



Red Hat Certificate System 9

Administration Guide (Common Criteria Edition)

Updated for Red Hat Certificate System 9.4 Common Criteria Certification
Edition 9.4-1

Red Hat Certificate System 9 Administration Guide (Common Criteria Edition)

Updated for Red Hat Certificate System 9.4 Common Criteria Certification
Edition 9.4-1

Red Hat
Customer Content Services

Legal Notice

Copyright © 2021 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-sa/3.0/). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide covers all aspects of installing, configuring, and managing Certificate System 9.4 in a Common Criteria environment. 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.

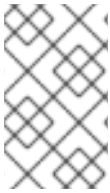
Table of Contents

CHAPTER 1. OVERVIEW OF RED HAT CERTIFICATE SYSTEM SUBSYSTEMS	5
1.1. USES FOR CERTIFICATES	5
1.2. A REVIEW OF CERTIFICATE SYSTEM SUBSYSTEMS	5
1.3. A LOOK AT MANAGING CERTIFICATES (NON-TMS)	5
1.4. A LOOK AT THE TOKEN MANAGEMENT SYSTEM (TMS)	6
1.5. RED HAT CERTIFICATE SYSTEM SERVICES	6
PART I. RED HAT CERTIFICATE SYSTEM USER INTERFACES	7
CHAPTER 2. USER INTERFACES	8
2.1. USER INTERFACES OVERVIEW	8
2.2. CLIENT NSS DATABASE INITIALIZATION	8
2.3. GRAPHICAL INTERFACE	9
2.4. WEB INTERFACE	11
2.5. COMMAND LINE INTERFACES	16
PART II. SETTING UP CERTIFICATE SERVICES	23
CHAPTER 3. MAKING RULES FOR ISSUING CERTIFICATES (CERTIFICATE PROFILES)	24
3.1. ABOUT CERTIFICATE PROFILES	24
3.2. SETTING UP CERTIFICATE PROFILES	27
3.3. DEFINING KEY DEFAULTS IN PROFILES	39
3.4. CONFIGURING PROFILES TO ENABLE RENEWAL	40
3.5. SETTING THE SIGNING ALGORITHMS FOR CERTIFICATES	41
3.6. MANAGING CA-RELATED PROFILES	43
3.7. MANAGING SUBJECT NAMES AND SUBJECT ALTERNATIVE NAMES	51
CHAPTER 4. SETTING UP KEY ARCHIVAL AND RECOVERY	57
4.1. ABOUT KEY ARCHIVAL AND RECOVERY	57
CHAPTER 5. REQUESTING, ENROLLING, AND MANAGING CERTIFICATES	62
5.1. ABOUT ENROLLING AND RENEWING CERTIFICATES	62
5.2. CREATING CERTIFICATE SIGNING REQUESTS	62
5.3. REQUESTING AND RECEIVING CERTIFICATES USING CMC	67
5.4. RENEWING CERTIFICATES	80
5.5. TRACING ISSUED CERTIFICATE TO CSR, AND CSR TO ISSUED CERTIFICATE	81
CHAPTER 6. REVOKING CERTIFICATES AND ISSUING CRLS	83
6.1. ABOUT REVOKING CERTIFICATES	83
6.2. REVOKING CERTIFICATES	85
6.3. ISSUING CRLS	104
6.4. SETTING FULL AND DELTA CRL SCHEDULES	112
6.5. USING THE ONLINE CERTIFICATE STATUS PROTOCOL (OCSP) RESPONDER	116
PART III. ADDITIONAL CONFIGURATION TO MANAGE CA SERVICES	127
CHAPTER 7. PUBLISHING CERTIFICATES AND CRLS	128
7.1. ABOUT PUBLISHING	128
7.2. CONFIGURING PUBLISHING TO A FILE	131
7.3. CONFIGURING PUBLISHING TO AN OCSP	134
7.4. CONFIGURING PUBLISHING TO AN LDAP DIRECTORY	136
7.5. CREATING RULES	143
7.6. ENABLING PUBLISHING	146
7.7. SETTING UP RESUMABLE CRL DOWNLOADS	148

7.8. PUBLISHING CROSS-PAIR CERTIFICATES	148
7.9. TESTING PUBLISHING TO FILES	149
7.10. VIEWING CERTIFICATES AND CRLS PUBLISHED TO FILE	150
7.11. UPDATING CERTIFICATES AND CRLS IN A DIRECTORY	151
CHAPTER 8. AUTHENTICATION FOR ENROLLING CERTIFICATES	153
8.1. AUTOMATIC APPROVAL BY AN AUTHENTICATION PLUG-IN	153
8.2. MANUAL APPROVAL BY A CA AGENT	156
8.3. MANUALLY REVIEWING THE CERTIFICATE STATUS USING THE COMMAND LINE	157
8.4. MANUALLY REVIEWING THE CERTIFICATE STATUS USING THE WEB INTERFACE	157
CHAPTER 9. AUTHORIZATION FOR ENROLLING CERTIFICATES (ACCESS EVALUATORS)	159
9.1. AUTHORIZATION MECHANISM	159
9.2. DEFAULT EVALUATORS	159
PART IV. MANAGING THE SUBSYSTEM INSTANCES	161
CHAPTER 10. SELF TESTS	162
10.1. RUNNING SELF-TESTS	162
10.2. DEBUGGING SELF-TESTS FAILURES	163
CHAPTER 11. MANAGING CERTIFICATE/KEY CRYPTO TOKEN	164
About Crypto Tokens	164
11.1. ABOUT CERTUTIL AND PKICERTIMPORT	164
11.2. IMPORTING A ROOT CERTIFICATE	167
11.3. IMPORTING AN INTERMEDIATE CERTIFICATE CHAIN	168
11.4. IMPORTING A CERTIFICATE INTO AN NSS DATABASE	168
CHAPTER 12. MANAGING CERTIFICATE SYSTEM USERS AND GROUPS	171
12.1. ABOUT AUTHORIZATION	171
12.2. DEFAULT GROUPS	171
12.3. MANAGING USERS AND GROUPS FOR A CA, OCSP, KRA, OR TKS	174
12.4. CONFIGURING ACCESS CONTROL FOR USERS	182
CHAPTER 13. CONFIGURING SUBSYSTEM LOGS	189
13.1. MANAGING LOGS	189
13.2. USING LOGS	192
CHAPTER 14. MANAGING SUBSYSTEM CERTIFICATES	197
14.1. REQUIRED SUBSYSTEM CERTIFICATES	197
14.2. RENEWING SUBSYSTEM CERTIFICATES	204
14.3. CHANGING THE NAMES OF SUBSYSTEM CERTIFICATES	206
14.4. MANAGING THE CERTIFICATE DATABASE	209
14.5. CHANGING THE TRUST SETTINGS OF A CA CERTIFICATE	216
14.6. MANAGING TOKENS USED BY THE SUBSYSTEMS	217
CHAPTER 15. SETTING TIME AND DATE IN RED HAT ENTERPRISE LINUX 7.6	219
CHANGING THE CURRENT TIME	219
CHANGING THE CURRENT DATE	219
CHAPTER 16. DETERMINING CERTIFICATE SYSTEM PRODUCT VERSION	220
CHAPTER 17. UPDATING RED HAT CERTIFICATE SYSTEM	221
CHAPTER 18. TROUBLESHOOTING	222
CHAPTER 19. SUBSYSTEM CONTROL AND MAINTENANCE	226

19.1. STARTING, STOPPING, RESTARTING, AND OBTAINING STATUS	226
19.2. SUBSYSTEM HEALTH CHECK	226
PART V. REFERENCES	227
APPENDIX A. CERTIFICATE PROFILE INPUT AND OUTPUT REFERENCE	228
A.1. INPUT REFERENCE	228
A.2. OUTPUT REFERENCE	230
APPENDIX B. DEFAULTS, CONSTRAINTS, AND EXTENSIONS FOR CERTIFICATES AND CRLS	231
B.1. DEFAULTS REFERENCE	231
B.2. CONSTRAINTS REFERENCE	266
B.3. STANDARD X.509 V3 CERTIFICATE EXTENSION REFERENCE	276
B.4. CRL EXTENSIONS	286
APPENDIX C. PUBLISHING MODULE REFERENCE	302
C.1. PUBLISHER PLUG-IN MODULES	302
C.2. MAPPER PLUG-IN MODULES	305
C.3. RULE INSTANCES	312
APPENDIX D. ACL REFERENCE	315
D.1. ABOUT ACL CONFIGURATION FILES	315
D.2. COMMON ACLS	316
D.3. CERTIFICATE MANAGER-SPECIFIC ACLS	323
D.4. KEY RECOVERY AUTHORITY-SPECIFIC ACLS	337
D.5. ONLINE CERTIFICATE STATUS MANAGER-SPECIFIC ACLS	342
D.6. TOKEN KEY SERVICE-SPECIFIC ACLS	346
D.7. TPS-SPECIFIC ACLS	348
APPENDIX E. AUDIT EVENTS	353
E.1. REQUIRED AUDIT EVENTS AND THEIR EXAMPLES	353
E.2. AUDIT EVENT DESCRIPTIONS	370
GLOSSARY	382
INDEX	397
APPENDIX F. REVISION HISTORY	408

CHAPTER 1. OVERVIEW OF RED HAT CERTIFICATE SYSTEM SUBSYSTEMS



NOTE

This chapter is an overview of Red Hat Certificate System and the various subsystems. For details on evaluated product features, please see the NIAP Product Compliant List at <https://www.niap-ccevs.org/Product/PCL.cfm>.



NOTE

Network security services (NSS) and Federal Information Processing Standard (FIPS) hardware security modules (HSM) were the only evaluated cryptographic providers.

Every common PKI operation – issuing, renewing and revoking certificates; archiving and recovering keys; publishing CRLs and verifying certificate status – is 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 are 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 (Common Criteria Edition)*.

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* in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

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 *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*:

- *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)



NOTE

Features in this section on TMS are not tested in the evaluation. This section is for reference only.

Certificate System creates, manages, renews, and revokes certificates, and it also archives and recovers keys. For organizations that 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 *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*:

- *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's role: administrators, agents, auditors, and end users. For an overview of the different functions that are performed through each interface, see *Red Hat Certificate System User Interfaces* section in the *Red Hat Certificate System 9 Planning, Installation, and Deployment Guide (Common Criteria Edition)* .

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 5.2.2, "Creating a CSR Using PKCS10Client"](#).

Certificate System's PKI Console interface is a graphical interface. [Section 2.3.1, "pkiconsole Initialization"](#) describes how to initialize it. [Section 2.3.2, "Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems"](#) gives an overview of using the console interface. 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.

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. More information about using the browser for specific tasks are included in the rest of the documentation, such as [Section 6.2.2.2, "Searching for Certificates \(Advanced\)"](#).



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

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 [Section 11.2, “Importing a Root Certificate”](#).

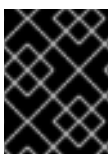
To request a new client certificate, see [Chapter 5, Requesting, Enrolling, and Managing Certificates](#).

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 [Chapter 11, Managing Certificate/Key Crypto Token](#):

```
$ 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"](#).

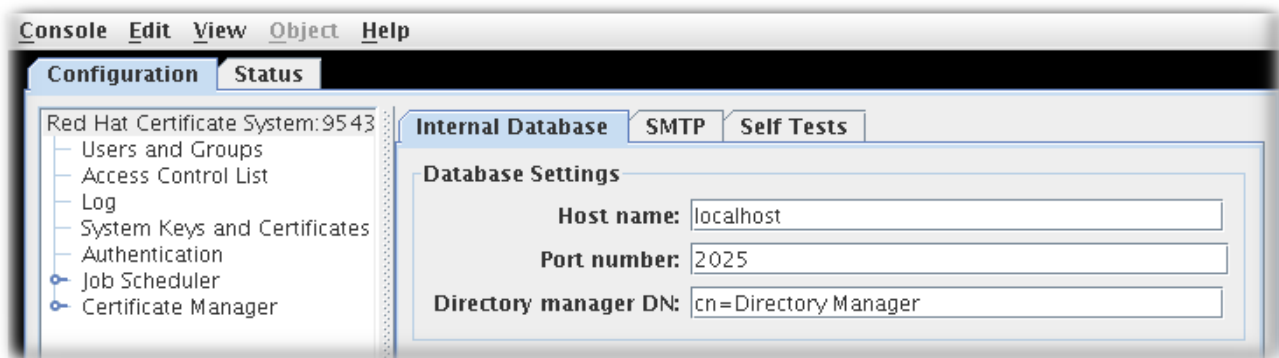


Figure 2.1. Certificate System Console

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.

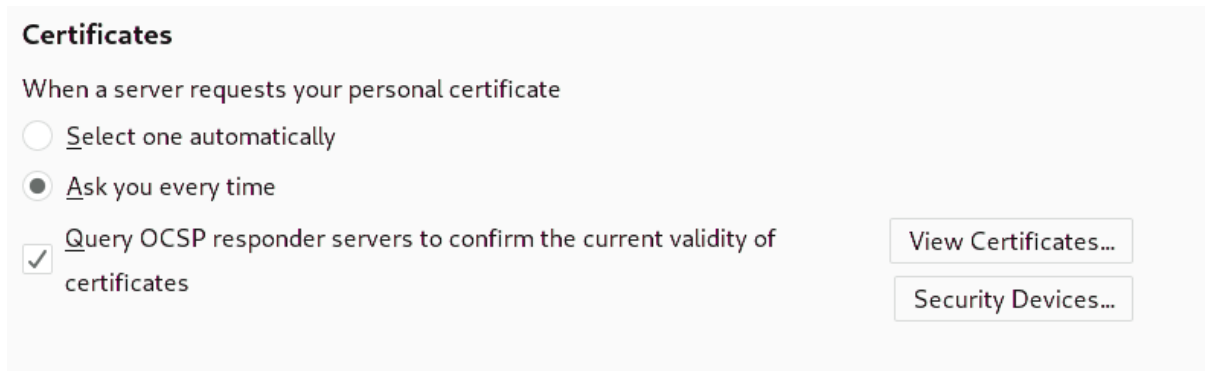
2.4. WEB INTERFACE

2.4.1. Browser Initialization

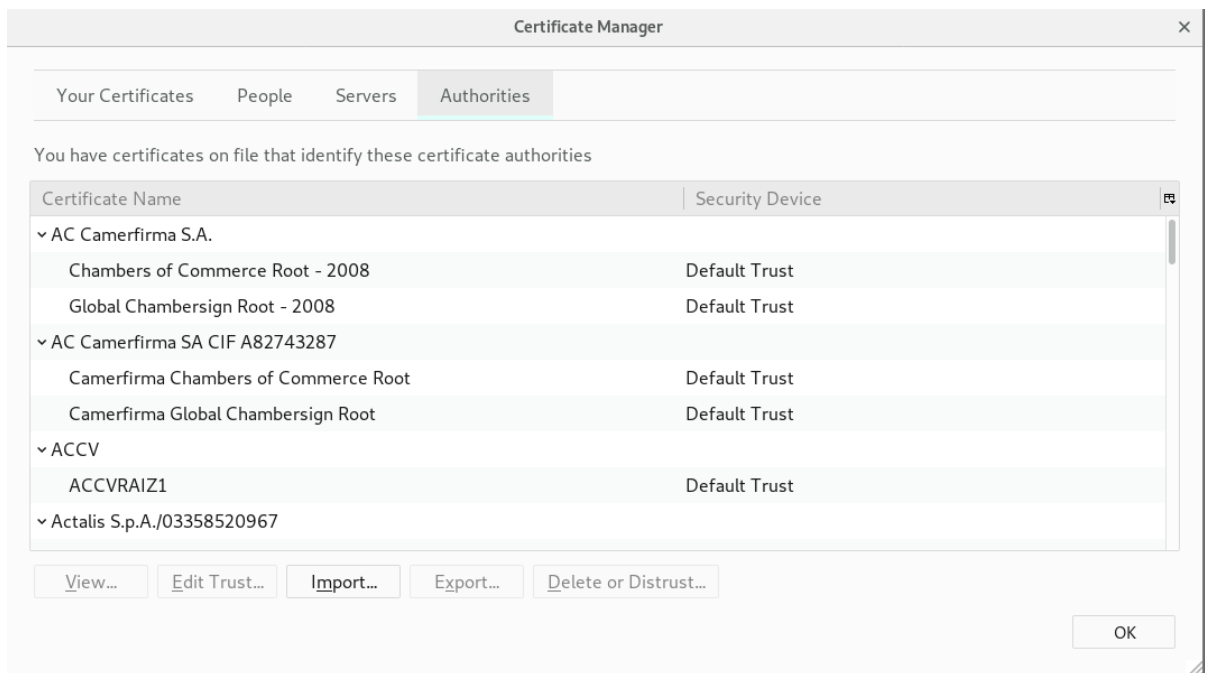
This section explains browser initialization for Firefox to access PKI services.

Importing a CA Certificate

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



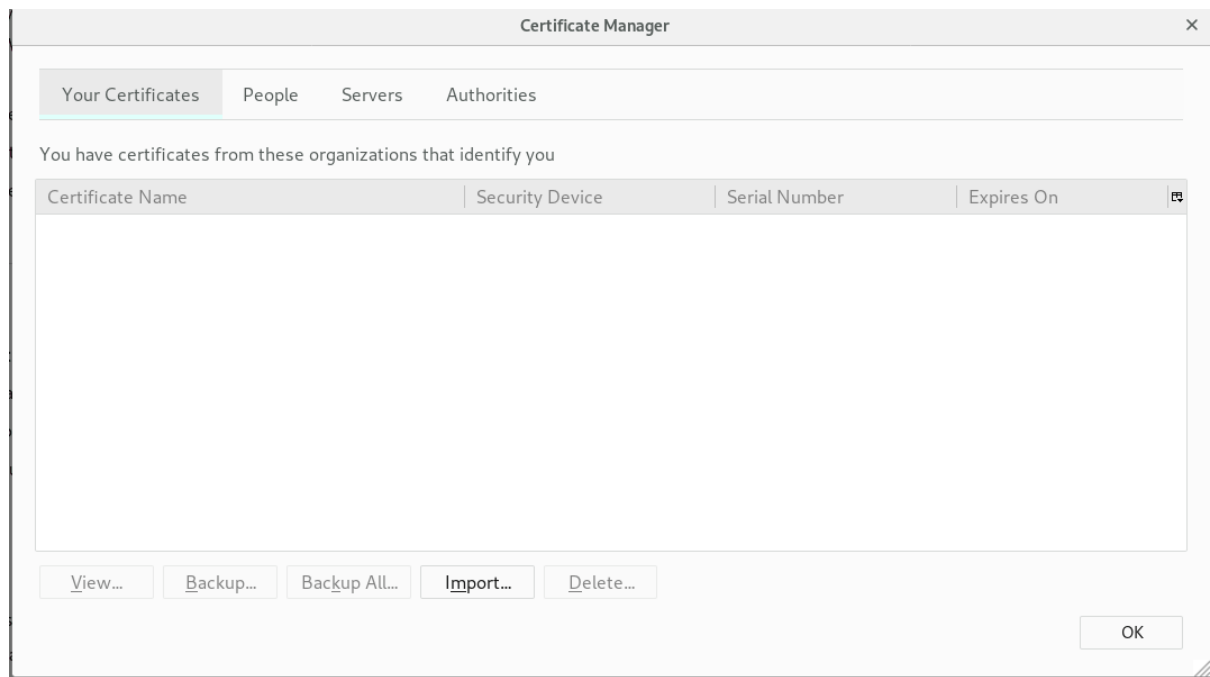
2. Select the **Authorities** tab and click the **Import** button.



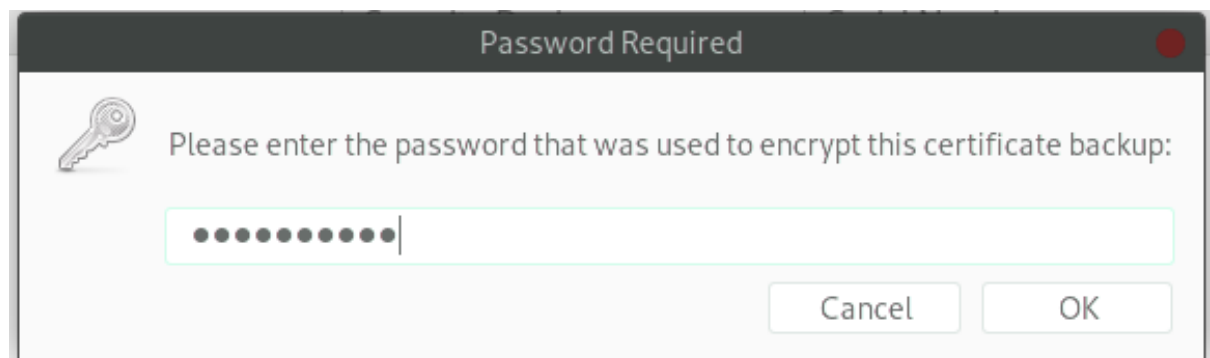
3. Select the **ca.crt** file and click **Import**.

Importing a Client Certificate

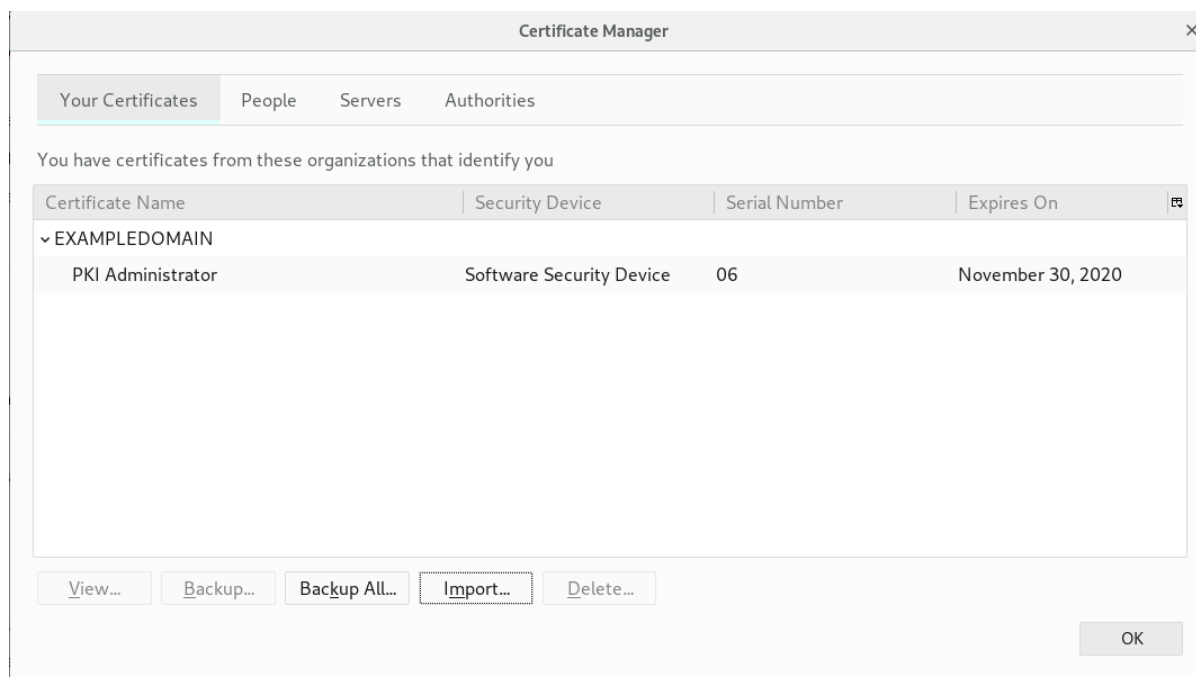
1. Click **Options** → **Preferences** → **Privacy & Security** → **View certificates**.
2. Select the **Your Certificates** tab.



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.

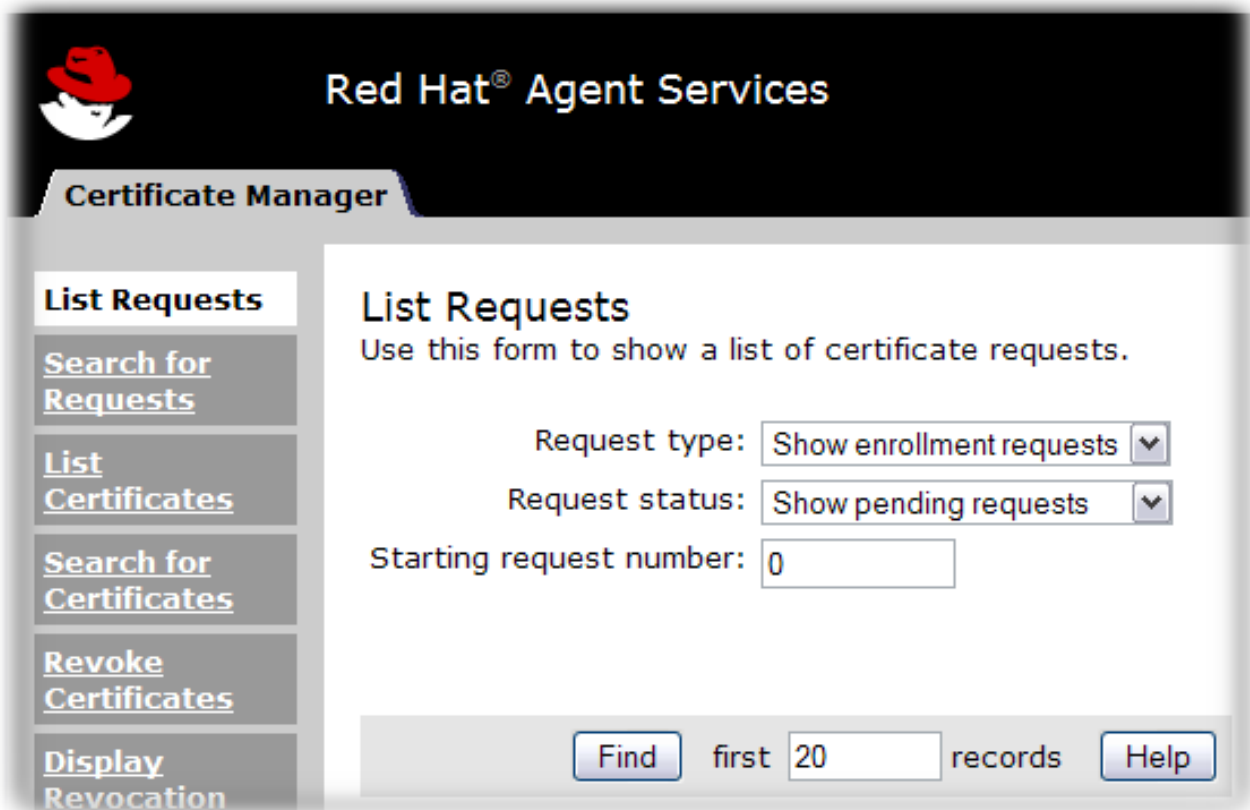


Figure 2.3. Certificate Manager's Agent Services Page

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.

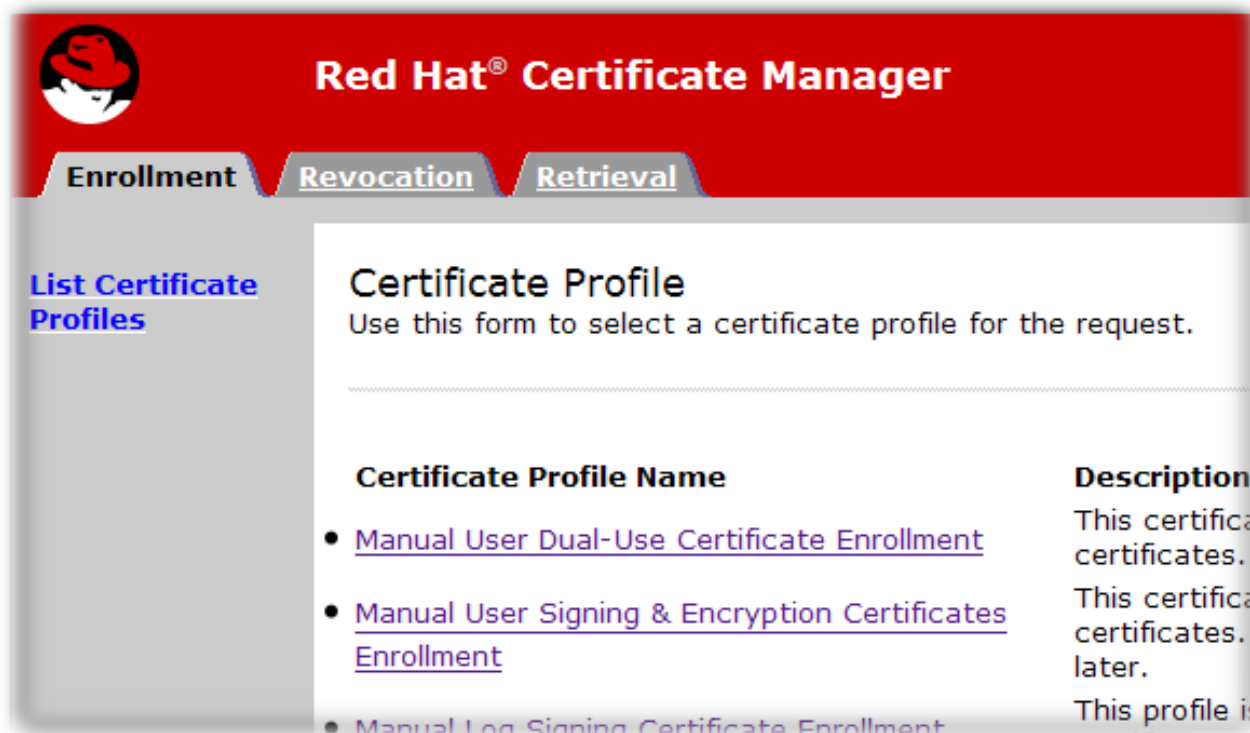


Figure 2.4. Certificate Manager's End-Entities Page

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 (Common Criteria Edition)*). 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 [Section 11.2, "Importing a Root Certificate"](#).

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 [Chapter 11, Managing Certificate/Key Crypto Token](#):

```
$ 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 14.2.2. Using Signed Audit Logs in the Red Hat Certificate System Administration Guide.

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. CMCRquest

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

```
$ CMCRquest example.cfg
```



NOTE

All options to the **CMCRquest** utility are specified as part of the configuration file passed to the utility. See the `CMCRquest(1)` man page for configuration file options and further information. Also see 4.3. Requesting and Receiving Certificates Using CMC and 6.2.1. Revoking a Certificate Using CMCRquest in Red Hat Certificate System Administration Guide.

2.5.6. CMCRvoke

Legacy. Do not use.

2.5.7. CMCSsharedToken

The CMCSsharedToken utility encrypts a user passphrase for shared-secret CMC requests. For example:

```
$ CMCSsharedToken -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 `CMCSsharedtoken(1)` man page and also 8.1.3.1. Creating a Shared Secret Token in Red Hat Certificate System Administration Guide.

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 4.2.4. Creating a CSR Using CRMFPopClient in Red Hat Certificate System Administration Guide.

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 **caSigningCert 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.

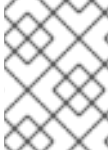
Example:

```
$ 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.

PART II. SETTING UP CERTIFICATE SERVICES



NOTE

Modification of **CS.cfg**, **server.xml** or any configuration file post-installation is expressly prohibited in a certified environment.

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

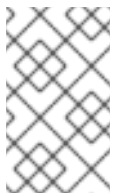
The parameters for each profile defining the inputs, outputs, and policy sets are listed in more detail in Table 11.1. Profile Configuration File Parameters in Red Hat Certificate System Planning, Installation and Deployment Guide.

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:

```

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

```

policysset.caCertSet.5.default.name=Basic Constraints Extension Default
policysset.caCertSet.5.default.params.basicConstraintsCritical=true
policysset.caCertSet.5.default.params.basicConstraintsIsCA=true
policysset.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:

-

```

policysset.caCertSet.5.constraint.class_id=basicConstraintsExtConstraintImpl
policysset.caCertSet.5.constraint.name=Basic Constraint Extension Constraint
policysset.caCertSet.5.constraint.params.basicConstraintsCritical=true
policysset.caCertSet.5.constraint.params.basicConstraintsIsCA=true
policysset.caCertSet.5.constraint.params.basicConstraintsMinPathLen=-1
policysset.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.1.33, “User Supplied Extension Default”](#).

3.1.3. Inputs and Outputs

Inputs set information that must be submitted to receive a certificate. In a Common Criteria environment, set the ***input.i1.class_id*** parameter in all enabled profiles to **cmcCertReqInputImpl**:

```
input.i1.class_id=cmcCertReqInputImpl
```

The outputs configured in the profile define the format of the certificate that is issued. In a Common Criteria environment, set the ***output.o1.class_id*** parameter in all enabled profiles to **certOutputImpl**:

```
output.o1.class_id=CertOutputImpl
```

In a Common Criteria-compliant Certificate System environment, users access profiles through the ***/ca/ee/ca/profileSubmitUserSignedCMCFull*** servlet that is accessed through the end-entities interface.

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
```

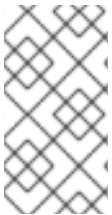
For example, to enable the **caCMCECserverCert** certificate profile:

```
# 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 caagent 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"](#).

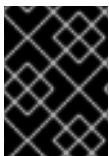
3. Enable the profile as a CA agent:

```
# 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

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.

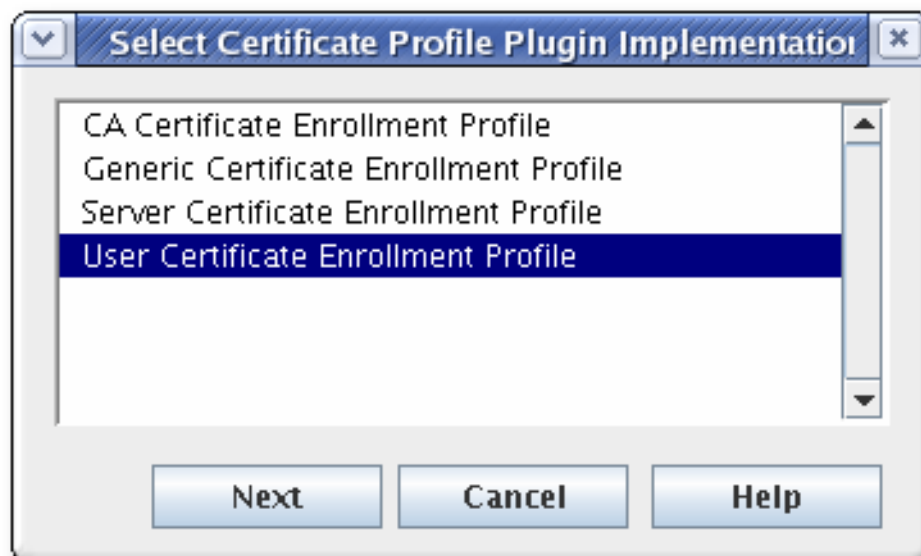
```
pkiconsole https://server.example.com:8443/ca
```

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**.

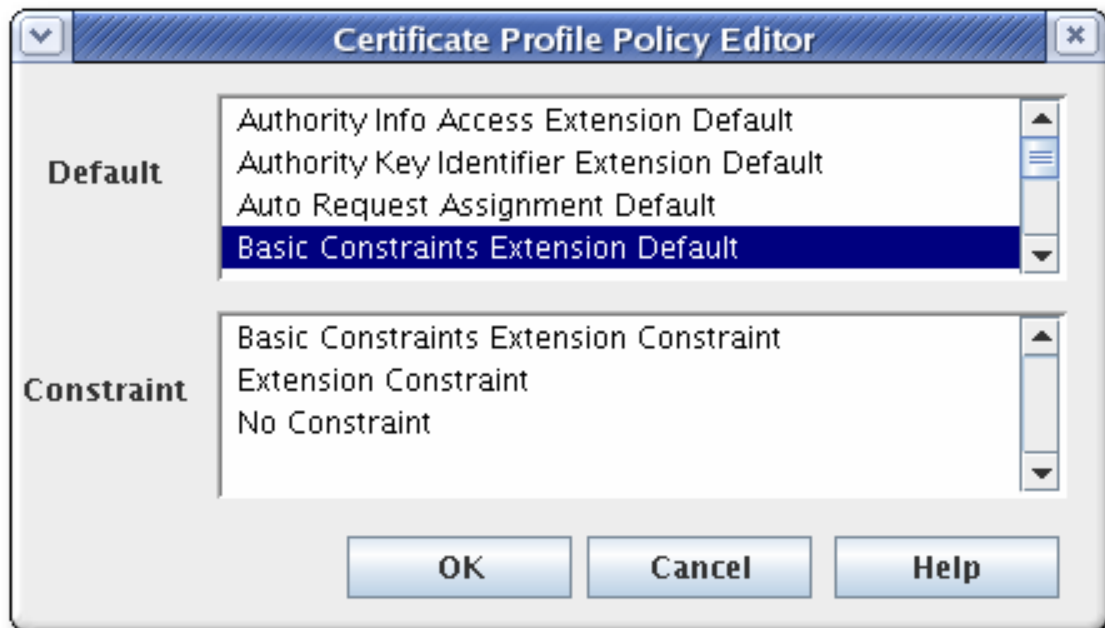
- **Certificate Profile Instance ID.** This is the ID used by the system to identify the profile.
- **Certificate Profile Name.** This is the user-friendly name for the profile.
- **Certificate Profile Description.**
- **End User Certificate Profile.** This sets whether the request must be made through 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**.

Certificate Profile Instance ID:

Policy Set ID:

Policy ID:

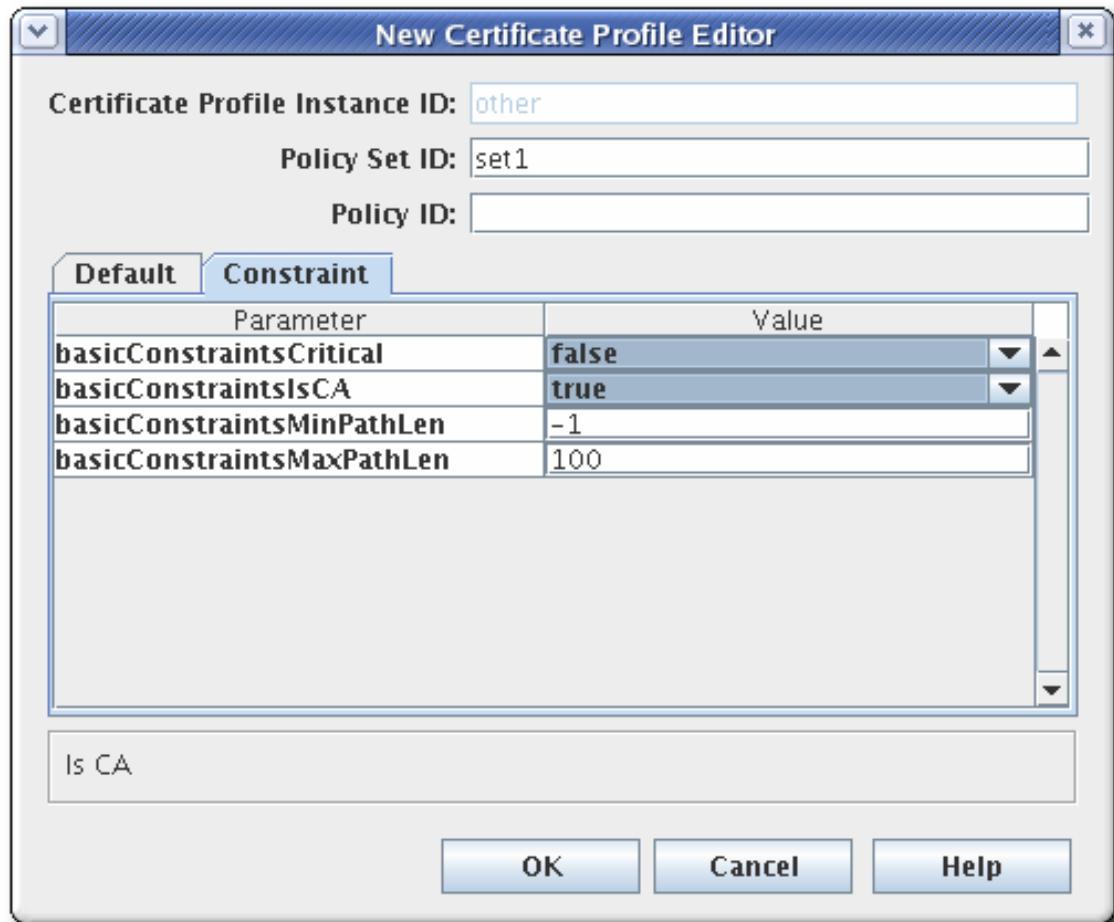
Default | **Constraint**

Parameter	Value
basicConstraintsCritical	false
basicConstraintsIsCA	true
basicConstraintsPathLen	-1

Criticality

OK Cancel Help

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 **cmCertReqInput** and delete other profiles by selecting them and clicking the **Delete** button.

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

The image shows two overlapping windows from a software application. The background window is titled "Certificate Profile Rule Editor" and contains several fields for configuring a certificate profile. The foreground window is titled "Certificate Profile Input Editor" and displays a list of input options.

Certificate Profile Rule Editor

Certificate Profile Instance ID: caCMCOtherCert
Certificate Profile Name: cmcOtherCert
Certificate Profile Description: cmcOtherCert
End User Certificate Profile: true
Certificate Profile Authentication: CMCAuth
Certificate Profile Plugin ID: caServerCertEnrollImpl

Inputs tab:

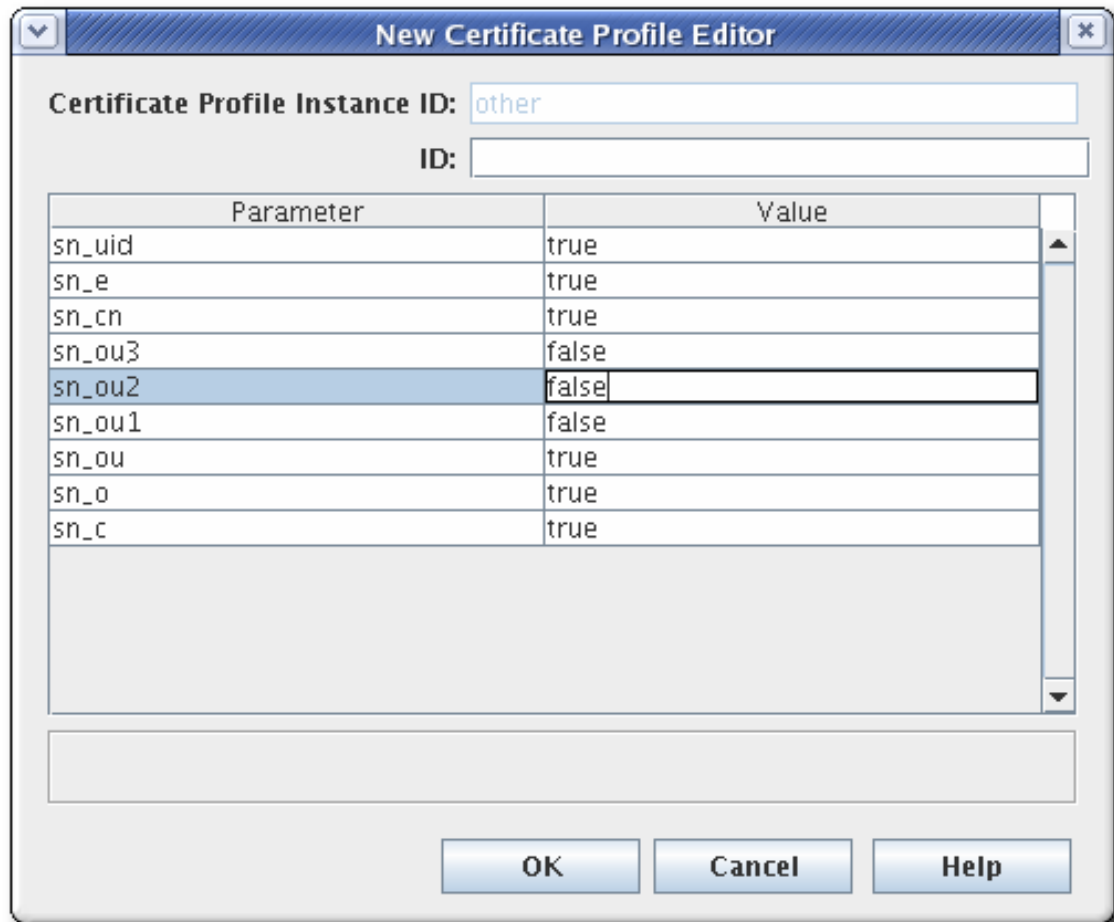
Id	Inputs
i1	Certificate Request Input

Certificate Profile Input Editor

Select one of the following inputs:

- Certificate Renewal Request Serial Number Input
- Certificate Request Input
- CMC Certificate Request Input**
- Dual Key Generation Input
- Encryption Key Generation Input
- Encryption Key Generation Input

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**.



