```

Table D.11. certServer.registry.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the administration registry, supported policy constraints, profile plug-in configuration, and the list of profile plug-ins.	Allow	Administrators Agents Auditors
modify	Register individual profile implementation plug-ins.	Allow	Administrators

D.3. CERTIFICATE MANAGER-SPECIFIC ACLS

This section covers the default access control configuration attributes which are set specifically for the Certificate Manager. The CA ACL configuration also includes all of the common ACLs listed in Section D.2, “Common ACLs”.

There are access control rules set for each of the CA's interfaces (administrative console and agents and end-entities services pages) and for common operations like listing and downloading certificates.

D.3.1. certServer.admin.ocsp

Limits access to the Certificate Manager's OCSP configuration to members of the enterprise OCSP administrators group.

```
allow (modify,read) group="Enterprise OCSP Administrators"
```

Table D.12. certServer.admin.ocsp ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Modify the OCSP configuration, OCSP stores configuration, and default OCSP store.	Allow	Enterprise OCSP Administrators
read	Read the OCSP configuration.	Allow	Enterprise OCSP Administrators

D.3.2. certServer.ca.certificate

Controls basic management operations for certificates in the agents services interface, including importing and revoking certificates. The default configuration is:

```
allow (import,unrevoke,revoke,read) group="Certificate Manager Agents"
```

Table D.13. certServer.ca.certificate ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
import	Retrieve a certificate by serial number.	Allow	Certificate Manager Agents
unrevoke	Change the status of a certificate from revoked.	Allow	Certificate Manager Agents
revoke	Change the status of a certificate to revoked.	Allow	Certificate Manager Agents
read	Retrieve certificates based on the request ID, and display certificate details based on the request ID or serial number.	Allow	Certificate Manager Agents

D.3.3. certServer.ca.certificates

Controls operations for listing or revoking certificates through the agent services interface. The default configuration is:

```
allow (revoke,list) group="Certificate Manager Agents"|| group="Registration Manager Agents"
```

Table D.14. certServer.ca.certificates ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
revoke	Revoke a certificates, or approve certificate revocation requests. Revoke a certificate from the TPS. Prompt users for additional data about a revocation request.	Allow	Certificate Manager Agents Registration Manager Agents
list	List certificates based on a search. Retrieve details about a range of certificates based on a range of serial numbers.	Allow	Certificate Manager Agents Registration Manager Agents

D.3.4. certServer.ca.configuration

Controls operations on the general configuration for a Certificate Manager. The default configuration is:

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators"
```

Table D.15. certServer.ca.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View CRL plug-in information, general CA configuration, CA connector configuration, CRL issuing points configuration, CRL profile configuration, request notification configuration, revocation notification configuration, request in queue notification configuration, and CRL extensions configuration. List CRL extensions configuration and CRL issuing points configuration.	Allow	Administrators Agents Auditors

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Add and delete CRL issuing points. Modify general CA settings, CA connector configuration, CRL issuing points configuration, CRL configuration, request notification configuration, revocation notification configuration, request in queue notification configuration, and CRL extensions configuration.	Allow	Administrators

D.3.5. certServer.ca.connector

Controls operations to submit requests over a special connector to the CA. The default configuration is:

```
allow (submit) group="Trusted Managers"
```

Table D.16. certServer.ca.connector ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit requests from remote trusted managers.	Allow	Trusted Managers

D.3.6. certServer.ca.connectorInfo

Controls access to the connector information to manage trusted relationships between a CA and KRA. These trust relationships are special configurations which allow a CA and KRA to automatically connect to perform key archival and recovery operations. These trust relationships are configured through special connector plug-ins.

```
allow (read) group="Enterprise KRA Administrators";allow (modify) group="Enterprise KRA Administrators" || group="Subsystem Group"
```

Table D.17. certServer.ca.connectorInfo ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read connector plug-in settings.	Allow	Enterprise KRA Administrators
modify	Modify connector plug-in settings.	Allow	Enterprise KRA Administrators Subsystem Group

D.3.7. certServer.ca.crl

Controls access to read or update CRLs through the agent services interface. The default setting is:

```
allow (read,update) group="Certificate Manager Agents"
```

Table D.18. certServer.ca.crl ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Display CRLs and get detailed information about CA CRL processing.	Allow	Certificate Manager Agents
update	Update CRLs.	Allow	Certificate Manager Agents

D.3.8. certServer.ca.directory

Controls access to the LDAP directory used for publishing certificates and CRLs.

```
allow (update) group="Certificate Manager Agents"
```

Table D.19. certServer.ca.directory ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
update	Publish CA certificates, CRLs, and user certificates to the LDAP directory.	Allow	Certificate Manager Agents

D.3.9. certServer.ca.group

Controls access to the internal database for adding users and groups for the Certificate Manager instance.

```
allow (modify,read) group="Administrators"
```

Table D.20. certServer.ca.group ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Create, edit, or delete user and group entries for the instance. Add or modify a user certificate within attributes	Allow	Administrators
read	View user and group entries for the instance.	Allow	Administrators

D.3.10. certServer.ca.ocsp

Controls the ability to access and read OCSP information, such as usage statistics, through the agent services interface.

```
allow (read) group="Certificate Manager Agents"
```

Table D.21. certServer.ca.ocsp ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Retrieve OCSP usage statistics.	Allow	Certificate Manager Agents

D.3.11. certServer.ca.profile

Controls access to certificate profile configuration in the agent services pages.

```
allow (read,approve) group="Certificate Manager Agents"
```

Table D.22. certServer.ca.profile ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the details of the certificate profiles.	Allow	Certificate Manager Agents

Operations	Description	Allow/Deny Access	Targeted Users/Groups
approve	Approve and enable certificate profiles.	Allow	Certificate Manager Agents

D.3.12. certServer.ca.profiles

Controls access to list certificate profiles in the agent services interface.

```
allow (list) group="Certificate Manager Agents"
```

Table D.23. certServer.ca.profiles ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	List certificate profiles.	Allow	Certificate Manager Agents

D.3.13. certServer.ca.registerUser

Defines which group or user can create an agent user for the instance. The default configuration is:

```
allow (modify,read) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" ||
group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators" ||
group="Enterprise TPS Administrators"
```

Table D.24. certServer.ca.registerUser ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Register a new agent.	Allow	Enterprise Administrators
read	Read existing agent information.	Allow	Enterprise Administrators

D.3.14. certServer.ca.request.enrollment

Controls how the enrollment request are handled and assigned. The default setting is:

```
allow (submit) user="anybody";allow (read,execute,assign,unassign) group="Certificate Manager
Agents"
```

Table D.25. certServer.ca.request.enrollment ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View an enrollment request.	Allow	Certificate Manager Agents
execute	Modify the approval state of a request.	Allow	Certificate Manager Agents
submit	Sumbit a request.	Allow	Anybody
assign	Assign a request to a Certificate Manager agent.	Allow	Certificate Manager Agents
unassign	Change the assignment of a request.	Allow	Certificate Manager Agents

D.3.15. certServer.ca.request.profile

Controls the handling of certificate profile-based requests. The default setting is:

```
allow (approve,read) group="Certificate Manager Agents"
```

Table D.26. certServer.ca.request.profile ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
approve	Modify the approval state of a certificate profile-based certificate request.	Allow	Certificate Manager Agents
read	View a certificate profile-based certificate request.	Allow	Certificate Manager Agents

D.3.16. certServer.ca.requests

Controls who can list certificate requests in the agents services interface.

```
allow (list) group="Certificate Manager Agents"|| group="Registration Manager Agents"
```

Table D.27. certServer.ca.requests ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	Retrieve details on a range of requests, and search for certificates using a complex filter.	Allow	Certificate Manager Agents Registration Manager Agents

D.3.17. certServer.ca.systemstatus

Controls who can view the statistics for the Certificate Manager instance.

```
allow (read) group="Certificate Manager Agents"
```

Table D.28. certServer.ca.systemstatus ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View statistics.	Allow	Certificate Manager Agents

D.3.18. certServer.ee.certchain

Controls who can access the CA certificate chain in the end-entities page.

```
allow (download,read) user="anybody"
```

Table D.29. certServer.ee.certchain ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
download	Download the CA's certificate chain.	Allow	Anyone
read	View the CA's certificate chain.	Allow	Anyone

D.3.19. certServer.ee.certificate

Controls who can access certificates, for most operations like importing or revoking certificates, through the end-entities page.

```
allow (renew,revoke,read,import) user="anybody"
```

Table D.30. certServer.ee.certificate ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
renew	Submit a request to renew an existing certificate.	Allow	Anyone
revoke	Submit a revocation request for a user certificate.	Allow	Anyone
read	Retrieve and view certificates based on the certificate serial number or request ID.	Allow	Anyone
import	Import a certificate based on serial number.	Allow	Anyone

D.3.20. certServer.ee.certificates

Controls who can list revoked certificates or submit a revocation request in the end-entities page.

```
allow (revoke,list) user="anybody"
```

Table D.31. certServer.ee.certificates ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
revoke	Submit a list of certificates to revoke.	Allow	Subject of Certificate to be Revoked must match Certificate presented to authenticate to the CA.
list	Search for certificates matching specified criteria.	Allow	Anyone

D.3.21. certServer.ee.crl

Controls access to CRLs through the end-entities page.

```
allow (read,add) user="anybody"
```

Table D.32. certServer.ee.crl ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Retrieve and view the certificate revocation list.	Allow	Anyone
add	Add CRLs to the OCSP server.	Allow	Anyone

D.3.22. certServer.ee.profile

Controls some access to certificate profiles in the end-entities page, including who can view details about a profile or submit a request through the profile.

```
allow (submit,read) user="anybody"
```

Table D.33. certServer.ee.profile ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit a certificate request through a certificate profile.	Allow	Anyone
read	Displaying details of a certificate profile.	Allow	Anyone

D.3.23. certServer.ee.profiles

Controls who can list active certificate profiles in the end-entities page.

```
allow (list) user="anybody"
```

Table D.34. certServer.ee.profiles ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	List certificate profiles.	Allow	Anyone

D.3.24. certServer.ee.request.ocsp

Controls access, based on IP address, on which clients submit OCSP requests.

```
allow (submit) ipaddress=".*"
```

Table D.35. certServer.ee.request.ocsp ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit OCSP requests.	Allow	All IP addresses

D.3.25. certServer.ee.request.revocation

Controls what users can submit certificate revocation requests in the end-entities page.

```
allow (submit) user="anybody"
```

Table D.36. certServer.ee.request.revocation ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit a request to revoke a certificate.	Allow	Anyone

D.3.26. certServer.ee.requestStatus

Controls who can view the status for a certificate request in the end-entities page.

```
allow (read) user="anybody"
```

Table D.37. certServer.ee.requestStatus ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Retrieve the status of a request and serial numbers of any certificates that have been issued against that request.	Allow	Anyone

D.3.27. certServer.job.configuration

Controls who can configure jobs for the Certificate Manager.

