Red Hat Certificate System 9

Planning, Installation, and Deployment Guide (Common Criteria Edition)

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

Red Hat Certificate System 9 Planning, Installation, and Deployment Guide (Common Criteria Edition)

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

Legal Notice

Abstract

This guide contains documentation about understanding, installing, and configuring Red Hat Certificate System 9.4 in a Common Criteria environment.

Table of Contents

PART I. INTRODUCTION TO RED HAT CERTIFICATE SYSTEM AND DEPLOYMENT PLANNING

This section provides an overview of Certificate System, including general PKI principles and specific features of Certificate System and its subsystems. Planning a deployment is vital to designing a PKI infrastructure that adequately meets the needs of your organization.

CHAPTER 1. ABOUT THIS GUIDANCE DOCUMENT

This guide contains documentation about understanding, installing, and configuring Red Hat Certificate System. It is structured into the following parts:

Part I, "Introduction to Red Hat Certificate System and Deployment Planning"

Part II, "Installing Red Hat Certificate System"

Part III, "Configuring Certificate System"

Part IV, "Uninstalling Certificate System Subsystems"

Glossary

> **NOTE**
>
> Administrators not familiar with Red Hat Certificate System are strongly encouraged to read Part I, "Introduction to Red Hat Certificate System and Deployment Planning" for a good understanding of RHCS and plan ahead accordingly before following Part II for installation.
>
> Chapter 2, *Introduction to Red Hat Certificate System* provides a topical overview of several different parts of Certificate System. Chapter 4, *Supported Platforms* lists various components and their supported version by Red Hat. Chapter 5, *Planning the Certificate System* contains helpful information for planning a Red Hat Certificate System installation.
>
> Refer to Chapter 3, *Allowed Standards and Protocols* for a list of compliant and supported standards and protocols.

To install Red Hat Certificate System in a compliant manner, follow the instructions in Part II, "Installing Red Hat Certificate System". Begin with Chapter 6, *Prerequisites and Preparation for Installation* to prepare the base operating system. Follow Section 7.3, "Installing Using the **pkispawn** Utility". Afterwards, follow the required post-installation steps in Section 7.4, "Post-installation Tasks" to ensure the complete installation is compliant. This last section links you to relevant parts of the guide which explains the steps to perform for full compliance.

> **IMPORTANT**
>
> If installation fails during the **pkispawn** phase (discussed in Section 7.3, "Installing Using the **pkispawn** Utility"), it is suggested to check your configuration carefully for mistakes and refer to the error logs. Prior to re-running the **pkispawn** utility to retry the installation, it is necessary to fully remove the old instance. Refer to Chapter 16, *Removing a Subsystem* for information about removing subsystems.

For a more complete list of configuration options which are supported and compliant, refer to Part III, "Configuring Certificate System". This part can only be followed after the installation is complete.

CHAPTER 2. INTRODUCTION TO RED HAT CERTIFICATE SYSTEM

> **NOTE**
>
> This chapter is an overview of the system. For details on evaluated product features, please see NIAP Product Compliant List at https://www.niap-ccevs.org/Product/PCL.cfm.

Every common PKI operation, such as 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 is described in this chapter.

2.1. A REVIEW OF CERTIFICATE SYSTEM SUBSYSTEMS

Red Hat Certificate System provides five different subsystems, each focusing on different aspects of a PKI deployment:

- A *certificate authority* called *Certificate Manager*. The CA is the core of the PKI; it issues and revokes all certificates. The Certificate Manager is also the core of the Certificate System. By establishing a *security domain* of trusted subsystems, it establishes and manages relationships between the other subsystems.

- A *key recovery authority* (KRA). Certificates are created based on a specific and unique key pair. If a private key is ever lost, then the data which that key was used to access (such as encrypted emails) is also lost because it is inaccessible. The KRA stores key pairs, so that a new, identical certificate can be generated based on recovered keys, and all of the encrypted data can be accessed even after a private key is lost or damaged.

  > **NOTE**
  >
  > In previous versions of Certificate System, KRA was also referred to as the data recovery manager (DRM). Some code, configuration file entries, web panels, and other resources might still use the term DRM instead of KRA.

- An *online certificate status protocol* (OCSP) responder. The OCSP verifies whether a certificate is valid and not expired. This function can also be done by the CA, which has an internal OCSP service, but using an external OCSP responder lowers the load of the issuing CA.

- A *token key service* (TKS). The TKS derives keys based on the token CCID, private information, and a defined algorithm. These derived keys are used by the TPS to format tokens and enroll certificates on the token.

- A *token processing system* (TPS). The TPS interacts directly with external tokens, like smart cards, and manages the keys and certificates on those tokens through a local client, the Enterprise Security Client (ESC). The ESC contacts the TPS when there is a token operation, and the TPS interacts with the CA, KRA, or TKS, as required, then send the information back to the token by way of the Enterprise Security Client.

Even with all possible subsystems installed, the core of the Certificate System is still the CA (or CAs), since they ultimately process all certificate-related requests. The other subsystems connect to the CA or CAs likes spokes in a wheel. These subsystems work together, in tandem, to create a public key

infrastructure (PKI). Depending on what subsystems are installed, a PKI can function in one (or both) of two ways:

- A *token management system* or *TMS* environment, which manages smart cards. This requires a CA, TKS, and TPS, with an optional KRA for server-side key generation.

- A traditional *non token management system* or *non-TMS* environment, which manages certificates used in an environment other than smart cards, usually in software databases. At a minimum, a non-TMS requires only a CA, but a non-TMS environment can use OCSP responders and KRA instances as well.

2.2. OVERVIEW OF CERTIFICATE SYSTEM SUBSYSTEMS

2.2.1. Separate versus Shared Instances

Red Hat Certificate System supports deployment of separate PKI instances for all subsystems:

- Separate PKI instances run as a single Java-based Apache Tomcat instance.

- Separate PKI instances contain a single PKI subsystem (CA, KRA, OCSP, TKS, or TPS).

- Separate PKI instances must utilize unique ports if co-located on the same physical machine or virtual machine (VM).

Alternatively, Certificate System supports deployment of a shared PKI instance:

- Shared PKI instances also run as a single Java-based Apache Tomcat instance.

- Shared PKI instances that contain a single PKI subsystem are identical to a separate PKI instance.

- Shared PKI instances may contain any combination of up to one of each type of PKI subsystem:

    - CA only

    - TKS only

    - CA and KRA

    - CA and OCSP

    - TKS and TPS

    - CA, KRA, TKS, and TPS

    - CA, KRA, OCSP, TKS, and TPS

    - etc.

- Shared PKI instances allow all of their subsystems contained within that instance to share the same ports.

- Shared PKI instances must utilize unique ports if more than one is co-located on the same physical machine or VM.

2.2.2. Instance Installation Prerequisites

2.2.2.1. Directory Server Instance Availability

Prior to installation of a Certificate System instance, a local or remote Red Hat Directory Server LDAP instance must be available. For instructions on installing Red Hat Directory Server, see the *Red Hat Directory Server Installation Guide*.

2.2.2.2. PKI Packages

Red Hat Certificate System is composed of packages listed below:

- The following base packages form the core of Certificate System, and are available in base Red Hat Enterprise Linux repositories:

    - pki-core.el7

        - pki-base

        - pki-base-java

        - pki-ca

        - pki-javadoc

        - pki-kra

        - pki-server

        - pki-symkey

        - pki-tools

- The packages listed below are *not* available in the base Red Hat Enterprise Linux subscription channel. To install these packages, you must first use **Subscription Manager** to attach the Red Hat Certificate System subscription pool, and enable the RHCS9 repository. See the Subscription Manager chapter of the Red Hat Enterprise Linux 7 System Administrator's Guide for instructions.

    - pki-console.el7pki

        - pki-console

    - pki-core.el7pki

        - pki-ocsp

        - pki-tks

        - pki-tps

    - redhat-pki.el7pki

        - redhat-pki

    - redhat-pki-theme.el7pki

        - redhat-pki-console-theme

        - redhat-pki-server-theme

Use a Red Hat Enterprise Linux 7.5 or later system that has been configured with a supported Hardware Security Module listed in Chapter 4, *Supported Platforms*, and make sure that all packages are up to date before installing Red Hat Certificate System.

To install all Certificate System packages (with the exception of pki-javadoc), use use **Yum** to install the redhat-pki metapackage:

```
# yum install redhat-pki
```

Alternatively, install one or more of the top level PKI subsystem packages as required; see the list above for exact package names. If you use this approach, make sure to also install the redhat-pki-server-theme package, and optionally redhat-pki-console-theme and pki-console to use the PKI Console.

Finally, developers and administrators may also want to install the JSS and PKI javadocs (the jss-javadoc and pki-javadoc).
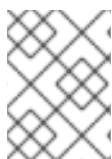
> **NOTE**
>
> The jss-javadoc package requires you to enable the Server-Optional repository in **Subscription Manager**.

2.2.2.3. Instance Installation and Configuration

The **pkispawn** command line tool is used to install and configure a new PKI instance. It eliminates the need for separate installation and configuration steps, and should be run as a batch process. The utility does not provide a way to install or configure the browser-based graphical interface.

For usage information, use the **pkispawn --help** command.

The **pkispawn** command:

1. Reads in its default **name=value** pairs from a plain text configuration file (**/etc/pki/default.cfg**).

2. Automatically overrides any pairs as specified and stores the final result as a Python dictionary.

3. Executes an ordered series of *scriptlets* to perform subsystem and instance installation.

4. The configuration scriptlet packages the Python dictionary as a JavaScript Object Notation (JSON) data object, which is then passed to the Java-based configuration servlet.

5. The configuration servlet utilizes this data to configure a new PKI subsystem, and then passes control back to the **pkispawn** executable, which finalizes the PKI setup. A copy of the final deployment file is stored in **/var/lib/pki/*instance_name*/*<subsystem>*/registry/*<subsystem>*/deployment.cfg**

See the **pkispawn** man page for additional information.

The default configuration file, **/etc/pki/default.cfg**, is a plain text file containing the default installation and configuration values which are read at the beginning of the process described above. It consists of **name=value** pairs divided into **[DEFAULT]**, **[Tomcat]**, **[CA]**, **[KRA]**, **[OCSP]**, **[TKS]**, and **[TPS]** sections.

If you use the **-s** option with **pkispawn** and specify a subsystem name, then only the section for that subsystem will be read.

The sections have a hierarchy: a **name=value** pair specified in a subsystem section will override the pair in the **[Tomcat]** section, which in turn override the pair in the **[DEFAULT]** section. Default pairs can further be overridden by pairs in a specified PKI instance configuration file.

> **NOTE**
>
> Whenever **pkispawn** configuration files are used to override default **name=value** pairs, they may be stored in any location and specified at any time. These files are referred to as **myconfig.txt** in the **pkispawn** man pages, but they are also often referred to as **.ini** files, or more generally as PKI instance configuration override files.

See the **pki_default.cfg** man page for more information.

The *Configuration Servlet* consists of Java bytecode stored in **/usr/share/java/pki/pki-certsrv.jar** as **com/netscape/certsrv/system/ConfigurationRequest.class**. The servlet processes data passed in as a JSON object from the configuration scriptlet using **pkispawn**, and then returns to **pkispawn** using Java bytecode served in the same file as **com/netscape/certsrv/system/ConfigurationResponse.class**.

An example of an interactive installation only involves running the **pkispawn** command on a command line as **root**:

```
# pkispawn
```

> **IMPORTANT**
>
> The interactive installation method mentioned in the pkispawn(8) man page is not supported in Common Criteria environments.

A non-interactive installation requires a PKI instance configuration override file, and the process may look similar to the following example:

1.
   ```
   # mkdir -p /root/pki
   ```

2. Use a text editor such as **vim** to create a configuration file named **/root/pki/ca.cfg** with the following contents:

   ```
   [DEFAULT]
   pki_admin_password=<password>
   pki_client_pkcs12_password=<password>
   pki_ds_password=<password>
   ```

3.
   ```
   # pkispawn -s CA -f /root/pki/ca.cfg
   ```

See the **pkispawn** man page for various configuration examples.

2.2.2.4. Instance Removal

To remove an existing PKI instance, use the **pkidestroy** command. It can be run interactively or as a batch process. Use **pkidestroy -h** to display detailed usage inforamtion on the command line.

The **pkidestroy** command reads in a PKI subsystem deployment configuration file which was stored when the subsystem was created

(**/var/lib/pki/***instance_name*/**<subsystem>**/**registry**/**<subsystem>**/**deployment.cfg**), uses the read-in file in order to remove the PKI subsystem, and then removes the PKI instance if it contains no additional subsystems. See the **pkidestroy** man page for more information.

An interactive removal procedure using **pkidestroy** may look similar to the following:

```
# pkidestroy
Subsystem (CA/KRA/OCSP/TKS/TPS) [CA]:
Instance [pki-tomcat]:

Begin uninstallation (Yes/No/Quit)? Yes

Log file: /var/log/pki/pki-ca-destroy.20150928183547.log
Loading deployment configuration from /var/lib/pki/pki-tomcat/ca/registry/ca/deployment.cfg.
Uninstalling CA from /var/lib/pki/pki-tomcat.
rm '/etc/systemd/system/multi-user.target.wants/pki-tomcatd.target'

Uninstallation complete.
```

A non-interactive removal procedure may look similar to the following example:

```
# pkidestroy -s CA -i pki-tomcat
Log file: /var/log/pki/pki-ca-destroy.20150928183159.log
Loading deployment configuration from /var/lib/pki/pki-tomcat/ca/registry/ca/deployment.cfg.
Uninstalling CA from /var/lib/pki/pki-tomcat.
rm '/etc/systemd/system/multi-user.target.wants/pki-tomcatd.target'

Uninstallation complete.
```

2.2.3. Execution Management (systemctl)

2.2.3.1. Starting, Stopping, Restarting, and Obtaining Status

Red Hat Certificate System subsystem instances can be stopped and started using the **systemctl** execution management system tool on Red Hat Enterprise Linux 7:

```
# systemctl start <unit-file>@instance_name.service
```

```
# systemctl status <unit-file>@instance_name.service
```

```
# systemctl stop <unit-file>@instance_name.service
```

```
# systemctl restart <unit-file>@instance_name.service
```

<unit-file> has one of the following values:

```
pki-tomcatd      With watchdog disabled
pki-tomcatd-nuxwdog   With watchdog enabled
```

For more details on the **watchdog** service, refer Section 2.3.11, "Passwords and Watchdog (nuxwdog)" and Section 9.3.2, "Using the Certificate System Watchdog Service" .

2.2.3.2. Starting the Instance Automatically

The **systemctl** utility in Red Hat Enterprise Linux 7 manages the automatic startup and shutdown settings for each process on the server. This means that when a system reboots, some services can be automatically restarted. System unit files control service startup to ensure that services are started in the correct order. The **systemd** service and **systemctl** utility are described in the *Red Hat Enterprise Linux System Administrator's Guide*

Certificate System instances can be managed by **systemctl**, so this utility can set whether to restart instances automatically. After a Certificate System instance is created, it is enabled on boot. This can be changed by using **systemctl**:

> # systemctl disable pki-tomcatd@*instance_name*.service

To re-enable the instance:

> # systemctl enable pki-tomcatd@*instance_name*.service

> **NOTE**
>
> The **systemctl enable** and **systemctl disable** commands do not immediately start or stop Certificate System.

2.2.4. Process Management (pki-server and pkidaemon)

2.2.4.1. The pki-server Command Line Tool

The primary process management tool for Red Hat Certificate System is **pki-server**. Use the **pki-server --help** command and see the **pki-server** man page for usage information.

The **pki-server** command-line interface (CLI) manages local server instances (for example server configuration or system certificates). Invoke the CLI as follows:

> $ pki-server [CLI options] <command> [command parameters]

The CLI uses the configuration files and NSS database of the server instance, therefore the CLI does not require any prior initialization. Since the CLI accesses the files directly, it can only be executed by the root user, and it does not require client certificate. Also, the CLI can run regardless of the status of the server; it does not require a running server.

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

> $ pki-server

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

> $ pki-server ca
> $ pki-server ca-audit

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

```
$ pki-server --help
$ pki-server ca-audit-event-find --help
```

2.2.4.2. Enabling and Disabling an Installed Subsystem Using pki-server

To enable or disable an installed subsystem, use the **pki-server** utility.

```
# pki-server subsystem-disable -i instance_id subsystem_id
```

```
# pki-server subsystem-enable -i instance_id subsystem_id
```

Replace *subsystem_id* with a valid subsystem identifier: **ca**, **kra**, **tks**, **ocsp**, or **tps**.

> **NOTE**
>
> One instance can have only one of each type of subsystem.

For example, to disable the OCSP subsystem on an instance named **pki-tomcat**:

```
# pki-server subsystem-disable -i pki-tomcat ocsp
```

To list the installed subsystems for an instance:

```
# pki-server subsystem-find -i instance_id
```

To show the status of a particular subsystem:

```
# pki-server subsystem-find -i instance_id subsystem_id
```

2.2.4.3. The pkidaemon Command Line Tool

Another process management tool for Red Hat Certificate System is the **pkidaemon** tool:

```
pkidaemon {start|status} instance-type [instance_name]
```

- **pkidaemon status tomcat** – Provides status information such as on/off, ports, URLs of each PKI subsystem of all PKI instances on the system.

- **pkidaemon status tomcat** *instance_name* – Provides status information such as on/off, ports, URLs of each PKI subsystem of a specific instance.

- **pkidaemon start tomcat** *instance_name*.**service** – Used internally using **systemctl**.

See the **pkidaemon** man page for additional information.

2.2.4.4. Finding the Subsystem Web Services URLs

The CA, KRA, OCSP, TKS, and TPS subsystems have web services pages for agents, as well as regular users and administrators, when appropriate. These web services can be accessed by opening the URL to the subsystem host over the subsystem's secure end user's port. For example, for the CA:

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

> **NOTE**
>
> To get a complete list of all of the interfaces, URLs, and ports for an instance, check the status of the service. For example:
>
> > pkidaemon status *instance_name*

The main web services page for each subsystem has a list of available services pages; these are summarized in Table 2.1, "Default Web Services Pages". To access any service specifically, access the appropriate port and append the appropriate directory to the URL. For example, to access the CA's end entities (regular users) web services:

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

If DNS is not configured, then an IPv4 or IPv6 address can be used to connect to the services pages. For example:

> https://192.0.2.1:8443/ca/services
> https://[2001:DB8::1111]:8443/ca/services

> **NOTE**
>
> Anyone can access the end user pages for a subsystem. However, accessing agent or admin web services pages requires that an agent or administrator certificate be issued and installed in the web browser. Otherwise, authentication to the web services fails.

Table 2.1. Default Web Services Pages

Port	Used for TLS	Used for Client Authentication[a]	Web Services	Web Service Location
Certificate Manager				
8080	No		End Entities	ca/ee/ca
8443	Yes	No	End Entities	ca/ee/ca
8443	Yes	Yes	Agents	ca/agent/ca
8443	Yes	No	Services	ca/services
8443	Yes	No	Console	pkiconsole https://*host:port*/ca

Port	Used for TLS	Used for Client Authentication[a]	Web Services	Web Service Location
Key Recovery Authority				
8080	No		End Entities[b]	kra/ee/kra
8443	Yes	No	End Entities[b]	kra/ee/kra
8443	Yes	Yes	Agents	kra/agent/kra
8443	Yes	No	Services	kra/services
8443	Yes	No	Console	pkiconsole https://*host:port*/kra
Online Certificate Status Manager				
8080	No		End Entities[c]	ocsp/ee/ocsp
8443	Yes	No	End Entities[c]	ocsp/ee/ocsp
8443	Yes	Yes	Agents	ocsp/agent/ocsp
8443	Yes	No	Services	ocsp/services
8443	Yes	No	Console	pkiconsole https://*host:port*/ocsp
Token Key Service				
8080	No		End Entities[b]	tks/ee/tks
8443	Yes	No	End Entities[b]	tks/ee/tks
8443	Yes	Yes	Agents	tks/agent/tks
8443	Yes	No	Services	tks/services
8443	Yes	No	Console	pkiconsole https://*host:port*/tks

Port	Used for TLS	Used for Client Authentication[a]	Web Services	Web Service Location
Token Processing System				
8080	No		Unsecure Services	tps/tps
8443	Yes		Secure Services	tps/tps
8080	No		Enterprise Security Client Phone Home	tps/phoneHome
8443	Yes		Enterprise Security Client Phone Home	tps/phoneHome
8443	Yes	Yes	Admin, Agent, and Operator Services [d]	tps/ui

[a] Services with a client authentication value of **No** can be reconfigured to require client authentication. Services which do not have either a **Yes** or **No** value cannot be configured to use client authentication.

[b] Although this subsystem type does have end entities ports and interfaces, these end-entity services are not accessible through a web browser, as other end-entity services are.

[c] Although the OCSP does have end entities ports and interfaces, these end-entity services are not accessible through a web browser, as other end-entity services are. End user OCSP services are accessed by a client sending an OCSP request.

[d] The agent, admin, and operator services are all accessed through the same web services page. Each role can only access specific sections which are only visible to the members of that role.

2.2.4.5. Starting the Certificate System Console

The CA, KRA, OCSP, and TKS subsystems have a Java interface which can be accessed to perform administrative functions. For the KRA, OCSP, and TKS, this includes very basic tasks like configuring logging and managing users and groups. For the CA, this includes other configuration settings such as creating certificate profiles and configuring publishing.

The Console is opened by connecting to the subsystem instance over its TLS port using the **pkiconsole** utility. This utility uses the format:

> pkiconsole https://server.example.com:*admin_port/subsystem_type*

The *subsystem_type* can be **ca**, **kra**, **ocsp**, or **tks**. For example, this opens the KRA console:

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

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

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

2.3. CERTIFICATE SYSTEM ARCHITECTURE OVERVIEW

Although each provides a different service, all RHCS subsystems (CA, KRA, OCSP, TKS, TPS) share a common architecture.

2.3.1. Java Application Server

Java application server is a Java framework to run server applications. The Certificate System is designed to run within a Java application server. Currently the only Java application server supported is Tomcat 8. Support for other application servers may be added in the future. More information can be found at http://tomcat.apache.org/.

Each Certificate System instance is a Tomcat server instance. The Tomcat configuration is stored in **server.xml**. The following link provides more information about Tomcat configuration: https://tomcat.apache.org/tomcat-8.0-doc/config/.

Each Certificate System subsystem (such as CA or KRA) is deployed as a web application in Tomcat. The web application configuration is stored in a **web.xml** file, which is defined in Java Servlet 3.1 specification. See https://www.jcp.org/en/jsr/detail?id=340 for details.

The Certificate System configuration itself is stored in **CS.cfg**.

See Section 2.3.16, "Instance Layout" for the actual locations of these files.

2.3.2. Java Security Manager
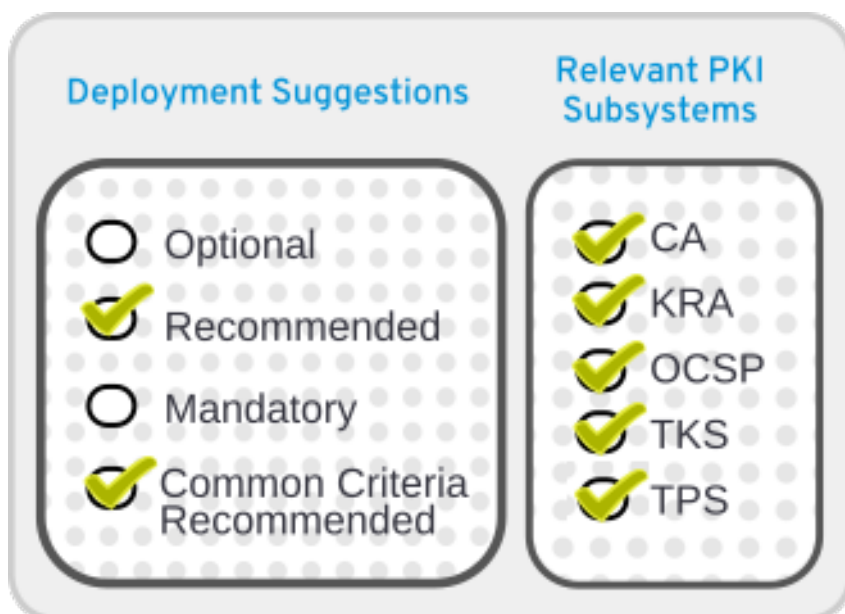
Java services have the option of having a Security Manager which defines unsafe and safe operations for applications to perform. When the subsystems are installed, they have the Security Manager enabled automatically, meaning each Tomcat instance starts with the Security Manager running.

The Security Manager is disabled if the instance is created by running pkispawn and using an override configuration file which specifies the **pki_security_manager=false** option under its own Tomcat section.

The Security Manager can be disabled from an installed instance using the following procedure:

1. # systemctl stop pki-tomcatd@*instance_name*.service

   or

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

   (if using the **nuxwdog** watchdog)

2. Open the **/etc/sysconfig/***instance_name* file, and set **SECURITY_MANAGER="false"**

3. # systemctl start pki-tomcatd@*instance_name*.service

   or

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

   (if using the **nuxwdog** watchdog)

When an instance is started or restarted, a Java security policy is constructed or reconstructed by **pkidaemon** from the following files:

```
/usr/share/pki/server/conf/catalina.policy
/usr/share/tomcat/conf/catalina.policy
/var/lib/pki/$PKI_INSTANCE_NAME/conf/pki.policy
/var/lib/pki/$PKI_INSTANCE_NAME/conf/custom.policy
```

Then, it is saved into **/var/lib/pki/***instance_name***/conf/catalina.policy**.

2.3.2.1. Running Subsystems under a Java Security Manager

Java services have the option of having a Security Manager which defines unsafe and safe operations for applications to perform. When the subsystems are installed, they have the Security Manager enabled automatically, meaning each Tomcat instance starts with the Security Manager running.

2.3.2.1.1. About the Security Manager Policy Files

When the five Java subsystems (the CA, OCSP, KRA, TKS, and TPS) run within the Java Security Manager, they use a combination of three sets of policies:

- The **catalina.policy** file from the default Tomcat policy located in the **/usr/share/tomcat/conf** directory; this is updated whenever the general Tomcat files are updated.

- A **pki.policy** file, in the **/var/lib/pki/***instance_name***/conf** directory, that is supplied with the subsystem instance.

- A **custom.policy** file, in the **/var/lib/pki/***instance_name***/conf** directory, that contains user-defined security policies.

These three files are concatenated together whenever the Tomcat service starts to create a revised **catalina.policy** file, also in the **/var/lib/pki/***instance_name***/conf** directory, which is used for the instance.

The default **pki.policy** file contains permissions that grant unrestricted access to the Tomcat, LDAP, and symkey services used by the PKI subsystems. For example:

```
// These permissions apply to Tomcat java as utilized by PKI instances
grant codeBase "file:/usr/share/java/tomcat/-" {
    permission java.security.AllPermission;
};
```

The **custom.policy** file is empty by default; administrators can write policies in that file which will be used in addition to the given PKI policies and Tomcat policies.

2.3.2.1.2. Starting a Subsystem Instance without the Java Security Manager

All Java subsystems configured under a PKI Tomcat instance are automatically run under a Java Security Manager (unless the instance was created by overriding **pki_security_manager=true** under the [Tomcat] section in the **/etc/pki/default.cfg** file). However, it is possible to start or restart an instance and run it *without* starting the Java Security Manager, as shown below.

Procedure 2.1. Starting an Instance Without the Java Security Manager

1. Stop the instance.

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

   OR

   ```
   # systemctl stop pki-tomcatd-nuxwdog@instance_name.service (if using the nuxwdog
   watchdog)
   ```

2. Edit the **/etc/sysconfig/instance_name** file and turn off the security manager:

   ```
   SECURITY_MANAGER="false"
   ```

3. Start the instance.

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

   OR

   ```
   systemctl start pki-tomcatd-nuxwdog@instance_name.service (if using the nuxwdog
   watchdog)
   ```

2.3.3. Interfaces

2.3.3.1. Servlet Interface

Each subsystem contains interfaces allowing interaction with various portions of the subsystem. All subsystems share a common administrative interface and have an agent interface that allows for agents to perform the tasks assigned to them. A CA Subsystem has an end-entity interface that allows end-entities to enroll in the PKI. An OCSP Responder subsystem has an end-entity interface allowing end-entities and applications to check for current certificate revocation status. Finally, a TPS has an operator interface.

While the application server provides the connection entry points, Certificate System completes the interfaces by providing the servlets specific to each interface.

The servlets for each subsystem are defined in the corresponding **web.xml** file, that is, **/usr/share/pki/ca/webapps/ca/WEB-INF/web.xml**. The same file also defines the URL of each servlet and the security requirements to access the servlets. See Section 2.3.1, "Java Application Server" for more information.

2.3.3.2. Administrative Interface
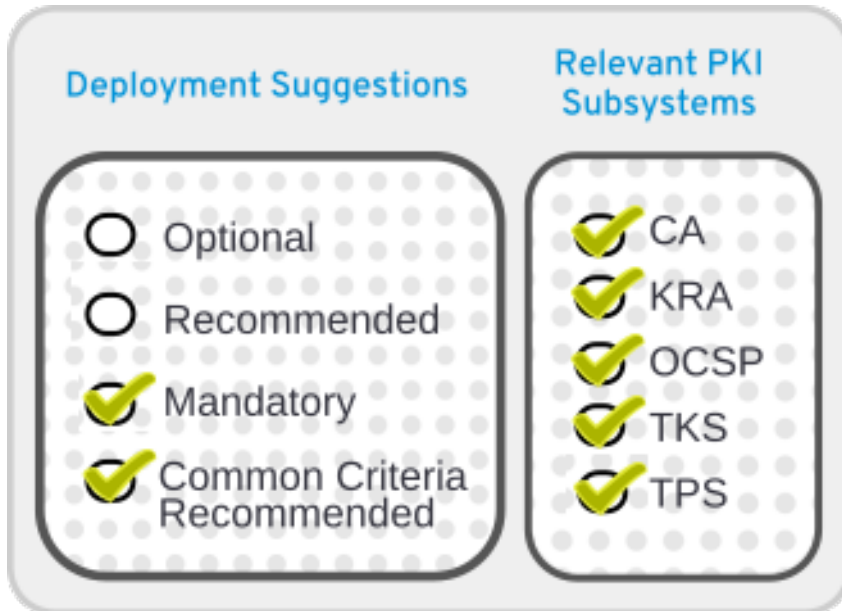
The agent interface provides Java servlets to process HTML form submissions coming from the agent entry point. Based on the information given in each form submission, the agent servlets allow agents to perform agent tasks, such as editing and approving requests for certificate approval, certificate renewal, and certificate revocation, approving certificate profiles. The agent interfaces for a KRA subsystem, or a TKS subsystem, or an OCSP Responder, or a TPS subsystem are specific to the subsystems.

In the non-TMS setup, the agent interface is also used for inter-CIMC boundary communication for the CA-to-KRA trusted connection. This connection is protected by TLS client-authentication and differentiated by separate trusted roles called *Trusted Managers*. Like the agent role, the Trusted Managers (pseudo-users created for inter-CIMC boundary connection only) are required to be TLS client-authenticated. However, unlike the agent role, they are not offered any agent capability.

In the TMS setup, inter-CIMC boundary communication goes from TPS-to-CA, TPS-to-KRA, and TPS-to-TKS.

2.3.3.3. End-Entity Interface
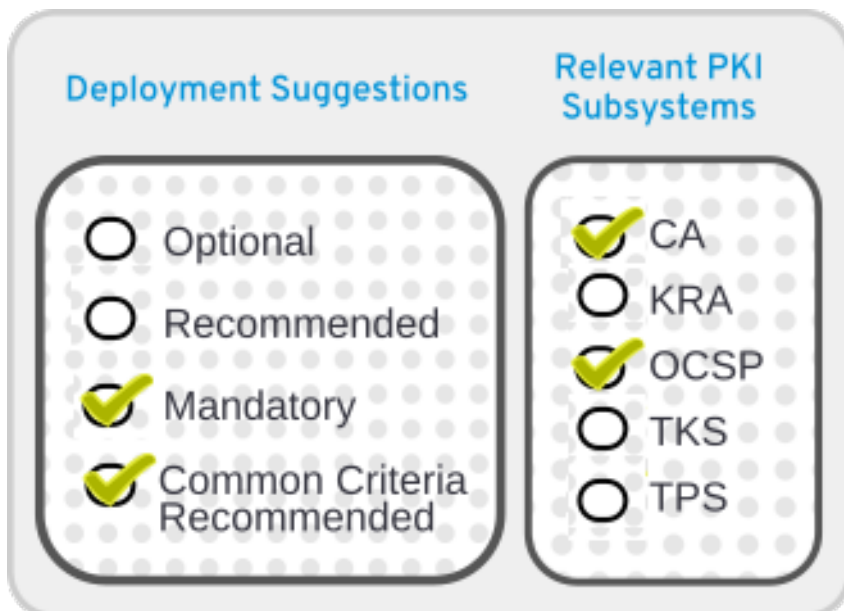
For the CA subsystem, the end-entity interface provides Java servlets to process HTML form submissions coming from the end-entity entry point. Based on the information received from the form submissions, the end-entity servlets allow end-entities to pick up issued certificates, import CA certificate chain, or download a certificate revocation list (CRL) . The OCSP Responder subsystem's End-Entity interface provides Java servlets to accept and process OCSP requests. The KRA, TKS, and TPS subsystems do not offer any End-Entity services.

2.3.3.4. Operator Interface
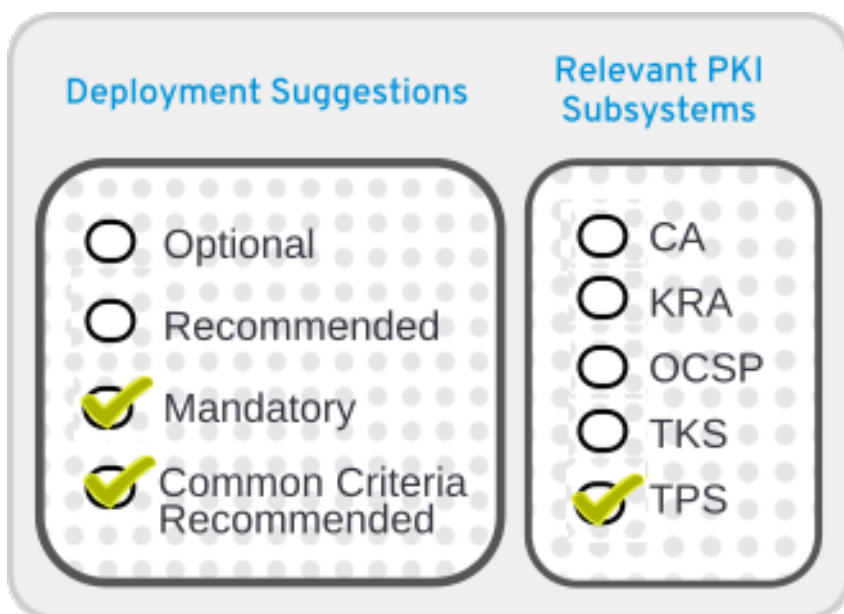
The operator interface is only found in the TPS subsystem.

2.3.4. REST Interface

Representational state transfer (REST) is a way to use HTTP to define and organize web services which will simplify interoperability with other applications. Red Hat Certificate System provides a REST interface to access various services on the server.

The REST services in Red Hat Certificate System are implemented using the RESTEasy framework. RESTEasy is actually running as a servlet in the web application, so the RESTEasy configuration can also

be found in the **web.xml** of the corresponding subsystem. More information about RESTEasy can be found at http://resteasy.jboss.org/.

Each REST service is defined as a separate URL. For example:

- CA certificate service: **http://*<host_name>*:*<port>*/ca/rest/certs/**

- KRA key service: **http://*<host_name>*:*<port>*/kra/rest/agent/keys/**

- TKS user service: **http://*<host_name>*:*<port>*/tks/rest/admin/users/**

- TPS group service: **http://*<host_name>*:*<port>*/tps/rest/admin/groups/**

Some services can be accessed using plain HTTP connection, but some others may require HTTPS connection for security.

The REST operation is specified as HTTP method (for example, GET, PUT, POST, DELETE). For example, to get the CA users the client will send a GET /ca/rest/users request.

The REST request and response messages can be sent in XML or JSON format. For example:

```
{
 "id":"admin",
 "UserID":"admin",
 "FullName":"Administrator",
 "Email":"admin@example.com",
 ...
}
```

The REST interface can be accessed using tools such as CLI, Web UI, or generic REST client. Certificate System also provides Java, Python, and JavaScript libraries to access the services programmatically.