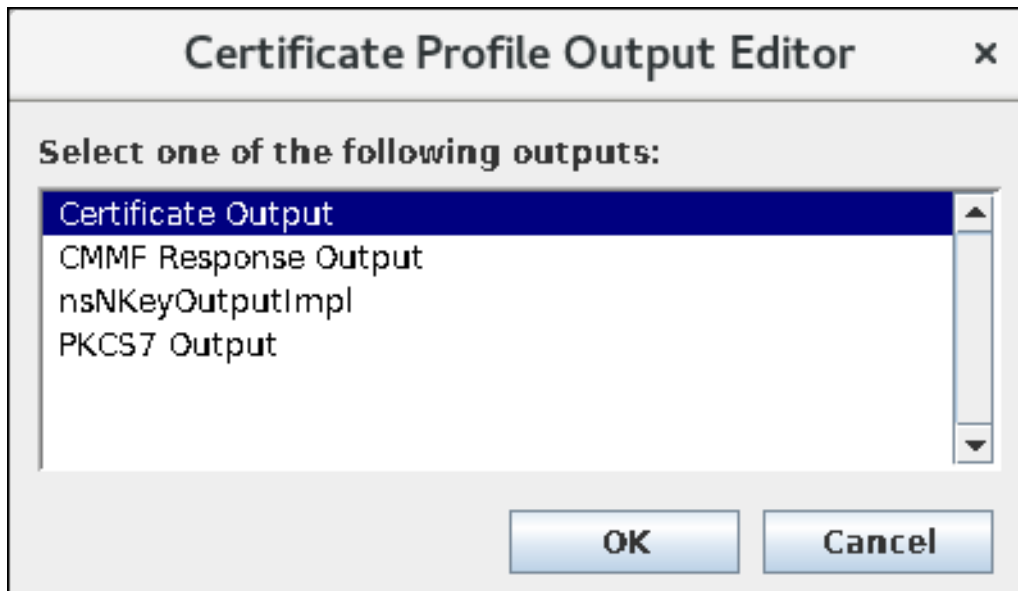
Inputs can be added and deleted. It is possible to select edit for an input, but since inputs have no parameters or other settings, there is nothing to configure.

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.

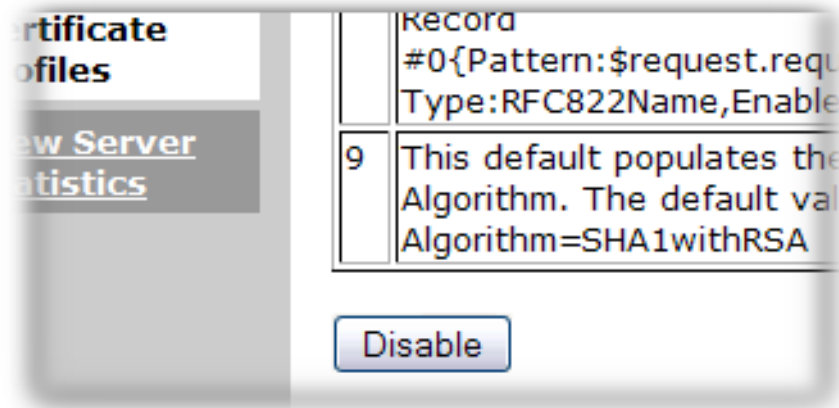
```
systemctl restart pki-tomcatd-nuxwdog@instance_name.service
```

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 recognize 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. Make 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.

For a list of supported profiles, see [Section 8.1.2, "CMC Authentication Plug-ins"](#).

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 with 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.

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
policysset.cmcUserCertSet.1.default.params.name=
policysset.cmcUserCertSet.4.default.class_id=authorityKeyIdentifierExtDefaultImpl
policysset.cmcUserCertSet.6.default.params.keyUsageKeyCertSign=false
policysset.cmcUserCertSet.10.default.class_id=noDefaultImpl
policysset.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:

```
policysset.set1.p3.constraint.class_id=noConstraintImpl
policysset.set1.p3.constraint.name=No Constraint
policysset.set1.p3.default.class_id=subjectKeyIdentifierExtDefaultImpl
policysset.set1.p3.default.name=Subject Key Identifier Default
...
policysset.set1.p11.constraint.class_id=keyConstraintImpl
policysset.set1.p11.constraint.name=Key Constraint
policysset.set1.p11.constraint.params.keyType=RSA
```

```

policysset.set1.p11.constraint.params.keyParameters=1024,2048,3072,4096
policysset.set1.p11.default.class_id=userKeyDefaultImpl
policysset.set1.p11.default.name=Key Default

```

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

```

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

```

3.4. CONFIGURING PROFILES TO ENABLE RENEWAL

Renewing a certificate regenerates the certificate using the same public key as the original certificate. Renewing a certificate can be preferable to simply generating new keys and installing new certificates; for example, if a new CA signing certificate is created, all of the certificates which that CA issued and signed must be reissued. If the CA signing certificate is renewed, then all of the issued certificates are still valid. A renewed certificate is identical to the original, only with an updated validity period and expiration date.

This section discusses how to set up profiles for renewals.

3.4.1. About Renewal

A renewed certificate is identical to the original certificate, which makes renewing certificates a much simpler and cleaner option for handling the expiration of many kinds of certificates, especially CA signing certificates.

3.4.1.1. The Renewal Process

There are two methods of renewing a certificate. *Regenerating* the certificate 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. *Re-keying* a certificate submits a certificate request through the original profile with the same information, so that a new key pair is generated.

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

```

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

```

3.4.1.1.1. 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:

```

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

```

3.4.1.1.2. 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 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.

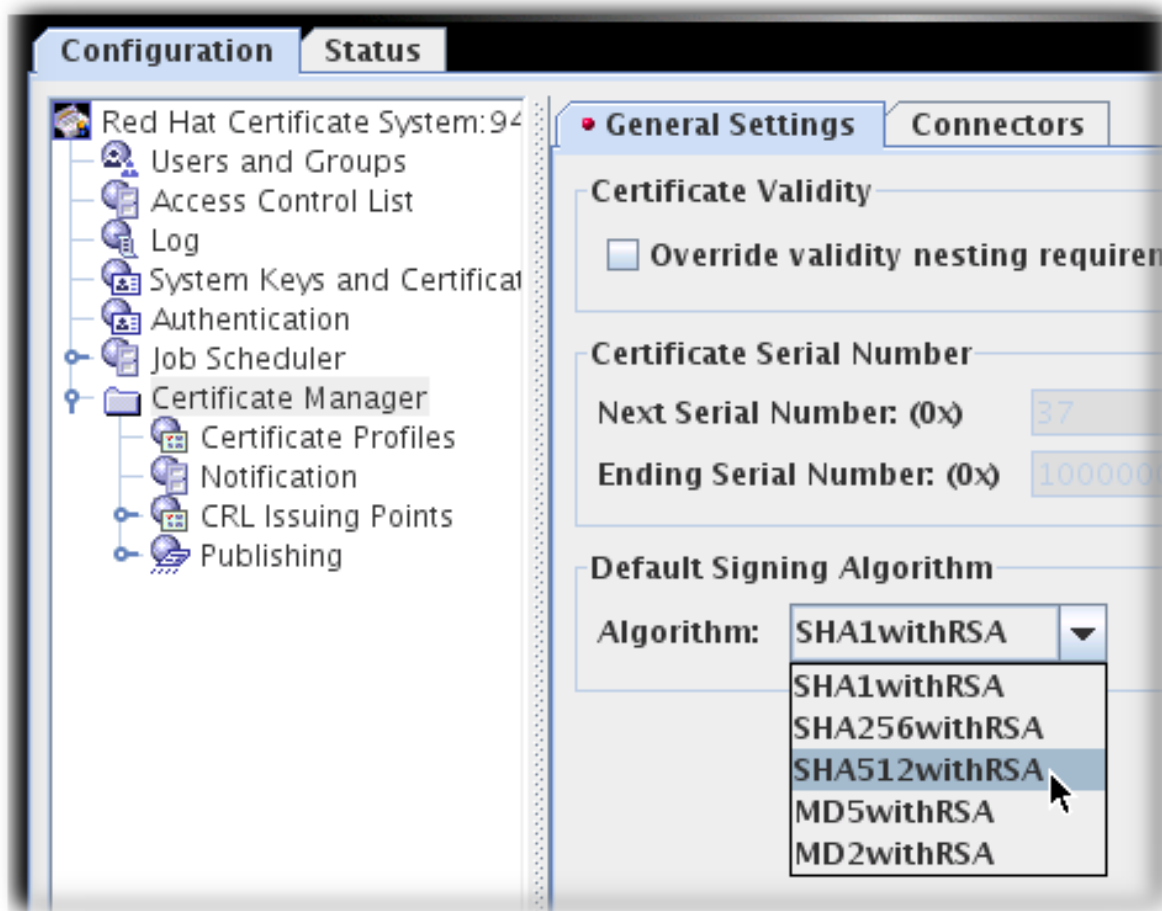
Each profile can define which cipher suite the CA should use to sign certificates processed through that profile. If no signing algorithm is set, then the profile uses whatever the default signing algorithm is.

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.



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:

```

policysset.cmcUserCertSet.8.constraint.class_id=signingAlgConstraintImpl
policysset.cmcUserCertSet.8.constraint.name=No Constraint
policysset.cmcUserCertSet.8.constraint.params.signingAlgsAllowed=SHA256withRSA,SHA512withRSA,SHA256withEC,SHA384withRSA,SHA384withEC,SHA512withEC
policysset.cmcUserCertSet.8.default.class_id=signingAlgDefaultImpl
policysset.cmcUserCertSet.8.default.name=Signing Alg
policysset.cmcUserCertSet.8.default.params.signingAlg=-

```

To configure the Signing Algorithm Default through the console:



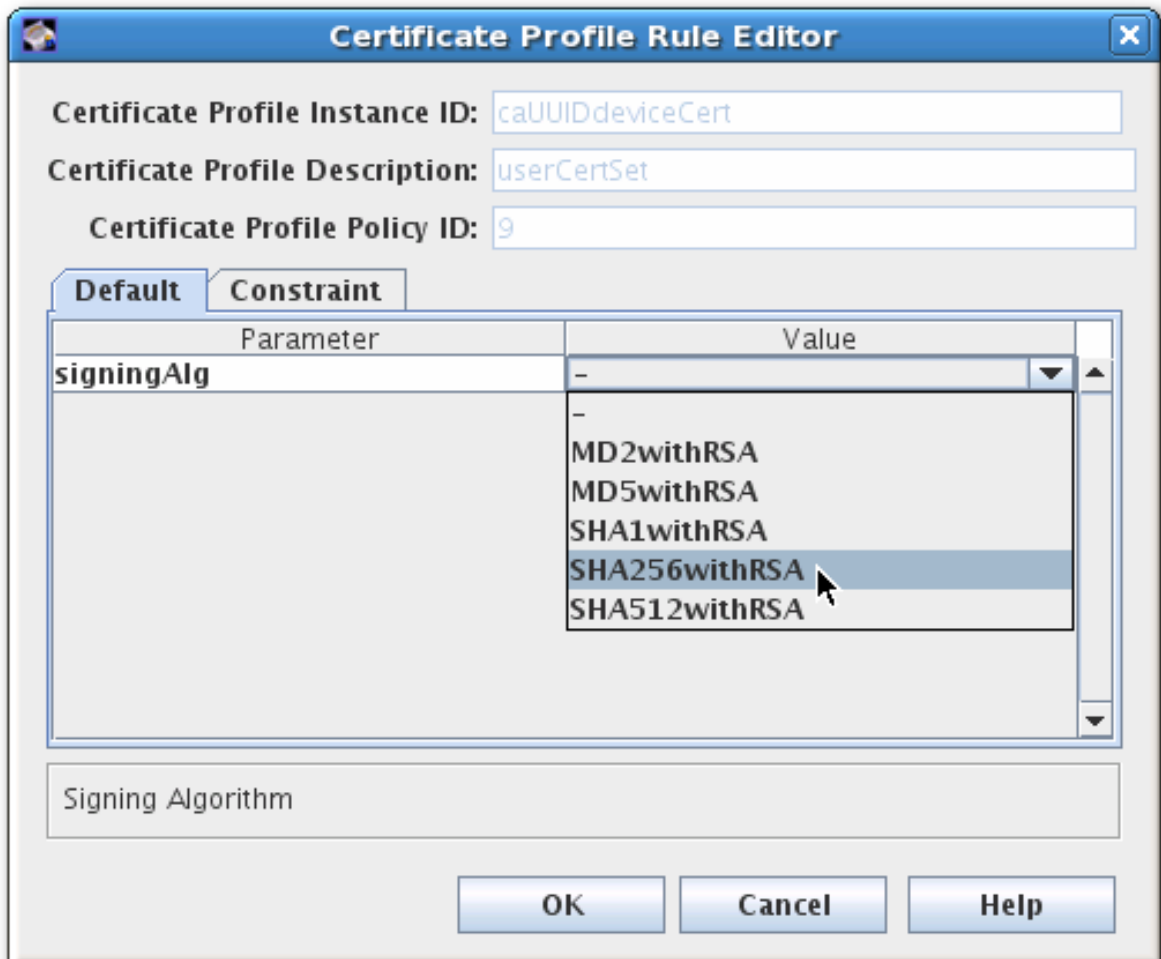
NOTE

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"](#).

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.

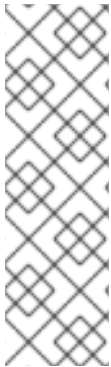
A certificate chain generally consists of an entity certificate, zero or more intermediate CA certificates, and a root CA certificate. The root CA certificate is either self-signed or signed by an external trusted CA. Once issued, the root CA certificate is loaded into a certificate database as a trusted 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.1.14, “Key Usage Extension Default”](#) and [Section B.1.9, “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.1.9, “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.

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.

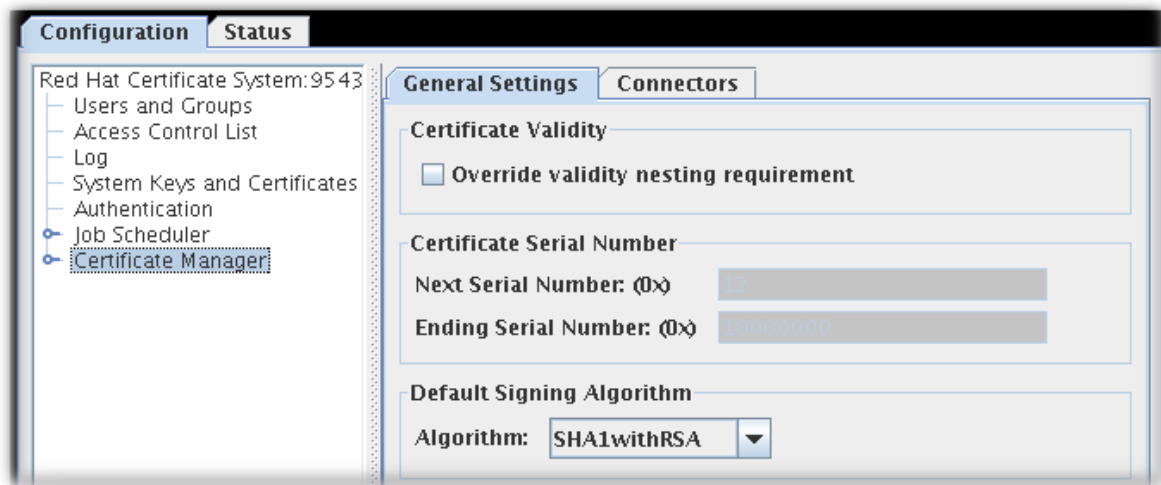


Figure 3.1. The General Settings Tab in non-subordinate CAs by default

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.

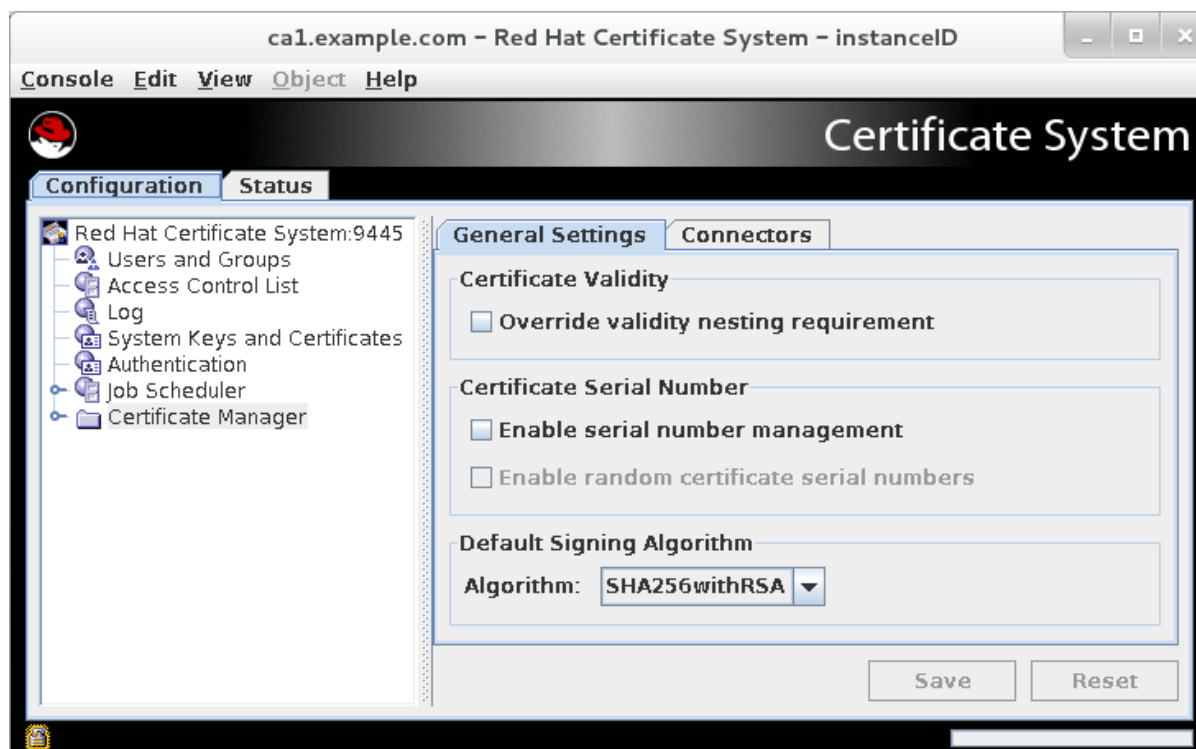


Figure 3.2. The General Settings Tab in cloned CAs by default

5. Click **Save**.

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.

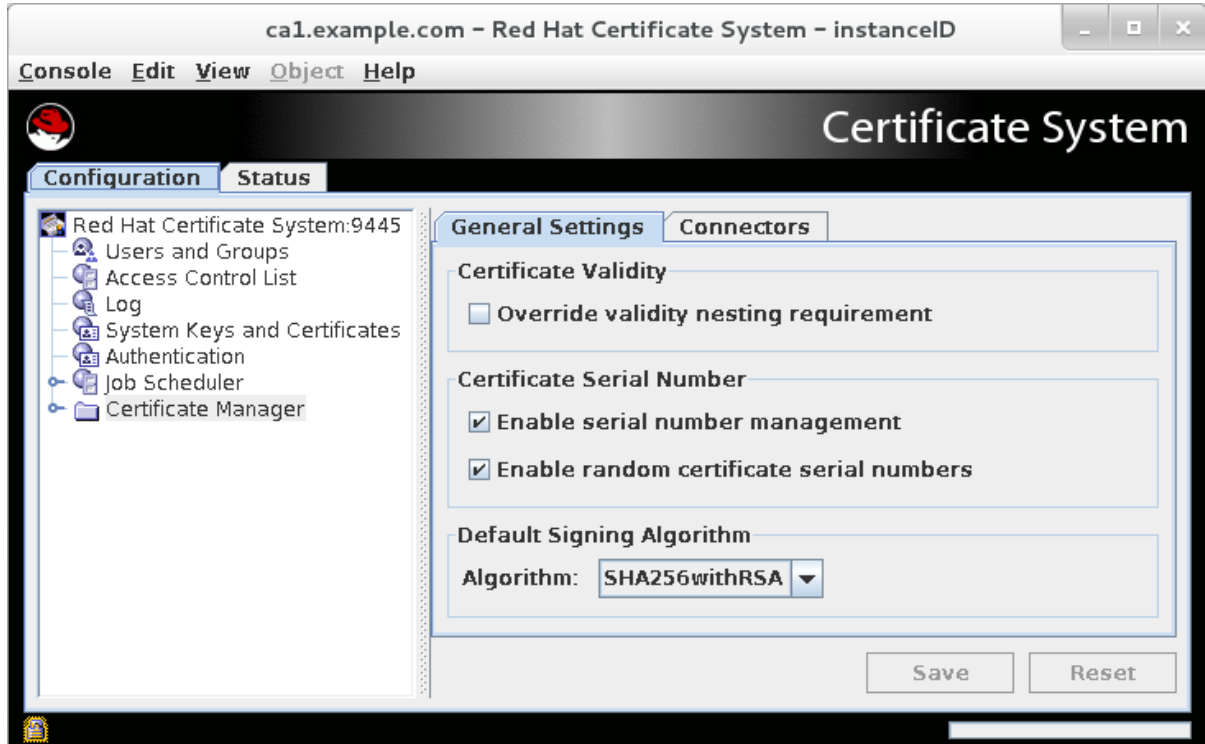


Figure 3.3. The General Settings Tab when Random Serial Number Assignment is enabled

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

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.

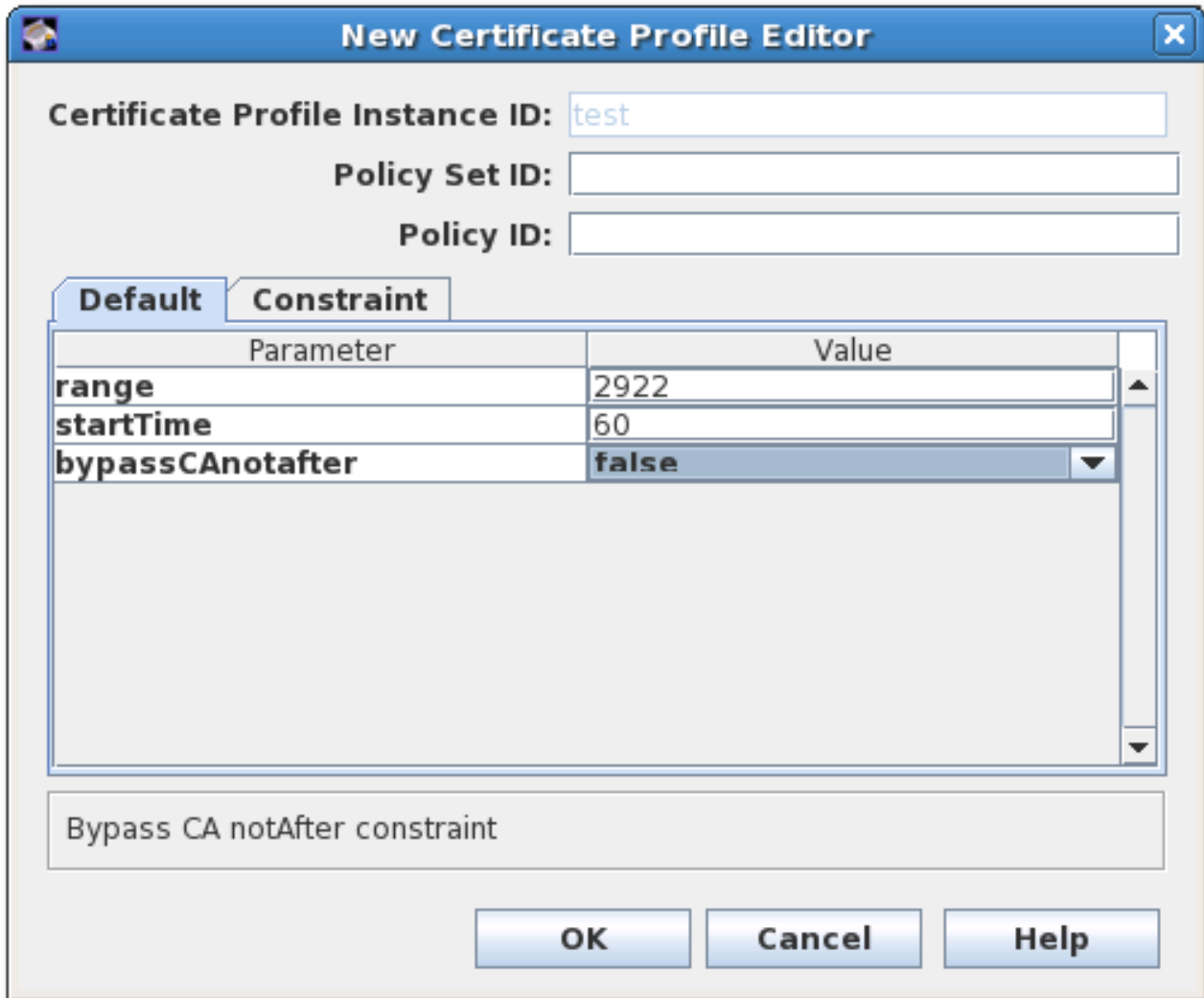


Figure 3.4. CA Validity Default Configuration

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/conf/ca/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.

```

policysset.caCertSet.2.default.name=CA Certificate Validity Default
policysset.caCertSet.2.default.params.range=2922
policysset.caCertSet.2.default.params.startTime=0
policysset.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.

The screenshot shows the 'Certificate Manager' interface. On the left is a sidebar with navigation options: List Requests, Search for Requests, List Certificates, Search for Certificates, Revoke Certificates, Display Revocation List, Update Revocation List, Update Directory Server, and OCSP Service. The main area displays 'Policy Information' for 'Certificate Profile caCertSet' and 'Set Id: caCertSet'. A table lists 'Extensions / Fields' with three entries. Entry 2 is highlighted with a red box and shows the 'Bypass CA notAfter constraint' set to 'true'. The table also shows 'Not Before' and 'Not After' dates.

#	Extensions / Fields	Const
1	This default populates a User-Supplied Certificate Subject Name to the request. <i>Subject Name:</i> CN=Certificate Authority,OU=pki-ca,C	This c
2	This default populates a Certificate Validity to the request. The default values are Range=2922 in days <i>Not Before:</i> 2011-12-21 11:47:18 <i>Not After:</i> 2020-12-21 11:47:18 <i>Bypass CA notAfter constraint:</i> true	This c
3	This default populates a User-Supplied Certificate Key to the request. <i>Key Type:</i> RSA - 1.2.840.113549.1.1.1	This c

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



NOTE

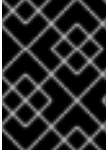
The CA Validity Default only applies to CA signing certificate renewals. Other certificates must still be issued and renewed within the CA's validity period.

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:

```

policysset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policysset.serverCertSet.1.constraint.name=Subject Name Constraint
policysset.serverCertSet.1.constraint.params.pattern=CN=[^,]+,+.+
policysset.serverCertSet.1.constraint.params.accept=true
policysset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policysset.serverCertSet.1.default.name=Subject Name Default
policysset.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:

```

policysset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policysset.serverCertSet.1.constraint.name=Subject Name Constraint
policysset.serverCertSet.1.constraint.params.pattern=UID=[^,]+,+.+
policysset.serverCertSet.1.constraint.params.accept=true
policysset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policysset.serverCertSet.1.default.name=Subject Name Default
policysset.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.1.28, "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:


```

policysset.userCertSet.8.default.class_id=subjectAltNameExtDefaultImpl
policysset.userCertSet.8.default.name=Subject Alt Name Constraint
policysset.userCertSet.8.default.params.subjAltNameExtCritical=false
policysset.userCertSet.8.default.params.subjAltExtType_0=RFC822Name
policysset.userCertSet.8.default.params.subjAltExtPattern_0=$request.requestor_email$
policysset.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.1.24, "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.

For information on setting a CMC shared token, see [Section 8.1.3.2, "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.1.28, "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"](#).

- 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.
\$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.

Policy Set Token	Description
\$request.request_name\$	The person who submitted the request.
\$request.upn\$	The Microsoft UPN. This has the format <i>(UTF8String)1.3.6.1.4.1.311.20.2.3,\$request.upn\$</i> .
\$server.source\$	Instructs the server to generate a version 4 UUID (random number) component in the subject name. This always has the format <i>(IA5String)1.2.3.4,\$server.source\$</i> .
\$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:

```
policysset.serverCertSet.12.constraint.class_id=noConstraintImpl
policysset.serverCertSet.12.constraint.name=No Constraint
policysset.serverCertSet.12.default.class_id=commonNameToSANDefaultImpl
policysset.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.1.33, "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 5.3.1, "The CMC Enrollment Process"](#) to complete the CMC enrollment.

CHAPTER 4. SETTING UP KEY ARCHIVAL AND RECOVERY

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



NOTE

Server-side key generation is an option provided for smart card enrollments performed through the TPS subsystem. This chapter deals with archiving keys through client-side key generation, not the server-side key generation and archivals initiated through the TPS.



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.

Archiving private keys offers protection for users, and for information, if that key is ever lost. Information is encrypted by the public key when it is stored. The corresponding private key must be available to decrypt the information. If the private key is lost, the data cannot be retrieved. A private key can be lost because of a hardware failure or because the key's owner forgets the password or loses the hardware token in which the key is stored. Similarly, encrypted data cannot be retrieved if the owner of the key is unavailable to supply it.



NOTE

In a cloned environment, it is necessary to set up key archival and recovery manually.

4.1. ABOUT KEY ARCHIVAL AND RECOVERY

From the end user perspective, key archival requires only two things: a client, such as the **pki** utility, which can generate a key and a certificate profile which is configured to support key archival. For details about the **pki** utility, see the *Command-Line Interface (CLI)* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

4.1.1. Key Archival

If an end entity loses a private encryption key or is unavailable to use the private key, the key must be recovered before any data that was encrypted with the corresponding public key can be read. Recovery is possible if the private key was archived when the key was generated.

The KRA stores private encryption keys in a secure key repository in its internal database in form of key records. The Certificate Manager automatically forwards certificate requests issued by a client to the KRA, if the requests contain the key archival option. In such a case, the private key is wrapped by the KRA transport key on the client and sent as an encrypted blob to the KRA. The KRA decrypts the blob and re-encrypts the key using its storage key. The KRA then gives the encrypted blob a unique key identifier and archives it in its key repository as a key record.

The archived copy of the key remains wrapped with the KRA's storage key. It can be decrypted, or unwrapped, only by using the corresponding private key pair of the storage certificate. A combination of one or more key recovery (or KRA) agents' certificates authorizes the KRA to complete the key recovery

to retrieve its private storage key and use it to decrypt/recover an archived private key. The KRA indexes stored keys by key number, owner name, and a hash of the public key, allowing for highly efficient searching.

Figure 4.1, “How the Key Archival Process Works” illustrates how the key archival process occurs when an end entity requests a certificate.

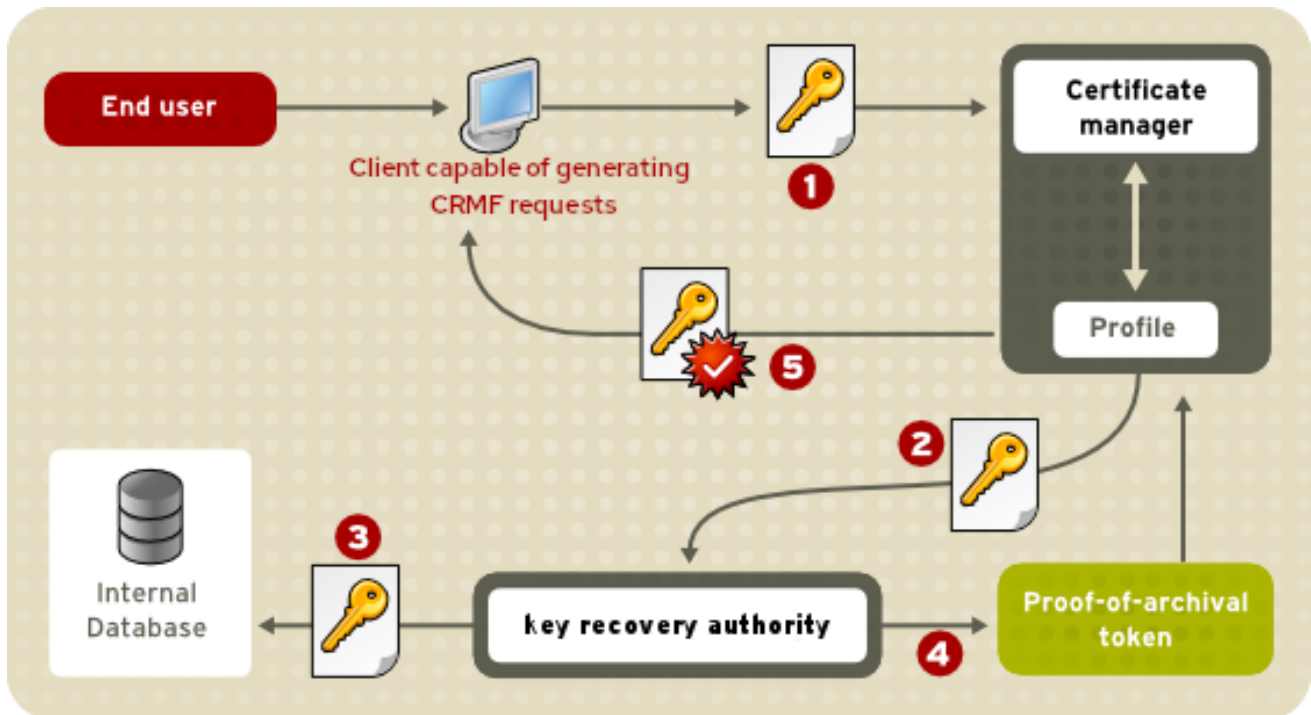


Figure 4.1. How the Key Archival Process Works

1. The client generates and encrypts its private key and submits the request to the CA.
2. After approving the certificate request and issuing the certificate, the Certificate Manager sends it to the KRA for storage, along with the public key. The Certificate Manager waits for verification from the KRA that the private key has been received and stored and that it corresponds to the public encryption key.
3. The KRA decrypts it with the transport private key. After confirming that the private encryption key corresponds to the public encryption key, the KRA encrypts it again with its public key pair of the storage key before storing it in its internal database.
4. Once the private encryption key has been successfully stored, the KRA uses the private key of its transport key pair to sign a token confirming that the key has been successfully stored; the KRA then sends the token to the Certificate Manager.
5. The Certificate Manager issues the certificate, which is embedded in the CMC response.

Both subsystems subject the request to configured certificate profile constraints at appropriate stages. If the request fails to meet any of the profile constraints, the subsystem rejects the request.

The KRA supports agent-initiated key recovery, when designated recovery agents use the key recovery form on the KRA agent services page to process and approve key recovery requests. With the approval of a specified number of agents, an organization can recover keys when the key's owner is unavailable or when keys have been lost.

Certificate System 9 uses an *m-of-n ACL-based recovery scheme*, rather than an older secret-splitting-based recovery scheme. In versions of Certificate System older than 7.1, the password for the storage

token was split and protected by individual recovery agent passwords. Now, Certificate System uses its existing access control scheme to ensure recovery agents are appropriately authenticated over TLS and requires that the agent belong to a specific recovery agent group, by default the Key Recovery Authority Agents Group. The recovery request is executed only when *m-of-n* (a required number of) recovery agents have granted authorization to the request.

The key recovery scheme can be changed by changing the KRA configuration, as described in the *Setting up Agent-Approved Key Recovery Schemes* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

For the actual key recovery process, one of the key recovery agents informs all required recovery agents about an impending key recovery and initiates the recovery process in the KRA's agent pages. A key recovery process can either be *synchronous*, meaning that the initial session must remain open as approvals come in, or *asynchronous*, meaning that snapshots of the recovery process are stored in the internal database and updated as approvals come in.

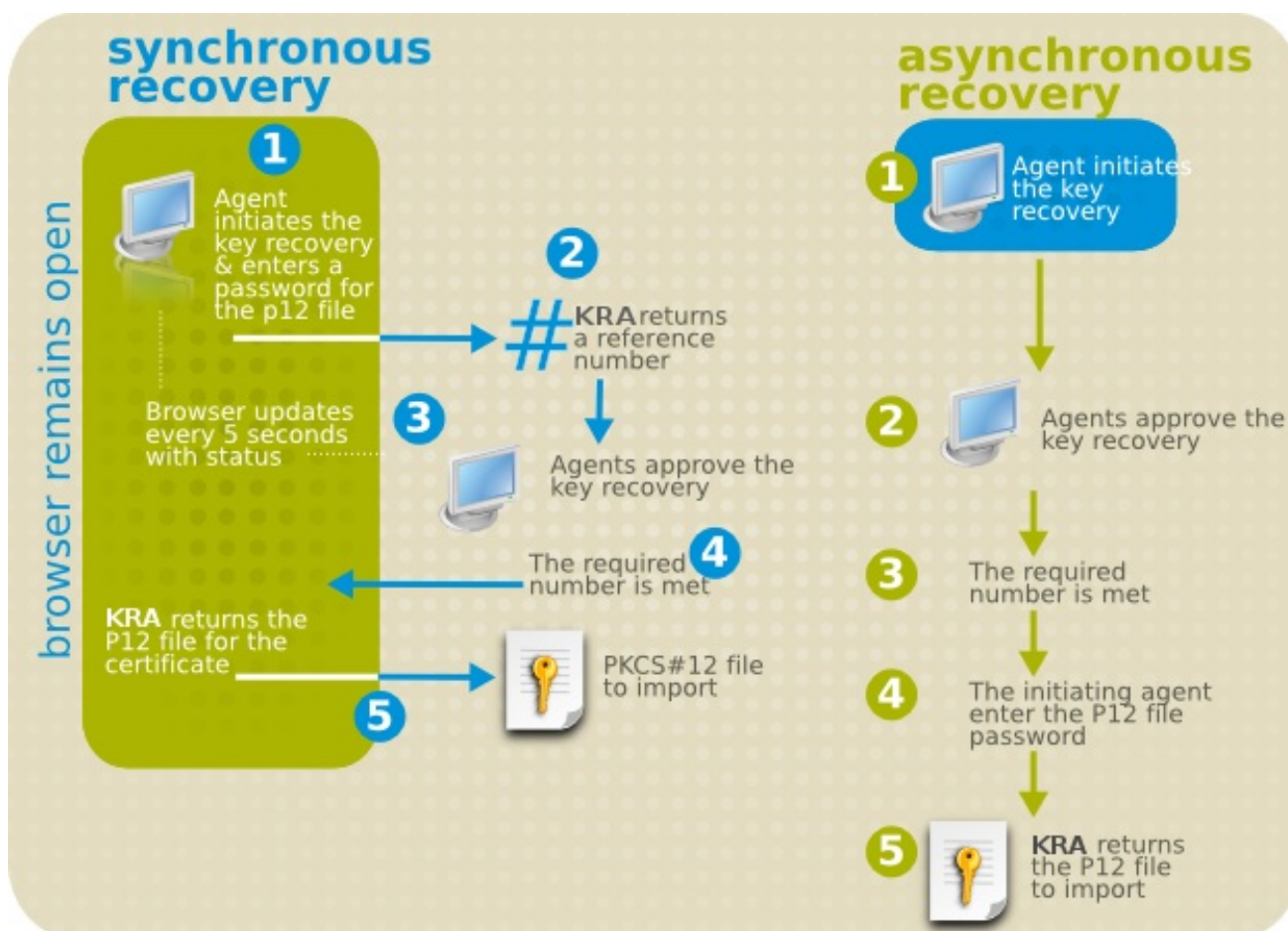


Figure 4.2. Async and Sync Recovery, Side by Side



IMPORTANT

Synchronous recovery is deprecated in Red Hat Certificate System.

For asynchronous recovery, each step is saved to the internal database. Agents can search for the key to recover and then click the **Grant Recovery** button to approve the recovery. Asynchronous recoveries will persist even if the KRA is restarted.

4.1.2. Key Recovery

To test whether a key can be successfully archived and recovered:

1. Generate a CRMF request and submit it through the CA's enrollment portal.

For more information, see [Section 5.3.2.3.1, "Example on Obtaining an Encryption-only certificate with Key Archival"](#).

2. Import the certificates into an email client capable of doing SMIME.
3. 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.
4. 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.
5. Delete the certificate. Check the encrypted email again; the mail client should not be able to decrypt the message.
6. 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.
 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**.

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

8. 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 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 14, *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 user generates a certificate request.
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 user retrieves the new certificate.

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

5.2. CREATING CERTIFICATE SIGNING REQUESTS

As explained in the *Enrolling with CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*, the **CMCRequest** utility accepts Certificate Signing Requests (CSR) in PKCS #10 and CRMF format.

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.

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

5.2.1. 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. 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.bin.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

MIICbTCCAUVCAQAwwKDEQMA4GA1UEChMHRXhhbXBsZTEUMBIGA1UEAxMLZXhhbXBs
...
```

This is a PKCS#10 PEM certificate request.

5. For the next steps, see [Section 5.3.1, "The CMC Enrollment Process"](#), but skip the step about creating the certificate request.

5.2.1.2. 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.

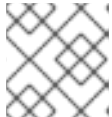
3. Optionally, verify that the CSR file is correct:

```
$ 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.

4. For the next steps, see [Section 5.3.1, "The CMC Enrollment Process"](#), but skip the step about creating the certificate request.



NOTE

Remove the header information from the CSR.

5.2.2. 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

5.2.2.1. 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-----
MIICzzCCAAbcCAQAwgYkx
...
-----END CERTIFICATE REQUEST-----
```

5.2.2.2. 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-----
MIICzzCCAAbcCAQAwgYkx
...
-----END CERTIFICATE REQUEST-----
```

5.2.3. 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.

5.2.3.1. 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 Algorithm: 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-----
MIICzzCCAAbcCAQAwgYkx
...
-----END CERTIFICATE REQUEST-----
```

5.2.3.2. 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 Algorithm: 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-----
MIICzzCCAAbcCAQAwgYkx
...
-----END CERTIFICATE REQUEST-----
```

5.3. REQUESTING AND RECEIVING CERTIFICATES 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 (Common Criteria Edition)*.
- The *Enrolling with CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.
- CMCRequest(1) man page
- CMCRresponse(1) man page

CMC enrollment is possible in various ways to meet the requirements for different scenarios. [Section 5.3.1, “The CMC Enrollment Process”](#) supplements the *Enrolling with CMC* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)* with more details. Additionally, the [Section 5.3.2, “Practical CMC Enrollment Scenarios”](#) section enables administrators to decide which mechanisms should be used in which scenario.

5.3.1. 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 [Chapter 11, *Managing Certificate/Key Crypto Token*](#):

```
$ 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.

- 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 [Section 8.1.2, "CMC Authentication Plug-ins"](#).

- Submit the CMC request to the CA:

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

- 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
```

5.3.2. 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 5.3.1, “The CMC Enrollment Process”](#).

5.3.2.1. Obtaining System and Server Certificates

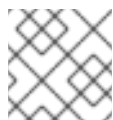
If a service, such as LDAP or a web server, requires a TLS server certificate, the administrator of this server creates a CSR based on the documentation of the service and sends it to the CA's agent for approval. Use the procedure described in [Section 5.3.1, “The CMC Enrollment Process”](#) for this process. Additionally, consider the following requirements:

Enrollment Profiles

The agent must either use one of the existing CMC profiles listed in [Section 8.1.2, “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 **PopLinkWitnessV2** feature must be disabled because the identification is checked by the agent.

5.3.2.2. 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 5.3.2.2.1, "Signing a CMC Request with an Agent Certificate"](#).
- Certificate enrollment is authenticated by using a Shared Secret. See [Section 5.3.2.2.2, "Authenticating for Certificate Enrollment Using a Shared Secret"](#).

5.3.2.2.1. 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 5.3.2.1, "Obtaining System and Server Certificates"](#). The only difference is that the user creates the CSR and sends it to a CA agent for approval.

5.3.2.2.2. 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 5.3.2.2.1, “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 5.3.1, “The CMC Enrollment Process”](#):

1. Create a Shared Token either as the user or CA administrator. For details, see [Section 8.1.3.1, “Creating a Shared Secret Token”](#).

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 8.1.3.2.1, “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 (Common Criteria Edition)*.
 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***

5.3.2.3. 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 5.3.2.2, "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.

5.3.2.3.1. 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 5.2.3.1, "Using CRMFPopClient to Create a CSR with Key Archival"](#) and [Section 5.2.3.2, "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.privKeyld** 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 cert8.db, key3.db and secmod.db
dbdir=/home/user_name/.dogtag/nssdb/

#password: password for cert8.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 cert8.db, key3.db and secmod.db
#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 cert8.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 cert8.db, key3.db and secmod.db
dbdir=/home/user_name/.dogtag/nssdb/

#password: password for cert8.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 **DecryptPOP** 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 cert8.db, key3.db and secmod.db
#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 cert8.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
```

5.4. RENEWING CERTIFICATES

This section explains how to use the different types of certificate renewal described in [Section 3.4.1, "About Renewal"](#). You can use the methods described in this section to renew a certificate both with and without agent approval. To renew a certificate as a user without agent approval, use profiles that require the **CMCUserSignedAuth** authentication plug-in, and to renew with agent approval, use profiles that require the **CMCAuth** authentication plug-in. For further details about these plug-ins and in which profiles they are enabled by default, see [Section 8.1.2, "CMC Authentication Plug-ins"](#).