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators"
```

Table D.38. certServer.job.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View basic job settings, job instance settings, and job plug-in settings. List job plug-ins and job instances.	Allow	Administrators Agents Auditors
modify	Add and delete job plug-ins and job instances. Modify job plug-ins and job instances.	Allow	Administrators

D.3.28. certServer.profile.configuration

Controls access to the certificate profile configuration. The default setting is:

```
allow (read) group="Administrators" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents" || group="Auditors";allow (modify) group="Administrators"
```

Table D.39. certServer.profile.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View certificate profile defaults and constraints, input, output, input configuration, output configuration, default configuration, policy constraints configuration, and certificate profile instance configuration. List certificate profile plug-ins and certificate profile instances.	Allow	Administrators Agents Auditors
modify	Add, modify, and delete certificate profile defaults and constraints, input, output, and certificate profile instances. Add and modify default policy constraints configuration.	Allow	Administrators

D.3.29. certServer.publisher.configuration

Controls who can view and edit the publishing configuration for the Certificate Manager. The default configuration is:

```
allow (read) group="Administrators" || group="Auditors" || group="Certificate Manager Agents" || group="Registration Manager Agents" || group="Key Recovery Authority Agents" || group="Online Certificate Status Manager Agents";allow (modify) group="Administrators"
```

Table D.40. certServer.publisher.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View LDAP server destination information, publisher plug-in configuration, publisher instance configuration, mapper plug-in configuration, mapper instance configuration, rules plug-in configuration, and rules instance configuration. List publisher plug-ins and instances, rules plug-ins and instances, and mapper plug-ins and instances.	Allow	Administrators Agents Auditors
modify	Add and delete publisher plug-ins, publisher instances, mapper plug-ins, mapper instances, rules plug-ins, and rules instances. Modify publisher instances, mapper instances, rules instances, and LDAP server destination information.	Allow	Administrators

D.3.30. certServer.securitydomain.domainxml

Controls access to the security domain information maintained in a registry by the domain host Certificate Manager. The security domain configuration is directly accessed and modified by subsystem instances during configuration, so appropriate access must always be allowed to subsystems, or configuration could fail.

```
allow (read) user="anybody";allow (modify) group="Subsystem Group"
```

Table D.41. certServer.securitydomain.domainxml ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the security domain configuration.	Allow	Anybody
modify	Modify the security domain configuration by changing instance information and adding and removing instances.	Allow	Subsystem Groups Enterprise Administrators

D.4. KEY RECOVERY AUTHORITY-SPECIFIC ACLS

This section covers the default access control configuration which apply specifically to the KRA. The KRA ACL configuration also includes all of the common ACLs listed in Section D.2, “Common ACLs”.

There are access control rules set for each of the KRA's interfaces (administrative console and agents and end-entities services pages) and for common operations like listing and downloading keys.

D.4.1. certServer.job.configuration

Controls who can configure jobs for the KRA.

```
allow (read) group="Administrators" || group="Key Recovery Authority Agents" ||
group="Auditors";allow (modify) group="Administrators"
```

Table D.42. certServer.job.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View basic job settings, job instance settings, and job plug-in settings. List job plug-ins and job instances.	Allow	Administrators Agents Auditors
modify	Add and delete job plug-ins and job instances. Modify job plug-ins and job instances.	Allow	Administrators

D.4.2. certServer.kra.certificate.transport

Controls who can view the transport certificate for the KRA.

```
allow (read) user="anybody"
```

Table D.43. certServer.kra.certificate.transport ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View the transport certificate for the KRA instance.	Allow	Anyone

D.4.3. certServer.kra.configuration

Controls who can configure and manage the setup for the KRA.

```
allow (read) group="Administrators" || group="Auditors" || group="Key Recovery Authority Agents" ||
allow (modify) group="Administrators"
```

Table D.44. certServer.kra.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read the number of required recovery agent approvals.	Allow	Administrators Agents Auditors
modify	Change the number of required recovery agent approvals.	Allow	Administrators

D.4.4. certServer.kra.connector

Controls what entities can submit requests over a special connector configured on the CA to connect to the KRA. The default configuration is:

```
allow (submit) group="Trusted Managers"
```

Table D.45. certServer.kra.connector ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit a new key archival request (for non-TMS only).	Allow	Trusted Managers

D.4.5. certServer.kra.GenerateKeyPair

Controls who can submit key recovery requests to the KRA. The default configuration is:

```
allow (execute) group="Key Recovery Authority Agents"
```

Table D.46. certServer.kra.GenerateKeyPair ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Execute server-side key generation (TMS only).	Allow	KRA Agents

D.4.6. certServer.kra.getTransportCert

Controls who can submit key recovery requests to the KRA. The default configuration is:

```
allow (download) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" ||
group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators" ||
group="Enterprise TPS Administrators"
```

Table D.47. certServer.kra.getTransportCert ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
download	Retrieve KRA transport certificate.	Allow	Enterprise Administrators

D.4.7. certServer.kra.group

Controls access to the internal database for adding users and groups for the KRA instance.

```
allow (modify,read) group="Administrators"
```

Table D.48. certServer.kra.group ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Create, edit, or delete user and group entries for the instance.	Allow	Administrators
read	View user and group entries for the instance.	Allow	Administrators

D.4.8. certServer.kra.key

Controls who can access key information through viewing, recovering, or downloading keys. The default configuration is:

```
allow (read,recover,download) group="Key Recovery Authority Agents"
```

Table D.49. certServer.kra.key ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Display public information about key archival record.	Allow	KRA Agents
recover	Retrieve key information from the database to perform a recovery operation.	Allow	KRA Agents
download	Download key information through the agent services pages.	Allow	KRA Agents

D.4.9. certServer.kra.keys

Controls who can list archived keys through the agent services pages.

```
allow (list) group="Key Recovery Authority Agents"
```

Table D.50. certServer.kra.keys ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	Search for and list a range of archived keys.	Allow	KRA Agents

D.4.10. certServer.kra.registerUser

Defines which group or user can create an agent user for the instance. The default configuration is:

```
allow (modify,read) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" ||
group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators" ||
group="Enterprise TPS Administrators"
```

Table D.51. certServer.kra.registerUser ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Register a new user.	Allow	Enterprise Administrators
read	Read existing user info.	Allow	Enterprise Administrators

D.4.11. certServer.kra.request

Controls who can view key archival and recovery requests in the agents services interface.

```
allow (read) group="Key Recovery Authority Agents"
```

Table D.52. certServer.kra.request ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View a key archival or recovery request.	Allow	KRA Agents

D.4.12. certServer.kra.request.status

Controls who can view the status for a key recovery request in the end-entities page.

```
allow (read) group="Key Recovery Authority Agents"
```

Table D.53. certServer.kra.request.status ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Retrieve the status of a key recovery request in the agents services pages.	Allow	KRA Agents

D.4.13. certServer.kra.requests

Controls who can list key archival and recovery requests in the agents services interface.

```
allow (list) group="Key Recovery Authority Agents"
```

Table D.54. certServer.kra.requests ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	Retrieve details on a range of key archival and recovery requests.	Allow	KRA Agents

D.4.14. certServer.kra.systemstatus

Controls who can view the statistics for the KRA instance.

```
allow (read) group="Key Recovery Authority Agents"
```

Table D.55. certServer.kra.systemstatus ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View statistics.	Allow	KRA Agents

D.4.15. certServer.kra.TokenKeyRecovery

Controls who can submit key recovery requests for a token to the KRA. This is a common request for replacing a lost token. The default configuration is:

```
allow (submit) group="Key Recovery Authority Agents"
```

Table D.56. certServer.kra.TokenKeyRecovery ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit or initiate key recovery requests for a token recovery.	Allow	KRA Agents

D.5. ONLINE CERTIFICATE STATUS MANAGER-SPECIFIC ACLS

This section covers the default access control configuration attributes which are set specifically for the Online Certificate Status Manager. The OCSP responder's ACL configuration also includes all of the common ACLs listed in Section D.2, “Common ACLs” .

There are access control rules set for each of the OCSP's interfaces (administrative console and agents and end-entities services pages) and for common operations like listing and downloading CRLs.

D.5.1. certServer.ee.crl

Controls access to CRLs through the end-entities page.

```
allow (read) user="anybody"
```

Table D.57. certServer.ee.crl ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Retrieve and view the certificate revocation list.	Allow	Anyone

D.5.2. certServer.ee.request.ocsp

Controls access, based on IP address, on which clients submit OCSP requests.