The REST interface supports two types of authentication methods:

- user name and password

- client certificate

The authentication method required by each service is defined in **/usr/share/pki/ca/conf/auth-method.properties**.

The REST interface may require certain permissions to access the service. The permissions are defined in the ACL resources in LDAP. The REST interface are mapped to the ACL resources in the **/usr/share/pki/*<subsystem>*/conf/acl.properties**.

For more information about the REST interface, see http://www.dogtagpki.org/wiki/REST.

2.3.5. NSS

Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled communications applications. Applications built with the NSS libraries support the TLS protocol for authentication, tamper detection, and encryption as well as the PKCS #11 interface for cryptographic token interfaces. Red Hat uses NSS to support these features in a wide range of products, including Certificate System. For further details about NSS, see http://www.mozilla.org/projects/security/pki/nss/overview.html.

2.3.6. JSS

Java Security Services (JSS) provides a Java interface for cryptographic operations performed by NSS. JSS and higher levels of the Certificate System architecture are built with Java Native Interface (JNI), which provides access to native system libraries from within the Java Virtual Machine (JVM). This design allows us to use FIPS approved cryptographic providers such as NSS which are distributed as part of the system. JSS supports most of the security standards and encryption technologies supported by NSS. JSS also provides a pure Java interface for ASN.1 types and BER-DER encoding.

2.3.7. Tomcatjss

Java-based subsystems in Red Hat Certificate System use a single JAR file called **tomcatjss** as a bridge between the Tomcat Server HTTP engine and JSS, the Java interface for security operations performed by NSS. **Tomcatjss** is a Java Secure Socket Extension (JSSE) implementation using Java Security Services (JSS) for Tomcat.

Tomcatjss implements the interfaces needed to use TLS and to create TLS sockets. The socket factory, which tomcatjss implements, makes use of the various properties listed below to create a TLS server listening socket and return it to tomcat. Tomcatjss itself, makes use of our java JSS system to ultimately communicate with the native NSS cryptographic services on the machine.

Tomcatjss is loaded when the Tomcat server and the Certificate System classes are loaded. The load process is described below:

1. The server is started.

2. Tomcat gets to the point where it needs to create the listening sockets for the Certificate System installation.

3. The **server.xml** file is processed. Configuration in this file tells the system to use a socket factory implemented by Tomcatjss.

4. For each requested socket, Tomcajss reads and processes the included attributes when it creates the socket. The resulting socket will behave as it has been asked to by those parameters.

5. Once the server is running, we have the required set of listening sockets waiting for incoming connections to the Tomcat-based Certificate System.

Note that when the sockets are created at startup, Tomcatjss is the *first* entity in Certificate System that actually deals with the underlying JSS security services. Once the first listening socket is processed, an instance of JSS is created for use going forward.

For further details about the **server.xml** file, see Section 9.4, "Configuration Files for the Tomcat Engine and Web Services".

2.3.8. PKCS #11

Public-Key Cryptography Standard (PKCS) #11 specifies an API used to communicate with devices that hold cryptographic information and perform cryptographic operations. Because it supports PKCS #11, Certificate System is compatible with a wide range of hardware and software devices.

At least one PKCS #11 module must be available to any Certificate System subsystem instance. A PKCS #11 module (also called a cryptographic module or cryptographic service provider) manages cryptographic services such as encryption and decryption. PKCS #11 modules are analogous to drivers

for cryptographic devices that can be implemented in either hardware or software. Certificate System contains a built-in PKCS #11 module and can support third-party modules.

A PKCS #11 module always has one or more slots which can be implemented as physical hardware slots in a physical reader such as smart cards or as conceptual slots in software. Each slot for a PKCS #11 module can in turn contain a token, which is a hardware or software device that actually provides cryptographic services and optionally stores certificates and keys.
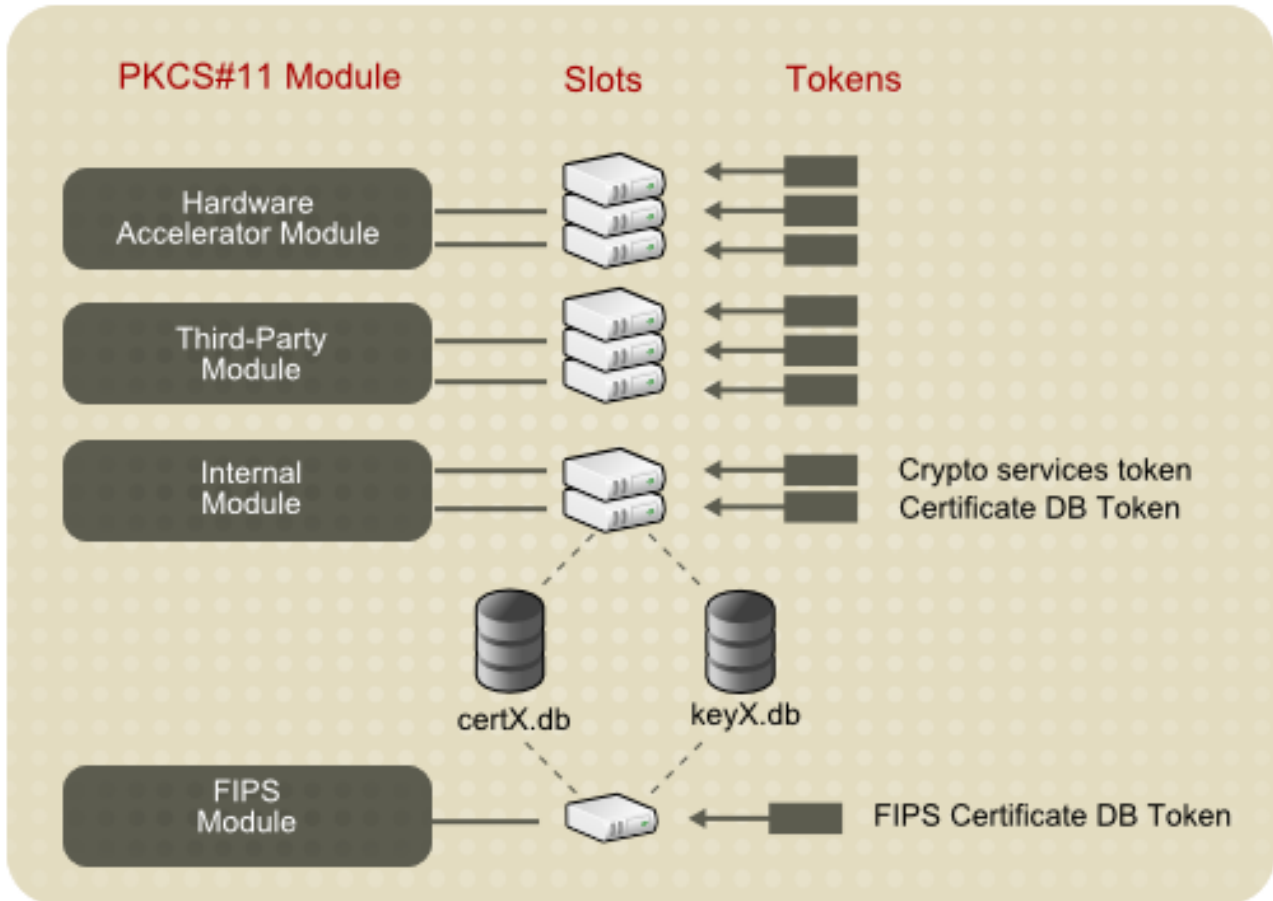
The Certificate System defines two types of tokens, the *internal* and the *external*. The internal token is used for storing certificate trust anchors. The external token is used for storing key pairs and certificates that belong to the Certificate System subsystems.

2.3.8.1. NSS Soft Token (internal token)

> **NOTE**
>
> Certificate System uses an NSS soft token for storing certificate trust anchors.

NSS Soft Token is also called an internal token or a software token. The software token consists of two files, which are usually called the certificate database (cert8.db) and key database (key3.db). The files are created during the Certificate System subsystem configuration. These security databases are located in the **/var/lib/pki/instance_name/alias/** directory.

Two cryptographic modules provided by the NSS soft token are included in the Certificate System:

- The default internal PKCS #11 module, which comes with two tokens:

- The internal crypto services token, which performs all cryptographic operations such as encryption, decryption, and hashing.

- The internal key storage token ("Certificate DB token"), which handles all communication with the certificate and key database files that store certificates and keys.

- The FIPS 140 module. This module complies with the FIPS 140 government standard for cryptographic module implementations. The FIPS 140 module includes a single, built-in FIPS 140 certificate database token, which handles both cryptographic operations and communication with the certificate and key database files.

Specific instructions on how to import certificates onto the NSS soft token are in Chapter 10, *Managing Certificate/Key Crypto Token*.

For more information on the Network Security Services (NSS), see Mozilla Developer web pages of the same name.

2.3.8.2. Hardware Security Module (HSM, external token)

> **NOTE**
>
> Certificate System uses an HSM for storing key pairs and certificates that belong to the Certificate System subsystems.

Any PKCS #11 module can be used with the Certificate System. To use an external hardware token with a subsystem, load its PKCS #11 module before the subsystem is configured, and the new token is available to the subsystem.

Available PKCS #11 modules are tracked in the **secmod.db** database for the subsystem. The **modutil** utility is used to modify this file when there are changes to the system, such as installing a hardware accelerator to use for signing operations. For more information on **modutil**, see Network Security Services (NSS) at Mozilla Developer webpage.

PKCS #11 hardware devices also provide key backup and recovery features for the information stored on hardware tokens. Refer to the PKCS #11 vendor documentation for information on retrieving keys from the tokens.

Specific instructions on how to import certificates and to manage the HSM are in Chapter 10, *Managing Certificate/Key Crypto Token*.

Supported Hardware Security Modules are located in Section 4.3, "Supported Hardware Security Modules".

2.3.9. Certificate System Serial Number Management

2.3.9.1. Serial Number Ranges

Certificate request and serial numbers are represented by Java's big integers

By default, due to their efficiency, certificate request numbers, certificate serial numbers, and replica IDs are assigned sequentially for CA subsystems.

Serial number ranges are specifiable for requests, certificates, and replica IDs:

- Current serial number management is based on assigning ranges of sequential serial numbers.

- Instances request new ranges when crossing below a defined threshold.

- Instances store information about a newly acquired range once it is assigned to the instance.

- Instances continue using old ranges until all numbers are exhausted from it, and then it moves to the new range.

- Cloned subsystems synchronize their range assignment through replication conflicts.

For new clones:

- Part of the current range of the master is transferred to a new clone in the process of cloning.

- New clones may request a new range if the transferred range is below the defined threshold.

All ranges are configurable at CA instance installation time by adding a **[CA]** section to the PKI instance override configuration file, and adding the following **name=value** pairs under that section as needed. Default values which already exist in **/etc/pki/default.cfg** are shown in the following example:

```
[CA]
pki_serial_number_range_start=1
pki_serial_number_range_end=10000000
pki_request_number_range_start=1
pki_request_number_range_end=10000000
pki_replica_number_range_start=1
pki_replica_number_range_end=100
```

2.3.9.2. Random Serial Number Management

In addition to sequential serial number management, Red Hat Certificate System provides optional random serial number management. Using random serial numbers is selectable at CA instance installation time by adding a **[CA]** section to the PKI instance override file and adding the following **name=value** pair under that section:

```
[CA]
pki_random_serial_numbers_enable=True
```

If selected, certificate request numbers and certificate serial numbers will be selected randomly within the specified ranges.

2.3.10. Security Domain

A *security domain* is a registry of PKI services. Services such as CAs register information about themselves in these domains so users of PKI services can find other services by inspecting the registry. The security domain service in RHCS manages both the registration of PKI services for Certificate System subsystems and a set of shared trust policies.

See Section 5.3, "Planning Security Domains" for further details.

2.3.11. Passwords and Watchdog (nuxwdog)

In the default setup, an RHCS subsystem instance needs to act as a client and authenticate to some other services, such as an LDAP internal database (unless TLS client authentication is set up, where a certificate will be used for authentication instead), the NSS token database, or sometimes an HSM with a password. The administrator is prompted to set up this password at the time of installation

configuration. This password is then written to the file **<instance_dir>/conf/password.conf**. At the same time, an identifying string is stored in the main configuration file **CS.cfg** as part of the parameter **cms.passwordlist**.

The configuration file, **CS.cfg**, is protected by Red Hat Enterprise Linux, and only accessible by the PKI administrators. No passwords are stored in **CS.cfg**.

During installation, the installer will select and log into either the internal software token or a hardware cryptographic token. The login passphrase to these tokens is also written to **password.conf**.

Configuration at a later time can also place passwords into **password.conf**. LDAP publishing is one example where the newly configured Directory Manager password for the publishing directory is entered into **password.conf**.

Nuxwdog (watchdog) is a lightweight auxiliary daemon process that is used to start, stop, monitor the status of, and reconfigure server programs. It is most useful when users need to be prompted for passwords to start a server, because it caches these passwords securely in the kernel keyring, so that restarts can be done automatically in the case of a server crash.
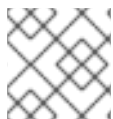
> **NOTE**
>
> **Nuxwdog** is the only allowed watchdog service.

Once installation is complete, it is possible to remove the **password.conf** file altogether. On restart, the **nuxwdog** watchdog program will prompt the administrator for the required passwords, using the parameter **cms.passwordlist** (and *cms.tokenList* if an HSM is used) as a list of passwords for which to prompt. The passwords are then cached by **nuxwdog** in the kernel keyring to allow automated recovery from a server crash. This automated recovery (automatic subsystem restart) happens in case of uncontrolled shutdown (crash). In case of a controlled shutdown by the administrator, administrators are prompted for passwords again.

When using the watchdog service, starting and stopping an RHCS instance are done differently. For details, see Section 9.3.2, "Using the Certificate System Watchdog Service" .

For further information, see Section 9.3, "Managing System Passwords".

2.3.12. Internal LDAP Database

Red Hat Certificate System employs Red Hat Directory Server (RHDS) as its internal database for storing information such as certificates, requests, users, roles, ACLs, as well as other miscellaneous internal information. TLS client-authentication is required between a Certificate System instance and Directory Server. It is therefore important to follow instruction to set up trust between these two entities.

Proper **pkispawn** options will also be needed for installing such Certificate System instance.

For details, see Section 6.5, "Installing Red Hat Directory Server" .

2.3.13. Security-Enhanced Linux (SELinux)

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering. SELinux is described in more detail in *Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide*.

Basically, SELinux identifies *objects* on a system, which can be files, directories, users, processes,

sockets, or any other resource on a Linux host. These objects correspond to the Linux API objects. Each object is then mapped to a *security context*, which defines the type of object and how it is allowed to function on the Linux server.

Objects can be grouped into domains, and then each domain is assigned the proper rules. Each security context has rules which set restrictions on what operations it can perform, what resources it can access, and what permissions it has.

SELinux policies for the Certificate System are incorporated into the standard system SELinux policies. These SELinux policies apply to every subsystem and service used by Certificate System. By running Certificate System with SELinux in enforcing mode, the security of the information created and maintained by Certificate System is enhanced.
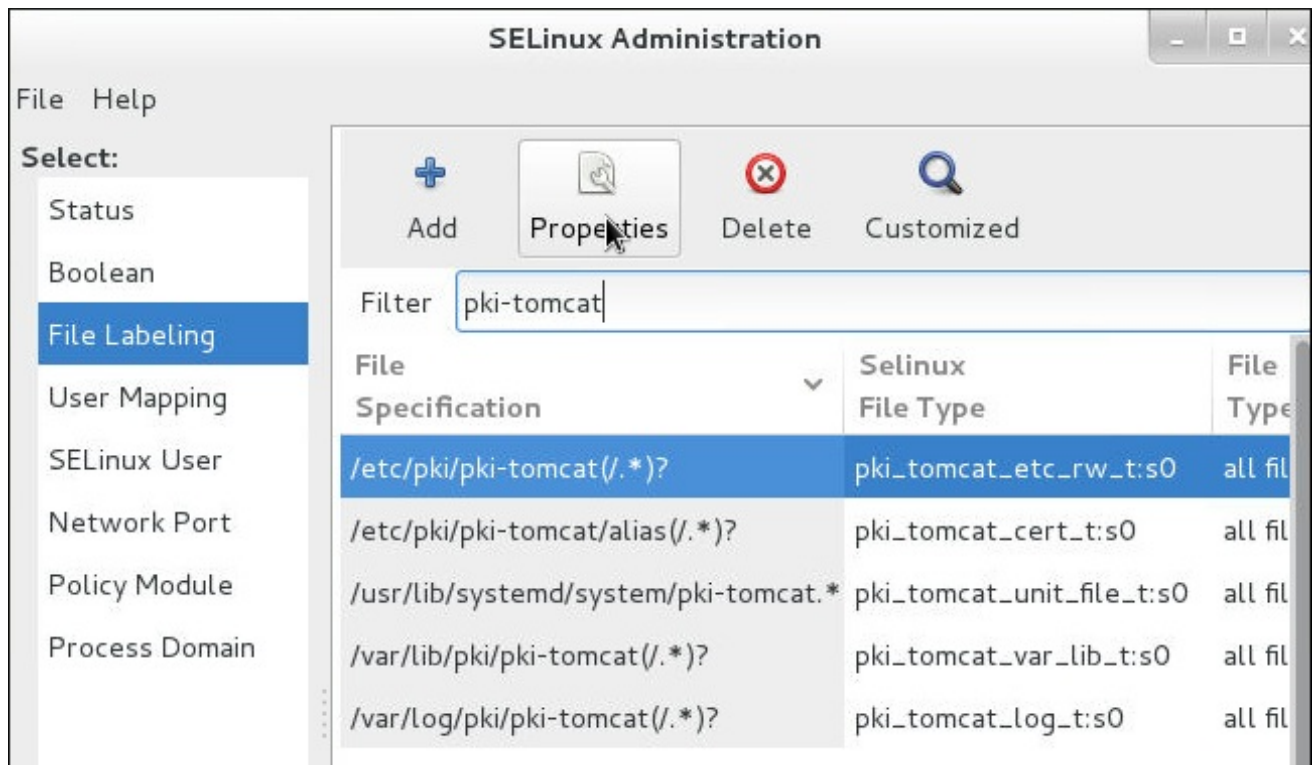
Figure 2.1. CA SELinux Port Policy

The Certificate System SELinux policies define the SELinux configuration for every subsystem instance:

- Files and directories for each subsystem instance are labeled with a specific SELinux context.

- The ports for each subsystem instance are labeled with a specific SELinux context.

- All Certificate System processes are constrained within a subsystem-specific domain.

- Each domain has specific rules that define what actions that are authorized for the domain.

- Any access not specified in the SELinux policy is denied to the Certificate System instance.

For Certificate System, each subsystem is treated as an SELinux object, and each subsystem has unique rules assigned to it. The defined SELinux policies allow Certificate System objects run with SELinux set in enforcing mode.

Every time **pkispawn** is run to configure a Certificate System subsystem, files and ports associated with that subsystem are labeled with the required SELinux contexts. These contexts are removed when the particular subsystems are removed using **pkidestroy**.

The central definition in an SELinux policy is the **pki_tomcat_t** domain. Certificate System instances are Tomcat servers, and the **pki_tomcat_t** domain extends the policies for a standard **tomcat_t** Tomcat domain. All Certificate System instances on a server share the same domain.

When each Certificate System process is started, it initially runs in an unconfined domain (**unconfined_t**) and then transitions into the **pki_tomcat_t** domain. This process then has certain access permissions, such as write access to log files labeled **pki_tomcat_log_t**, read and write access to configuration files labeled **pki_tomcat_etc_rw_t**, or the ability to open and write to **http_port_t** ports.

The SELinux mode can be changed from enforcing to permissive, or even off, though this is not recommended.

2.3.14. Self-tests

Red Hat Certificate System provides a Self-Test framework which allows the PKI system integrity to be checked during startup or on demand or both. In the event of a non-critical self test failure, the message will be stored in the log file, while in the event of a critical self test failure, the message will be stored in the log file, while the Certificate System subsystem will properly shut down. The administrator is expected to watch the self-test log during the startup of the subsystem if they wish to see the self-test report during startup. They can also view the log after startup.

When a subsystem is shut down due to a self-test failure, it will also be automatically disabled. This is done to ensure that the subsystem does not partially run and produce misleading responses. Once the issue is resolved, the subsystem can be re-enabled by running the following command on the server:

```
# pki-server subsystem-enable <subsystem>
```

For detail on how to configure self-tests, see Section 13.3.2, "Configuring Self-Tests".

2.3.15. Logs

The Certificate System subsystems create log files that record events related to activities, such as administration, communications using any of the protocols the server supports, and various other processes employed by the subsystems. While a subsystem instance is running, it keeps a log of information and error messages on all the components it manages. Additionally, the Apache and Tomcat web servers generate error and access logs.

Each subsystem instance maintains its own log files for installation, audit, and other logged functions.

Log plug-in modules are listeners which are implemented as Java™ classes and are registered in the configuration framework.

All the log files and rotated log files, except for audit logs, are located in whatever directory was specified in **pki_subsystem_log_path** when the instance was created with **pkispawn**. Regular audit logs are located in the log directory with other types of logs, while signed audit logs are written to **/var/log/pki/*instance_name*/*subsystem_name*/signedAudit**. The default location for logs can be changed by modifying the configuration.

For details about log configuration during the installation and additional information, see Chapter 13, *Configuring Logs*.

For details about log administration after the installation, see the *Configuring Subsystem Logs* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)* .

2.3.15.1. Audit Log

The audit log contains records for selectable events that have been set up as recordable events. You can configure audit logs also to be signed for integrity-checking purposes.
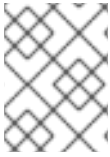
> **NOTE**
>
> Audit records should be kept according to the audit retention rules specified in Section 13.4, "Audit Retention"

2.3.15.2. System Log

This log, **system**, records information about requests to the server (all HTTP and HTTPS requests) and the responses from the server. Information recorded in this log includes the IP address (both IPv4 and IPv6) of the client machine that accessed the server; operations performed, such as search, add, and edit; and the result of the access, such as the number of entries returned:

> *id_number processor - [date:time] [number_of_operations] [result] servlet: message*

> **Example 2.1. TKS System Log**
>
> 10439.http-13443-Processor25 - [19/May/2021:14:16:51 CDT] [11] [3] UGSubsystem: Get User Error User not found

This log is on by default.

2.3.15.3. Transactions Log

This log, **transactions**, records any kind of operation performed or submitted to the subsystem.

> *id_number.processor - [date:time] [number_of_operations] [result] servlet: message*

These messages are specific to the certificate service, such as the CA receiving certificate requests, the KRA archiving or retrieving keys, and the TKS registering a new TPS. This log can also be used to detect any unauthorized access or activity.

> **Example 2.2. Transactions Log**
>
> 11438.http-8443-Processor25 - [27/May/2021:07:56:18 CDT] [1] [1] archival reqID 4 fromAgent agentID: CA-server.example.com-8443 authenticated by noAuthManager is completed DN requested: UID=recoverykey,E=recoverykey@email.com,CN=recover key serial number: 0x3

This log is on by default.

2.3.15.4. Debug Logs

Debug logs, which are enabled by default, are maintained for all subsystems, with varying degrees and types of information.

Debug logs for each subsystem record much more detailed information than system, transaction, and access logs. Debug logs contain very specific information for every operation performed by the subsystem, including plug-ins and servlets which are run, connection information, and server request and

response messages.

Services which are recorded to the debug log include authorization requests, processing certificate requests, certificate status checks, and archiving and recovering keys, and access to web services.

The debug logs record detailed information about the processes for the subsystem. Each log entry has the following format:

> [*date:time*] [*processor*]: *servlet*: *message*

The *message* can be a return message from the subsystem or contain values submitted to the subsystem.

For example, the TKS records this message for connecting to an LDAP server:

> [10/Jun/2021:05:14:51][main]: Established LDAP connection using basic authentication to host localhost port 389 as cn=Directory Manager

The *processor* is **main**, and the *message* is the message from the server about the LDAP connection, and there is no servlet.

The CA, on the other hand, records information about certificate operations as well as subsystem connections:

> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestowner$ value=KRA-server.example.com-8443

In this case, the *processor* is the HTTP protocol over the CA's agent port, while it specifies the servlet for handling profiles and contains a *message* giving a profile parameter (the subsystem owner of a request) and its value (that the KRA initiated the request).

> **Example 2.3. CA Certificate Request Log Messages**
>
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.profileapprovedby$ value=admin
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.cert_request$
> value=MIIBozCCAZ8wggEFAgQqTfoHMIHHgAECpQ4wDDEKMAgGA1UEAxMBeKaBnzANBgkqhki
> G9w0BAQEFAAOB...
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profile$
> value=true
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.cert_request_type$ value=crmf
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.requestversion$ value=1.0.0
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_locale$
> value=en
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.requestowner$ value=KRA-server.example.com-8443
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.dbstatus$
> value=NOT_UPDATED
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.subject$
> value=uid=jsmith, e=jsmith@example.com
> [06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
> key=$request.requeststatus$ value=begin

[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.user$ value=uid=KRA-server.example.com-8443,ou=People,dc=example,dc=com
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_key$ value=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDreuEsBWq9WuZ2MaBwtNYxvkLP^M
HcN0cusY7gxLzB+XwQ/VsWEoObGldg6WwJPOcBdvLiKKfC605wFdynbEgKs0fChV^M
k9HYDhmJ8hX6+PaquiHJSVNhsv5tOshZkCfMBbyxwrKd8yZ5G5I+2gE9PUznxJaM^M
HTmlOqm4HwFxzy0RRQIDAQAB
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.authmgrinstname$ value=raCertAuth
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.uid$ value=KRA-server.example.com-8443
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.userid$ value=KRA-server.example.com-8443
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestor_name$ value=
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profileid$ value=caUserCert
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.userdn$ value=uid=KRA-server.example.com-4747,ou=People,dc=example,dc=com
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestid$ value=20
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.authtime$ value=1212782378071
[06/Jun/2021:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_x509info$ value=MIICIKADAgECAgEAMA0GCSqGSIb3DQEBBQUAMEAxHjAcBgNVBAoTFVJlZGJ1ZGNv^M

bXB1dGVyIERvbWFpbjEeMBwGA1UEAxMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4X^M
DTA4MDYwNjE5NTkzOFoXDTA4MTIwMzE5NTkzOFowOzEhMB8GCSqGSIb3DQEJARYS^M
anNtaXRoQGV4YW1wbGUuY29tMRYwFAYKCZImiZPyLGQBARMGanNtaXRoMIGfMA0G^M
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDreuEsBWq9WuZ2MaBwtNYxvkLPHcN0cusY^M
7gxLzB+XwQ/VsWEoObGldg6WwJPOcBdvLiKKfC605wFdynbEgKs0fChVk9HYDhmJ^M
8hX6+PaquiHJSVNhsv5tOshZkCfMBbyxwrKd8yZ5G5I+2gE9PUznxJaMHTmlOqm4^M
HwFxzy0RRQIDAQABo4HFMIHCMB8GA1UdIwQYMBaAFG8gWeOJIMt+aO8VuQTMzPBU^M
78k8MEoGCCsGAQUFBwEBBD4wPDA6BggrBgEFBQcwAYYuaHR0cDovL3Rlc3Q0LnJl^M
ZGJ1ZGNvbXB1dGVyLmxvY2FsOjkwODAvY2Evb2NzcDAOBgNVHQ8BAf8EBAMCBeAw^M
HQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMEMCQGA1UdEQQdMBuBGSRyZXF1^M
ZXN0LnJlcXVlc3Rvcl9lbWFpbCQ=

Likewise, the OCSP shows OCSP request information:

[07/Jul/2021:06:25:40][http-11180-Processor25]: OCSPServlet: OCSP Request:
[07/Jul/2021:06:25:40][http-11180-Processor25]: OCSPServlet: MEUwQwIBADA+MDwwOjAJBgUrDgMCGgUABBSEWjCarLE6/BiSiENSsV9kHjqB3QQU

2.3.15.5. Installation Logs

All subsystems keep an install log.

Every time a subsystem is created either through the initial installation or creating additional instances

with **pkispawn**, an installation file with the complete debug output from the installation, including any errors and, if the installation is successful, the URL and PIN to the configuration interface for the instance. The file is created in the **/var/log/pki/** directory for the instance with a name in the form **pki-*subsystem_name*-spawn.*timestamp*.log**.

Each line in the install log follows a step in the installation process.

> **Example 2.4. CA Install Log**
>
> ```
> ...
> 2015-07-22 20:43:13 pkispawn    : INFO     ... finalizing
> 'pki.server.deployment.scriptlets.finalization'
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... cp -p /etc/sysconfig/pki/tomcat/pki-
> tomcat/ca/deployment.cfg /var/log/pki/pki-
> tomcat/ca/archive/spawn_deployment.cfg.20150722204136
> 2015-07-22 20:43:13 pkispawn    : DEBUG    ........... chmod 660 /var/log/pki/pki-
> tomcat/ca/archive/spawn_deployment.cfg.20150722204136
> 2015-07-22 20:43:13 pkispawn    : DEBUG    ........... chown 26445:26445 /var/log/pki/pki-
> tomcat/ca/archive/spawn_deployment.cfg.20150722204136
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... generating manifest file called
> '/etc/sysconfig/pki/tomcat/pki-tomcat/ca/manifest'
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... cp -p /etc/sysconfig/pki/tomcat/pki-
> tomcat/ca/manifest /var/log/pki/pki-tomcat/ca/archive/spawn_manifest.20150722204136
> 2015-07-22 20:43:13 pkispawn    : DEBUG    ........... chmod 660 /var/log/pki/pki-
> tomcat/ca/archive/spawn_manifest.20150722204136
> 2015-07-22 20:43:13 pkispawn    : DEBUG    ........... chown 26445:26445 /var/log/pki/pki-
> tomcat/ca/archive/spawn_manifest.20150722204136
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... executing 'systemctl enable pki-tomcatd.target'
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... executing 'systemctl daemon-reload'
> 2015-07-22 20:43:13 pkispawn    : INFO     ....... executing 'systemctl restart pki-tomcatd@pki-
> tomcat.service'
> 2015-07-22 20:43:14 pkispawn    : INFO     END spawning subsystem 'CA' of instance 'pki-tomcat'
> 2015-07-22 20:43:14 pkispawn    : DEBUG
> ```

2.3.15.6. Tomcat Error and Access Logs

The CA, KRA, OCSP, TKS, and TPS subsystems use a Tomcat web server instance for their agent and end-entities' interfaces.

Error and access logs are created by the Tomcat web server, which are installed with the Certificate System and provide HTTP services. The error log contains the HTTP error messages the server has encountered. The access log lists access activity through the HTTP interface.

Logs created by Tomcat:

- admin.*timestamp*

- catalina.*timestamp*

- catalina.out

- host-manager.*timestamp*

- localhost.*timestamp*

- localhost_access_log.*timestamp*

- manager.*timestamp*

These logs are not available or configurable within the Certificate System; they are only configurable within Apache or Tomcat. See the Apache documentation for information about configuring these logs.

2.3.15.7. Self-Tests Log

The self-tests log records information obtained during the self-tests run when the server starts or when the self-tests are manually run. The tests can be viewed by opening this log. This log is not configurable through the Console. This log can only be configured by changing settings in the **CS.cfg** file. The information about logs in this section does not pertain to this log. See Section 2.6.5, "Self-Tests" for more information about self-tests.

2.3.15.8. journalctl Logs

When starting a Certificate System instance, there is a short period of time before the logging subsystem is set up and enabled. During this time, log contents are written to standard out, which is captured by **systemd** and exposed via the **journalctl** utility.

To view these logs, run the following command:

```
# journalctl -u pki-tomcatd@instance_name.service
```

If using the **nuxwdog** service:

```
# journalctl -u pki-tomcatd-nuxwdog@instance_name.service
```

Often it is helpful to watch these logs as the instance is starting (for example, in the event of a self-test failure on startup). To do this, run these commands in a separate console prior to starting the instance:

```
# journalctl -f -u pki-tomcatd@instance_name.service
```

If using the **nuxwdog** service:

```
# journalctl -f -u pki-tomcatd-nuxwdog@instance_name.service
```

2.3.16. Instance Layout

Each Certificate System instance depends on a number of files. Some of them are located in instance-specific folders, while some others are located in a common folder which is shared with other server instances.

For example, the server configuration files are stored in **/etc/pki/*instance_name*/server.xml**, which is instance-specific, but the CA servlets are defined in **/usr/share/pki/ca/webapps/ca/WEB-INF/web.xml**, which is shared by all server instances on the system.

2.3.16.1. File and Directory Locations for Certificate System

Certificate System servers are Tomcat instances which consist of one or more Certificate System subsystems. Certificate System subsystems are web applications that provide specific type of PKI functions. General, shared subsystem information is contained in non-relocatable, RPM-defined shared

libraries, Java archive files, binaries, and templates. These are stored in a fixed location.

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

The directories contain customized configuration files and templates, profiles, certificate databases, and other files for the subsystem.

Table 2.2. Tomcat Instance Information

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat
Configuration Directory	/etc/pki/pki-tomcat
Configuration File	/etc/pki/pki-tomcat/server.xml /etc/pki/pki-tomcat/password.conf
Security Databases	/var/lib/pki/pki-tomcat/alias
Subsystem Certificates	TLS server certificate Subsystem certificate [a]
Log Files	/var/log/pki/pki-tomcat
Web Services Files	/usr/share/pki/server/webapps/ROOT – Main page /usr/share/pki/server/webapps/pki/admin – Admin templates /usr/share/pki/server/webapps/pki/js – JavaScript libraries

[a] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

NOTE

The **/var/lib/pki/***instance_name***/conf/** directory is a symbolic link to the **/etc/pki/***instance_name***/** directory.

2.3.16.2. CA Subsystem Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

Table 2.3. CA Subsystem Information

Setting	Value

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat/ca
Configuration Directory	/etc/pki/pki-tomcat/ca
Configuration File	/etc/pki/pki-tomcat/ca/CS.cfg
Subsystem Certificates	CA signing certificate OCSP signing certificate (for the CA's internal OCSP service) Audit log signing certificate
Log Files	/var/log/pki/pki-tomcat/ca
Install Logs	/var/log/pki/pki-ca-spawn.YYYYMMDDhhmmss.log
Profile Files	/var/lib/pki/pki-tomcat/ca/profiles/ca
Email Notification Templates	/var/lib/pki/pki-tomcat/ca/emails
Web Services Files	/usr/share/pki/ca/webapps/ca/agent – Agent services /usr/share/pki/ca/webapps/ca/admin – Admin services /usr/share/pki/ca/webapps/ca/ee – End user services

2.3.16.3. KRA Subsystem Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

Table 2.4. KRA Subsystem Information

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat/kra
Configuration Directory	/etc/pki/pki-tomcat/kra
Configuration File	/etc/pki/pki-tomcat/kra/CS.cfg
Subsystem Certificates	Transport certificate Storage certificate Audit log signing certificate
Log Files	/var/log/pki/pki-tomcat/kra

Setting	Value
Install Logs	/var/log/pki/pki-kra-spawn.YYYYMMDDhhmmss.log
Web Services Files	/usr/share/pki/kra/webapps/kra/agent – Agent services /usr/share/pki/kra/webapps/kra/admin – Admin services

2.3.16.4. OCSP Subsystem Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

Table 2.5. OCSP Subsystem Information

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat/ocsp
Configuration Directory	/etc/pki/pki-tomcat/ocsp
Configuration File	/etc/pki/pki-tomcat/ocsp/CS.cfg
Subsystem Certificates	OCSP signing certificate Audit log signing certificate
Log Files	/var/log/pki/pki-tomcat/ocsp
Install Logs	/var/log/pki/pki-ocsp-spawn.YYYYMMDDhhmmss.log
Web Services Files	/usr/share/pki/ocsp/webapps/ocsp/agent – Agent services /usr/share/pki/ocsp/webapps/ocsp/admin – Admin services

2.3.16.5. TKS Subsystem Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

Table 2.6. TKS Subsystem Information

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat/tks
Configuration Directory	/etc/pki/pki-tomcat/tks
Configuration File	/etc/pki/pki-tomcat/tks/CS.cfg

Setting	Value
Subsystem Certificates	Audit log signing certificate
Log Files	/var/log/pki/pki-tomcat/tks
Install Logs	/var/log/pki/pki-tomcat/pki-tks-spawn.YYYYMMDDhhmmss.log

2.3.16.6. TPS Subsystem Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tomcat**; the true value is whatever is specified at the time the subsystem is created with **pkispawn**.