5.4.1. Renewal Using the Same Key

[Section 5.3.1, "The CMC Enrollment Process"](#) describes how to request and issue a certificate using CMC. When a user submits the same CMC request created during this process again with the same enrollment profile, Certificate System renews the certificate with the same key.

**NOTE**

For renewing a certificate as the user using the same key, the enrollment profile must contain the **uniqueKeyConstraint** entry with the **params.allowSameKeyRenewal** parameter set to **True** as described in [Section 3.4.1, "About Renewal"](#) and [Section 3.4.1.1.1, "Renewing Using the Same Key"](#).

5.4.2. Renewal Using a New Key

To renew a certificate using a new key, follow the procedure described in [Section 5.3.1, "The CMC Enrollment Process"](#). The process for renewal is the same as for a new enrollment. When you sign the request with the same signing certificate, the newly issued certificate contains the same **subjectDN** attribute as the signing certificate.

5.5. TRACING ISSUED CERTIFICATE TO CSR, AND CSR TO ISSUED CERTIFICATE

This section details how a CA agent can trace an issued certificate to the original submitted CSR, and from a CSR to an issued certificate.

If a certificate request has been approved successfully, a CA agent can do the following to search for the request and see the CSR matching the certificate:

1. Access the **https://host_name:port/ca/agent/ca**.
2. Click **Search for Requests**
3. Select and fill in **Request ID Range** (for example **12** for **Lowest Request ID** and **12** for **Highest Request ID**).
4. Select **Request Type** and choose **enrollment** type.
5. Select **Request Status** and choose **completed** status
6. Make sure everything else is unselected.
7. Click **Submit**.
8. Click on the request number. You see the certificate in clear text at this point.
9. To display the CSR and certificate linking, right-click and select **This Frame** and **View Frame Source**.
 - Search for **inputList.inputName="Certificate Request"**; The request is the **inputList.inputVal** below that.
 - Search for **outputList.outputSyntax="pretty_print"**; The certificate is the **outputList.outputVal** below that.

To search from certificate to CSR:

1. Access the **https://host_name:port/ca/agent/ca**.
2. Click **Find**.
3. Click **Details**.

4. You see the certificate in clear text, along with a Request ID link. Click on the link to open the **Request** page.
5. To display the certificate and CSR linking:
 - Search for **inputList.inputName="Certificate Request"**; The request is the **inputList.inputVal** below that.
 - Search for **outputList.outputSyntax="pretty_print"**; The certificate is the **outputList.outputVal** below that.

CHAPTER 6. 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.

6.1. ABOUT REVOKING CERTIFICATES

Certificates can be revoked by an end user (the original owner of the certificate) or by a Certificate Manager agent. An end user can revoke only certificates that contain the same subject name as the certificate presented for authentication.

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 6.3.3, "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 9.2.3.11. Setting a CA to Use a Different Certificate to Sign CRLs in Red Hat Certificate System's Planning, Installation, and Deployment Guide.

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.

6.1.1. 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.

6.1.2. 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.

6.1.3. 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 7, *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 7.7, "Setting up Resumable CRL Downloads"](#).

6.2. REVOKING CERTIFICATES

6.2.1. Performing a CMC Revocation

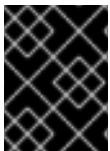
Similar to Certificate Management over CMS (CMC) enrollment, CMC revocation enables users to set up a revocation client, and sign the revocation request with either an agent certificate or a user certificate with a matching **subjectDN** attribute. Then the user can send the signed request to the Certificate Manager.

Alternatively, CMC revocation can also be authenticated using the Shared Secret Token mechanism. For details, see [Section 8.1.3, "CMC SharedSecret Authentication"](#).

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 6.2.1.1, "Revoking a Certificate Using CMCRequest"](#).
- **CMCRevoke**. For details, see [Section 6.2.1.2, "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**.

6.2.1.1. 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 cert8.db, key3.db and secmod.db

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

#password: password for cert8.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 TLS client authentication certificate
#can be found (default is internal)
#This parameter will be ignored if secure=false
tokenname=internal

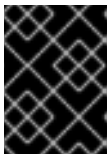
#dbdir: directory for cert8.db, key3.db and secmod.db
#This parameter will be ignored if secure=false
dbdir=/home/user_name/.dogtag/nssdb/

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

#password: password for cert8.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/profileSubmitUserSignedCMCFull
```

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.

6.2.1.2. 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).

**IMPORTANT**

Using **CMCRevoke** requires that the CA administrator followed the instructions specified in the *Enabling CMCRevoke for the Web User Interface* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)* during the installation.

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*.

6.2.1.2.1. 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.

6.2.2. Performing Revocation as an Agent from the Web UI

A Certificate Manager agent can use the agent services page to find a specific certificate issued by the Certificate System or to retrieve a list of certificates that match specified criteria. The certificates which are retrieved can be examined or revoked by the agent. The Certificate Manager agent can also manage the certificate revocation list (CRL).

6.2.2.1. Listing Certificates

It is possible to list certificates within a range of serial numbers. All certificates within the range may be displayed or, if the agent selects, only those that are currently valid.

To find a specific certificate or to list certificates by serial number:

1. Open the Certificate Manager agent services page.
2. Click **List Certificates**.

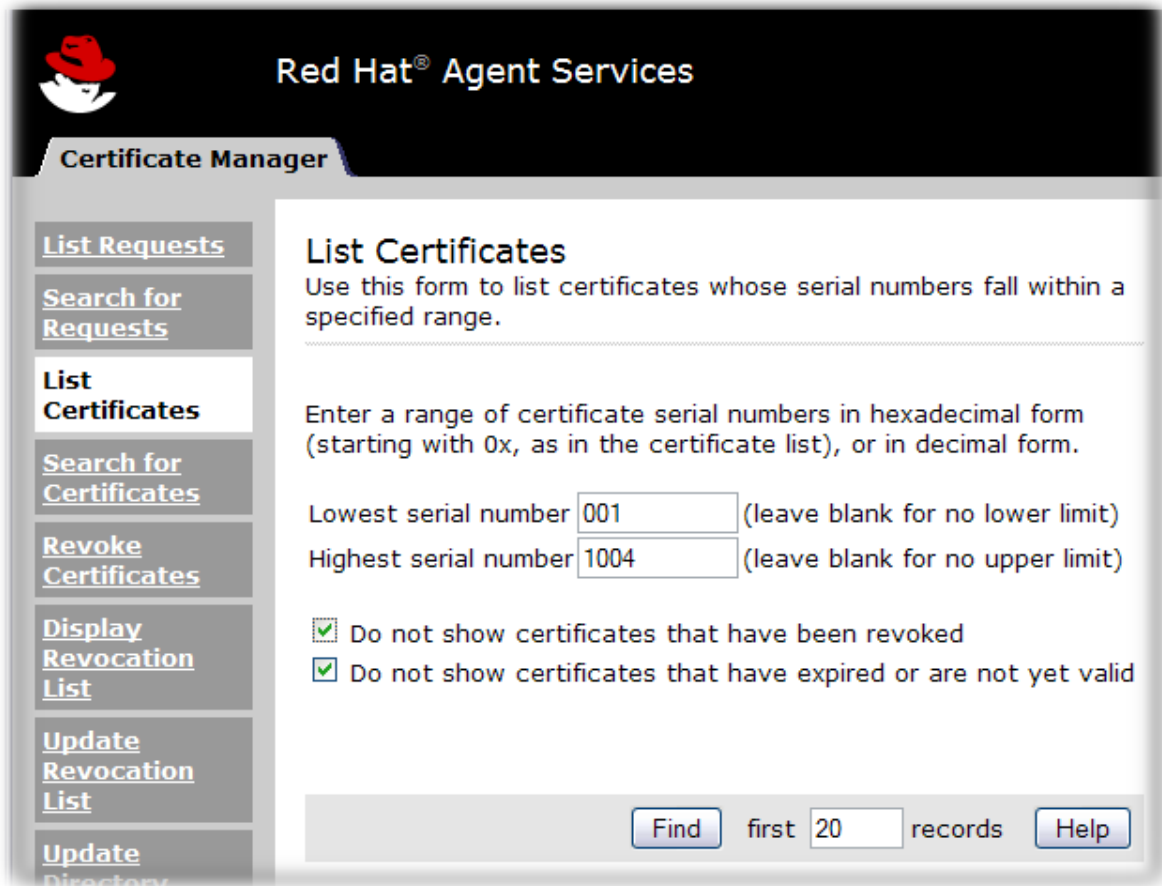


Figure 6.1. List Certificates

- To find a certificate with a specific serial number, enter the serial number in both the upper limit and lower limit fields of the **List Certificates** form, in either decimal or hexadecimal form. Use **0x** to indicate the beginning of a hexadecimal number; for example, **0x00000006**. Serial numbers are displayed in hexadecimal form in the **Search Results** and **Details** pages.
- To find all certificates within a range of serial numbers, enter the upper and lower limits of the serial number range in decimal or hexadecimal form.

Leaving either the lower limit or upper limit field blank displays the certificate with the specified number, plus all certificates before or after it in sequence.

3. To limit the returned list to valid certificates, select the check boxes labeled with filtering methods. It is possible to include revoked certificates, to include expired certificates or certificates that are not yet valid, or to display only valid certificates.
4. Enter the maximum number of certificates matching the criteria that should be returned in the results page.

When any number is entered, the first certificates up to that number matching the criteria are displayed.

5. Click **Find**.

The Certificate System displays a list of the certificates that match the search criteria. Select a certificate in the list to examine it in more detail or perform various operations on it. For more information, refer to [Section 6.2.2.3, "Examining Certificate Details"](#).

6.2.2.2. Searching for Certificates (Advanced)

Search for certificates by more complex criteria than serial number using the advanced search form. To perform an advanced search for certificates:

1. Open the Certificate Manager agent services page. The agent must submit the proper client certificate to access this page.
2. Click **Search for Certificates** to display the **Search for Certificates** form to specify search criteria.

The screenshot shows the Red Hat Agent Services interface. The main heading is 'Red Hat® Agent Services' with the Red Hat logo. Below it is a 'Certificate Manager' tab. A sidebar on the left contains several menu items: 'List Requests', 'Search for Requests', 'List Certificates', 'Search for Certificates' (which is highlighted), 'Revoke Certificates', 'Display Revocation List', 'Update Revocation List', and 'Update Directory'. The main content area is titled 'Search for Certificates' and contains the following text: 'Use this form to compose queries based on properties of the certificate.' Below this is a paragraph: 'Each section below filters the search. Check the box at the top of the section if you want to use that filter in your search, then complete the fields. Leave a box unchecked to ignore that filter. You can click more than one box to get a combination of search criteria.' There are two filter sections: 'Serial Number Range' and 'Status'. The 'Serial Number Range' section has a checkbox, two input fields for 'Lowest serial number' and 'Highest serial number', and a note: 'Enter a range of certificate serial numbers in hexadecimal form (starting with 0x, as in the certificate list), or in decimal form.' The 'Status' section has a checkbox and a dropdown menu currently set to 'VALID'.

3. To search by particular criteria, use one or more of the sections of the **Search for Certificates** form. To use a section, select the check box, then fill in any necessary information.
 - *Serial Number Range*. Finds a certificate with a specific serial number or lists all certificates within a range of serial numbers.
 - To find a certificate with a specific serial number, enter the serial number in both the upper limit and lower limit fields in either decimal or hexadecimal. Use **0x** to indicate the beginning of a hexadecimal number, such as **0x2A**. Serial numbers are displayed in hexadecimal form in the **Search Results** and **Details** pages.
 - To find all certificates within a range of serial numbers, enter the upper and lower limits of the serial number range in decimal or hexadecimal. Leaving either the lower limit or upper limit field blank returns all certificates before or after the number specified.
 - *Status*. Selects certificates by their status. A certificate has one of the following status codes:
 - *Valid*. A valid certificate has been issued, its validity period has begun but not ended, and it has not been revoked.
 - *Invalid*. An invalid certificate has been issued, but its validity period has not yet begun.
 - *Revoked*. The certificate has been revoked.

- *Expired.* An expired certificate has passed the end of its validity period.
- *Revoked and Expired.* The certificate has passed its validity period and been revoked.
- *Subject Name.* Lists certificates belonging to a particular owner; it is possible to use wildcards in this field.



NOTE

Certificate System certificate request forms support all UTF-8 characters for the common name, organizational unit, and requester name fields. The common name and organization unit fields are included in the subject name of the certificate. This means that the searches for subject names support UTF-8 characters.

This support does not include supporting internationalized domain names.

- *Revocation Information.* Lists certificates that have been revoked during a particular period, by a particular agent, or for a particular reason. For example, an agent can list all certificates revoked between July 2005 and April 2006 or all certificates revoked by the agent with the username **admin**.
 - To list certificates revoked within a time period, select the day, month, and year from the drop-down lists to identify the beginning and end of the period.
 - To list certificates revoked by a particular agent, enter the name of the agent; it is possible to use wildcards in this field.
 - To list certificates revoked for a specific reason, select the revocation reasons from the list.
- *Issuing Information.* Lists certificates that have been issued during a particular period or by a particular agent. For example, an agent can list all certificates issued between July 2005 and April 2006 or all certificates issued by the agent with the username **jsmith**.
 - To list certificates issued within a time period, select the day, month, and year from the drop-down lists to identify the beginning and end of the period.
 - To list certificates issued by a particular agent, enter the name of the agent; it is possible to use wildcards in this field.
 - To list certificates enrolled through a specific profile, enter the name of the profile.
- *Dates of Validity.* List certificates that become effective or expire during a particular period. For example, an agent can list all certificates that became valid on June 1, 2003, or that expired between January 1, 2006, and June 1, 2006.

It is also possible to list certificates that have a validity period of a certain length of time, such as all certificates that are valid for less than one month.

- To list certificates that become effective or expire within a time period, select the day, month, and year from the drop-down lists to identify the beginning and end of the period.

- To list certificates that have a validity period of a certain length in time, select **Not greater than** or **Not less than** from the drop-down list, enter a number, and select a time unit from the drop-down list: days, weeks, months, or years.
 - *Basic Constraints*. Shows CA certificates that are based on the Basic Constraints extension.
 - *Type*. Lists certain types of certificates, such as all certificates for subordinate CAs. This search works only for certificates containing the Netscape Certificate Type extension, which stores type information. For each type, choose from the drop-down list to find certificates where that type is **On**, **Off**, or **Do Not Care**.
- 4. To find a certificate with a specific subject name, use the **Subject Name** section. Select the check box, then enter the subject name criteria. Enter values for the included search criteria and leave the others blank.

The standard tags or components are as follows:

- *Email address*. Narrows the search by email address.
- *Common name*. Finds certificates associated with a specific person or server.
- *UserID*. Searches certificates by the user ID for the person to whom the certificate belongs.
- *Organization unit*. Narrows the search to a specific division, department, or unit within an organization.
- *Organization*. Narrows the search by organization.
- *Locality*. Narrows the search by locality, such as the city.
- *State*. Narrows the search by state or province.
- *Country*. Narrows the search by country; use the two-letter country code, such as **US**.



NOTE

Certificate System certificate request forms support all UTF-8 characters for the common name and organizational unit fields. The common name and organization unit fields are included in the subject name of the certificate. This means that the searches for subject names or those elements in the subject name support UTF-8 characters.

This support does not include supporting internationalized domain names, such as in email addresses.

- 5. After entering the field values for the server to match, specify the type of search to perform:
 - Exact searches for certificate subject names match the exact components specified and contain none of the components left blank. Wildcards cannot be used in this type of search.
 - Partial searches for certificate subject names match the specified components, but the returned certificates may also contain values in components that were left blank. Wildcard patterns can be used in this type of search by using a question mark (?) to match an arbitrary single character and an asterisk (*) to match an arbitrary string of characters.

**NOTE**

Placing a single asterisk in a search field means that the component must be in the certificate's subject name but may have any value. Leave the field blank if it does not matter if the field is present.

- After entering the search criteria, scroll to the bottom of the form, and enter the number of certificates matching the specified criteria that should be returned.

Setting the number of certificates to be returned returns the first certificates found that match the search criteria up to that number. It is also possible to put a time limit on the search in seconds.

- Click **Find**.
- The **Search Results** form appears, showing a list of the certificates that match the search criteria. Select a certificate in the list to examine it in more detail. For more information, refer to [Section 6.2.2.3, "Examining Certificate Details"](#).

The screenshot shows the Red Hat Agent Services interface. The main heading is "Red Hat® Agent Services" with the Red Hat logo. Below it is the "Certificate Manager" section. On the left, there is a sidebar with navigation options: "List Requests", "Search for Requests", "List Certificates", "Search for Certificates", "Revoke Certificates", "Display Revocation List", and "Update Revocation List". The "Search for Certificates" option is highlighted. The main content area is titled "Search Results" and shows the following information:

Issuer: CN=Certificate Authority,O=Example Domain
 Total number of records found: 10 (Maximum size reached)

Serial number	Subject name	
0x00000001	CN=Certificate Authority, O=Example Domain	
Version	Certificate Type	Subject public key algorithm
3	X.509	PKCS #1 RSA with 2048-bit key
Not valid before		Not valid after
5/19/2009 9:35:36		5/9/2011 9:35:36
Issued on		Issued by
5/19/2009 9:35:36		system

Buttons for "Details" and "Revoke" are visible next to the certificate entry.

6.2.2.3. Examining Certificate Details

- On the agent services page, click **List Certificates** or **Search for Certificates**, specify search criteria, and click **Find** to display a list of certificates.
- On the **Search Results** form, select a certificate to examine.

If the desired certificate is not shown, scroll to the bottom of the list, specify an additional number of certificates to be returned, and click **Find**. The system displays the next certificates up to that number that match the original search criteria.

- After selecting a certificate, click the **Details** button at the left side of its entry.
- The **Certificate** page shows the detailed contents of the selected certificate and instructions for installing the certificate in a server or in a web browser.

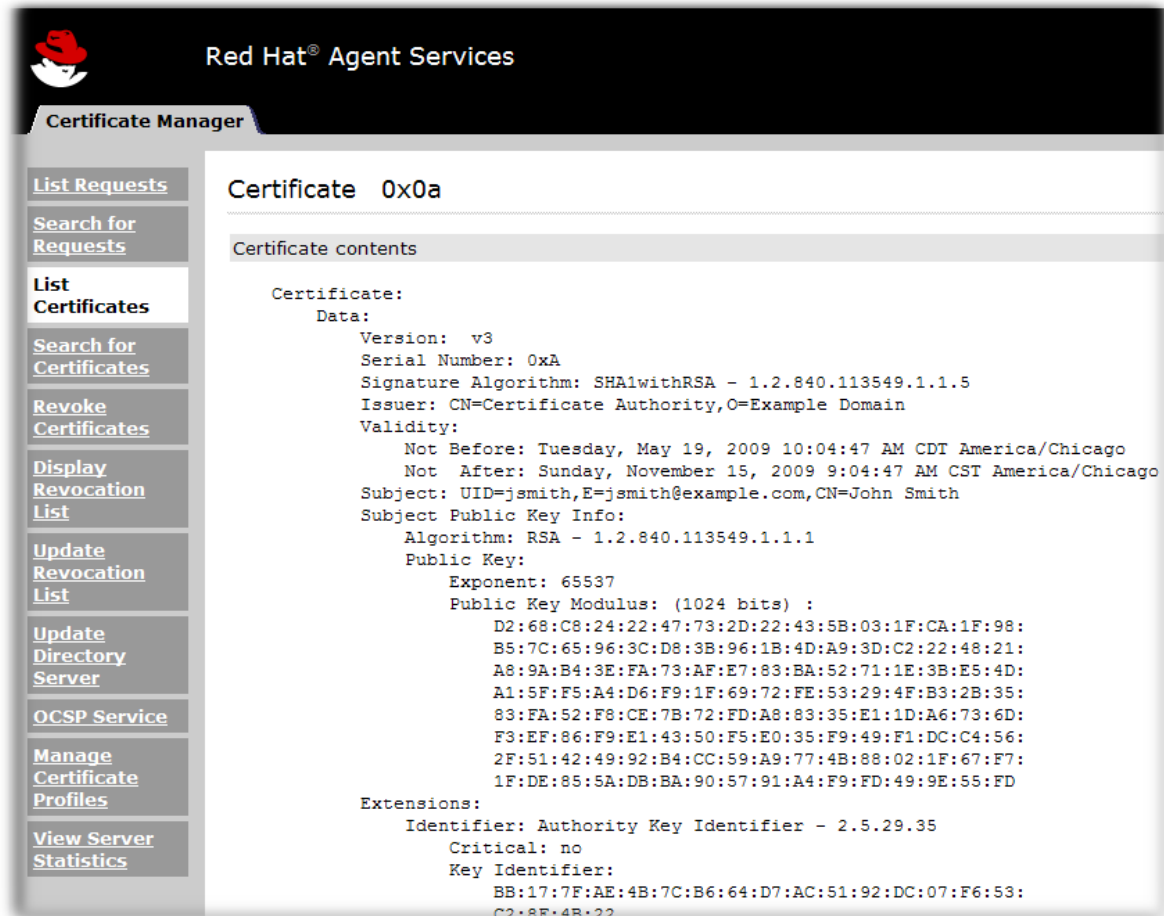


Figure 6.2. Certificate Details

- The certificate is shown in base-64 encoded form at the bottom of the **Certificate** page, under the heading **Installing this certificate in a server**.

6.2.2.4. Revoking Certificates

Only Certificate Manager agents can revoke certificates other than their own. A certificate must be revoked if one of the following situations occurs:

- The owner of the certificate has changed status and no longer has the right to use the certificate.
- The private key of a certificate owner has been compromised.

These two reasons are not the only ones why a certificate would need revoked; there are six reasons available for revoking a certificate.

To revoke one or more certificates, search for the certificates to revoke using the **Revoke Certificates** button. While the search is similar to the one through the **Search for Certificates** form, the **Search Results** form returned by this search offers the option of revoking one or all of the returned certificates.

6.2.2.4.1. Revoking Certificates

- Open the Certificate Manager agent services page.
- Click **Revoke Certificates**.

**NOTE**

The search form that appears has the same search criteria sections as the **Search for Certificates** form.

3. Specify the search criteria by selecting the check boxes for the sections and filling in the required information.
4. Scroll to the bottom of the form, and set the number of matching certificates to display.
5. Click **Find**.
6. The search returns a list of matching certificates. It is possible to revoke one or all certificates in the list.

The screenshot shows the Red Hat Agent Services Certificate Manager interface. The main content area displays search results for a certificate. The issuer is identified as 'CN=Certificate Authority, O=Example Domain'. A total of 10 records were found, which is the maximum size reached. The search results are presented in a table with the following data:

Serial number	Subject name	
0x00000001	CN=Certificate Authority, O=Example Domain	
Version	Certificate Type	Subject public key algorithm
3	X.509	PKCS #1 RSA with 2048-bit key
Not valid before		Not valid after
5/19/2009 9:35:36		5/9/2011 9:35:36
Issued on		Issued by
5/19/2009 9:35:36		system

Buttons for 'Details' and 'Revoke' are visible next to the certificate entry.

**NOTE**

If the search criteria are very specific and all of the certificates returned are to be revoked, then click the **Revoke ALL # Certificates** button at the bottom of the page. The number shown on the button is the total number of certificates returned by the search. This is usually a larger number than the number of certificates displayed on the current page.

Verify that all of the certificates returned by the search should be revoked, *not* only those displayed on the current page.

7. Click the **Revoke** button next to the certificate to be revoked.



WARNING

Whether revoking a single certificate or a list of certificates, be extremely careful that the correct certificate has been selected or that the list contains only certificates which should be revoked. Once a revocation operation has been confirmed, there is no way to undo it.

8. Select an invalidity date. The invalidity date is the date which it is known or suspected that the user's private key was compromised or that the certificate became invalid. A set of drop down lists allows the agent to select the correct invalidity date.

Red Hat® Agent Services

Certificate Manager

- List Requests
- Search for Requests
- List Certificates
- Search for Certificates
- Revoke Certificates**
- Display Revocation List
- Update Revocation List
- Update Directory Server
- OCSP Service
- Manage Certificate Profiles

Certificate Revocation Confirmation

Use this form to confirm certificate revocation by selecting appropriate revocation reason and submitting the form.

Important: When making this request you must use the browser environment in which you have access to your authentication certificate and key.

Certificate Details

The details of the certificate being revoked are below:

Serial Number: 0x0a
 Subject Name: UID=jsmith, E=jsmith@example.com, CN=John Smith
 Valid: not before: 5/19/2009 and not after: 11/15/2009

Select Invalidity Date

Please select the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid.

Invalidity date: 6 May 2009

Select Revocation Reason

Please select reason for revocation.

- Unspecified
- Key compromised
- CA key compromised
- Affiliation changed
- Certificate superseded
- Cessation of operation
- Certificate is on hold

Additional Comments

If you want to include any additional comments in your revocation request, write them

9. Select a reason for the revocation.
 - Key compromised
 - CA key compromised
 - Affiliation changed
 - Certificate superseded

- Cessation of operation
- Certificate is on hold

10. Enter any additional comment. The comment is included in the revocation request.

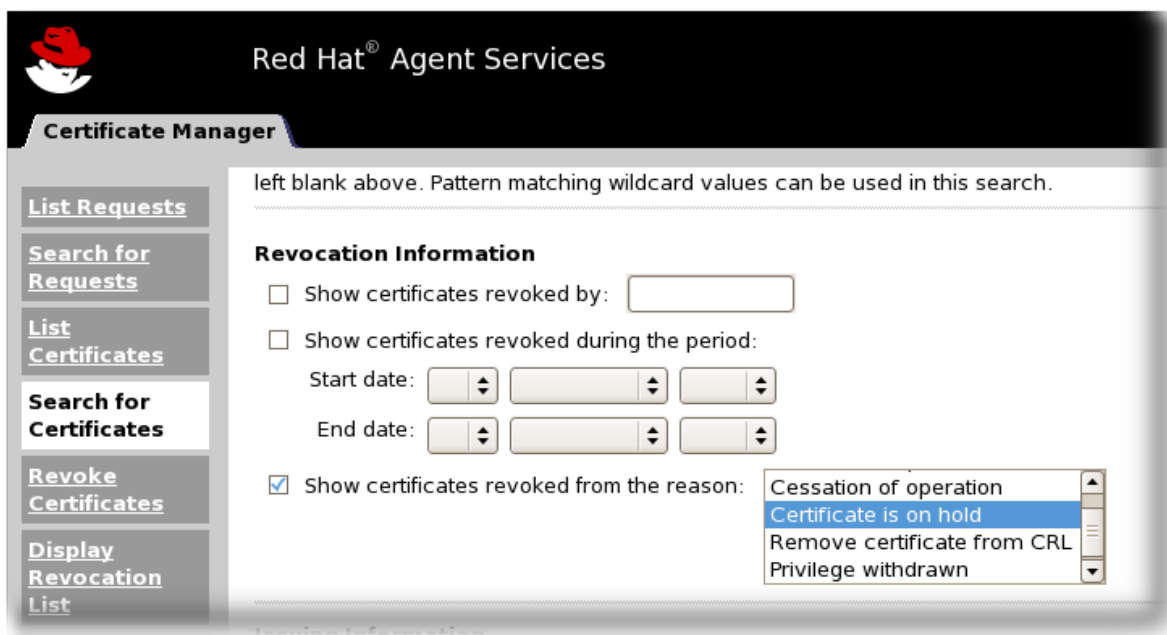
When the revocation request is submitted, it is automatically approved, and the certificate is revoked. Revocation requests are viewed by listing requests with a status of **Completed**.

6.2.2.4.2. Taking Certificates Off Hold

There can be instances when a certificate is inaccessible, and therefore should be treated as revoked, but that certificate can be recovered. For example, a user may have a personal email certificate stored on a flash drive which he accidentally leaves at home. The certificate is not compromised, but it should be temporarily suspended.

That certificate can be temporarily revoked by putting it on hold (one of the options given when revoking a certificate, as in [Section 6.2.2.4.1, "Revoking Certificates"](#)). At a later time – such as when the forgotten flash drive is picked up – that certificate can be taken off hold and is again active.

1. Search for the on hold certificate, as in [Section 6.2.2.2, "Searching for Certificates \(Advanced\)"](#). Scroll to the **Revocation Information** section, and set the **Certificate is on hold** revocation reason as the search criterion.



2. In the results list, click the **Off Hold** button by the certificate to take off hold.

The screenshot shows the Red Hat Agent Services Certificate Manager interface. The main content area displays search results for a certificate. The issuer is identified as 'CN=Certificate Authority,O=Example Domain' and one record was found. The certificate details are as follows:

Serial number	Subject name	
0x0000000f	UID=jsmith, E=jsmith@example.com, CN=John Smith, OU=engineering, OU=content services, OU=people, C=us	
Version	Certificate Type	Subject public key algorithm
3	X.509	PKCS #1 RSA with 512-bit key
Not valid before		Not valid after
4/29/2010 18:19:25		10/26/2010 18:19:25
Issued on		Issued by
4/29/2010 18:19:56		admin
Revoked on		Revoked by
4/29/2010 18:34:39		admin
Revocation Reason		
Certificate is on hold		

Buttons for 'Details' and 'Off Hold' are visible next to the certificate information.

6.2.2.5. Managing the Certificate Revocation List

Revoking a certificate notifies other users that the certificate is no longer valid. This notification is done by publishing a list of the revoked certificates, called the *certificate revocation list* (CRL), to an LDAP directory or to a flat file. This list is publicly available and ensures that revoked certificates are not misused.

6.2.2.5.1. Viewing or Examining CRLs

It may be necessary to view or examine a CRL, such as before manually updating a directory with the latest CRL. To view or display the CRL:

1. Go to the Certificate Manager agent services page.
2. Click **Display Certificate Revocation List** to display the form for viewing the CRL.
3. Select the CRL to view. If the administrator has created multiple issuing points, these are listed in the **Issuing point** drop-down list. Otherwise, only the master CRL is shown.
4. Choose how to display the CRL by selecting one of the options from the **Display Type** menu. The choices on this menu are as follows:
 - *Cached CRL*. Views the CRL from the cache rather than from the CRL itself. This option displays results faster than viewing the entire CRL.
 - *Entire CRL*. Retrieves and displays the entire CRL.
 - *CRL header*. Retrieves and displays the CRL header only.
 - *Base 64 Encoded*. Retrieves and displays the CRL in base-64 encoded format.

- *Delta CRL*. Retrieves and displays a delta CRL, which is a subset of the CRL showing only new revocations since the last CRL was published. This option is available only if delta CRL generation is enabled.

5. To examine the selected CRL, click **Display**.

The CRL appears in the browser window. This allows the agent to check whether a particular certificate (by its serial number) appears in the list and to note recent changes such as the total number of certificates revoked since the last update, the total number of certificates taken off hold since the last update, and the total number of certificates that expired since the last update.

6.2.2.5.2. Updating the CRL

CRLs can be automatically updated if a schedule for automatic CRL generation is enabled, and the schedule can set the CRL to be generated at set time schedules or whenever there are certificate revocations.

Likewise, CRLs can be also automatically published if CRL publishing is enabled.

In some cases, the CRL may need to be updated manually, such as updating the list after the system has been down or removing expired certificates to reduce the file size. (Expired certificates do not need to be included in the CRL because they are already invalid because of the expiration date.) Only a Certificate Manager agent can manually update the CRL.

To update the CRL manually:

1. Open the Certificate Manager agent services page.
2. Click **Update Revocation List** to display the form for updating the CRL.

Red Hat® Agent Services

Certificate Manager

Update Certificate Revocation List

In most cases, the certificate revocation list (CRL) is updated automatically. In a few situations, however, you may want to update the CRL manually. Use this form to update the CRL manually.

Issuing point:

Signature algorithm:

Wait for update:

Clear CRL cache:

Issuing point	CRL numbers	Number of entries	Recent changes
MasterCRL	<u>12</u>	0	0, 0, 0

Figure 6.3. Update Certificate Revocation List

3. Select the CRL issuing point which will update the CRL. There can be multiple issuing points configured for a single CA.
4. Select the algorithm to use to sign the new CRL. Before choosing an algorithm, make sure that any system or network applications that need to read or view this CRL support the algorithm.
 - SHA-256 with RSA.
 - SHA-384 with RSA.
 - SHA-512 with RSA.

Before selecting an algorithm, make sure that the Certificate System has that algorithm enabled. The Certificate System administrator will have that information.

5. Click **Update** to update the CRL with the latest certificate revocation information.

6.2.3. Performing Revocation on Own Certificate as a User Using the Web UI

Revoking a certificate invalidates it before its expiration date. This can be necessary if a certificate is lost, compromised, or no longer needed.

6.2.3.1. Revoking Your User Certificate

1. Click the **Revocation** tab.
2. Click the **User Certificate** link.
3. Select the reason why the certificate is being revoked, and click **Submit**.



The screenshot shows the Red Hat Certificate Manager web interface. The top navigation bar is red with the Red Hat logo and the text "Red Hat® Certificate Manager". Below the navigation bar are three tabs: "Enrollment", "Revocation", and "Retrieval". The "Revocation" tab is selected. On the left side, there is a sidebar with "User Certificate" and "CMC Revoke" links. The main content area is titled "User Certificate Revocation" and contains the following text:

Use this form to revoke your certificate automatically.

After you click the submit button, a window will pop up with a list of certificates you can send to the server. Select the certificate you want to revoke from this window.

Important: This is an irreversible operation. If you still want to continue, be sure to request revocation on the computer where the private key and certificate to be revoked are stored.

Revocation Reason
Select a revocation reason

- Unspecified
- Key Compromise
- Cessation of Operation
- Affiliation Changed
- Superseded

At the bottom right of the form, there are two buttons: "Submit" and "Reset".

4. Select the certificates to revoke from the list.

6.2.3.2. Checking Whether a Certificate Is Revoked

1. Click the **Retrieval** tab.
2. Click the **Import Certificate Revocation List** link.
3. Select the radio button by **Check whether the following certificate is included in CRL cache** or **Check whether the following certificate is listed by CRL**, and enter the serial number of the certificate.

The screenshot shows the Red Hat Certificate Manager web interface. The top navigation bar is red with the Red Hat logo and the text "Red Hat® Certificate Manager". Below the navigation bar are three tabs: "Enrollment", "Revocation", and "Retrieval". The "Retrieval" tab is selected. On the left side, there is a sidebar with several links: "Check Request Status", "List Certificates", "Search Certificates", "Import CA Certificate Chain", "Import Certificate Revocation List", and "Import Certificate Revocation List". The main content area is titled "Import Certificate Revocation List" and contains the following text: "Use this form to check whether a particular certificate has been revoked or to import the latest Certificate Revocation List." Below this text, there is a section titled "Select CRL issuing point" with a dropdown menu showing "MasterCRL". Another section titled "Select one of these actions" contains several radio buttons: "Check whether the following certificate is included in CRL cache", "Check whether the following certificate is listed by CRL" (which is selected), "Import the latest CRL to your browser", "Import the latest delta CRL to your browser", "Download the latest CRL in binary form", "Download the latest delta CRL in binary form", and "Display the CRL information:" with a dropdown menu showing "Cached CRL". A text input field for "Certificate serial number" contains the value "15". At the bottom right of the form is a "Submit" button.

4. Click the **Submit** button.

A message is returned either saying that the certificate is not listed in any CRL or giving the information for the CRL which contains the certificate.

6.2.3.3. Downloading and Importing CRLs

Certificate revocation lists (CRLs) can be downloaded and installed in a web client, application, or machine. They can also be viewed to see what certificates have been revoked.

1. Click the **Retrieval** tab.
2. Click the **Import Certificate Revocation List** link.
3. Select the radio button to view, download, or import the CRL.

Red Hat® Certificate Manager

Enrollment | **Revocation** | Retrieval

[Check Request Status](#)
[List Certificates](#)
[Search Certificates](#)
[Import CA Certificate Chain](#)
[Import Certificate Revocation List](#)

Import Certificate Revocation List

Use this form to check whether a particular certificate has been revoked or to import the latest Certificate Revocation List.

Select CRL issuing point
 Issuing point:

Select one of these actions

Check whether the following certificate is included in CRL cache

Check whether the following certificate is listed by CRL
 Certificate serial number:

Import the latest CRL to your browser

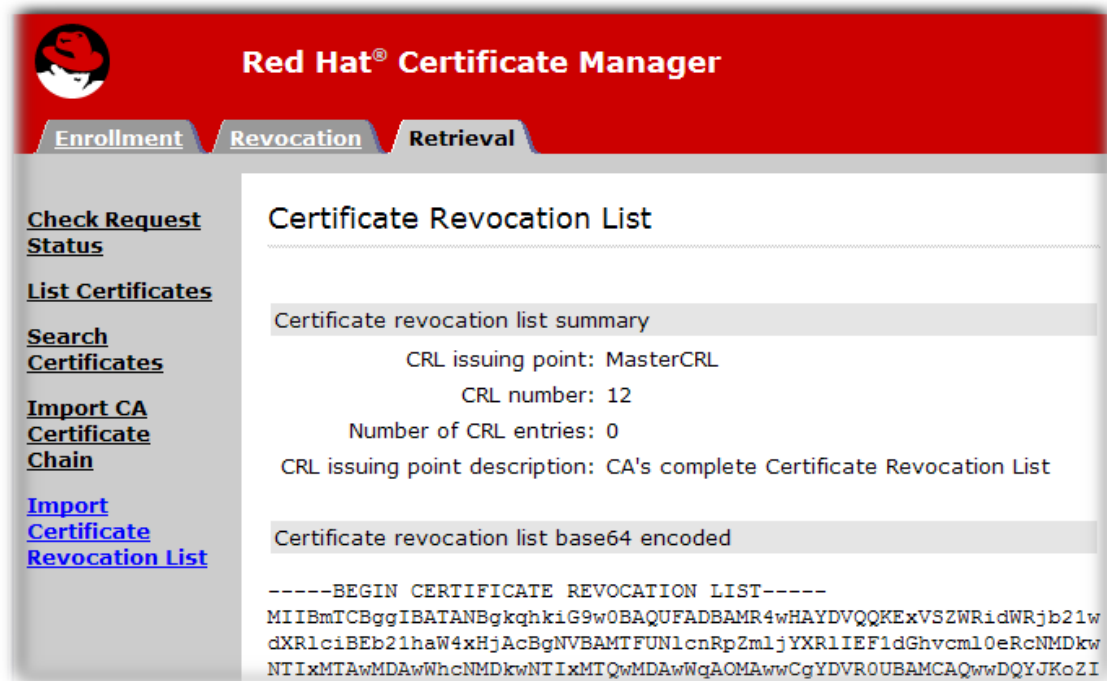
Import the latest delta CRL to your browser

Download the latest CRL in binary form

Download the latest delta CRL in binary form

Display the CRL information:

- To import the CRL into the browser or download and save it, select the appropriate radio button. There are two options: to download/import the full CRL or the delta CRL. The delta CRL only imports/downloads the list of certificates which have been revoked since the last time the CRL was generated.
- To view the CRL, select **Display the CRL information** and select which CRL subset (called an *issuing point*) to view. This shows the CRL information, including the number of certificates included in it.



4. Click the **Submit** button.
5. Save the file or approve the import operation.

6.3. ISSUING CRLS

1. The Certificate Manager uses its OCSP signing 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 9.2.3.11. Setting a CA to Use a Different Certificate to Sign CRLs in Red Hat Certificate System's Planning, Installation, and Deployment Guide for more information.
2. Set up CRL issuing points. An issuing point is already set up and enabled for a master CRL.

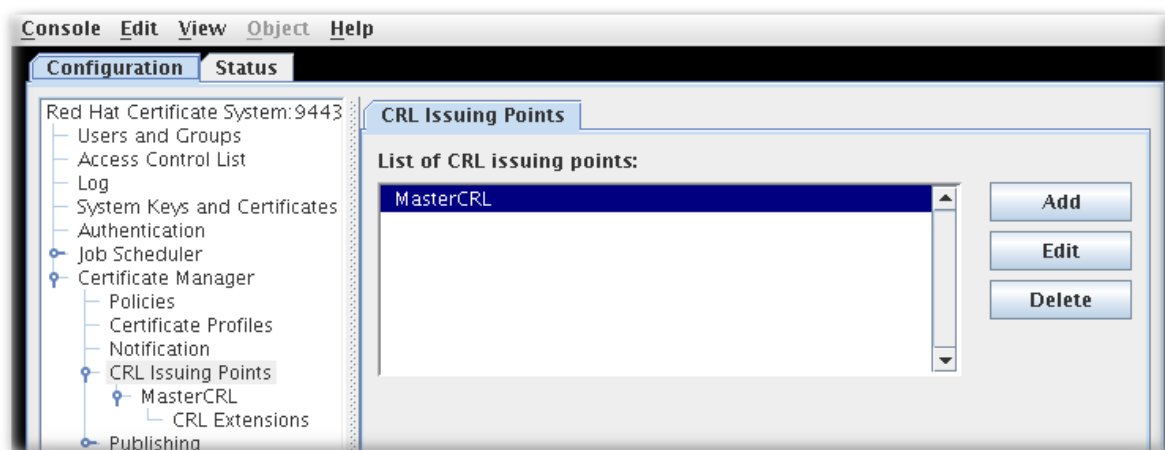


Figure 6.4. Default CRL Issuing Point

Additional issuing points for the CRLs can be created. See [Section 6.3.1, "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 6.3.2, “Configuring CRLs for Each Issuing Point”](#) for details.
 4. Set up the CRL extensions which are configured for the issuing point. See [Section 6.3.3, “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 7, *Publishing Certificates and CRLs*](#) for details about setting up publishing.

6.3.1. 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.

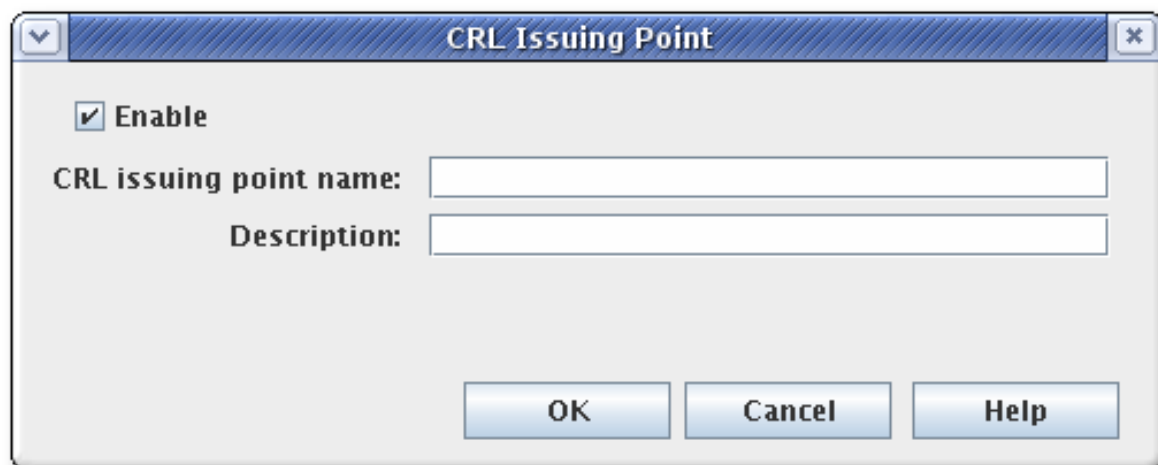


Figure 6.5. CRL Issuing Point Editor



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 6.3.2, "Configuring CRLs for Each Issuing Point"](#) for details on configuring an issuing point. See [Section 6.3.3, "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.

6.3.2. Configuring CRLs for Each Issuing Point

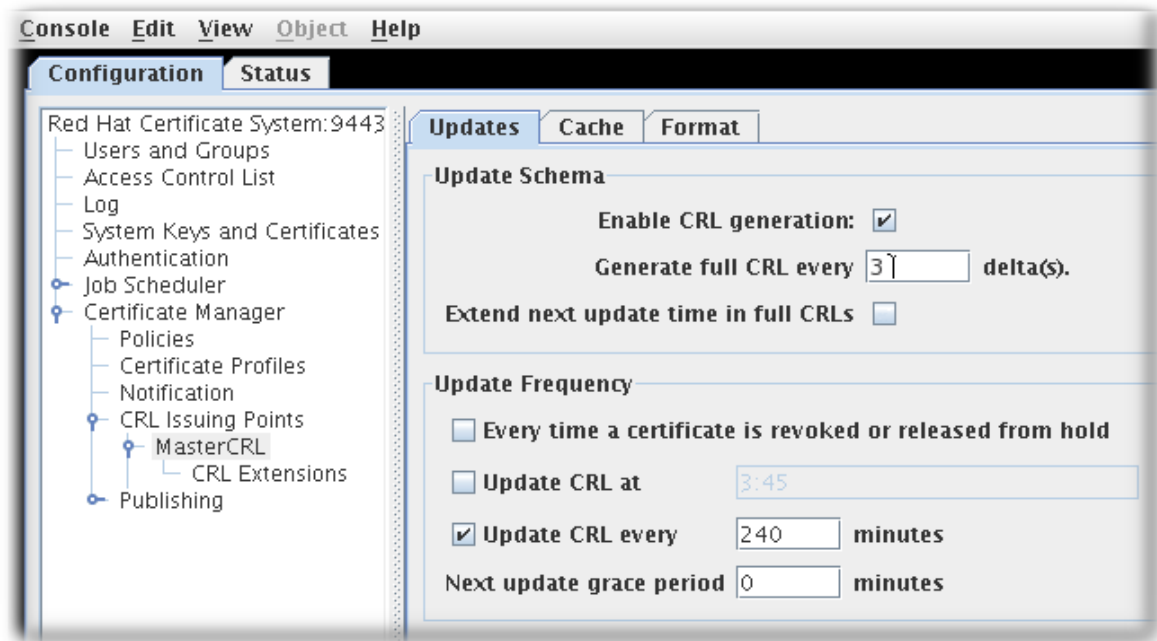
Information, such as the generation interval, the CRL version, CRL extensions, and the signing algorithm, can all be configured for the CRLs for the issuing point. The CRLs must be configured for each issuing point.

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-separated 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.

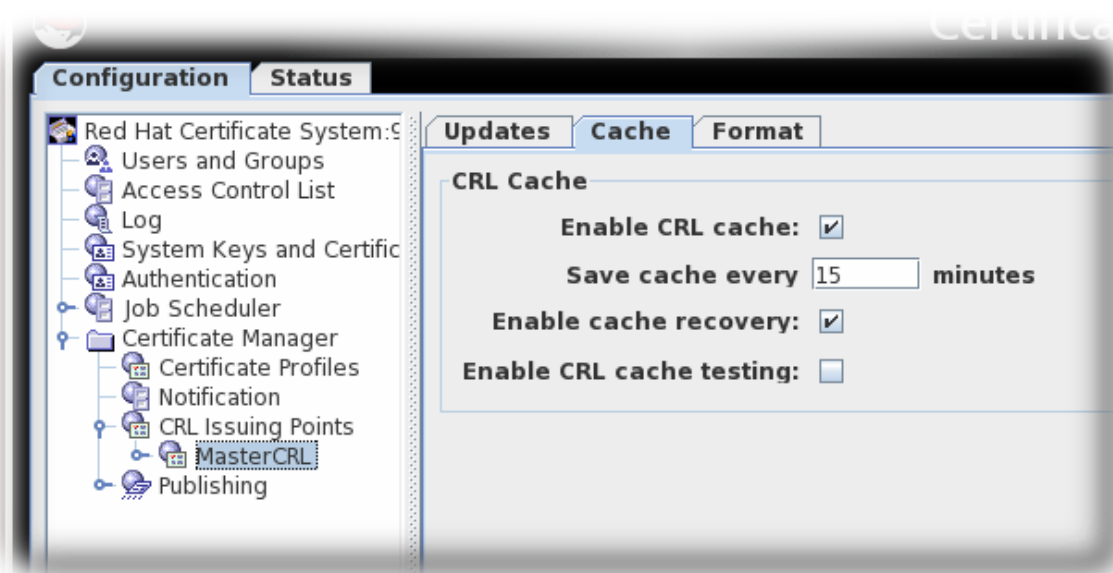


Figure 6.6. CRL Cache Tab

- **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 6.1, "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**.

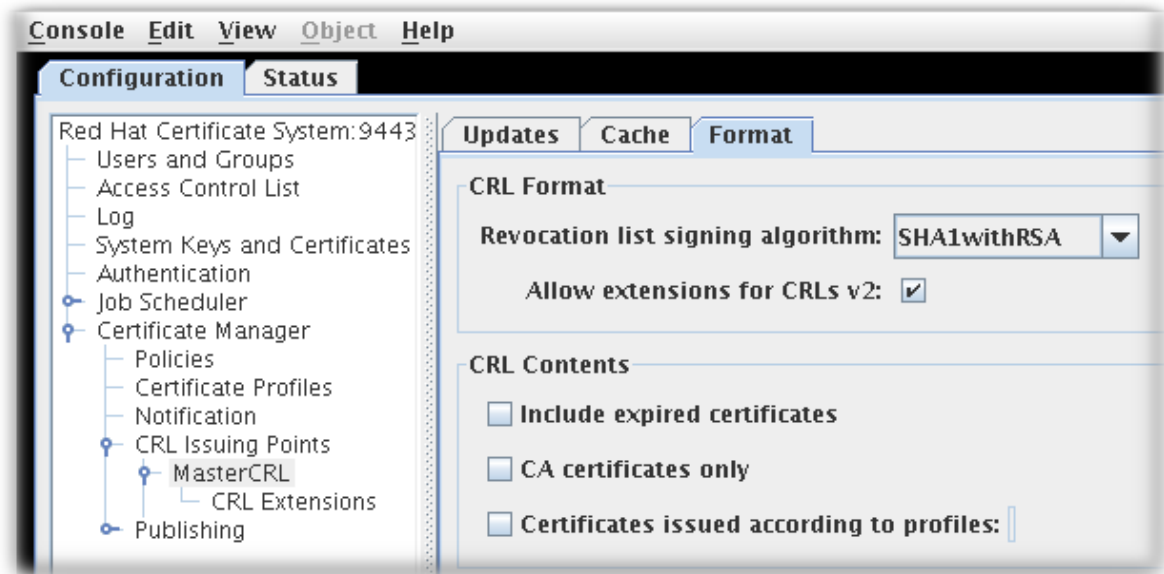


Figure 6.7. CRL Format Tab

- 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 6.3.3, “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.

for those extensions.

6. Click **OK**.
7. Click **Refresh** to see the updated status of all the rules.

6.3.4. 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

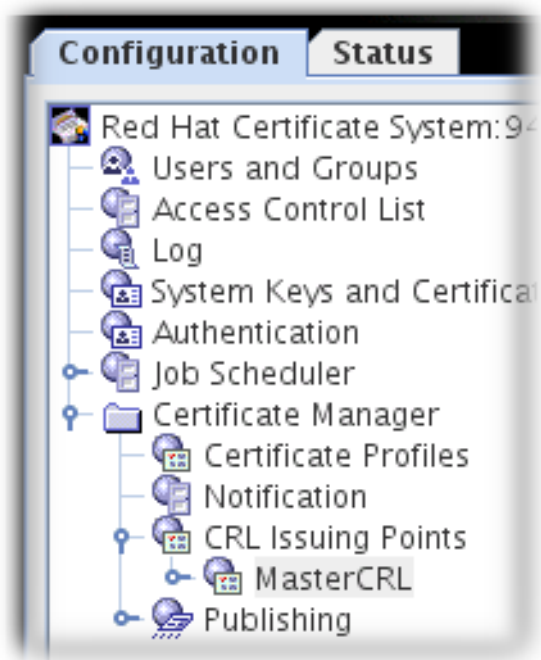
Because of the performance enhancement from generating CRLs from cache, enable the ***enableCRLCache*** parameter in most environments. However, the ***Enable CRL cache testing*** parameter *should not* be enabled in a production environment.

6.3.4.1. Configuring CRL Generation from Cache in the Console

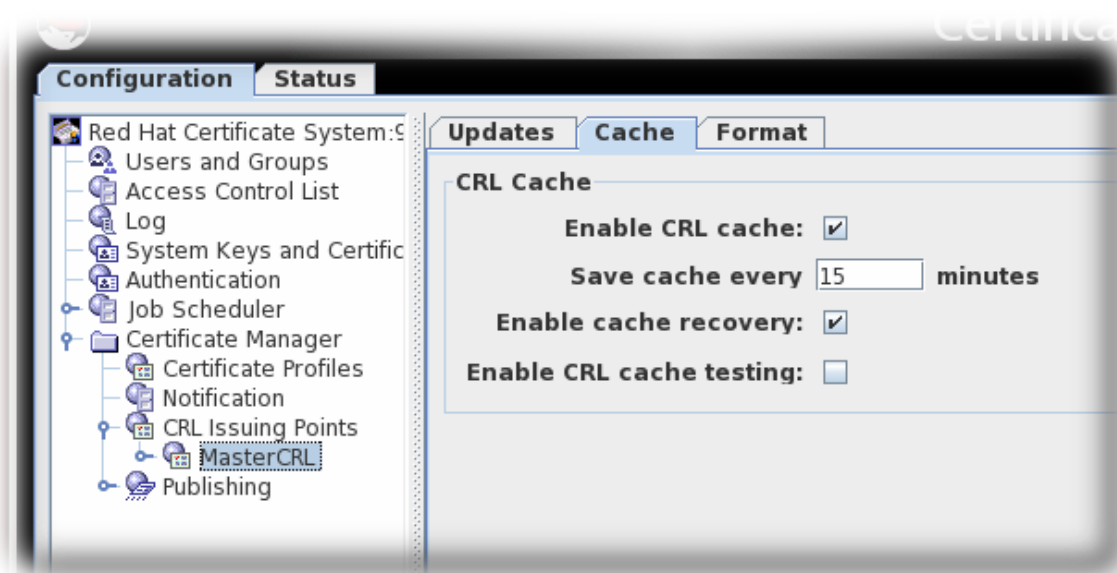
1. Open the console.