```
allow (submit) ipaddress=".*"
```

Table D.58. certServer.ee.request.ocsp ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit OCSP requests.	Allow	All IP addresses

D.5.3. certServer.ocsp.ca

Controls who can instruct the OCSP responder. The default setting is:

```
allow (add) group="Online Certificate Status Manager Agents"
```

Table D.59. certServer.ocsp.ca ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Add	Instruct the OCSP responder to respond to OCSP requests for a new CA.	Allow	OCSP Manager Agents

D.5.4. certServer.ocsp.cas

Controls who can list, in the agent services interface, all of the Certificate Managers which publish CRLs to the Online Certificate Status Manager. The default setting is:

```
allow (list) group="Online Certificate Status Manager Agents"
```

Table D.60. certServer.ocsp.cas ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
list	Lists all of the Certificate Managers which publish CRLs to the OCSP responder.	Allow	Agents

D.5.5. certServer.ocsp.certificate

Controls who can validate the status of a certificate. The default setting is:

```
allow (validate) group="Online Certificate Status Manager Agents"
```

Table D.61. certServer.ocsp.certificate ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
validate	Verifies the status of a specified certificate.	Allow	OCSP Agents

D.5.6. certServer.ocsp.configuration

Controls who can access, view, or modify the configuration for the Certificate Manager's OCSP services. The default configuration is:

```
allow (read) group="Administrators" || group="Online Certificate Status Manager Agents" ||
group="Auditors";allow (modify) group="Administrators"
```

Table D.62. certServer.ocsp.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View OCSP plug-in information, OCSP configuration, and OCSP stores configuration. List OCSP stores configuration.	Allow	Administrators Online Certificate Status Manager Agents Auditors
modify	Modify the OCSP configuration, OCSP stores configuration, and default OCSP store.	Allow	Administrators

D.5.7. certServer.ocsp.crl

Controls access to read or update CRLs through the agent services interface. The default setting is:

```
allow (add) group="Online Certificate Status Manager Agents" || group="Trusted Managers"
```

Table D.63. certServer.ocsp.crl ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
add	Add new CRLs to those managed by the OCSP responder.	Allow	OCSP Agents Trusted Managers

D.5.8. certServer.ocsp.group

Controls access to the internal database for adding users and groups for the Online Certificate Status Manager instance.

```
allow (modify,read) group="Administrators"
```

Table D.64. certServer.ocsp.group ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Create, edit or delete user and group entries for the instance.	Allow	Administrators
read	View user and group entries for the instance.	Allow	Administrators

D.5.9. certServer.ocsp.info

Controls who can read information about the OCSP responder.

```
allow (read) group="Online Certificate Status Manager Agents"
```

Table D.65. certServer.ocsp.info ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View OCSP responder information.	Allow	OCSP Agents

D.6. TOKEN KEY SERVICE-SPECIFIC ACLS

This section covers the default access control configuration attributes which are set specifically for the Token Key Service (TKS). The TKS ACL configuration also includes all of the common ACLs listed in Section D.2, “Common ACLs”.

There are access control rules set for the TKS's administrative console and for access by other subsystems to the TKS.

D.6.1. certServer.tks.encrypteddata

Controls who can encrypt data.

```
allow(execute) group="Token Key Service Manager Agents"
```

Table D.66. certServer.tks.encrypteddata ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Encrypted data stored in the TKS.	Allow	TKS Agents

D.6.2. certServer.tks.group

Controls access to the internal database for adding users and groups for the TKS instance.

```
allow (modify,read) group="Administrators"
```

Table D.67. certServer.tks.group ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Create, edit, or delete user and group entries for the instance.	Allow	Administrators
read	View user and group entries for the instance.	Allow	Administrators

D.6.3. certServer.tks.importTransportCert

Controls who can import the transport certificate used by the TKS to deliver keys.

```
allow (modify,read) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" ||
group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators" ||
group="Enterprise TPS Administrators"
```

Table D.68. certServer.tks.importTransportCert ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Update the transport certificate.	Allow	Enterprise Administrators
read	Import the transport certificate.	Allow	Enterprise Administrators

D.6.4. certServer.tks.keysetdata

Controls who can view information about key sets derived and stored by the TKS.

```
allow (execute) group="Token Key Service Manager Agents"
```

Table D.69. certServer.tks.keysetdata ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Create diversified key set data.	Allow	TKS Agents

D.6.5. certServer.tks.registerUser

Defines which group or user can create an agent user for the instance. The default configuration is:

```
allow (modify,read) group="Enterprise CA Administrators" || group="Enterprise KRA Administrators" ||
group="Enterprise OCSP Administrators" || group="Enterprise TKS Administrators" ||
group="Enterprise TPS Administrators"
```

Table D.70. certServer.tks.registerUser ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
modify	Register a new agent.	Allow	Enterprise Administrators
read	Read existing agent information.	Allow	Enterprise Administrators

D.6.6. certServer.tks.sessionkey

Controls who can create the session keys used by the TKS instance to connections to the TPS.

```
allow (execute) group="Token Key Service Manager Agents"
```

Table D.71. certServer.tks.sessionkey ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Create session keys generated by the TKS.	Allow	TKS Agents

D.6.7. certServer.tks.randomdata

Controls who can create random data.

```
allow (execute) group="Token Key Service Manager Agents"
```

Table D.72. certServer.tks.randomdata ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Generate random data.	Allow	TKS Agents

APPENDIX E. AUDIT EVENTS

This Appendix provides individual audit events and their parameter description and format. Every audit event in the log is accompanied by the following information:

- The Java identifier of the thread. For example:

  ```
  0.localhost-startStop-1
  ```

- The time stamp the event occurred at. For example:

  ```
  [21/Jan/2019:17:53:00 IST]
  ```

- The log source (14 is SIGNED_AUDIT):

  ```
  [14]
  ```

- The current log level (6 is Security-related events. See the *Log Levels (Message Categories)* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide* . For example:

  ```
  [6]
  ```

- The information about the log event (which is log event specific; see Section E.1, "Audit Event Descriptions" for information about each field in a particular log event). For example:

  ```
  [AuditEvent=AUDIT_LOG_STARTUP][SubjectID=$System$][Outcome=Success] audit function startup
  ```

E.1. AUDIT EVENT DESCRIPTIONS

The following lists the audit events provided in Certificate System:

```
######################## SIGNED AUDIT EVENTS ############################
# Common fields:
# - Outcome: "Success" or "Failure"
# - SubjectID: The UID of the user responsible for the operation
#     "$System$" or "SYSTEM" if system-initiated operation (e.g. log signing).
#
#########################################################################
# Required Audit Events
#
# Event: ACCESS_SESSION_ESTABLISH with [Outcome=Failure]
# Description: This event is used when access session failed to establish.
# Applicable subsystems: CA, KRA, OCSP, TKS, TPS
# Enabled by default: Yes
# Fields:
# - ClientIP: Client IP address.
# - ServerIP: Server IP address.
# - SubjectID: Client certificate subject DN.
# - Outcome: Failure
```

```
    # - Info: Failure reason.
    #
    LOGGING_SIGNED_AUDIT_ACCESS_SESSION_ESTABLISH_FAILURE=\
    <type=ACCESS_SESSION_ESTABLISH>:[AuditEvent=ACCESS_SESSION_ESTABLISH]{0}
access session establish failure
    #
    # Event: ACCESS_SESSION_ESTABLISH with [Outcome=Success]
    # Description: This event is used when access session was established successfully.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - ClientIP: Client IP address.
    # - ServerIP: Server IP address.
    # - SubjectID: Client certificate subject DN.
    # - Outcome: Success
    #
    LOGGING_SIGNED_AUDIT_ACCESS_SESSION_ESTABLISH_SUCCESS=\
    <type=ACCESS_SESSION_ESTABLISH>:[AuditEvent=ACCESS_SESSION_ESTABLISH]{0}
access session establish success
    #
    # Event: ACCESS_SESSION_TERMINATED
    # Description: This event is used when access session was terminated.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - ClientIP: Client IP address.
    # - ServerIP: Server IP address.
    # - SubjectID: Client certificate subject DN.
    # - Info: The TLS Alert received from NSS
    # - Outcome: Success
    # - Info: The TLS Alert received from NSS
    #
    LOGGING_SIGNED_AUDIT_ACCESS_SESSION_TERMINATED=\
    <type=ACCESS_SESSION_TERMINATED>:[AuditEvent=ACCESS_SESSION_TERMINATED]{0}
access session terminated
    #
    # Event: AUDIT_LOG_SIGNING
    # Description: This event is used when a signature on the audit log is generated (same as "flush"
time).
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: Predefined to be "$System$" because this operation
    #     associates with no user.
    # - Outcome: Success
    # - sig: The base-64 encoded signature of the buffer just flushed.
    #
    LOGGING_SIGNED_AUDIT_AUDIT_LOG_SIGNING_3=[AuditEvent=AUDIT_LOG_SIGNING]
[SubjectID={0}][Outcome={1}] signature of audit buffer just flushed: sig: {2}
    #
    # Event: AUDIT_LOG_STARTUP
    # Description: This event is used at audit function startup.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
```