Table 2.7. TPS Subsystem Information

Setting	Value
Main Directory	/var/lib/pki/pki-tomcat/tps
Configuration Directory	/etc/pki/pki-tomcat/tps
Configuration File	/etc/pki/pki-tomcat/tps/CS.cfg
Subsystem Certificates	Audit log signing certificate
Log Files	/var/log/pki/pki-tomcat/tps
Install Logs	/var/log/pki/pki-tps-spawn.YYYYMMDDhhhmmss.log
Web Services Files	/usr/share/pki/tps/webapps/tps – TPS services

2.3.16.7. Shared Certificate System Subsystem File Locations

There are some directories used by or common to all Certificate System subsystem instances for general server operations, listed in Table 2.8, "Subsystem File Locations".

Table 2.8. Subsystem File Locations

Directory Location	Contents

Directory Location	Contents
/usr/share/pki	Contains common files and templates used to create Certificate System instances. Along with shared files for all subsystems, there are subsystem-specific files in subfolders: ● pki/ca (CA) ● pki/kra (KRA) ● pki/ocsp (OCSP) ● pki/tks (TKS) ● pki/tps (TPS)
/usr/bin	Contains the **pkispawn** and **pkidestroy** instance configuration scripts and tools (Java, native, and security) shared by the Certificate System subsystems.
/usr/share/java/pki	Contains Java archive files shared by local Tomcat web applications and shared by the Certificate System subsystems.

2.4. PKI WITH CERTIFICATE SYSTEM

The Certificate System is comprised of subsystems which each contribute different functions of a public key infrastructure. A PKI environment can be customized to fit individual needs by implementing different features and functions for the subsystems.

NOTE

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. A *TMS* environment manages the 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.

2.4.1. Issuing Certificates

As stated, the Certificate Manager is the heart of the Certificate System. It manages certificates at every stage, from requests through enrollment (issuing), renewal, and revocation.

The Certificate System supports enrolling and issuing certificates and processing certificate requests from a variety of end entities, such as web browsers, servers, and virtual private network (VPN) clients. Issued certificates conform to X.509 version 3 standards. Certificate System must be configured to handle CMC requests only through the evaluated interface.

2.4.1.1. Enrollment Using the Command Line

This section describes the general workflows when enrolling certificates using the command line.

2.4.1.1.1. Enrolling with CMC

To enroll a certificate with CMC, proceed as follows:

1. Generate a PKCS #10 or CRMF certificate signing request (CSR) using a utility, such as **certutil**, **openssl**, **PKCS10Client**, or **CRMFPopClient**. For details, see the *Creating Certificate Signing Requests* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.
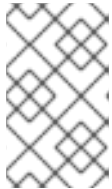
   > **NOTE**
   >
   > If key archival is enabled in the Key Recovery Agent (KRA), use the **CRMFPopClient** utility with the KRA's transport certificate in Privacy Enhanced Mail (PEM) format set in the **kra.transport** file.

2. Use the **CMCRequest** utility to convert the CSR into a CMC request. The **CMCRequest** utility uses a configuration file as input. This file contains, for example, the path to the CSR and the CSR's format.

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

3. Use the **HttpClient** utility to send the CMC request to the CA. **HttpClient** uses a configuration file with settings, such as the path to the CMC request file and the servlet.

   If the **HttpClient** command succeeds, the utility receives a PKCS #7 chain with CMC status controls in a CMC response from the CA.

   For details about what parameters the utility provides, enter the **HttpClient** command without any parameters.

4. Use the **CMCResponse** utility to check the issuance result of the CMC response file generated by **HttpClient**. If the request is successful, **CMCResponse** displays the certificate chain in a readable format along with status information.

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

5. Import the new certificate into the application. For details, follow the instructions of the application to which you want to import the certificate.

   > **NOTE**
   >
   > The certificate retrieved by **HttpClient** is in CMC response format that contains PKCS #7. If the application supports only Base64-encoded certificates, use the **BtoA** utility to convert the certificate.
   >
   > Additionally, certain applications require a header and footer for certificates in Privacy Enhanced Mail (PEM) format. If these are required, add them manually to the PEM file after you converted the certificate.

2.4.1.1.2. CMC Enrollment without POP

In situations when Proof Of Possession (POP) is missing, the **HttpClient** utility receives an **EncryptedPOP** CMC status, which is displayed by the **CMCResponse** command. In this case, enter the **CMCRequest** command again with different parameters in the configuration file.

For details, see the *The CMC Enrollment Process* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

2.4.1.1.3. Signed CMC Requests

CMC requests can either be signed by a user or a CA agent:

- If an agent signs the request, set the authentication method in the profile to **CMCAuth**.

- If a user signs the request, set the authentication method in the profile to **CMCUserSignedAuth**.

For details, see the *CMC Authentication Plug-ins* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

2.4.1.1.4. Unsigned CMC Requests

When the **CMCUserSignedAuth** authentication plug-in is configured in the profile, you must use an unsigned CMC request in combination with the Shared Secret authentication mechanism.

> **NOTE**
>
> Unsigned CMC requests are also called **self-signed CMC requests**.

For details, see the *CMC Authentication Plug-ins* and *The CMC Shared Secret Feature* sections in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

2.4.1.1.5. The Shared Secret Workflow

Certificate System provides the Shared Secret authentication mechanism for CMC requests according to RFC 5272. In order to protect the passphrase, an issuance protection certificate must be provided when using the **CMCSharedToken** command. The issuance protection certificate works similar to the KRA transport certificate. For further details, see the CMCSharedToken(1) man page and the *The CMC Shared Secret Feature* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Shared Secret Create by the End Entity User (Preferred)
The following describes the workflow, if the user generates the shared secret:

1. The end entity user obtains the issuance protection certificate from the CA administrator.

2. The end entity user uses the **CMCSharedToken** utility to generate a shared secret token.

> **NOTE**
>
> The **-p** option sets the passphrase that is shared between the CA and the user, not the password of the token.

3. The end entity user sends the encrypted shared token generated by the **CMCSharedToken** utility to the administrator.

4. The administrator adds the shared token into the **shrTok** attribute in the user's LDAP entry.

5. The end entity user uses the passphrase to set the ***witness.sharedSecret*** parameter in the configuration file passed to the **CMCRequest** utility.

Shared Secret Created by the CA Administrator
The following describes the workflow, if the CA administrator generates the shared secret for a user:

1. The administrator uses the **CMCSharedToken** utility to generate a shared secret token for the user.

   > **NOTE**
   >
   > The **-p** option sets the passphrase that is shared between the CA and the user, not the password of the token.

2. The administrator adds the shared token into the **shrTok** attribute in the user's LDAP entry.

3. The administrator shares the passphrase with the user.

4. The end entity user uses the passphrase to set the ***witness.sharedSecret*** parameter in the configuration file passed to the **CMCRequest** utility.

2.4.1.1.6. Simple CMC Requests

Certificate System allows simple CMC requests. However, this process does not support the same level of security requirements as full CMC requests and, therefore, must only be used in a secure environment.

When using simple CMC requests, set the following in the **HttpClient** utility's configuration file:

```
servlet=/ca/ee/ca/profileSubmitCMCSimple?profileId=caECSimpleCMCUserCert
```

2.4.1.2. Certificate Profiles

The Certificate System uses certificate profiles to configure the content of the certificate, the constraints for issuing the certificate, the enrollment method used, and the input and output forms for that enrollment. A single certificate profile is associated with issuing a particular type of certificate.

A set of certificate profiles is included for the most common certificate types; the profile settings can be modified. Certificate profiles are configured by an administrator, and then sent to the agent services page for agent approval. Once a certificate profile is approved, it is enabled for use. In case of a UI-enrollment, a dynamically-generated HTML form for the certificate profile is used in the end-entities page for certificate enrollment, which calls on the certificate profile. In case of a command line-based enrollment, the certificate profile is called upon to perform the same processing, such as authentication, authorization, input, output, defaults, and constraints. The server verifies that the defaults and constraints set in the certificate profile are met before acting on the request and uses the certificate profile to determine the content of the issued certificate.

The Certificate Manager can issue certificates with any of the following characteristics, depending on the configuration in the profiles and the submitted certificate request:

- Certificates that are X.509 version 3-compliant

- Unicode support for the certificate subject name and issuer name

- Support for empty certificate subject names

- Support for customized subject name components

- Support for customized extensions

By default, the certificate enrollment profiles are stored in ***<instance directory>/ca/profiles/ca*** with names in the format of ***<profile id>*.cfg**. LDAP-based profiles are possible with proper **pkispawn** configuration parameters.

2.4.1.3. Authentication for Certificate Enrollment

Certificate System provides authentication options for certificate enrollment. These include agent-approved enrollment, in which an agent processes the request, and automated enrollment, in which an authentication method is used to authenticate the end entity and then the CA automatically issues a certificate. CMC enrollment is also supported, which automatically processes a request approved by an agent.

2.4.1.4. Cross-Pair Certificates

It is possible to create a trusted relationship between two separate CAs by issuing and storing cross-signed certificates between these two CAs. By using cross-signed certificate pairs, certificates issued outside the organization's PKI can be trusted within the system.

2.4.2. Renewing Certificates

When certificates reach their expiration date, they can either be allowed to lapse, or they can be renewed.

Renewal regenerates a certificate request using the existing key pairs for that certificate, and then resubmits the request to Certificate Manager. The renewed certificate is identical to the original (since it was created from the same profile using the same key material) with one exception — it has a different, later expiration date.

Renewal can make managing certificates and relationships between users and servers much smoother, because the renewed certificate functions precisely as the old one. For user certificates, renewal allows encrypted data to be accessed without any loss.

2.4.3. Publishing Certificates and CRLs

Certificates can be published to files and an LDAP directory, and CRLs to files, an LDAP directory, and an OCSP responder. The publishing framework provides a robust set of tools to publish to all three places and to set rules to define with more detail which types of certificates or CRLs are published where.

2.4.4. Revoking Certificates and Checking Status

End entities can request that their own certificates be revoked. When an end entity makes the request, the certificate has to be presented to the CA. If the certificate and the keys are available, the request is processed and sent to the Certificate Manager, and the certificate is revoked. The Certificate Manager marks the certificate as revoked in its database and adds it to any applicable CRLs.

An agent can revoke any certificate issued by the Certificate Manager by searching for the certificate in the agent services interface and then marking it revoked. Once a certificate is revoked, it is marked revoked in the database and in the publishing directory, if the Certificate is set up for publishing.

If the internal OCSP service has been configured, the service determines the status of certificates by looking them up in the internal database.

Automated notifications can be set to send email messages to end entities when their certificates are revoked by enabling and configuring the certificate revoked notification message.

2.4.4.1. Revoking Certificates

Users can revoke their certificates using the **CMCRequest** utility on the command line. For details, see the *Performing a CMC Revocation* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

2.4.4.2. Certificate Status

2.4.4.2.1. CRLs

The Certificate System can create certificate revocation lists (CRLs) from a configurable framework which allows user-defined issuing points so a CRL can be created for each issuing point. Delta CRLs can also be created for any issuing point that is defined. CRLs can be issued for each type of certificate, for a specific subset of a type of certificate, or for certificates generated according to a profile or list of profiles. The extensions used and the frequency and intervals when CRLs are published can all be configured.

The Certificate Manager issues X.509-standard CRLs. A CRL can be automatically updated whenever a certificate is revoked or at specified intervals.

2.4.4.2.2. OCSP Services

The Certificate System CA supports the Online Certificate Status Protocol (OCSP) as defined in PKIX standard RFC 2560. The OCSP protocol enables OCSP-compliant applications to determine the state of a certificate, including the revocation status, without having to directly check a CRL published by a CA to the validation authority. The validation authority, which is also called an *OCSP responder*, checks for the application.

1. A CA is set up to issue certificates that include the Authority Information Access extension, which identifies an OCSP responder that can be queried for the status of the certificate.

2. The CA periodically publishes CRLs to an OCSP responder.

3. The OCSP responder maintains the CRL it receives from the CA.

4. An OCSP-compliant client sends requests containing all the information required to identify the certificate to the OCSP responder for verification. The applications determine the location of the OCSP responder from the value of the Authority Information Access extension in the certificate being validated.

5. The OCSP responder determines if the request contains all the information required to process it. If it does not or if it is not enabled for the requested service, a rejection notice is sent. If it does have enough information, it processes the request and sends back a report stating the status of the certificate.

2.4.4.2.2.1. OCSP Response Signing

Every response that the client receives, including a rejection notification, is digitally signed by the responder; the client is expected to verify the signature to ensure that the response came from the

responder to which it submitted the request. The key the responder uses to sign the message depends on how the OCSP responder is deployed in a PKI setup. RFC 2560 recommends that the key used to sign the response belong to one of the following:

- The CA that issued the certificate whose status is being checked.

- A responder with a public key trusted by the client. Such a responder is called a *trusted responder*.

- A responder that holds a specially marked certificate issued to it directly by the CA that revokes the certificates and publishes the CRL. Possession of this certificate by a responder indicates that the CA has authorized the responder to issue OCSP responses for certificates revoked by the CA. Such a responder is called a *CA-designated responder* or a *CA-authorized responder*.

The end-entities page of a Certificate Manager includes a form for manually requesting a certificate for the OCSP responder. The default enrollment form includes all the attributes that identify the certificate as an OCSP responder certificate. The required certificate extensions, such as OCSPNoCheck and Extended Key Usage, can be added to the certificate when the certificate request is submitted.

2.4.4.2.2.2. OCSP Responses

The OCSP response that the client receives indicates the current status of the certificate as determined by the OCSP responder. The response could be any of the following:

- *Good or Verified* . Specifies a positive response to the status inquiry, meaning the certificate has not been revoked. It does not necessarily mean that the certificate was issued or that it is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the responder regarding the status of the certificate.

- *Revoked* . Specifies that the certificate has been revoked, either permanently or temporarily.

Based on the status, the client decides whether to validate the certificate.

> **NOTE**
>
> The OCSP responder will never return a response of *Unknown*. The response will always be either *Good* or *Revoked*.

2.4.4.2.2.3. OCSP Services

There are two ways to set up OCSP services:

- The OCSP built into the Certificate Manager

- The Online Certificate Status Manager subsystem

In addition to the built-in OCSP service, the Certificate Manager can publish CRLs to an OCSP-compliant validation authority. CAs can be configured to publish CRLs to the Certificate System Online Certificate Status Manager. The Online Certificate Status Manager stores each Certificate Manager's CRL in its internal database and uses the appropriate CRL to verify the revocation status of a certificate when queried by an OCSP-compliant client.

The Certificate Manager can generate and publish CRLs whenever a certificate is revoked and at specified intervals. Because the purpose of an OCSP responder is to facilitate immediate verification of certificates, the Certificate Manager should publish the CRL to the Online Certificate Status Manager

every time a certificate is revoked. Publishing only at intervals means that the OCSP service is checking an outdated CRL.

> **NOTE**
>
> If the CRL is large, the Certificate Manager can take a considerable amount of time to publish the CRL.

The Online Certificate Status Manager stores each Certificate Manager's CRL in its internal database and uses it as the CRL to verify certificates. The Online Certificate Status Manager can also use the CRL published to an LDAP directory, meaning the Certificate Manager does not have to update the CRLs directly to the Online Certificate Status Manager.

2.4.5. Archiving, Recovering, and Rotating Keys

To archive private encryption keys and recover them later, the PKI configuration must include the following elements:

- Clients that can generate keys and CSRs in CRMF format. For details, see the *Creating Certificate Signing Requests* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

- Clients that can transform the CRMF request into CMC, such as the **CMCRequest** utility.

- Clients that can submit the CMC request with embedded CRMF request to the CA through a specified enrollment profile.

- An installed and configured KRA.

Only keys that are used exclusively for encrypting data should be archived; signing keys in particular should never be archived. Having two copies of a signing key makes it impossible to identify with certainty who used the key; a second archived copy could be used to impersonate the digital identity of the original key owner.

2.4.5.1. Archiving Keys

The KRA automatically archives private encryption keys if archiving is configured.

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.

There are some common situations when it is necessary to recover encryption keys:

- An employee loses the private encryption key and cannot read encrypted mail messages.

- An employee is on an extended leave, and someone needs to access an encrypted document.

- An employee leaves the company, and company officials need to perform an audit that requires gaining access to the employee's encrypted mail.

The KRA stores private encryption keys in a secure key repository in its internal database; each key is encrypted and stored as a key record and is given a unique key identifier.

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. The key recovery agents have the privilege to insert, delete, and search for key records.

- When the key recovery agents search by the key ID, only the key that corresponds to that ID is returned.

- When the agents search by user name, all stored keys belonging to that owner are returned.

- When the agents search by the public key in a certificate, only the corresponding private key is returned.

When a Certificate Manager receives a certificate request that contains the key archival option, it automatically forwards the request to the KRA to archive the encryption key. The private key is encrypted by the transport key, and the KRA receives the encrypted copy and stores the key in its key repository. To archive the key, the KRA uses two special key pairs:

- A transport key pair and corresponding certificate.

- A storage key pair.

Figure 2.2, "How the Key Archival Process Works" illustrates how the key archival process occurs when an end entity requests a certificate.
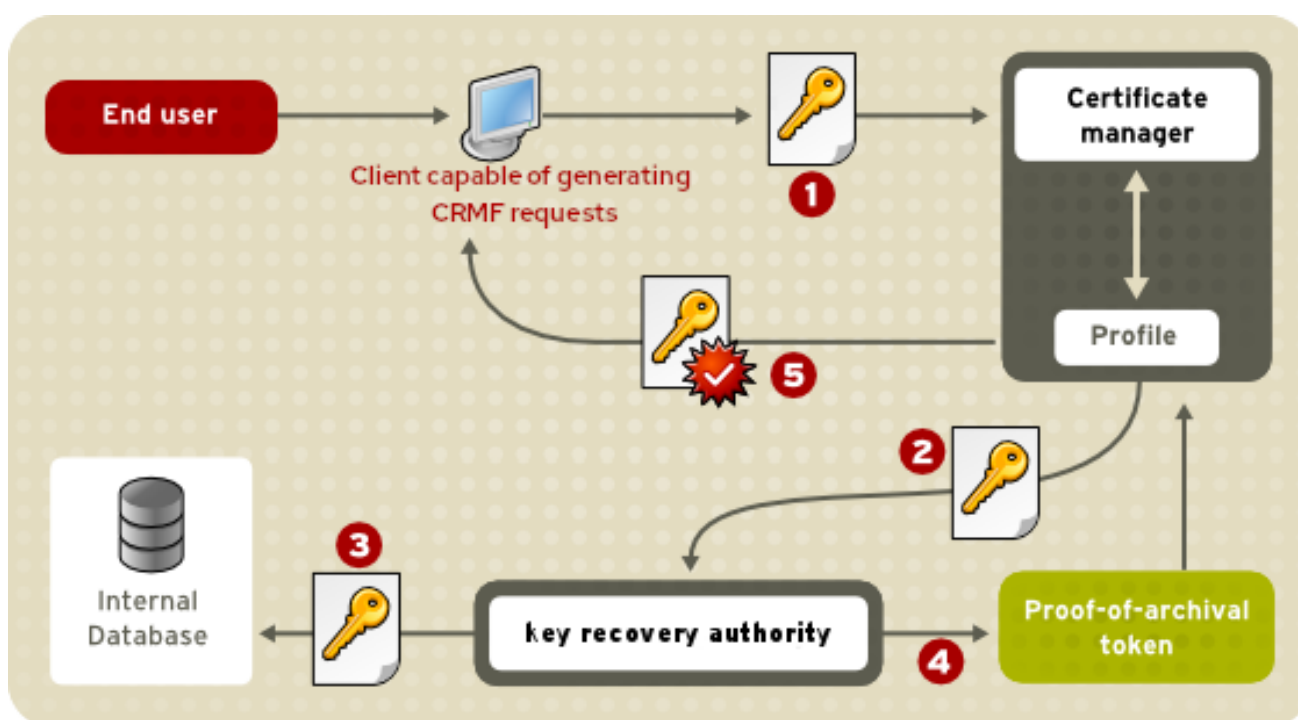
Figure 2.2. How the Key Archival Process Works

1. The client generates a CRMF request and submits it through the CA's enrollment portal.

   For more information on this procedure, see *Example on Obtaining an Encryption-only certificate with Key Archival* procedure in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

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 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 two certificates for the signing and encryption key pairs and returns them to the end entity.

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.

2.4.5.2. Recovering Keys

The KRA supports *agent-initiated key recovery*. Agent-initiated recovery is 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.

Through the KRA agent services page, key recovery agents can collectively authorize and retrieve private encryption keys and associated certificates in a PKCS #12 package, which can then be imported into the client.

In key recovery authorization, one of the key recovery agents informs all required recovery agents about an impending key recovery. All recovery agents access the KRA key recovery page. One of the agents initiates the key recovery process. The KRA returns a notification to the agent includes a recovery authorization reference number identifying the particular key recovery request that the agent is required to authorize. Each agent uses the reference number and authorizes key recovery separately.

KRA supports *Asynchronous recovery*, meaning that each step of the recovery process — the initial request and each subsequent approval or rejection — is stored in the KRA's internal LDAP database, under the key entry. The status data for the recovery process can be retrieved even if the original browser session is closed or the KRA is shut down. Agents can search for the key to recover, without using a reference number.

This asynchronous recovery option is illustrated in Figure 2.3, "Asynchronous Recovery".

Figure 2.3. Asynchronous Recovery

The KRA informs the agent who initiated the key recovery process of the status of the authorizations.

When all of the authorizations are entered, the KRA checks the information. If the information presented is correct, it retrieves the requested key and returns it along with the corresponding certificate in the form of a PKCS #12 package to the agent who initiated the key recovery process.

> **WARNING**
>
> The PKCS #12 package contains the private key. To minimize the risk of key compromise, the recovery agent must use a secure method to deliver the PKCS #12 package and password to the key recipient. The agent should use a good password to encrypt the PKCS #12 package and set up an appropriate delivery mechanism.

The *key recovery agent scheme* configures the KRA to recognize to which group the key recovery agents belong and specifies how many of these agents are required to authorize a key recovery request before the archived key is restored.

Certificate System uses an *m-of-n ACL-based recovery scheme*, rather than a secret-splitting-based recovery scheme. Its existing access control scheme ensures that 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.

> **IMPORTANT**
>
> The above information refers to using a web browser, such as Firefox. However, functionality critical to KRA usage is no longer included in Firefox version 31.6 that was released on Red Hat Enterprise Linux 7 platforms. In such cases, it is necessary to use the **pki** utility to replicate this behavior. For more information, see the pki(1) and pki-key(1) man pages.

Apart from storing symmetric keys, KRA can also store secrets similar to symmetric keys, such as volume encryption secrets, or even passwords and passphrases. The **pki** utility supports options that enable storing and retrieving these other types of secrets.

2.4.5.3. KRA Transport Key Rotation

KRA transport rotation allows for seamless transition between CA and KRA subsystem instances using a current and a new transport key. This allows KRA transport keys to be periodically rotated for enhanced security by allowing both old and new transport keys to operate during the time of the transition; individual subsystem instances take turns being configured while other clones continue to serve with no downtime.

In the KRA transport key rotation process, a new transport key pair is generated, a certificate request is submitted, and a new transport certificate is retrieved. The new transport key pair and certificate have to be included in the KRA configuration to provide support for the second transport key. Once KRA supports two transport keys, administrators can start transitioning CAs to the new transport key. KRA support for the old transport key can be removed once all CAs are moved to the new transport key.

To configure KRA transport key rotation:

1. Generate a new KRA transport key and certificate

2. Transfer the new transport key and certificate to KRA clones

3. Update the CA configuration with the new KRA transport certificate

4. Update the KRA configuration to use only the new transport key and certificate

After this, the rotation of KRA transport certificates is complete, and all the affected CAs and KRAs use the new KRA certificate only. For more information on how to perform the above steps, see the procedures below.

Generating the new KRA transport key and certificate

1. Request the KRA transport certificate.

   a. Stop the KRA:

   ```
   systemctl stop pki-tomcatd@pki-kra.service
   ```

   OR

> systemctl stop pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

b. Go to the KRA NSS database directory:

> cd /etc/pki/pki-kra/alias

c. Create a subdirectory and save all the NSS database files into it. For example:

> mkdir *nss_db_backup*
> cp *.db *nss_db_backup*

d. Create a new request by using the **PKCS10Client** utility. For example:

> PKCS10Client -p *password* -d '.' -o '*req.txt*' -n 'CN=KRA Transport 2 Certificate,O=example.com Security Domain'

Alternatively, use the **certutil** utility. For example:

> certutil -d . -R -k rsa -g 2048 -s 'CN=KRA Transport 2 Certificate,O=example.com Security Domain' -f *password-file* -a -o *transport-certificate-request-file*

e. Submit the transport certificate request on the **Manual Data Recovery Manager Transport Certificate Enrollment** page of the CA End-Entity page.

f. Wait for the agent approval of the submitted request to retrieve the certificate by checking the request status on the End-Entity retrieval page.

2. Approve the KRA transport certificate through the CA Agent Services interface.

3. Retrieve the KRA transport certificate.

a. Go to the KRA NSS database directory:

> cd /etc/pki/pki-kra/alias

b. Wait for the agent approval of the submitted request to retrieve the certificate by checking the request status on the End-Entity retrieval page.

c. Once the new KRA transport certificate is available, paste its Base64-encoded value into a text file, for example a file named **cert-serial_number.txt**. Do not include the header (**-----BEGIN CERTIFICATE-----**) or the footer (**-----END CERTIFICATE-----**).

4. Import the KRA transport certificate.

a. Go to the KRA NSS database directory:

> cd /etc/pki/pki-kra/alias

b. Import the transport certificate into the KRA NSS database:

> certutil -d . -A -n '*transportCert-serial_number* cert-pki-kra KRA' -t 'u,u,u' -a -i *cert-serial_number*.txt

5. Update the KRA transport certificate configuration.

   a. Go to the KRA NSS database directory:

   > cd /etc/pki/pki-kra/alias

   b. Verify that the new KRA transport certificate is imported:

   > certutil -d . -L
   > certutil -d . -L -n '*transportCert-serial_number* cert-pki-kra KRA'

   c. Open the **/var/lib/pki/pki-kra/kra/conf/CS.cfg** file and add the following line:

   > kra.transportUnit.newNickName=*transportCert-serial_number* cert-pki-kra KRA

Propagating the new transport key and certificate to KRA clones

1. Start the KRA:

   > systemctl start pki-tomcatd@pki-kra.service

   OR

   > systemctl start pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

2. Extract the new transport key and certificate for propagation to clones.

   a. Go to the KRA NSS database directory:

   > cd /etc/pki/pki-kra/alias

   b. Stop the KRA:

   > systemctl stop pki-tomcatd@pki-kra.service

   OR

   > systemctl stop pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

   c. Verify that the new KRA transport certificate is present:

   > certutil -d . -L
   > certutil -d . -L -n '*transportCert-serial_number* cert-pki-kra KRA'

   d. Export the KRA new transport key and certificate:

> pk12util -o transport.p12 -d . -n '*transportCert-serial_number* cert-pki-kra KRA'

e. Verify the exported KRA transport key and certificate:

> pk12util -l transport.p12

3. Perform these steps on each KRA clone:

   a. Copy the **transport.p12** file, including the transport key and certificate, to the KRA clone location.

   b. Go to the clone NSS database directory:

   > cd /etc/pki/pki-kra/alias

   c. Stop the KRA clone:

   > systemctl stop pki-tomcatd@pki-kra.service

   OR

   > systemctl stop pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

   d. Check the content of the clone NSS database:

   > certutil -d . -L

   e. Import the new transport key and certificate of the clone:

   > pk12util -i transport.p12 -d .

   f. Add the following line to the **/var/lib/pki/pki-kra/kra/conf/CS.cfg** file on the clone:

   > kra.transportUnit.newNickName=*transportCert-serial_number* cert-pki-kra KRA

   g. Start the KRA clone:

   > systemctl start pki-tomcatd@pki-kra.service

   OR

   > systemctl start pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

Updating the CA configuration with the new KRA transport certificate

1. Format the new KRA transport certificate for inclusion in the CA.

   a. Obtain the **_cert-serial_number_.txt** KRA transport certificate file created when retrieving the KRA transport certificate in the previous procedure.

b. Convert the Base64–encoded certificate included in ***cert-serial_number*.txt** to a single–line file:

> tr -d '\n' < *cert-serial_number*.txt > *cert-one-line-serial_number*.txt

2. Do the following for the CA and all its clones corresponding to the KRA above:

   a. Stop the CA:

   > systemctl stop pki-tomcatd@pki-ca.service

   OR

   > systemctl stop pki-tomcatd-nuxwdog@pki-ca.service (if using the nuxwdog watchdog)

   b. In the **/var/lib/pki/pki-ca/ca/conf/CS.cfg** file, locate the certificate included in the following line:

   > ca.connector.KRA.transportCert=*certificate*

   Replace that certificate with the one contained in ***cert-one-line-serial_number*.txt**.

   c. Start the CA:

   > systemctl start pki-tomcatd@pki-ca.service

   OR

   > systemctl start pki-tomcatd-nuxwdog@pki-ca.service (if using the nuxwdog watchdog)

> **NOTE**
>
> While the CA and all its clones are being updated with the new KRA transport certificate, the CA instances that have completed the transition use the new KRA transport certificate, and the CA instances that have not yet been updated continue to use the old KRA transport certificate. Because the corresponding KRA and its clones have already been updated to use both transport certificates, no downtime occurs.

Updating the KRA configuration to use only the new transport key and certificate

For the KRA and each of its clones, do the following:

1. Go to the KRA NSS database directory:

   > cd /etc/pki/pki-kra/alias

2. Stop the KRA:

   > systemctl stop pki-tomcatd@pki-kra.service

   OR

> systemctl stop pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

3. Verify that the new KRA transport certificate is imported:

> certutil -d . -L
> certutil -d . -L -n '*transportCert-serial_number* cert-pki-kra KRA'

4. Open the **/var/lib/pki/pki-kra/kra/conf/CS.cfg** file, and look for the ***nickName*** value included in the following line:

> kra.transportUnit.nickName=*transportCert* cert-pki-kra KRA

Replace the ***nickName*** value with the ***newNickName*** value included in the following line:

> kra.transportUnit.newNickName=*transportCert-serial_number* cert-pki-kra KRA

As a result, the **CS.cfg** file includes this line:

> kra.transportUnit.nickName=*transportCert-serial_number* cert-pki-kra KRA

5. Remove the following line from **/var/lib/pki/pki-kra/kra/conf/CS.cfg**:

> kra.transportUnit.newNickName=*transportCert-serial_number* cert-pki-kra KRA

6. Start the KRA:

> systemctl start pki-tomcatd@pki-kra.service

OR

> systemctl start pki-tomcatd-nuxwdog@pki-kra.service (if using the nuxwdog watchdog)

2.5. SMART CARD TOKEN MANAGEMENT WITH CERTIFICATE SYSTEM

A *smart card* is a hardware cryptographic device containing cryptographic certificates and keys. It can be employed by the user to participate in operations such as secure website access and secure mail. It can also serve as an authentication device to log in to various operating systems such as Red Hat Enterprise Linux. The management of these cards or tokens throughout their entire lifetime in service is accomplished by the *Token Management System* (TMS).

A TMS environment requires a Certificate Authority (CA), Token Key Service (TKS), and Token Processing System (TPS), with an optional Key Recovery Authority (KRA) for server–side key generation and key archival and recovery. Online Certificate Status Protocol (OCSP) can also be used to work with the CA to serve online certificate status requests. This chapter provides an overview of the TKS and TPS systems, which provide the smart card management functions of Red Hat Certificate System, as well as Enterprise Security Client (ESC), that works with TMS from the user end.
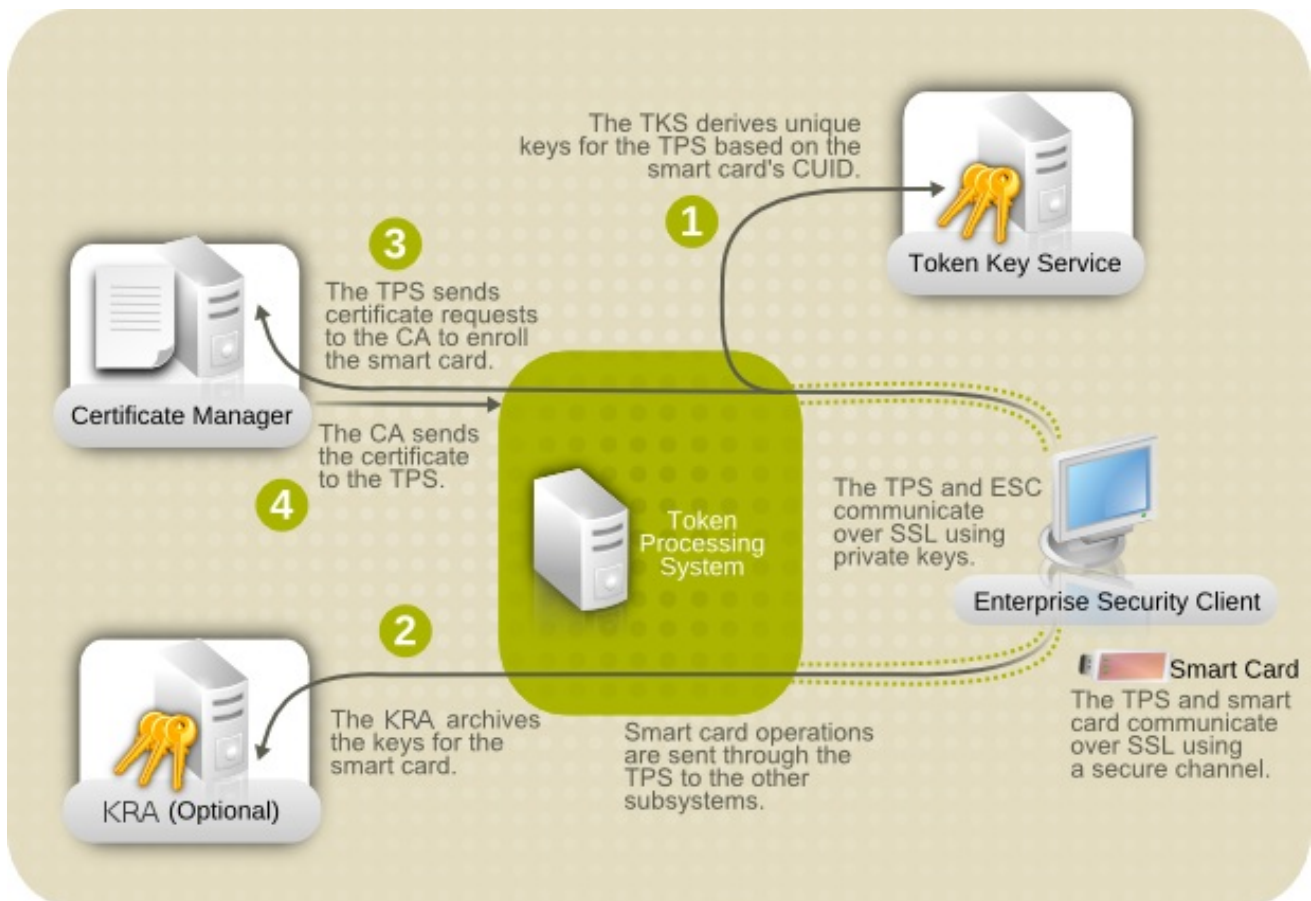
Figure 2.4. How the TMS Manages Smart Cards

2.5.1. Token Key Service (TKS)

The Token Key Service (TKS) is responsible for managing one or more master keys. It maintains the **master keys** and is the only entity within the TMS that has access to the key materials. In an operational environment, each valid smart card token contains a set of symmetric keys that are derived from both the master key and the ID that is unique to the card (CUID).

Initially, a default (unique only per manufacturer master key) set of symmetric keys is initialized on each smart card by the manufacturer. This default set should be changed at the deployment site by going through a **Key Changeover** operation to generate the new master key on TKS. As the sole owner to the master key, when given the CUID of a smart card, TKS is capable of deriving the set of symmetric keys residing on that particular smart card, which would then allow TKS to establish a session-based **Secure Channel** for secure communication between TMS and each individual smart card.

> **NOTE**
>
> Because of the sensitivity of the data that the TKS manages, the TKS should be set behind the firewall with restricted access.

2.5.1.1. Master Keys and Key Sets

The TKS supports multiple smart card key sets. Each smart card vendor creates different default (developer) static key sets for their smart card token stocks, and the TKS is equipped with the static key set (per manufacturer) to kickstart the format process of a blank token.