```
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.

6.4. SETTING FULL AND DELTA CRL SCHEDULES

CRLs are generated periodically. Setting that period is touched on in the configuration in [Section 6.3.2, "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

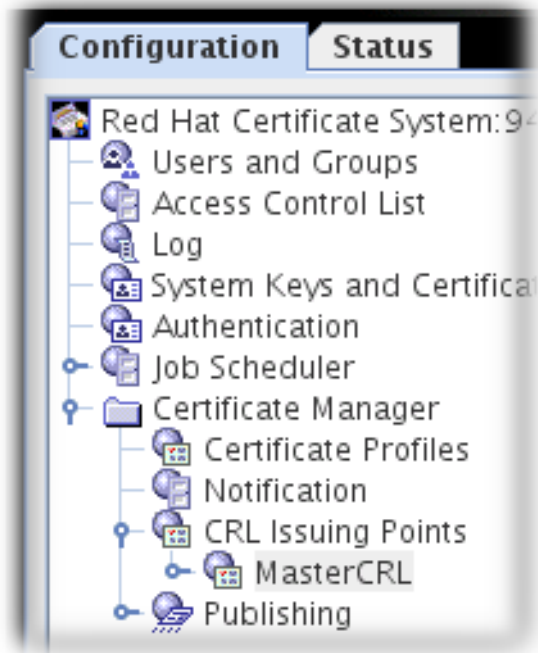
For delta CRLs to be generated in addition to full CRLs, the CRL cache must be enabled.

6.4.1. Configuring CRL Update Intervals in the Console

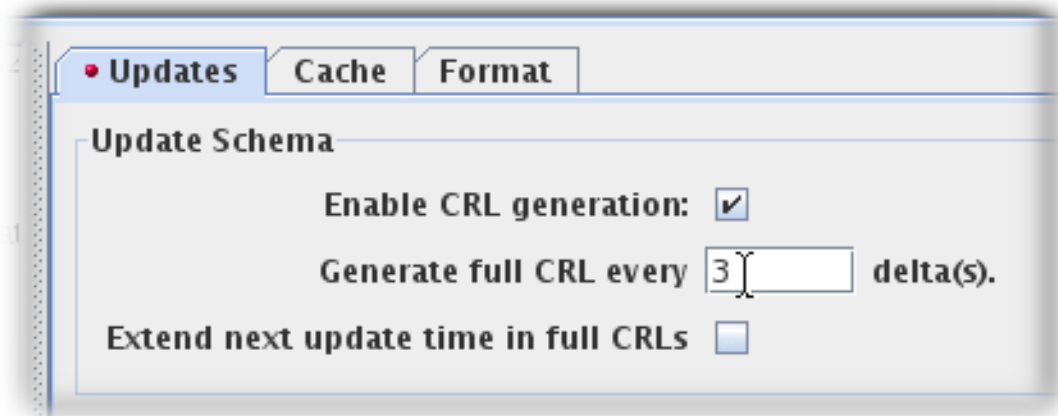
1. Open the console.

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

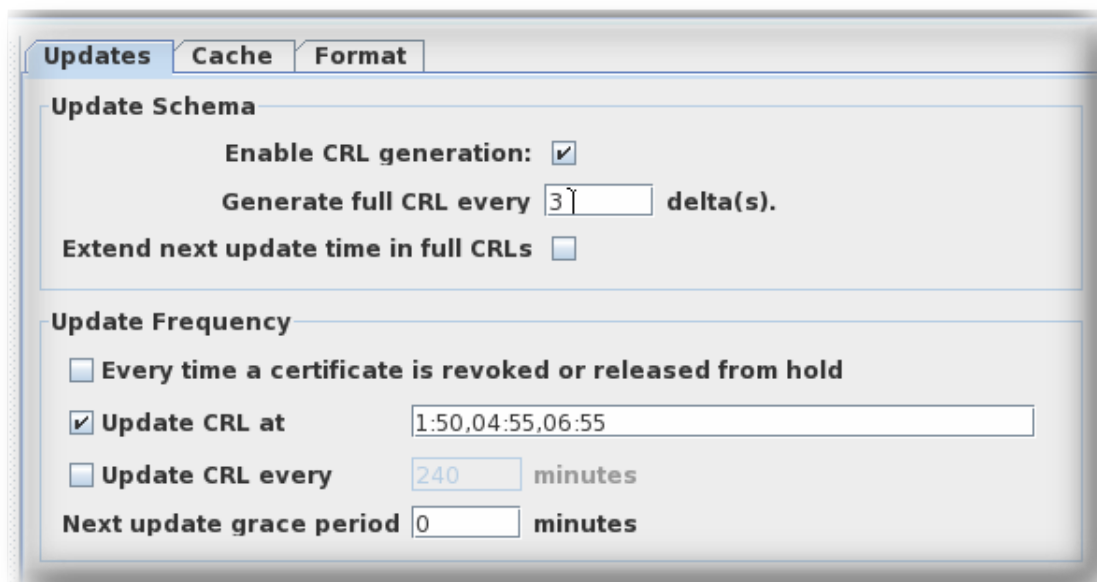
2. In the **Configuration** tab, expand the **Certificate Manager** folder and the **CRL Issuing Points** subfolder.
3. Select the **MasterCRL** node.



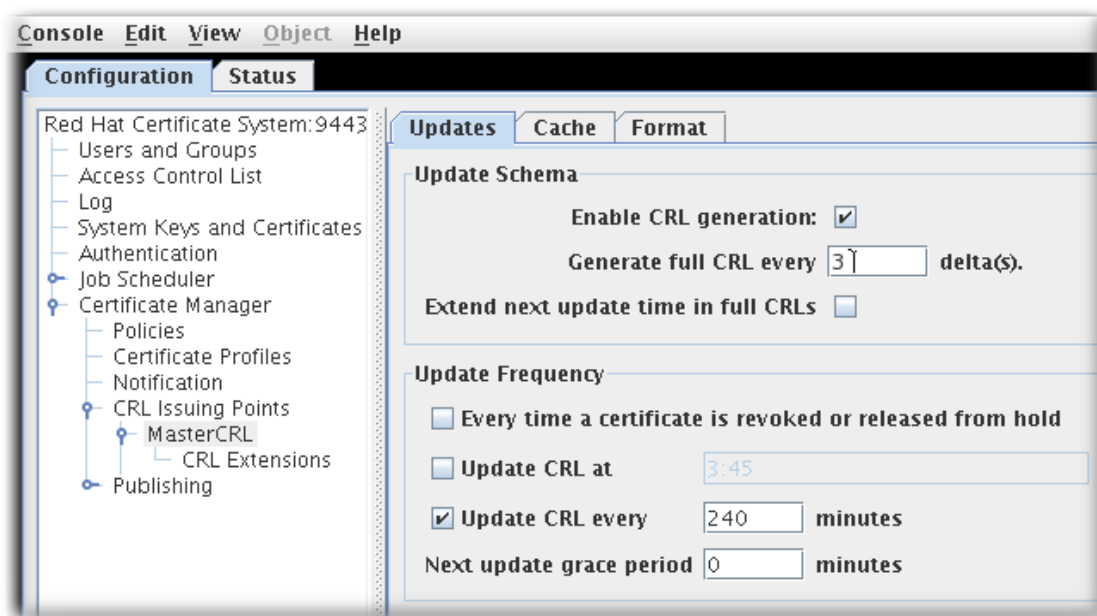
4. Enter the required interval in the **Generate full CRL every # delta(s)** field.



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.

The screenshot shows a configuration window with three tabs: 'Updates', 'Cache', and 'Format'. The 'Updates' tab is active. It contains two main sections: 'Update Schema' and 'Update Frequency'. In the 'Update Schema' section, 'Enable CRL generation' is checked, 'Generate full CRL every' is set to 3 delta(s), and 'Extend next update time in full CRLs' is unchecked. In the 'Update Frequency' section, 'Every time a certificate is revoked or released from hold' is unchecked, 'Update CRL at' is checked with a value of 1:00, 'Update CRL every' is checked with a value of 240 minutes, and 'Next update grace period' is set to 0 minutes.

Only one **Update CRL at** value will be accepted when updating CRLs by interval.

6.4.2. 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.

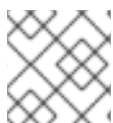
NOTE

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**

Both the semicolon and asterisk syntax works in the console.

6.5. USING THE ONLINE CERTIFICATE STATUS PROTOCOL (OCSP) RESPONDER

6.5.1. 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 7, 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 6.5.4, "Enabling the Certificate Manager's Internal OCSP Service"](#)).
4. Configure the OCSP Responder.
 - Configure the Revocation Info store ([Section 6.5.2.2, "Configure the Revocation Info Stores: Internal Database"](#) and [Section 6.5.2.3, "Configure the Revocation Info Stores: LDAP Directory"](#)).
 - Identify every publishing Certificate Manager to the OCSP responder ([Section 6.5.2, "Identifying the CA to the OCSP Responder"](#)).
 - If necessary, configure the trust settings for the CA which signed the OCSP signing certificate ([Section 14.5, "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 6.5.2.1, "Verify Certificate Manager and Online Certificate Status Manager Connection"](#)).

6.5.2. 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 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).

6.5.2.1. 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 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.

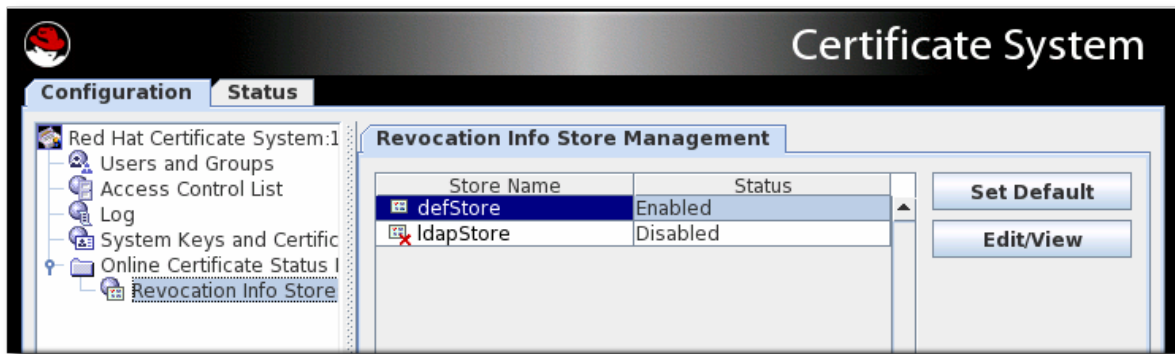
6.5.2.2. Configure the Revocation Info Stores: Internal Database

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.

6.5.2.3. 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

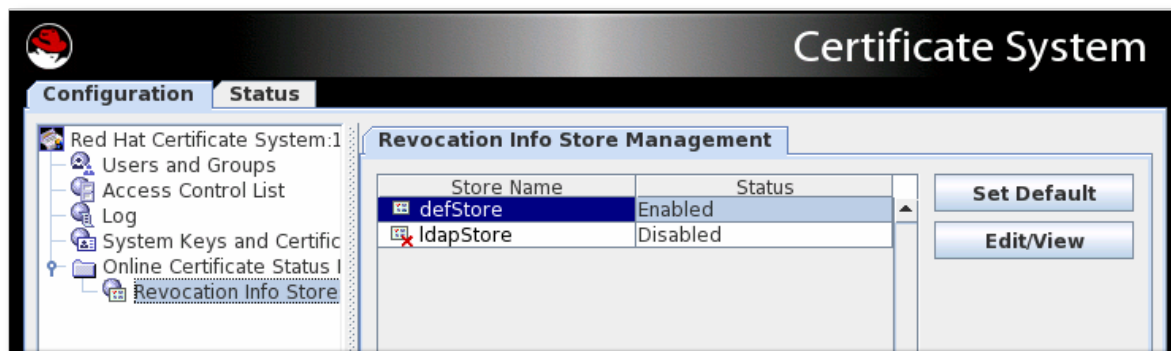
If the **ldapStore** method is enabled, the OCSP user interface does not check the certificate status.

To configure the Online Certificate Status Manager to use an LDAP directory:

1. Open the Online Certificate Status Manager Console.

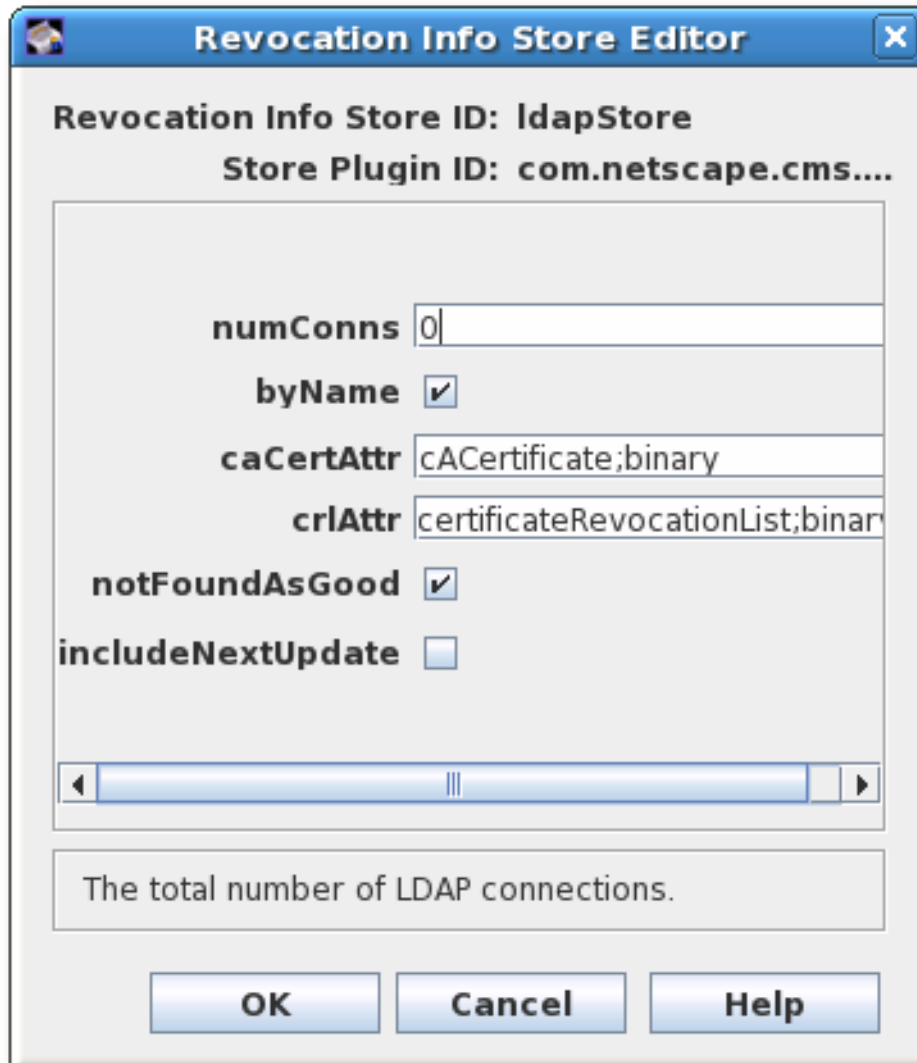
```
pkiconsole https://server.example.com:8443/ocsp
```

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-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.

6.5.2.4. 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.

6.5.3. 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.

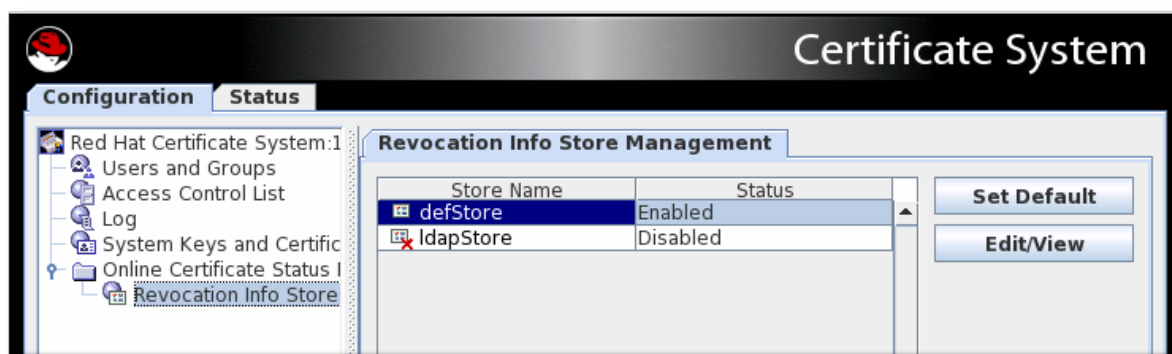
The ***notFoundAsGood*** parameter sets how the OCSP handles a certificate with an invalid serial number. This parameter is enabled by default, which means that if a certificate is present with a bad serial number but the certificate is otherwise valid, the OCSP returns a status of **GOOD** for the certificate.

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 interpret as an error.



- Restart the OCSP Manager.

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

6.5.4. 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-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.

- 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.

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

6.5.5. 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 6.1. Available OCSPClient Options

Option	Description
<code>-d database</code>	Security database location (default: current directory)
<code>-h hostname</code>	OCSP server hostname (default: example.com)
<code>-p port</code>	OCSP server port number (default: 8080)
<code>-t path</code>	OCSP service path (default: /ocsp/ee/ocsp)
<code>-c nickname</code>	CA certificate nickname (default: CA Signing Certificate)
<code>-n times</code>	Number of submissions (default: 1)
<code>--serial serial_number</code>	Serial number of certificate to be checked
<code>--input input_file</code>	Input file containing DER-encoded OCSP request
<code>--output output_file</code>	Output file to store DER-encoded OCSP response
<code>-v, --verbose</code>	Run in verbose mode

Option	Description
--help	Show help message

6.5.6. 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+MDwwOjAJBgUrDgMCGGUABBT4cyABkyiClhU4JpmlBewdDnn8ZgQUbyBZ44kgy
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
cyABkyiClhU4JpmlBewdDnn8ZgQUbyBZ44kgy35o7xW5BMzM8FTvyTwCAQE=
```

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+MDwwOjAJBgUrDgMCGGUABBT4cyABkyiClhU4JpmlBewdDnn8ZgQUbyBZ44kgy
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
cyABkyiClhU4JpmlBewdDnn8ZgQUbyBZ44kgy35o7xW5BMzM8FTvyTwCAQE= >
ocsresp.der
```

3. Parse the response using **openssl**:

```
openssl ocsp -respin ocsresp.der -resp_text
```

PART III. ADDITIONAL CONFIGURATION TO MANAGE CA SERVICES

CHAPTER 7. PUBLISHING CERTIFICATES AND CRLS



NOTE

Features in this section on TMS are not tested in the evaluation. This section is for reference only.

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 6, *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.

7.1. 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.

Setting up publishing involves configuring publishers, mappers, and rules.

7.1.1. 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.

7.1.2. 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.

7.1.3. 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.

7.1.4. 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 2021**, is **MasterCRL-20210128-153600.der**.

7.1.5. 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 6.5, "Using the Online Certificate Status Protocol \(OCSP\) Responder"](#).

7.1.6. 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 TLS (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 7.11.2, “Manually Updating the CRL in the Directory”](#).

7.2. 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.

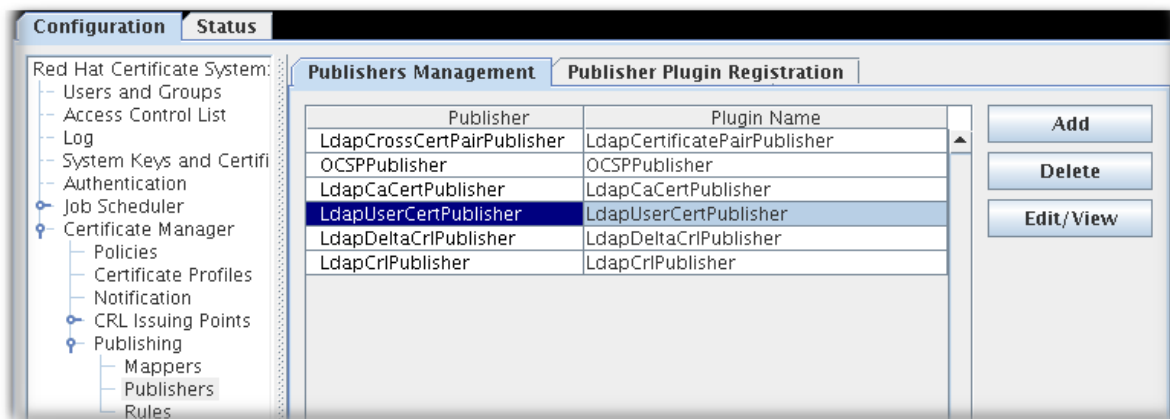
To create publishers for publishing to files:

1. Log into the Certificate Manager Console.

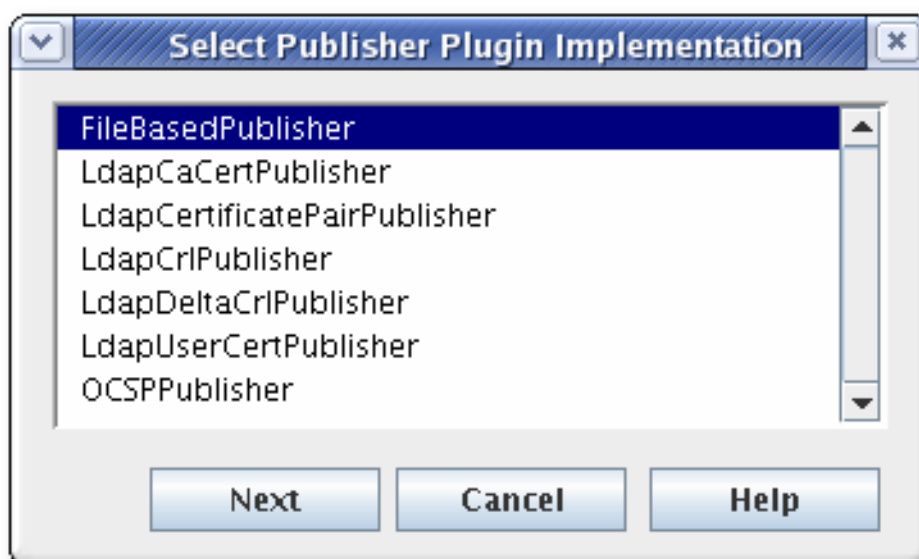
 pkiconsole https://server.example.com:8443/ca

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.

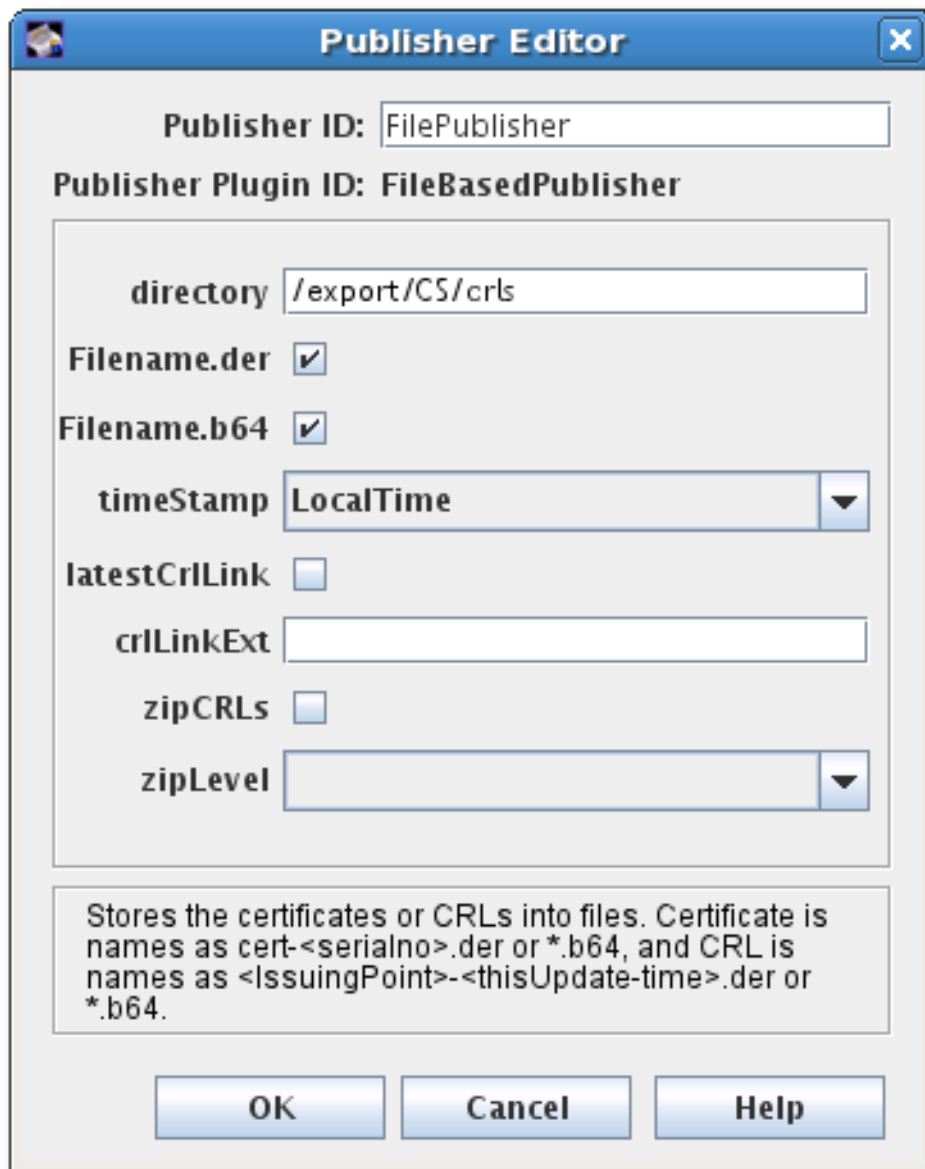


3. Click **Add** to open the **Select Publisher Plug-in Implementation** window, which lists registered publisher modules.



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 7.5, "Creating Rules"](#).

7.3. 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.

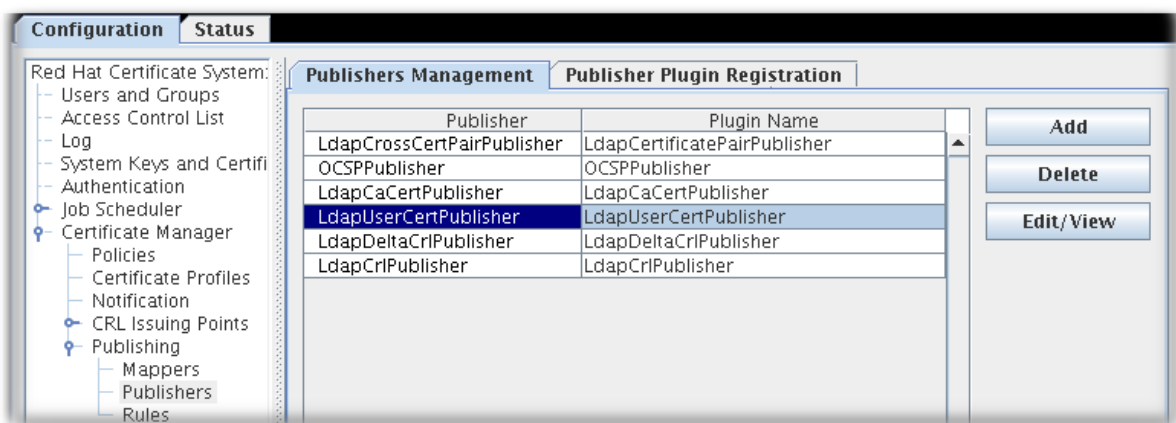
A publisher must be created and configured for each publishing location; publishers are not automatically created for publishing to the OCSP responder. Create a single publisher to publish everything to a single location, or create a publisher for every location to which CRLs will be published. Each location can contain a different kind of CRL.

7.3.1. Enabling Publishing to an OCSP with Client Authentication

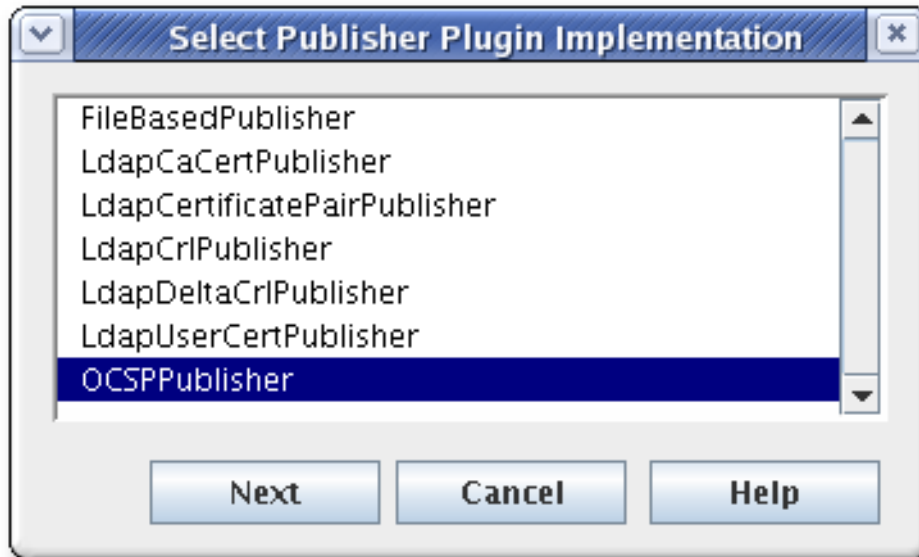
1. Log into the Certificate Manager Console.

`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 12.3.2.1, "Creating Users"](#).

After configuring the publisher, configure the rules for the published certificates and CRLs, as described in [Section 7.5, "Creating Rules"](#).

7.4. 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 7.4.3, "Creating Mappers"](#).

4. Create rules to connect publishers to mappers, as described in [Section 7.5, "Creating Rules"](#).
5. Enable publishing, as described in [Section 7.6, "Enabling Publishing"](#).

7.4.1. Configuring the LDAP Directory

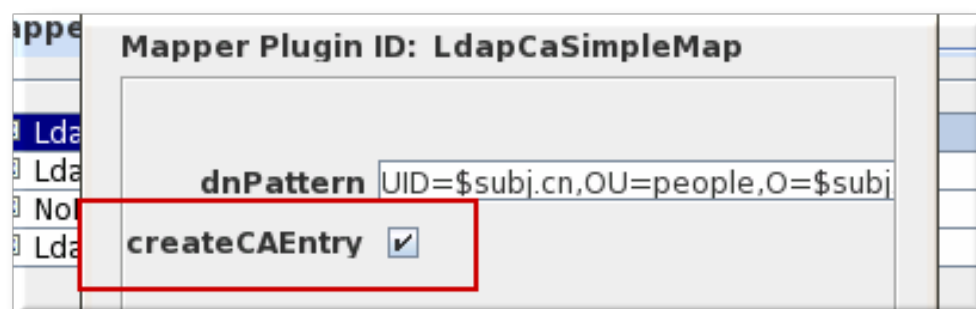
Before certificates and CRLs can be published, the Directory Server must be configured to work with the publishing system. This means that user entries must have attributes that allow them to receive certificate information, and entries must be created to represent the CRLs.

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)	<p>This is the attribute to which the Certificate Manager publishes the certificate.</p> <p>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.</p> <p>If the directory object that it finds does not allow the userCertificate;binary attribute, adding or removing the certificate fails.</p>
CA certificate	caCertificate;binary (attribute)	<p>This is the attribute to which the Certificate Manager publishes the certificate.</p> <p>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.</p> <p>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.</p>

Object Type	Schema	Reason
CRL	certificateRevocationList;binary (attribute)	<p>This is the attribute to which the Certificate Manager publishes the CRL.</p> <p>The Certificate Manager publishes the CRL to its own LDAP directory entry. The entry corresponds to the Certificate Manager's issuer name.</p> <p>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.</p>
Delta CRL	deltaRevocationList;binary (attribute)	<p>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.</p> <p>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.</p>

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); TLS without client authentication (simple username and password); and TLS with client authentication (certificate-based).

See the Red Hat Directory Server documentation for instructions on setting up these methods of communication with the server.

7.4.2. 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 7.1. 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.

7.4.3. 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 7.2, "Default Mappers"](#).

Table 7.2. 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.

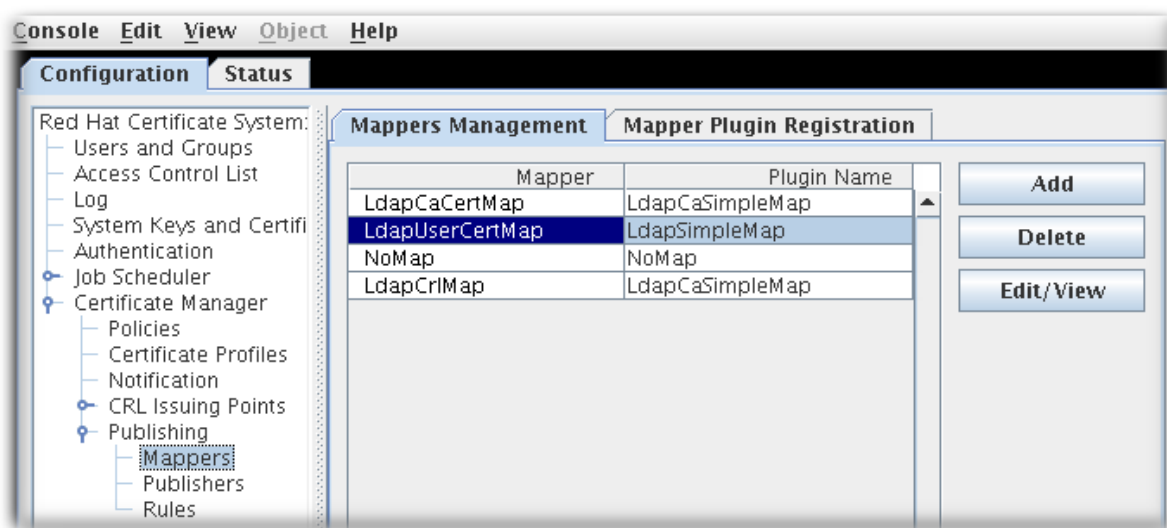
To use the default mappers, configure each of the macros by specifying the DN pattern and whether to create the CA entry in the directory. To use other mappers, create and configure an instance of the mapper. For more information, see [Section C.2, "Mapper Plug-in Modules"](#).

1. Log into the Certificate Manager Console.

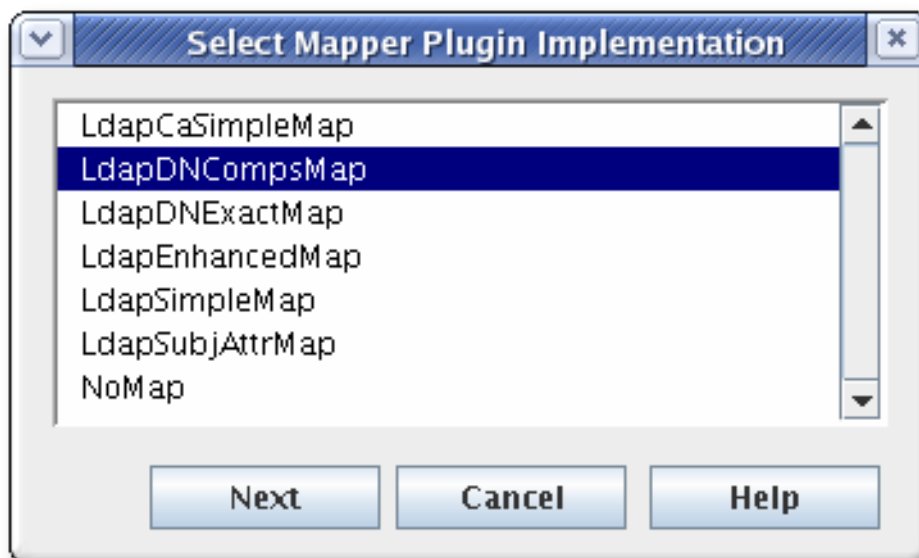
pkiconsole https://server.example.com:8443/ca

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Mappers**.