```
    # - Outcome:
    #
    LOGGING_SIGNED_AUDIT_AUDIT_LOG_STARTUP_2=<type=AUDIT_LOG_STARTUP>:
[AuditEvent=AUDIT_LOG_STARTUP][SubjectID={0}][Outcome={1}] audit function startup
    #
    # Event: AUTH with [Outcome=Failure]
    # Description: This event is used when authentication fails.
    #   In case of TLS-client auth, only webserver env can pick up the TLS violation.
    #   CS authMgr can pick up certificate mismatch, so this event is used.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID:
    # - Outcome: Failure
    #     (obviously, if authentication failed, you won't have a valid SubjectID, so
    #     in this case, SubjectID should be $Unidentified$)
    # - AuthMgr: The authentication manager instance name that did
    #     this authentication.
    # - AttemptedCred: The credential attempted and failed.
    #
    LOGGING_SIGNED_AUDIT_AUTH_FAIL=<type=AUTH>:[AuditEvent=AUTH]{0} authentication
failure
    #
    # Event: AUTH with [Outcome=Success]
    # Description: This event is used when authentication succeeded.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of user who has been authenticated
    # - Outcome: Success
    # - AuthMgr: The authentication manager instance name that did
    #     this authentication.
    #
    LOGGING_SIGNED_AUDIT_AUTH_SUCCESS=<type=AUTH>:[AuditEvent=AUTH]{0}
authentication success
    #
    # Event: AUTHZ with [Outcome=Failure]
    # Description: This event is used when authorization has failed.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of user who has failed to be authorized for an action
    # - Outcome: Failure
    # - aclResource: The ACL resource ID as defined in ACL resource list.
    # - Op: One of the operations as defined with the ACL statement
    #    e.g. "read" for an ACL statement containing "(read,write)".
    # - Info:
    #
    LOGGING_SIGNED_AUDIT_AUTHZ_FAIL=<type=AUTHZ>:[AuditEvent=AUTHZ]{0} authorization
failure
    #
    # Event: AUTHZ with [Outcome=Success]
    # Description: This event is used when authorization is successful.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
```

```
    # - SubjectID: id of user who has been authorized for an action
    # - Outcome: Success
    # - aclResource: The ACL resource ID as defined in ACL resource list.
    # - Op: One of the operations as defined with the ACL statement
    #     e.g. "read" for an ACL statement containing "(read,write)".
    #
    LOGGING_SIGNED_AUDIT_AUTHZ_SUCCESS=<type=AUTHZ>:[AuditEvent=AUTHZ]{0}
authorization success
    #
    # Event: CERT_PROFILE_APPROVAL
    # Description: This event is used when an agent approves/disapproves a certificate profile set by
the
    #   administrator for automatic approval.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of the CA agent who approved the certificate enrollment profile
    # - Outcome:
    # - ProfileID: One of the profiles defined by the administrator
    #     and to be approved by an agent.
    # - Op: "approve" or "disapprove".
    #
    LOGGING_SIGNED_AUDIT_CERT_PROFILE_APPROVAL_4=
<type=CERT_PROFILE_APPROVAL>:[AuditEvent=CERT_PROFILE_APPROVAL][SubjectID={0}]
[Outcome={1}][ProfileID={2}][Op={3}] certificate profile approval
    #
    # Event: CERT_REQUEST_PROCESSED
    # Description: This event is used when certificate request has just been through the approval
process.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: The UID of the agent who approves, rejects, or cancels
    #     the certificate request.
    # - Outcome:
    # - ReqID: The request ID.
    # - InfoName: "certificate" (in case of approval), "rejectReason"
    #     (in case of reject), or "cancelReason" (in case of cancel)
    # - InfoValue: The certificate (in case of success), a reject reason in
    #     text, or a cancel reason in text.
    # - CertSerialNum:
    #
    LOGGING_SIGNED_AUDIT_CERT_REQUEST_PROCESSED=
<type=CERT_REQUEST_PROCESSED>:[AuditEvent=CERT_REQUEST_PROCESSED]{0}
certificate request processed
    #
    # Event: CERT_SIGNING_INFO
    # Description: This event indicates which key is used to sign certificates.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
    # - Outcome: Success
    # - SKI: Subject Key Identifier of the certificate signing certificate
    # - AuthorityID: (applicable only to lightweight CA)
    #
```

```
    LOGGING_SIGNED_AUDIT_CERT_SIGNING_INFO=<type=CERT_SIGNING_INFO>:
[AuditEvent=CERT_SIGNING_INFO]{0} certificate signing info
    #
    # Event: CERT_STATUS_CHANGE_REQUEST
    # Description: This event is used when a certificate status change request (e.g. revocation)
    #   is made (before approval process).
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of uer who performed the action
    # - Outcome:
    # - ReqID: The request ID.
    # - CertSerialNum: The serial number (in hex) of the certificate to be revoked.
    # - RequestType: "revoke", "on-hold", "off-hold"
    #
    LOGGING_SIGNED_AUDIT_CERT_STATUS_CHANGE_REQUEST=
<type=CERT_STATUS_CHANGE_REQUEST>:[AuditEvent=CERT_STATUS_CHANGE_REQUEST]
{0} certificate revocation/unrevocation request made
    #
    # Event: CERT_STATUS_CHANGE_REQUEST_PROCESSED
    # Description: This event is used when certificate status is changed (revoked, expired, on-hold,
    #   off-hold).
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: The UID of the agent that processed the request.
    # - Outcome:
    # - ReqID: The request ID.
    # - RequestType: "revoke", "on-hold", "off-hold"
    # - Approval: "complete", "rejected", or "canceled"
    #     (note that "complete" means "approved")
    # - CertSerialNum: The serial number (in hex).
    # - RevokeReasonNum: One of the following number:
    #     reason number       reason
    #     ---------------------------------------
    #     0             Unspecified
    #     1             Key compromised
    #     2             CA key compromised (should not be used)
    #     3             Affiliation changed
    #     4             Certificate superceded
    #     5             Cessation of operation
    #     6             Certificate is on-hold
    # - Info:
    #
    LOGGING_SIGNED_AUDIT_CERT_STATUS_CHANGE_REQUEST_PROCESSED=
<type=CERT_STATUS_CHANGE_REQUEST_PROCESSED>:
[AuditEvent=CERT_STATUS_CHANGE_REQUEST_PROCESSED]{0} certificate status change
request processed
    #
    # Event: CLIENT_ACCESS_SESSION_ESTABLISH with [Outcome=Failure]
    # Description: This event is when access session failed to establish when Certificate System acts
as client.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - ClientHost: Client hostname.
```

```
    # - ServerHost: Server hostname.
    # - ServerPort: Server port.
    # - SubjectID: SYSTEM
    # - Outcome: Failure
    # - Info:
    #
    LOGGING_SIGNED_AUDIT_CLIENT_ACCESS_SESSION_ESTABLISH_FAILURE=\
    <type=CLIENT_ACCESS_SESSION_ESTABLISH>:
[AuditEvent=CLIENT_ACCESS_SESSION_ESTABLISH]{0} access session failed to establish when
Certificate System acts as client
    #
    # Event: CLIENT_ACCESS_SESSION_ESTABLISH with [Outcome=Success]
    # Description: This event is used when access session was established successfully when
    #   Certificate System acts as client.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - ClientHost: Client hostname.
    # - ServerHost: Server hostname.
    # - ServerPort: Server port.
    # - SubjectID: SYSTEM
    # - Outcome: Success
    #
    LOGGING_SIGNED_AUDIT_CLIENT_ACCESS_SESSION_ESTABLISH_SUCCESS=\
    <type=CLIENT_ACCESS_SESSION_ESTABLISH>:
[AuditEvent=CLIENT_ACCESS_SESSION_ESTABLISH]{0} access session establish successfully
when Certificate System acts as client
    #
    # Event: CLIENT_ACCESS_SESSION_TERMINATED
    # Description: This event is used when access session was terminated when Certificate System
acts as client.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - ClientHost: Client hostname.
    # - ServerHost: Server hostname.
    # - ServerPort: Server port.
    # - SubjectID: SYSTEM
    # - Outcome: Success
    # - Info: The TLS Alert received from NSS
    #
    LOGGING_SIGNED_AUDIT_CLIENT_ACCESS_SESSION_TERMINATED=\
    <type=CLIENT_ACCESS_SESSION_TERMINATED>:
[AuditEvent=CLIENT_ACCESS_SESSION_TERMINATED]{0} access session terminated when
Certificate System acts as client
    #
    # Event: CMC_REQUEST_RECEIVED
    # Description: This event is used when a CMC request is received.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: The UID of user that triggered this event.
    #     If CMC requests is signed by an agent, SubjectID should
    #     be that of the agent.
    #     In case of an unsigned request, it would bear $Unidentified$.
    # - Outcome:
```

```
    # - CMCRequest: Base64 encoding of the CMC request received
    #
    LOGGING_SIGNED_AUDIT_CMC_REQUEST_RECEIVED_3=
<type=CMC_REQUEST_RECEIVED>:[AuditEvent=CMC_REQUEST_RECEIVED][SubjectID={0}]
[Outcome={1}][CMCRequest={2}] CMC request received
    #
    # Event: CMC_RESPONSE_SENT
    # Description: This event is used when a CMC response is sent.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: The UID of user that triggered this event.
    # - Outcome:
    # - CMCResponse: Base64 encoding of the CMC response sent
    #
    LOGGING_SIGNED_AUDIT_CMC_RESPONSE_SENT_3=<type=CMC_RESPONSE_SENT>:
[AuditEvent=CMC_RESPONSE_SENT][SubjectID={0}][Outcome={1}][CMCResponse={2}] CMC
response sent
    #
    # Event: CMC_SIGNED_REQUEST_SIG_VERIFY
    # Description: This event is used when agent signed CMC certificate requests or revocation
requests
    #   are submitted and signature is verified.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: the user who signed the CMC request (success case)
    # - Outcome:
    # - ReqType: The request type (enrollment, or revocation).
    # - CertSubject: The certificate subject name of the certificate request.
    # - SignerInfo: A unique String representation for the signer.
    #
    LOGGING_SIGNED_AUDIT_CMC_SIGNED_REQUEST_SIG_VERIFY=
<type=CMC_SIGNED_REQUEST_SIG_VERIFY>:
[AuditEvent=CMC_SIGNED_REQUEST_SIG_VERIFY]{0} agent signed CMC request signature
verification
    #
    # Event: CMC_USER_SIGNED_REQUEST_SIG_VERIFY
    # Description: This event is used when CMC (user-signed or self-signed) certificate requests or
revocation requests
    #   are submitted and signature is verified.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: the user who signed the CMC request (success case)
    # - Outcome:
    # - ReqType: The request type (enrollment, or revocation).
    # - CertSubject: The certificate subject name of the certificate request.
    # - CMCSignerInfo: A unique String representation for the CMC request signer.
    # - info:
    #
    LOGGING_SIGNED_AUDIT_CMC_USER_SIGNED_REQUEST_SIG_VERIFY_FAILURE=
<type=CMC_USER_SIGNED_REQUEST_SIG_VERIFY>:
[AuditEvent=CMC_USER_SIGNED_REQUEST_SIG_VERIFY]{0} User signed CMC request
signature verification failure
    LOGGING_SIGNED_AUDIT_CMC_USER_SIGNED_REQUEST_SIG_VERIFY_SUCCESS=
```