During the format process of a blank smart card token, a Java applet and the uniquely derived symmetric key set are injected into the token. Each master key (in some cases referred to as **keySet**)

that the TKS supports is to have a set of entries in the TKS configuration file (**CS.cfg**). Each TPS profile contains a configuration to direct its enrollment to the proper TKS keySet for the matching key derivation process that would essentially be responsible for establishing the Secure Channel secured by a set of session-specific keys between TMS and the smart card token.

On TKS, master keys are defined by named keySets for references by TPS. On TPS, depending on the enrollment type (internal or external registration), The keySet is either specified in the TPS profile, or determined by the keySet Mapping Resolver.

2.5.1.2. Key Ceremony (Shared Key Transport)

A **Key Ceremony** is a process for transporting highly sensitive keys in a secure way from one location to another. In one scenario, in a highly secure deployment environment, the master key can be generated in a secure vault with no network to the outside. Alternatively, an organization might want to have TKS and TPS instances on different physical machines. In either case, under the assumption that no one single person is to be trusted with the key, Red Hat Certificate System TMS provides a utility called **tkstool** to manage the secure key transportation.

2.5.1.3. Key Update (Key Changeover)

When Global Platform-compliant smart cards are created at the factory, the manufacturer will burn a set of default symmetric keys onto the token. The TKS is initially configured to use these symmetric keys (one KeySet entry per vendor in the TKS configuration). However, since these symmetric keys are not unique to the smart cards from the same stock, and because these are well-known keys, it is strongly encouraged to replace these symmetric keys with a set that is unique per token, not shared by the manufacturer, to restrict the set of entities that can manipulate the token.

The changing over of the keys takes place with the assistance of the Token Key Service subsystem. One of the functions of the TKS is to oversee the Master Keys from which the previously discussed smart card token keys are derived. There can be more than one master key residing under the control of the TKS.

> **IMPORTANT**
>
> When this key changeover process is done on a token, the token may become unusable in the future since it no longer has the default key set enabled. The key is essentially only as good as long as the TPS and TKS system that provisioned the token is valid. Because of this, it is essential to keep all the master keys, even if any of them are outdated.
>
> You can disable the old master keys in TKS for better control, but do not delete them unless disabled tokens are part of your plan. There is support to revert the token keys back to the original key set, which is viable if the token is to be reused again in some sort of a testing scenario.

2.5.1.4. APDUs and Secure Channels

The Red Hat Certificate System Token Management System (TMS) supports the *GlobalPlatform* smart card specification, in which the Secure Channel implementation is done with the Token Key System (TKS) managing the master key and the Token Processing System (TPS) communicating with the smart card (tokens) with *Application Protocol Data Units* (APDUs).

There are two types of APDUs:

- **Command APDUs**, sent by the TPS to smart cards

- **Response APDUs**, sent by smart cards to the TPS as response to command APDUs

The initiation of the APDU commands may be triggered when clients take action and connect to the Certificate System server for requests. A secure channel begins with an **InitializeUpdate** APDU sent from TPS to the smart card token, and is fully established with the **ExternalAuthenticate** APDU. Then, both the token and TMS would have established a set of shared secrets, called session keys, which are used to encrypt and authenticate the communication. This authenticated and encrypted communication channel is called Secure Channel.

Because TKS is the only entity that has access to the master key which is capable of deriving the set of unique symmetric on-token smart card keys, the Secure Channel provides the adequately safeguarded communication between TMS and each individual token. Any disconnection of the channel will require reestablishment of new session keys for a new channel.

2.5.2. Token Processing System (TPS)

The Token Processing System (TPS) is a registration authority for smart card certificate enrollment. It acts as a conduit between the user-centered Enterprise Security Client (ESC), which interacts with client side smart card tokens, and the Certificate System back end subsystems, such as the Certificate Authority (CA) and the Key Recovery Authority (KRA).

In TMS, the TPS is required in order to manage smart cards, as it is the only TMS entity that understands the APDU commands and responses. TPS sends commands to the smart cards to help them generate and store keys and certificates for a specific entity, such as a user or device. Smart card operations go through the TPS and are forwarded to the appropriate subsystem for action, such as the CA to generate certificates or the KRA to generate, archive, or recover keys.

2.5.2.1. Coolkey Applet

Red Hat Certificate System includes the **Coolkey** Java applet, written specifically to run on TMS-supported smart card tokens. The **Coolkey** applet connects to a PKCS#11 module that handles the certificate and key related operations. During a token format operation, this applet is injected onto the smart card token using the Secure Channel protocol, and can be updated per configuration.

2.5.2.2. Token Operations

The TPS in Red Hat Certificate System is available to provision smart cards on the behalf of end users of the smart cards. The Token Processing System provides support for the following major token operations:

- **Token Format** – The format operation is responsible for installing the proper Coolkey applet onto the token. The applet provides a platform where subsequent cryptographic keys and certificates can be later placed.

- **Token Enrollment** – The enrollment operation results in a smart card populated with required cryptographic keys and cryptographic certificates. This material allows the user of the smart card to participate in operations such as secure web site access and secure mail. Two types of enrollments are supported, which is configured globally:

  - **Internal Registration** - Enrollment by TPS profiles determined by the profile **Mapping Resolver**.

  - **External Registration** - Enrollment by TPS profiles determined by the entries in the user's LDAP record.

- **Token PIN Reset** – The token PIN reset operation allows the user of the token to specify a new PIN that is used to log into the token, making it available for performing cryptographic operations.

The following other operations can be considered supplementary or inherent operations to the main ones listed above. They can be triggered per relevant configuration or by the state of the token.

- **Key Generation** – Each PKI certificate is comprised of a public/private key pair. In Red Hat Certificate System, the generation of the keys can be done in two ways, depending on the **TPS profile** configuration:

  - **Token Side Key Generation** – The PKI key pairs are generated on the smart card token. Generating the key pairs on the token side does *not* allow for key archival.

  - **Server Side Key Generation** – The PKI key pairs are generated on the TMS server side. The key pairs are then sent back to the token using Secure Channel. Generating the key pairs on the server side allows for key archival.

- **Certificate Renewal** – This operation allows a previously enrolled token to have the certificates currently on the token reissued while reusing the same keys. This is useful in situations where the old certificates are due to expire and you want to create new ones but maintain the original key material.

- **Certificate Revocation** – Certificate revocation can be triggered based on TPS profile configuration or based on token state.

  Normally, only the CA which issued a certificate can revoke it, which could mean that retiring a CA would make it impossible to revoke certain certificates. However, it is possible to route revocation requests for tokens to the retired CA while still routing all other requests such as enrollment to a new, active CA. This mechanism is called **Revocation Routing**.

- **Token Key Changeover** – The key changeover operation, triggered by a format operation, results in the ability to change the internal keys of the token from the default developer key set to a new key set controlled by the deployer of the Token Processing System. This is usually done in any real deployment scenario since the developer key set is better suited to testing situations.

- **Applet Update** – During the course of a TMS deployment, the Coolkey smart card applet can be updated or downgraded if required.

2.5.2.3. TPS Profiles

The Certificate System Token Processing System subsystem facilitates the management of smart card tokens. Tokens are provisioned by the TPS such that they are taken from a blank state to either a Formatted or Enrolled condition. A Formatted token is one that contains the **CoolKey** applet supported by TPS, while an Enrolled token is personalized (a process called *binding*) to an individual with the requisite certificates and cryptographic keys. This fully provisioned token is ready to use for crytptographic operations.

The TPS can also manage *Profiles*. The notion of a token Profile is related to:

- The steps taken to Format or Enroll a token.

- The attributes contained within the finished token after the operation has been successfully completed.

The following list contains some of the quantities that make up a unique token profile:

- How does the TPS connect to the user's authentication LDAP database?

- Will user authentication be required for this token operation? If so, what authentication manager will be used?

- How does the TPS connect to a Certificate System CA from which it will obtain certificates?

- How are the private and public keys generated on this token? Are they generated on the token side or on the server side?

- What key size (in bits) is to be used when generating private and public keys?

- Which certificate enrollment profile (provisioned by the CA) is to be used to generate the certificates on this token?

> **NOTE**
>
> This setting will determine the final structure of the certificates to be written to the token. Different certificates can be created for different uses, based on extensions included in the certificate. For example, one certificate can specialize in data encryption, and another one can be used for signature operations.

- What version of the Coolkey applet will be required on the token?

- How many certificates will be placed on this token for an enrollment operation?

These above and many others can be configured for each token type or profile. A full list of available configuration options is available in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Another question to consider is how a given token being provisioned by a user will be mapped to an individual token profile. There are two types of registration:

- **Internal Registration** – In this case, the TPS profile (**tokenType**) is determined by the profile *Mapping Resolver*. This filter-based resolver can be configured to take any of the data provided by the token into account and determine the target profile.

- **External Registration** – When using external registration, the profile (in name only – actual profiles are still defined in the TPS in the same fashion as those used by the internal registration) is specified in each user's LDAP record, which is obtained during authentication. This allows the TPS to obtain key enrollment and recovery information from an external registration Directory Server where user information is stored. This gives you the control to override the enrollment, revocation, and recovery policies that are inherent to the TPS internal registration mechanism. The user LDAP record attribute names relevant to external registration are configurable.

  External registration can be useful when the concept of a "group certificate" is required. In that case, all users within a group can have a special record configured in their LDAP profiles for downloading a shared certificate and keys.

The registration to be used is configured globally per TPS instance.

2.5.2.4. Token Database

The Token Processing System makes use of the LDAP token database store, which is used to keep a list of active tokens and their respective certificates, and to keep track of the current state of each token. A brand new token is considered *Uninitialized*, while a fully enrolled token is *Enrolled*. This data store is

constantly updated and consulted by the TPS when processing tokens.

2.5.2.4.1. Token States and Transitions

The Token Processing System stores states in its internal database in order to determine the current token status as well as actions which can be performed on the token.

2.5.2.4.1.1. Token States

The following table lists all possible token states:

Table 2.9. Possible Token States

Name	Code	Label
FORMATTED	0	Formatted (uninitialized)
DAMAGED	1	Physically damaged
PERM_LOST	2	Permanently lost
SUSPENDED	3	Suspended (temporarily lost)
ACTIVE	4	Active
TERMINATED	6	Terminated
UNFORMATTED	7	Unformatted

The command line interface displays token states using the Name listed above. The graphical interface uses the Label instead.

> **NOTE**
>
> The above table contains no state with code **5**, which previously belonged to a state that was removed.

2.5.2.4.1.2. Token State Transitions Done Using the Graphical or Command Line Interface

Each token state has a limited amount of next states it can transition into. For example, a token can change state from **FORMATTED** to **ACTIVE** or **DAMAGED**, but it can never transition from **FORMATTED** to **UNFORMATTED**.

Furthermore, the list of states a token can transition into is different depending on whether the transition is triggered manually using a command line or the graphical interface, or automatically using a token operation. The list of allowed manual transitions is stored in the **tokendb.allowedTransitions** property, and the **tps.operations.allowedTransitions** property controls allowed transitions triggered by token operations.

The default configurations for both manual and token operation-based transitions are stored in the **/usr/share/pki/tps/conf/CS.cfg** configuration file.

2.5.2.4.1.2.1. Token State Transitions Using the Command Line or Graphical Interface

All possible transitions allowed in the command line or graphical interface are described in the TPS configuration file using the **tokendb.allowedTransitions** property:

tokendb.allowedTransitions=0:1,0:2,0:3,0:6,3:2,3:6,4:1,4:2,4:3,4:6,6:7

The property contains a comma-separated list of transitions. Each transition is written in the format of ***\<current code>:\<new code>***. The codes are described in Table 2.9, "Possible Token States". The default configuration is preserved in **/usr/share/pki/tps/conf/CS.cfg**.

The following table describes each possible transition in more detail:

Table 2.10. Possible Manual Token State Transitions

Transition	Current State	Next State	Description
0:1	FORMATTED	DAMAGED	This token has been physically damaged.
0:2	FORMATTED	PERM_LOST	This token has been permanently lost.
0:3	FORMATTED	SUSPENDED	This token has been suspended (temporarily lost).
0:6	FORMATTED	TERMINATED	This token has been terminated.
3:2	SUSPENDED	PERM_LOST	This suspended token has been permanently lost.
3:6	SUSPENDED	TERMINATED	This suspended token has been terminated.
4:1	ACTIVE	DAMAGED	This token has been physically damaged.
4:2	ACTIVE	PERM_LOST	This token has been permanently lost.
4:3	ACTIVE	SUSPENDED	This token has been suspended (temporarily lost).
4:6	ACTIVE	TERMINATED	This token has been terminated.
6:7	TERMINATED	UNFORMATTED	Reuse this token.

The following transitions are generated automatically depending on the token's original state. If a token was originally **FORMATTED** and then became **SUSPENDED**, it can only return to the **FORMATTED** state. If a token was originally **ACTIVE** and then became **SUSPENDED**, it can only return to the **ACTIVE** state.

Table 2.11. Token State Transitions Triggered Automatically

Transition	Current State	Next State	Description
3:0	SUSPENDED	FORMATTED	This suspended (temporarily lost) token has been found.
3:4	SUSPENDED	ACTIVE	This suspended (temporarily lost) token has been found.

2.5.2.4.1.3. Token State Transitions using Token Operations

All possible transitions that can be done using token operations are described in the TPS configuration file using the **tokendb.allowedTransitions** property:

```
tps.operations.allowedTransitions=0:0,0:4,4:4,4:0,7:0
```

The property contains a comma-separated list of transitions. Each transition is written in the format of *<current code>:<new code>*. The codes are described in Table 2.9, "Possible Token States". The default configuration is preserved in **/usr/share/pki/tps/conf/CS.cfg**.

The following table describes each possible transition in more detail:

Table 2.12. Possible Token State Transitions using Token Operations

Transition	Current State	Next State	Description
0:0	FORMATTED	FORMATTED	This allows reformatting a token or upgrading applet/key in a token.
0:4	FORMATTED	ACTIVE	This allows enrolling a token.
4:4	ACTIVE	ACTIVE	This allows re-enrolling an active token. May be useful for external registration.
4:0	ACTIVE	FORMATTED	This allows formatting an active token.

Transition	Current State	Next State	Description
7:0	UNFORMATTED	FORMATTED	This allows formatting a blank or previously used token.

2.5.2.4.1.4. Token State and Transition Labels

The default labels for token states and transitions are stored in the **/usr/share/pki/tps/conf/token-states.properties** configuration file. By default, the file has the following contents:

```
# Token states
UNFORMATTED         = Unformatted
FORMATTED           = Formatted (uninitialized)
ACTIVE              = Active
SUSPENDED           = Suspended (temporarily lost)
PERM_LOST           = Permanently lost
DAMAGED             = Physically damaged
TEMP_LOST_PERM_LOST = Temporarily lost then permanently lost
TERMINATED          = Terminated

# Token state transitions
FORMATTED.DAMAGED        = This token has been physically damaged.
FORMATTED.PERM_LOST      = This token has been permanently lost.
FORMATTED.SUSPENDED      = This token has been suspended (temporarily lost).
FORMATTED.TERMINATED     = This token has been terminated.
SUSPENDED.ACTIVE         = This suspended (temporarily lost) token has been found.
SUSPENDED.PERM_LOST      = This suspended (temporarily lost) token has become permanently
lost.
SUSPENDED.TERMINATED     = This suspended (temporarily lost) token has been terminated.
SUSPENDED.FORMATTED      = This suspended (temporarily lost) token has been found.
ACTIVE.DAMAGED           = This token has been physically damaged.
ACTIVE.PERM_LOST         = This token has been permanently lost.
ACTIVE.SUSPENDED         = This token has been suspended (temporarily lost).
ACTIVE.TERMINATED        = This token has been terminated.
TERMINATED.UNFORMATTED   = Reuse this token.
```

2.5.2.4.1.5. Customizing Allowed Token State Transitions

To customize the list of token state transition, edit the following properties in **/var/lib/pki/*instance_name*/tps/conf/CS.cfg**:

- **tokendb.allowedTransitions** to customize the list of allowed transitions performed using the command line or graphical interface

- **tps.operations.allowedTransitions** to customize the list of allowed transitions using token operations

Transitions can be removed from the default list if necessary, but new transitions cannot be added unless they were in the default list. The defaults are stored in **/usr/share/pki/tps/conf/CS.cfg**.

2.5.2.4.1.6. Customizing Token State and Transition Labels

To customize token state and transition labels, copy the default **/usr/share/pki/tps/conf/token-states.properties** into your instance folder (**/var/lib/pki/***instance_name/tps/conf/CS.cfg*), and change the labels listed inside as needed.

Changes will be effective immediately, the server does not need to be restarted. The TPS user interface may require a reload.

To revert to default state and label names, delete the edited **token-states.properties** file from your instance folder.

2.5.2.4.1.7. Token Activity Log

Certain TPS activities are logged. Possible events in the log file are listed in the table below.

Table 2.13. TPS Activity Log Events

Activity	Description
add	A token was added.
format	A token was formatted.
enrollment	A token was enrolled.
recovery	A token was recovered.
renewal	A token was renewed.
pin_reset	A token PIN was reset.
token_status_change	A token status was changed using the command line or graphical interface.
token_modify	A token was modified.
delete	A token was deleted.
cert_revocation	A token certificate was revoked.
cert_unrevocation	A token certificate was unrevoked.

2.5.2.4.2. Token Policies

In case of internal registration, each token can be governed by a set of token policies. The default policies are:

```
RE_ENROLL=YES;RENEW=NO;FORCE_FORMAT=NO;PIN_RESET=NO;RESET_PIN_RESET_TO
_NO=NO;RENEW_KEEP_OLD_ENC_CERTS=YES
```

All TPS operations under internal registration are subject to the policies specified in the token's record. If no policies are specified for a token, the TPS uses the default set of policies.

2.5.2.5. Mapping Resolver

The *Mapping Resolver* is an extensible mechanism used by the TPS to determine which token profile to assign to a specific token based on configurable criteria. Each mapping resolver instance can be uniquely defined in the configuration, and each operation can point to various defined mapping resolver instance.

> **NOTE**
>
> The mapping resolver framework provides a platform for writing custom plug-ins. However instructions on how to write a plug-in is outside the scope of this document.

FilterMappingResolver is the only mapping resolver implementation provided with the TPS by default. It allows you to define a set of *mappings* and a target result for each mapping. Each mapping contains a set of filters, where:

- If the input filter parameters pass *all* filters within a mapping, the **target** value is assigned.

- If the input parameters fail a filter, that mapping is skipped and the next one in order is tried.

- If a filter has no specified value, it always passes.

- If a filter does have a specified value, then the input parameters must match exactly.

- The order in which mappings are defined is important. The first mapping which passes is considered resolved and is returned to the caller.

The input filter parameters are information received from the smart card token with or without extensions. They are run against the **FilterMappingResolver** according to the above rules. The following input filter parameters are supported by **FilterMappingResolver**:

- **appletMajorVersion** – The major version of the Coolkey applet on the token.

- **appletMinorVersion** – The minor version of the Coolkey applet on the token.

- **keySet** or **tokenType**

  - **keySet** – can be set as an extension in the client request. Must match the value in the filter if the extension is specified. The keySet mapping resolver is meant for determining keySet value when using external registration. The Key Set Mapping Resolver is necessary in the external registration environment when multiple key sets are supported (for example, different smart card token vendors). The keySet value is needed for identifying the master key on TKS, which is crucial for establishing Secure Channel. When a user's LDAP record is populated with a set tokenType (TPS profile), it does not know which card will end up doing the enrollment, and therefore keySet cannot be predetermined. The **keySetMappingResolver** helps solve the issue by allowing the keySet to be resolved before authentication.

  - **tokenType** – okenType can be set as an extension in the client request. It must match the value in the filter if the extension is specified. tokenType (also referred to as TPS Profile) is determined at this time for the internal registration environment.

- **tokenATR** – The token's Answer to Reset (ATR).

- **tokenCUID** – "start" and "end" define the range the Card Unique IDs (CUID) of the token must fall in to pass this filter.

2.5.2.6. TPS Roles

The TPS supports the following roles by default:

- **TPS Administrator** – this role is allowed to:
  - Manage TPS tokens
  - View TPS certificates and activities
  - Manage TPS users and groups
  - Change general TPS configuration
  - Manage TPS authenticators and connectors
  - Configure TPS profiles and profile mappings
  - Configure TPS audit logging
- **TPS Agent** – this role is allowed to:
  - Configure TPS tokens
  - View TPS certificates and activities
  - Change the status of TPS profiles
- **TPS Operator** – this role is allowed to:
  - View TPS tokens, certificates, and activities

2.5.3. TKS/TPS Shared Secret

During TMS installation, a shared symmetric key is established between the Token Key Service and the Token Processing System. The purpose of this key is to wrap and unwrap session keys which are essential to Secure Channels.

> **NOTE**
>
> The shared secret key is currently only kept in a software cryptographical database. There are plans to support keeping the key on a Hardware Security Module (HSM) devices in a future release of Red Hat Certificate System. Once this functionality is implemented, you will be instructed to run a Key Ceremony using **tkstool** to transfer the key to the HSM.

2.5.4. Enterprise Security Client (ESC)

The *Enterprise Security Client* is an HTTP client application, similar to a web browser, that communicates with the TPS and handles smart card tokens from the client side. While an HTTPS connection is established between the ESC and the TPS, an underlying Secure Channel is also established between the token and the TMS within each TLS session.

2.6. RED HAT CERTIFICATE SYSTEM SERVICES

Certificate System has a number of different features for administrators to use which makes it easier to maintain the individual subsystems and the PKI as a whole.

2.6.1. Notifications

When a particular event occurs, such as when a certificate is issued or revoked, then a notification can be sent directly to a specified email address. The notification framework comes with default modules that can be enabled and configured.

2.6.2. Jobs

Automated jobs run at defined intervals.

2.6.3. Logging

The Certificate System and each subsystem produce extensive system and error logs that record system events so that the systems can be monitored and debugged. All log records are stored in the local file system for quick retrieval. Logs are configurable, so logs can be created for specific types of events and at the required logging level.

Certificate System allows logs to be signed digitally before archiving them or distributing them for auditing. This feature enables log files to be checked for tampering after being signed.

2.6.4. Auditing

The Certificate System maintains audit logs for all events, such as requesting, issuing and revoking certificates and publishing CRLs. These logs are then signed. This allows authorized access or activity to be detected. An outside auditor can then audit the system if required. The assigned auditor user account is the only account which can view the signed audit logs. This user's certificate is used to sign and encrypt the logs. Audit logging is configured to specify the events that are logged.

2.6.5. Self-Tests

The Certificate System provides the framework for system self-tests that are automatically run at startup and can be run on demand. A set of configurable self-tests are already included with the Certificate System.

2.6.6. Users, Authorization, and Access Controls

Certificate System users can be assigned to groups, which are also known as roles, and they then have the privileges of whichever group they are members. A user only has privileges for the instance of the subsystem in which the user is created and the privileges of the group to which the user is a member.

Authentication is the means that Certificate System subsystems use to verify the identity of clients, whether they are authenticating to a certificate profile or to one of the services interfaces. There are a number of different ways that a client can authenticate, including simple user name/password, TLS client authentication, LDAP authentication, NIS authentication, or CMC. Authentication can be performed for any access to the subsystem; for certificate enrollments, for example, the profile defines how the requestor authenticates to the CA.

Once the client is identified and authenticated, then the subsystems perform an *authorization* check to determine what level of access to the subsystem that particular user is allowed.

Authorization is tied to group, or role, permissions, rather than directly to individual users. The

Certificate System provides an authorization framework for creating groups and assigning access control to those groups. The default access control on preexisting groups can be modified, and access control can be assigned to individual users and IP addresses. Access points for authorization have been created for the major portions of the system, and access control rules can be set for each point.

2.6.6.1. Default Administrative Roles

> **NOTE**
>
> Red Hat Certificate System uses the words *roles* and *groups* interchangeably in the context of the permissions given to users.

The Certificate System is configured by default with three roles with different access levels to the system:

- *Administrators*, who can perform any administrative or configuration task for a subsystem.

- *Agents*, who perform PKI management tasks, like approving certificate requests, managing token enrollments, or recovering keys.

- *Auditors*, who can view and configure audit logs.

> **NOTE**
>
> By default, for bootstrapping purposes, an administrative user processing both Administrator and Agent privileges is created during the Red Hat Certificate System instance creation, when running the **pkispawn** utility. This bootstrap administrator uses the **caadmin** user name by default but can be overridden by the *pki_admin_uid* parameter in the configuration file passed to the **pkispawn** command. The purpose of the bootstrap administrator is to create the first administrator and agent user. This operation requires the administrator privilege to mange user and groups, and the agent privilege to issue certificates.

2.6.6.2. Built-in Subsystem Trust Roles

Additionally, when a security domain is created, the CA subsystem which hosts the domain is automatically granted the special role of *Security Domain Administrator*, which gives the subsystem the ability to manage the security domain and the subsystem instances within it. Other security domain administrator roles can be created for the different subsystem instances. These special roles should not have actual users as members.

CHAPTER 3. ALLOWED STANDARDS AND PROTOCOLS

Red Hat Certificate System is based on many public and standard protocols and RFCs, to ensure the best possible performance and interoperability. The major standards and protocols used or allowed by Certificate System 9 are outlined in this chapter, to help administrators plan their client services effectively.

3.1. ALLOWED TLS CIPHER SUITES

The Transport Layer Security (TLS) protocol is a universally accepted standard for authenticated and encrypted communication between clients and servers. Red Hat Certificate System supports TLS 1.1 and 1.2.

Red Hat Certificate System supports the following cipher suites when the server is acting either as a server or as a client:

ECC

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

RSA

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA

- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256

- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

3.2. ALLOWED KEY ALGORITHMS AND THEIR SIZES

Red Hat Certificate System supports the following key algorithms and sizes if they are provided by the underlying PKCS #11 module.

- Allowed RSA key sizes:

  - 2048 bits or greater

- Allowed EC curves or equivalent as defined in the FIPS PUB 186-4 standard:

  - nistp256

  - nistp384

  - nistp521

3.3. ALLOWED HASH FUNCTIONS

The following keyed hash messages authentication (HMAC) are allowed:

- SHA-256

- SHA-384

- SHA-512

The following cryptographic hashing functions are allowed:

- SHA-256

- SHA-384

- SHA-512

3.4. ALLOWED PKIX FORMATS AND PROTOCOLS

The Certificate System supports many of the protocols and formats defined in Public-Key Infrastructure (X.509) by the IETF. In addition to the PKIX standards listed here, other PKIX-listed standards are available at the IETF Datatracker website.

Table 3.1. PKIX Standards Allowed in Certificate System 9

Format or Protocol	RFC or Draft	Description
X.509 version 3		Digital certificate formats recommended by the International Telecommunications Union (ITU).
Certificate Request Message Format (CRMF)	RFC 4211	A message format to send a certificate request to a CA.

Format or Protocol	RFC or Draft	Description
Certificate Management Messages over CS (CMC)	RFC 5272	Certificate Management protocol using the Cryptographic Message Syntax (CMS). CMC incorporates CRMF and PKCS #10.
Cryptographic Message Syntax (CMS)	RFC 2630	A superset of PKCS #7 syntax used for digital signatures and encryption.
PKIX Certificate and CRL Profile	RFC 5280	A standard developed by the IETF for a public-key infrastructure for the Internet. It specifies profiles for certificates and CRLs.
Online Certificate Status Protocol (OCSP)	RFC 6960	A protocol useful in determining the current status of a digital certificate without requiring CRLs.

CHAPTER 4. SUPPORTED PLATFORMS

This section describes the different server platforms, hardware, tokens, and software supported by Red Hat Certificate System 9.4.

4.1. SERVER SUPPORT

Running the Certificate Authority (CA), Key Recovery Authority (KRA), Online Certificate Status Protocol (OCSP), Token Key Service (TKS), and Token Processing System (TPS) subsystems of Certificate System 9.4 is supported on Red Hat Enterprise Linux 7.6 and later. The supported Directory Server version is 10.3 and later.

> **NOTE**
>
> Certificate System 9.4 is supported running on a Red Hat Enterprise Linux virtual guest on a certified hypervisor. For details, see the Which hypervisors are certified to run Red Hat Enterprise Linux? solution article.

4.2. SUPPORTED WEB BROWSERS

Certificate System 9.4 supports the following browsers:

Table 4.1. Supported Web Browsers by Platform

Platform	Agent Services	End User Pages
Red Hat Enterprise Linux	Firefox 60 and later [a]	Firefox 60 and later [a]
Windows 7	Firefox 60 and later [a]	Firefox 60 and later Internet Explorer 10 [b]

[a] This Firefox version no longer supports the **crypto** web object used to generate and archive keys from the browser. As a result, expect limited functionality in this area.

[b] Internet Explorer 11 is currently not supported by Red Hat Certificate System 9 because the enrollment code for this web browser depends upon Visual Basic Script, which has been deprecated in Internet Explorer 11.

> **NOTE**
>
> The only fully-supported browser for the HTML-based instance configuration is Mozilla Firefox.

4.3. SUPPORTED HARDWARE SECURITY MODULES

The following table lists Hardware Security Modules (HSM) supported by Red Hat Certificate System:

HSM	Firmware	Appliance Software	Client Software
nCipher nCipher nShield Connect 6000	2.61.2	CipherTools-linux64-dev-12.30.00	CipherTools-linux64-dev-12.30.00
Gemalto SafeNet Luna SA 1700 / 7000 (limited) (Limited support)	6.24.0	6.2.0-15	libcryptoki-6.2.x86_64

CHAPTER 5. PLANNING THE CERTIFICATE SYSTEM

Each Red Hat Certificate System subsystem can be installed on the same server machine, installed on separate servers, or have multiple instances installed across an organization. Before installing any subsystem, it is important to plan the deployment out: what kind of PKI services do you need? What are the network requirements? What people need to access the Certificate System, what are their roles, and what are their physical locations? What kinds of certificates do you want to issue and what constraints or rules need to be set for them?

This chapter covers some basic questions for planning a Certificate System deployment. Many of these decisions are interrelated; one choice impacts another, like deciding whether to use smart cards determines whether to install the TPS and TKS subsystems.

5.1. DECIDING ON THE REQUIRED SUBSYSTEMS

The Certificate System subsystems cover different aspects of managing certificates. Planning which subsystems to install is one way of defining what PKI operations the deployment needs to perform.

Certificates, like software or equipment, have a lifecycle with defined stages. At its most basic, there are three steps:

- It is requested and issued.

- It is valid.

- It expires.

However, this simplified scenario does not cover a lot of common issues with certificates:

- What if an employee leaves the company before the certificate expires?

- When a CA signing certificate expires, all of the certificates issued and signed using that certificate also expire. So will the CA signing certificate be renewed, allowing its issued certificates to remain valid, or will it be reissued?

- What if an employee loses a smart card or leaves it at home. Will a replacement certificate be issued using the original certificates keys? Will the other certificates be suspended or revoked? Are temporary certificates allowed?

- When a certificate expires, will a new certificate be issued or will the original certificate be renewed?

This introduces three other considerations for managing certificates: revocation, renewal, and replacements.

Other considerations are the loads on the certificate authority. Are there a lot of issuance or renewal requests? Is there a lot of traffic from clients trying to validate whether certificates are valid? How are people requesting certificates supposed to authenticate their identity, and does that process slow down the issuance process?

5.1.1. Using a Single Certificate Manager

The core of the Certificate System PKI is the Certificate Manager, a certificate authority. The CA receives certificate requests and issues all certificates.
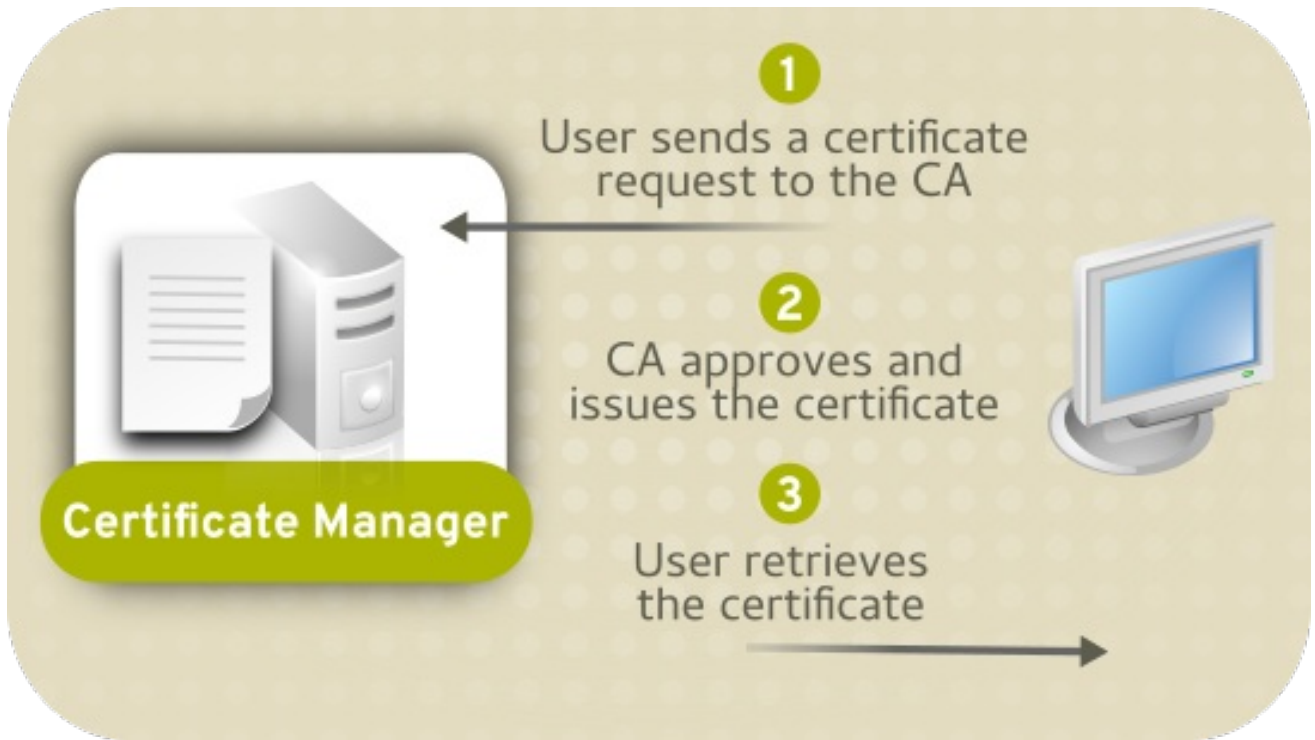
Figure 5.1. CA Only Certificate System

All of the basic processing for requests and issuing certificates can be handled by the Certificate Manager, and it is the only required subsystem. There can be a single Certificate Manager or many Certificate Managers, arranged in many different relationships, depending on the demands of the organization.

Along with issuing certificates, a Certificate Manager can also revoke certificates. One question for administrators is how to handle certificates if they are lost, compromised, or if the person or equipment for which they are issued leaves the company. Revoking a certificate invalidates it before its expiration date, and a list of revoked certificates is compiled and published by the issuing CA so that clients can verify the certificate status.
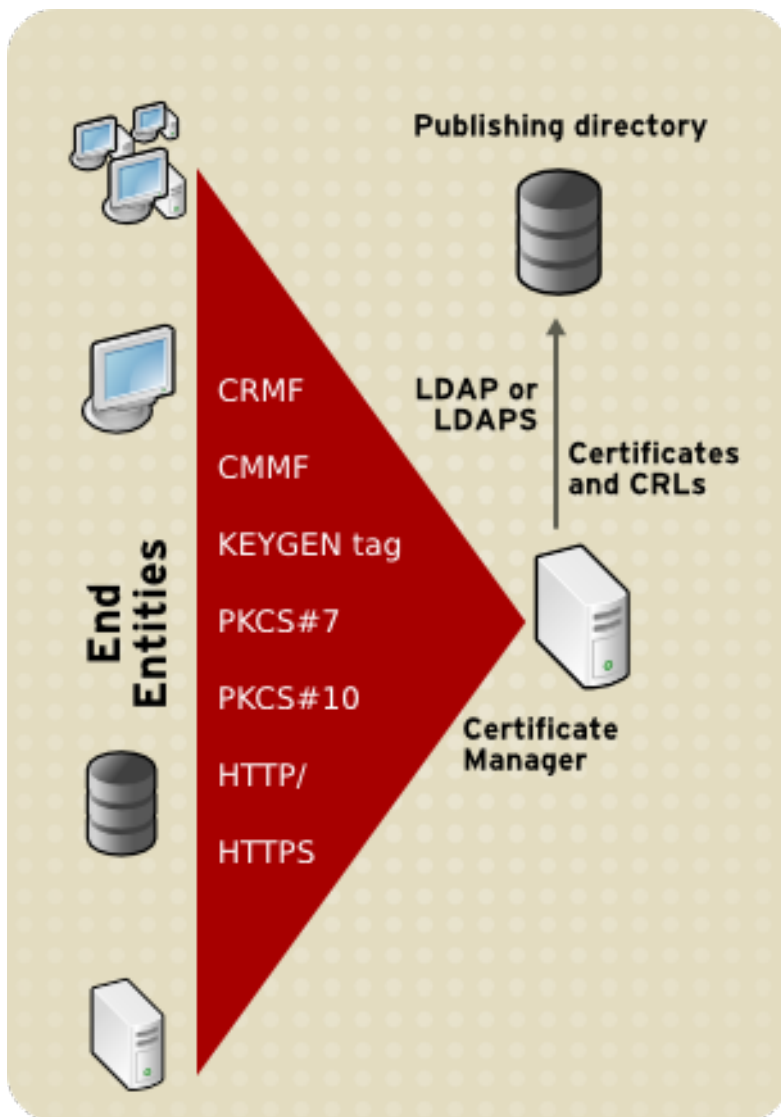
5.1.2. Planning for Lost Keys: Key Archival and Recovery

One operation the CA cannot perform, though, is key archival and recovery. A very real scenario is that a user is going to lose his private key — for instance, the keys could be deleted from a browser database or a smart card could be left at home. Many common business operations use encrypted data, like encrypted email, and losing the keys which unlock that data means the data itself is lost. Depending on the policies in the company, there probably has to be a way to recover the keys in order to regenerate or reimport a replacement certificate, and both operations require the private key.