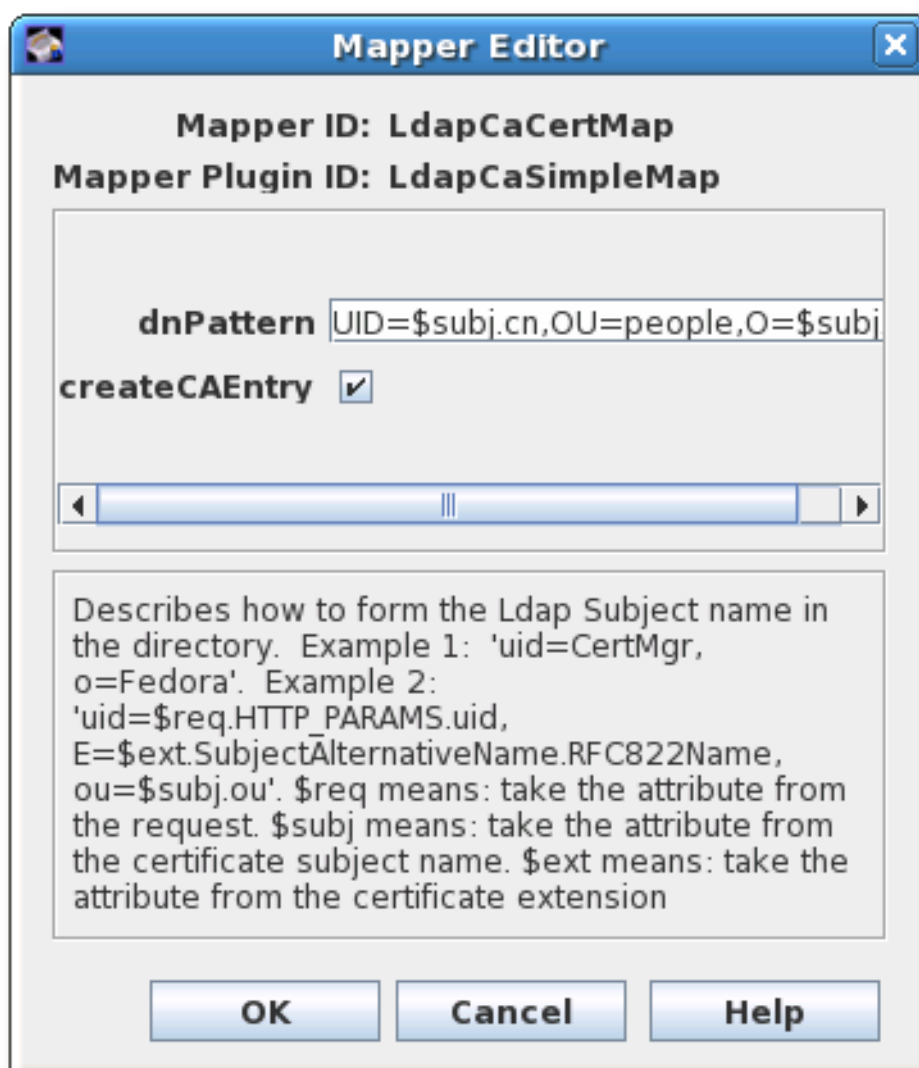
The **Mappers Management** tab, which lists configured mappers, opens on the right.



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.

7.4.4. 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 7.5, “Creating Rules”](#).

Once the configuration is complete, enable publishing, as described in [Section 7.6, “Enabling Publishing”](#).

7.5. 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.

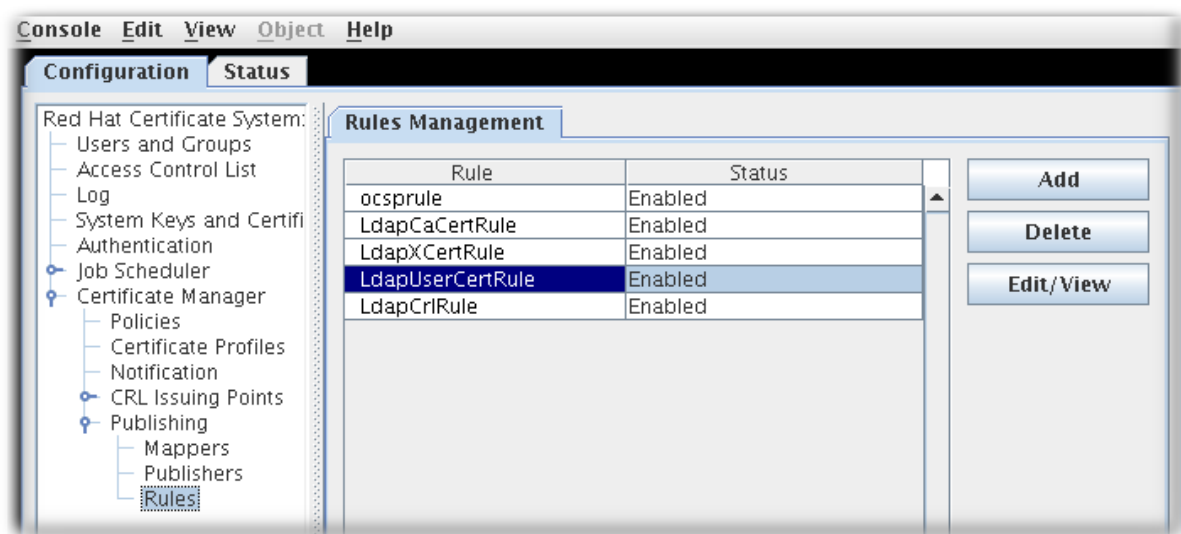
Modify publishing rules by doing the following:

1. Log into the Certificate Manager Console.

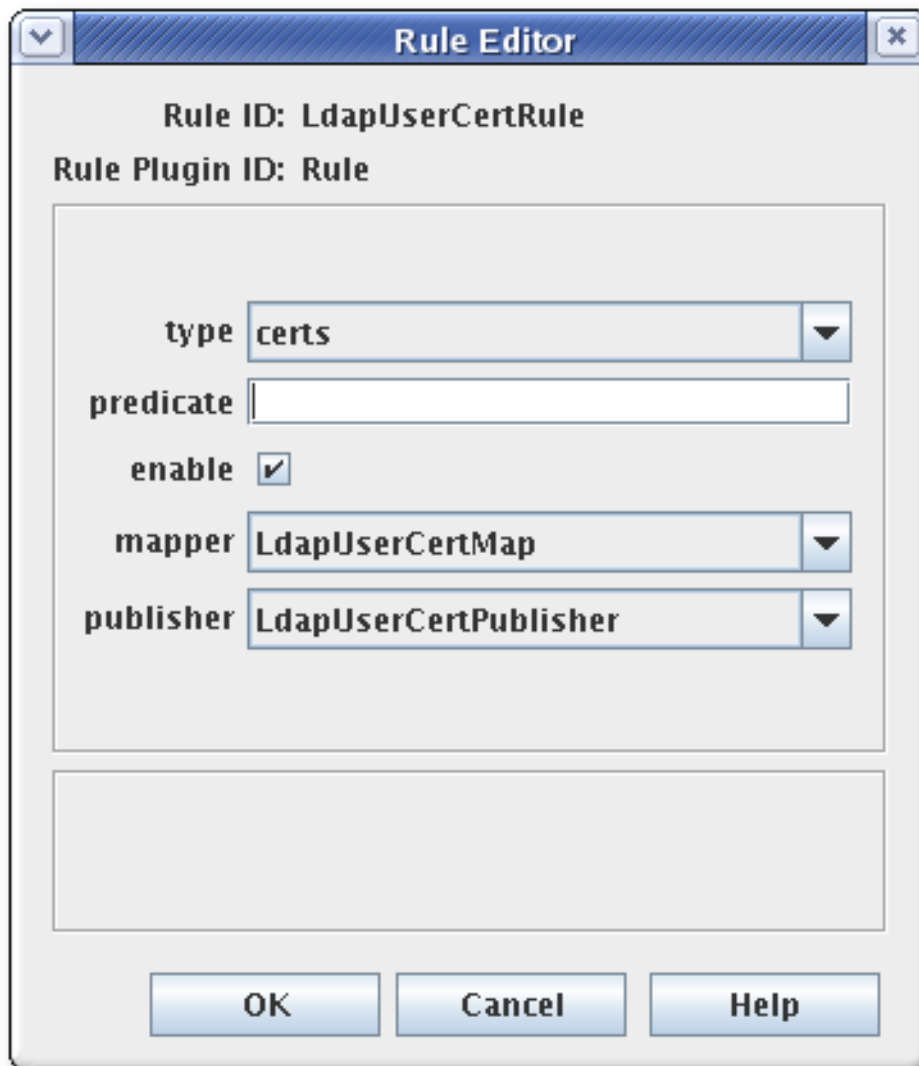
`pkiconsole https://server.example.com:8443/ca`

2. In the **Configuration** tab, select **Certificate Manager** from the navigation tree on the left. Select **Publishing**, and then **Rules**.

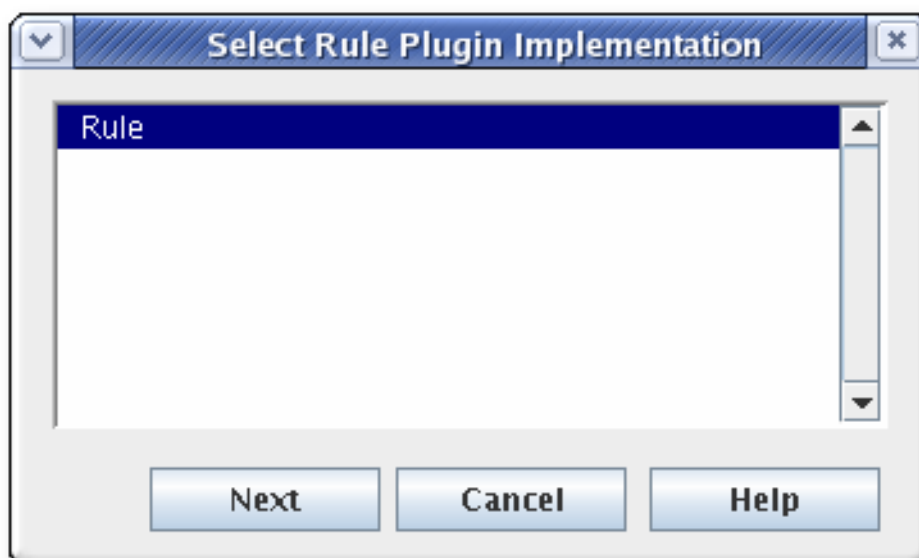
The **Rules Management** tab, which lists configured rules, opens on the right.



3. To edit an existing rule, select that rule from the list, and click **Edit**. This opens the **Rule Editor** window.

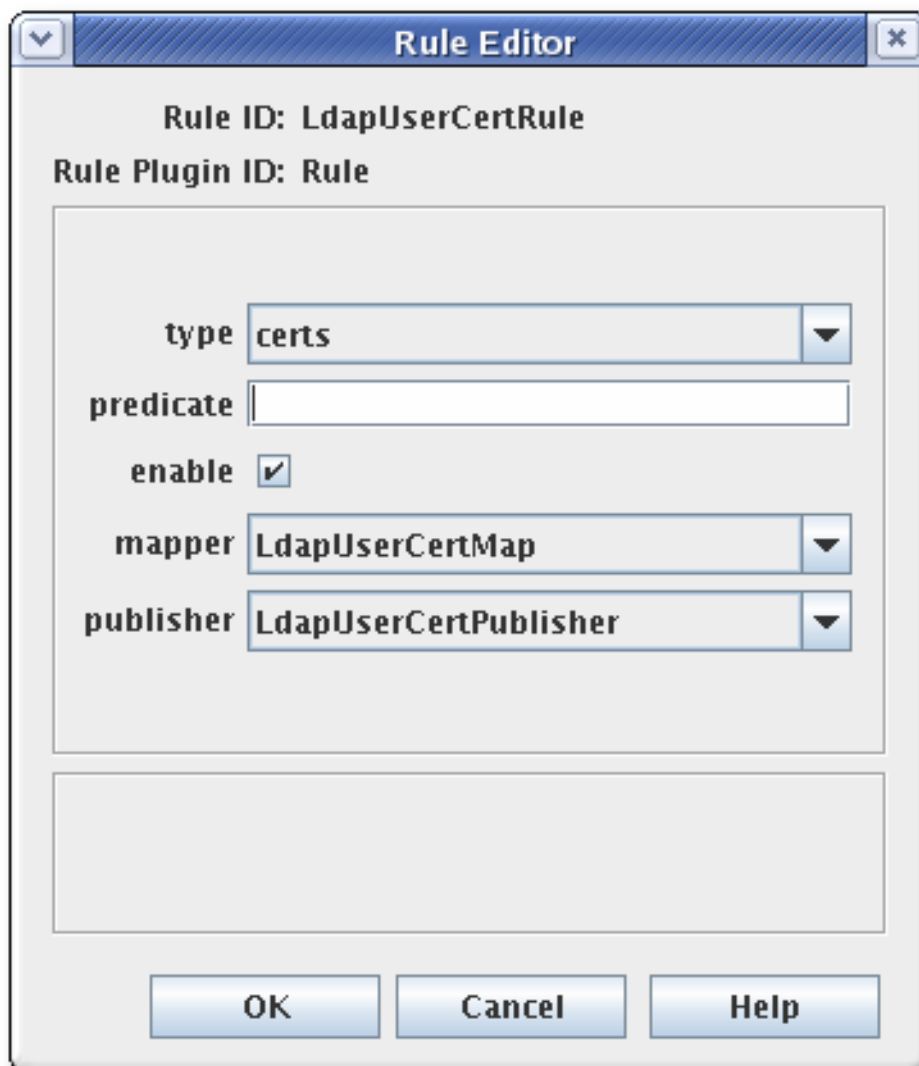


4. To create a rule, click **Add**. This opens the **Select Rule Plug-in Implementation** window.



Select the **Rule** module. This is the only default module. If any custom modules have been 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 **ca-cert**. 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 7.3, "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 7.3, "Predicate Expressions"](#) lists the predicates that can be used to identify CRL issuing points and delta CRLs and certificate profiles.

Table 7.3. Predicate Expressions

Predicate Type	Predicate
CRL Issuing Point	<p>issuingPointId==<i>Issuing_Point_Instance_ID</i> && isDeltaCRL==[true false]</p> <p>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.</p>
Certificate Profile	<p>profileId==<i>profile_name</i></p> <p>To publish certificates based on the profile used to issue them, set profileId== to a profile name, such as caServerCert.</p>

7.6. 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 6, Revoking Certificates and Issuing CRLs](#).

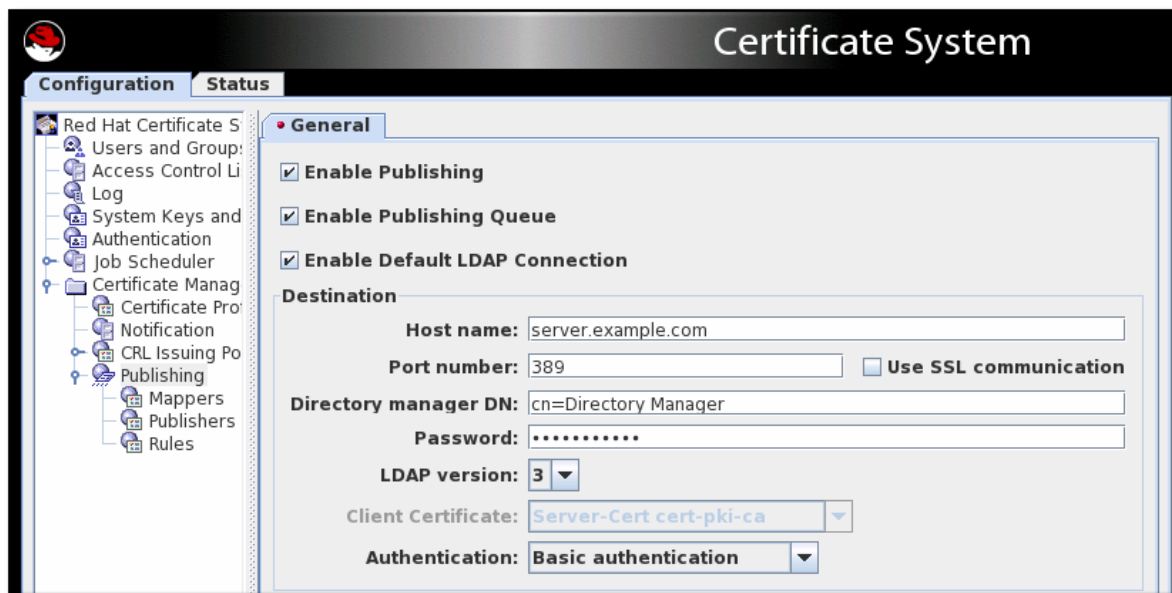
1. Log into the Certificate Manager Console.

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 TLS client authenticated communication, the name must match the **cn** component in the subject DN of the Directory Server's TLS 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
```

- **Client certificate.** This sets the certificate the Certificate Manager uses for TLS client authentication to the publishing directory. By default, the Certificate Manager uses its TLS 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 **TLS client authentication**.

If the Directory Server is configured for basic authentication or for TLS communication without client authentication, select **Basic authentication** and specify values for the Directory manager DN and password.

If the Directory Server is configured for TLS communication with client authentication, select **TLS client authentication** and the **Use TLS communication** option, and identify

the certificate that the Certificate Manager must use for TLS 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.

7.7. 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.

7.7.1. 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 7.4, "wget Options to Use for Retrieving CRLs"](#).

Table 7.4. wget Options to Use for Retrieving CRLs

Argument	Description
<i>no argument</i>	Retrieves the full CRL.
-N	Retrieves the CRL that is newer than the local copy (delta CRL).
-c	Retrieves a partially-downloaded file.
--no-check-certificate	Skips TLS for the connection, so it is not necessary to configure TLS between the host and client.
-d	Prints debug information.

7.8. 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**.

7.9. 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-----
MMIIBtgYJYIZIAyb4QgIFoIIBpzCCAZ8wggGbMIIBRaADAgEAAgEBMA0GCSqGSIb3DQEEBB
AUAMFcxC
AJBgNVBAYTAIVTMSwwKgYDVQQKEyNOZXRxZyY2FwZSBDb21tdW5pY2F0aWhfyyuougijjg
mkgjkgmjg
fjfgjjgfyjfyj9ucyBDb3Jwb3JhdGlvbJpMEaMBgGA1UECxMRSXNzdWluZyhgdfhbfdfpfjphotoo
gdhkBBdXRob3JpdHkwHhcNOTYxMTA4MDkwNzMM0WhcNOTgxMTA4MDkwNzMM0WjBXMQ
swCQYDVQQGEwJ
```

```
VUzEsMCoGA1UEChMjTmV0c2NhcGUgQ29tbXVuaWNhdGlvbnMgQ29ycG9yY2F0aW9ucyB
Db3Jwb3Jhd
GlvbnpMEaMBgGA1UECXMRSXNzdWluZyBBdXRob3JpdHkwHh
-----END CERTIFICATE-----
```

- 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.

- 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.

- Revoke the certificate.
- 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.

- Convert the DER-encoded CRL to its base 64-encoded format using the Binary to ASCII tool.

```
BtoA input_file output_file
```

- Convert the base 64-encoded CRL to readable form using the Pretty Print CRL tool.

```
PrettyPrintCrl input_file [output_file]
```

- Compare the output.

7.10. 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:

- Convert the base-64 file to binary. For example:

```
AtoB /tmp/example.b64 /tmp/example.bin
```

- 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
```

7.11. 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.

7.11.1. 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.

- 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.

7.11.2. 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.

CHAPTER 8. AUTHENTICATION FOR ENROLLING CERTIFICATES

8.1. AUTOMATIC APPROVAL BY AN AUTHENTICATION PLUG-IN

The ***auth.instance_id*** parameter in a profile specifies the authentication mechanism. A certificate request can either be automatically approved through an authentication plug-in, or be manually approved by a CA agent.



NOTE

For instructions on how to edit certificate enrollment profiles, see [Section 3.2, "Setting up Certificate Profiles"](#).

8.1.1. Setting up Auto-approval of Enrollment Requests

Configuring that enrollment requests are automatically approved depends on the type of requests:

- For agent-pre-approved CMC requests, set in the CA profile:

```
auth.instance_id=CMCAuth
authz.acl=group="Certificate Manager Agents"
```

The ***authz.acl*** parameter defines the group that is allowed to approve requests.

- For user-initiated requests:
 - When using CMC Shared Token, set in the CA profile:

```
auth.instance_id=CMCUserSignedAuth
```

Required default and constraint:

```
policyset.cmcUserCertSet.1.constraint.class_id=cmcSharedTokenSubjectNameConstraintImpl
policyset.cmcUserCertSet.1.constraint.name=CMC Shared Token Subject Name Constraint
policyset.cmcUserCertSet.1.default.class_id=authTokenSubjectNameDefaultImpl
policyset.cmcUserCertSet.1.default.name=Subject Name Default
```

- When using User-signed requests, set in the CA profile:

```
auth.instance_id=CMCUserSignedAuth
```

Required default and constraint:

```
policyset.cmcUserCertSet.1.default.params.name=
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
```

■

For details about editing a profile, see [Section 3.2, "Setting up Certificate Profiles"](#).

8.1.2. CMC Authentication Plug-ins

Certificate System provides the following 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 8.1.3, "CMC SharedSecret Authentication"](#).

By default, the following enrollment profiles use the **CMCUserSignedAuth** plug-in:

- **caFullCMCUserSignedCert**
- **caECFullCMCUserSignedCert**
- **caFullCMCSharedTokenCert**
- **caECFullCMCSharedTokenCert**

8.1.3. 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.

8.1.3.1. Creating a Shared Secret Token

The *The Shared Secret Workflow* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)* 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 8.1.3.2, "Setting a CMC Shared Secret"](#).

8.1.3.2. 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 (Common Criteria Edition)*.

8.1.3.2.1. 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
```

8.1.3.2.2. 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
```

8.2. MANUAL APPROVAL BY A CA AGENT

If manual approval is configured in a CA profile, users can create CMC requests, but requests must be manually approved by a CA agent.

To configure manual agent approval in a CA profile:

1. Set the ***auth.instance_id*** to an empty value:

```
auth.instance_id=
```


2. Do not set the ***authz.acl*** parameter.

For details about editing a profile, see [Section 3.2, “Setting up Certificate Profiles”](#).

8.3. 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.

8.4. 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.

List Requests
Use this form to show a list of certificate requests.

Request type: ▼

Request status: ▼

Starting request number:

first records

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.

Additional Notes

▼

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 9. AUTHORIZATION FOR ENROLLING CERTIFICATES (ACCESS EVALUATORS)

This chapter describes the authorization mechanism using access evaluators.



NOTE

For instructions on how to edit certificate enrollment profiles, see [Section 3.2, “Setting up Certificate Profiles”](#).

9.1. 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.

9.2. DEFAULT EVALUATORS

Red Hat Certificate System provides four default evaluators.

```
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.

PART IV. MANAGING THE SUBSYSTEM INSTANCES

CHAPTER 10. SELF TESTS

10.1. 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.

10.1.1. Running Self-Tests

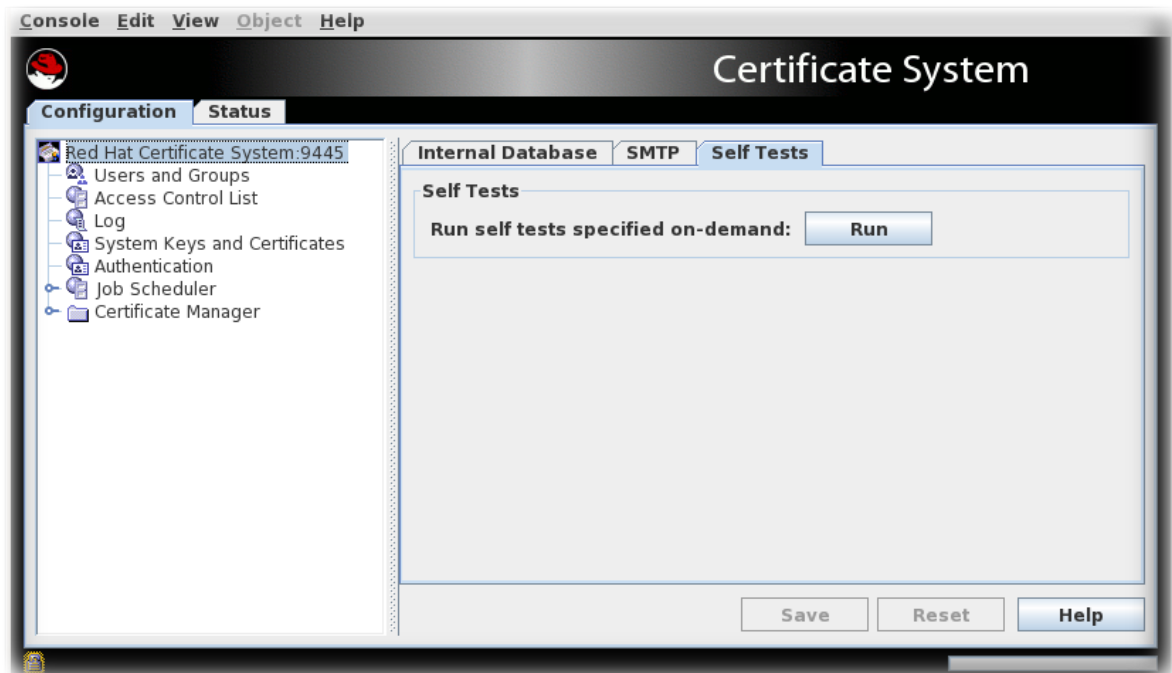
The on-demand self-test for the CA, OCSP, KRA, or TKS subsystems are run from the console. The on-demand self-tests for the TPS system are run from the web services page.

10.1.1.1. Running Self-Tests from the Console

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.

10.1.1.2. 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**

10.2. DEBUGGING SELF-TESTS FAILURES

In the event of self-test failure, the Certificate System instance will stop completely and will not respond to any HTTP or HTTPS requests.

To diagnose a manually run self-test failure, refer to the various logs described in [Section 10.2.1, “Self-Test Logging”](#). Often other logs are useful as well, including debug logs. For more information on subsystem logs, refer to [Chapter 13, *Configuring Subsystem Logs*](#). For more information on debug logs, refer to the *Logs* section under the *Certificate System Architecture Overview* in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

Common causes of self-test failures are services (such as LDAP) are down or unreachable, certificates are expired, or the system configuration is wrong. A precise cause of self-test failure is given in the logs.

After the cause of the self-test failure is identified and fixed, please restart the Certificate System server to resume normal operations:

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

10.2.1. Self-Test Logging

A separate log, **selftests.log**, is added to the log directory that contains reports for both the start up self-tests and the on-demand self-tests.

CHAPTER 11. MANAGING CERTIFICATE/KEY CRYPTO TOKEN

This chapter contains instructions on how to manage certificate/key token database on the crypto tokens, specifically on how to import and verify certificates for various scenarios.

About Crypto Tokens

For information about NSS soft token, please see 2.3.8.1. NSS Soft Token (internal token) in Red Hat Certificate System's Planning, Installation, and Deployment Guide.

11.1. ABOUT CERTUTIL AND PKICERTIMPORT

The **certutil** command is provided by Network Security Services (NSS). **certutil** is used for validating and importing certificates. A basic overview of how we use **certutil** is presented below, however, **PKICertImport** is our wrapper script of choice for safely validating and importing certificates. Using **certutil** to do so requires multiple command invocations and correct usage is outside the scope of this documentation.

11.1.1. certutil Basic Usage

certutil [command] [options]

Each **certutil** invocation takes a command flag, usually denoted by a capital letter, and a series of options which control what the command does. If an option takes a value, that value is named between "<" and ">" symbols.

11.1.2. PKICertImport Basic Usage

PKICertImport [options]

Each **PKICertImport** invocation accepts a series of options to validate and import a specified certificate. Unlike the broad use cases of **certutil**, **PKICertImport** is only focused on safely importing and validating certificates. See [Section 11.1.4, "Common certutil and PKICertImport Options"](#) for more information about available options.



NOTE

PKICertImport prompts for the NSS DB and/or HSM passwords multiple times throughout the course of its execution. This is expected as **PKICertImport** has to interact with the NSS DB multiple times. To avoid having to input the NSS DB password repetitively, specify a password file via **-f <filename>**. When done, be sure to delete the password file.

11.1.3. certutil Common Commands

The following commands are specific to **certutil** and provide a brief overview of several common commands. **PKICertImport** is not compatible with nor require these command flags.

certutil -A

The **-A** command denotes "adding" a certificate. It requires a certificate to import (**-i**), a nickname (**-n**) for that certificate, and a series of trust flags (**-t**) for the certificate.

certutil -V

The **-V** command denotes "verifying" a certificate. It requires a certificate nickname to validate (**-n**) and a type of verification (**-u**) to perform.

certutil -D

The **-D** command denotes "deleting" a certificate. It requires a certificate nickname (**-n**) to remove.

Note that it ONLY removes the PUBLIC KEY portion of the certificate, and WILL NOT remove any private keys, if present.

certutil -M

The **-M** command denotes "modifying" a certificate. It requires a certificate nickname (**-n**) to modify, and a series of trust flags (**-t**) to give the certificate.

certutil -L

The **-L** command denotes "listing" a certificate or all certificates. If given the nickname option (**-n**), it will list detailed information about that certificate, else if omitted, it will list general information about all certificates present.

The result of **certutil -L** would show each certificate by its nickname along with its trust info. For example:

Table 11.1. Certificate nickname and trust info

Certificate Nickname	Trust Attributes SSL, S/MIME, JAR/XPI
caSigningCert pki-ca1	CT, C, C



NOTE

The trust attributes displayed by **certutil -L** correspond to what is specified with the **-t** option.

The **certutil -L** does not modify the database, and can thus be executed safely as many times as desired.

11.1.4. Common certutil and PKICertImport Options

When following the steps below, ensure the values are relevant and correct for your specific deployment scenario. Many of these options are available to **PKICertImport** as well.

-n <nickname>

The **-n <nickname>** option specifies the nickname for a certificate. This can be any text and is only used as a reference to the certificate. It **MUST** be unique.

Update this value as appropriate for your configuration.

-d <directory>

The **-d <directory>** option specifies the path to the NSS DB directory in use. We usually assume you are already in this directory and use "." to refer to the current directory.

Update this value as appropriate for your configuration.

-t <trust>

The **-t <trust>** option specifies the trust level for the certificate.

There are three main categories of trust:

- trust for TLS
- trust for email
- trust for object signing

Each trust position can have one or more trust letters which specify the desired level of trust. The trust letters we use below are **c**, **C**, and **T**.

- **c** states that this certificate should be a Certificate Authority (CA).
- **C** states that this is a trusted certificate authority for signing server certificates (**C** implies lowercase **c**, hence you do not need to specify both).
- **T** states that this certificate is a trusted authority for signing client certificates (**T** implies lowercase **c**, hence you do not need to specify both **T** and **c**).

To specify the trust flags for each position, join the letters with commas. For example, the option **-t CT,C,c** means that the certificate is trusted for signing client and server TLS certificates, signing server email certificates (S/MIME), and is a valid CA for object signing (though untrusted).

- This ensures that, if this certificate signs another certificate, which in turn is used for object signing, it will be deemed invalid.

No trust (or the lack of trust) can be specified by using **-t ,,**.

To see the trust levels of all certificates in the database, run:

- **certutil -L -d**
- Each certificate's nickname will be listed and the trust flags will be specified at the end of the line.

See the notes about HSMs in the **-h** option.

Note that more trust levels are specified in the certutil manpage. To reference this documentation, please run the **man certutil** command on a system with certutil properly installed.

-h <HSM>

The **-h <HSM>** option specifies the name of the HSM to perform operations on.

-h option is incompatible with the **-t** option, as HSMs cannot store trust. Only an NSS DB can store trust, so using the **certutil -A** command or the **certutil -M** command in conjunction with **-h <HSM>** will fail. Instead, specify the desired trust level on a separate **certutil -M** command without the **-h** option.

Update this value as appropriate for your configuration.

-e

The **-e** option specifies that the validity of the signature is checked as well, when used in conjunction with the **certutil -V** command. **PKICertImport** always performs the certificate signature validation and does not understand the **-e** option.

-a

The **-a** option specifies that the key in question is in PEM (ASCII) format.

-i <certificate>

The **-i <certificate>** option specifies the path to the certificate. This is only used in the **certutil -A** command to specify the path to the certificate to import.

Update this value as appropriate for your configuration.

-u <usage>

The **-u <usage>** option specifies that usage of the certificate to verify when used in conjunction with the **certutil -V** command.

There are several usage letters referenced in the following sections.

- **-u C** stands for verify a client TLS certificate. Note that this mostly accepts any certificate, but will check expiration date and signature.
- **-u V** stands for verify a server TLS certificate. Note that this will reject CA certificates and will check expiration date and signature.
- **-u L** stands for verify a CA TLS certificate. Note that this will validate trust flags (to see if **c** is present) and will check key usage to ensure that the key is a CA key. This also checks expiration and signatures.
- **-u O** stands for verify a OCSP status responder certificate. Note that this checks expiry and signatures.
- **-u J** stands for verify an object signing certificate. Note that this checks expiry and signatures.

If the wrong usage option is specified or the trust flags on the certificate are wrong (such as a missing **c** flag for an CA TLS certificate), **certutil -V** will give incorrect results.



NOTE

More usage options are specified in the certutil manpage. To reference this documentation, run the **man certutil** command on a system with certutil properly installed.

11.2. IMPORTING A ROOT CERTIFICATE

First, change directories into the NSS DB:

- **cd /path/to/nssdb**

Ensure that your web service is taken offline (stopped, disabled, etc.) while performing these steps and ensure no concurrent access to the NSS DB by other processes (such as a browser). Doing so may corrupt the NSS DB or result in improper usage of these certificates.

When needing to import a new root certificate, ensure you acquire this certificate in a secure manner as it will be able to sign a number of certificates. We assume you already have it in a file named **ca_root.crt**. Please substitute the correct name and path to this file as appropriate for your scenario.

For more information about the **certutil** and **PKICertImport** options used below, see [Section 11.1, "About certutil and PKICertImport"](#).

To import the root certificate:

- Execute **PKICertImport -d . -n "CA Root" -t "CT,C,C" -a -i ca_root.crt -u L** command.

This command validates and imports the root certificate into your NSS DB. The validation succeeds when no error message is printed and the return code is 0. To check the return code, execute **echo \$?** immediately after executing the previous command above. In most cases, a visual error message is printed. The certificate usually fails to validate because it is expired or because it is not a CA certificate. Therefore, make sure your certificate file is correct and up-to-date. Contact the issuer and ensure that all intermediate and root certificates are present on your system.

11.3. IMPORTING AN INTERMEDIATE CERTIFICATE CHAIN

Before beginning, please change directories into the NSS DB:

- **cd /path/to/nssdb**

Ensure that your web service is offline (stopped, disabled, etc.) while performing these steps and ensure no concurrent access to the NSS DB by other processes (such as a browser). Doing so may corrupt the NSS DB or result in improper usage of these certificates.

If you have not imported and trusted the root certificate, see [Section 11.2, "Importing a Root Certificate"](#).

When given a series of intermediate certificates between your root and end server or client certificates, we need to import and validate the signed certificate chain in order from closest to furthest from the root CA certificate. We assume the Intermediate CAs are in files named **ca_sub_<num>.crt** (for example **ca_sub_1.crt**, **ca_sub_2.crt**, and so on). Substitute names and paths for your certificates as appropriate to your deployment.



NOTE

In the unlikely scenario that you are instead given a single file named **fullchain.crt**, **fullchain.pem**, or similar and it contains multiple certificates, split it into the above format by copying each block (between and including the **-----BEGIN CERTIFICATE-----** and an **-----END CERTIFICATE-----** markers) to its own file. The first ones should be named **ca_sub_<num>.crt** and the last will be your server cert named **service.crt**. Server certificates are discussed in later sections.

First, we will import and validate any intermediate CAs in order of closest to furthest from the root CA certificate. If you don't have any, you can skip to the next section.

For more information about the **certutil** and **PKICertImport** options used below, see [Section 11.1, "About certutil and PKICertImport"](#).

For every intermediate certificate in the chain:

- Execute **PKICertImport -d . -n "CA Sub \$num" -t "CT,C,C" -a -i ca_sub_\$num.crt -u L**

This command validates and imports the Intermediate CA certificate into your NSS DB. The validation succeeds when no error message is printed and the return code is 0. To check the return code, execute **echo \$?** immediately after executing the previous command above. In most cases, a visual error message is printed. If the validation does not succeed, contact the issuer and ensure that all intermediate and root certificates are present on your system.

11.4. IMPORTING A CERTIFICATE INTO AN NSS DATABASE

Ensure that your web service is taken offline (stopped, disabled, etc.) while performing these steps and ensure no concurrent access to the NSS database by other processes (such as a browser). Doing so may corrupt the NSS database or result in improper usage of these certificates.

Note that which set of instructions you follow will depend on the usage for the certificate in question.

- For any subsystem's **auditSigningCert**, please follow the steps below for validating an *object Signing certificate*.
- For the CA subsystem's **caSigningCert**, please follow the steps above for importing and validating an *intermediate certificate chain*, but do so only with the `caSigningCert`.
- For the CA subsystem's **ocspSigningCert**, please follow the steps below for validating an *OCSP certificate*.
- For user's client or S/MIME certificate, follow the *Client Certificate steps*.

For more information about the **certutil** and **PKICertImport** options used below, see [Section 11.1, "About certutil and PKICertImport"](#).

Importing a Client Certificate Into the NSS Database

To import a client certificate into the NSS database:

1. Change into the NSS database directory. For example:

```
# cd /path/to/nssdb/
```

2. Import and trust the root certificate, if it is not already imported and trusted. For details, see [Section 11.2, "Importing a Root Certificate"](#).
3. Import and validate the intermediate certificates, if not already imported and validated. For details, see [Section 11.3, "Importing an Intermediate Certificate Chain"](#).
4. Validate and import the client certificate:

```
# PKICertImport -d . -n "client name" -t "," -a -i client.crt -u C
```

The validation succeeds when no error message is printed and the return code is 0. To check the return code, execute **echo \$?** immediately after executing the previous command above. In most cases, a visual error message is printed. If the validation does not succeed, contact the issuer and ensure that all intermediate and root certificates are present on your system.

Importing an Object Signing Certificate

To import an object signing certificate:

1. Change into the NSS database directory. For example:

```
# cd /path/to/nssdb/
```

2. Import and trust the root certificate, if it is not already imported and trusted. For details, see [Section 11.2, "Importing a Root Certificate"](#).
3. Import and validate the intermediate certificates, if not already imported and validated. For details, see [Section 11.3, "Importing an Intermediate Certificate Chain"](#).

4. Validate and import the object signing certificate:

```
# PKICertImport -d . -n "certificate name" -t ",,P" -a -i objectsigning.crt -u J
```

The validation succeeds when no error message is printed and the return code is 0. To check the return code, execute **echo \$?** immediately after executing the previous command above. In most cases, a visual error message is printed. If the validation does not succeed, contact the issuer and ensure that all intermediate and root certificates are present on your system.

Importing an OCSP Responder

To import an OCSP responder:

1. Change into the NSS database directory. For example:

```
# cd /path/to/nssdb/
```

2. Import and trust the root certificate, if it is not already imported and trusted. For details, see [Section 11.2, "Importing a Root Certificate"](#).
3. Import and validate the intermediate certificates, if not already imported and validated. For details, see [Section 11.3, "Importing an Intermediate Certificate Chain"](#).
4. Validate and import the OCSP responder certificate:

```
# PKICertImport -d . -n "certificate name" -t ",," -a -i ocspr.crt -u O
```

The validation succeeds when no error message is printed and the return code is 0. To check the return code, execute **echo \$?** immediately after executing the previous command above. In most cases, a visual error message is printed. If the validation does not succeed, contact the issuer and ensure that all intermediate and root certificates are present on your system.

CHAPTER 12. 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.

12.1. ABOUT AUTHORIZATION



NOTE

The ACLs associated with each group discussed in this section must not be modified.

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.

Authorization goes through the following process:

1. The users authenticate to the interface using a certificate.
2. The server authenticates the user by checking the certificate against one stored in the database. 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.
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.

12.2. 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.

12.2.1. 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 12.1. Security Domain User Roles

Role	Description
Security Domain Administrators	<ul style="list-style-type: none"> • 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. <p>By default, the CA administrator of the CA hosting the domain is assigned as the security domain administrator.</p>
Enterprise CA Administrators	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● 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.

12.2.2. 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.

12.2.3. Agents

Agents are users who have been assigned end-entity certificate and key-management privileges. Agents can access the agent services interface.

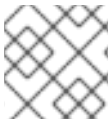
Agents are created by assigning a user to the appropriate subsystem agent group and identifying certificates that the agents must use for TLS client authentication to the subsystem for it to service requests from the agents. Each subsystem has its own agent group:

- 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 TLS client certificate in its internal database.

12.2.4. 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 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 TLS client authentication with the subsystem certificate issued during configuration by the security domain CA.

12.3. 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.

12.3.1. Managing Groups

12.3.1.1. 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.

The screenshot shows a dialog box titled "Edit Group Information". It has a title bar with a dropdown arrow on the left, the text "Edit Group Information", and a close button on the right. The main area contains three text input fields: "Group name:" with the value "New Group", "Group description:" with the value "group for viewing logs", and "Group Members:" with a list box containing "user0" and "user1". To the right of the list box are "Add User" and "Delete" buttons. At the bottom are "OK", "Cancel", and "Help" buttons.

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 12.4.3, "Editing ACLs"](#) for more information. If no ACLs are added to the ACLs for the group, the group will have no access permissions to any part of Certificate System.

12.3.1.2. 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**.

12.3.2. 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.

12.3.2.1. 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 and audit trails, create individual accounts for Certificate System users and administrators.

12.3.2.1.1. 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 -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 -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:

```
# PKCS10Client -d ~/.dogtag/pki-instance_name/ -p password \  
-n "user_name" -o ~/user_name.req
```

This command stores the CSR in **pkcs10** format in the `~/user_name.req` file.

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 cert8.db, key3.db and secmod.db  
dbdir=~/dogtag/pki-instance_name/  
  
#password: password for cert8.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 TLS client authentication cert can be found (default is
internal)
#This parameter will be ignored if secure=false
tokenname=internal

#dbdir: directory for cert8.db, key3.db and secmod.db
#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 cert8.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
MIIJ1wYJKoZIhvcNAQcCoIIJyDCCCcQCAQMxDzANBgIghkgBZQMEEAgEFADAxBggr
...
The response in data format is stored in ~/cmc.role_crmf.resp

```

- c. Verify the result:

```

# CMCRResponse ~/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: CMCRStatusInfoV2
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 -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:

```
pkiconsole -c password -n caadmin ca-user-cert-add example --serial 0x6
```

12.3.2.1.2. Creating Users Using the Console

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.

The screenshot shows a dialog box titled "Edit User Information". It contains the following fields and values:

- User ID:** example1
- Full name:** Example User
- Password:** *****
- Confirm Password:** *****
- E-Mail:** email@example.com
- Phone:** 100-555-1212
- User State:** 1
- Group:** Certificate Manager Agents (selected from a dropdown menu)

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

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.

12.3.2.2. 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.

12.3.2.3. 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 administrator 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 certificate. For details, see [Section 5.4, "Renewing Certificates"](#).
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.

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**.

12.3.2.4. 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.

12.4. 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.

12.4.1. About Access Control

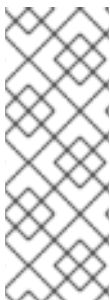
Access control lists (ACLs) are the mechanisms that specify the authorization to server operations. An ACL exists for each set of operations where an authorization check occurs. Additional operations can be added to a ACL.