```
<type=CMC_USER_SIGNED_REQUEST_SIG_VERIFY>:
[AuditEvent=CMC_USER_SIGNED_REQUEST_SIG_VERIFY]{0} User signed CMC request
signature verification success
   #
   # Event: CONFIG_ACL
   # Description: This event is used when configuring ACL information.
   # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
   # Enabled by default: Yes
   # Fields:
   # - SubjectID: id of administrator who performed the action
   # - Outcome:
   # - ParamNameValPairs: A name-value pair
   #     (where name and value are separated by the delimiter ;;)
   #     separated by + (if more than one name-value pair) of config params changed.
   #
   LOGGING_SIGNED_AUDIT_CONFIG_ACL_3=<type=CONFIG_ACL>:
[AuditEvent=CONFIG_ACL][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}] ACL
configuration parameter(s) change
   #
   # Event: CONFIG_AUTH
   # Description: This event is used when configuring authentication.
   # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
   # Enabled by default: Yes
   # Fields:
   # - SubjectID: id of administrator who performed the action
   # - Outcome:
   # - ParamNameValPairs: A name-value pair
   #     (where name and value are separated by the delimiter ;;)
   #     separated by + (if more than one name-value pair) of config params changed.
   #     --- Password MUST NOT be logged ---
   #
   LOGGING_SIGNED_AUDIT_CONFIG_AUTH_3=<type=CONFIG_AUTH>:
[AuditEvent=CONFIG_AUTH][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}] authentication
configuration parameter(s) change
   #
   # Event: CONFIG_CERT_PROFILE
   # Description: This event is used when configuring certificate profile
   #   (general settings and certificate profile).
   # Applicable subsystems: CA
   # Enabled by default: Yes
   # Fields:
   # - SubjectID: id of administrator who performed the action
   # - Outcome:
   # - ParamNameValPairs: A name-value pair
   #     (where name and value are separated by the delimiter ;;)
   #     separated by + (if more than one name-value pair) of config params changed.
   #
   LOGGING_SIGNED_AUDIT_CONFIG_CERT_PROFILE_3=<type=CONFIG_CERT_PROFILE>:
[AuditEvent=CONFIG_CERT_PROFILE][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}]
certificate profile configuration parameter(s) change
   #
   # Event: CONFIG_CRL_PROFILE
   # Description: This event is used when configuring CRL profile
   #   (extensions, frequency, CRL format).
   # Applicable subsystems: CA
   # Enabled by default: Yes
```

```
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_CONFIG_CRL_PROFILE_3=<type=CONFIG_CRL_PROFILE>:
[AuditEvent=CONFIG_CRL_PROFILE][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}] CRL
profile configuration parameter(s) change
    #
    # Event: CONFIG_DRM
    # Description: This event is used when configuring KRA.
    #   This includes key recovery scheme, change of any secret component.
    # Applicable subsystems: KRA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #     --- secret component (password) MUST NOT be logged ---
    #
    LOGGING_SIGNED_AUDIT_CONFIG_DRM_3=<type=CONFIG_DRM>:
[AuditEvent=CONFIG_DRM][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}] DRM
configuration parameter(s) change
    #
    # Event: CONFIG_OCSP_PROFILE
    # Description: This event is used when configuring OCSP profile
    #   (everything under Online Certificate Status Manager).
    # Applicable subsystems: OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_CONFIG_OCSP_PROFILE_3=<type=CONFIG_OCSP_PROFILE>:
[AuditEvent=CONFIG_OCSP_PROFILE][SubjectID={0}][Outcome={1}][ParamNameValPairs={2}]
OCSP profile configuration parameter(s) change
    #
    # Event: CONFIG_ROLE
    # Description: This event is used when configuring role information.
    #   This includes anything under users/groups, add/remove/edit a role, etc.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
```

```
    LOGGING_SIGNED_AUDIT_CONFIG_ROLE=<type=CONFIG_ROLE>:
[AuditEvent=CONFIG_ROLE]{0} role configuration parameter(s) change
    #
    # Event: CONFIG_SERIAL_NUMBER
    # Description: This event is used when configuring serial number ranges
    #   (when requesting a serial number range when cloning, for example).
    # Applicable subsystems: CA, KRA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_CONFIG_SERIAL_NUMBER_1=
<type=CONFIG_SERIAL_NUMBER>:[AuditEvent=CONFIG_SERIAL_NUMBER][SubjectID={0}]
[Outcome={1}][ParamNameValPairs={2}] serial number range update
    #
    # Event: CONFIG_SIGNED_AUDIT
    # Description: This event is used when configuring signedAudit.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id of administrator who performed the action
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_CONFIG_SIGNED_AUDIT=<type=CONFIG_SIGNED_AUDIT>:
[AuditEvent=CONFIG_SIGNED_AUDIT]{0} signed audit configuration parameter(s) change
    #
    # Event: CONFIG_TRUSTED_PUBLIC_KEY
    # Description: This event is used when:
    #   1. "Manage Certificate" is used to edit the trustness of certificates
    #      and deletion of certificates
    #   2. "Certificate Setup Wizard" is used to import CA certificates into the
    #      certificate database (Although CrossCertificatePairs are stored
    #      within internaldb, audit them as well)
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: ID of administrator who performed this configuration
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_CONFIG_TRUSTED_PUBLIC_KEY=
<type=CONFIG_TRUSTED_PUBLIC_KEY>:[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY]{0}
certificate database configuration
    #
    # Event: CRL_SIGNING_INFO
    # Description: This event indicates which key is used to sign CRLs.
    # Applicable subsystems: CA
```

```
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
    # - Outcome:
    # - SKI: Subject Key Identifier of the CRL signing certificate
    #
    LOGGING_SIGNED_AUDIT_CRL_SIGNING_INFO=<type=CRL_SIGNING_INFO>:
[AuditEvent=CRL_SIGNING_INFO]{0} CRL signing info
    #
    # Event: DELTA_CRL_GENERATION
    # Description: This event is used when delta CRL generation is complete.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $Unidentified$
    # - Outcome: "Success" when delta CRL is generated successfully, "Failure" otherwise.
    # - CRLnum: The CRL number that identifies the CRL
    # - Info:
    # - FailureReason:
    #
    LOGGING_SIGNED_AUDIT_DELTA_CRL_GENERATION=<type=DELTA_CRL_GENERATION>:
[AuditEvent=DELTA_CRL_GENERATION]{0} Delta CRL generation
    #
    # Event: FULL_CRL_GENERATION
    # Description: This event is used when full CRL generation is complete.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
    # - Outcome: "Success" when full CRL is generated successfully, "Failure" otherwise.
    # - CRLnum: The CRL number that identifies the CRL
    # - Info:
    # - FailureReason:
    #
    LOGGING_SIGNED_AUDIT_FULL_CRL_GENERATION=<type=FULL_CRL_GENERATION>:
[AuditEvent=FULL_CRL_GENERATION]{0} Full CRL generation
    #
    # Event: PROFILE_CERT_REQUEST
    # Description: This event is used when a profile certificate request is made (before approval
process).
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: The UID of user that triggered this event.
    #     If CMC enrollment requests signed by an agent, SubjectID should
    #     be that of the agent.
    # - Outcome:
    # - CertSubject: The certificate subject name of the certificate request.
    # - ReqID: The certificate request ID.
    # - ProfileID: One of the certificate profiles defined by the
    #     administrator.
    #
    LOGGING_SIGNED_AUDIT_PROFILE_CERT_REQUEST_5=
<type=PROFILE_CERT_REQUEST>:[AuditEvent=PROFILE_CERT_REQUEST][SubjectID={0}]
[Outcome={1}][ReqID={2}][ProfileID={3}][CertSubject={4}] certificate request made with certificate
profiles
```