That requires a KRA, the subsystem which specially archives and retrieves keys.
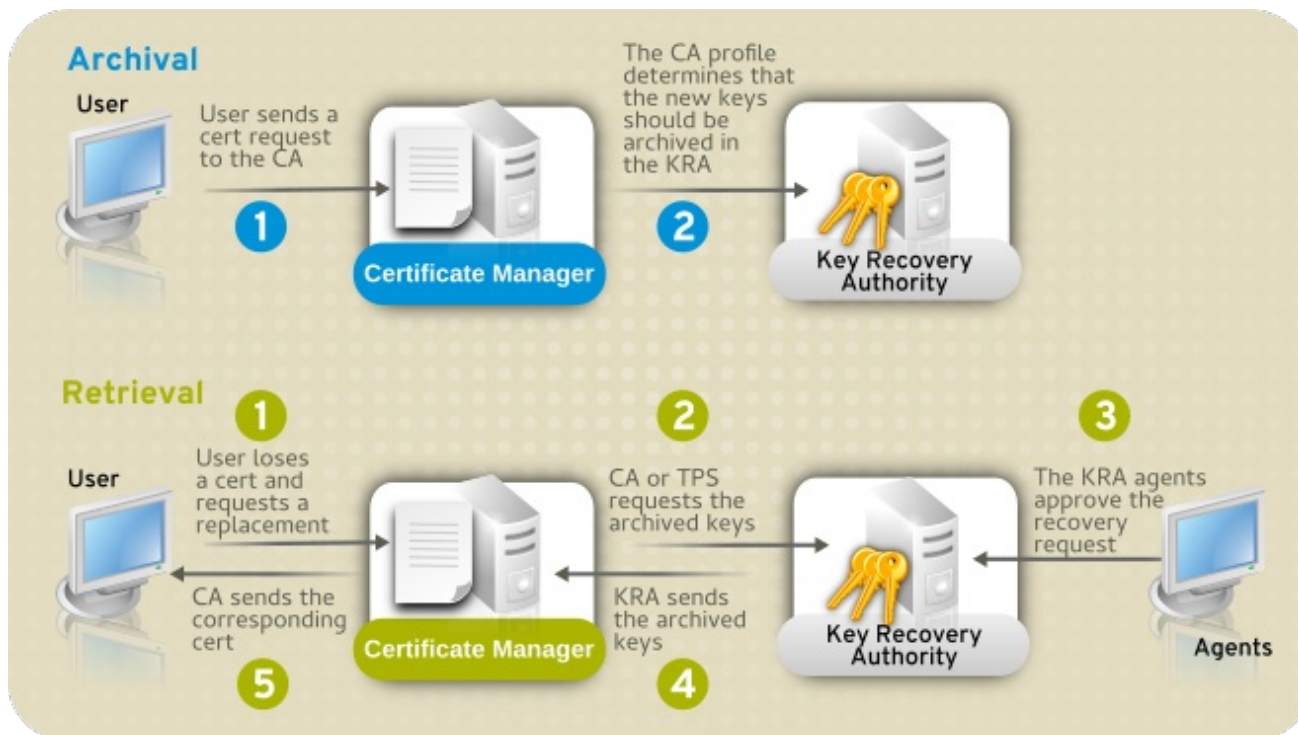
Figure 5.2. CA and KRA

The Key Recovery Authority stores encryption keys (key archival) and can retrieve those keys so that the CA can reissue a certificate (key recovery). A KRA can store keys for any certificate generated by Certificate System, whether it is done for a regular certificate or when enrolling a smart card.

The key archival and recovery process is explained in more detail in Section 2.4.5, "Archiving, Recovering, and Rotating Keys".

> **NOTE**
>
> The KRA is intended for archival and recovery of private encryption keys only. Therefore, end users must use a browser that supports dual-key generation to store their public-private key pairs.

5.1.3. Balancing Certificate Request Processing

Another aspect of how the subsystems work together is load balancing. If a site has high traffic, then it is possible to simply install a lot of CAs, as clones of each other or in a flat hierarchy (where each CA is independent) or in a tree hierarchy (where some CAs are subordinate to other CAs); this is covered more in Section 5.2, "Defining the Certificate Authority Hierarchy".

5.1.4. Balancing Client OCSP Requests

If a certificate is within its validity period but needs be invalidated, it can be revoked. A Certificate Manager can publish lists of revoked certificates, so that when a client needs to verify that a certificate is still valid, it can check the list. These requests are *online certificate status protocol requests*, meaning that they have a specific request and response format. The Certificate Manager has a built-in OCSP responder so that it can verify OCSP requests by itself.

However, as with certificate request traffic, a site may have a significant number of client requests to verify certificate status. Example Corp. has a large web store, and each customer's browser tries to verify the validity of their TLS certificates. Again, the CA can handle issuing the number of certificates, but the

high request traffic affects its performance. In this case, Example Corp. uses the external OCSP Manager subsystem to verify certificate statuses, and the Certificate Manager only has to publish updated CRLs every so often.
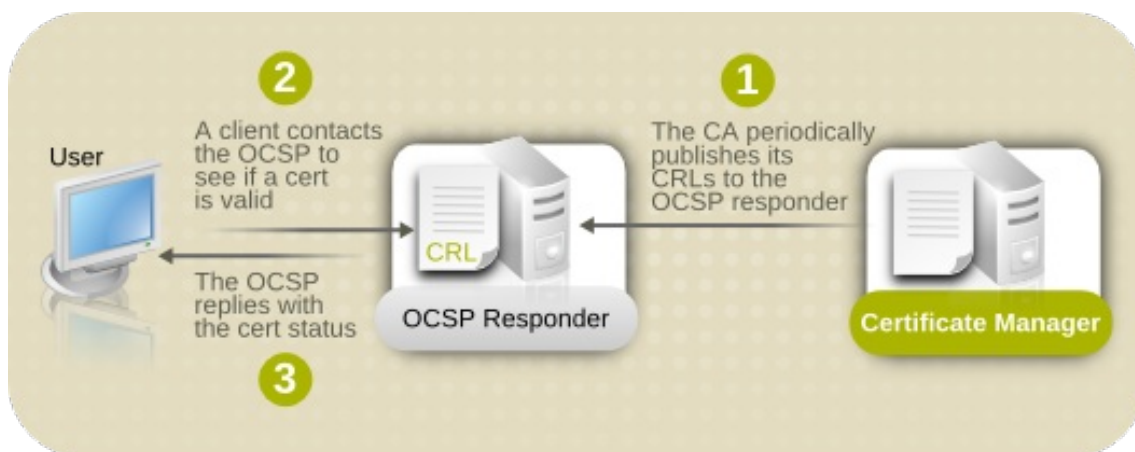
Figure 5.3. CA and OCSP

5.2. DEFINING THE CERTIFICATE AUTHORITY HIERARCHY

The CA is the center of the PKI, so the relationship of CA systems, both to each other (CA hierarchy) and to other subsystems (security domain) is vital to planning a Certificate System PKI.

When there are multiple CAs in a PKI, the CAs are structured in a hierarchy or chain. The CA above another CA in a chain is called an *root CA*; a CA below another CA in the chain is called a *subordinate CA*. A CA can also be subordinate to a root outside of the Certificate System deployment; for example, a CA which functions as a root CA within the Certificate System deployment can be subordinate to a third-party CA.

A Certificate Manager (or CA) is subordinate to another CA because its CA signing certificate, the certificate that allows it to issue certificates, is issued by another CA. The CA that issued the subordinate CA signing certificate controls the CA through the contents of the CA signing certificate. The CA can constrain the subordinate CA through the kinds of certificates that it can issue, the extensions that it is allowed to include in certificates, the number of levels of subordinate CAs the subordinate CA can create, and the validity period of certificates it can issue, as well as the validity period of the subordinate CAs signing certificate.

> **NOTE**
>
> Although a subordinate CA can create certificates that violate these constraints, a client authenticating a certificate that violates those constraints will not accept that certificate.

A self-signed root CA signs its own CA signing certificate and sets its own constraints as well as setting constraints on the subordinate CA signing certificates it issues.

A Certificate Manager can be configured as either a root CA or a subordinate CA. It is easiest to make the first CA installed a self-signed root, so that it is not necessary to apply to a third party and wait for the certificate to be issued. Before deploying the full PKI, however, consider whether to have a root CA, how many to have, and where both root and subordinate CAs will be located.

5.2.1. Subordination to a Public CA

Chaining the Certificate System CA to a third-party public CA introduces the restrictions that public

CAs place on the kinds of certificates the subordinate CA can issue and the nature of the certificate chain. For example, a CA that chains to a third-party CA might be restricted to issuing only Secure Multipurpose Internet Mail Extensions (S/MIME) and TLS client authentication certificates, but not TLS server certificates. There are other possible restrictions with using a public CA. This may not be acceptable for some PKI deployments.

One benefit of chaining to a public CA is that the third party is responsible for submitting the root CA certificate to a web browser or other client software. This can be a major advantage for an extranet with certificates that are accessed by different companies with browsers that cannot be controlled by the administrator. Creating a root CA in the CA hierarchy means that the local organization must get the root certificate into all the browsers which will use the certificates issued by the Certificate System. There are tools to do this within an intranet, but it can be difficult to accomplish with an extranet.

5.2.2. Subordination to a Certificate System CA

The Certificate System CA can function as a *root CA*, meaning that the server signs its own CA signing certificate as well as other CA signing certificates, creating an organization-specific CA hierarchy. The server can alternatively be configured as a *subordinate CA*, meaning the server's CA signing key is signed by another CA in an existing CA hierarchy.

Setting up a Certificate System CA as the root CA means that the Certificate System administrator has control over all subordinate CAs by setting policies that control the contents of the CA signing certificates issued. A subordinate CA issues certificates by evaluating its own authentication and certificate profile configuration, without regard for the root CA's configuration.

5.2.3. Linked CA

The Certificate System Certificate Manager can function as a *linked CA*, chaining up to many third-party or public CAs for validation; this provides cross-company trust, so applications can verify certificate chains outside the company certificate hierarchy. A Certificate Manager is chained to a third-party CA by requesting the Certificate Manager's *CA signing certificate* from the third-party CA.

Related to this, the Certificate Manager also can issue *cross-pair* or *cross-signed certificates*. Cross-pair certificates create a trusted relationship between two separate CAs by issuing and storing cross-signed certificates between these two CAs. By using cross-signed certificate pairs, certificates issued outside the organization's PKI can be trusted within the system.

These are also called *bridge certificates*, related to the Federal Bridge Certification Authority (FBCA) definition.

5.3. PLANNING SECURITY DOMAINS

A *security domain* is a registry of PKI services. PKI services, such as CAs, register information about themselves in these domains so users of PKI services can find other services by inspecting the registry. The security domain service in Certificate System manages both the registration of PKI services for Certificate System subsystems and a set of shared trust policies.

The registry provides a complete view of all PKI services provided by the subsystems within that domain. Each Certificate System subsystem must be either a host or a member of a security domain.

A CA subsystem is the only subsystem which can host a security domain. The security domain shares the CA internal database for privileged user and group information to determine which users can update the security domain, register new PKI services, and issue certificates.

A security domain is created during CA configuration, which automatically creates an entry in the

security domain CA's LDAP directory. Each entry contains all the important information about the domain. Every subsystem within the domain, including the CA registering the security domain, is recorded under the security domain container entry.

The URL to the CA uniquely identifies the security domain. The security domain is also given a friendly name, such as **Example Corp Intranet PKI**. All other subsystems — KRA, TPS, TKS, OCSP, and other CAs — must become members of the security domain by supplying the security domain URL when configuring the subsystem.

Each subsystem within the security domain shares the same trust policies and trusted roots which can be retrieved from different servers and browsers. The information available in the security domain is used during configuration of a new subsystem, which makes the configuration process streamlined and automated. For example, when a TPS needs to connect to a CA, it can consult the security domain to get a list of available CAs.

Each CA has its own LDAP entry. The security domain is an organizational group underneath that CA entry:

```
ou=Security Domain,dc=server.example.com-pki-ca
```

Then there is a list of each subsystem type beneath the security domain organizational group, with a special object class (**pkiSecurityGroup**) to identify the group type:

```
cn=KRAList,ou=Security Domain,o=pki-tomcat-CA
objectClass: top
objectClass: pkiSecurityGroup
cn: KRAList
```

Each subsystem instance is then stored as a member of that group, with a special **pkiSubsystem** object class to identify the entry type:

```
dn: cn=kra.example.com:8443,cn=KRAList,ou=Security Domain,o=pki-tomcat-CA
objectClass: top
objectClass: pkiSubsystem
cn: kra.example.com:8443
host: server.example.com
UnSecurePort: 8080
SecurePort: 8443
SecureAdminPort: 8443
SecureAgentPort: 8443
SecureEEClientAuthPort: 8443
DomainManager: false
Clone: false
SubsystemName: KRA kra.example.com 8443
```

If a subsystem needs to contact another subsystem to perform an operation, it contacts the CA which hosts the security domain (by invoking a servlet which connects over the administrative port of the CA). The security domain CA then retrieves the information about the subsystem from its LDAP database, and returns that information to the requesting subsystem.

The subsystem authenticates to the security domain using a subsystem certificate.

Consider the following when planning the security domain:

- The CA hosting the security domain can be signed by an external authority.

- Multiple security domains can be set up within an organization. However, each subsystem can belong to only one security domain.

- Subsystems within a domain can be cloned. Cloning subsystem instances distributes the system load and provides failover points.

- The security domain streamlines configuration between the CA and KRA; the KRA can push its KRA connector information and transport certificates automatically to the CA instead of administrators having to manually copy the certificates over to the CA.

- The Certificate System security domain allows an offline CA to be set up. In this scenario, the offline root has its own security domain. All online subordinate CAs belong to a different security domain.

- The security domain streamlines configuration between the CA and OCSP. The OCSP can push its information to the CA for the CA to set up OCSP publishing and also retrieve the CA certificate chain from the CA and store it in the internal database.

5.4. DETERMINING THE REQUIREMENTS FOR SUBSYSTEM CERTIFICATES

The CA configuration determines many of the characteristics of the certificates which it issues, regardless of the actual type of certificate being issued. Constraints on the CA's own validity period, distinguished name, and allowed encryption algorithms impact the same characteristics in their issued certificates. Additionally, the Certificate Managers have predefined profiles that set rules for different kinds of certificates that they issue, and additional profiles can be added or modified. These profile configurations also impact issued certificates.

5.4.1. Determining Which Certificates to Install

When a Certificate System subsystem is first installed and configured, the certificates necessary to access and administer it are automatically created. These include an agent's certificate, server certificate, and subsystem-specific certificates. These initial certificates are shown in Table 5.1, "Initial Subsystem Certificates".

Table 5.1. Initial Subsystem Certificates

Subsystem	Certificates
Certificate Manager	- CA signing certificate - OCSP signing certificate - TLS server certificate - Subsystem certificate - User's (agent/administrator) certificate - Audit log signing certificate

Subsystem	Certificates
OCSP	- OCSP signing certificate - TLS server certificate - Subsystem certificate - User's (agent/administrator) certificate - Audit log signing certificate
KRA	- Transport certificate - Storage certificate - TLS server certificate - Subsystem certificate - User's (agent/administrator) certificate - Audit log signing certificate
TKS	- TLS server certificate - User's (agent/administrator) certificate - Audit log signing certificate
TPS	- TLS server certificate - User's (agent/administrator) certificate - Audit log signing certificate

There are some cautionary considerations about replacing existing subsystem certificates.

- Generating new key pairs when creating a new self-signed CA certificate for a root CA will invalidate all certificates issued under the previous CA certificate.

  This means none of the certificates issued or signed by the CA using its old key will work; subordinate Certificate Managers, KRAs, OCSPs, TKSs, and TPSs will no longer function, and agents can no longer to access agent interfaces.

  This same situation occurs if a subordinate CA's CA certificate is replaced by one with a new key pair; all certificates issued by that CA are invalidated and will no longer work.

  Instead of creating new certificates from new key pairs, consider renewing the existing CA signing certificate.

- If the CA is configured to publish to the OCSP and it has a new CA signing certificate or a new CRL signing certificate, the CA must be identified again to the OCSP.

- If a new transport certificate is created for the KRA, the KRA information must be updated in the CA's configuration file, **CS.cfg**. The existing transport certificate must be replaced with the new one in the **ca.connector.KRA.transportCert** parameter.

- If a CA is cloned, then when creating a new TLS server certificate for the master Certificate Manager, the clone CAs' certificate databases all need updated with the new TLS server certificate.

- If the Certificate Manager is configured to publish certificates and CRLs to an LDAP directory and uses the TLS server certificate for TLS client authentication, then the new TLS server certificate must be requested with the appropriate extensions. After installing the certificate, the publishing directory must be configured to use the new server certificate.

- Any number of TLS server certificates can be issued for a subsystem instance, but it really only needs one TLS certificate. This certificate can be renewed or replaced as many times as necessary.

5.4.2. Planning the CA Distinguished Name

The core elements of a CA are a signing unit and the Certificate Manager identity. The signing unit digitally signs certificates requested by end entities. A Certificate Manager must have its own distinguished name (DN), which is listed in every certificate it issues.

Like any other certificate, a CA certificate binds a DN to a public key. A DN is a series of name-value pairs that in combination uniquely identify an entity. For example, the following DN identifies a Certificate Manager for the Engineering department of a corporation named Example Corporation:

cn=demoCA, o=Example Corporation, ou=Engineering, c=US

Many combinations of name-value pairs are possible for the Certificate Manager's DN. The DN must be unique and readily identifiable, since any end entity can examine it.

5.4.3. Setting the CA Signing Certificate Validity Period

Every certificate, including a Certificate Manager signing certificate, must have a validity period. The Certificate System does not restrict the validity period that can be specified. Set as long a validity period as possible, depending on the requirements for certificate renewal, the place of the CA in the certificate hierarchy, and the requirements of any public CAs that are included in the PKI.

A Certificate Manager cannot issue a certificate that has a validity period longer than the validity period of its CA signing certificate. If a request is made for a period longer than the CA certificate's validity period, the requested validity date is ignored and the CA signing certificate validity period is used.

5.4.4. Choosing the Signing Key Type and Length

A signing key is used by a subsystem to verify and "seal" something. CAs use a CA signing certificate to sign certificates or CRLs that it issues; OCSPs use signing certificates to verify their responses to certificate status requests; all subsystems use log file signing certificates to sign their audit logs.

The signing key must be cryptographically strong to provide protection and security for its signing operations. The following signing algorithms are considered secure:

- SHA256withRSA

- SHA384withRSA

- SHA512withRSA

- SHA256withEC

- SHA384withEC

- SHA512withEC
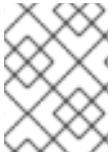
> **NOTE**
>
> Certificate System includes native ECC support. It is also possible to load and use a third-party PKCS #11 module with ECC-enabled.

Along with a key *type*, each key has a specific *bit length*. Longer keys are considered cryptographically stronger than shorter keys. However, longer keys require more time for signing operations.

The default RSA key length at installation is 2048 bits; for certificates that provide access to highly sensitive data or services, consider increasing the length to 4096 bits. ECC keys are much stronger than RSA keys, so the recommended length for ECC keys is 256 bits, which is equivalent in strength to a 2048-bit RSA key.

5.4.5. Using Certificate Extensions

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, require Netscape-specific extensions because they were developed before PKIX part 1 standards were defined.

The X.509 v1 certificate specification was originally designed to bind public keys to names in an X.500 directory. As certificates began to be used on the Internet and extranets and directory lookups could not always be performed, problem areas emerged that were not covered by the original specification.

- *Trust.* The X.500 specification establishes trust by means of a strict directory hierarchy. By contrast, Internet and extranet deployments frequently involve distributed trust models that do not conform to the hierarchical X.500 approach.

- *Certificate use.* Some organizations restrict how certificates are used. For example, some certificates may be restricted to client authentication only.

- *Multiple certificates.* It is not uncommon for certificate users to possess multiple certificates with identical subject names but different key material. In this case, it is necessary to identify which key and certificate should be used for what purpose.

- *Alternate names.* For some purposes, it is useful to have alternative subject names that are also bound to the public key in the certificate.

- *Additional attributes.* Some organizations store additional information in certificates, such as when it is not possible to look up information in a directory.

- *Relationship with CA.* When certificate chaining involves intermediate CAs, it is useful to have information about the relationships among CAs embedded in their certificates.

- *CRL checking*. Since it is not always possible to check a certificate's revocation status against a directory or with the original certificate authority, it is useful for certificates to include information about where to check CRLs.

The X.509 v3 specification addressed these issues by altering the certificate format to include additional information within a certificate by defining a general format for certificate extensions and specifying extensions that can be included in the certificate. The extensions defined for X.509 v3 certificates enable additional attributes to be associated with users or public keys and manage the certification hierarchy. The *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* recommends a set of extensions to use for Internet certificates and standard locations for certificate or CA information. These extensions are called *standard extensions*.

NOTE

For more information on standard extensions, see RFC 2459, RFC 5280, and RFC 3279.

The X.509 v3 standard for certificates allows organizations to define custom extensions and include them in certificates. These extensions are called *private*, *proprietary*, or *custom* extensions, and they carry information unique to an organization or business. Applications may not able to validate certificates that contain private critical extensions, so it not recommended that these be used in wide-spread situations.

The X.500 and X.509 specifications are controlled by the International Telecommunication Union (ITU), an international organization that primarily serves large telecommunication companies, government organizations, and other entities concerned with the international telecommunications network. The Internet Engineering Task Force (IETF), which controls many of the standards that underlie the Internet, is currently developing public-key infrastructure X.509 (PKIX) standards. These proposed standards further refine the X.509 v3 approach to extensions for use on the Internet. The recommendations for certificates and CRLs have reached proposed standard status and are in a document referred to as *PKIX Part 1*.

Two other standards, Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER), are used with Certificate System and certificates in general. These 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.

5.4.5.1. Structure of Certificate Extensions

In RFC 3280, an X.509 certificate extension is defined as follows:

```
Extension  ::=  SEQUENCE  {

extnID OBJECT IDENTIFIER,
critical BOOLEAN DEFAULT FALSE,
extnValue OCTET STRING  }
```

The means a certificate extension consists of the following:

- 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 certificate, 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**).

- A flag or Boolean field called **critical**.

The value, which can be either **true** or **false**, assigned to this field indicates whether the extension is critical or noncritical to the certificate.

- If the extension is critical and the certificate is sent to an application that does not understand the extension based on the extension's ID, the application must reject the certificate.

- If the extension is not critical and the certificate 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 certificate.

- An octet string containing the DER encoding of the value of the extension.

Typically, the application receiving the certificate 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.

Some of the standard extensions defined in the X.509 v3 standard include the following:

- Authority Key Identifier extension, which identifies the CA's public key, the key used to sign the certificate.

- Subject Key Identifier extension, which identifies the subject's public key, the key being certified.

> **NOTE**
>
> Not all applications support certificates with version 3 extensions. Applications that do support these extensions may not be able to interpret some or all of these specific extensions.

5.4.6. Using and Customizing Certificate Profiles

Certificates have different types and different applications. They can be used to establish a single sign-on environment for a corporate network, to set up VPNs, to encrypt email, or to authenticate to a website. The requirements for all of these certificates can be different, just as there may also be different requirements for the same type of certificate for different kinds of users. These certificate characteristics are set in *certificate profiles*. The Certificate Manager defines a set of certificate profiles that it uses as enrollment forms when users or machines request certificates.

For reference, see the *Setting up Certificate Profiles* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Certificate Profiles

A certificate profile defines everything associated with issuing a particular type of certificate, including the authentication method, the certificate content (defaults), constraints for the values of the content, and the contents of the input and output for the certificate profile. Enrollment 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 example, a certificate profile for user certificates defines all aspects of that certificate, including the validity period of the certificate. The default validity period can be set to two years, and a constraint can be set on the profile that the validity period for certificates requested through this certificate profile cannot exceed two years. When a user requests a certificate using the input form associated with this

certificate profile, the issued certificate contains the information specified in the defaults and will be valid for two years. If the user submits a pre-formatted request for a certificate with a validity period of four years, the request is rejected since the constraints allow a maximum of two years validity period for this type of certificate.

A set of certificate profiles have been predefined for the most common certificates issued. These certificate profiles define defaults and constraints, associate the authentication method, and define the needed inputs and outputs for the certificate profile.

Modifying the Certificate Profile Parameters

The parameters of the default certificate profiles can be modified; this includes the authentication method, the defaults, the constraints used in each profile, the values assigned to any of the parameters in a profile, the input, and the output. It is also possible to create new certificate profiles for other types of certificates or for creating more than one certificate profile for a certificate type. There can be multiple certificate profiles for a particular type of certificate to issue the same type of certificate with a different authentication method or different definitions for the defaults and constraints. For example, there can be two certificate profiles for enrollment of TLS server certificates where one certificate profile issues certificates with a validity period of six months and another certificate profile issues certificates with a validity period of two years.

An input sets a text field in the enrollment form and what kind of information needs gathered from the end entity; this includes setting the text area for a certificate request to be pasted, which allows a request to be created outside the input form with any of the request information required. The input values are set as values in the certificate. The default inputs are not configurable in the Certificate System. Only CMC requests are accepted.

An output specifies how the response page to a successful enrollment is presented. It usually displays the certificate in a user-readable format. Some default output show a printable version of the resultant certificate; other outputs set the type of information generated at the end of the enrollment, such as PKCS #7. Only CMC response output is returned for CMC full response, and PKCS #7 for CMC simple response.

Policy sets are sets of constraints and default extensions attached to every certificate processed through the profile. The extensions define certificate content such as validity periods and subject name requirements.

Certificate Profile Administration

An administrator sets up a certificate profile by associating an existing authentication plug-in, or method, with the certificate profile; enabling and configuring defaults and constraints; and defining inputs and outputs. The administrator can use the existing certificate profiles, modify the existing certificate profiles, create new certificate profiles, and delete any certificate profile that will not be used in this PKI.

Once a certificate profile is set up, it appears on the **Manage Certificate Profiles** page of the agent services page where an agent can approve, and thus enable, a certificate profile. Once the certificate profile is enabled, it appears on the **Certificate Profile** tab of the end-entities page where end entities can enroll for a certificate using the certificate profile.

The certificate profile enrollment page in the end-entities interface contains links to each certificate profile that has been enabled by the agents. When an end entity selects one of those links, an enrollment page appears containing an enrollment form specific to that certificate profile. The enrollment page is dynamically generated from the inputs defined for the profile. If an authentication plug-in is configured, additional fields may be added to authenticate the user.

When an end entity submits a certificate profile request that is associated with an agent-approved (manual) enrollment, an enrollment where no authentication plug-in is configured, the certificate request is queued in the agent services interface. The agent can change some aspects of the

enrollment, request, validate it, cancel it, reject it, update it, or approve it. The agent is able to update the request without submitting it or validate that the request adheres to the profile's defaults and constraints. This validation procedure is only for verification and does not result in the request being submitted. The agent is bound by the constraints set; they cannot change the request in such a way that a constraint is violated. The signed approval is immediately processed, and a certificate is issued.

When a certificate profile is associated with an authentication method, the request is approved immediately and generates a certificate automatically if the user successfully authenticates, all the information required is provided, and the request does not violate any of the constraints set up for the certificate profile. There are profile policies which allow user-supplied settings like subject names or validity periods. The certificate profile framework can also preserve user-defined content set in the original certificate request in the issued certificate.

The issued certificate contains the content defined in the defaults for this certificate profile, such as the extensions and validity period for the certificate. The content of the certificate is constrained by the constraints set for each default. Multiple policies (defaults and constraints) can be set for one profile, distinguishing each set by using the same value in the policy set ID. This is particularly useful for dealing with dual keys enrollment where encryption keys and signing keys are submitted to the same profile. The server evaluates each set with each request it receives. When a single certificate is issued, one set is evaluated, and any other sets are ignored. When dual-key pairs are issued, the first set is evaluated with the first certificate request, and the second set is evaluated with the second certificate request. There is no need for more than one set for issuing a single certificate or more than two sets for issuing dual-key pairs.

Guidelines for Customizing Certificate Profiles

Tailor the profiles for the organization to the real needs and anticipated certificate types used by the organization:

- Decide which certificate profiles are needed in the PKI. There should be at least one profile for each type of certificate issued. There can be more than one certificate profile for each type of certificate to set different authentication methods or different defaults and constraints for a particular type of certificate type. Any certificate profile available in the administrative interface can be approved by an agent and then used by an end entity to enroll.

- Delete any certificate profiles that will not be used.

- Modify the existing certificate profiles for specific characteristics for the company's certificates.

    - Change the defaults set up in the certificate profile, the values of the parameters set in the defaults, or the constraints that control the certificate content.

    - Change the constraints set up by changing the value of the parameters.

    - Change the authentication method.

    - Change the inputs by adding or deleting inputs in the certificate profile, which control the fields on the input page.

    - Add or delete the output.

5.4.6.1. Adding SAN Extensions to the TLS Server Certificate

Certificate System enables adding Subject Alternative Name (SAN) extensions to the TLS server certificate during the installation of non-root CA or other Certificate System instances. To do so, follow the instructions in the **/usr/share/pki/ca/profiles/ca/caInternalAuthServerCert.cfg** file and add the following parameters to the configuration file supplied to the **pkispawn** utility:

pki_san_inject

Set the value of this parameter to **True**.

pki_san_for_server_cert

Provide a list of the required SAN extensions separated by commas (,).

For example:

```
pki_san_inject=True
pki_san_for_server_cert=intca01.example.com,intca02.example.com,intca.example.com
```

5.4.7. Planning Authentication Methods

As implied in Section 5.4.6, "Using and Customizing Certificate Profiles" , *authentication* for the certificate process means the way that a user or entity requesting a certificate proves that they are who they say they are.

See the *Authentication for Enrolling Certificates* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

5.4.8. Publishing Certificates and CRLs

A CA can publish both certificates and CRLs. Certificates can be published to a plain file or to an LDAP directory; CRLs can be published to file or an LDAP directory, as well, and can also be published to an OCSP responder to handle certificate verification.

Configuring publishing is fairly straightforward and is easily adjusted. For continuity and accessibility, though, it is good to plan out where certificates and CRLs need to be published and what clients need to be able to access them.

Publishing to an LDAP directory requires special configuration in the directory for publishing to work:

- If certificates are published to the directory, than every user or server to which a certificate is issued must have a corresponding entry in the LDAP directory.

- If CRLs are published to the directory, than they must be published to an entry for the CA which issued them.

- For TLS, the directory service has to be configured in TLS and, optionally, be configured to allow the Certificate Manager to use certificate-based authentication.

- The directory administrator should configure appropriate access control rules to control DN (entry name) and password based authentication to the LDAP directory.

5.4.9. Renewing or Reissuing CA Signing Certificates

When a CA signing certificate expires, all certificates signed with the CA's corresponding signing key become invalid. End entities use information in the CA certificate to verify the certificate's authenticity. If the CA certificate itself has expired, applications cannot chain the certificate to a trusted CA.

There are two ways of resolving CA certificate expiration:

- *Renewing* a CA certificate involves issuing a new CA certificate with the same subject name and

public and private key material as the old CA certificate, but with an extended validity period. As long as the new CA certificate is distributed to all users before the old CA certificate expires, renewing the certificate allows certificates issued under the old CA certificate to continue working for the full duration of their validity periods.

- *Reissuing* a CA certificate involves issuing a new CA certificate with a new name, public and private key material, and validity period. This avoids some problems associated with renewing a CA certificate, but it requires more work for both administrators and users to implement. All certificates issued by the old CA, including those that have not yet expired, must be renewed by the new CA.

There are problems and advantages with either renewing or reissuing a CA certificate. Begin planning the CA certificate renewal or re-issuance before installing any Certificate Managers, and consider the ramifications the planned procedures may have for extensions, policies, and other aspects of the PKI deployment.

> **NOTE**
>
> Correct use of extensions, for example the **authorityKeyIdentifier** extension, can affect the transition from an old CA certificate to a new one.

5.5. PLANNING FOR NETWORK AND PHYSICAL SECURITY

When deploying any Certificate System subsystem, the physical and network security of the subsystem instance has to be considered because of the sensitivity of the data generated and stored by the subsystems.

5.5.1. Considering Firewalls

There are two considerations about using firewalls with Certificate System subsystems:

- Protecting sensitive subsystems from unauthorized access

- Allowing appropriate access to other subsystems and clients outside of the firewall

The CA, KRA, and TKS are always placed inside a firewall because they contain critical information that can cause devastating security consequences if they are compromised.

The TPS and OCSP can be placed outside the firewall. Likewise, other services and clients used by the Certificate System can be on a different machine outside the firewall. In that case, the local networks have to be configured to allow access between the subsystems behind the firewall and the services outside it.

The LDAP database can be on a different server, even on a different network, than the subsystem which uses it. In this case, all LDAP ports (**389** for LDAP and **636** for LDAPS, by default) need to be open in the firewall to allow traffic to the directory service. Without access to the LDAP database, all subsystem operations can fail.

As part of configuring the firewalls, if iptables is enabled, then it must have configured policies to allow communication over the appropriate Certificate System ports. Configuring iptables is described in the *Red Hat Security Guide*.

5.5.2. Considering Physical Security and Location

Because of the sensitive data they hold, consider keeping the CA, KRA, and TKS in a secure facility with adequate physical access restrictions. Just as network access to the systems needs to be restricted, the physical access also needs to be restricted.

Along with finding a secure location, consider the proximity to the subsystem agents and administrators. Key recovery, for example, can require multiple agents to give approval; if these agents are spread out over a large geographical area, then the time differences may negatively impact the ability to retrieve keys. Plan the physical locations of the subsystems, then according to the locations of the agents and administrators who will manage the subsystem.

5.5.3. Planning Ports

Each Certificate System server uses up to four ports:

- A non-secure HTTP port for end entity services that do not require authentication

- A secure HTTP port for end entity services, agent services, administrative console, and admin services that require authentication

- A Tomcat Server Management port

- A Tomcat AJP Connector Port

All of the service pages and interfaces described in the *Red Hat Certificate System User Interfaces* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)* are connected to using the instance's URL and the corresponding port number. For example:

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

To access the admin console, the URL specifies the admin port:

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

All agent and admin functions require TLS client authentication. For requests from end entities, the Certificate System listens on both the TLS (encrypted) port and non-TLS ports.

The ports are defined in the **server.xml** file. If a port is not used, it can be disabled in that file. For example:

> ```
> <Service name="Catalina">
>     <!--Connector port="8080" ... /-->  unused standard port
>     <Connector port="8443" ... />
> ```

Whenever a new instance in installed, make sure that the new ports are unique on the host system.

To verify that a port is available for use, check the appropriate file for the operating system. Port numbers for network-accessible services are usually maintained in a file named **services**. On Red Hat Enterprise Linux, it is also helpful to confirm that a port is not assigned by SELinux, by running the command **semanage port -l** to list all ports which currently have an SELinux context.

When a new subsystem instance is created, any number between 1 and 65535 can be specified as the secure port number.

5.6. A CHECKLIST FOR PLANNING THE PKI

Q: How many security domains will be created, and what subsystem instances will be placed in each domain?