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, *0:0:0:0:0:13.168.3*, *FF01::43*, *0:0:0:0:0:13.168.3,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:

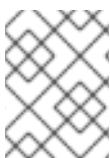
```
allow (read,modify) group="Administrators"
```

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 ACL 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 IP 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:

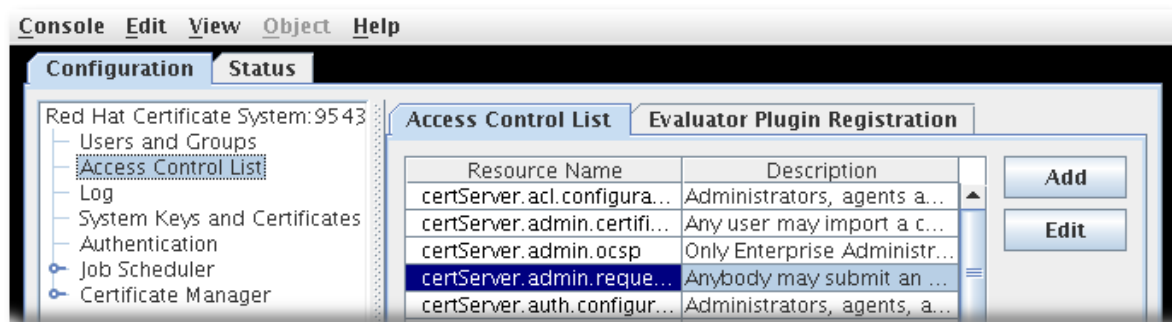
```
user="BobC" || group="Auditors" || group="Administrators"
```

12.4.2. Adding ACLs

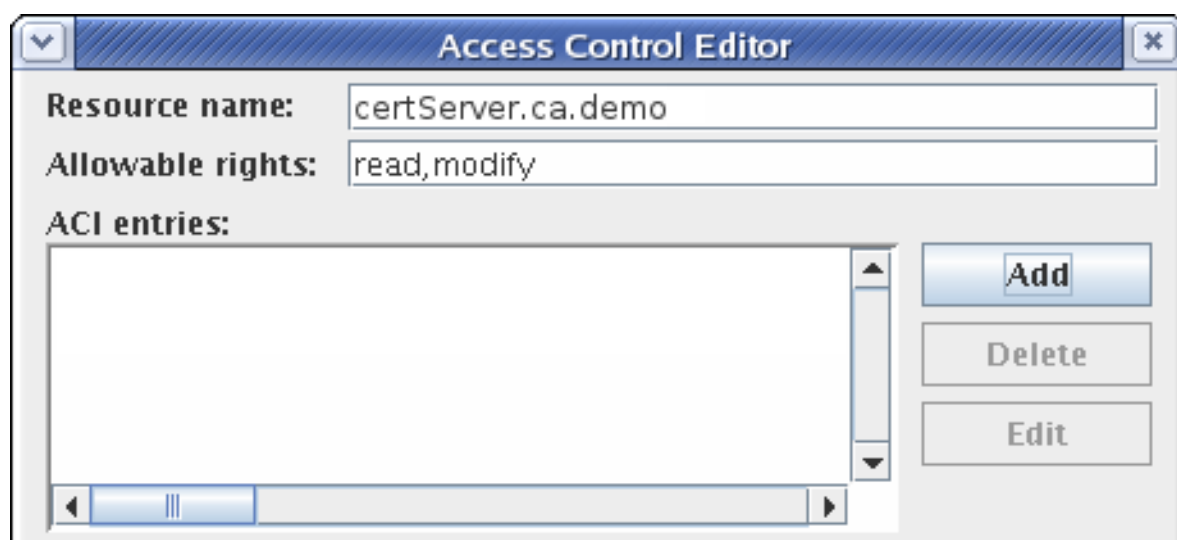
ACLs are stored in the internal database and can only be modified in the administrative console.

To add a new ACL:

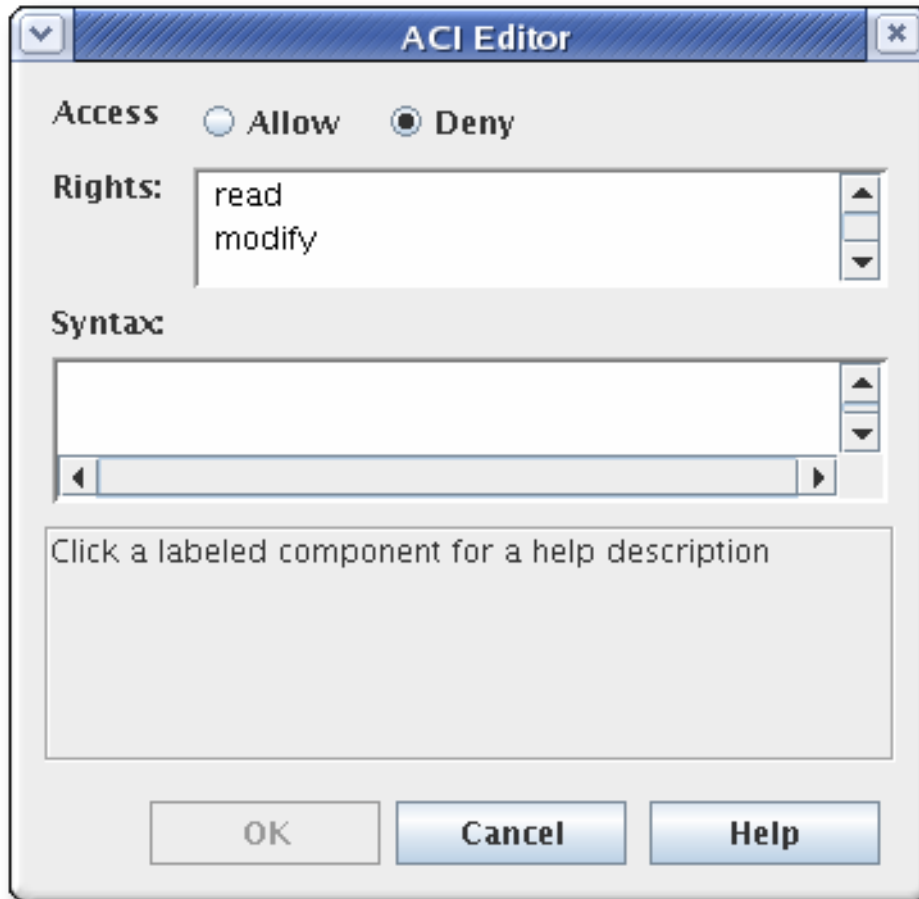
1. Log into the administrative console.
2. Select **Access Control List**.



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.



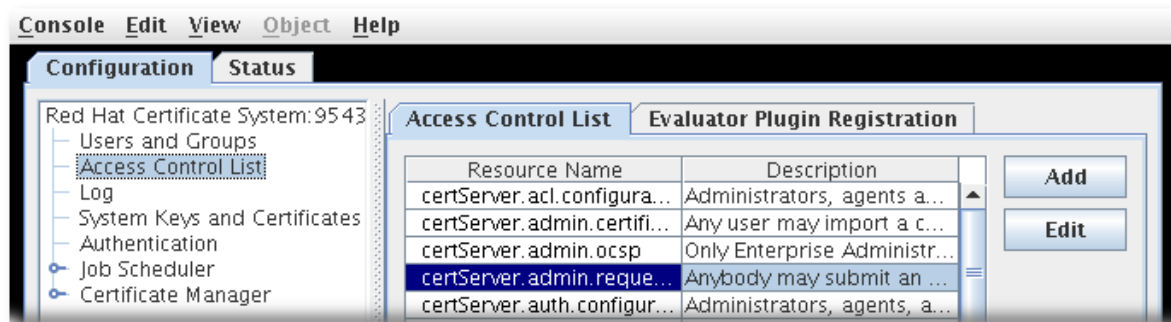
- a. Select the allow or deny radio button from the **Access** field to allow or deny the operation to the groups, users, or IP addresses specified. For more information about allowing or denying access, see [Section 12.4.1, "About Access Control"](#).
 - b. Set the rights. The available options are **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 12.4.1, "About Access Control"](#) for details on syntax.
6. Click **OK** to return to the **Access Control Editor** window.
 7. Click **OK** to store the ACL.

12.4.3. Editing ACLs

ACLs are stored in the internal database and can only be modified in the administrative console.

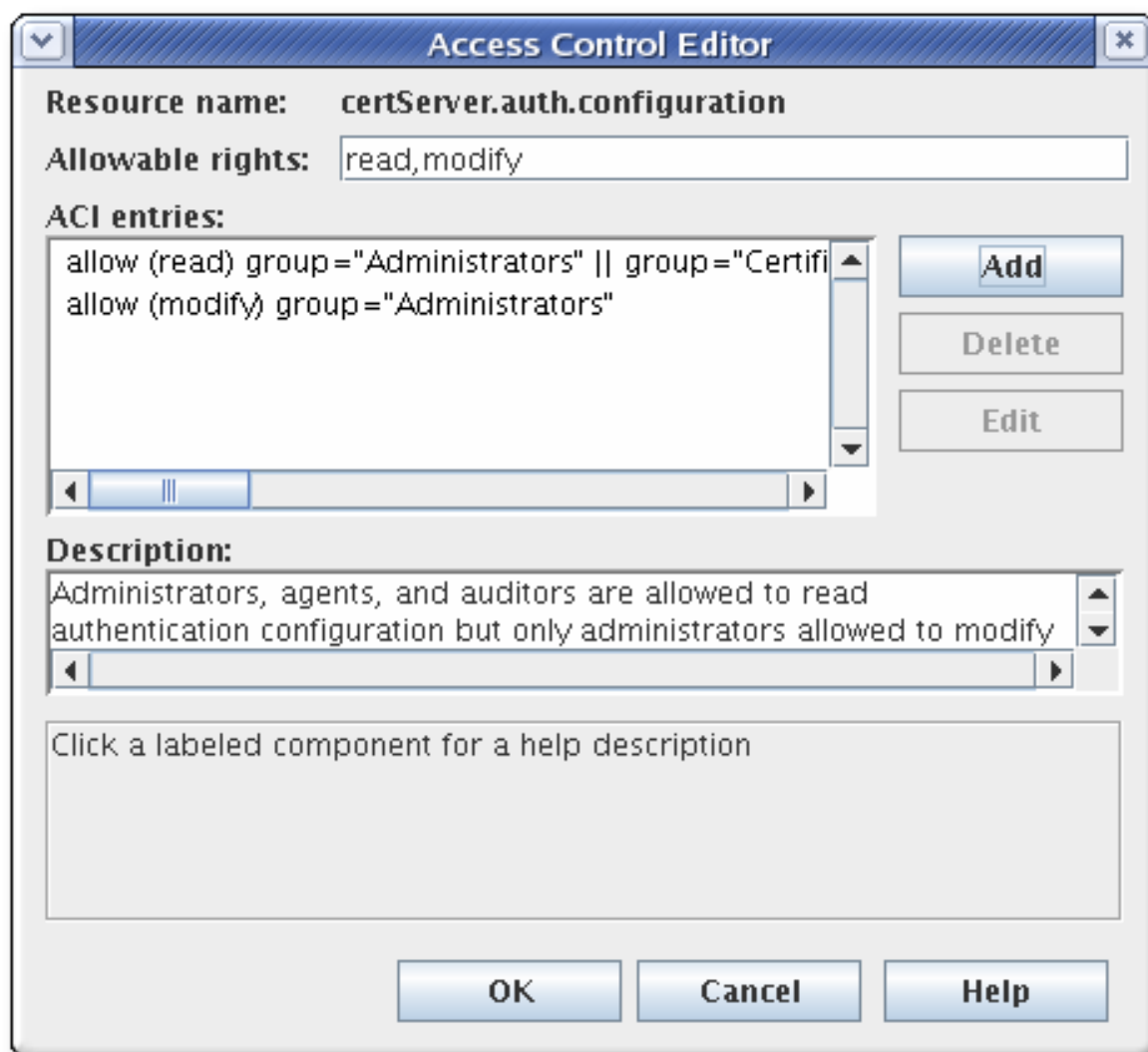
To edit the existing ACLs:

1. Log into the administrative console.
2. Select **Access Control List** in the left navigation menu.



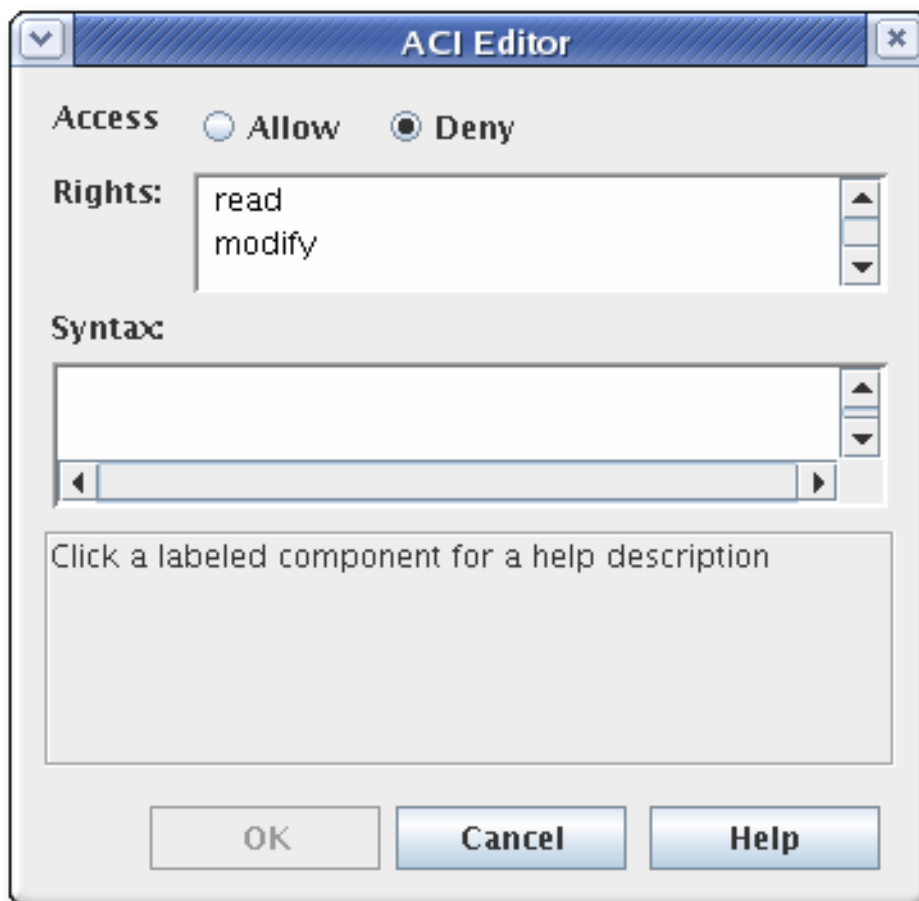
3. Select the ACL to edit from the list, and click **Edit**.

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 12.4.1, "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 12.4.1, "About Access Control"](#) for details on syntax.

CHAPTER 13. CONFIGURING SUBSYSTEM LOGS

For an overview on logs, see the *Logs* section under the *Certificate System Architecture Overview* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

For log configuration during the installation and additional information, see the *Configuring Logs* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

13.1. 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.

13.1.1. Configuring Logs in the Console

Logs can be configured through the subsystem Console. Specialized logs, such as signed audit logs and custom logs, can also be created through the Console or 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 13.1, “Log Event Listener Fields”](#).

Table 13.1. Log Event Listener Fields

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. system creates error and system logs; 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 .
fileName	Gives the full path, including the file name, to the log file. The subsystem user should have read/write permission to the file.

Field	Description
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.
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. 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.

13.1.2. Managing Audit Logs

The audit log contains audit records for events that have been set up as recordable events. If you enabled the audit log signing option as described in the *Enabling and Configuring Signed Audit Logs* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*, 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.



NOTE

Signed audit logs are optional. To enable them, please refer to [Section 13.1.2.1, “Configuring a Signed Audit Log in the Console”](#).

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.

It is also possible to edit the configuration or change the signing certificates after configuration, as covered in [Section 13.1.2.1, “Configuring a Signed Audit Log in the Console”](#).

13.1.2.1. 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 the installation.



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.
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 two fields which must be reset in the **Log Event Listener Editor** window.
 - Set the **logSigning** field to **true** to enable signed logging.



NOTE

For more fine-grained audit event select, set audit event filters during the installation configuration. For details, see the *Filtering Audit Events* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

- 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. 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 12.3.2.1, "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. For further details, see [Section 13.2.2, "Using Signed Audit Logs"](#).

13.1.2.2. 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 log signing is enabled and 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. When this is completed, the administrators can restart the Certificate System.

13.2. USING LOGS

13.2.1. 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 13.2.2, "Using Signed Audit Logs"](#).

To view the contents of an active or rotated system 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. Choose **Current** to view the currently active system log file.
5. Click **Refresh**.

The table displays the system log entries. The entries are in reverse chronological order, with the most current entry placed at the top. Use the scroll arrows on the right edge of the panel to scroll through the log entries.

Each entry has the following information shown:

- **Source** – The component or resource that logged the message.
- **Level** – The severity of the corresponding entry.
- **Date** – The date on which the entry was logged.
- **Time** – The time at which the entry was logged.
- **Details** – A brief description of the log.

- To view a full entry, double-click it, or select the entry, and click **View**.

13.2.2. Using Signed Audit Logs

This section explains how a user in the Auditor group displays and verifies signed audit logs.

13.2.2.1. 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.

13.2.2.2. 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**:

- Optionally, list the available log files on the CA. See [Section 13.2.2.1, "Listing Audit Logs"](#).
- 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
```

13.2.2.3. 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 the *Command-line Initialization* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.
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 13.2.2.2, "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.

13.2.3. 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 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

To display operating system-level access logs, use the **ausearch** utility as root or as a privileged user with the **sudo** utility.

13.2.3.1. 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
```

13.2.3.2. 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
```

13.2.3.3. 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
```

13.2.3.4. 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
```

13.2.3.5. 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
```


CHAPTER 14. 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.

14.1. 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 *Types of Certificates* section in the *Red Hat Certificate System 9 Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

14.1.1. Certificate Manager Certificates

When a Certificate Manager is installed, the keys and requests for the CA signing certificate, TLS 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 14.1.1.1, "CA Signing Key Pair and Certificate"](#)
- [Section 14.1.1.2, "OCSP Signing Key Pair and Certificate"](#)
- [Section 14.1.1.3, "Subsystem Certificate"](#)
- [Section 14.1.1.4, "TLS Server Key Pair and Certificate"](#)
- [Section 14.1.1.5, "Audit Log Signing Key Pair and Certificate"](#)

14.1.1.1. 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.

14.1.1.2. 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.

14.1.1.3. 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 TLS 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***.

14.1.1.4. TLS Server Key Pair and Certificate

Every Certificate Manager has at least one TLS 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 TLS 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 TLS-enabled communication with a publishing directory, it uses its TLS server certificate for client authentication to the publishing directory by default. The Certificate Manager can also be configured to use a different certificate for TLS client authentication.

14.1.1.5. 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.

14.1.2. 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, TLS 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 14.1.2.2, “TLS Server Key Pair and Certificate”](#)
- [Section 14.1.2.3, “Subsystem Certificate”](#)
- [Section 14.1.2.4, “Audit Log Signing Key Pair and Certificate”](#)
- [Section 14.1.2.5, “Recognizing Online Certificate Status Manager Certificates”](#)

14.1.2.1. 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.

14.1.2.2. TLS Server Key Pair and Certificate

Every Online Certificate Status Manager has at least one TLS 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.

14.1.2.3. 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 TLS certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-instance_ID**.

14.1.2.4. 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.

14.1.2.5. Recognizing Online Certificate Status Manager Certificates

Depending on the CA which signed the Online Certificate Status Manager's TLS 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 TLS 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.

14.1.3. Key Recovery Authority Certificates

The KRA uses the following key pairs and certificates:

- [Section 14.1.3.1, "Transport Key Pair and Certificate"](#)
- [Section 14.1.3.2, "Storage Key Pair"](#)
- [Section 14.1.3.3, "TLS Server Certificate"](#)
- [Section 14.1.3.4, "Subsystem Certificate"](#)
- [Section 14.1.3.5, "Audit Log Signing Key Pair and Certificate"](#)

14.1.3.1. 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.

14.1.3.2. 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.

14.1.3.3. TLS Server Certificate

Every Certificate System KRA has at least one TLS server certificate. The first TLS 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 TLS 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 TLS server certificate for server-side authentication to the KRA agent services interface. By default, the Key Recovery Authority uses a single TLS server certificate for authentication. However, additional TLS server certificates can be requested and installed for the KRA.

14.1.3.4. 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 TLS certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert cert-*instance_ID***.

14.1.3.5. 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.

14.1.4. TKS Certificates

The TKS has three certificates. The TLS server and subsystem certificates are used for standard operations. An additional signing certificate is used to protect audit logs.

- [Section 14.1.4.1, "TLS Server Certificate"](#)
- [Section 14.1.4.2, "Subsystem Certificate"](#)
- [Section 14.1.4.3, "Audit Log Signing Key Pair and Certificate"](#)

14.1.4.1. TLS Server Certificate

Every Certificate System TKS has at least one TLS server certificate. The first TLS server certificate is generated when the TKS is configured. The default nickname for the certificate is **Server-Cert** *cert-instance_ID*.

14.1.4.2. 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 TLS certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert** *cert-instance_ID*.

14.1.4.3. 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.

14.1.5. TPS Certificates

The TPS only uses three certificates: a server certificate, subsystem certificate, and audit log signing certificate.

- [Section 14.1.5.1, "TLS Server Certificate"](#)
- [Section 14.1.5.2, "Subsystem Certificate"](#)
- [Section 14.1.5.3, "Audit Log Signing Key Pair and Certificate"](#)

14.1.5.1. TLS Server Certificate

Every Certificate System TPS has at least one TLS server certificate. The first TLS server certificate is generated when the TPS is configured. The default nickname for the certificate is **Server-Cert** *cert-instance_ID*.

14.1.5.2. 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 TLS certificate. This certificate is signed by the security domain CA.

The default nickname for the certificate is **subsystemCert** *cert-instance_ID*.

14.1.5.3. 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.

14.1.6. 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 14.1. 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 (Common Criteria Edition)*.
- The `pki_default.cfg(5)` man page for descriptions of the parameters and examples.

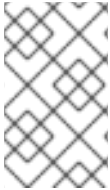
14.1.7. Using an HSM to Store Subsystem Certificates

By default, keys and certificates are stored in locally-managed databases, **key3.db** and **cert8.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. For example:

```
serverCert="nethsm:Server-Cert cert-instance_ID
```



NOTE

A single HSM can be used to store certificates and keys for multiple 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.

14.2. RENEWING SUBSYSTEM CERTIFICATES

For details about renewing subsystem certificates, see [Section 3.4.1, "About Renewal"](#).

The process for renewing a subsystem certificate is the same as for renewing a user certificate.

To renew a system certificate submit the request using the corresponding enrollment profile when using the **HttpClient** utility. For details about the different system certificate profiles, see [Section 5.3.2.1, "Obtaining System and Server Certificates"](#).

14.2.1. 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 converted to a CMC request to be submitted to the CA. For details, see [Section 5.2.1, "Creating a CSR Using certutil"](#).



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 -k "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 **-k** 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 **-k** 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.

14.2.2. Renewing Expired Certificate System Server Certificates

Certificate System does not automatically renew its server certificates online while the PKI server is running. In general, administrators should renew the system certificates before they expire. However, if a system certificate expires, Certificate System will fail to start.

To renew system certificates after expiration, you can set up a temporary server certificate:

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:

```
# systemctl start pki-tomcatd-nuxwdog@instance_name.service
```

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:

```
# systemctl stop pki-tomcatd-nuxwdog@instance_name.service
```

- 5. Import the new certificate to replace the expired certificate:

```
# pki-server cert-import certificate_ID
```

- 6. Start Certificate System:

```
# systemctl start pki-tomcatd-nuxwdog@instance_name.service
```

14.3. 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.

These tables list all of the configuration parameters for each of the subsystem's certificates:

- [Table 14.1, "CA Certificate Nickname Parameters"](#)
- [Table 14.2, "KRA Certificate Nickname Parameters"](#)
- [Table 14.3, "OCSP Certificate Nickname Parameters"](#)
- [Table 14.4, "TKS Certificate Nickname Parameters"](#)
- [Table 14.5, "TPS Nickname Parameters"](#)

Table 14.1. CA Certificate Nickname Parameters

CA Signing Certificate	<ul style="list-style-type: none"> ● ca.cert.signing.nickname ● ca.signing.cacertnickname ● ca.signing.certnickname ● ca.signing.nickname ● cloning.signing.nickname
------------------------	---

OCSP Signing Certificate	<ul style="list-style-type: none"> ● ca.ocsp_signing.cacertnickname ● ca.ocsp_signing.certnickname ● ca.cert.ocsp_signing.nickname ● ca.ocsp_signing.nickname ● cloning.ocsp_signing.nickname
Subsystem Certificate	<ul style="list-style-type: none"> ● ca.cert.subsystem.nickname ● ca.subsystem.nickname ● cloning.subsystem.nickname ● pkiremove.cert.subsystem.nickname
Server Certificate	<ul style="list-style-type: none"> ● ca.sslserver.nickname ● ca.cert.sslserver.nickname
Audit Signing Certificate	<ul style="list-style-type: none"> ● ca.audit_signing.nickname ● ca.cert.audit_signing.nickname ● cloning.audit_signing.nickname

Table 14.2. KRA Certificate Nickname Parameters

Transport Certificate	<ul style="list-style-type: none"> ● cloning.transport.nickname ● kra.cert.transport.nickname ● kra.transport.nickname ● tks.kra_transport_cert_nickname <p>Note that this parameter is in the TKS configuration file. This needs changed in the TKS configuration if the <i>KRA</i> transport certificate nickname changes, even if the TKS certificates all stay the same.</p>
Storage Certificate	<ul style="list-style-type: none"> ● cloning.storage.nickname ● kra.storage.nickname ● kra.cert.storage.nickname

Server Certificate	<ul style="list-style-type: none"> ● kra.cert.sslserver.nickname ● kra.sslserver.nickname
Subsystem Certificate	<ul style="list-style-type: none"> ● cloning.subsystem.nickname ● kra.cert.subsystem.nickname ● kra.subsystem.nickname ● pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	<ul style="list-style-type: none"> ● cloning.audit_signing.nickname ● kra.cert.audit_signing.nickname ● kra.audit_signing.nickname

Table 14.3. OCSP Certificate Nickname Parameters

OCSP Signing Certificate	<ul style="list-style-type: none"> ● cloning.signing.nickname ● ocsp.signing.certnickname ● ocsp.signing.cacertnickname ● ocsp.signing.nickname
Server Certificate	<ul style="list-style-type: none"> ● ocsp.cert.sslserver.nickname ● ocsp.sslserver.nickname
Subsystem Certificate	<ul style="list-style-type: none"> ● cloning.subsystem.nickname ● ocsp.subsystem.nickname ● ocsp.cert.subsystem.nickname ● pkiremove.cert.subsystem
Audit Log Signing Certificate	<ul style="list-style-type: none"> ● cloning.audit_signing.nickname ● ocsp.audit_signing.nickname ● ocsp.cert.audit_signing.nickname

Table 14.4. TKS Certificate Nickname Parameters

KRA Transport Certificate ^[a]	<ul style="list-style-type: none"> ● tks.kra_transport_cert_nickname
Server Certificate	<ul style="list-style-type: none"> ● tks.cert.sslserver.nickname ● tks.sslserver.nickname
Subsystem Certificate	<ul style="list-style-type: none"> ● cloning.subsystem.nickname ● tks.cert.subsystem.nickname ● tks.subsystem.nickname ● pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	<ul style="list-style-type: none"> ● cloning.audit_signing.nickname ● tks.audit_signing.nickname ● tks.cert.audit_signing.nickname
<p>[a] This needs changed in the TKS configuration if the <i>KRA</i> transport certificate nickname changes, even if the TKS certificates all stay the same.</p>	

Table 14.5. TPS Nickname Parameters

Server Certificate	<ul style="list-style-type: none"> ● tps.cert.sslserver.nickname
Subsystem Certificate	<ul style="list-style-type: none"> ● tps.cert.subsystem.nickname ● selftests.plugin.TPSValidity.nickname ● selftests.plugin.TPSPresence.nickname ● pkiremove.cert.subsystem.nickname
Audit Log Signing Certificate	<ul style="list-style-type: none"> ● tps.cert.audit_signing.nickname

14.4. 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 14.4.1, "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.

14.4.1. 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 14.4.1.1, "Installing Certificates through the Console"](#)
- [Section 14.4.1.2, "Installing Certificates Using certutil"](#)
- [Section 14.4.1.3, "About CA Certificate Chains"](#)

14.4.1.1. Installing Certificates through the Console

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
NVBAYTAIVTMREwDwYDVQQKEWhvZXRzY2FwZTENMA8GA1UECXMzEUEwDQYJ
KCoZIhdfNAQEBBQ
EXMBUGA1UEAxMOU3Vwcml5SBTAgV0dHkkgZ8wDQYJKoZIhdfNAQEBBQ
ADgY0AMIGJAoGBAMr6eZiPGfjX3uRjgEjmKiqG7SdATYzBcABu1AVyd7
chRFOGD3wNktbf6hRo6EAmM5R1Askzf8AW7LiQZBcrXpc0k4du+2j6xJ
u2MPm8WKuMOTuvzpo+SGXelmHVChEqooCwfdiZywyZNmgaMa2MS6pUkf
QVAgMBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAWIAgD
-----END CERTIFICATE-----
```

- 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.

- 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 14.5, "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.

14.4.1.2. Installing Certificates Using certutil

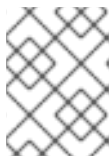
To install subsystem certificates in the Certificate System instance's security databases using **certutil**, do the following:

- Open the subsystem's security database directory.

```
cd /var/lib/pki/instance_name/alias
```

- 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>.

14.4.1.3. 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 TLS-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.

14.4.2. Viewing Database Content

The certificates stored in the subsystem certificates database, **cert8.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 14.4.2.1, “Viewing Database Content through the Console”](#)
- [Section 14.4.2.2, “Viewing Database Content Using certutil”](#)



NOTE

The certificates listed in the **cert8.db** database are the subsystem certificates used for subsystem operations. User certificates are stored with the user entries in the LDAP internal database.

14.4.2.1. Viewing Database Content through the Console

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.

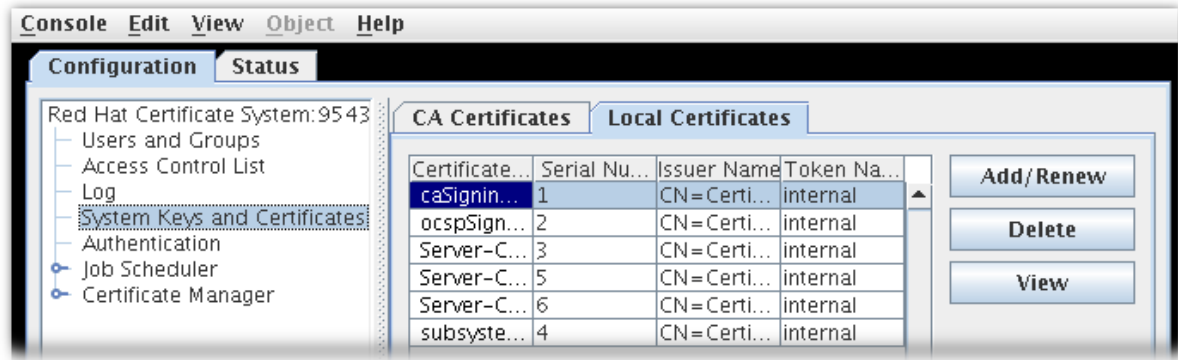


Figure 14.1. Certificate Database Tab

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.

14.4.2.2. 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>.

14.4.3. 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 14.5, “Changing the Trust Settings of a CA Certificate”](#).

- [Section 14.4.3.1, “Deleting Certificates through the Console”](#)
- [Section 14.4.3.2, “Deleting Certificates Using certutil”](#)

14.4.3.1. Deleting Certificates through the Console

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.

14.4.3.2. 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"
```

- 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>.

14.5. CHANGING THE TRUST SETTINGS OF A CA CERTIFICATE

Certificate System subsystems use the CA certificates in their certificate databases to validate certificates received during a TLS-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 14.4.3, "Deleting Certificates from the Database"](#).

14.5.1. Changing Trust Settings through the Console

To change the trust setting of a CA certificate, do the following:

- Open the subsystem console.

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

- In the **Configuration** tab, **System Keys and Certificates** from the left navigation tree.
- Select the **CA certificates** tab.
- Select the CA certificate to modify, and click **Edit**.
- 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.

14.5.2. Changing Trust Settings Using certutil

To change the trust setting of a certificate using **certutil**, do the following:

- Open the instance's certificate databases directory.

```
cd /var/lib/pki/instance_name/alias
```

- 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
```

- 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 .
```

- 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>.

14.6. MANAGING TOKENS USED BY THE SUBSYSTEMS



NOTE

Features in this section on TMS are not tested in the evaluation. This section is for reference only.

Certificate System manages 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.

14.6.1. 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.

14.6.2. 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
```

14.6.3. 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.

CHAPTER 15. SETTING TIME AND DATE IN RED HAT ENTERPRISE LINUX 7.6

The section contains how to set time and date in Red Hat Enterprise Linux 7.6.

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 13.2.1.3. Auditing Time Change Events in Red Hat Certificate System's Planning, Installation, and Deployment Guide.

CHAPTER 16. 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 9.4 (Batch Update 3)
```

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/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 17. UPDATING RED HAT CERTIFICATE SYSTEM

To update Certificate System the operating system is running on, use the **yum update** command. This downloads, verifies, and installs updates to Certificate System and operating system packages. For further information on updating Certificate System and validating that the update was successful, see *Updating Certificate System Packages* section in the *Red Hat Certificate System Planning, Installation, and Deployment Guide (Common Criteria Edition)*.

CHAPTER 18. 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
systemctl restart pki-tomcatd-nuxwdog@instance_name.service
```

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 and usually means 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:6
3)
    at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:5
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 **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.

**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.

For details about installing a VNC server, see the [VNC Server](#) section in the *Red Hat System Administrator's Guide*.

2. Use a VNC viewer to connect to the host running the VNC server. For details, see the [VNC Viewer](#) section in the *Red Hat System Administrator's Guide*.
3. 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 subsystem instances which rely on it. The instance debug logs will specifically identify the problem with the LDAP connection. For example, if the LDAP server was not online:

```
[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 19. 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.

19.1. 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 7.

To start an instance:

```
# systemctl start unit_file@instance_name.service
```

To stop an instance:

```
# systemctl stop unit_file@instance_name.service
```

To restart an instance:

```
# systemctl restart unit_file@instance_name.service
```

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

19.2. 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 [Chapter 10, Self Tests](#).

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.

- [Section A.1, "Input Reference"](#)
- [Section A.2, "Output Reference"](#)

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. 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.1.

```
caCMCUserCert.cfg:input.i1.class_id=cmcCertReqInputImpl
```

A.1.2. nsHKeyCertRequest (Token Key) Input

In Token Management Systems (TMS), 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.2.

```
caTempTokenDeviceKeyEnrollment.cfg:input.i1.class_id=nsHKeyCertReqInputImpl
```

A.1.3. nsNKeyCertRequest (Token User Key) Input

In TMS, 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.3.

```
caTempTokenUserEncryptionKeyEnrollment.cfg:input.i1.class_id=nsNKeyCertReqInputImpl
```

A.1.4. Subject DN Input

In TMS, 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.4.

```
caAdminCert.cfg:input.i3.class_id=subjectDNInputImpl
```

A.1.5. Subject Alternative Name Extension Input

In TMS, 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.5.

```
input.i3.class_id=
input.i3.name=subjectAltNameExtInputImplsubjectAltNameExtInputImpl
...
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=2
```



A.2. OUTPUT REFERENCE

An output is the response to the end user of a successful enrollment.

A.2.1. CMC Certificate Output

The profile framework enables you to specify output plug-ins. In case of certificate enrollments using CMC requests, the output is implicitly set to CMC output in conformance with [RFC 5272](#). Therefore, the output value specified in the profile is ignored.

A.2.2. nsNSKeyOutput

In TMS, this class implements the output plug-in that returns the DER encoded certificates for token keys.

Example A.6. caTokenUserDelegateAuthKeyEnrollment.cfg

```
output.list=o1  
output.o1.class=nsNKeyOutputImpl
```

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 5280](#).

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.1.8, “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_ <i>n</i>	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: <ul style="list-style-type: none">● omsp (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_ <i>n</i>	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: <ul style="list-style-type: none">● DirectoryName● DNSName● EDIPartyName● IPAddress● OID● RFC822Name● URIName

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, *SharedToken*.

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. CMC User-signed Subject Name Default

This profile default populates subject names based on the CMC request signer's **subjectDN**. The following constraint must be used with this default:

- **CMCUserSignedSubjectNameConstraint**: See [Section B.2.14, "CMC User-signed Subject Name Constraint"](#).

B.1.5. 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.
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	<p>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.</p> <p>The maxPathLen parameter has no effect if the extension is set in end-entity certificates.</p> <p>The permissible values are 0 or <i>n</i>. 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. <i>n</i> must be an integer greater than zero. It specifies the maximum number of subordinate CA certificates allowed below the subordinate CA certificate.</p> <p>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.</p>

B.1.6. 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.2.15, "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.1.7. 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.1.8. 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_ <i>n</i>	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_ <i>n</i>	<p>Specifies the name of the CRL distribution point, the name can be in any of the following formats:</p> <ul style="list-style-type: none"> ● An X.500 directory name in the RFC 2253 syntax. The name looks similar to the subject name in a certificate, like <i>cn=CA Central, ou=Research Dept, o=Example Corporation, c=US</i>. ● A URName, such as <i>http://testCA.example.com:80</i>. ● An RDN which specifies a location relative to the CRL issuer. In this case, the value of the Type attribute must be RelativeToIssuer.
Reasons_ <i>n</i>	<p>Specifies revocation reasons covered by the CRL maintained at the distribution point. Provide a comma-separated list of the following constants:</p> <ul style="list-style-type: none"> ● unused ● keyCompromise ● cACompromise ● affiliationChanged ● superseded ● cessationOfOperation ● certificateHold
IssuerType_ <i>n</i>	<p>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:</p> <ul style="list-style-type: none"> ● RFC822Name ● DirectoryName ● DNSName ● EDIPartyName ● URName ● IPAddress ● OIDName ● OtherName
IssuerName_ <i>n</i>	

Parameter	Description
B.1.9. 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, <i>2.16.840.1.113730.1.99</i> . 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.1.10. 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_ <i>n</i>	Select true to enable this point; select false to disable this point.
PointType_ <i>n</i>	Specifies the type of issuing point, either DirectoryName or URIName .

Parameter	Description
PointName_n	<ul style="list-style-type: none"> ● If pointType is set to directoryName, the value must be an X.500 name, similar to the subject name in a certificate. For example, <i>cn=CACentral,ou=Research Dept,o=Example Corporation,c=US</i>. ● If pointType is set to URIName, the name must be a URI, an absolute pathname that specifies the host. For example, <i>http://testCA.example.com/get/crls/here/</i>.
PointIssuerName_n	<p>Specifies the name of the issuer that has signed the CRL. The name can be in any of the following formats:</p> <ul style="list-style-type: none"> ● For RFC822Name, the value must be a valid Internet mail address. For example, <i>testCA@example.com</i>. ● For DirectoryName, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example, <i>cn=SubCA,ou=Research Dept,o=Example Corporation,c=US</i>. ● For DNSName, the value must be a valid fully-qualified domain name. For example, <i>testCA.example.com</i>. ● For EDIPartyName, the value must be an IA5String. For example, <i>Example Corporation</i>. ● 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, <i>http://testCA.example.com</i>. 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 <i>n.n.n.n</i> or <i>n.n.n.n,m.m.m.m</i>. For example, <i>128.21.39.40</i> or <i>128.21.39.40,255.255.255.00</i>. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <i>0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0</i>, and <i>FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000</i>. ● For OIDName, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example,

Parameter	Description
	<p>1.2.3.4.55.6.5.99.</p> <ul style="list-style-type: none"> ● OtherName is used for names with any other format; this supports PrintableString, IA5String, UTF8String, BMPString, Any, and KerberosName. KerberosName has the format <i>Realm/NameType/NameStrings</i>, such as realm1 0 userID1,userID2. <p>OtherName must have the format <i>(type)oid,string</i>. For example, (IA5String)1.2.3.4,MyExample.</p> <p>The name value must comply with the format specified in PointType_.</p>
PointType_ <i>n</i>	<p>Specifies the general name type of the CRL issuer that signed the CRL. The permissible values are as follows:</p> <ul style="list-style-type: none"> ● RFC822Name ● DirectoryName ● DNSName ● EDIPartyName ● URIName ● IPAddress ● OIDName ● OtherName <p>The value for this parameter must correspond to the value in the PointIssuerName field.</p>

B.1.11. 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.1.12. 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	<i>This policy must be marked as critical. Select true to mark this extension critical; select false to mark the extension noncritical.</i>
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.1.13. 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	<p>This sets the type of name extension to be used, which can be one of the following:</p> <ul style="list-style-type: none"> ● RFC822Name ● DirectoryName ● DNSName ● EDIPartyName ● URIName ● IPAddress ● OIDName
issuerAltExtPattern	<p>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.</p> <p>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, <i>Certificate Authority</i>.</p>

B.1.14. 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 TLS client certificates and S/MIME signing certificates. Select true to set.
nonRepudiation	<p>Specifies whether to use for S/MIME signing certificates. Select true to set.</p> <div data-bbox="817 611 1428 1003" style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;">  <p>WARNING</p> <p>Using this bit is controversial. Carefully consider the legal consequences of its use before setting it for any certificate.</p> </div>
keyEncipherment	Specifies whether the public key in the subject is used to encipher private or secret keys. This is set for TLS 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.1.15. 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. <ul style="list-style-type: none"> • -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_ <i>n</i>	<p>Specifies the maximum number of permitted subtrees.</p> <ul style="list-style-type: none">● -1 specifies that the field should not be set in the extension.● 0 specifies that the maximum number of subtrees is zero.● <i>n</i> must be an integer that is greater than zero. It sets the maximum number of subtrees allowed.
PermittedSubtreeNameChoice_ <i>n</i>	<p>Specifies the general name type for the permitted subtree to include in the extension. The permissible values are as follows:</p> <ul style="list-style-type: none">● RFC822Name● DirectoryName● DNSName● EDIPartyName● URIName● IPAddress● OIDName● OtherName

Parameter	Description
PermittedSubtreeNameValue_n	<p>Specifies the general name value for the permitted subtree to include in the extension.</p> <ul style="list-style-type: none"> ● For RFC822Name, the value must be a valid Internet mail address. For example, <i>testCA@example.com</i>. ● For DirectoryName, the value must be a string form of X.500 name, similar to the subject name in a certificate. For example, <i>cn=SubCA, ou=Research Dept, o=Example Corporation, c=US</i>. ● For DNSName, the value must be a valid fully-qualified domain name. For example, <i>testCA.example.com</i>. ● For EDIPartyName, the value must be an IA5String. For example, <i>Example Corporation</i>. ● 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, <i>http://testCA.example.com</i>. 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 <i>n.n.n.n</i> format, or <i>n.n.n.n/m</i> with a netmask - for example, <i>10.34.3.133</i> or <i>110.34.3.133/24</i>. IPv6 addresses must also conform to CIDR notation; examples with netmasks include <i>2620:52:0:2203:527b:9dff:fe56:4495/64</i> or <i>2001:db8::/64</i>. ● For OIDName, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, <i>1.2.3.4.55.6.5.99</i>. ● OtherName is used for names with any other format; this supports PrintableString, IA5String, UTF8String, BMPString, Any, and KerberosName. KerberosName has the format <i>Realm/NameType/NameStrings</i>, such as realm1 0 userID1,userID2. <p>OtherName must have the format <i>(type)oid,string</i>. For example, (IA5String)1.2.3.4,MyExample.</p>
PermittedSubtreeEnable_n	Select true to enable this permitted subtree entry.

Parameter	Description
ExcludedSubtrees n .min	<p>Specifies the minimum number of excluded subtrees.</p> <ul style="list-style-type: none">● -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	<p>Specifies the maximum number of excluded subtrees.</p> <ul style="list-style-type: none">● -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	<p>Specifies the general name type for the excluded subtree to include in the extension. The permissible values are as follows:</p> <ul style="list-style-type: none">● RFC822Name● DirectoryName● DNSName● EDIPartyName● URIName● IPAddress● OIDName● OtherName

Parameter	Description
ExcludedSubtreeNameValue_n	<p>Specifies the general name value for the permitted subtree to include in the extension.</p> <ul style="list-style-type: none"> ● For RFC822Name, the value must be a valid Internet mail address. For example, <i>testCA@example.com</i>. ● For DirectoryName, the value must be an X.500 name, similar to the subject name in a certificate. For example, <i>cn=SubCA, ou=Research Dept, o=Example Corporation, c=US</i>. ● For DNSName, the value must be a valid fully-qualified domain name. For example, <i>testCA.example.com</i>. ● For EDIPartyName, the value must be an IA5String. For example, <i>Example Corporation</i>. ● 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, <i>http://testCA.example.com</i>. 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 <i>n.n.n.n</i> format, or <i>n.n.n.n/m</i> with a netmask - for example, <i>10.34.3.133</i> or <i>110.34.3.133/24</i>. IPv6 addresses must also conform to CIDR notation; examples with netmasks include <i>2620:52:0:2203:527b:9dff:fe56:4495/64</i> or <i>2001:db8::/64</i>. ● For OIDName, the value must be a unique, valid OID specified in dot-separated numeric component notation. For example, <i>1.2.3.4.55.6.5.99</i>. ● For OtherName, the values are names with any other format. This supports PrintableString, IA5String, UTF8String, BMPString, Any, and KerberosName. KerberosName has the format <i>Realm/NameType/NameStrings</i>, such as realm1 0 userID1,userID2. <p>OtherName must have the format <i>(type)oid,string</i>. For example, (IA5String)1.2.3.4,MyExample.</p>
ExcludedSubtreeEnable_n	Select true to enable this excluded subtree entry.