```
    #
    # Event: PROOF_OF_POSSESSION
    # Description: This event is used for proof of possession during certificate enrollment processing.
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: id that represents the authenticated user
    # - Outcome:
    # - Info: some information on when/how it occurred
    #
    LOGGING_SIGNED_AUDIT_PROOF_OF_POSSESSION_3=
<type=PROOF_OF_POSSESSION>:[AuditEvent=PROOF_OF_POSSESSION][SubjectID={0}]
[Outcome={1}][Info={2}] proof of possession
    #
    # Event: OCSP_ADD_CA_REQUEST_PROCESSED
    # Description: This event is used when an add CA request to the OCSP Responder is processed.
    # Applicable subsystems: OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: OCSP administrator user id
    # - Outcome: "Success" when CA is added successfully, "Failure" otherwise.
    # - CASubjectDN: The subject DN of the leaf CA cert in the chain.
    #
    LOGGING_SIGNED_AUDIT_OCSP_ADD_CA_REQUEST_PROCESSED=
<type=OCSP_ADD_CA_REQUEST_PROCESSED>:
[AuditEvent=OCSP_ADD_CA_REQUEST_PROCESSED]{0} Add CA for OCSP Responder
    #
    # Event: OCSP_GENERATION
    # Description: This event is used when an OCSP response generated is complete.
    # Applicable subsystems: CA, OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $NonRoleUser$
    # - Outcome: "Success" when OCSP response is generated successfully, "Failure" otherwise.
    # - FailureReason:
    #
    LOGGING_SIGNED_AUDIT_OCSP_GENERATION=<type=OCSP_GENERATION>:
[AuditEvent=OCSP_GENERATION]{0} OCSP response generation
    #
    # Event: OCSP_REMOVE_CA_REQUEST_PROCESSED with [Outcome=Failure]
    # Description: This event is used when a remove CA request to the OCSP Responder is processed
and failed.
    # Applicable subsystems: OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: OCSP administrator user id
    # - Outcome: Failure
    # - CASubjectDN: The subject DN of the leaf CA certificate in the chain.
    #
    LOGGING_SIGNED_AUDIT_OCSP_REMOVE_CA_REQUEST_PROCESSED_FAILURE=
<type=OCSP_REMOVE_CA_REQUEST_PROCESSED>:
[AuditEvent=OCSP_REMOVE_CA_REQUEST_PROCESSED]{0} Remove CA for OCSP Responder
has failed
    #
    # Event: OCSP_REMOVE_CA_REQUEST_PROCESSED with [Outcome=Success]
    # Description: This event is used when a remove CA request to the OCSP Responder is processed
```

```
successfully.
    # Applicable subsystems: OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: OCSP administrator user id
    # - Outcome: "Success" when CA is removed successfully, "Failure" otherwise.
    # - CASubjectDN: The subject DN of the leaf CA certificate in the chain.
    #
    LOGGING_SIGNED_AUDIT_OCSP_REMOVE_CA_REQUEST_PROCESSED_SUCCESS=
<type=OCSP_REMOVE_CA_REQUEST_PROCESSED>:
[AuditEvent=OCSP_REMOVE_CA_REQUEST_PROCESSED]{0} Remove CA for OCSP Responder
is successful
    #
    # Event: OCSP_SIGNING_INFO
    # Description: This event indicates which key is used to sign OCSP responses.
    # Applicable subsystems: CA, OCSP
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
    # - Outcome:
    # - SKI: Subject Key Identifier of the OCSP signing certificate
    # - AuthorityID: (applicable only to lightweight CA)
    #
    LOGGING_SIGNED_AUDIT_OCSP_SIGNING_INFO=<type=OCSP_SIGNING_INFO>:
[AuditEvent=OCSP_SIGNING_INFO]{0} OCSP signing info
    #
    # Event: ROLE_ASSUME
    # Description: This event is used when a user assumes a role.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID:
    # - Outcome:
    # - Role: One of the valid roles:
    #     "Administrators", "Certificate Manager Agents", or "Auditors".
    #     Note that customized role names can be used once configured.
    #
    LOGGING_SIGNED_AUDIT_ROLE_ASSUME=<type=ROLE_ASSUME>:
[AuditEvent=ROLE_ASSUME]{0} assume privileged role
    #
    # Event: SECURITY_DOMAIN_UPDATE
    # Description: This event is used when updating contents of security domain
    #   (add/remove a subsystem).
    # Applicable subsystems: CA
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: CA administrator user ID
    # - Outcome:
    # - ParamNameValPairs: A name-value pair
    #     (where name and value are separated by the delimiter ;;)
    #     separated by + (if more than one name-value pair) of config params changed.
    #
    LOGGING_SIGNED_AUDIT_SECURITY_DOMAIN_UPDATE_1=
<type=SECURITY_DOMAIN_UPDATE>:[AuditEvent=SECURITY_DOMAIN_UPDATE][SubjectID=
{0}][Outcome={1}][ParamNameValPairs={2}] security domain update
    #
```