A: A subsystem can only communicate with another subsystem if they have a trusted relationship. Because the subsystems within a security domain have automatic trusted relationships with each other, it is important what domain a subsystem joins. Security domains can have different certificate issuing policies, different kinds of subsystems within them, or a different Directory Server database. Map out where (both on the physical machine and in relation to each other) each subsystem belongs, and assign it to the security domain accordingly.

Q: What ports should be assigned for each subsystem? Is it necessary to have a single TLS port, or is it better to have port separation for extra security?

A: The most secure solution is to use separate ports for each TLS interface. However, the feasibility of implementing multiple ports may depends on your network setup, firewall rules, and other network conditions.

Q: What subsystems should be placed behind firewalls? What clients or other subsystems need to access those firewall-protected subsystems and how will access be granted? Is firewall access allowed for the LDAP database?

A: The location to install a subsystem depends on the network design. The OCSP subsystem is specifically designed to operate outside a firewall for user convenience, while the CA, KRA, and TPS should all be secured behind a firewall for increased security.

When deciding the location of the subsystems, it is critical to plan what *other* subsystems or services that server needs to access (including the internal database, a CA, or external users) and look at firewall, subnet, and VPN configuration.

Q: What subsystems need to be physically secured? How will access be granted, and who will be granted access?

A: The CA, TKS, and KRA all store extremely sensitive key and certificate information. For some deployments, it may be necessary to limit physical access to the machines these subsystems run on. In that case, both the subsystems and their host machines must be included in the larger infrastructure inventory and security design.

Q: What is the physical location of all agents and administrators? What is the physical location of the subsystems? How will administrators or agents access the subsystem services in a timely-manner? Is it necessary to have subsystems in each geographical location or time zone?

A: The physical location of Certificate System users may impact things like request approval and token enrollment, as well as system performance because of network lag time. The importance of response time for processing certificate operations should be taken into account when deciding where and how many subsystems to install.

Q: How many subsystems do you need to install?

A: The primary consideration is the expected load for each subsystem and then, secondarily, geographical or departmental divisions. Subsystems with fairly low loads (like the KRA or TKS) may only require a single instance for the entire PKI. Systems with high load (the CA) or which may benefit from local instances for local agents (like the TPS) may require multiple instances.

Subsystems can be cloned, meaning they essentially are clustered, operating as a single unit, which is good for load balancing and high availability. Cloning is especially good for CAs and KRAs, where the same key and certificate data may need to be accessed by larger numbers of users or to have reliable failover.

When planning for multiple subsystem instances, keep in mind how the subsystems fit within the established security domains. Security domains create trusted relationships between subsystems, allowing them to work together to find available subsystems to respond to immediate needs. Multiple security domains can be used in a single PKI, with multiple instances of any kind of subsystem, or a single security domain can be used for all subsystems.

Q: **Will any subsystems need to be cloned and, if so, what are the methods for securely storing their key materials?**

A: Cloned subsystems work together, essentially as a single instance. This can be good for high demand systems, failover, or load balancing, but it can become difficult to maintain. For example, cloned CAs have serial number ranges for the certificates they issue, and a clone could hit the end of its range.

Q: **Will the subsystem certificates and keys be stored on the internal software token in Certificate System or on an external hardware token?**

A: Certificate System supports two hardware security modules (HSM): nCipher nShield and Safenet LunaSA. Using a hardware token can require additional setup and configuration before installing the subsystems, but it also adds another layer of security.

Q: **What are the requirements for the CA signing certificate? Does the Certificate System need control over attributes like the validity period? How will the CA certificates be distributed?**

A: The real question here is, will the CA bee a root CA which sets its own rules on issuing certificates or will it be a subordinate CA where another CA (another CA in your PKI or even an external CA) sets restrictions on what kind of certificates it can issue.

A Certificate Manager can be configured as either a root CA or a subordinate CA. The difference between a root CA and a subordinate CA is who signs the CA signing certificate. A root CA signs its own certificate. A subordinate CA has another CA (either internal or external) sign its certificate.

A self-signing root CA issues and signs its own CA signing certificate. This allows the CA to set its own configuration rules, like validity periods and the number of allowed subordinate CAs.

A subordinate CA has its certificates issued by a public CA or another Certificate System root CA. This CA is *subordinate* to the other CA's rules about its certificate settings and how the certificate can be used, such as the kinds of certificates that it can issue, the extensions that it is allowed to include in certificates, and the levels of subordinate CAs the subordinate CA can create.

One option is to have the Certificate manager subordinate to a public CA. This can be very restrictive, since it introduces the restrictions that public CAs place on the kinds of certificates the subordinate CA can issue and the nature of the certificate chain. On the other hand, one benefit of chaining to a public CA is that the third party is responsible for submitting the root CA certificate to a web browser or other client software, which is a major advantage for certificates that are accessed by different companies with browsers that cannot be controlled by the administrator.

The other option is make the CA subordinate to a Certificate System CA. Setting up a

Certificate System CA as the root CA means that the Certificate System administrator has control over all subordinate CAs by setting policies that control the contents of the CA signing certificates issued.

It is easiest to make the first CA installed a self-signed root, so that it is not necessary to apply to a third party and wait for the certificate to be issued. Make sure that you determine how many root CAs to have and where both root and subordinate CAs will be located.

Q: **What kinds of certificates will be issued? What characteristics do they need to have, and what profile settings are available for those characteristics? What restrictions need to be placed on the certificates?**

A: As touched on in Section 5.4.6, "Using and Customizing Certificate Profiles" , the profiles (the forms which configure each type of certificate issued by a CA) can be customized. This means that the subject DN, the validity period, the type of TLS client, and other elements can be defined by an administrator for a specific purpose. For security, profiles should provide only the functionality that is required by the PKI. Any unnecessary profiles should be disabled.

Q: **What are the requirements for approving a certificate request? How does the requester authenticate himself, and what kind of process is required to approve the request?**

A: The request approval process directly impacts the security of the certificate. Automated processes are much faster but are less secure since the identity of the requester is only superficially verified. Likewise, agent-approved enrollments are more secure (since, in the most secure scenario they can require an in-person verification and supporting identification) but they can also be the most time-consuming.

First determine how secure the identity verification process needs to be, then determine what authentication (approval) mechanism is required to validate that identity.

Q: **Will many external clients need to validate certificate status? Can the internal OCSP in the Certificate Manager handle the load?**

A: Publishing CRLs and validating certificates is critical. Determine what kinds of clients will be checking the certificate status (will it mainly be internal clients? external clients? will they be validating user certificates or server certificates?) and also try to determine how many OCSP checks will be run. The CA has an internal OCSP which can be used for internal checks or infrequent checks, but a large number of OCSP checks could slow down the CA performance. For a larger number of OCSP checks and especially for large CRLs, consider using a separate OCSP Manager to take the load off the CA.

Q: **Will the PKI allow replacement keys? Will it require key archival and recovery?**

A: If lost certificates or tokens are simply revoked and replaced, then there does not need to be a mechanism to recover them. However, when certificates are used to sign or encrypt data such as emails, files, or harddrives, then the key must always be available so that the data can be recovered. In that case, use a KRA to archive the keys so the data can always be accessed.

Q: **Will the organization use smart cards? If so, will temporary smart cards be allowed if smart cards are mislaid, requiring key archival and recovery?**

A: If no smart cards are used, then it is never necessary to configure the TPS or TKS, since those subsystems are only used for token management. However, if smart cards are used, then the TPS and TKS are required. If tokens can be lost and replaced, then it is also necessary to have a KRA to

Q: **Where will certificates and CRLs be published? What configuration needs to be done on the receiving end for publishing to work? What kinds of certificates or CRLs need to be published and how frequently?**

A: The important thing to determine is what clients need to be able to access the certificate or CRL information and how that access is allowed. From there, you cna define the publishing policy.

PART II. INSTALLING RED HAT CERTIFICATE SYSTEM

This section describes the requirements and procedures for installing Red Hat Certificate System.

CHAPTER 6. PREREQUISITES AND PREPARATION FOR INSTALLATION

The Red Hat Certificate System installation process requires some preparation of the environment. This chapter describes the requirements, dependencies, and other prerequisites when installing Certificate System subsystems.

6.1. INSTALLING RED HAT ENTERPRISE LINUX

Certificate System requires Red Hat Enterprise Linux 7.6. For details about installing Red Hat Enterprise Linux, see the *Red Hat Enterprise Linux Installation Guide*.

To enable the Federal Information Processing Standard (FIPS) on Red Hat Enterprise Linux, see the *Red Hat Security Guide*.

> **IMPORTANT**
>
> FIPS mode must be enabled before you install Certificate System. To ensure FIPS mode is enabled:
>
> ```
> # sysctl crypto.fips_enabled
> ```
>
> If the returned value is **1**, FIPS mode is enabled.

6.2. SECURING THE SYSTEM USING SELINUX

Security-Enhanced Linux (SELinux) is an implementation of a mandatory access control mechanism in the Linux kernel, checking for allowed operations after standard discretionary access controls are checked. SELinux can enforce rules on files and processes in a Linux system, and on their actions, based on defined policies.

In most cases, no actions are required to run Certificate System with SELinux in **enforcing** mode. If a procedure in the Certificate System documentation requires to manually set SELinux-related settings, such as when using a Hardware Security Module (HSM), it is mentioned in the corresponding section.

For further details about SELinux, see the *SELinux User's and Administrator's Guide*.

6.2.1. Verifying if SELinux is Running in Enforcing Mode

By default, after installing Red Hat Enterprise Linux, SELinux is enabled and running in **enforcing** mode and no further actions are required.

To display the current SELinux mode, enter:

```
# getenforce
```

6.3. FIREWALL CONFIGURATION

The following table lists the default ports used by Certificate System subsystems:

Table 6.1. Certificate System Default Ports

Service	Port	Protocol
HTTP	8080	TCP
HTTPS	8443	TCP
Tomcat Management	8005	TCP

When you set up Certificate System using the **pkispawn** utility, you can customize the port numbers. If you use different ports than the defaults listed above, open them correspondingly in the firewall as described in Section 6.3.1, "Opening the Required Ports in the Firewall" . For further details about ports, see Section 5.5.3, "Planning Ports".

For ports required to access Directory Server, see corresponding section in the *Directory Server Installation Guide*.

6.3.1. Opening the Required Ports in the Firewall

To enable communication between the clients and Certificate System, open the required ports in your firewall:

1. Make sure the **firewalld** service is running.

   ```
   # systemctl status firewalld
   ```

2. To start **firewalld** and configure it to start automatically when the system boots:

   ```
   # systemctl start firewalld
   # systemctl enable firewalld
   ```

3. Open the required ports using the **firewall-cmd** utility. For example, to open the Certificate System default ports in the default firewall zone:

   ```
   # firewall-cmd --permanent --add-port={8080/tcp,8443/tcp,8009/tcp,8005/tcp}
   ```

   For details on using **firewall-cmd** to open ports on a system, see the *Red Hat Enterprise Linux Security Guide* or the firewall–cmd(1) man page.

4. Reload the firewall configuration to ensure that the change takes place immediately:

   ```
   # firewall-cmd --reload
   ```

6.4. HARDWARE SECURITY MODULE

To use a Hardware Security Module (HSM), a Federal Information Processing Standard (FIPS) 140-2 validated HSM is required. See your HSM documentation for installing, configuring, and how to set up the HSM in FIPS mode.

6.4.1. Setting up SELinux for an HSM

Certain HSMs require that you manually update SELinux settings before you can install Certificate System.

The following section describes the required actions for supported HSMs:

nCipher nShield

After you installed the HSM and before you start installing Certificate System:

1. Reset the context of files in the **/opt/nfast/** directory:

    ```
    # restorecon -R /opt/nfast/
    ```

2. Restart the **nfast** software.

    ```
    # /opt/nfast/sbin/init.d-ncipher restart
    ```

Gemalto Safenet LunaSA HSM

No SELinux-related actions are required before you start installing Certificate System.

For details about the supported HSMs, see Section 4.3, "Supported Hardware Security Modules" .

6.4.2. Enabling FIPS Mode on an HSM

To enable FIPS Mode on HSMs, please refer to your HSM vendor's documentation for specific instructions.

> **IMPORTANT**
>
> **nCipher HSM**
>
> On a nCipher HSM, the FIPS mode can only be enabled when generating the Security World, this cannot be changed afterwards. While there is a variety of ways to generate the Security World, the preferred method is always to use the **new-world** command. For guidance on how to generate a FIPS-compliant Security World, please follow the nCipher HSM vendor's documentation.
>
> **LunaSA HSM**
>
> Similarly, enabling the FIPS mode on a Luna HSM must be done during the initial configuration, since changing this policy zeroizes the HSM as a security measure. For details, please refer to the Luna HSM vendor's documentation.

6.4.3. Verifying if FIPS Mode is Enabled on an HSM

This section describes how to verify if FIPS mode is enabled for certain HSMs. For other HSMs, see the hardware manufacturer's documentation.

6.4.3.1. Verifying if FIPS Mode is Enabled on an nCipher HSM

> **NOTE**
>
> Please refer to your HSM vendor's documentation for the complete procedure.

To verify if the FIPS mode is enabled on an nCipher HSM, enter:

```
# /opt/nfast/bin/nfkminfo
```

With older versions of the software, if the **StrictFIPS140** is listed in the state flag, the FIPS mode is enabled. In newer vesions, it is however better to check the new **mode** line and look for **fips1402level3**. In all cases, there should also be an **hkfips** key present in the **nfkminfo** output.

6.4.3.2. Verifying if FIPS Mode is Enabled on a Luna SA HSM

> **NOTE**
>
> Please refer to your HSM vendor's documentation for the complete procedure.

To verify if the FIPS mode is enabled on a Luna SA HSM:

1. Open the **lunash** management console

2. Use the **hsm show** command and verify that the output contains the text **The HSM is in FIPS 140-2 approved operation mode.**:

   ```
   lunash:> hsm show
   ...
       FIPS 140-2 Operation:
       =====================
       The HSM is in FIPS 140-2 approved operation mode.
   ...
   ```

6.4.4. Preparing for Installing Certificate System with an HSM

In Section 7.3, "Installing Using the **pkispawn** Utility", you are instructed to use the following parameters in the configuration file you pass to the **pkispawn** utility when installing Certificate System with an HSM:

```
...
[DEFAULT]
##########################
# Provide HSM parameters #
##########################
pki_hsm_enable=True
pki_hsm_libfile=hsm_libfile
pki_hsm_modulename=hsm_modulename
pki_token_name=hsm_token_name
pki_token_password=pki_token_password

#######################################
# Provide PKI-specific HSM token names #
#######################################
pki_audit_signing_token=hsm_token_name
pki_ssl_server_token=hsm_token_name
pki_subsystem_token=hsm_token_name
...
```

- The values of the *pki_hsm_libfile* and *pki_token_name* parameter depend on your specific HSM installation. These values allow the **pkispawn** utility to set up your HSM and enable Certificate System to connect to it.

- The value of the *pki_token_password* depends upon your particular HSM token's password. The password gives the **pkispawn** utility read and write permissions to create new keys on the HSM.

- The value of the *pki_hsm_modulename* is a name used in later **pkispawn** operations to identify the HSM. The string is an identifier you can set as whatever you like. It allows **pkispawn** and Certificate System to refer to the HSM and configuration information by name in later operations.

The following section provides settings for individual HSMs. If your HSM is not listed, consult your HSM manufacturer's documentation.

6.4.4.1. nCipher HSM Parameters

For a nCipher HSM, such as a nCipher nShield Connect 6000, set the following parameters:

```
pki_hsm_libfile=/opt/nfast/toolkits/pkcs11/libcknfast.so
pki_hsm_modulename=nfast
```

Note that you can set the value of *pki_hsm_modulename* to any value. The above is a suggested value.

Example 6.1. Identifying the Token Name

To identify the token name, run the following command as the **root** user:

```
[root@example911 ~]# /opt/nfast/bin/nfkminfo
World
 generation  2

...~snip~...

Cardset
 name          "NHSM6000-OCS"
 k-out-of-n    1/4
 flags        NotPersistent PINRecoveryRequired(enabled) !RemoteEnabled
 timeout      none

...~snip~...
```

The value of the *name* field in the **Cardset** section lists the token name.

Set the token name as follows:

```
pki_token_name=NHSM6000-OCS
```

6.4.4.2. SafeNet / Luna SA HSM Parameters

For a SafeNet / Luna SA HSM, such as a SafeNet Luna Network HSM, specify the following parameters:

```
pki_hsm_libfile=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
pki_hsm_modulename=lunasa
```

Note that you can set the value of ***pki_hsm_modulename*** to any value. The above is a suggested value.

Example 6.2. Identifying the Token Name

To identify the token name, run the following command as the **root** user:

```
# /usr/safenet/lunaclient/bin/vtl verify
```

The following Luna SA Slots/Partitions were found:

```
Slot    Serial #          Label
====    ================  =====
  0     1209461834772     lunasaQE
```

The value in the ***label*** column lists the token name.

Set the token name as follows:

```
pki_token_name=lunasaQE
```

6.4.5. Backing up Keys on Hardware Security Modules

It is not possible to export keys and certificates stored on an HSM to a **.p12** file. If such an instance is to be backed-up, contact the manufacturer of your HSM for support.

6.5. INSTALLING RED HAT DIRECTORY SERVER

Certificate System uses Red Hat Directory Server to store system certificates and user data. You can install both Directory Server and Certificate System on the same or any other host in the network.

6.5.1. Preparing a Directory Server Instance for Certificate System

Perform the following steps to install Red Hat Directory Server:

1. Attach a Directory Server subscription to the host.

2. Install the Directory Server and openldap-clients packages:

   ```
   # yum install redhat-ds openldap-clients
   ```

3. Set up a Directory Server instance. For example:

   a. Create a setup configuration file, such as **/tmp/ds-setup.inf** with the following content:

      ```
      [General]
      FullMachineName=server.example.com
      SuiteSpotUserID=dirsrv
      SuiteSpotGroup=dirsrv
      ```

```
[slapd]
ServerIdentifier=instance_name
ServerPort=389
Suffix=dc=example,dc=org
RootDN=cn=Directory Manager
RootDNPwd=password
```

b. Create the instance using the setup configuration file:

```
setup-ds.pl --silent --file=/tmp/ds-setup.inf
```

For a detailed procedure, see the *Red Hat Directory Server Installation Guide*.

6.5.2. Enabling TLS Support in Directory Server

Every Certificate System component communicates with the Directory Server instance using a TLS-encrypted connection where the Certificate System component is required to authenticate to the Directory Server using client-authentication (mutual authentication).

For details about enabling TLS support in Directory Server, see the *Enabling TLS in Directory Server* section in the *Directory Server Administration Guide*.

For details on how to request and issue a TLS server certificate for Directory Server, see the *Using a Certificate Issued by Certificate System in Directory Server* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

As described in the Directory Server documentation, you can configure TLS either using a certificate issued by an external Certificate Authority (CA) or a temporary self-signed server certificate. However, after setting up the Certificate System CA, you can use this CA to issue a certificate and replace it with the one used when you set up Directory Server.

6.5.2.1. How to Enable LDAPS for new Red Hat Certificate System Subsystems Using Examples Values

> **NOTE**
>
> When performing this TLS LDAP procedure, the final goal is to have the connection over TLS client authentication. During the process, we have to move along step by step, with intermediate goals. One such goal is to simply set TLS server authentication running first. At the end, the process will double-back to get full client authentication operational.

1. Stop the Directory Server instance to avoid concurrent changes to the NSS database:

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

2. Store the Directory Manager's password in the **/etc/dirsrv/*instance_name*/password.txt** file. For example:

```
# echo password > /etc/dirsrv/slapd-instance_name/password.txt
# chown dirsrv.dirsrv /etc/dirsrv/slapd-instance_name/password.txt
# chmod 400 /etc/dirsrv/slapd-instance_name/password.txt
```

3. Store the Directory Manager's password in the **/etc/dirsrv/*instance_name*/pin.txt** file. For example:

```
# echo "Internal (Software) Token:password" > /etc/dirsrv/slapd-instance_name/pin.txt
# chown dirsrv.dirsrv /etc/dirsrv/slapd-instance_name/pin.txt
# chmod 400 /etc/dirsrv/slapd-instance_name/pin.txt
```

4. Set the NSS database password:

```
# certutil -W -d /etc/dirsrv/slapd-instance_name/ -f
/etc/dirsrv/slapd-instance_name/password.txt
```

5. Create a temporary self-signed certificate for Directory Server:

```
$ cd /etc/dirsrv/slapd-instance_name
$ openssl rand -out noise.bin 2048
$ certutil -S \
 -x \
 -d . \
 -f password.txt \
 -z noise.bin \
 -n "DS Certificate" \
 -s "CN=$HOSTNAME" \
 -t "CT,C,C" \
 -m $RANDOM \
 -k rsa \
 -g 2048 \
 -Z SHA256 \
 --keyUsage certSigning,keyEncipherment
```

6. Verify whether the Directory Server certificate entry is available in the NSS database:

```
# certutil -L -d /etc/dirsrv/slapd-instance_name/
```

7. Export the certificate:

```
# certutil -L -d /etc/dirsrv/slapd-instance_name -n "DS Certificate" -a > ds.crt
```

8. Verify the Directory Server certificate is self-signed:

```
# certutil -L -d /etc/dirsrv/slapd-instance_name -n "DS Certificate"
Issuer: "CN=server.example.com"
Subject: "CN=server.example.com"
```

9. Start the Directory Server instance:

```
# systemctl start dirsrv@instance_name
```

10. Enable secure connection:

```
# ldapmodify -x -p 389 -h $HOSTNAME -D "cn=Directory Manager" -w password << EOF
dn: cn=config
```

```
changetype: modify
replace: nsslapd-security
nsslapd-security: on

dn: cn=RSA,cn=encryption,cn=config
changetype: add
objectclass: top
objectclass: nsEncryptionModule
cn: RSA
nsSSLPersonalitySSL: DS Certificate
nsSSLToken: internal (software)
nsSSLActivation: on
EOF
```

11. Optionally, set a different LDAPS port than the default (**636**).

    a. For example, to set the LDAPS port to **11636**:

    ```
    ldapmodify -x -p 389 -h $HOSTNAME -D "cn=Directory Manager" -w password << EOF
    dn: cn=config
    changetype: modify
    replace: nsslapd-secureport
    nsslapd-secureport: 11636
    EOF
    ```

    b. Set the SELinux policy for this non–standard port:

    ```
    # semanage port -a -t ldap_port_t -p tcp 11636
    ```

12. Restart the Directory Server instance:

    ```
    # systemctl restart dirsrv@instance_name
    ```

13. Verify in the **/var/log/dirsrv/slapd-instance_name/errors** file that Directory Server started in TLS mode:

    ```
    [30/Jun/2016:00:23:31 +0200] - SSL alert: Security Initialization: Enabling default cipher set.
    [30/Jun/2016:00:23:31 +0200] - SSL alert: Configured NSS Ciphers
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA: enabled
    [30/Jun/2016:00:23:31 +0200] - SSL alert:
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA: enabled
    ```

```
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_DHE_RSA_WITH_AES_128_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_DHE_DSS_WITH_AES_128_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_DHE_RSA_WITH_AES_256_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_DHE_DSS_WITH_AES_256_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256: enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_RSA_WITH_AES_128_GCM_SHA256:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_RSA_WITH_AES_128_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_RSA_WITH_AES_128_CBC_SHA256:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_RSA_WITH_AES_256_CBC_SHA:
enabled
[30/Jun/2016:00:23:31 +0200] - SSL alert:      TLS_RSA_WITH_AES_256_CBC_SHA256:
enabled
[30/Jun/2016:00:23:31 +0200] SSL Initialization - Configured SSL version range: min:
TLS1.0, max: TLS1.2
[30/Jun/2016:00:23:31 +0200] - 389-Directory/1.3.4.11 B2016.166.1911 starting up
```

14. Verify the TLS connection using the **openldap-clients** and NSS database:

```
$ LDAPTLS_CACERTDIR=/etc/dirsrv/slapd-instance_name \
 ldapsearch -H ldaps://$HOSTNAME:11636 \
 -x -D "cn=Directory Manager" -w Secret.123 \
 -b "dc=example,dc=org" -s base "(objectClass=*)"
```

6.5.3. Preparing for Configuring Certificate System

In Section 7.3, "Installing Using the **pkispawn** Utility", you are instructed to use the following parameters in the configuration file you pass to the **pkispawn** utility when installing Certificate System:

> **NOTE**
>
> We need to first create a basic TLS server authentication connection. At the end, during post-installation, we will return and make the connection require a client authentication certificate to be presented to Directory Server. At that time, once client authentication is set up, the *pki_ds_password* would no longer be relevant.

```
pki_ds_database=back_end_database_name
pki_ds_hostname=host_name
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
pki_ds_password=password
pki_ds_ldaps_port=port
pki_ds_bind_dn=cn=Directory Manager
```

The value of the ***pki_ds_database*** parameter is a name used by the **pkispawn** utility to create the corresponding subsystem database on the Directory Server instance.

The value of the ***pki_ds_hostname*** parameter depends on the install location of the Directory Server instance. This depends on the values used in Section 6.5.1, "Preparing a Directory Server Instance for Certificate System" and Section 6.5.2, "Enabling TLS Support in Directory Server".

When you set **pki_ds_secure_connection=True**, the following parameters must be set:

- ***pki_ds_secure_connection_ca_pem_file***: Sets the fully-qualified path including the file name of the file which contains an exported copy of the Directory Server's CA certificate. This file must exist prior to **pkispawn** being able to utilize it.

- ***pki_ds_ldaps_port***: Sets the value of the secure LDAPS port Directory Server is listening to. The default is **636**.

6.5.4. Replacing the Temporary Certificate

> **NOTE**
>
> This section requires a root CA installed and running. You will be instructed to follow instruction in this section during post-installation.

This section describes the process to replace the temporary self-signed Directory Server certificate with a permanent Directory Server certificate issued by the newly-installed CA, or to replace an old Directory Server certificate with a new one.

1. Obtain a new Directory Server server certificate. Submit the request for the new certificate signed by the CA. To get a new Directory Server certificate using the CMC method, see the *Obtaining System and Server Certificates* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

   In the above section, follow the guidance to create a TLS server certificate. Once the certificate is created, it must be imported back into the Directory Server's certificate database.

2. Stop the Directory Server instance before accessing the NSS database:

   ```
   # systemctl stop dirsrv@instance_name
   ```

3. Delete the old Directory Server certificate:

   ```
   # certutil -F -d /etc/dirsrv/slapd-instance_name -f
   /etc/dirsrv/slapd-instance_name/password.txt -n "DS Certificate"
   ```

4. Import the CA certificate downloaded earlier:

```
# PKICertImport -d /etc/dirsrv/slapd-instance_name -f
/etc/dirsrv/slapd-instance_name/password.txt -n "CA Certificate" -t "CT,C,C" -a -i ca.crt -u L
```

5. Import the new Directory Server certificate downloaded earlier:

```
# PKICertImport -d /etc/dirsrv/slapd-instance_name -f
/etc/dirsrv/slapd-instance_name/password.txt -n "DS Certificate" -t ",," -a -i ds.crt -u V
```

See also Section 10.4, "Importing a certificate into an HSM" .

6. Start the Directory Server instance:

```
systemctl start dirsrv@instance_name
```

7. Remove the old Directory Server Certificate from PKI CA:

   a. Stop the Certificate System instance:

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

   b. Remove the certificate:

```
# certutil -D -d /var/lib/pki/instance_name/alias/ -n "DS Certificate"
```

   c. Start the Certificate System instance:

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

8. Verify that the new Directory Server certificate is signed by the CA installed in the NSS database:

   a. Display the Directory Server certificate

```
$ certutil -L -d /etc/dirsrv/slapd-instance_name -n "DS Certificate"

Issuer: "CN=CA Signing Certificate,O=EXAMPLE"
Subject: "CN=server.example.com"
```

   b. Verify the old Directory Server certificate no longer exists in the PKI NSS database:

```
$ certutil -L -d /var/lib/pki/instance_name/alias
```

   c. Verify Certificate System can connect to Directory Server using the new certificate:

```
$ pki cert-find
```

6.5.5. Enabling TLS Client Authentication

> **NOTE**
>
> This section requires a root CA installed and running. If you used a temporary LDAP server certificate, replace it first by following Section 6.5.4, "Replacing the Temporary Certificate". Follow instruction in this section during post-installation.

The basic TLS server authentication has been configured and running during installation of the CS subsystem, we can now double back and attempt to enable client authentication from a given subsystem to the LDAP server.

There are two parts to setting up client authentication. The first part is configuring the LDAP directory to require TLS mutual authentication. This procedure is detailed in 9.8. Using Certificate-based Client Authentication in Red Hat Directory Server Administration Guide.

Note the following:

- **pkispawn** already automatically created a **pkidbuser** on its internal directory server, where the CS instance's "subsystem certificate" (for example **subsystemCert cert-pki-ca**) that is used for outbound TLS client authentication is stored in the user entry. Therefore, there is no need to create another LDAP user or another certificate for the TLS client-authentication.

- When creating content for **/etc/dirsrv/slapd-instance_name/certmap.conf**, use the following format:

  ```
  certmap rhcs <certificate issuer DN>
  rhcs:CmapLdapAttr seeAlso
  rhcs:verifyCert on
  ```

  For example

  ```
  certmap rhcs CN=CA Signing Certificate,OU=pki-tomcat-ca,O=pki-tomcat-ca-SD
  rhcs:CmapLdapAttr seeAlso
  rhcs:verifyCert on
  ```

- After configuring, restart the Directory Server.

The second part is adding configuration to the Red Hat Certificate System instance so that it knows which port and certificate is to be used to communicate with its internal LDAP server using TLS mutual authentication. This involves editing the RHCS instance's **CS.cfg** file located at **<instance directory>/<subsystem type>/conf/CS.cfg**. For example **/var/lib/pki/pki-tomcat/ca/conf/CS.cfg**

In the **CS.cfg**, please add the RHCS instance's subsystem certificate nickname to **internaldb.ldapauth.clientCertNickname**, and remove the two unused entries:

```
internaldb.ldapauth.bindDN
internaldb.ldapauth.bindPWPrompt
```

As shown below:

```
internaldb._000=##
internaldb._001=## Internal Database
internaldb._002=##
internaldb.basedn=o=pki-tomcat-ca-SD
internaldb.database=pki-tomcat-ca
```

```
internaldb.maxConns=15
internaldb.minConns=3
internaldb.ldapauth.authtype=SslClientAuth
internaldb.ldapauth.clientCertNickname=HSM-A:subsystemCert pki-tomcat-ca
internaldb.ldapconn.host=example.com
internaldb.ldapconn.port=11636
internaldb.ldapconn.secureConn=true
```

Restart the CS instance at the end of the Post-installation step.

> **NOTE**
>
> The ***internaldb.basedn*** and ***internaldb.database*** parameters must be configured to match your specific LDAP instance.
>
> For compliance, **internaldb.ldapauth.authtype=SslClientAuth** and **internaldb.ldapconn.secureConn=true** must be set, and the value of the ***internaldb.ldapauth.clientCertNickname*** must match the nickname of the TLS client certificate to authenticate against LDAP with in the NSS DB.
>
> All other values can be changed as desired to reflect your environment or availability needs.

6.6. ATTACHING A RED HAT SUBSCRIPTION AND ENABLING THE CERTIFICATE SYSTEM PACKAGE REPOSITORY

Before you can install and update Certificate System using the **yum** utility, you must enable the corresponding repository:

1. Attach the Red Hat subscriptions to the system:

   Skip this step, if your system is already registered or has a Certificate Server subscription attached.

   a. Register the system to the Red Hat subscription management service:

      ```
      # subscription-manager register --auto-attach
      Username: admin@example.com
      Password:
      The system has been registered with id: 566629db-a4ec-43e1-aa02-9cbaa6177c3f

      Installed Product Current Status:
      Product Name:        Red Hat Enterprise Linux Server
      Status:            Subscribed
      ```

      Use the **--auto-attach** option to automatically apply a subscription for the operating system.

   b. List the available subscriptions and note the pool ID providing the Red Hat Certificate System. For example:

      ```
      # subscription-manager list --available --all
      ...
      Subscription Name:   Red Hat Enterprise Linux Developer Suite
      Provides:            ...
      ```

> Red Hat Certificate System
>
> ...
> Pool ID: *7aba89677a6a38fc0bba7dac673f7993*
> Available: 1
> ...

In case you have a lot of subscriptions, the output of the command can be very long. You can optionally redirect the output to a file:

> # subscription-manager list --available --all > /root/subscriptions.txt

c. Attach the Certificate System subscription to the system using the pool ID from the previous step:

> # subscription-manager attach --pool=*7aba89677a6a38fc0bba7dac673f7993*
> Successfully attached a subscription for: Red Hat Enterprise Linux Developer Suite

2. Enable the Certificate Server repository:

> # subscription-manager repos --enable=rhel-7-server-rhcmsys-9-rpms
> Repository 'rhel-7-server-rhcmsys-9-rpms' is enabled for this system.

Installing the required packages is described in the Chapter 7, *Installing and Configuring Certificate System* chapter.

> **NOTE**
>
> For compliance, only enable Red Hat approved repositories. Only Red Hat approved repositories can be enabled through **subscription-manager** utility.

6.7. CERTIFICATE SYSTEM OPERATING SYSTEM USERS AND GROUPS

When installing Certificate System, the **pkiuser** account and the corresponding **pkiuser** group are automatically created. Certificate System uses this account and group to start services.

As part of a post-installation procedure, you will later be instructed to create an operating system user for each Certificate System administrator who should be able to start and stop or directly configure the instance on the operating system. For details, see Section 7.4, "Post-installation Tasks".

CHAPTER 7. INSTALLING AND CONFIGURING CERTIFICATE SYSTEM

Red Hat Certificate System provides different subsystems that can be installed individually. For example, you can install multiple subsystem instances on a single server or you can run them independently on different hosts. This enables you to adapt the installation to your environment to provide a higher availability, scalability, and failover support. This chapter describes the package installation and how to set up the individual subsystems.

The Certificate System includes the following subsystems:

- Certificate Authority (CA)

- Key Recovery Authority (KRA)

- Online Certificate Status Protocol (OCSP) Responder

- Token Key Service (TKS)

- Token Processing System (TPS)

7.1. SUBSYSTEM CONFIGURATION ORDER

The order in which the individual subsystems are set up is important because of relationships between the different subsystems:

1. At least one CA is required before any of the other public key infrastructure (PKI) subsystems can be installed.

2. Install the OCSP after the CA has been configured.

3. The KRA, and TKS subsystems can be installed in any order, after the CA and OCSP have been configured.

4. The TPS subsystem depends on the CA and TKS, and optionally on the KRA and OCSP subsystem.

> **NOTE**
>
> For a non-Token Management setup, you can install CA, OCSP, and KRA subsystems, while in a Token Management setup, you can install CA, OCSP, KRA, TKS, and TPS.

7.2. CERTIFICATE SYSTEM PACKAGES

When installing the Certificate System packages you can either install them for each subsystem individually or all at once.

> **IMPORTANT**
>
> To install and update Certificate Server packages, you must enable the corresponding repository. For details, see Section 6.6, "Attaching a Red Hat Subscription and Enabling the Certificate System Package Repository".

The following subsystem packages and components are available:

- pki-ca: Provides the Certificate Authority (CA) subsystem.

- pki-kra: Provides the Key Recovery Authority (KRA) subsystem.

- pki-ocsp: Provides the Online Certificate Status Protocol (OCSP) responder.

- pki-tks: Provides the Token Key Service (TKS).

- pki-tps: Provides the Token Processing Service (TPS).

- pki-console and redhat-pki-console-theme: Provides the Java-based Red Hat PKI console. Both packages must be installed.

- pki-server and redhat-pki-server-theme: Provides the web-based Certificate System interface. Both packages must be installed.

  This package is installed as a dependency if you install one of the following packages: pki-ca, pki-kra, pki-ocsp, pki-tks, pki-tps

7.2.1. Installing Certificate System Packages in non-TMS Environments

To install subsystems in a non-Token Management System (TMS) environment:

```
# yum install pki-ca redhat-pki-server-theme pki-console \
    redhat-pki-console-theme pki-kra pki-ocsp
```

Follow instructions in Section 7.2.4, "Determining Certificate System Product Version" to check the product version.

7.2.2. Installing Certificate System Packages in TMS Environments

To install subsystems in a Token Management System (TMS) environment:

```
# yum install redhat-pki
```

The redhat-pki installs all Certificate System subsystem packages and components automatically.

Follow instructions in Section 7.2.4, "Determining Certificate System Product Version" to check the product version.