B.1.16. 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 TLS certificate, client TLS certificate, or S/MIME certificate. This restricts the usage of a certificate to predetermined purposes.

B.1.17. 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.1.18. 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.1.19. 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.1.20. 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	<p>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.</p> <ul style="list-style-type: none"> ● -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. ● <i>n</i> 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. <p>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.</p>
inhibitPolicyMapping	<p>Specifies the total number of certificates permitted in the path before policy mapping is no longer permitted.</p> <ul style="list-style-type: none"> ● -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. ● <i>n</i> 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.1.21. 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_ <i>n</i>	Specifies the OID assigned to the policy statement of the issuing CA to map with the policy statement of another CA. For example, <i>1.2.3.4.5</i> .
SubjectDomainPolicy_ <i>n</i>	Specifies the OID assigned to the policy statement of the subject CA that corresponds to the policy statement of the issuing CA. For example, <i>6.7.8.9.10</i> .

B.1.22. 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.1.23. 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: <ul style="list-style-type: none"> • SHA256withRSA • SHA384withRSA • SHA512withRSA • SHA256withEC • SHA384withEC • SHA512withEC

B.1.24. 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

```

policysset.serverCertSet.9.constraint.name=No Constraint
policysset.serverCertSet.9.default.class_id=subjectAltNameExtDefaultImpl
policysset.serverCertSet.9.default.name=Subject Alternative Name Extension Default
policysset.serverCertSet.9.default.params.subjAltExtGNEnable_0=true
policysset.serverCertSet.9.default.params.subjAltExtPattern_0=$request.requester_email$
policysset.serverCertSet.9.default.params.subjAltExtType_0=RFC822Name
policysset.serverCertSet.9.default.params.subjAltExtGNEnable_1=true
policysset.serverCertSet.9.default.params.subjAltExtPattern_1=$request.SAN1$
policysset.serverCertSet.9.default.params.subjAltExtType_1=DNSName
policysset.serverCertSet.9.default.params.subjAltExtGNEnable_2=true
policysset.serverCertSet.9.default.params.subjAltExtPattern_2=http://www.server.example.com
policysset.serverCertSet.9.default.params.subjAltExtType_2=URIName
policysset.serverCertSet.9.default.params.subjAltExtType_3=OtherName
policysset.serverCertSet.9.default.params.subjAltExtPattern_3=(IA5String)1.2.3.4,$server.source$
policysset.serverCertSet.9.default.params.subjAltExtSource_3=UUID4
policysset.serverCertSet.9.default.params.subjAltExtGNEnable_3=true
policysset.serverCertSet.9.default.params.subjAltExtType_4=RFC822Name
policysset.serverCertSet.9.default.params.subjAltExtGNEnable_4=false
policysset.serverCertSet.9.default.params.subjAltExtPattern_4=
policysset.serverCertSet.9.default.params.subjAltNameExtCritical=false
policysset.serverCertSet.9.default.params.subjAltNameNumGNS=4

```

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
<code>\$request.auth_token.cn\$</code>	The LDAP common name (cn) attribute of the user who requested the certificate.
<code>\$request.auth_token.mail\$</code>	The value of the LDAP email (mail) attribute of the user who requested the certificate.
<code>\$request.auth_token.tokenCertSubject\$</code>	The certificate subject name.
<code>\$request.auth_token.uid\$</code>	The LDAP user ID (uid) attribute of the user who requested the certificate.
<code>\$request.auth_token.user\$</code>	
<code>\$request.auth_token.userDN\$</code>	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.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 <i>n.n.n.n</i> or <i>n.n.n.n,m.m.m.m</i> . For example, <i>128.21.39.40</i> or <i>128.21.39.40,255.255.255.00</i> . An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <i>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:FF FF:255.255.255.0</i> , and <i>FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:00 00</i> .
\$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 <i>n.n.n.n</i> or <i>n.n.n.n,m.m.m.m</i> . For example, <i>128.21.39.40</i> or <i>128.21.39.40,255.255.255.00</i> . An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <i>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:FF FF:255.255.255.0</i> , and <i>FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:00 00</i> .
\$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, <i>uid=jsmith, e=jsmith@example.com</i> .
\$request.tokenuid\$	The card unique ID (CUID) of the smart card token used for requesting the enrollment.
\$request.upn\$	The Microsoft UPN. This has the format <i>(UTF8String)1.3.6.1.4.1.311.20.2.3,\$request.upn\$</i> .

Policy Set Token	Description
\$server.source\$	Instructs the server to generate a version 4 UUID (random number) component in the subject name. This always has the format <i>(IA5String)1.2.3.4,\$server.source\$</i> .

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 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 **3** to insert the **RFC822Name**, **DNSName**, and **URIName** names (generic names **_0**, **_1**, and **_2**).

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, <i>\$request.requester_email\$</i> .

Parameter	Description
Type	<p>Specifies the general name type for the request attribute.</p> <ul style="list-style-type: none"> ● Select RFC822Name if the request-attribute value is an email address in the local-part@domain format. For example, <i>jdoe@example.com</i> ● Select DirectoryName if the request-attribute value is an X.500 directory name, similar to the subject name in a certificate. For example, <i>cn=Jane Doe, ou=Sales Dept, o=Example Corporation, c=US</i>. ● Select DNSName if the request-attribute value is a DNS name. For example, <i>corpDirectory.example.com</i>. ● Select EDIPartyName if the request-attribute value is an EDI party name. For example, <i>Example Corporation</i>. ● 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 host. For example, <i>http://hr.example.com</i>. Certificate 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, <i>128.21.39.40</i>. An IPv4 address must be in the format <i>n.n.n.n</i> or <i>n.n.n.n.m.m.m.m</i>. For example, <i>128.21.39.40</i> or <i>128.21.39.40,255.255.255.00</i>. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <i>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</i>, and <i>FF01::43, FFFF:FFFF:FFFF:FFFF: FFFF:FFFF: FF00:0000</i>. ● Select OIDName if the request-attribute value is a unique, valid OID specified in the dot-separated numeric component notation. For example, <i>1.2.3.4.55.6.5.99</i>. ● Select OtherName for names with any other format. This supports PrintableString, IA5String, UTF8String, BMPString, Any, and KerberosName. KerberosName has the format <i>Realm/NameType/NameStrings</i>, such as realm1 0 userID1,userID2. <p>OtherName must have the format <i>(type)oid,string</i>. For example, (IA5String)1.2.3.4,MyExample.</p>

Parameter	Description
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.1.25. 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, <code>\$request.requester_email\$</code> .
Enable	Sets whether that attribute is able to be added to the certificate. Select true to enable the attribute.

B.1.26. 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. <ul style="list-style-type: none"> ● URIName ● Directory name ● DNS Name ● EID Party Name ● IP Address ● OID Name ● RFC822Name
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.1.27. 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.1.28. 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.

```

policysset.userCertSet.1.default.class_id=subjectNameDefaultImpl
policysset.userCertSet.1.default.name=Subject Name Default
policysset.userCertSet.1.default.params.name=$request.auth_token.tokenCertSubject$

```

B.1.29. 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.1.30. 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.1.31. 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.1.32. 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.2.15, "Validity Constraint"](#).
- No Constraints; see [Section B.2.8, "No Constraint"](#).

B.1.33. 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.

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

```

policysset.set1.2.constraint.class_id=extendedKeyUsageExtConstraintImpl
policysset.set1.2.constraint.name=Extended Key Usage Extension
policysset.set1.2.constraint.params.exKeyUsageCritical=false
policysset.set1.2.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.2,1.3.6.1.5.5.7.3.4
policysset.set1.2.default.class_id=userExtensionDefaultImpl
policysset.set1.2.default.name=User Supplied Extension Default
policysset.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 TLS 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 the section called Creating and Editing Certificate Profiles Directly on the File System in Red Hat Certificate System Planning, Installation and Deployment Guide.

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:

```

policysset.serverCertSet.2.constraint.class_id=extendedKeyUsageExtConstraintImpl
policysset.serverCertSet.2.constraint.name=Extended Key Usage Extension
policysset.serverCertSet.2.constraint.params.exKeyUsageCritical=false
policysset.serverCertSet.2.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.2,1.3.6.1.5.5.7.3.4
policysset.serverCertSet.2.default.class_id=userExtensionDefaultImpl
policysset.serverCertSet.2.default.name=User Supplied Extension Default
policysset.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:

```

policysset.serverCertSet.13.constraint.class_id=keyUsageExtConstraintImpl
policysset.serverCertSet.13.constraint.name=Key Usage Extension Constraint
policysset.serverCertSet.13.constraint.params.keyUsageCritical=-
policysset.serverCertSet.13.constraint.params.keyUsageCrlSign=false
policysset.serverCertSet.13.constraint.params.keyUsageDataEncipherment=-
policysset.serverCertSet.13.constraint.params.keyUsageDecipherOnly=-
policysset.serverCertSet.13.constraint.params.keyUsageDigitalSignature=-
policysset.serverCertSet.13.constraint.params.keyUsageEncipherOnly=-
policysset.serverCertSet.13.constraint.params.keyUsageKeyAgreement=true
policysset.serverCertSet.13.constraint.params.keyUsageKeyCertSign=-
policysset.serverCertSet.13.constraint.params.keyUsageKeyEncipherment=-
policysset.serverCertSet.13.constraint.params.keyUsageNonRepudiation=-
policysset.serverCertSet.13.default.class_id=userExtensionDefaultImpl
policysset.serverCertSet.13.default.name=User Supplied Key Usage Extension
policysset.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 5.2.1.2, "Using **certutil** to Create a CSR With User-defined Extensions"](#) .

B.1.34. 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.2.15, "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	<p>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.</p> <p>This parameter has no effect if the extension is set in end-entity certificates.</p> <p>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.</p> <p>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.</p> <p><i>n</i> must be an integer greater than zero. This is the minimum number of subordinate CA certificates allowed below the subordinate CA certificate being used.</p>

Parameter	Description
basicConstraintsMaxPathLen	<p>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.</p> <p>This parameter has no effect if the extension is set in end-entity certificates.</p> <p>The permissible values are 0 or <i>n</i>. The value must be greater than the path length specified in the Basic Constraints extension of the CA signing certificate.</p> <p>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.</p> <p><i>n</i> must be an integer greater than zero. This is the maximum number of subordinate CA certificates allowed below the subordinate CA certificate being used.</p> <p>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.</p>

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.



IMPORTANT

Key Usage extension and Extended Key Usage extension consistency has to be maintained. For further details, see the *Key Usage and Extended Key Usage Consistency* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

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.
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.

Parameter	Description
KeyParameters	<p>Defines the specific key parameters. The parameters which are set for the key differ, depending on the value of the keyType parameter (meaning, depending on the key type).</p> <ul style="list-style-type: none"> 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.




IMPORTANT

Key Usage extension and Extended Key Usage extension consistency has to be maintained. For further details, see the *Key Usage and Extended Key Usage Consistency* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

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 TLS 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.

Parameter	Description
keyUsageNonRepudiation	<p>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.</p> <div data-bbox="815 409 1428 797" style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;">  <p>WARNING</p> <p>Using this bit is controversial. Carefully consider the legal consequences of its use before setting it for any certificate.</p> </div>
keyEncipherment	<p>Specifies whether to set the extension for TLS 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.</p>
keyUsageDataEncipherment	<p>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.</p>
keyUsageKeyAgreement	<p>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.</p>
keyUsageCertsign	<p>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.</p>

Parameter	Description
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, <i>after</i> the certificate expires that it can be submitted for renewal. If the certificate has been expired longer than that time, then the renewal request is rejected. If no value is given, there is no limit.
renewal.graceBefore	Sets the period, in days, <i>before</i> 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
signingAlgsAllowed	<p>Sets the signing algorithms that can be specified to sign the certificate. The algorithms can be any or all of the following:</p> <ul style="list-style-type: none"> ● SHA256withRSA ● SHA384withRSA ● 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 pattern 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:

```

policysset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl
policysset.serverCertSet.1.constraint.name=Subject Name Constraint
policysset.serverCertSet.1.constraint.params.pattern=CN=[^,]+,.+
policysset.serverCertSet.1.constraint.params.accept=true
policysset.serverCertSet.1.default.class_id=subjectNameDefaultImpl
policysset.serverCertSet.1.default.name=Subject Name Default
policysset.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	<p>A request is considered a renewal and is accepted if this parameter is set to true, if a public key is not unique, <i>and</i> 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.</p> <p>When the parameter is set to false, a duplicate public key request will be rejected.</p>

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	<p>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.</p>

B.2.14. CMC User-signed Subject Name Constraint

The CMC user-signed Subject Name Constraint restricts the certificates being issued to the user that signed the CMC request. Use it with **CMCUserSignedSubjectNameDefault**.

B.2.15. 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
-----------	-------------

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	Verifies 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

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 1 (0x1)
  Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
```


Issuer: "CN=CA Signing Certificate,OU=pki-tomcat,O=EXAMPLE"

Validity:

Not Before: Fri Feb 22 19:06:56 2019

Not After : Tue Feb 22 19:06:56 2039

Subject: "CN=CA Signing Certificate,OU=pki-tomcat,O=EXAMPLE"

Subject Public Key Info:

Public Key Algorithm: PKCS #1 RSA Encryption

RSA Public Key:

Modulus:

dd:6d:ad:02:10:43:12:ad:ec:6c:10:82:b3:bc:ec:6d:
4b:e9:46:bc:a3:19:63:15:86:cf:6d:62:43:92:6b:a6:
3d:72:54:4b:4f:d5:ad:a9:1d:76:8d:1c:e9:15:24:10:
a1:03:1e:1b:14:5e:08:0a:0f:5b:02:aa:e9:3f:85:e1:
d4:a6:01:1e:58:ab:7b:f2:67:32:f4:95:3d:35:9c:76:
3a:cb:3b:ef:e3:7d:32:04:bb:35:46:68:bd:21:0c:16:
b6:63:aa:e7:bb:cd:0f:55:66:21:09:e6:a6:f7:4c:fd:
af:c8:a6:d1:98:03:aa:89:b8:76:e7:dd:df:2b:23:c5:
b3:06:16:1d:4a:13:8b:0b:56:0c:d5:a2:9a:22:5e:7d:
08:af:e4:bf:a0:f6:28:ee:ae:0f:2c:b2:4d:2a:09:5b:
6f:32:2e:05:3a:3b:92:5d:d6:1d:69:95:09:0d:f4:b8:
52:ac:48:0f:a8:4f:0a:22:1b:01:4c:d2:79:89:e0:bc:
cd:1c:84:f8:88:e6:92:16:ed:08:ad:6d:9c:17:8d:70:
92:bd:18:74:1a:31:5f:9b:f7:eb:f7:6e:f8:9a:e6:37:
fe:7a:c6:07:9b:8a:6c:e8:5b:77:7c:37:e0:66:39:72:
62:5d:5d:d0:65:a2:d9:b0:7f:d3:ba:ed:4b:42:89:47

Exponent: 65537 (0x10001)

Signed Extensions:

Name: Certificate Authority Key Identifier

Key ID:

88:fb:c7:45:a8:b8:e9:74:ab:71:a2:ab:ce:4e:26:b9:
a5:97:dc:05

Name: Certificate Basic Constraints

Critical: True

Data: Is a CA with no maximum path length.

Name: Certificate Key Usage

Critical: True

Usages: Digital Signature

Non-Repudiation

Certificate Signing

CRL Signing

Name: Certificate Subject Key ID

Data:

88:fb:c7:45:a8:b8:e9:74:ab:71:a2:ab:ce:4e:26:b9:
a5:97:dc:05

Name: Authority Information Access

Method: PKIX Online Certificate Status Protocol

Location:

URI: "http://localhost.localdomain:8080/ca/ocsp"

Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption

Signature:

6b:ed:d8:2b:de:40:a4:14:dd:e8:ce:52:2d:40:0a:f1:

```

88:57:36:3b:7f:c4:e8:77:2b:95:e9:60:fd:57:9b:c2:
2d:17:a2:67:4e:c0:23:00:7a:2c:ef:5f:12:13:05:cc:
9e:d7:4f:70:55:68:88:eb:29:34:94:cd:59:a6:92:31:
c6:36:74:dd:e5:a2:1f:b1:9e:6d:f0:41:95:c2:7f:4c:
38:46:62:d9:f3:27:f4:a3:a7:f3:a2:ba:1c:e5:77:4a:
d3:2d:50:10:47:03:2e:4f:f2:ef:75:92:36:d8:99:6d:
f6:ef:f5:ee:17:70:c2:e0:c1:a1:26:fa:00:e2:ec:35:
d5:11:4d:df:66:8d:3c:84:fa:72:ff:47:a5:95:08:c2:
80:e6:19:60:ab:51:d6:f1:aa:ac:72:77:d0:01:97:1f:
13:f0:c9:55:09:4d:d9:62:5b:bc:4a:21:5a:af:77:cb:
4e:cf:48:aa:3d:fc:f6:5e:c8:e2:e0:e3:58:58:40:39:
2b:9c:15:d3:65:62:d0:96:1b:35:3f:6e:35:96:ae:36:
c2:6c:2b:46:e8:a3:d3:52:21:f0:47:5a:73:5e:1a:b0:
99:2f:5d:1b:bc:a1:81:65:68:16:08:e8:3e:2f:5e:32:
79:ca:8e:25:e5:78:a1:fc:cd:c0:b3:aa:83:02:18:43

```

Fingerprint (SHA-256):

```

2B:2F:05:59:12:F7:A4:6D:DE:22:43:82:59:EC:9F:45:AD:6C:1E:0A:63:6B:79:57:B1:34:3E:1B:BA:D2
:13:AC

```

Fingerprint (SHA1):

```
E1:87:42:85:AF:07:6C:B2:5F:07:CB:50:4D:49:17:AB:43:99:31:F7
```

Mozilla-CA-Policy: false (attribute missing)

Certificate Trust Flags:

SSL Flags:

Valid CA

Trusted CA

Trusted Client CA

Email Flags:

Valid CA

Trusted CA

Object Signing Flags:

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 5280)*, available at [RFC 5280](#). 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-caissuers)** 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 **authorityCertIssuer** 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.



IMPORTANT

Key Usage extension and Extended Key Usage extension consistency has to be maintained. For further details, see the *Key Usage and Extended Key Usage Consistency* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

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
OCSP Signing	1.3.6.1.5.5.7.3.9 ^[a]
<p>^[a] OCSP Signing is not defined in PKIX Part 1, but in RFC 2560, <i>X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP</i>.</p>	

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.



IMPORTANT

Key Usage extension and Extended Key Usage extension consistency has to be maintained. For further details, see the *Key Usage and Extended Key Usage Consistency* section in the *Red Hat Certificate System 9 Planning, Installation and Deployment Guide (Common Criteria Edition)*.

If this extension is included at all, set the bits as follows:

- **digitalSignature (0)** for TLS 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 TLS 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	<ul style="list-style-type: none"> ● keyCertSign ● cRLSign
TLS Client	digitalSignature
TLS Server	keyEncipherment
S/MIME Signing	digitalSignature
S/MIME Encryption	keyEncipherment
Certificate Signing	keyCertSign
Object Signing	digitalSignature

B.3.9. nameConstraints

This extension, which can be 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](http://www.rfc-editor.org/rfc/rfc5280), 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 be 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&crllssuingPoint=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:
 URName: 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.

Certificate 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 Information 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.

Parameter	Description
<code>accessMethodn</code>	The only accepted value for this parameter is <code>calssuers</code> . The <code>calssuers</code> 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.
<code>accessLocationType_n</code>	Specifies the type of access location for the <i>n</i> access description. The options are either DirectoryName or URI .
<code>accessLocation_n</code>	<p>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, <code>CN=CACentral,OU=Research Dept,O=Example Corporation,C=US</code>.</p> <p>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, <code>http://testCA.example.com/get/crls/here/</code>.</p>

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
<code>enable</code>	Specifies whether the rule is enabled or disabled. The default is to have this extension disabled.
<code>critical</code>	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.

Parameter	Description
-----------	-------------

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 <i>n</i>	Specifies the type of issuing point for the <i>n</i> issuing point. For each number specified in numPoints , there is an equal number of pointType parameters. The options are either DirectoryName or URIName .
pointName <i>n</i>	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, <i>CN=CACentral,OU=Research Dept,O=Example Corporation,C=US</i> . 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, <i>http://testCA.example.com/get/crls/here/</i> .

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	<p>Specifies the general-name type; this can be any of the following:</p> <ul style="list-style-type: none"> ● 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 <i>n.n.n.n</i> or <i>n.n.n.n,m.m.m.m</i>. For example, <i>128.21.39.40</i> or <i>128.21.39.40,255.255.255.00</i>. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <i>0:0:0:0:0:13.168.3,FF01::43</i>, <i>0:0:0:0:0:13.168.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0</i>, and <i>FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000</i>. ● 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	<p>Specifies the general-name value; the allowed values depend on the name type specified in the nameType field.</p> <ul style="list-style-type: none"> ● For rfc822Name, the value must be a valid Internet mail address in the <i>local-part@domain</i> format. ● For directoryName, the value must be a string X.500 name, similar to the subject name in a certificate. For example, <i>CN=CACentral,OU=Research Dept,O=Example Corporation,C=US</i>. ● For dNSName, the value must be a valid domain name in the DNS format. For example, <i>testCA.example.com</i>. ● For ediPartyName, the name must be an IA5String. For example, <i>Example Corporation</i>.

Parameter	Description
	<ul style="list-style-type: none"> For URL, the value must be a non-relative URL. For example, <code>http://testCA.example.com</code>. 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 <code>n.n.n.n</code> or <code>n.n.n.n,m.m.m.m</code>. For example, <code>128.21.39.40</code> or <code>128.21.39.40,255.255.255.00</code>. An IPv6 address uses a 128-bit namespace, with the IPv6 address separated by colons and the netmask separated by periods. For example, <code>0:0:0:0:0:0:13.1.68.3,FF01::43</code>, <code>0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0</code>, and <code>FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000</code>. For OID, the value must be a unique, valid OID specified in the dot-separated numeric component notation. For example, <code>1.2.3.4.55.6.5.99</code>. 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 <code>/var/lib/pki/pki-tomcat/ca/othername.txt</code>. KerberosName has the format <code>Realm NameType NameStrings</code>, such as <code>realm1 O userID1,userID2</code>. The name must be the absolute path to the file that contains the general name in its base-64 encoded format. For example, <code>/var/lib/pki/pki-tomcat/ca/extn/ian/othername.txt</code>.

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	<p>Specifies the type of the issuing distribution point from the following:</p> <ul style="list-style-type: none"> ● 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	<p>Gives the name of the issuing distribution point. The name of the distribution point depends on the value specified for the pointType parameter.</p> <ul style="list-style-type: none"> ● For directoryName, the name must be an X.500 name. For example, <i>cn=CRLCentral,ou=Research Dept,o=Example Corporation,c=US</i>. ● For URIName, the name must be a URI that is an absolute pathname and specifies the host. For example, <i>http://testCA.example.com/get/crls/here/</i>. <div style="display: flex; align-items: flex-start; margin-top: 10px;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p>NOTE</p> <p>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.</p> </div> </div>

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 Invalidation 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: TLS Client certificate
- bit 1: TLS Server certificate

- bit 2: S/MIME certificate
- bit 3: Object Signing certificate
- bit 4: reserved
- bit 5: TLS 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

2.16.840.1.113730.1.3

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, “OCSPublisher”](#)

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 7.10, “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 TLS client, S/MIME, TLS 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 TLS 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	<p>Creates a CA's entry, if selected (default).</p> <p>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.</p>

Parameter	Description
dnPattern	<p>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.</p> <p>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.</p> <ul style="list-style-type: none"> ● 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 <p>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.</p>

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	<p>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.</p> <p>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 <i>org</i> and <i>country</i> are replaced with values from the DN in the certificate.</p> <p>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.</p> <p>The permissible values are valid DN components or attributes separated by commas.</p>

Parameter	Description
filterComps	<p>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.</p> <p>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.</p> <p>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.</p>

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

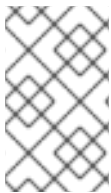
Parameter	Value	Description
type	crl	Specifies the type of certificate that will be published.
predicate		Specifies a predicate for the publisher.
enable	yes	Enables the rule.
mapper	LdapCrlMap	Specifies the mapper used with the rule. See Section C.2.1.2 , “ LdapCrlMap ” for details on the mapper.
publisher	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 ACLs 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* (ACLs) are located in the **acl.ldif** files in the instance's `/var/lib/pki/instance_name/conf` directory.



NOTE

This section provides only a very brief overview of access control concepts. Access control is described in much more detail in *Managing Access Control* 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"**.

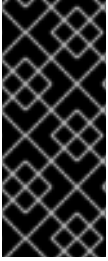
Each area of the **resourceACLs** attribute value is defined in [Table D.1, "Sections of the ACL Attribute Value"](#).

Table D.1. Sections of the ACL Attribute Value

Value	Description
<i>class_name</i>	The plug-in class to which the ACI is applied.
<i>all operations</i>	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.
<i>(operations)</i>	The operations being allowed or denied.
<i>type=target</i>	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" .
<i>description</i>	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.Idif** 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.Idif** 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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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.transactions

Controls who can view the instance's transactions 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.10. certServer.log.content.transactions ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View log content. List all logs.	Allow	Administrators Agents Auditors

D.2.10. certServer.log.content.signedAudit

Controls who has access to the signed audit logs. The default setting is:

```
allow (read) group="Auditors"
```

Table D.11. certServer.log.content.signedAudit ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	View log content. List logs.	Allow	Auditors

D.2.11. 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.12. certServer.registry.configuration ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
------------	-------------	-------------------	-----------------------

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.13. 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.14. 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.15. 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	<div style="border: 1px solid black; padding: 5px;"> Certificate Manager Agents Registration Manager Agents </div>

Operations	Description	Allow/Deny Access	Targeted Users/Groups		
list	List certificates based on a search. Retrieve details about a range of certificates based on a range of serial numbers.	Allow	<table border="1"> <tr> <td>Certificate Manager Agents</td> </tr> <tr> <td>Registration Manager Agents</td> </tr> </table>	Certificate Manager Agents	Registration Manager Agents
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.16. 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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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.17. 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.18. 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.19. 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.20. 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.21. 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.22. 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.23. 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.24. 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.25. 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.26. 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	Submit 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.27. 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.28. 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	<div style="border: 1px solid black; padding: 5px;"> Certificate Manager Agents Registration Manager Agents </div>

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.29. 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.30. 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.31. 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.32. 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.33. 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.34. 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.35. 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.36. certServer.ee.request.ocsp ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
submit	Submit OSCP 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.37. 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.38. 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.39. 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.40. 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.41. 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.42. 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	<table border="1"> <tr> <td>Subsystem Groups</td> </tr> <tr> <td>Enterprise Administrators</td> </tr> </table>	Subsystem Groups	Enterprise Administrators
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.43. 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	<table border="1"> <tr> <td>Administrators</td> </tr> <tr> <td>Agents</td> </tr> <tr> <td>Auditors</td> </tr> </table>	Administrators	Agents	Auditors
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.44. 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.45. 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.46. 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.47. 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.48. 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.49. 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.50. 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.51. 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.52. 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.53. 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.54. 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.55. 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.56. 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.57. 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.58. 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.59. 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.60. 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.61. 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.62. 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.63. 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.64. 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	<table border="1"> <tr> <td>OCSP Agents</td> </tr> <tr> <td>Trusted Managers</td> </tr> </table>	OCSP Agents	Trusted Managers
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.65. 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.66. 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.67. 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.68. 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.69. 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.70. 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.71. 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.72. 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.73. certServer.tks.randomdata ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
Execute	Generate random data.	Allow	TKS Agents

D.7. TPS-SPECIFIC ACLS

This section covers the default access control configuration attributes which are set specifically for the Token Processing System (TPS). The TPS ACL configuration also includes all of the common ACLs listed in [Section D.2, "Common ACLs"](#).

D.7.1. certServer.tps.account

Controls that users can log in and log out.

```
allow (login,logout) user="anybody"
```

Table D.74. certServer.tps.account ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
login	Log into the TPS	Allow	All users
logout	Log out from the TPS	Allow	All users

D.7.2. certServer.tps.authenticators

Controls that only administrators can access authenticators.

```
allow (read,change-status,add,modify,remove) group="Administrators"
```

Table D.75. certServer.tps.authenticators ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read authenticators	Allow	Administrators
change-status	Change status of authenticators	Allow	Administrators
add	Add authenticators	Allow	Administrators
modify	Update authenticators	Allow	Administrators
remove	Remove authenticators	Allow	Administrators

D.7.3. certServer.tps.audit

Controls that only administrators can access the audit configuration.

```
allow (read,modify) group="Administrators"
```

Table D.76. certServer.tps.audit ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read configuration audit settings	Allow	Administrators
modify	Update configuration audit settings	Allow	Administrators

D.7.4. certServer.tps.config

Controls that only administrators can access the configuration.

```
allow (read,modify) group="Administrators"
```

Table D.77. certServer.tps.config ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read configuration settings	Allow	Administrators
modify	Update configuration settings	Allow	Administrators

D.7.5. certServer.tps.connectors

Controls that only administrators can access connectors.

```
allow (read,change-status,add,modify,remove) group="Administrators"
```

Table D.78. certServer.tps.connectors ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read connectors	Allow	Administrators
change-status	Change the status of connectors	Allow	Administrators
add	Add connectors	Allow	Administrators
modify	Update connectors	Allow	Administrators
remove	Remove connectors	Allow	Administrators

D.7.6. certServer.tps.groups

Enables administrators to execute group operations.

```
allow (execute) group="Administrators"
```

Table D.79. certServer.tps.groups ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
execute	Execute group operations	Allow	Administrators

D.7.7. certServer.tps.users

Enables administrators to execute user operations.

```
allow (execute) group="Administrators"
```

Table D.80. certServer.tps.users ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
execute	Execute user operations	Allow	Administrators

D.7.8. certServer.tps.profiles

Allows that administrators and TPS agents can read and change the status of profiles. However, only administrators can add, modify, and remove profiles.

```
allow (read,change-status) group="Administrators" || group="TPS Agents" ; allow
(add,modify,remove) group="Administrators"
```

Table D.81. certServer.tps.profiles ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read profiles	Allow	Administrators, TPS agents
change-status	Change status of profiles	Allow	Administrators, TPS agents
add	Add profiles	Allow	Administrators
modify	Update profiles	Allow	Administrators
remove	Remove profiles	Allow	Administrators

D.7.9. certServer.tps.profile-mappings

Controls that only administrators can access profile mappings.

```
allow (read,change-status,add,modify,remove) group="Administrators"
```

Table D.82. certServer.tps.users ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read profile mappings	Allow	Administrators
change-status	Change status of profile mappings	Allow	Administrators
add	Add profile mappings	Allow	Administrators
modify	Update profile settings	Allow	Administrators
remove	Remove profile settings	Allow	Administrators

D.7.10. certServer.tps.selftests

Controls that only administrators can access self tests.

```
allow (read,execute) group="Administrators"
```

Table D.83. certServer.tps.selftests ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read self tests	Allow	Administrators
execute	Execute self tests	Allow	Administrators

D.7.11. certServer.tps.tokens

Controls that administrators, agents, and operators can read tokens. However, only administrators can add and remove tokens, and only agents can modify tokens.

```
allow (read) group="Administrators" || group="TPS Agents" || group="TPS Operators"; allow
(add,remove) group="Administrators" ; allow (modify) group="TPS Agents"
```

Table D.84. certServer.tps.tokens ACL Summary

Operations	Description	Allow/Deny Access	Targeted Users/Groups
read	Read tokens	Allow	Administrators, TPS agents, TPS operators
add	Add tokens	Allow	Administrators
remove	Remove tokens	Allow	Administrators
modify	Update tokens	Allow	TPS agents

APPENDIX E. AUDIT EVENTS

This appendix contains two parts. The first part, [Section E.1, “Required Audit Events and Their Examples”](#), contains a list of required audit events grouped by the requirement ID from the CA Protection Profile V2.1, where each audit event is accompanied by one or more examples. The second part, [Section E.2, “Audit Event Descriptions”](#) provides individual audit event 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 (Common Criteria Edition)*). For example:

```
[6]
```

- The information about the log event (which is log event specific; see [Section E.2, “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. REQUIRED AUDIT EVENTS AND THEIR EXAMPLES

This section contains all required audit events per Common Criteria CA Protection Profile v.2.1.

For audit events descriptions, see [Section E.2, “Audit Event Descriptions”](#).

FAU_GEN.1

- Start-up of the TSF audit functions

- **AUDIT_LOG_STARTUP**

```
0.localhost-startStop-1 - [21/Jan/2019:17:53:00 IST] [14] [6]
[AuditEvent=AUDIT_LOG_STARTUP][SubjectID=$System$][Outcome=Success] audit function startup
```

- All administrative actions invoked through the TFS interface

- **CONFIG_CERT_PROFILE**

0.http-bio-20443-exec-35 - [02/Jan/2019:05:05:09 EST] [14] [6]
 [AuditEvent=CONFIG_CERT_PROFILE][SubjectID=caadmin][Outcome=Success]
 [ParamNameValPairs=Scope;;rules+Operation;;OP_ADD+Resource;;caAgentExample+class_id;;caEnrollImpl+name;;caAgentExample Enrollment Profile+description;;This certificate profile is for enrolling user certificates+visible;;true] certificate profile configuration parameter(s) change

- **CERT_PROFILE_APPROVAL**

0.http-bio-8443-exec-8 - [15/Nov/2018:15:37:19 PST] [14] [6]
 [AuditEvent=CERT_PROFILE_APPROVAL][SubjectID=cfuEC-0830-agent-2]
 [Outcome=Success][ProfileID=caTPSCert][Op=disapprove] certificate profile approval

- **CONFIG_OCSP_PROFILE**

0.http-bio-22443-exec-11 - [30/Jan/2019:06:18:02 EST] [14] [6]
 [AuditEvent=CONFIG_OCSP_PROFILE][SubjectID=ocspadmin][Outcome=Success]
 [ParamNameValPairs=Scope;;ocspStoresRules+Operation;;OP_MODIFY+Resource;;ldap Store+includeNextUpdate;;false+byName;;true+implName;;com.netscape.cms.ocsp.LDAPStore+numConns;;0+caCertAttr;;cACertificate;binary+notFoundAsGood;;true+crlAttr;;certificateRevocationList;binary] OCSP profile configuration parameter(s) change

- **CONFIG_CRL_PROFILE**

0.http-bio-20443-exec-48 - [29/Jan/2019:04:29:29 EST] [14] [6]
 [AuditEvent=CONFIG_CRL_PROFILE][SubjectID=caadmin][Outcome=Success]
 [ParamNameValPairs=Scope;;crl+Operation;;OP_MODIFY+Resource;;MasterCRL+enable CRLUpdates;;true+updateSchema;;1+extendedNextUpdate;;true+alwaysUpdate;;false+enableDailyUpdates;;true+dailyUpdates;;4:30+enableUpdateInterval;;true+autoUpdateInterval;;240+nextUpdateGracePeriod;;0+nextAsThisUpdateExtension;;0] CRL profile configuration parameter(s) change

- **CONFIG_AUTH**

0.http-bio-20443-exec-11 - [15/Jan/2019:08:36:39 EST] [14] [6]
 [AuditEvent=CONFIG_AUTH][SubjectID=caadmin][Outcome=Success]
 [ParamNameValPairs=Scope;;instance+Operation;;OP_ADD+Resource;;plug502+implName;;UidPwdDirAuth+ldap.ldapconn.host;;server.example.com+dnpattern;;uid=test,ou=people, o=topology-02-CA+ldapStringAttributes;;mail+ldap.ldapconn.version;;3+ldap.ldapconn.port;;3389+ldap.maxConns;;10+ldap.basedn;;dc=example,dc=com+ldap.minConns;;3+ldap.ldapconn.secureConn;;false+ldapByteAttributes;;uid+ldap.password;;(sensitive)+ldap.ldapauth.authtype;;BasicAuth+ldap.ldapauth.bindDN;;cn=directory manager] authentication configuration parameter(s) change

0.http-bio-20080-exec-25 - [29/Jan/2019:04:54:14 EST] [14] [6]
 [AuditEvent=CONFIG_AUTH][SubjectID=caadmin][Outcome=Success]
 [ParamNameValPairs=Scope;;instance+Operation;;OP_ADD+Resource;;plug7487+implName;;AgentCertAuth] authentication configuration parameter(s) change

- **CONFIG_ROLE(success)**

```
0.http-bio-20443-exec-50 - [18/Jan/2019:04:08:45 EST] [14] [6]
[AuditEvent=CONFIG_ROLE][SubjectID=caadmin][Outcome=Success]
[ParamNameValPairs=Scope;;certs+Operation;;OP_ADD+Resource;;CA_AdminV+cert;;-
----BEGIN CERTIFICATE-----MIIDYTCCAkmgAwIBAgIBfz...-----END CERTIFICATE-----
] role configuration parameter(s) change
```

- **CONFIG_ROLE(Failure)**

```
0.http-bio-20443-exec-39 - [18/Jan/2019:04:08:57 EST] [14] [6]
[AuditEvent=CONFIG_ROLE][SubjectID=caadmin][Outcome=Failure]
[ParamNameValPairs=Scope;;users+Operation;;OP_ADD+Resource;;CA_AdminUnTruste
d+password;;*****+phone;;<null>+fullname;;CA_AdminUnTrusted+state;;
<null>+userType;;<null>+email;;<null>] role configuration parameter(s) change
```

- **CONFIG_ACL**

- CA

```
CA = 0.http-bio-20443-exec-18 - [29/Jan/2019:05:15:16 EST] [14] [6]
[AuditEvent=CONFIG_ACL][SubjectID=caadmin][Outcome=Success]
[ParamNameValPairs=Scope;;acls+Operation;;OP_MODIFY+Resource;;testACL+aci;;
allow (read,allow) group="testGroup"+desc;;ALLOW READ to
testGroup+rights;;read,allow] ACL configuration parameter(s) change
```

- **CONFIG_SIGNED_AUDIT**

- CA

```
0.http-bio-20443-exec-20 - [29/Jan/2019:02:44:04 EST] [14] [6]
[AuditEvent=CONFIG_SIGNED_AUDIT][SubjectID=caadmin][Outcome=Success]
[ParamNameValPairs=Action;;disable] signed audit configuration parameter(s)
change
```

- KRA

```
0.http-bio-21443-exec-9 - [30/Jan/2019:08:15:11 EST] [14] [6]
[AuditEvent=CONFIG_SIGNED_AUDIT][SubjectID=kraadmin][Outcome=Success]
[ParamNameValPairs=Action;;enable] signed audit configuration parameter(s)
change
```

- OCSP

```
0.http-bio-22443-exec-17 - [30/Jan/2019:08:17:06 EST] [14] [6]
[AuditEvent=CONFIG_SIGNED_AUDIT][SubjectID=ocspadmin][Outcome=Success]
[ParamNameValPairs=Action;;enable] signed audit configuration parameter(s)
change
```

- TKS

```
0.http-bio-23443-exec-15 - [30/Jan/2019:08:18:52 EST] [14] [6]
[AuditEvent=CONFIG_SIGNED_AUDIT][SubjectID=tkadmin][Outcome=Success]
[ParamNameValPairs=Action;;enable] signed audit configuration parameter(s)
change
```

- TPS

```
0.http-bio-25443-exec-5 - [30/Jan/2019:08:20:03 EST] [14] [6]
[AuditEvent=CONFIG_SIGNED_AUDIT][SubjectID=tpsadmin][Outcome=Success]
[ParamNameValPairs=Action;;enable] signed audit configuration parameter(s)
change
```

- CONFIG_TRUSTED_PUBLIC_KEY

- CA

```
0.http-bio-20443-exec-9 - [29/Jan/2019:03:25:02 EST] [14] [6]
[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY][SubjectID=caadmin]
[Outcome=Success]
[ParamNameValPairs=Scope;;installCert+Operation;;OP_MODIFY+Resource;;trusted
CACert+pkcs10;;-----BEGIN CERTIFICATE-----MIIEBDCCAuygAwI...-----END
CERTIFICATE-----+nickname;;<null>+pathname;;<null>+serverRoot;;
<null>+serverID;;instanceID] certificate database configuration
```

- KRA

```
0.http-bio-21443-exec-17 - [30/Jan/2019:08:29:07 EST] [14] [6]
[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY][SubjectID=kraadmin]
[Outcome=Success]
[ParamNameValPairs=Scope;;installCert+Operation;;OP_MODIFY+Resource;;trusted
CACert+pkcs10;;-----BEGIN CERTIFICATE-----MIIEBDCCAuygAwI...-----END
CERTIFICATE-----+nickname;;<null>+pathname;;<null>+serverRoot;;
<null>+serverID;;instanceID] certificate database configuration
```

- OCSP

```
0.http-bio-22443-exec-25 - [30/Jan/2019:08:41:08 EST] [14] [6]
[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY][SubjectID=ocspadmin]
[Outcome=Success]
[ParamNameValPairs=Scope;;installCert+Operation;;OP_MODIFY+Resource;;trusted
CACert+pkcs10;;-----BEGIN CERTIFICATE-----MIIEBDCCAuygAwIB...-----END
CERTIFICATE-----+nickname;;<null>+pathname;;<null>+serverRoot;;
<null>+serverID;;instanceID] certificate database configuration
```

- TKS

```
0.http-bio-23443-exec-23 - [30/Jan/2019:08:45:40 EST] [14] [6]
[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY][SubjectID=tkadmin]
[Outcome=Success]
[ParamNameValPairs=Scope;;installCert+Operation;;OP_MODIFY+Resource;;trusted
CACert+pkcs10;;-----BEGIN CERTIFICATE-----MIIEBDCCAuygAwIBA...-----END
CERTIFICATE-----+nickname;;<null>+pathname;;<null>+serverRoot;;
<null>+serverID;;instanceID] certificate database configuration
```

- TPS

```
0.http-bio-22443-exec-23 - [30/Jan/2019:08:46:13 EST] [14] [6]
```



```
[AuditEvent=CONFIG_TRUSTED_PUBLIC_KEY][SubjectID=tpsadmin]
[Outcome=Success]
[ParamNameValPairs=Scope;;installCert+Operation;;OP_MODIFY+Resource;;trusted
CACert+pkcs10;;-----BEGIN CERTIFICATE-----MIEBDCCAuygAwIBA...-----END
CERTIFICATE-----+nickname;;<null>+pathname;;<null>+serverRoot;;
<null>+serverID;;instanceID] certificate database configuration
```

- **CONFIG_DRM**

```
0.http-bio-21443-exec-1 - [24/Jan/2019:09:36:52 EST] [14] [6]
[AuditEvent=CONFIG_DRM][SubjectID=kraadmin][Outcome=Success]
[ParamNameValPairs=Scope;;general+Operation;;OP_MODIFY+Resource;;RS_ID_CON
FIG+noOfRequiredRecoveryAgents;;2] DRM configuration parameter(s) change
```

- **OCSP_ADD_CA_REQUEST_PROCESSED**

- Success

```
0.http-bio-22443-exec-24 - [29/Jan/2019:03:15:59 EST] [14] [6]
[AuditEvent=OCSP_ADD_CA_REQUEST_PROCESSED][SubjectID=ocspadmin]
[Outcome=Success][CASubjectDN=CN=CA Signing Certificate,OU=topology-02-
CA,O=topology-02_example.com] Add CA for OCSP Responde
```

- Failure

```
0.http-bio-22443-exec-12 - [30/Jan/2019:06:44:32 EST] [14] [6]
[AuditEvent=OCSP_ADD_CA_REQUEST_PROCESSED][SubjectID=ocspadmin]
[Outcome=Failure][CASubjectDN=<null>] Add CA for OCSP Responder
```

- **OCSP_REMOVE_CA_REQUEST_PROCESSED**

```
0.http-bio-22443-exec-24 - [29/Jan/2019:03:13:43 EST] [14] [6]
[AuditEvent=OCSP_REMOVE_CA_REQUEST_PROCESSED][SubjectID=ocspadmin]
[Outcome=Success][CASubjectDN=CN=CA Signing Certificate,OU=topology-02-
CA,O=topology-02_example.com] Remove CA for OCSP Responder is successful
```

- **SECURITY_DOMAIN_UPDATE**

- Operation: Issue_token

```
0.http-bio-20443-exec-10 - [16/Jan/2019:03:19:57 EST] [14] [6]
[AuditEvent=SECURITY_DOMAIN_UPDATE][SubjectID=caadmin]
[Outcome=Success]
[ParamNameValPairs=operation;;issue_token+token;;2433856184928074456+ip;;192.
0.2.1+uid;;caadmin+groupname;;Enterprise TKS Administrators] security domain
update
```

- Operation: Add

```
0.http-bio-20443-exec-18 - [02/Jan/2019:04:39:21 EST] [14] [6]
[AuditEvent=SECURITY_DOMAIN_UPDATE][SubjectID=caadmin]
[Outcome=Success][ParamNameValPairs=host;;server.example.com+name;;OCSP
```

```
server.example.com 22443+sport;;22443+clone;;false+type;;OCSP+operation;;add]
security domain update
```

- **CONFIG_SERIAL_NUMBER**

- CA

```
0.http-bio-20443-exec-2 - [29/Jan/2019:07:53:21 EST] [14] [6]
[AuditEvent=CONFIG_SERIAL_NUMBER][SubjectID=caadmin][Outcome=Success]
[ParamNameValPairs=source;;updateNumberRange+type;;request+beginNumber;;99
90001+endNumber;;10000000] serial number range update
```

- KRA

```
0.http-bio-21443-exec-7 - [18/Jan/2019:19:11:47 EST] [14] [6]
[AuditEvent=CONFIG_SERIAL_NUMBER][SubjectID=caadmin][Outcome=Success]
[ParamNameValPairs=source;;updateNumberRange+type;;serialNo+beginNumber;;fff
0001+endNumber;;10000000] serial number range update
```

FDP_CER_EXT.1 (extended)

- Certificate generation

- **CERT_REQUEST_PROCESSED (SUCCESS)**

```
0.http-bio-8443-exec-24 - [07/Sep/2018:10:21:57 PDT] [14] [6]
[AuditEvent=CERT_REQUEST_PROCESSED][SubjectID=caadmin][Outcome=Success]
[ReqID=7][CertSerialNum=7] certificate request processed
```

FDP_CER_EXT.2 (extended)

- Linking of certificates to certificate requests

- **PROFILE_CERT_REQUEST**

```
0.http-bio-8443-exec-24 - [07/Sep/2018:10:21:57 PDT] [14] [6]
[AuditEvent=PROFILE_CERT_REQUEST][SubjectID=caadmin][Outcome=Success]
[ReqID=7][ProfileID=caECFullCMCUserCert][CertSubject=CN=cfuEC-0830] certificate
request made with certificate profiles
```



NOTE

The **ReqID** field effectively links to the **ReqID** field of a successful **CERT_REQUEST_PROCESSED** event.

FDP_CER_EXT.3

- Failed certificate approvals

- **CERT_REQUEST_PROCESSED (FAILURE)**

```
0.http-bio-20443-exec-4 - [21/Jan/2019:00:24:16 EST] [14] [6]
[AuditEvent=CERT_REQUEST_PROCESSED][SubjectID=$NonRoleUser$]
```

```
[Outcome=Failure][ReqID=1483][InfoName=rejectReason][InfoValue=Request 1483
Rejected - Subject Name Not Matched
UID=testuser00,E=example@example.com,CN=MyTestUser] certificate request
processed
```

FIA_X509_EXT.1, FIA_X509_EXT.2

- Failed certificate validations; failed authentications

- **ACCESS_SESSION_ESTABLISH (FAILURE)**

- User with revoked cert trying to perform an operation.