```
    # Event: SELFTESTS_EXECUTION
    # Description: This event is used when self tests are run.
    # Applicable subsystems: CA, KRA, OCSP, TKS, TPS
    # Enabled by default: Yes
    # Fields:
    # - SubjectID: $System$
    # - Outcome:
    #
    LOGGING_SIGNED_AUDIT_SELFTESTS_EXECUTION_2=<type=SELFTESTS_EXECUTION>:
[AuditEvent=SELFTESTS_EXECUTION][SubjectID={0}][Outcome={1}] self tests execution (see
selftests.log for details)
```

GLOSSARY

A

access control

The process of controlling what particular users are allowed to do. For example, access control to servers is typically based on an identity, established by a password or a certificate, and on rules regarding what that entity can do. See also access control list (ACL).

access control instructions (ACI)

An access rule that specifies how subjects requesting access are to be identified or what rights are allowed or denied for a particular subject. See access control list (ACL).

access control list (ACL)

A collection of access control entries that define a hierarchy of access rules to be evaluated when a server receives a request for access to a particular resource. See access control instructions (ACI).

administrator

The person who installs and configures one or more Certificate System managers and sets up privileged users, or agents, for them. See also agent.

Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES), like its predecessor Data Encryption Standard (DES), is a FIPS-approved symmetric-key encryption standard. AES was adopted by the US government in 2002. It defines three block ciphers, AES-128, AES-192 and AES-256. The National Institute of Standards and Technology (NIST) defined the AES standard in U.S. FIPS PUB 197. For more information, see http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

agent

A user who belongs to a group authorized to manage agent services for a Certificate System manager. See also Certificate Manager agent, Key Recovery Authority agent.

agent services

1. Services that can be administered by a Certificate System agent through HTML pages served by the Certificate System subsystem for which the agent has been assigned the necessary privileges.

2. The HTML pages for administering such services.

agent-approved enrollment

An enrollment that requires an agent to approve the request before the certificate is issued.

APDU

Application protocol data unit. A communication unit (analogous to a byte) that is used in communications between a smart card and a smart card reader.

attribute value assertion (AVA)

An assertion of the form *attribute = value*, where *attribute* is a tag, such as **o** (organization) or **uid** (user ID), and *value* is a value such as "Red Hat, Inc." or a login name. AVAs are used to form the distinguished name (DN) that identifies the subject of a certificate, called the subject name of the certificate.

audit log

A log that records various system events. This log can be signed, providing proof that it was not tampered with, and can only be read by an auditor user.

auditor

A privileged user who can view the signed audit logs.

authentication

Confident identification; assurance that a party to some computerized transaction is not an impostor. Authentication typically involves the use of a password, certificate, PIN, or other information to validate identity over a computer network. See also password-based authentication, certificate-based authentication, client authentication, server authentication.

authentication module

A set of rules (implemented as a Java™ class) for authenticating an end entity, agent, administrator, or any other entity that needs to interact with a Certificate System subsystem. In the case of typical end-user enrollment, after the user has supplied the information requested by the enrollment form, the enrollment servlet uses an authentication module associated with that form to validate the information and authenticate the user's identity. See servlet.

authorization

Permission to access a resource controlled by a server. Authorization typically takes place after the ACLs associated with a resource have been evaluated by a server. See access control list (ACL).

automated enrollment

A way of configuring a Certificate System subsystem that allows automatic authentication for end-entity enrollment, without human intervention. With this form of authentication, a certificate request that completes authentication module processing successfully is automatically approved for profile processing and certificate issuance.

B

bind DN

A user ID, in the form of a distinguished name (DN), used with a password to authenticate to Red Hat Directory Server.

C

CA certificate

A certificate that identifies a certificate authority. See also certificate authority (CA), subordinate CA, root CA.

CA hierarchy

A hierarchy of CAs in which a root CA delegates the authority to issue certificates to subordinate CAs. Subordinate CAs can also expand the hierarchy by delegating issuing status to other CAs. See also certificate authority (CA), subordinate CA, root CA.

CA server key

The SSL server key of the server providing a CA service.

CA signing key

The private key that corresponds to the public key in the CA certificate. A CA uses its signing key to sign certificates and CRLs.

certificate

Digital data, formatted according to the X.509 standard, that specifies the name of an individual, company, or other entity (the subject name of the certificate) and certifies that a public key, which is also included in the certificate, belongs to that entity. A certificate is issued and digitally signed by a certificate authority (CA). A certificate's validity can be verified by checking the CA's digital signature through public-key cryptography techniques. To be trusted within a public-key infrastructure (PKI), a certificate must be issued and signed by a CA that is trusted by other entities enrolled in the PKI.

certificate authority (CA)

A trusted entity that issues a certificate after verifying the identity of the person or entity the certificate is intended to identify. A CA also renews and revokes certificates and generates CRLs. The entity named in the issuer field of a certificate is always a CA. Certificate authorities can be independent third parties or a person or organization using certificate-issuing server software, such as Red Hat Certificate System.

certificate chain

A hierarchical series of certificates signed by successive certificate authorities. A CA certificate identifies a certificate authority (CA) and is used to sign certificates issued by that authority. A CA certificate can in turn be signed by the CA certificate of a parent CA, and so on up to a root CA. Certificate System allows any end entity to retrieve all the certificates in a certificate chain.

certificate extensions

An X.509 v3 certificate contains an extensions field that permits any number of additional fields to be added to the certificate. Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates. A number of standard extensions have been defined by the PKIX working group.

certificate fingerprint

A one-way hash associated with a certificate. The number is not part of the certificate itself, but is produced by applying a hash function to the contents of the certificate. If the contents of the certificate changes, even by a single character, the same function produces a different number. Certificate fingerprints can therefore be used to verify that certificates have not been tampered with.

Certificate Management Message Formats (CMMF)

Message formats used to convey certificate requests and revocation requests from end entities to a Certificate Manager and to send a variety of information to end entities. A proposed standard from the Internet Engineering Task Force (IETF) PKIX working group. CMMF is subsumed by another proposed standard, Certificate Management Messages over Cryptographic Message Syntax (CMC) . For detailed information, see https://tools.ietf.org/html/draft-ietf-pkix-cmmf-02.

Certificate Management Messages over Cryptographic Message Syntax (CMC)

Message format used to convey a request for a certificate to a Certificate Manager. A proposed standard from the Internet Engineering Task Force (IETF) PKIX working group. For detailed information, see https://tools.ietf.org/html/draft-ietf-pkix-cmc-02.

Certificate Manager

An independent Certificate System subsystem that acts as a certificate authority. A Certificate Manager instance issues, renews, and revokes certificates, which it can publish along with CRLs to an LDAP directory. It accepts requests from end entities. See certificate authority (CA).

Certificate Manager agent

A user who belongs to a group authorized to manage agent services for a Certificate Manager. These services include the ability to access and modify (approve and reject) certificate requests and issue certificates.

certificate profile

A set of configuration settings that defines a certain type of enrollment. The certificate profile sets policies for a particular type of enrollment along with an authentication method in a certificate profile.

Certificate Request Message Format (CRMF)

Format used for messages related to management of X.509 certificates. This format is a subset of CMMF. See also Certificate Management Message Formats (CMMF) . For detailed information, see https://tools.ietf.org/html/rfc2511.

certificate revocation list (CRL)

As defined by the X.509 standard, a list of revoked certificates by serial number, generated and signed by a certificate authority (CA).

Certificate System

See Red Hat Certificate System, Cryptographic Message Syntax (CS) .

Certificate System console

A console that can be opened for any single Certificate System instance. A Certificate System console allows the Certificate System administrator to control configuration settings for the corresponding Certificate System instance.

Certificate System subsystem

One of the five Certificate System managers: Certificate Manager, Online Certificate Status Manager, Key Recovery Authority, Token Key Service, or Token Processing System.

certificate-based authentication

Authentication based on certificates and public-key cryptography. See also password-based authentication.

chain of trust

See certificate chain.

chained CA

See linked CA.

cipher

See cryptographic algorithm.

client authentication

The process of identifying a client to a server, such as with a name and password or with a certificate and some digitally signed data. See certificate-based authentication, password-based authentication, server authentication.

client SSL certificate

A certificate used to identify a client to a server using the SSL protocol. See Secure Sockets Layer (SSL).

CMC

See Certificate Management Messages over Cryptographic Message Syntax (CMC) .

CMC Enrollment

Features that allow either signed enrollment or signed revocation requests to be sent to a Certificate Manager using an agent's signing certificate. These requests are then automatically processed by the Certificate Manager.

CMMF

See Certificate Management Message Formats (CMMF).

CRL

See certificate revocation list (CRL).

CRMF

See Certificate Request Message Format (CRMF).

cross-certification

The exchange of certificates by two CAs in different certification hierarchies, or chains. Cross-certification extends the chain of trust so that it encompasses both hierarchies. See also certificate authority (CA).

cross-pair certificate

A certificate issued by one CA to another CA which is then stored by both CAs to form a circle of trust. The two CAs issue certificates to each other, and then store both cross-pair certificates as a certificate pair.

cryptographic algorithm

A set of rules or directions used to perform cryptographic operations such as encryption and decryption.

Cryptographic Message Syntax (CS)

The syntax used to digitally sign, digest, authenticate, or encrypt arbitrary messages, such as CMMF.

cryptographic module

See PKCS #11 module.

cryptographic service provider (CSP)

A cryptographic module that performs cryptographic services, such as key generation, key storage, and encryption, on behalf of software that uses a standard interface such as that defined by PKCS #11 to request such services.

CSP

See cryptographic service provider (CSP).

D

decryption

Unscrambling data that has been encrypted. See encryption.

delta CRL

A CRL containing a list of those certificates that have been revoked since the last full CRL was issued.

digital ID

See certificate.

digital signature

To create a digital signature, the signing software first creates a one-way hash from the data to be signed, such as a newly issued certificate. The one-way hash is then encrypted with the private key of the signer. The resulting digital signature is unique for each piece of data signed. Even a single comma added to a message changes the digital signature for that message. Successful decryption of the digital signature with the signer's public key and comparison with another hash of the same data provides tamper detection. Verification of the certificate chain for the certificate containing the public key provides authentication of the signer. See also nonrepudiation, encryption.

distinguished name (DN)

A series of AVAs that identify the subject of a certificate. See attribute value assertion (AVA).

distribution points

Used for CRLs to define a set of certificates. Each distribution point is defined by a set of certificates that are issued. A CRL can be created for a particular distribution point.

dual key pair

Two public-private key pairs, four keys altogether, corresponding to two separate certificates. The

private key of one pair is used for signing operations, and the public and private keys of the other pair are used for encryption and decryption operations. Each pair corresponds to a separate certificate. See also encryption key, public-key cryptography, signing key.

Key Recovery Authority

An optional, independent Certificate System subsystem that manages the long-term archival and recovery of RSA encryption keys for end entities. A Certificate Manager can be configured to archive end entities' encryption keys with a Key Recovery Authority before issuing new certificates. The Key Recovery Authority is useful only if end entities are encrypting data, such as sensitive email, that the organization may need to recover someday. It can be used only with end entities that support dual key pairs: two separate key pairs, one for encryption and one for digital signatures.

Key Recovery Authority agent

A user who belongs to a group authorized to manage agent services for a Key Recovery Authority, including managing the request queue and authorizing recovery operation using HTML-based administration pages.

Key Recovery Authority recovery agent

One of the *m of n* people who own portions of the storage key for the Key Recovery Authority.

Key Recovery Authority storage key

Special key used by the Key Recovery Authority to encrypt the end entity's encryption key after it has been decrypted with the Key Recovery Authority's private transport key. The storage key never leaves the Key Recovery Authority.

Key Recovery Authority transport certificate

Certifies the public key used by an end entity to encrypt the entity's encryption key for transport to the Key Recovery Authority. The Key Recovery Authority uses the private key corresponding to the certified public key to decrypt the end entity's key before encrypting it with the storage key.

E

eavesdropping

Surreptitious interception of information sent over a network by an entity for which the information is not intended.

Elliptic Curve Cryptography (ECC)

A cryptographic algorithm which uses elliptic curves to create additive logarithms for the mathematical problems which are the basis of the cryptographic keys. ECC ciphers are more efficient to use than RSA ciphers and, because of their intrinsic complexity, are stronger at smaller bits than RSA ciphers.

encryption

Scrambling information in a way that disguises its meaning. See decryption.

encryption key

A private key used for encryption only. An encryption key and its equivalent public key, plus a signing key and its equivalent public key, constitute a dual key pair.

end entity

In a public-key infrastructure (PKI), a person, router, server, or other entity that uses a certificate to identify itself.