7.2.3. Updating Certificate System Packages

To update Certificate System and operating system packages, use the following procedure:

1. Follow instructions in Section 7.2.4, "Determining Certificate System Product Version" to check the product version.

2. Execute **# yum update**

   The command above updates the whole system including the RHCS packages.

> **NOTE**
>
> We suggest scheduling a maintenance window during which you can take the PKI infrastructure offline to install the update.

> **IMPORTANT**
>
> Updating Certificate System requires the PKI infrastructure to be restarted.

3. Then check version again by following Section 7.2.4, "Determining Certificate System Product Version".

   The version number should confirm that the update was successfully installed.

To optionally download updates without installing, use the **--downloadonly** option in the above procedure:

```
yum update --downloadonly
```

The downloaded packages are stored in the **/var/cache/yum/** directory. The **yum update** will later use the packages if they are the latest versions.

7.2.4. 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***/tks/admin/tks/getStatus**

- **http://***host_name***:***port_number***/tps/admin/tps/getStatus**

> **NOTE**
>
> Note that each component is a separate package and thus could have a separate version number. The above will show the version number for each currently running component.

7.3. INSTALLING USING THE PKISPAWN UTILITY

Read the following section carefully to ensure that you make the right choices when following the installation examples in this section.

7.3.1. About the pkispawn Utility

In Red Hat Certificate System, administrators set up the individual public key infrastructure (PKI) subsystems by running the **pkispawn** utility.

The **pkispawn** utility requires a configuration file. This file contains parameters that are grouped into sections. These sections are stacked, so that parameters defined in earlier sections can be overridden by parameters defined in later sections. The sections are read in the following order:

1. **[DEFAULT]**

2. **[Tomcat]**

3. The subsystem-specific sections: **[CA]**, **[KRA]**, **[OCSP]**, **[TKS]**, or **[TPS]**

During the setup, **pkispawn**:

1. First reads the default values from the **/etc/pki/default.cfg** file.

   > **IMPORTANT**
   >
   > Do not edit the **/etc/pki/default.cfg** file. You will be instructed to create a configuration file that overrides the defaults when passed to the **pkispawn** utility. Parameters not set in the configuration file use the defaults. For details about using a configuration file, see Section 7.3.2, "Two-step Installation Using **pkispawn**".

2. Reads the administrator-provided configuration file mentioned above to override the default values.

3. Performs the installation of the requested PKI subsystem.

4. Passes the settings to a Java servlet that performs the configuration based on the settings.

For general information about **pkispawn** and examples, see the pkispawn(8) and pki_default.cfg(5) man pages.

For instructions about how to install a CA, KRA, or OCSP subsystem, see Section 7.3.2, "Two-step Installation Using **pkispawn**".

For instructions about how to install a TKS or TPS subsystem, see Section 7.3.3, "Single-step Installation Using **pkispawn** (TMS-only)".

7.3.2. Two-step Installation Using pkispawn

To install a CA, KRA, or OCSP subsystem, use the **pkispawn** two-step installation to enable the administrator to manually update configuration files between the two parts of the installation.

The name **two-step installation** comes from the fact that the **pkispawn** utility is often run twice with a slightly different set of parameters in its overwrite configuration file to complete the installation.

The two-step installation is consisted of the following major parts:

1. **Installation using the pkispawn Utility** (Step 1)

During this step, **pkispawn** copies configuration files from the **/usr/share/pki/** directory to the instance-specific **/etc/pki/***instance_name***/** directory. Additionally, **pkispawn** sets the settings based on values defined in the deployment configuration file.

This part of the installation contains the following substeps:

1. Section 7.3.2.1, "Creating the Configuration File for the First Step of the Installation"

2. Section 7.3.2.2, "Starting the **pkispawn** Step One Installation"

2. **Between-step configuration and customization**

For details, see Section 7.3.2.3, "Configuration Between the Two **pkispawn** Installation Steps".

3. **Configuration Using the pkispawn Utility** (Step 2)

During this step, **pkispawn** is run to continue the installation based on the configuration files in the instance-specific **/etc/pki/***instance_name***/** directory.

For details about this part of the installation see Section 7.3.2.4, "Starting the **pkispawn** Step Two Installation".

Depending on your environment, additional configuration and customization steps are required.

7.3.2.1. Creating the Configuration File for the First Step of the Installation

Create a text file for the **pkispawn** configuration settings, such as **/root/pki/config.subsystem.txt**, and fill it with the settings described below. Every setting must be added.

> **IMPORTANT**
>
> This section describes a minimum configuration with Directory Server running on the same host as Certificate System. Depending on your environment, additional parameters may be necessary. For additional examples, see the *EXAMPLES* section in the pkispawn(8) man page.

Subsystem-independent Settings

Independently of the subsystem you install, the following settings are required in the configuration file in the:

- **[DEFAULT]** section:

  1. Set a unique instance name:

     ```
     pki_instance_name=instance_name
     ```

  2. Set the host name and security domain name:

     ```
     pki_host=server.example.com
     pki_security_domain_name=example.com Security Domain
     pki_security_domain_user=caadmin
     pki_security_domain_password=SD password
     ```

> **NOTE**
>
> Certificate System creates unique certificate nicknames based on these parameters and the instance name. The uniqueness of certificate nicknames is very important in keeping the HSM token shared by multiple PKI instances functional.

3. Set the port numbers for the HTTP and HTTPS protocols:

   ```
   pki_https_port=port_number
   pki_http_port=port_number
   ```

> **IMPORTANT**
>
> To run more than one Certificate System instance on the same host, you must set ports in the **pki_https_port** and **pki_http_port** parameters that are not used by any other service on the host. By default, Certificate System uses port **8080** for HTTP and **8443** for HTTPS.

4. Set the HSM-specific parameters:

   ```
   pki_hsm_enable=True
   pki_hsm_libfile=HSM_PKCS11_library_path
   pki_hsm_modulename=HSM_module_name
   pki_token_name=HSM_token_name
   pki_token_password=HSM_token_password


   pki_audit_signing_token=HSM_token_name
   pki_subsystem_token=HSM_token_name
   pki_sslserver_token=HSM_token_name
   ```

   For further details, see Section 6.4.4, "Preparing for Installing Certificate System with an HSM".

5. If building an RSA Certificate System instance, use the following configuration:

   ```
   pki_admin_key_algorithm=SHA256withRSA
   pki_admin_key_size=4096
   pki_admin_key_type=rsa
   pki_sslserver_key_algorithm=SHA256withRSA
   pki_sslserver_key_size=4096
   pki_sslserver_key_type=rsa
   pki_subsystem_key_algorithm=SHA256withRSA
   pki_subsystem_key_size=4096
   pki_subsystem_key_type=rsa
   pki_audit_signing_key_algorithm=SHA256withRSA
   pki_audit_signing_key_size=4096
   pki_audit_signing_key_type=rsa
   pki_audit_signing_signing_algorithm=SHA256withRSA
   ```

   Allowed choices for the **key_algorithm** parameters are:

- **SHA256withRSA**

- **SHA384withRSA**

- **SHA512withRSA**

> **NOTE**
>
> You can set this parameter to each value specified in the *ca.profiles.defaultSigningAlgsAllowed* parameter in the CA's **CS.cfg** file. For details, see the *Signing Algorithm Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Signature algorithm (RSA) must match **key_type** (**rsa**).

Allowed choices for **key_type** is only **rsa**.

Allowed choices for **key_size** are **2048** and **4096**.

6. To use Elliptic Curve Cryptography (ECC) instead of RSA when creating certificates, set:

```
pki_admin_key_algorithm=SHA256withEC
pki_admin_key_size=nistp256
pki_admin_key_type=ecc
pki_sslserver_key_algorithm=SHA256withEC
pki_sslserver_key_size=nistp256
pki_sslserver_key_type=ecc
pki_subsystem_key_algorithm=SHA256withEC
pki_subsystem_key_size=nistp256
pki_subsystem_key_type=ecc
pki_audit_signing_key_algorithm=SHA256withEC
pki_audit_signing_key_size=nistp256
pki_audit_signing_key_type=ecc
pki_audit_signing_signing_algorithm=SHA256withEC
```

Allowed choices for the **key_algorithm** parameters are:

- **SHA256withEC**

- **SHA384withEC**

- **SHA512withEC**

> **NOTE**
>
> You can set this parameter to each value specified in the *ca.profiles.defaultSigningAlgsAllowed* parameter in the CA's **CS.cfg** file. For details, see the *Signing Algorithm Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Signature algorithm (EC) must match **key_type** (**ecc**).

Allowed choices for **key_size** are:

- **nistp256**

- **nistp384**

- **nistp521**

Allowed choice for **_key_type_** is **ecc**

7. Set the client directory of the bootstrap **admin** user:

   > pki_client_dir=_bootstrap_admin_directory_

   By default, the path is set to ~/**.dogtag**/**_instance_name_**/

   > **IMPORTANT**
   >
   > The **_pki_admin_*_** and **_pki_client_*_** parameters belong to the bootstrap user that is automatically created by the installation process. For further information about default roles (privileges) and the bootstrap user, see Section 2.6.6, "Users, Authorization, and Access Controls" .
   >
   > For descriptions about the parameters covered in this section, see the pki_default.cfg(5) man page.

8. Set various passwords for the bootstrap **admin** user:

   > pki_admin_password=_password_
   > pki_client_database_password=_password_
   > pki_client_pkcs12_password=_password_

9. Set the parameters for the LDAPS connection to Directory Server running on the same host:

   > pki_ds_database=_back-end database name_
   > pki_ds_hostname=_hostname_
   > pki_ds_secure_connection=True
   > pki_ds_secure_connection_ca_pem_file=_path_to_CA_or_self-signed_certificate_
   > pki_ds_password=_password_

10. If you use a self-signed certificate in Directory Server use the following command to export it from the Directory Server's Network Security Services (NSS) database:

    > # certutil -L -d /etc/dirsrv/slapd-_instance_name_/ \
    >   -n "server-cert" -a -o _/root/ds.crt_

- **[Tomcat]** section:

  1. Set the Tomcat JServ Protocol (AJP) port:

     > pki_ajp_port=_Tomcat_AJP_port_

  2. Set the Tomcat server port:

```
pki_tomcat_server_port=Tomcat_server_port
```

IMPORTANT

To run more than one PKI subsystem instance on the same host, you must set ports in the ***pki_ajp_port*** and ***pki_tomcat_server_port*** parameters that are not used by any other service on the host. By default, the ***pki_ajp_port*** is set to **8009** and ***pki_tomcat_server_port*** to **8005**.

For non-TMS, after you created the initial configuration file, add the subsystem-specific settings to it. See:

- the section called "CA Settings"

- the section called "Settings for Other Subsystems"

For TMS, to continue creating the configuration file for the **pkispawn** single-step Installation, see Section 7.3.3.1, "Creating the Configuration File for pkispawn single-step Installation" .

CA Settings

In addition to the section called "Subsystem-independent Settings", you need the following settings in the **[CA]** section to install a CA:

1. Enable random serial numbers by adding following setting:

   ```
   pki_random_serial_numbers_enable=true
   ```

2. Set the following parameters to specify the data of the bootstrap **admin** user, which will be automatically created during the installation:

   ```
   pki_admin_nickname=bootstrap_CA_admin
   pki_admin_name=bootstrap_CA_administrator_name
   pki_admin_uid=caadmin
   pki_admin_email=caadmin@example.com
   ```

   Certificate System assigns administrator and agent privileges to this bootstrap account. Use this account after the installation to create actual administrator, agent, and auditor accounts.

3. Specify the HSM token for the subsystem-dependent certificates:

   ```
   pki_ca_signing_token=HSM_token_name
   pki_ocsp_signing_token=HSM_token_name
   ```

4. If building an RSA-based CA, use the following configuration:

   ```
   pki_ca_signing_key_algorithm=SHA256withRSA
   pki_ca_signing_key_size=4096
   pki_ca_signing_key_type=rsa
   pki_ca_signing_signing_algorithm=SHA256withRSA
   pki_ocsp_signing_key_algorithm=SHA256withRSA
   pki_ocsp_signing_key_size=4096
   pki_ocsp_signing_key_type=rsa
   pki_ocsp_signing_signing_algorithm=SHA256withRSA
   ```

Allowed choices for the *key_algorithm* parameters are:

- **SHA256withRSA**

- **SHA384withRSA**

- **SHA384withRSA**

> **NOTE**
>
> You can set this parameter to each value specified in the *ca.profiles.defaultSigningAlgsAllowed* parameter in the CA's **CS.cfg** file. For details, see the *Signing Algorithm Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Signature algorithm (RSA) must match *key_type* (**rsa**).

Allowed choices for *key_size* parameters are:

- **2048**

- **4096**

Allowed choices for *key_type* parameters are only **rsa**.

5. By default, the key type for system certificates is **rsa** and the key length is **2048** bit. To use Elliptic Curve Cryptography (ECC) instead of RSA when creating certificates, set:

```
pki_ocsp_signing_key_algorithm=SHA256withEC
pki_ocsp_signing_key_size=nisp256
pki_ocsp_signing_key_type=ecc
pki_ocsp_signing_signing_algorithm=SHA256withEC
```

Allowed choices for *key_algorithm* parameters are:

- **SHA256withEC**

- **SHA384withEC**

- **SHA512withEC**

> **NOTE**
>
> You can set this parameter to each value specified in the *ca.profiles.defaultSigningAlgsAllowed* parameter in the CA's **CS.cfg** file. For details, see the *Signing Algorithm Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Signature algorithm (EC) must match *key_type* (**ecc**).

Allowed choices for *key_size* parameters are:

- **nistp256**

- **nistp384**

- **nistp521**

Allowed choices for *key_type* parameters is **ecc**.

Settings for Other Subsystems

In addition to the section called "Subsystem-independent Settings", you need the following settings in the **[CA]**, **[KRA]**, or **[OCSP]** section to install a subordinate CA, KRA, or OCSP:

1. Subsystems that are not a root CA use the two-step installation mechanism that is sometimes referred to as *External CA*. For the first step, add the following parameters under each respective subsystem-specific section:

   ```
   pki_external=True
   pki_external_step_two=False
   ```

2. This step depends on the subsystem you are installing:

   ### Subordinate CA

   Add the following settings to the **[CA]** section:

   1. Add all settings from the **[CA]** section of the root CA.

   2. Set the output paths for the system certificate signing requests (CSR):

      ```
      pki_ca_signing_csr_path=path
      pki_ocsp_signing_csr_path=path
      pki_audit_signing_csr_path=path
      pki_sslserver_csr_path=path
      pki_subsystem_csr_path=path
      pki_admin_csr_path=path
      ```

   ### OCSP

   Set the following in the **[OCSP]** section:

   1. Specify the following parameters to set the data of the bootstrap OCSP **admin** user which is automatically created during the installation:

      ```
      pki_admin_nickname=bootstrap_OCSP_admin
      pki_admin_name=bootstrap_OCSP_administrator_name
      pki_admin_uid=ocspadmin
      pki_admin_email=ocspadmin@example.com
      ```

      Certificate System assigns administrator and agent privileges to this bootstrap account. Use this account after the installation to create actual administrator, agent, and auditor accounts.

   2. Specify the HSM token for the subsystem-dependent certificates:

      ```
      pki_ocsp_signing_token=HSM_token_name
      ```

   3. If building an RSA Certificate System instance, use the following configuration:

      ```
      pki_ocsp_signing_key_algorithm=SHA256withRSA
      ```

```
pki_ocsp_signing_key_size=4096
pki_ocsp_signing_key_type=rsa
pki_ocsp_signing_signing_algorithm=SHA256withRSA
```

Allowed choices for the ***key_algorithm*** parameters are:

- **SHA256withRSA**

- **SHA384withRSA**

- **SHA384withRSA**

> **NOTE**
>
> You can set this parameter to each value specified in the ***ca.profiles.defaultSigningAlgsAllowed*** parameter in the CA's **CS.cfg** file. For details, see the *Signing Algorithm Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

Signature algorithm (RSA) must match ***key_type*** (**rsa**).

Allowed choices for ***key_size*** parameters are:

- **2048**

- **4096**

Allowed choices for ***key_type*** parameters are only **rsa**.

> **NOTE**
>
> It is suggested to match your CA instance type (RSA or ECC) with your OCSP responder type (RSA or ECC).

4. To use Elliptic Curve Cryptography (ECC) instead of RSA when creating certificates, set:

```
pki_ocsp_signing_key_algorithm=SHA256withEC
pki_ocsp_signing_key_size=nisp256
pki_ocsp_signing_key_type=ecc
pki_ocsp_signing_signing_algorithm=SHA256withEC
```

Allowed choices for the ***key_algorithm*** parameters are:

- **SHA256withEC**

- **SHA384withEC**

- **SHA512withEC**

> **NOTE**
>
> You can set this parameter to each value specified in the
> ***ca.profiles.defaultSigningAlgsAllowed*** parameter in the CA's **CS.cfg**
> file. For details, see the *Signing Algorithm Constraint* section in the
> *Red Hat Certificate System Administration Guide (Common Criteria
> Edition)*.

Signature algorithm (EC) must match ***key_type*** (**ecc**).

Allowed choices for ***key_size*** parameters are:

- **nistp256**

- **nistp384**

- **nistp521**

Allowed choices for ***key_type*** parameters are only **ecc**.

> **NOTE**
>
> It is suggested to match your CA instance type (RSA or ECC) with your
> OCSP responder type (RSA or ECC).

5. Set the output paths for the system certificate signing requests (CSR):

   ```
   pki_ocsp_signing_csr_path=path
   pki_audit_signing_csr_path=path
   pki_sslserver_csr_path=path
   pki_subsystem_csr_path=path
   pki_admin_csr_path=path
   ```

KRA

Set the following in the **[KRA]** section:

1. Specify the following parameters to set the data of the bootstrap KRA **admin** user which is automatically created during the installation:

   ```
   pki_admin_nickname=bootstrap_KRA_admin
   pki_admin_name=bootstrap_KRA_administrator_name
   pki_admin_uid=kraadmin
   pki_admin_email=kraadmin@example.com
   ```

   Certificate System assigns administrator and agent privileges to this bootstrap account. Use this account after the installation to create actual administrator, agent, and auditor accounts.

2. Specify the HSM token for the subsystem-dependent certificates:

   ```
   pki_transport_token=HSM_token_name
   pki_storage_token=HSM_token_name
   ```

3. RSA is the only key type supported for both the KRA storage and transport certificates. By default, the key length is **2048** bit. To choose a **4096** bit key instead, add the following parameters:

```
pki_transport_key_size=4096
pki_storage_key_size=4096
```

4. Set the output paths for the system certificate signing requests (CSR):

```
pki_kra_signing_csr_path=path
pki_audit_signing_csr_path=path
pki_sslserver_csr_path=path
pki_subsystem_csr_path=path
pki_admin_csr_path=path
```

7.3.2.2. Starting the **pkispawn** Step One Installation

After you prepared the configuration file as described in Section 7.3.2.1, "Creating the Configuration File for the First Step of the Installation", start the first step of the installation:

- For a root CA:

```
# pkispawn -f /root/pki/config.root-CA.txt -s CA
```

- For a subordinate CA or other non-CA subsystems:

```
# pkispawn -f /root/pki/config.subsystem.txt -s subsystem
```

Replace *subsystem* with one of the following subsystems: **CA**, **KRA**, or **OCSP**. For example, for a KRA installation:

```
# pkispawn -f /root/pki/config.KRA.txt -s KRA
```

7.3.2.3. Configuration Between the Two **pkispawn** Installation Steps

After the installation step described in Section 7.3.2.2, "Starting the **pkispawn** Step One Installation" has finished successfully, you can manually update the instance-specific configuration files before Section 7.3.2.4, "Starting the **pkispawn** Step Two Installation" is performed, where the actual configuration begins.

For required procedures, perform the actions described in this section after you have completed step 1 of the **pkispawn** procedure.

For optional procedures, see Part III, "Configuring Certificate System". Useful between-installation-step procedures include:

- Section 11.1.3, "Configuring CA System Certificate Profiles"

7.3.2.3.1. Enabling Signed Audit Logging

The signed audit logging feature enables the prevention of unauthorized log manipulation.

For details, see Section 13.3.1, "Enabling and Configuring Signed Audit Log" .

7.3.2.3.2. Setting the KRA into Encryption Mode

If you are using a Hardware Security Module (HSM) that does not support secure key extraction, it is necessary to set the Key Recovery Authority (KRA) into encryption mode. For details, see Section 12.2.2.2, "Solving Limitations of HSMs When Using AES Encryption in KRAs" .

7.3.2.3.3. Enabling OCSP

For details about enabling the Online Certificate Status Protocol (OCSP) see Section 9.4.1.2, "Enabling Certificate Revocation Checking for Subsystems".

7.3.2.4. Starting the **pkispawn** Step Two Installation

After you have completed Section 7.3.2.3, "Configuration Between the Two **pkispawn** Installation Steps", start the second step of the **pkispawn** installation:

- Root CA:

    ```
    # pkispawn -f /root/pki/config.root-CA.txt -s CA --skip-installation
    ```

    If the configuration step succeeds, the **pkispawn** utility displays an installation summary. For example:

    ```
    ==================================================================
                  INSTALLATION SUMMARY
    ==================================================================

      Administrator's username:         caadmin
      Administrator's PKCS #12 file:
          /root/.dogtag/instance_name/ca_admin_cert.p12

      To check the status of the subsystem:
          systemctl status pki-tomcatd@instance_name.service

      To restart the subsystem:
          systemctl restart pki-tomcatd@instance_name.service

      The URL for the subsystem is:
          https://server.example.com:8443/ca/

      PKI instances will be enabled upon system boot


    ==================================================================
    ```

- Subordinate CA or other non-CA subsystems:

    1. Submit the CSRs to a CA to issue the certificates.

        One major difference between installing non-root CA subsystems and a root CA is that prior to running step two of the **pkispawn** utility, other subsystems require submitting the system certificate signing requests (CSR) to a CA:

        In the first step of the section called "Settings for Other Subsystems" , you added the

*_csr_path parameters in the **pkispawn** configuration file for the subsystems to be installed. If the first step of **pkispawn** succeeded, Certificate System generated and stored the CSRs in the specified paths. These certificate requests are in PKCS#10 format and you must convert them to Certificate Management over CMS (CMC) format before submitting them to the CA. Follow instructions specified at 4.2. Creating Certificate Signing Requests in Red Hat Certificate System Administration Guide. You need the resulting system certificates when you execute the second step of the **pkispawn** command.

Additionally, you need the issuing CA's certificate chain in PKCS#7 format. To retrieve it:

a. Access the issuing CA's EE URL.

b. Navigate to **Retrieval → Import CA Certificate Chain** and click **Display the CA certificate chain in PKCS#7 for importing into a server**.

c. Click **Submit**.

d. Copy the displayed PKCS#7-formatted output into a text file, such as **/root/pki/cert_chain.p7b**, on the Certificate System host.

2. Create a configuration file for the second step of the **pkispawn** command:

a. Copy the **pkispawn** configuration file from the first step and edit the newly created file for step two of the **pkispawn** command:

i. In the **[DEFAULT]** section, set the path to the issuing CA's certificate chain file you prepared in the previous step and the CA signing certificate's nickname. For example:

```
[DEFAULT]
pki_ca_signing_nickname=CA Signing Certificate
pki_ca_signing_cert_path=/root/pki/cert_chain.p7b
```

ii. Add the following settings to the subsystem-specific section (**[CA]**, **[KRA]**, or **[OCSP]**) depending on the subsystem you install:

Subordinate CA

```
pki_ca_signing_crt_path=/root/pki/subsystem/ocsp_signing.crt
pki_subsystem_crt_path=/root/pki/subsystem/subsystem.crt
pki_sslserver_crt_path=/root/pki/subsystem/sslserver.crt
pki_audit_signing_crt_path=/root/pki/subsystem/ca_audit_signing.crt
pki_admin_crt_path=/root/pki/subsystem/ca_admin.crt
pki_external_step_two=True
```

OCSP

```
pki_ocsp_signing_crt_path=/root/pki/subsystem/ocsp_signing.crt
pki_subsystem_crt_path=/root/pki/subsystem/subsystem.crt
pki_sslserver_crt_path=/root/pki/subsystem/sslserver.crt
pki_audit_signing_crt_path=/root/pki/subsystem/ocsp_audit_signing.crt
pki_admin_crt_path=/root/pki/subsystem/ocsp_admin.crt
pki_external_step_two=True
```

KRA

```
pki_storage_crt_path=/root/pki/subsystem/kra_storage.crt
pki_transport_crt_path=/root/pki/subsystem/kra_transport.crt
pki_subsystem_crt_path=/root/pki/subsystem/subsystem.crt
pki_sslserver_crt_path=/root/pki/subsystem/sslserver.crt
pki_audit_signing_crt_path=/root/pki/subsystem/kra_audit_signing.crt
pki_admin_crt_path=/root/pki/subsystem/kra_admin.crt
pki_external_step_two=True
```

3. Run the *pkispawn* command:

```
# pkispawn -f /root/pki/config.subsystem.txt -s subsystem
```

Replace *subsystem* with one of the following subsystems: **CA**, **KRA**, or **OCSP**. For example, for a KRA installation:

```
# pkispawn -f /root/pki/config.KRA.txt -s KRA
```

7.3.3. Single-step Installation Using `pkispawn` (TMS-only)

TPS and TKS subsystems can be installed by using the **pkispawn single-step installation** process.

pkispawn requires a configuration file.

7.3.3.1. Creating the Configuration File for pkispawn single-step Installation

Follow the section called "Subsystem-independent Settings" to fill in the subsystem-independent part of the **pkispawn** configuration file.

In addition, under the **[DEFAULT]** section add the following:

```
pki_security_domain_hostname=example.redhat.com
pki_security_domain_https_port=8443
pki_security_domain_user=caadmin
pki_security_domain_password=[SD password]
```

Then add the subsystem-specific section (**[TKS]** or **[TPS]**) as following:

TKS

Set the following in the **[TKS]** section:

- Specify the following parameters to set the data of the bootstrap **TKS** admin user which is automatically created during the installation:

```
pki_admin_nickname=bootstrap_TKS_admin
pki_admin_name=bootstrap_TKS_administrator_name
pki_admin_uid=tksadmin
pki_admin_email=tksadmin@example.com
```

Red Hat Certificate System assigns administrator and agent privileges to this bootstrap account. Use this account after the installation to create the administrator, and auditor accounts.

TPS

Set the following in the **[TPS]** section:

1. Specify the following parameters to set the data of the bootstrap **TPS** admin user, which is automatically created during the installation:

   ```
   pki_admin_nickname=bootstrap_TPS_admin
   pki_admin_name=bootstrap_TPS_administrator_name
   pki_admin_uid=tpsadmin
   pki_admin_email=tpsadmin@example.com
   ```

   Red Hat Certificate System assigns administrator and agent privileges to this bootstrap account. Use this account after the installation to create the administrator, and the auditor accounts.

2. Set the LDAP base DN of the **TPS** authentication database:

   ```
   pki_authdb_basedn=base_DN_of_the_TPS_authentication_database
   ```

3. Enable the server-side key generation and archival:

   ```
   pki_enable_server_side_keygen=True
   ```

7.3.3.2. Running pkispawn for single-step Installation

Run the pkispawn command:

 # pkispawn -f /root/pki/config.TKS.txt -s TKS

or

 # pkispawn -f /root/pki/config.TPS.txt -s TPS

7.3.3.3. Post-Installation for Single-Step Installation

Follow the procedures below:

- Section 7.3.2.3.1, "Enabling Signed Audit Logging"

- Section 9.4.1.1, "TLS Cipher Configuration"

- Section 7.3.2.3.3, "Enabling OCSP"

Once you completed the procedures above, follow Section 7.4, "Post-installation Tasks" for additional post-installation actions.

7.4. POST-INSTALLATION TASKS

Once installation using the **pkispawn** utility is complete, certain actions are required after the installation. In addition, some optional actions would also be helpful, depending on the site's preferences.

For optional procedures, see Part III, "Configuring Certificate System". Useful post-installation-step procedures include:

- Configuring or adding certificate enrollment profiles (CA). For details, see Section 11.1, "Creating and Editing Certificate Profiles Directly on the File System"

For required procedures, perform the actions described below in Section 7.4, "Post-installation Tasks" after you have installed Certificate System.

7.4.1. Setting Date/Time for RHCS

It is important that the time is correct for running RHCS; see Chapter 15. Setting Time and Date in Red Hat Enterprise Linux 7.6 in Red Hat Certificate System's Administration Guide.

7.4.2. Replacing a Temporary Self-Signed Certificate in Directory Server (CA)

When the internal LDAP server was created initially with a temporary self-signed server certificate, this is time to replace it with a new certificate that is issued by the CA you just installed.

For details, see Section 6.5.4, "Replacing the Temporary Certificate".

7.4.3. Enabling TLS Client Authentication for the Internal LDAP Server

Red Hat Certificate System is required to communicate with its internal LDAP server via TLS mutual authentication. For further details see Enabling TLS Client Authentication.

7.4.4. Configuring Session Timeout

Various timeout configurations exist on the system that could affect how long a TLS session is allowed to remain idle before termination. For details, see Section 9.4.1.3, "Session Timeout".

7.4.5. CRL or Certificate Publishing

CRL publishing is critical in providing OCSP service. Certificate publishing is optional but often desired by sites. For details, see Chapter 7. Publishing Certificates and CRLs in Red Hat Certificate System Administration Guide.

7.4.6. Disabling Certificate Enrollment Profiles (CA)

Only CMC certificate enrollment profiles are allowed. All other profiles need to be disabled.

For details, see Section 11.1.5, "Disabling Certificate Enrolment Profiles".

7.4.7. Enabling Access Banner

User interface banners are required.

For details, see Section 9.5.1, "Enabling an Access Banner".

7.4.8. Enabling the Watchdog Service

The watchdog (**nuxwdog**) service provides secure system password management.

For details, see Section 9.3.2.1, "Enabling the Watchdog Service".

7.4.9. Configuration for CMC Enrollment and Revocation (CA)

Certificate enrollments and revocation have to be done via CMC.

- For details about enabling the CMC Shared Token Feature, see Section 9.6.3, "Enabling the CMC Shared Secret Feature".

- For details about enabling the **PopLinkWittness** feature, see Section 9.6.2, "Enabling the **PopLinkWittnessV2** Feature".

- For details about enabling **CMCRevoke** for the web user interface, see Section 9.6.4, "Enabling CMCRevoke for the Web User Interface".

7.4.10. Requiring TLS client-authentication for the Java Console

Certificate System administrators are required to present a user TLS client certificate when logging into the Java console. See Section 9.2.3.14, "Setting Requirement for **pkiconsole** to use TLS Client Certificate Authentication".

7.4.11. Creating a Role User

Real role users have to be created so the bootstrap user could be removed.

Create users and assign them to different privileged roles to manage Certificate System. See Chapter 14, *Creating a Role User* .

7.4.12. Removing the Bootstrap User

Bootstrap user is to be removed once the real role users are created.

After creating a new administrator account which is assigned to an individual person, remove the account which was automatically created during the installation. For details, see Chapter 15, *Deleting the Bootstrap User*.

7.4.13. Disabling Multi-role Support

Once the bootstrap user is removed, the multi-role support needs to be disabled.

For details, see Section 15.1, "Disabling Multi-roles Support" .

7.4.14. KRA Configurations

7.4.14.1. Adding Requirement for Multiple Agent Approval for Key Recovery Authority (KRA)

Multiple KRA agents are required to approve key recovery.

For details, see Section 12.3.2, "Configuring Agent-Approved Key Recovery in the Command Line" .

7.4.14.2. Configuring KRA Encryption Settings

Only certain key encryption/wrapping algorithms are allowed. For details, see Section 12.2, "Encryption Of KRA Operations".

7.4.15. Setting up Users to use User Interfaces

Before a user could use an approved user interface, initialization needs to be performed.

Users (administrative roles or otherwise) are required to setup their clients for accessing the user interface. See 2.1. Client NSS Database Initialization in Red Hat Certificate System's Administration Guide.

CHAPTER 8. TROUBLESHOOTING INSTALLATION

This chapter covers some of the more common installation and migration issues that are encountered when installing Certificate System.

8.1. FREQUENTLY ASKED QUESTIONS

8.1.1. Installation

Q: **I cannot see any Certificate System packages or updates.**

A: Verify that your system is correctly registered to the Red Hat subscription management service, a valid subscription is assigned, and the Certificate System repository is enabled. For details, see Section 6.6, "Attaching a Red Hat Subscription and Enabling the Certificate System Package Repository".

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. To see what errors have occurred, examine the **journal** log by running the following command:

> journalctl -u pki-tomcatd@*instance_name*.service

OR

> journalctl -u pki-tomcatd-nuxwdog@*instance_name*.service (if using the nuxwdog watchdog)

Alternatively, examine the debug log files at **/var/log/pki/***instance_name*/*subsystem_type*/**debug**.

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, set the correct classpath in the /etc/**sysconfig**/*instance_name* configuration file. Then restart the CA using the following command:

> systemctl restart pki-tomcatd@*instance_name*.service

OR

> systemctl restart pki-tomcatd-nuxwdog@*instance_name*.service (if using the nuxwdog watchdog)

Q: **I want to customize the subject name for the CA signing certificate, but do not see a way to do this using the pkispawn interactive install mode.**

A: To do this, a configuration file representing delta links to the /etc/pki/default.cfg file is required. See the pkispawn(8) and pki_default.cfg(5) man pages.

Q: **I am seeing an HTTP 500 error code when I try to connect to the web services pages after configuring my subsystem instance.**

A: This is an unexpected generic error which can have many different causes. Check in the **journal**, **system**, and **debug** log files for the instance to see what errors have occurred. This lists a couple of common errors, but there are many other possibilities.

Error #1: The LDAP database is not running.
If the Red Hat Directory Server instance use for the internal database is not running, then you cannot connect to the instance. This will be apparent in exceptions in the **journal** file that the instance is not ready:

> java.io.IOException: CS server is not ready to serve.
>   com.netscape.cms.servlet.base.CMSServlet.service(CMSServlet.java:409)
>   javax.servlet.http.HttpServlet.service(HttpServlet.java:688)

The Tomcat logs will specifically identify the problem with the LDAP connection:

> 5558.main - [29/Oct/2010:11:13:40 PDT] [8] [3] In Ldap (bound) connection pool
> to host ca1 port 389, Cannot connect to LDAP server. Error:
> netscape.ldap.LDAPException: failed to connect to server
> ldap://ca1.example.com:389 (91)

As will the instance's **debug** log:

> [29/Oct/2010:11:39:10][main]: CMS:Caught EBaseException
> Internal Database Error encountered: Could not connect to LDAP server host
> ca1 port 389 Error netscape.ldap.LDAPException: failed to connect to
> server ldap://ca1:389 (91)
>   at com.netscape.cmscore.dbs.DBSubsystem.init(DBSubsystem.java:262)

Error #2: A VPN is blocking access.
Another possibility is that you are connecting to the subsystem over a VPN. The VPN **must** have a configuration option like **Use this connection only for resources on its network** enabled. If that option is not enabled, then the **journal** log file for the instance's Tomcat service shows a series of connection errors that result in the HTTP 500 error:

```
May 26, 2010 7:09:48 PM org.apache.catalina.core.StandardWrapperValve invoke
SEVERE: Servlet.service() for servlet services threw exception
java.io.IOException: CS server is not ready to serve.
     at com.netscape.cms.servlet.base.CMSServlet.service(CMSServlet.java:441)
     at javax.servlet.http.HttpServlet.service(HttpServlet.java:803)
     at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:269)

     at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
     at
org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:210)
     at
org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:172)
     at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:127)
     at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:117)
     at org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:542)
     at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:108)
     at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:151)
     at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:870)
     at
org.apache.coyote.http11.Http11BaseProtocol$Http11ConnectionHandler.processConnection(Htt
p11BaseProtocol.java:665)
     at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:528)
     at
org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.jav
a:81)
     at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:685)
  at java.lang.Thread.run(Thread.java:636)
```

8.1.2. Java Console

Q: I cannot open the **pkiconsole** and I am 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 1.7.

Q: I tried to run **pkiconsole**, and I got Socket exceptions in stdout. Why?

A: This means that there is a port problem. Either there are incorrect TLS settings for the administrative port (meaning there is bad configuration in the **server.xml**) or the wrong port was given to access the admin interface.