```
0.http-bio-21443-exec-9 - [12/Feb/2019:14:52:26 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2]
[SubjectID=UID=KRA_AgentR,E=KRA_AgentR@example.org,CN=KRA_AgentR,OU=I
DMQE,C=US][Outcome=Failure][Info=CERTIFICATE_REVOKED] access session
establish failure
```

- User with expired cert trying to perform an operation.

```
0.http-bio-21443-exec-9 - [12/Feb/2019:14:52:26 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2]
[SubjectID=UID=KRA_AgentR,E=KRA_AgentR@example.org,CN=KRA_AgentR,OU=I
DMQE,C=US][Outcome=Failure][Info=CERTIFICATE_EXPIRED] access session
establish failure
```

- CMC enrollment request submitted using a TLS client cert issued by an unknown CA.

```
0.http-bio-20443-exec-28 - [12/Feb/2019:16:31:08 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2][SubjectID=CN=CA Signing Certificate,OU=pki-
tomcat,O=EXAMPLE][Outcome=Failure][Info=UNKNOWN_CA] access session
establish failure
```

- When client protocol does not match. For example: client use **ssl3** but server does not support.

```
0.http-bio-20443-exec-11 - [12/Feb/2019:16:35:26 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2][SubjectID=][Outcome=Failure][Info=HANDSHAKE_FAILURE]
access session establish failure
```

- For incorrect protocol version. Example server supports **tls1.1** and **tls1.2** but client sends **tls1**.

```
0.http-bio-20443-exec-46 - [12/Feb/2019:16:39:10 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2][SubjectID=][Outcome=Failure][Info=PROTOCOL_VERSION]
access session establish failure
```

- When client sends list of cipher but Server have no list of ciphers.

Server:

```
0.http-bio-21443-exec-3 - [13/Feb/2019:07:40:44 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2][SubjectID=][Outcome=Failure][Info=INTERNAL_ERROR]
access session establish failure
```

FIA_UIA_EXT.1

- Privileged user identification and authentication

- **ACCESS_SESSION_ESTABLISH**

- CA Example

```
0.http-bio-8443-exec-1 - [10/Oct/2018:15:42:13 PDT] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=][Outcome=Success] access session establish
success
```

- TPS Example

```
0.http-bio-25443-exec-1 - [02/Jan/2019:04:44:12 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=][Outcome=Success] access session establish
success
```

- **AUTH**

- CA Example

```
0.http-bio-8443-exec-1 - [28/Nov/2018:16:23:15 PST] [14] [6] [AuditEvent=AUTH]
[SubjectID=caagentJoe][Outcome=Success][AuthMgr=CMCAuth] authentication
success
```

- TPS Example

```
0.http-bio-25443-exec-1 - [25/Jan/2019:13:00:59 IST] [14] [6] [AuditEvent=AUTH]
[SubjectID=tpsadmin][Outcome=Success][AuthMgr=passwdUserDBAuthMgr]
authentication success
```

- **AUTHZ**

- CA Example

```
0.http-bio-8443-exec-1 - [28/Nov/2018:16:23:15 PST] [14] [6] [AuditEvent=AUTHZ]
[SubjectID=caagentJoe][Outcome=Success][aclResource=certServer.ee.profile]
[Op=submit] authorization success
```

- TPS Example

```
0.http-bio-25443-exec-1 - [25/Jan/2019:13:00:59 IST] [14] [6] [AuditEvent=AUTHZ]
[SubjectID=tpsadmin][Outcome=Success][aclResource=certServer.tps.account]
[Op=login][Info=AccountResource.login] authorization success
```

- **ROLE_ASSUME**

- CA Example

```
0.http-bio-8443-exec-1 - [28/Nov/2018:16:23:15 PST] [14] [6]
[AuditEvent=ROLE_ASSUME][SubjectID=caagentJoe][Outcome=Success]
[Role=Certificate Manager Agents] assume privileged role
```

- TPS Example

```
0.http-bio-25443-exec-9 - [25/Jan/2019:13:00:07 IST] [14] [6]
[AuditEvent=ROLE_ASSUME][SubjectID=cfu][Outcome=Success][Role=Certificate
Manager Agents] assume privileged role
```

FMT_SMR.2

- Modifications to the group of users that are part of a role

- **CONFIG_ROLE**

See **CONFIG_ROLE** event above.

FPT_FLS.1

- Failure with preservation of secure state

- **SELFTESTS_EXECUTION**

- CA Example

```
0.localhost-startStop-1 - [10/Jan/2019:00:47:57 EST] [14] [6]
[AuditEvent=SELFTESTS_EXECUTION][SubjectID=$System$][Outcome=Failure]
self tests execution (see selftests.log for details)
```

- TPS Example

```
0.localhost-startStop-1 - [22/Jan/2019:11:55:32 IST] [14] [6]
[AuditEvent=SELFTESTS_EXECUTION][SubjectID=$System$][Outcome=Failure]
self tests execution (see selftests.log for details)
```

FPT_KST_EXT.2

- Private/secret keys are stored by the HSM and the only operations to "access" those keys are through the TSF as signing operations.

```
CERT_REQUEST_PROCESSED (failure)
0.http-bio-20443-exec-8 - [28/Jan/2019:13:48:14 EST] [14] [6]
[AuditEvent=CERT_REQUEST_PROCESSED][SubjectID=$Unidentified$][Outcome=Failure]
[ReqID=28][InfoName=rejectReason][InfoValue=Request Key Type RSA Not Matched
Rejected - {1}] certificate request processed
```

FPT_RCV.1

- The fact that a failure or service discontinuity occurred. Resumption of the regular operation.
 - Failure: **SELFTESTS_EXECUTION (Failure)**
 - CA Example


```
0.localhost-startStop-1 - [29/Jan/2019:13:29:03 UTC] [14] [6]
[AuditEvent=SELFTESTS_EXECUTION][SubjectID=$System$][Outcome=Failure]
self tests execution (see selftests.log for details)
```
 - TPS Example


```
0.localhost-startStop-1 - [22/Jan/2019:11:55:32 IST] [14] [6]
[AuditEvent=SELFTESTS_EXECUTION][SubjectID=$System$][Outcome=Failure]
self tests execution (see selftests.log for details)
```
 - Self-test log, see 13.3.2. Configuring Self-Tests in Red Hat Certificate System's Planning, Installation, and Deployment Guide.
 - Resumption: **AUDIT_LOG_STARTUP; SELFTESTS_EXECUTION (Success)**
 - TPS Example


```
0.localhost-startStop-1 - [21/Jan/2019:16:47:44 IST] [14] [6]
[AuditEvent=AUDIT_LOG_STARTUP][SubjectID=$System$][Outcome=Success]
audit function startup
```
 - CA Example


```
0.localhost-startStop-1 - [04/Feb/2019:18:29:38 EST] [14] [6]
[AuditEvent=SELFTESTS_EXECUTION][SubjectID=$System$][Outcome=Success]
self tests execution (see selftests.log for details)
```

FPT_STM.1

- Changes to the time.

See [Section 13.2.3.3, "Displaying Time Change Events"](#)

FPT_TUD_EXT.1

- Initiation of update.

See [Section 13.2.3.4, "Displaying Package Update Events"](#).

FTA_SSL.4

- The termination of an interactive session.
 - **ACCESS_SESSION_TERMINATED**
 - ```
0.http-bio-20443-exec-7 - [21/Jan/2019:03:42:17 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_TERMINATED][ClientIP=192.0.2.1]
```

```
[ServerIP=192.0.2.1][SubjectID=CN=PKI
Administrator,E=caadmin@example.com,OU=topology-02-CA,O=topology-
02_example.com][Outcome=Success][Info=CLOSE_NOTIFY] access session
terminated
```

- TPS

```
0.http-bio-25443-exec-1 - [02/Jan/2019:04:44:12 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_TERMINATED][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=][Outcome=Success][Info=CLOSE_NOTIFY] access
session
```

## FTP\_TRP.1

- Initiation of the trusted channel. Termination of the trusted channel. Failures of the trusted path functions.

- **ACCESS\_SESSION\_ESTABLISH**

- 2529:0.http-bio-20443-exec-8 - [29/Jan/2019:02:41:10 EST] [14] [6]

```
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=CN=PKI
Administrator,E=tpsadmin@server.example.com,OU=topology-02-TPS,O=topology-
02_example.com][Outcome=Failure][Info=UNKNOWN_CA] access session establish
failure
```

- TPS

```
0.http-bio-25443-exec-4 - [25/Jan/2019:12:58:31 IST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=0:0:0:0:0:0:1]
[ServerIP=0:0:0:0:0:0:1][SubjectID=][Outcome=Failure]
[Info=RECORD_OVERFLOW] access session establish failure
```

- **ACCESS\_SESSION\_TERMINATED**

- 0.http-bio-20443-exec-48 - [29/Jan/2019:04:30:49 EST] [14] [6]

```
[AuditEvent=ACCESS_SESSION_TERMINATED][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=][Outcome=Success][Info=CLOSE_NOTIFY] access
session terminated
```

- TPS

```
TPS=0.http-bio-25443-exec-19 - [25/Jan/2019:12:47:07 IST] [14] [6]
[AuditEvent=ACCESS_SESSION_TERMINATED][ClientIP=192.0.2.1]
[ServerIP=192.0.2.1][SubjectID=][Outcome=Success][Info=CLOSE_NOTIFY] access
session terminated
```

## FCS\_CKM.1 and FCS\_CKM.2

- Not available. There are no TOE-related functions where a TOE subsystem generates (or requests the OE to generate) a non-ephemeral key. All system certificates are generated in the same manner as user keys during the installation, before the TOE is running and, thus, before the it can audit.

## FCS\_CKM\_EXT.4

- Not available

## FCS\_COP.1(2)

- All occurrences of signature generation using a CA signing key.

- **CERT\_SIGNING\_INFO** records CA signing certificate key info at system startup

```
0.authorityMonitor - [03/Jan/2019:02:33:35 EST] [14] [6]
[AuditEvent=CERT_SIGNING_INFO][SubjectID=$System$][Outcome=Success]
[SKI=E3:D2:5B:2A:F5:76:FF:7B:48:CA:94:18:5F:7B:BD:6B:95:FB:8F:30]
[AuthorityID=dbec10a4-1264-4759-96d5-6d2aadbf9d34] certificate signing info
```

- **CERT\_REQUEST\_PROCESSED (success)**

```
0.http-bio-20443-exec-378 - [19/Jan/2019:05:57:39 EST] [14] [6]
[AuditEvent=CERT_REQUEST_PROCESSED][SubjectID=caadmin][Outcome=Success]
[ReqID=1352][CertSerialNum=984] certificate request processed
```

- **OCSP\_SIGNING\_INFO** records OCSP signing certificate key info at system startup

```
0.http-bio-29443-exec-3 - [10/Oct/2018:14:15:24 PDT] [14] [6]
[AuditEvent=OCSP_SIGNING_INFO][SubjectID=$System$][Outcome=Success]
[SKI=71:B1:D0:AE:44:DF:ED:D0:20:15:2B:E3:37:E8:EE:04:EB:D6:F1:44] OCSP signing
info
```

- **OCSP\_GENERATION (success)**

```
0.http-nio-22080-exec-3 - [31/Jan/2019:15:34:47 EST] [14] [6]
[AuditEvent=OCSP_GENERATION][SubjectID=$NonRoleUser$][Outcome=Success]
OCSP response generation
```

- **CRL\_SIGNING\_INFO** records CRL signing certificate key info at system startup

```
0.localhost-startStop-1 - [10/Jan/2019:09:10:27 EST] [14] [6]
[AuditEvent=CRL_SIGNING_INFO][SubjectID=$System$][Outcome=Success]
[SKI=23:98:ED:52:5B:2C:27:C6:FF:7C:34:D1:D5:48:57:E9:B8:D1:4E:95] CRL signing
info
```

- **FULL\_CRL\_GENERATION (success)**

```
0.CRLIssuingPoint-testing123 - [30/Jan/2019:08:35:02 EST] [14] [6]
[AuditEvent=FULL_CRL_GENERATION][SubjectID=$System$][Outcome=Success]
[CRLnum=6] Full CRL generation
```

- **DELTA\_CRL\_GENERATION (success)**

```
0.CRLIssuingPoint-testing123 - [30/Jan/2019:08:35:01 EST] [14] [6]
[AuditEvent=DELTA_CRL_GENERATION][SubjectID=$Unidentified$]
[Outcome=Success][CRLnum=5] Delta CRL generation
```



- Failure in signature generation.
  - **CERT\_REQUEST\_PROCESSED (failure)**

```
0.http-bio-20443-exec-8 - [28/Jan/2019:13:48:14 EST] [14] [6]
[AuditEvent=CERT_REQUEST_PROCESSED][SubjectID=$Unidentified$]
[Outcome=Failure][ReqID=28][InfoName=rejectReason][InfoValue=Request Key Type
RSA Not Matched Rejected - {1}] certificate request processed
```
  - **OCSP\_GENERATION (failure)**

```
0.http-nio-22080-exec-6 - [31/Jan/2019:15:35:38 EST] [14] [6]
[AuditEvent=OCSP_GENERATION][SubjectID=$NonRoleUser$][Outcome=Failure]
[FailureReason=Missing issuer certificate] OCSP response generation
```
  - **FULL\_CRL\_GENERATION (failure)**

### FCS\_HTTPS\_EXT.1 and FCS\_TLSS\_EXT.2

- Failure to establish a HTTPS/TLS session.
  - **ACCESS\_SESSION\_ESTABLISH (Failure)**

```
See FTP_TRP.1
```
- Establishment/termination of a HTTPS/TLS session
  - **ACCESS\_SESSION\_TERMINATED**

```
See FIA_UIA_EXT.1
```

### FCS\_TLSC\_EXT.2

- Failure to establish a TLS session.
  - **CLIENT\_ACCESS\_SESSION\_ESTABLISH (Failure)**

```
0.http-bio-20443-exec-21 - [13/Feb/2019:07:48:08 EST] [14] [6]
[AuditEvent=CLIENT_ACCESS_SESSION_ESTABLISH][ClientHost=192.0.2.1]
[ServerHost=pki1.example.com][ServerPort=21443][SubjectID=SYSTEM]
[Outcome=Failure][Info=send:java.io.IOException: SocketException cannot write on
socket] access session failed to establish when Certificate System acts as client
```

When Server is not reachable by Client and Session ran into failures. In this scenario, CA acts as a client for KRA during Key Archival and KRA is not reachable by CA.

```
0.http-bio-20443-exec-11 - [12/Feb/2019:18:20:03 EST] [14] [6]
[AuditEvent=CLIENT_ACCESS_SESSION_ESTABLISH][ClientHost=192.0.2.1]
[ServerHost=pki1.example.com][ServerPort=21443][SubjectID=SYSTEM]
[Outcome=Failure][Info=send:java.io.IOException: Socket has been closed, and cannot
be reused.] access session failed to establish when Certificate System acts as client
```

When CA's subsystem cert is revoked and it tried to access KRA.

- KRA

```
0.http-bio-21443-exec-3 - [13/Feb/2019:08:15:53 EST] [14] [6]
[AuditEvent=ACCESS_SESSION_ESTABLISH][ClientIP=192.0.2.1]
[ServerIP=192.0.2.2][SubjectID=CN=Subsystem Certificate,OU=topology-02-
CA,O=topology-02_Foobarmaster.org][Outcome=Failure]
[Info=CERTIFICATE_REVOKED] access session establish failure
```

- CA

```
0.http-bio-20443-exec-10 - [13/Feb/2019:08:16:08 EST] [14] [6]
[AuditEvent=CLIENT_ACCESS_SESSION_ESTABLISH][ClientHost=192.0.2.1]
[ServerHost=pki1.example.com][ServerPort=21443][SubjectID=SYSTEM]
[Outcome=Failure][Info=send:java.io.IOException: SocketException cannot write on
socket] access session failed to establish when Certificate System acts as client
```

- Establishment/termination of a TLS session.

- **CLIENT\_ACCESS\_SESSION\_TERMINATED**

```
0.http-bio-8443-exec-6 - [10/Oct/2018:15:10:54 PDT] [14] [6]
[AuditEvent=CLIENT_ACCESS_SESSION_TERMINATED][ClientHost=192.0.2.1]
[ServerHost=192.0.2.1][ServerPort=29443][SubjectID=SYSTEM][Outcome=Success]
[Info=CLOSE_NOTIFY] access session terminated when Certificate System acts as
client
```

## FDP\_CRL\_EXT.1

- Failure to generate a CRL.

- **FULL\_CRL\_GENERATION (failure)**

```
0.http-bio-20444-exec-9 - [01/Feb/2019:15:40:38 EST] [14] [6]
[AuditEvent=FULL_CRL_GENERATION][SubjectID=caadmin][Outcome=Failure]
[FailureReason=Record not found] Full CRL generation
```

## FDP\_OCSPG\_EXT.1

- Failure to generate certificate status information.

- **OCSP\_GENERATION (failure)**

## FIA\_AFL.1

- The reaching of the threshold for the Unsuccessful Authentication Attempts. The action Taken. The re-enablement of disabled non-administrative accounts.

Not available. For password authentication only. Certificate System provides certificate-based authentication only.

## FIA\_CMCS\_EXT.1

- CMC requests (generated or received) containing certificate requests or revocation requests. CMC responses issued.

- **CMC\_SIGNED\_REQUEST\_SIG\_VERIFY**

```
0.http-bio-20080-exec-22 - [24/Jan/2019:08:44:51 EST] [14] [6]
[AuditEvent=CMC_SIGNED_REQUEST_SIG_VERIFY][SubjectID=$NonRoleUser$]
[Outcome=Failure][ReqType=$Unidentified$][CertSubject=$Unidentified$]
[SignerInfo=$Unidentified$] agent signed CMC request signature verification
```

- **CMC\_USER\_SIGNED\_REQUEST\_SIG\_VERIFY**

- Successful request:

```
0.http-bio-20443-exec-1 - [18/Feb/2019:12:07:20 EST] [14] [6]
[AuditEvent=CMC_USER_SIGNED_REQUEST_SIG_VERIFY]
[SubjectID=UID=test10,CN=test10,O=example.org][Outcome=Success]
[ReqType=enrollment][CertSubject=<null>]
[SignerInfo=UID=test10,CN=test10,O=example.org] User signed CMC request
signature verification success
```

- **CMC\_REQUEST\_RECEIVED**

- Successful request:

```
0.http-bio-20443-exec-13 - [29/Jan/2019:04:26:49 EST] [14] [6]
[AuditEvent=CMC_REQUEST_RECEIVED][SubjectID=$Unidentified$]
[Outcome=Success][CMCRequest=MIICoAYJKoZlhv...] CMC request received
```

- Failed request:

```
0.http-bio-20443-exec-14 - [29/Jan/2019:07:15:27 EST] [14] [6]
[AuditEvent=CMC_REQUEST_RECEIVED][SubjectID=$Unidentified$]
[Outcome=Success][CMCRequest=MIGOBgkqhkiG9w...] CMC request received
```

- **PROOF\_OF\_POSSESSION** (Enrollment Event)

```
0.http-bio-20443-exec-13 - [29/Jan/2019:04:26:49 EST] [14] [6]
[AuditEvent=PROOF_OF_POSSESSION][SubjectID=user1a][Outcome=Success]
[Info=method=EnrollProfile: verifyPOP:] proof of possession
```

- **PROFILE\_CERT\_REQUEST** (Enrollment Event)

```
0.http-bio-20443-exec-13 - [29/Jan/2019:04:26:49 EST] [14] [6]
[AuditEvent=PROFILE_CERT_REQUEST][SubjectID=user1a][Outcome=Success]
[ReqID=31][ProfileID=caECFullCMCSharedTokenCert]
[CertSubject=UID=user1a,OU=People,DC=rhel76,DC=test] certificate request made with
certificate profiles
```

- **CERT\_STATUS\_CHANGE\_REQUEST**

- Success:

```
0.http-bio-20443-exec-5 - [05/Feb/2019:05:57:12 EST] [14] [6]
[AuditEvent=CERT_STATUS_CHANGE_REQUEST][SubjectID=caadmin]
[Outcome=Success][ReqID=121][CertSerialNum=0x67][RequestType=on-hold]
```

certificate revocation/unrevocation request made

- Failure:

0.http-bio-20443-exec-13 - [05/Feb/2019:05:58:55 EST] [14] [6]  
 [AuditEvent=CERT\_STATUS\_CHANGE\_REQUEST][SubjectID=caadmin]  
 [Outcome=Failure][ReqID=<null>][CertSerialNum=0x67][RequestType=on-hold]  
 certificate revocation/unrevocation request made

- **CERT\_REQUEST\_PROCESSED**

- Successful request:

0.http-bio-20443-exec-13 - [29/Jan/2019:04:26:49 EST] [14] [6]  
 [AuditEvent=CERT\_REQUEST\_PROCESSED][SubjectID=\$Unidentified\$]  
 [Outcome=Success][ReqID=31][CertSerialNum=20] certificate request processed

- **CERT\_STATUS\_CHANGE\_REQUEST\_PROCESSED**

- Successful request:

0.http-bio-20443-exec-9 - [29/Jan/2019:07:43:36 EST] [14] [6]  
 [AuditEvent=CERT\_STATUS\_CHANGE\_REQUEST\_PROCESSED]  
 [SubjectID=UID=user1a,OU=People,DC=rhel76,DC=test][Outcome=Success]  
 [ReqID=32][CertSerialNum=20][RequestType=revoke]  
 [RevokeReasonNum=Certificate\_Hold][Approval=complete] certificate status change  
 request processed

- Failed request:

- 0.http-bio-20443-exec-14 - [29/Jan/2019:07:15:27 EST] [14] [6]  
 [AuditEvent=CERT\_STATUS\_CHANGE\_REQUEST\_PROCESSED][SubjectID=  
 <null>][Outcome=Failure][ReqID=<null>][CertSerialNum=20]  
 [RequestType=revoke][RevokeReasonNum=Certificate\_Hold]  
 [Approval=rejected][Info=CMCOutputTemplate:  
 SharedSecret.getSharedToken(BigInteger serial): shrTok not found in metaInfo]  
 certificate status change request processed

- 0.http-bio-20443-exec-20 - [29/Jan/2019:07:30:41 EST] [14] [6]  
 [AuditEvent=CERT\_STATUS\_CHANGE\_REQUEST\_PROCESSED]  
 [SubjectID=UID=user1a,OU=People,DC=rhel76,DC=test][Outcome=Failure]  
 [ReqID=<null>][CertSerialNum=20][RequestType=revoke]  
 [RevokeReasonNum=Certificate\_Hold][Approval=rejected][Info= certificate  
 issuer DN and revocation request issuer DN do not match] certificate status  
 change request processed

- 0.http-bio-20443-exec-16 - [29/Jan/2019:07:55:27 EST] [14] [6]  
 [AuditEvent=CERT\_STATUS\_CHANGE\_REQUEST\_PROCESSED][SubjectID=  
 <null>][Outcome=Failure][ReqID=<null>][CertSerialNum=20]  
 [RequestType=revoke][RevokeReasonNum=Certificate\_Hold]  
 [Approval=rejected][Info= shared secret not found] certificate status change  
 request processed

- **CMC\_RESPONSE\_SENT**

- Enrollment

- Successful response

```
0.http-bio-20443-exec-13 - [29/Jan/2019:04:26:49 EST] [14] [6]
[AuditEvent=CMC_RESPONSE_SENT][SubjectID=user1a][Outcome=Success]
[CMCResponse=MIIHTAYJKoZl...] CMC response sent
```

- Revocation

- Successful revocation

```
0.http-bio-20443-exec-9 - [29/Jan/2019:07:43:36 EST] [14] [6]
[AuditEvent=CMC_RESPONSE_SENT][SubjectID=$Unidentified$]
[Outcome=Success][CMCResponse=MIIEgYJKoZl...] CMC response sent
```

- Failed revocation

- Revocation does not happen

```
0.http-bio-20443-exec-20 - [29/Jan/2019:07:30:41 EST] [14] [6]
[AuditEvent=CMC_RESPONSE_SENT][SubjectID=$Unidentified$]
[Outcome=Success][CMCResponse=MIIFDgYJKoZlh...] CMC response sent
```

## FPT\_SKY\_EXT.1(2)/OTH

- **AUTHZ**

- Failure: Agent user attempts to retrieve audit log:

```
0.http-bio-8443-exec-2 - [22/Feb/2019:15:03:38 PST] [14] [6] [AuditEvent=AUTHZ]
[SubjectID=EC-CA-agent-2][Outcome=Failure]
[aclResource=certServer.log.content.signedAudit][Op=read][Info=Authorization Error]
authorization failure
```

- Success: Auditor user retrieved audit log:

```
0.http-bio-8443-exec-13 - [22/Feb/2019:15:25:34 PST] [14] [6] [AuditEvent=AUTHZ]
[SubjectID=EC-CA-auditor][Outcome=Success]
[aclResource=certServer.log.content.signedAudit][Op=read]
[Info=AuditResource.getAuditFile] authorization success
```

## FTP\_ITC.1

- Initiation of the trusted channel. Termination of the trusted channel. Failure of the trusted channel functions.
  - See FCS\_HTTPS\_EXT.1
  - See FCS\_TLSC\_EXT.2

## E.2. AUDIT EVENT DESCRIPTIONS

For required audit events and their examples, see [Section E.1, "Required Audit Events and Their Examples"](#).

```
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 TLS 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 TLS certificate**

A certificate used to identify a client to a server using the TLS protocol. See [Transport Layer Security \(TLS\)](#).

**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 \(CMMF\)](#) .

**Common Criteria**

A certification standard that evaluates computer security, both for software and hardware components. The software or hardware vendor defines the operating environment and specified configuration, identifies any threats, and outlines both the development and deployment processes for the target of evaluation (the thing being evaluated). The Common Criteria certification laboratory then tests the implementation design to look for any vulnerabilities.

**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/itl/fipscurrent.cfm>.

## 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 a TLS 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 [Transport Layer Security \(TLS\)](#) 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 TLS 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 communication based on a shared master key generated by the TKS and the smart card APDUs.

### 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 TLS certificate

A certificate used to identify a server to a client using the [Transport Layer Security \(TLS\)](#) 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-1

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-1](#) 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](#).

### 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.

### TLS

See [Transport Layer Security \(TLS\)](#).

## 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.

### Transport Layer Security (TLS)



A protocol that allows mutual authentication between a client and server and the establishment of an authenticated and encrypted connection. TLS runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols.

### 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

#### adding

extensions

to CRLs, [Setting CRL Extensions](#)

#### administrators

creating, [Creating Users](#)

deleting, [Deleting a Certificate System User](#)

modifying

group membership, [Changing Members in a Group](#)

tools provided

Certificate System console, [Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems](#)

#### agents

creating, [Creating Users](#)

deleting, [Deleting a Certificate System User](#)

modifying

group membership, [Changing Members in a Group](#)

role defined, [Agents](#)

See also Agent Services interface, [Agents](#)

#### archiving

users' private encryption keys, [Setting up Key Archival and Recovery](#)

auditors

creating, [Creating Users](#)

authorityInfoAccess, [authorityInfoAccess](#)

authorityKeyIdentifier, [Setting Restrictions on CA Certificates](#), [authorityKeyIdentifier](#), [authorityKeyIdentifier](#)

## B

base-64 encoded file

viewing content, [Viewing Certificates and CRLs Published to File](#)

basicConstraints, [basicConstraints](#)

## C

CA

configuring ECC signing algorithm, [Setting the Signing Algorithms for Certificates](#)

CA certificate mapper, [LdapCaSimpleMap](#)

CA certificate publisher, [LdapCaCertPublisher](#), [LdapCertificatePairPublisher](#)

CA signing certificate, [CA Signing Key Pair and Certificate](#)

changing trust settings of, [Changing the Trust Settings of a CA Certificate](#)

deleting, [Deleting Certificates from the Database](#)

nickname, [CA Signing Key Pair and Certificate](#)

viewing details of, [Viewing Database Content through the Console](#)

certificate

viewing content, [Viewing Certificates and CRLs Published to File](#)

certificate chains

installing in the certificate database, [Installing Certificates through the Console](#)

why install, [About CA Certificate Chains](#)

certificate database

how to manage, [Managing the Certificate Database](#)

what it contains, [Managing the Certificate Database](#)

where it is maintained, [Managing the Certificate Database](#)

Certificate Manager

administrators

creating, [Creating Users](#)

agents

creating, [Creating Users](#)

---

## key pairs and certificates

CA signing certificate, [CA Signing Key Pair and Certificate](#)

OCSP signing certificate, [OCSP Signing Key Pair and Certificate](#)

subsystem certificate, [Subsystem Certificate](#)

TLS CA signing certificate, [OCSP Signing Key Pair and Certificate](#)

TLS server certificate, [TLS Server Key Pair and Certificate](#)

manual updates to publishing directory, [Updating Certificates and CRLs in a Directory](#)

serial number range, [Changing the Restrictions for CAs on Issuing Certificates](#)

## certificate profiles

signing algorithms, [Setting the Signing Algorithms for Certificates](#)

certificate renewal, [Configuring Profiles to Enable Renewal](#)

## Certificate Setup Wizard

using to install certificate chains, [Installing Certificates through the Console](#)

using to install certificates, [Installing Certificates through the Console](#)

## Certificate System console

Configuration tab, [Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems](#)

managing logs, [Viewing Logs in the Console](#)

Status tab, [Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems](#)

certificateIssuer, [certificateIssuer](#)

certificatePolicies, [certificatePoliciesExt](#)

## certificates

extensions for, [Setting Restrictions on CA Certificates, Defaults, Constraints, and Extensions for Certificates and CRLs](#)

finding, [Performing Revocation as an Agent from the Web UI](#)

installing, [Installing Certificates in the Certificate System Database](#)

publishing to files, [Publishing to Files](#)

publishing to LDAP directory

required schema, [Configuring the LDAP Directory](#)

searching for, [Searching for Certificates \(Advanced\)](#)

signing algorithms, [Setting the Signing Algorithms for Certificates](#)

taking off hold, [Taking Certificates Off Hold](#)

## certutil

requesting certificates, [Creating Certificate Signing Requests](#)

## changing

group members, [Changing Members in a Group](#)

trust settings in certificates, [Changing the Trust Settings of a CA Certificate](#)

why would you change, [Changing the Trust Settings of a CA Certificate](#)

Configuration tab, [Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems](#)

CRL

viewing content, [Viewing Certificates and CRLs Published to File](#)

CRL Distribution Point extension, [CRL Issuing Points](#)

CRL extension modules

CRLReason, [Freshest CRL Extension Default](#)

CRL publisher, [LdapCrIPublisher](#)

CRL signing certificate, [About Revoking Certificates](#)

cRLDistributionPoints, [CRLDistributionPoints](#)

CRLNumber, [CRLNumber](#)

CRLReason, [CRLReason](#)

CRLs

defined, [About Revoking Certificates](#)

entering multiple update times, [Configuring CRLs for Each Issuing Point](#)

entering update period, [Configuring CRLs for Each Issuing Point](#)

extension-specific modules, [About CRL Extensions](#)

extensions for, [Standard X.509 v3 CRL Extensions Reference](#)

issuing or distribution points, [CRL Issuing Points](#)

publishing of, [About Revoking Certificates](#)

publishing to files, [Publishing to Files](#)

publishing to LDAP directory, [Publishing CRLs, LDAP Publishing](#)

required schema, [Configuring the LDAP Directory](#)

supported extensions, [About Revoking Certificates](#)

when automated updates take place, [About Revoking Certificates](#)

when generated, [About Revoking Certificates](#)

who generates it, [About Revoking Certificates](#)

D

deleting

privileged users, [Deleting a Certificate System User](#)

deltaCRLIndicator, [deltaCRLIndicator](#)

DER-encoded file

viewing content, [Viewing Certificates and CRLs Published to File](#)

DN components mapper, [LdapDNCompsMap](#)

downloading certificates, [Installing Certificates in the Certificate System Database](#)

E

ECC

configuring, [Setting the Signing Algorithms for Certificates](#)

requesting, [Creating Certificate Signing Requests](#)

encrypted file system (EFS), [Extended Key Usage Extension Default](#)

end-entity certificate publisher, [LdapUserCertPublisher](#)

end-entity certificates

renewal, [Configuring Profiles to Enable Renewal](#)

Extended Key Usage extension

OIDs for encrypted file system, [Extended Key Usage Extension Default](#)

extensions, [Setting Restrictions on CA Certificates, Defaults, Constraints, and Extensions for Certificates and CRLs](#)

an example, [Standard X.509 v3 Certificate Extension Reference](#)

authorityInfoAccess, [authorityInfoAccess](#)

authorityKeyIdentifier, [Setting Restrictions on CA Certificates](#), [authorityKeyIdentifier](#), [authorityKeyIdentifier](#)

basicConstraints, [basicConstraints](#)

CA certificates and, [Setting Restrictions on CA Certificates](#)

certificateIssuer, [certificateIssuer](#)

certificatePolicies, [certificatePoliciesExt](#)

cRLDistributionPoints, [CRLDistributionPoints](#)

CRLNumber, [CRLNumber](#)

CRLReason, [CRLReason](#)

deltaCRLIndicator, [deltaCRLIndicator](#)

extKeyUsage, [extKeyUsage](#)

invalidityDate, [invalidityDate](#)

issuerAltName, [issuerAltName Extension](#), [issuerAltName](#)

issuingDistributionPoint, [issuingDistributionPoint](#)

keyUsage, [keyUsage](#)

nameConstraints, [nameConstraints](#)

netscape-cert-type, [netscape-cert-type](#)

Netscape-defined, [Netscape-Defined Certificate Extensions Reference](#)

policyConstraints, [policyConstraints](#)

policyMappings, [policyMappings](#)

privateKeyUsagePeriod, [privateKeyUsagePeriod](#)

subjectAltName, [subjectAltName](#)

subjectDirectoryAttributes, [subjectDirectoryAttributes](#)

X.509 certificate, summarized, [Standard X.509 v3 Certificate Extension Reference](#)

X.509 CRL, summarized, [Standard X.509 v3 CRL Extensions Reference](#)

extKeyUsage, [extKeyUsage](#)

## F

file-based publisher, [FileBasedPublisher](#)

## G

groups

changing members, [Changing Members in a Group](#)

## I

installing certificates, [Installing Certificates in the Certificate System Database](#)

invalidityDate, [invalidityDate](#)

issuerAltName, [issuerAltName Extension](#), [issuerAltName](#)

issuingDistributionPoint, [issuingDistributionPoint](#)

## K

key archival

how keys are stored, [Key Archival](#)

reasons to archive, [Key Archival](#)

key recovery, [Key Archival](#)

Key Recovery Authority

administrators

creating, [Creating Users](#)

agents

creating, [Creating Users](#)

key pairs and certificates

list of, [Key Recovery Authority Certificates](#)

storage key pair, [Storage Key Pair](#)

subsystem certificate, [Subsystem Certificate](#)

transport certificate, [Transport Key Pair and Certificate](#)

keyUsage, [keyUsage](#)

## L

LDAP publishing

defined, [LDAP Publishing](#)

manual updates, [Updating Certificates and CRLs in a Directory](#)

when to do, [Manually Updating Certificates in the Directory](#)

who can do this, [Updating Certificates and CRLs in a Directory](#)

logging

managing from Certificate System console, [Viewing Logs in the Console](#)

---

## M

### managing

certificate database, [Managing the Certificate Database](#)

### mappers

created during installation, [Creating Mappers](#), [LdapCaSimpleMap](#), [LdapSimpleMap](#)

### mappers that use

CA certificate, [LdapCaSimpleMap](#)

DN components, [LdapDNCompsMap](#)

### modifying

privileged user's group membership, [Changing Members in a Group](#)

## N

### Name extension modules

Issuer Alternative Name, [Issuer Alternative Name Extension Default](#)

nameConstraints, [nameConstraints](#)

netscape-cert-type, [netscape-cert-type](#)

### nickname

for CA signing certificate, [CA Signing Key Pair and Certificate](#)

for OCSP signing certificate, [OCSP Signing Key Pair and Certificate](#)

for signing certificate, [OCSP Signing Key Pair and Certificate](#)

for subsystem certificate, [Subsystem Certificate](#), [Subsystem Certificate](#), [Subsystem Certificate](#)

for TLS server certificate, [TLS Server Key Pair and Certificate](#), [TLS Server Key Pair and Certificate](#)

for TLS signing certificate, [OCSP Signing Key Pair and Certificate](#)

## O

OCSP publisher, [OCSPPublisher](#)

OCSP signing certificate, [OCSP Signing Key Pair and Certificate](#)

nickname, [OCSP Signing Key Pair and Certificate](#)

### Online Certificate Status Manager

#### administrators

creating, [Creating Users](#)

#### agents

creating, [Creating Users](#)

#### key pairs and certificates

signing certificate, [OCSP Signing Key Pair and Certificate](#)

subsystem certificate, [Subsystem Certificate](#)  
TLS server certificate, [TLS Server Key Pair and Certificate](#)

## P

### plug-in modules

for CRL extensions

CRLReason, [Freshest CRL Extension Default](#)

for publishing

FileBasedPublisher, [FileBasedPublisher](#)

LdapCaCertPublisher, [LdapCaCertPublisher](#), [LdapCertificatePairPublisher](#)

LdapCaSimpleMap, [LdapCaSimpleMap](#)

LdapCrlPublisher, [LdapCrlPublisher](#)

LdapDNCompsMap, [LdapDNCompsMap](#)

LdapUserCertPublisher, [LdapUserCertPublisher](#)

OCSPPublisher, [OCSPPublisher](#)

Issuer Alternative Name, [Issuer Alternative Name Extension Default](#)

policyConstraints, [policyConstraints](#)

policyMappings, [policyMappings](#)

privateKeyUsagePeriod, [privateKeyUsagePeriod](#)

privileged users

deleting, [Deleting a Certificate System User](#)

modifying privileges

group membership, [Changing Members in a Group](#)

types

agents, [Agents](#)

profiles

how profiles work, [The Enrollment Profile](#)

publishers

created during installation, [Configuring LDAP Publishers](#), [LdapCaCertPublisher](#),  
[LdapUserCertPublisher](#), [LdapCertificatePairPublisher](#)

publishers that can publish to

CA's entry in the directory, [LdapCaCertPublisher](#), [LdapCrlPublisher](#),  
[LdapCertificatePairPublisher](#)

files, [FileBasedPublisher](#)

OCSF responder, [OCSFPublisher](#)

users' entries in the directory, [LdapUserCertPublisher](#)



---

**publishing**

- of certificates

  - to files, [Publishing to Files](#)

- of CRLs, [About Revoking Certificates](#)

  - to files, [Publishing to Files](#)

  - to LDAP directory, [Publishing CRLs, LDAP Publishing](#)

- viewing content, [Viewing Certificates and CRLs Published to File](#)

**publishing directory**

- defined, [LDAP Publishing](#)

**R**

- recovering users' private keys, [Key Archival](#)

**registering**

- custom OIDs, [Standard X.509 v3 Certificate Extension Reference](#)

**requesting certificates**

- ECC certificates, [Creating Certificate Signing Requests](#)

- using certutil, [Creating Certificate Signing Requests](#)

**revoking certificates**

- taking certificate off hold, [Taking Certificates Off Hold](#)

**roles**

- agent, [Agents](#)

**RSA**

- configuring, [Setting the Signing Algorithms for Certificates](#)

**S**

- setting CRL extensions, [Setting CRL Extensions](#)

- signing algorithms, [Setting the Signing Algorithms for Certificates](#)

  - ECC certificates, [Setting the Signing Algorithms for Certificates](#)

  - RSA certificates, [Setting the Signing Algorithms for Certificates](#)

- signing certificate, [OCSP Signing Key Pair and Certificate](#)

  - changing trust settings of, [Changing the Trust Settings of a CA Certificate](#)

  - deleting, [Deleting Certificates from the Database](#)

  - nickname, [OCSP Signing Key Pair and Certificate](#)

  - viewing details of, [Viewing Database Content through the Console](#)

- Status tab, [Using pkiconsole for CA, OCSP, KRA, and TKS Subsystems](#)

storage key pair, [Storage Key Pair](#)

subjectAltName, [subjectAltName](#)

subjectDirectoryAttributes, [subjectDirectoryAttributes](#)

subjectKeyIdentifier

subjectKeyIdentifier, [subjectKeyIdentifier](#)

subsystem certificate, [Subsystem Certificate](#), [Subsystem Certificate](#), [Subsystem Certificate](#)

nickname, [Subsystem Certificate](#), [Subsystem Certificate](#), [Subsystem Certificate](#)

## T

TLS CA signing certificate, [OCSP Signing Key Pair and Certificate](#)

nickname, [OCSP Signing Key Pair and Certificate](#)

TLS server certificate, [TLS Server Key Pair and Certificate](#), [TLS Server Key Pair and Certificate](#)

changing trust settings of, [Changing the Trust Settings of a CA Certificate](#)

deleting, [Deleting Certificates from the Database](#)

nickname, [TLS Server Key Pair and Certificate](#), [TLS Server Key Pair and Certificate](#)

viewing details of, [Viewing Database Content through the Console](#)

## Token Key Service

administrators

creating, [Creating Users](#)

agents

creating, [Creating Users](#)

## tokens

changing password of, [Changing a Token's Password](#)

managing, [Managing Tokens Used by the Subsystems](#)

viewing which tokens are installed, [Viewing Tokens](#)

transport certificate, [Transport Key Pair and Certificate](#)

changing trust settings of, [Changing the Trust Settings of a CA Certificate](#)

deleting, [Deleting Certificates from the Database](#)

viewing details of, [Viewing Database Content through the Console](#)

when used, [Key Archival](#)

## trusted managers

deleting, [Deleting a Certificate System User](#)

modifying

group membership, [Changing Members in a Group](#)

## U

users

creating, [Creating Users](#)

## 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 9.4-1** **Thu Feb 11, 2021**  
Various minor corrections for 9.4 Common Criteria Maintenance Update.

**Revision 9.4-0** **Wed Apr 10, 2019**  
Initial version of this guide.