enrollment

The process of requesting and receiving an X.509 certificate for use in a public-key infrastructure (PKI). Also known as *registration*.

extensions field

See certificate extensions.

F

Federal Bridge Certificate Authority (FBCA)

A configuration where two CAs form a circle of trust by issuing cross-pair certificates to each other and storing the two cross-pair certificates as a single certificate pair.

fingerprint

See certificate fingerprint.

FIPS PUBS 140

Federal Information Standards Publications (FIPS PUBS) 140 is a US government standard for implementations of cryptographic modules, hardware or software that encrypts and decrypts data or performs other cryptographic operations, such as creating or verifying digital signatures. Many products sold to the US government must comply with one or more of the FIPS standards. See http://www.nist.gov.

firewall

A system or combination of systems that enforces a boundary between two or more networks.

I

impersonation

The act of posing as the intended recipient of information sent over a network. Impersonation can take two forms: spoofing and misrepresentation.

input

In the context of the certificate profile feature, it defines the enrollment form for a particular certificate profile. Each input is set, which then dynamically creates the enrollment form from all inputs configured for this enrollment.

intermediate CA

A CA whose certificate is located between the root CA and the issued certificate in a certificate chain.

IP spoofing

The forgery of client IP addresses.

J

JAR file

A digital envelope for a compressed collection of files organized according to the Java™ archive (JAR) format.

Java™ archive (JAR) format

A set of conventions for associating digital signatures, installer scripts, and other information with files in a directory.

Java™ Cryptography Architecture (JCA)

The API specification and reference developed by Sun Microsystems for cryptographic services. See http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.Introduction.

Java™ Development Kit (JDK)

Software development kit provided by Sun Microsystems for developing applications and applets using the Java™ programming language.

Java™ Native Interface (JNI)

A standard programming interface that provides binary compatibility across different implementations of the Java™ Virtual Machine (JVM) on a given platform, allowing existing code written in a language such as C or C++ for a single platform to bind to Java™. See http://java.sun.com/products/jdk/1.2/docs/guide/jni/index.html.

Java™ Security Services (JSS)

A Java™ interface for controlling security operations performed by Network Security Services (NSS).

K

KEA

See Key Exchange Algorithm (KEA).

key

A large number used by a cryptographic algorithm to encrypt or decrypt data. A person's public key, for example, allows other people to encrypt messages intended for that person. The messages must then be decrypted by using the corresponding private key.

key exchange

A procedure followed by a client and server to determine the symmetric keys they will both use during an SSL session.

Key Exchange Algorithm (KEA)

An algorithm used for key exchange by the US Government.

L

Lightweight Directory Access Protocol (LDAP)

A directory service protocol designed to run over TCP/IP and across multiple platforms. LDAP is a simplified version of Directory Access Protocol (DAP), used to access X.500 directories. LDAP is under IETF change control and has evolved to meet Internet requirements.

linked CA

An internally deployed certificate authority (CA) whose certificate is signed by a public, third-party CA. The internal CA acts as the root CA for certificates it issues, and the third- party CA acts as the root CA for certificates issued by other CAs that are linked to the same third-party root CA. Also known as "chained CA" and by other terms used by different public CAs.

M

manual authentication

A way of configuring a Certificate System subsystem that requires human approval of each certificate request. With this form of authentication, a servlet forwards a certificate request to a request queue after successful authentication module processing. An agent with appropriate privileges must then approve each request individually before profile processing and certificate issuance can proceed.

MD5

A message digest algorithm that was developed by Ronald Rivest. See also one-way hash.

message digest

See one-way hash.

misrepresentation

The presentation of an entity as a person or organization that it is not. For example, a website might pretend to be a furniture store when it is really a site that takes credit-card payments but never sends any goods. Misrepresentation is one form of impersonation. See also spoofing.

N

Network Security Services (NSS)

A set of libraries designed to support cross-platform development of security-enabled communications applications. Applications built using the NSS libraries support the Secure Sockets Layer (SSL) protocol for authentication, tamper detection, and encryption, and the PKCS #11 protocol for cryptographic token interfaces. NSS is also available separately as a software development kit.

non-TMS

Non-token management system. Refers to a configuration of subsystems (the CA and, optionally, KRA and OCSP) which do *not* handle smart cards directly.

See Also token management system (TMS).

nonrepudiation

The inability by the sender of a message to deny having sent the message. A digital signature provides one form of nonrepudiation.

O

object signing

A method of file signing that allows software developers to sign Java code, JavaScript scripts, or any kind of file and allows users to identify the signers and control access by signed code to local system resources.

object-signing certificate

A certificate whose associated private key is used to sign objects; related to object signing.

OCSP

Online Certificate Status Protocol.

one-way hash

1. A number of fixed-length generated from data of arbitrary length with the aid of a hashing algorithm. The number, also called a message digest, is unique to the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.

2. The content of the hashed data cannot be deduced from the hash.

operation

The specific operation, such as read or write, that is being allowed or denied in an access control instruction.

output

In the context of the certificate profile feature, it defines the resulting form from a successful certificate enrollment for a particular certificate profile. Each output is set, which then dynamically creates the form from all outputs configured for this enrollment.

P

password-based authentication

Confident identification by means of a name and password. See also authentication, certificate-based authentication.

PKCS #10

The public-key cryptography standard that governs certificate requests.

PKCS #11

The public-key cryptography standard that governs cryptographic tokens such as smart cards.

PKCS #11 module

A driver for a cryptographic device that provides cryptographic services, such as encryption and decryption, through the PKCS #11 interface. A PKCS #11 module, also called a *cryptographic module* or *cryptographic service provider*, can be implemented in either hardware or software. A PKCS #11 module always has one or more slots, which may be implemented as physical hardware slots in some form of physical reader, such as for smart cards, or as conceptual slots in software. Each slot for a PKCS #11 module can in turn contain a token, which is the hardware or software device that actually provides cryptographic services and optionally stores certificates and keys. Red Hat provides a built-in PKCS #11 module with Certificate System.

PKCS #12

The public-key cryptography standard that governs key portability.

PKCS #7

The public-key cryptography standard that governs signing and encryption.

private key

One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data encrypted with the corresponding public key.

proof-of-archival (POA)

Data signed with the private Key Recovery Authority transport key that contains information about an archived end-entity key, including key serial number, name of the Key Recovery Authority, subject name of the corresponding certificate, and date of archival. The signed proof-of-archival data are the response returned by the Key Recovery Authority to the Certificate Manager after a successful key archival operation. See also Key Recovery Authority transport certificate.

public key

One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a certificate. It is typically used to encrypt data sent to the public key's owner, who then decrypts the data with the corresponding private key.

public-key cryptography

A set of well-established techniques and standards that allow an entity to verify its identity electronically or to sign and encrypt electronic data. Two keys are involved, a public key and a private key. A public key is published as part of a certificate, which associates that key with a particular identity. The corresponding private key is kept secret. Data encrypted with the public key can be decrypted only with the private key.

public-key infrastructure (PKI)

The standards and services that facilitate the use of public-key cryptography and X.509 v3 certificates in a networked environment.

R

RC2, RC4

Cryptographic algorithms developed for RSA Data Security by Rivest. See also cryptographic algorithm.

Red Hat Certificate System

A highly configurable set of software components and tools for creating, deploying, and managing certificates. Certificate System is comprised of five major subsystems that can be installed in different Certificate System instances in different physical locations: Certificate Manager, Online Certificate Status Manager, Key Recovery Authority, Token Key Service, and Token Processing System.

registration

See enrollment.

root CA

The certificate authority (CA) with a self-signed certificate at the top of a certificate chain. See also CA certificate, subordinate CA.

RSA algorithm

Short for Rivest-Shamir-Adleman, a public-key algorithm for both encryption and authentication. It was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman and introduced in 1978.

RSA key exchange

A key-exchange algorithm for SSL based on the RSA algorithm.

S

sandbox

A Java™ term for the carefully defined limits within which Java™ code must operate.

secure channel

A security association between the TPS and the smart card which allows encrypted communciation based on a shared master key generated by the TKS and the smart card APDUs.

Secure Sockets Layer (SSL)

A protocol that allows mutual authentication between a client and server and the establishment of an authenticated and encrypted connection. SSL runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols.

security domain

A centralized repository or inventory of PKI subsystems. Its primary purpose is to facilitate the installation and configuration of new PKI services by automatically establishing trusted relationships between subsystems.

self tests

A feature that tests a Certificate System instance both when the instance starts up and on-demand.

server authentication

The process of identifying a server to a client. See also client authentication.

server SSL certificate

A certificate used to identify a server to a client using the Secure Sockets Layer (SSL) protocol.

servlet

Java™ code that handles a particular kind of interaction with end entities on behalf of a Certificate System subsystem. For example, certificate enrollment, revocation, and key recovery requests are each handled by separate servlets.

SHA

Secure Hash Algorithm, a hash function used by the US government.

signature algorithm

A cryptographic algorithm used to create digital signatures. Certificate System supports the MD5 and SHA signing algorithms. See also cryptographic algorithm, digital signature.

signed audit log

See audit log.

signing certificate

A certificate whose public key corresponds to a private key used to create digital signatures. For example, a Certificate Manager must have a signing certificate whose public key corresponds to the private key it uses to sign the certificates it issues.

signing key

A private key used for signing only. A signing key and its equivalent public key, plus an encryption key and its equivalent public key, constitute a dual key pair.

single sign-on

1. In Certificate System, a password that simplifies the way to sign on to Red Hat Certificate System by storing the passwords for the internal database and tokens. Each time a user logs on, he is required to enter this single password.

2. The ability for a user to log in once to a single computer and be authenticated automatically by a variety of servers within a network. Partial single sign-on solutions can take many forms, including mechanisms for automatically tracking passwords used with different servers. Certificates support single sign-on within a public-key infrastructure (PKI). A user can log in once to a local client's private-key database and, as long as the client software is running, rely on certificate-based authentication to access each server within an organization that the user is allowed to access.

slot

The portion of a PKCS #11 module, implemented in either hardware or software, that contains a token.

smart card

A small device that contains a microprocessor and stores cryptographic information, such as keys and certificates, and performs cryptographic operations. Smart cards implement some or all of the PKCS #11 interface.

spoofing

Pretending to be someone else. For example, a person can pretend to have the email address **jdoe@example.com**, or a computer can identify itself as a site called **www.redhat.com** when it is not. Spoofing is one form of impersonation. See also misrepresentation.

SSL

See Secure Sockets Layer (SSL).

subject

The entity identified by a certificate. In particular, the subject field of a certificate contains a subject name that uniquely describes the certified entity.

subject name

A distinguished name (DN) that uniquely describes the subject of a certificate.

subordinate CA

A certificate authority whose certificate is signed by another subordinate CA or by the root CA. See CA certificate, root CA.

symmetric encryption

An encryption method that uses the same cryptographic key to encrypt and decrypt a given message.

T

tamper detection

A mechanism ensuring that data received in electronic form entirely corresponds with the original version of the same data.

token

A hardware or software device that is associated with a slot in a PKCS #11 module. It provides cryptographic services and optionally stores certificates and keys.

token key service (TKS)

A subsystem in the token management system which derives specific, separate keys for every smart card based on the smart card APDUs and other shared information, like the token CUID.

token management system (TMS)

The interrelated subsystems – CA, TKS, TPS, and, optionally, the KRA – which are used to manage certificates on smart cards (tokens).

token processing system (TPS)

A subsystem which interacts directly the Enterprise Security Client and smart cards to manage the keys and certificates on those smart cards.

tree hierarchy

The hierarchical structure of an LDAP directory.

trust

Confident reliance on a person or other entity. In a public-key infrastructure (PKI), trust refers to the relationship between the user of a certificate and the certificate authority (CA) that issued the certificate. If a CA is trusted, then valid certificates issued by that CA can be trusted.

V

virtual private network (VPN)

A way of connecting geographically distant divisions of an enterprise. The VPN allows the divisions to communicate over an encrypted channel, allowing authenticated, confidential transactions that would normally be restricted to a private network.

INDEX

A

B

C

K

L

M

N

O

P

R

S

W

APPENDIX F. REVISION HISTORY

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat Certificate System.

Revision 10.1-1 **Mon Jan 25 2021** **Florian Delehaye**
Various fixes and improvements.

Revision 10.1-0 **Wed Dec 02 2020** **Florian Delehaye**
Red Hat Certificate System 10.1 release of the guide.

Revision 10.0-0 **Wed Sep 17 2020** **Florian Delehaye**
Red Hat Certificate System 10.0 release of the guide.