Port errors will look like the following:

```
NSS Cipher Supported '0xff04'
java.io.IOException: SocketException cannot read on socket
     at org.mozilla.jss.ssl.SSLSocket.read(SSLSocket.java:1006)
     at org.mozilla.jss.ssl.SSLInputStream.read(SSLInputStream.java:70)
     at
com.netscape.admin.certsrv.misc.HttpInputStream.fill(HttpInputStream.java:303)
     at
com.netscape.admin.certsrv.misc.HttpInputStream.readLine(HttpInputStream.java:224)
     at
```

```
com.netscape.admin.certsrv.connection.JSSConnection.readHeader(JSSConnection.java:439)
        at
com.netscape.admin.certsrv.connection.JSSConnection.initReadResponse(JSSConnection.java:4
30)
        at
com.netscape.admin.certsrv.connection.JSSConnection.sendRequest(JSSConnection.java:344)

        at
com.netscape.admin.certsrv.connection.AdminConnection.processRequest(AdminConnection.java
:714)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:62
3)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:59
0)
        at
com.netscape.admin.certsrv.connection.AdminConnection.authType(AdminConnection.java:323)

        at
com.netscape.admin.certsrv.CMSServerInfo.getAuthType(CMSServerInfo.java:113)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:499)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:548)
        at com.netscape.admin.certsrv.Console.main(Console.java:1655)
```

Q: I attempt to start the console, and the system prompts me for my user name and password. After I enter these credentials, the console fails to appear.

A: Make sure the user name and password you entered are valid. If so, enable the debug output and examine it.

To enable the debug output, open the **/usr/bin/pkiconsole** file, and add the following lines:

```
=============================================
${JAVA} ${JAVA_OPTIONS} -cp ${CP} -Djava.util.prefs.systemRoot=/tmp/.java -
Djava.util.prefs.userRoot=/tmp/java com.netscape.admin.certsrv.Console -s instanceID -D 9:all
-a $1
----------
note: "-D 9:all" is for verbose output on the console.
=============================================
```

8.2. HARDWARE SECURITY MODULES

8.2.1. Detecting Tokens

Once you have installed Certificate System, to see if a token can be detected, use the **TokenInfo** utility, pointing to the **alias** directory for the subsystem instance. This is a Certificate System tool which is available after the Certificate System packages are installed.

For example:

> # TokenInfo /var/lib/pki/pki-tomcat/alias

This utility returns all tokens which can be detected by the Certificate System, not only tokens which are installed in the Certificate System.

8.2.2. Viewing Tokens

Once you have installed Certificate System, to view the list of the tokens, use the **modutil** utility.

1. Change to the instance **alias** directory. For example:

   > # cd /var/lib/pki/pki-tomcat/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

PART III. CONFIGURING CERTIFICATE SYSTEM

CHAPTER 9. THE CERTIFICATE SYSTEM CONFIGURATION FILES

The primary configuration file for every subsystem is its **CS.cfg** file. This chapter covers basic information about and rules for editing the **CS.cfg** file. This chapter also describes some other useful configuration files used by the subsystems, such as password and web services files.

9.1. FILE AND DIRECTORY LOCATIONS FOR CERTIFICATE SYSTEM SUBSYSTEMS

Certificate System servers consist of an Apache Tomcat instance, which contains *one or more* subsystems. Each subsystem consists of a web application, which handles requests for a specific type of PKI function.

The available subsystems are: CA, KRA, OCSP, TKS, and TPS. Each instance can contain only one of each type of a PKI subsystem.

A subsystem can be installed within a particular instance using the **pkispawn** command.

9.1.1. Instance-specific Information

For instance information for the default instance (pki-tomcat), see Table 2.2, "Tomcat Instance Information"

Table 9.1. Certificate Server Port Assignments (Default)

Port Type	Port Number	Notes
Secure port	8443	Main port used to access PKI services by end-users, agents, and admins over HTTPS.
Insecure port	8080	Used to access the server insecurely for some end-entity functions over HTTP. Used for instance to provide CRLs, which are already signed and therefore need not be encrypted.
AJP port	8009	Used to access the server from a front end Apache proxy server through an AJP connection. Redirects to the HTTPS port.
Tomcat port	8005	Used by the web server.

9.1.2. CA Subsystem Information

This section contains details about the CA subsystem, which is one of the possible subsystems that can be installed as a web application in a Certificate Server instance.

Table 9.2. CA Subsystem Information for the Default Instance (pki-tomcat)

Setting	Value	
Main directory	/var/lib/pki/pki-tomcat/ca/	
Configuration directory	/var/lib/pki/pki-tomcat/ca/conf/[a]	
Configuration file	/var/lib/pki/pki-tomcat/ca/conf/CS.cfg	
Subsystem certificates	CA signing certificate	
	OCSP signing certificate (for the CA's internal OCSP service)	
	TLS server certificate	
	Audit log signing certificate	
	Subsystem certificate[b]	
Security databases	/var/lib/pki/pki-tomcat/alias/[c]	
Log files	/var/lib/pki/pki-tomcat/ca/logs/[d]	
Install log	/var/log/pki/pki-ca-spawn.*date*.log	
Uninstall log	/var/log/pki/pki-ca-destroy.*date*.log	
Audit logs	/var/log/pki/pki-tomcat/ca/signedAudit/	
Profile files	/var/lib/pki/pki-tomcat/ca/profiles/ca/	
Email notification templates	/var/lib/pki/pki-tomcat/ca/emails/	
Web services files	*Agent services:* /var/lib/pki/pki-tomcat/ca/webapps/ca/agent/	
	Admin services: /var/lib/pki/pki-tomcat/ca/webapps/ca/admin/	
	End user services: /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/	

[a] Aliased to /etc/pki/pki-tomcat/ca/

[b] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

[c] Note that all subsystem certificates are stored in the instance security database

[d] Aliased to /var/log/pki/pki-tomcat/ca/

9.1.3. KRA Subsystem Information

This section contains details about the KRA subsystem, which is one of the possible subsystems that can be installed as a web application in a Certificate Server instance.

Table 9.3. KRA Subsystem Information for the Default Instance (pki-tomcat)

Setting	Value	
Main directory	/var/lib/pki/pki-tomcat/kra/	
Configuration directory	/var/lib/pki/pki-tomcat/kra/conf/[a]	
Configuration file	/var/lib/pki/pki-tomcat/kra/conf/CS.cfg	
Subsystem certificates	Transport certificate	
	Storage certificate	
	TLS server certificate	
	Audit log signing certificate	
	Subsystem certificate[b]	
Security databases	/var/lib/pki/pki-tomcat/alias/[c]	
Log files	/var/lib/pki/pki-tomcat/kra/logs/	
Install log	/var/log/pki/pki-kra-spawn-*date*.log	
Uninstall log	/var/log/pki/pki-kra-destroy-*date*.log	
Audit logs	/var/log/pki/pki-tomcat/kra/signedAudit/	
Web services files	*Agent services:* /var/lib/pki/pki-tomcat/kra/webapps/kra/agent/	
	Admin services: /var/lib/pki/pki-tomcat/kra/webapps/kra/admin/	

[a] Linked to /etc/pki/pki-tomcat/kra/

[b] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

[c] Note that all subsystem certificates are stored in the instance security database

9.1.4. OCSP Subsystem Information

This section contains details about the OCSP subsystem, which is one of the possible subsystems that can be installed as a web application in a Certificate Server instance.

Table 9.4. OCSP Subsystem Information for the Default Instance (pki-tomcat)

Setting	Value	
Main directory	/var/lib/pki/pki-tomcat/ocsp/	
Configuration directory	/var/lib/pki/pki-tomcat/ocsp/conf/[a]	
Configuration file	/var/lib/pki/pki-tomcat/ocsp/conf/CS.cfg	
Subsystem certificates	Transport certificate	
	Storage certificate	
	TLS server certificate	
	Audit log signing certificate	
	Subsystem certificate[b]	
Security databases	/var/lib/pki/pki-tomcat/alias/[c]	
Log files	/var/lib/pki/pki-tomcat/ocsp/logs/	
Install log	/var/log/pki/pki-ocsp-spawn-*date*.log	
Uninstall log	/var/log/pki/pki-ocsp-destroy-*date*.log	
Audit logs	/var/log/pki/pki-tomcat/ocsp/signedAudit/	
Web services files	*Agent services:* /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/agent/	
	Admin services: /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/admin/	

[a] Linked to /etc/pki/pki-tomcat/ocsp/

[b] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

[c] Note that all subsystem certificates are stored in the instance security database

9.1.5. TKS Subsystem Information

This section contains details about the TKS subsystem, which is one of the possible subsystems that can be installed as a web application in a Certificate Server instance.

Table 9.5. TKS Subsystem Information for the Default Instance (pki-tomcat)

Setting	Value	
Main directory	/var/lib/pki/pki-tomcat/tks/	
Configuration directory	/var/lib/pki/pki-tomcat/tks/conf/[a]	
Configuration file	/var/lib/pki/pki-tomcat/tks/conf/CS.cfg	
Subsystem certificates	Transport certificate	
	Storage certificate	
	TLS server certificate	
	Audit log signing certificate	
	Subsystem certificate[b]	
Security databases	/var/lib/pki/pki-tomcat/alias/[c]	
Log files	/var/lib/pki/pki-tomcat/tks/logs/	
Install log	/var/log/pki/pki-tks-spawn-*date*.log	
Uninstall log	/var/log/pki/pki-tks-destroy-*date*.log	
Audit logs	/var/log/pki/pki-tomcat/tks/signedAudit/	
Web services files	*Agent services:* /var/lib/pki/pki-tomcat/tks/webapps/tks/agent/	
	Admin services: /var/lib/pki/pki-tomcat/tks/webapps/tks/admin/	

[a] Linked to /etc/pki/pki-tomcat/tks/

[b] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

[c] Note that all subsystem certificates are stored in the instance security database

9.1.6. TPS Subsystem Information

This section contains details about the TPS subsystem, which is one of the possible subsystems that can be installed as a web application in a Certificate Server instance.

Table 9.6. TPS Subsystem Information for the Default Instance (pki-tomcat)

Setting	Value
Main directory	/var/lib/pki/pki-tomcat/tps/
Configuration directory	/var/lib/pki/pki-tomcat/tps/conf/[a]
Configuration file	/var/lib/pki/pki-tomcat/tps/conf/CS.cfg
Subsystem certificates	Transport certificate
	Storage certificate
	TLS server certificate
	Audit log signing certificate
	Subsystem certificate[b]
Security databases	/var/lib/pki/pki-tomcat/alias/[c]
Log files	/var/lib/pki/pki-tomcat/tps/logs/
Install log	/var/log/pki/pki-tps-spawn-*date*.log
Uninstall log	/var/log/pki/pki-tps-destroy-*date*.log
Audit logs	/var/log/pki/pki-tomcat/tps/signedAudit/
Web services files	*Agent services:* /var/lib/pki/pki-tomcat/tps/webapps/tps/agent/
	Admin services: /var/lib/pki/pki-tomcat/tps/webapps/tps/admin/

[a] Linked to /etc/pki/pki-tomcat/tps/

[b] The subsystem certificate is always issued by the security domain so that domain-level operations that require client authentication are based on this subsystem certificate.

[c] Note that all subsystem certificates are stored in the instance security database

9.1.7. Shared Certificate System Subsystem File Locations

There are some directories used by or common to all Certificate System subsystem instances for general server operations, listed in Table 2.8, "Subsystem File Locations" .

Table 9.7. Subsystem File Locations

Directory Location	Contents
/var/lib/*instance_name*	Contains the main instance directory, which is the location for user-specific directory locations and customized configuration files, profiles, certificate databases, web files, and other files for the subsystem instance.
/usr/share/java/pki	Contains Java archive files shared by the Certificate System subsystems. Along with shared files for all subsystems, there are subsystem-specific files in subfolders: <table><tr><td>pki/ca/ (CA)</td></tr><tr><td>pki/kra/ (KRA)</td></tr><tr><td>pki/ocsp/ (OCSP)</td></tr><tr><td>pki/tks/ (TKS)</td></tr></table> *Not used by the TPS subsystem.*
/usr/share/pki	Contains common files and templates used to create Certificate System instances. Along with shared files for all subsystems, there are subsystem-specific files in subfolders: <table><tr><td>pki/ca/ (CA)</td></tr><tr><td>pki/kra/ (KRA)</td></tr><tr><td>pki/ocsp/ (OCSP)</td></tr><tr><td>pki/tks/ (TKS)</td></tr><tr><td>pki/tps (TPS)</td></tr></table>
/usr/bin	Contains the **pkispawn** and **pkidestroy** instance configuration scripts and tools (Java, native, and security) shared by the Certificate System subsystems.
/var/lib/tomcat5/common/lib	Contains links to Java archive files shared by local Tomcat web applications and shared by the Certificate System subsystems. *Not used by the TPS subsystem.*
/var/lib/tomcat5/server/lib	Contains links to Java archive files used by the local Tomcat web server and shared by the Certificate System subsystems. *Not used by the TPS subsystem.*

Directory Location	Contents
/usr/shared/pki	Contains the Java archive files used by the Tomcat server and applications used by the Certificate System instances. *Not used by the TPS subsystem.*
/usr/lib/httpd/modules /usr/lib64/httpd/modules	Contains Apache modules used by the TPS subsystem. *Not used by the CA, KRA, OCSP, or TKS subsystems.*
/usr/lib/mozldap /usr/lib64/mozldap	Mozilla LDAP SDK tools used by the TPS subsystem. *Not used by the CA, KRA, OCSP, or TKS subsystems.*

9.2. CS.CFG FILES

The runtime properties of a Certificate System subsystem are governed by a set of configuration parameters. These parameters are stored in a file that is read by the server during startup, **CS.cfg**.

The **CS.cfg**, an ASCII file, is created and populated with the appropriate configuration parameters when a subsystem is first installed. The way the instance functions are modified is by making changes through the subsystem console, which is the recommended method. The changes made in the administrative console are reflected in the configuration file.

It is also possible to edit the **CS.cfg** configuration file directly, and in some cases this is the easiest way to manage the subsystem.

9.2.1. Locating the CS.cfg File

Each instance of a Certificate System subsystem has its own configuration file, **CS.cfg**. The contents of the file for each subsystem instance is different depending on the way the subsystem was configured, additional settings and configuration (like adding new profiles or enabling self-tests), and the type of subsystem.

The **CS.cfg** file is located in the configuration directory for the instance.

> /var/lib/pki/*instance_name*/conf

For example:

> /var/lib/pki/pki-tomcat/ca/conf

9.2.2. Editing the Configuration File

> ⚠️ **WARNING**
>
> Do not edit the configuration file directly without being familiar with the configuration parameters or without being sure that the changes are acceptable to the server. The Certificate System fails to start if the configuration file is modified incorrectly. Incorrect configuration can also result in data loss.

To modify the **CS.cfg** file:

1. Stop the subsystem instance.

   > systemctl stop pki-tomcatd@*instance_name*.service

   OR

   > systemctl stop pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

   The configuration file is stored in the cache when the instance is started. Any changes made to the instance through the Console are changed in the cached version of the file. When the server is stopped or restarted, the configuration file stored in the cache is written to disk. Stop the server before editing the configuration file or the changes will be overwritten by the cached version when the server is stopped.

2. Open the **/var/lib/pki/*instance_name*/conf** directory.

3. Open the **CS.cfg** file in a text editor.

4. Edit the parameters in the file, and save the changes.

5. Start the subsystem instance.

   > systemctl start pki-tomcatd@*instance_name*.service

   OR

   > systemctl start pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

9.2.3. Overview of the CS.cfg Configuration File

Each subsystem instances has its own main configuration file, **CS.cfg**, which contains all of the settings for the instance, such as plug-ins and Java classes for configuration. The parameters and specific settings are different depending on the type of subsystem, but, in a general sense, the **CS.cfg** file defines these parts of the subsystem instance:

- Basic subsystem instance information, like its name, port assignments, instance directory, and hostname

- Logging

- Plug-ins and methods to authenticate to the instance's user directory (authorization)

- The security domain to which the instance belongs

- Subsystem certificates

- Other subsystems used by the subsystem instance

- Database types and instances used by the subsystem

- Settings for PKI-related tasks, like the key profiles in the TKS, the certificate profiles in the CA, and the required agents for key recovery in the KRA

Many of the configuration parameters (aside from the ones for PKI tasks) are very much the same between the CA, OCSP, KRA, and TKS because they all use a Java-based console, so configuration settings which can be managed in the console have similar parameters.

The **CS.cfg** file a basic *parameter=value* format.

```
#comment
parameter=value
```

In the **CS.cfg** file, many of the parameter blocks have descriptive comments, commented out with a pound (#) character. Comments, blank lines, unknown parameters, or misspelled parameters are ignored by the server.

> **NOTE**
>
> A bug in the TPS prevents it from ignoring lines which are commented out in the **CS.cfg** file. Rather than commenting out lines in the TPS **CS.cfg** file, simply delete those lines.

Parameters that configure the same area of the instance tend to be grouped together into the same block.

Example 9.1. Logging Settings in the CS.cfg File

```
log.instance.System._000=##
log.instance.System._001=## System Logging
log.instance.System._002=##
log.instance.System.bufferSize=512
log.instance.System.enable=true
log.instance.System.expirationTime=0
log.instance.System.fileName=/var/lib/pki/pki-tomcat/ca/logs/system
log.instance.System.flushInterval=5
log.instance.System.level=3
log.instance.System.maxFileSize=2000
log.instance.System.pluginName=file
log.instance.System.rolloverInterval=2592000
log.instance.System.type=system
```

Some areas of functionality are implemented through plug-ins, such as self-tests, jobs, and authorization to access the subsystem. For those parameters, the plug-in instance has a unique identifier (since there can be multiple instances of even the same plug-in called for a subsystem), the

implementation plug-in name, and the Java class.

Example 9.2. Subsystem Authorization Settings

```
authz.impl._000=##
authz.impl._001=## authorization manager implementations
authz.impl._002=##
authz.impl.BasicAclAuthz.class=com.netscape.cms.authorization.BasicAclAuthz
authz.instance.BasicAclAuthz.pluginName=BasicAclAuthz
```

> **NOTE**
>
> The values for configuration parameters must be properly formatted, so they must obey two rules:
>
> - The values that need to be localized must be in UTF8 characters.
>
> - The **CS.cfg** file supports forward slashes (/) in parameter values. If a back slash (\) is required in a value, it must be escaped with a back slash, meaning that two back slashes in a row must be used.

The following sections are snapshots of **CS.cfg** file settings and parameters. These are not exhaustive references or examples of **CS.cfg** file parameters. Also, the parameters available and used in each subsystem configuration file is very different, although there are similarities.

9.2.3.1. Basic Instance Parameters for the CA: pkispawn file ca.cfg

Basic settings are specific to the instance itself, without directly relating to the functionality or behavior of the subsystem. This includes settings for the instance name, root directory, the user ID for the process, and port numbers. Many of the settings assigned when the instance is first installed or configured are prefaced with **pkispawn**.

Example 9.3. Basic Instance Parameters for the CA

```
[DEFAULT]
pki_admin_password=Secret.123
pki_client_pkcs12_password=Secret.123
pki_ds_password=Secret.123
# Optionally keep client databases
pki_client_database_purge=False
# Separated CA instance name and ports
pki_instance_name=pki-ca
pki_http_port=18080
pki_https_port=18443
# This Separated CA instance will be its own security domain
pki_security_domain_https_port=18443

[Tomcat]
# Separated CA Tomcat ports
pki_ajp_port=18009
pki_tomcat_server_port=18005
```
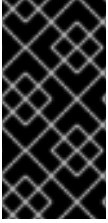
> **IMPORTANT**
>
> While information like the port settings is *included* in the **CS.cfg** file, it is not *set* in the **CS.cfg**. The server configuration is set in the **server.xml** file.
>
> The ports in **CS.cfg** and **server.xml** *must* match for a working RHCS instance.

9.2.3.2. Logging Settings

There are several different types of logs that can be configured, depending on the type of subsystem. Each type of log has its own configuration entry in the **CS.cfg** file.

For example, the CA has this entry for transaction logs, which allows log rotation, buffered logging, and log levels, among other settings:

```
log.instance.Transactions._000=##
log.instance.Transactions._001=## Transaction Logging
log.instance.Transactions._002=##
log.instance.Transactions.bufferSize=512
log.instance.Transactions.enable=true
log.instance.Transactions.expirationTime=0
log.instance.Transactions.fileName=/var/log/pki-ca/logs/transactions
log.instance.Transactions.flushInterval=5
log.instance.Transactions.level=1
log.instance.Transactions.maxFileSize=2000
log.instance.Transactions.pluginName=file
log.instance.Transactions.rolloverInterval=2592000
log.instance.Transactions.type=transaction
```

For more information about these parameters and their values, see Section 13.1, "Certificate System Log Settings". As long as audit logging is enabled, these values do not affect compliance.

9.2.3.3. Authentication and Authorization Settings

The **CS.cfg** file sets how users are identified to access a subsystem instance (authentication) and what actions are approved (authorization) for each authenticated user.

A CS subsystem uses authentication plug-ins to define the method for logging into the subsystem.

The following example shows an authentication instance named **SharedToken** that instantiates a JAVA plug-in named **SharedSecret**.

```
auths.impl.SharedToken.class=com.netscape.cms.authentication.SharedSecret
auths.instance.SharedToken.pluginName=SharedToken
auths.instance.SharedToken.dnpattern=
auths.instance.SharedToken.ldap.basedn=ou=People,dc=example,dc=org
auths.instance.SharedToken.ldap.ldapauth.authtype=BasicAuth
auths.instance.SharedToken.ldap.ldapauth.bindDN=cn=Directory Manager
auths.instance.SharedToken.ldap.ldapauth.bindPWPrompt=Rule SharedToken
auths.instance.SharedToken.ldap.ldapauth.clientCertNickname=
auths.instance.SharedToken.ldap.ldapconn.host=server.example.com
auths.instance.SharedToken.ldap.ldapconn.port=636
auths.instance.SharedToken.ldap.ldapconn.secureConn=true
auths.instance.SharedToken.ldap.ldapconn.version=3
auths.instance.SharedToken.ldap.maxConns=
```

```
auths.instance.SharedToken.ldap.minConns=
auths.instance.SharedToken.ldapByteAttributes=
auths.instance.SharedToken.ldapStringAttributes=
auths.instance.SharedToken.shrTokAttr=shrTok
```

For some authorization settings, it is possible to select an authorization method that uses an LDAP database to store user entries, in which case the database settings are configured along with the plug-in as shown below.

```
authz.impl.DirAclAuthz.class=com.netscape.cms.authorization.DirAclAuthz
authz.instance.DirAclAuthz.ldap=internaldb
authz.instance.DirAclAuthz.pluginName=DirAclAuthz
authz.instance.DirAclAuthz.ldap._000=##
authz.instance.DirAclAuthz.ldap._001=## Internal Database
authz.instance.DirAclAuthz.ldap._002=##
authz.instance.DirAclAuthz.ldap.basedn=dc=server.example.com-pki-ca
authz.instance.DirAclAuthz.ldap.database=server.example.com-pki-ca
authz.instance.DirAclAuthz.ldap.maxConns=15
authz.instance.DirAclAuthz.ldap.minConns=3
authz.instance.DirAclAuthz.ldap.ldapauth.authtype=SslClientAuth
authz.instance.DirAclAuthz.ldap.ldapauth.bindDN=cn=Directory Manager
authz.instance.DirAclAuthz.ldap.ldapauth.bindPWPrompt=Internal LDAP Database
authz.instance.DirAclAuthz.ldap.ldapauth.clientCertNickname=
authz.instance.DirAclAuthz.ldap.ldapconn.host=localhost
authz.instance.DirAclAuthz.ldap.ldapconn.port=11636
authz.instance.DirAclAuthz.ldap.ldapconn.secureConn=true
authz.instance.DirAclAuthz.ldap.multipleSuffix.enable=false
```

For more information on securely configuring LDAP and an explanation of parameters, refer to Section 6.5.5, "Enabling TLS Client Authentication". The parameters paths differ than what is shown there, but the same names and values are allowed in both places.

The CA also has to have a mechanism for approving user requests. As with configuring authorization, this is done by identifying the appropriate authentication plug-in and configuring an instance for it:

```
auths.impl.AgentCertAuth.class=com.netscape.cms.authentication.AgentCertAuthentication
auths.instance.AgentCertAuth.agentGroup=Certificate Manager Agents
auths.instance.AgentCertAuth.pluginName=AgentCertAuth
```

9.2.3.4. Subsystem Certificate Settings

Several of the subsystems have entries for each subsystem certificate in the configuration file.

```
ca.sslserver.cert=MIIDmDCCAoCgAwIBAgIBAzANBgkqhkiG9w0BAQUFADBAMR4wHAYDVQQKExVS
ZWR...
ca.sslserver.certreq=MIICizCCAXMCAQAwRjEeMBwGA1UEChMVUmVkYnVkY29tcHV0ZXIgRG9tYWl
uMSQwIgYDV...
ca.sslserver.nickname=Server-Cert cert-pki-ca
ca.sslserver.tokenname=Internal Key Storage Token
```

9.2.3.5. Settings for Required Subsystems

At a minimum, each subsystem depends on a CA, which means that the CA (and any other required subsystem) has to be configured in the subsystem's settings. Any connection to another subsystem is prefaced by **conn.** and then the subsystem type and number.

```
conn.ca1.clientNickname=subsystemCert cert-pki-tps
conn.ca1.hostadminport=server.example.com:9445
conn.ca1.hostagentport=server.example.com:9444
conn.ca1.hostport=server.example.com:9443
conn.ca1.keepAlive=true
conn.ca1.retryConnect=3
conn.ca1.servlet.enrollment=/ca/ee/ca/profileSubmitSSLClient
conn.ca1.servlet.renewal=/ca/ee/ca/profileSubmitSSLClient
conn.ca1.servlet.revoke=/ca/subsystem/ca/doRevoke
conn.ca1.servlet.unrevoke=/ca/subsystem/ca/doUnrevoke
conn.ca1.timeout=100
```

9.2.3.6. Database Settings

All of the subsystems use an LDAP directory to store their information. This internal database is configured in the **internaldb** parameters, except for the TPS which configured it in the **tokendb** parameters with a lot of other configuration settings.

```
internaldb._000=##
internaldb._000=##
internaldb._001=## Internal Database
internaldb._002=##
internaldb.basedn=o=pki-tomcat-ca-SD
internaldb.database=pki-tomcat-ca
internaldb.maxConns=15
internaldb.minConns=3
internaldb.ldapauth.authtype=SslClientAuth
internaldb.ldapauth.clientCertNickname=HSM-A:subsystemCert pki-tomcat-ca
internaldb.ldapconn.host=example.com
internaldb.ldapconn.port=11636
internaldb.ldapconn.secureConn=true
internaldb.multipleSuffix.enable=false
```

For further information on securely configuring LDAP and an explanation of parameters, refer to Section 6.5.5, "Enabling TLS Client Authentication". No additional configuration is necessary outside of what is done as part of Section 6.5.5, "Enabling TLS Client Authentication".

9.2.3.7. Enabling and Configuring a Publishing Queue

Enabling the publishing queue by editing the **CS.cfg** file allows administrators to set other options for publishing, like the number of threads to use for publishing operations and the queue page size.

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

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

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

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

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

> ca.publish.queue.enable=true

4. Set other related publishing queue parameters:

   - ***ca.publish.queue.maxNumberOfThreads*** sets the maximum number of threads that can be opened for publishing operations. The default is 3.

   - ***ca.publish.queue.priorityLevel*** sets the priority for publishing operations. The priority value ranges from **-2** (lowest priority) to **2** (highest priority). Zero (0) is normal priority and is also the default.

   - ***ca.publish.queue.pageSize*** sets the maximum number of requests that can be stored in the publishing queue page. The default is 40.

   - ***ca.publish.queue.saveStatus*** sets the interval to save its status every specified number of publishing operations. This allows the publishing queue to be recovered if the CA is restarted or crashes. The default is 200, but any non-zero number will recover the queue when the CA restarts. Setting this parameter to 0 disables queue recovery.

> ca.publish.queue.maxNumberOfThreads=1
> ca.publish.queue.priorityLevel=0
> ca.publish.queue.pageSize=100
> ca.publish.queue.saveStatus=200

> **NOTE**
>
> Setting ***ca.publish.queue.enable*** to false and ***ca.publish.queue.maxNumberOfThreads*** to 0 disables both the publishing queue and using separate threads for publishing issued certificates.

5. Restart the CA server.

> systemctl start pki-tomcatd-nuxwdog@*instance_name*.service

9.2.3.8. Settings for PKI Tasks

The **CS.cfg** file is used to configure the PKI tasks for every subsystem. The parameters are different for every single subsystem, without any overlap.

For example, the KRA has settings for a required number of agents to recover a key.

> kra.noOfRequiredRecoveryAgents=1

Review the **CS.cfg** file for each subsystem to become familiar with its PKI task settings; the comments in the file are a decent guide for learning what the different parameters are.

- The CA configuration file lists all of the certificate profiles and policy settings, as well as rules for generating CRLs.

- The TPS configures different token operations.

- The TKS lists profiles for deriving keys from different key types.

- The OCSP sets key information for different key sets.

9.2.3.9. Changing DN Attributes in CA-Issued Certificates

In certificates issued by the Certificate System, DNs identify the entity that owns the certificate. In all cases, if the Certificate System is connected with a Directory Server, the format of the DNs in the certificates should match the format of the DNs in the directory. It is not necessary that the names match exactly; certificate mapping allows the subject DN in a certificate to be different from the one in the directory.

In the Certificate System, the DN is based on the components, or attributes, defined in the X.509 standard. Table 9.8, "Allowed Characters for Value Types" lists the attributes supported by default. The set of attributes is extensible.

Table 9.8. Allowed Characters for Value Types

Attribute	Value Type	Object Identifier
cn	DirectoryString	2.5.4.3
ou	DirectoryString	2.5.4.11
o	DirectoryString	2.5.4.10
c	PrintableString , two-character	2.5.4.6
l	DirectoryString	2.5.4.7
st	DirectoryString	2.5.4.8
street	DirectoryString	2.5.4.9
title	DirectoryString	2.5.4.12
uid	DirectoryString	0.9.2342.19200300.100.1.1
mail	IA5String	1.2.840.113549.1.9.1
dc	IA5String	0.9.2342.19200300.100.1.2.25
serialnumber	PrintableString	2.5.4.5
unstructuredname	IA5String	1.2.840.113549.1.9.2
unstructuredaddress	PrintableString	1.2.840.113549.1.9.8

By default, the Certificate System supports the attributes identified in Table 9.8, "Allowed Characters

By default, the Certificate System supports the attributes identified in Table 9.8, "Allowed Characters for Value Types". This list of supported attributes can be extended by creating or adding new attributes. The syntax for adding additional **X.500Name** attributes, or components, is as follows:

```
X500Name.NEW_ATTRNAME.oid=n.n.n.n
X500Name.NEW_ATTRNAME.class=string_to_DER_value_converter_class
```

The value converter class converts a string to an ASN.1 value; this class must implement the **netscape.security.x509.AVAValueConverter** interface. The string-to-value converter class can be one of the following:

- **netscape.security.x509.PrintableConverter** converts a string to a **PrintableString** value. The string must have only printable characters.

- **netscape.security.x509.IA5StringConverter** converts a string to an **IA5String** value. The string must have only IA5String characters.

- **netscape.security.x509.DirStrConverter** converts a string to a **DirectoryString**. The string is expected to be in **DirectoryString** format according to RFC 2253.

- **netscape.security.x509.GenericValueConverter** converts a string character by character in the following order, from the smallest characterset to the largest:

  - Printable

  - IA5String

  - BMPString

  - Universal String

An attribute entry looks like the following:

```
X500Name.MY_ATTR.oid=1.2.3.4.5.6
X500Name.MY_ATTR.class=netscape.security.x509.DirStrConverter
```

9.2.3.9.1. Adding New or Custom Attributes

To add a new or proprietary attribute to the Certificate System schema, do the following:

1. Stop the Certificate Manager.

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

2. Open the **/var/lib/pki/cs_instance/conf/** directory.

3. Open the configuration file, **CS.cfg**.

4. Add the new attributes to the configuration file.

   For example, to add three proprietary attributes, **MYATTR1** that is a **DirectoryString**, **MYATTR2** that is an **IA5String**, and **MYATTR3** that is a **PrintableString**, add the following lines at the end of the configuration file:

   ```
   X500Name.attr.MYATTR1.oid=1.2.3.4.5.6
   ```

```
X500Name.attr.MYATTR1.class=netscape.security.x509.DirStrConverter
X500Name.attr.MYATTR2.oid=11.22.33.44.55.66
X500Name.attr.MYATTR2.class=netscape.security.x509.IA5StringConverter
X500Name.attr.MYATTR3.oid=111.222.333.444.555.666
X500Name.attr.MYATTR3.class=netscape.security.x509.PrintableConverter
```

5. Save the changes, and close the file.

6. Restart the Certificate Manager.

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

7. Reload the enrollment page and verify the changes; the new attributes should show up in the form.

8. To verify that the new attributes are in effect, request a certificate using the manual enrollment form.

   Enter values for the new attributes so that it can be verified that they appear in the certificate subject names. For example, enter the following values for the new attributes and look for them in the subject name:

   ```
   MYATTR1: a_value
   MYATTR2: a.Value
   MYATTR3: aValue
   cn: John Doe
   o: Example Corporation
   ```

9. Open the agent services page, and approve the request.

10. When the certificate is issued, check the subject name. The certificate should show the new attribute values in the subject name.

9.2.3.9.2. Changing the DER-Encoding Order

It is possible to change the DER-encoding order of a **DirectoryString**, so that the string is configurable since different clients support different encodings.

The syntax for changing the DER-encoding order of a **DirectoryString** is as follows:

```
X500Name.directoryStringEncodingOrder=encoding_list_separated_by_commas
```

The possible encoding values are as follows:

- **Printable**

- **IA5String**

- **UniversalString**

- **BMPString**

- **UTF8String**

For example, the DER-encoding ordered can be listed as follows:

> X500Name.directoryStringEncodingOrder=Printable,BMPString

To change the **DirectoryString** encoding, do the following:

1. Stop the Certificate Manager.

   > systemctl stop pki-tomcatd-nuxwdog@*instance_name*.service

2. Open the **/var/lib/pki/*cs_instance*/conf/** directory.

3. Open the **CS.cfg** configuration file.

4. Add the encoding order to the configuration file.

   For example, to specify two encoding values, **PrintableString** and **UniversalString**, and the encoding order is **PrintableString** first and **UniversalString** next, add the following line at the end of the configuration file:

   > X500Name.directoryStringEncodingOrder=PrintableString, UniversalString

5. Save the changes, and close the file.

6. Start the Certificate Manager.

   > systemctl start pki-tomcatd-nuxwdog@*instance_name*.service

7. To verify that the encoding orders are in effect, enroll for a certificate using the manual enrollment form. Use **John_Doe** for the **cn**.

8. Open the agent services page, and approve the request.

9. When the certificate is issued, use the **dumpasn1** tool to examine the encoding of the certificate.

   The **cn** component of the subject name should be encoded as a **UniversalString**.

10. Create and submit a new request using **John Smith** for the **cn**.

    The **cn** component of the subject name should be encoded as a **PrintableString**.

9.2.3.10. Setting a CA to Use a Different Certificate to Sign CRLs

A Certificate Manager uses the key pair corresponding to its OCSP signing certificate for signing certificates and certificate revocation lists (CRLs). To use a different key pair to sign the CRLs that the Certificate Manager generates, then a CRL signing certificate can be created. The Certificate Manager's CRL signing certificate must be signed or issued by itself.

To enable a Certificate Manager to sign CRLs with a different key pair, do the following:

1. Request and install a CRL signing certificate for the Certificate Manager using CMC. For details about requesting a system certificate, see 5.3.2.1. Obtaining System and Server Certificates in Red Hat Certificate System's Administration Guide.

   Note that the profile used to obtain the certificate must use the **keyUsageExtDefaultImpl** class id and the corresponding *keyUsageCrlSign* parameter set to **true**:

```
policyset.userCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.userCertSet.6.default.params.keyUsageCrlSign=true
```

2. After the CRL signing certificate is generated, install the certificate in the Certificate Manager's database through **System Keys and Certificates** in the console.

3. Stop the Certificate Manager.

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

4. Update the Certificate Manager's configuration to recognize the new key pair and certificate.

   1. Change to the Certificate Manager instance configuration directory.

      ```
      ]# cd /var/lib/pki/instance_name/ca/conf/
      ```

   2. Open the **CS.cfg** file and add the following lines:

      ```
      ca.crl_signing.cacertnickname=nickname cert-instance_ID
      ca.crl_signing.defaultSigningAlgorithm=signing_algorithm
      ca.crl_signing.tokenname=token_name
      ```

      nickname is the name assigned to the CRL signing certificate.

      instance_ID is the name of the Certificate Manager instance.

      If the installed CA is a RSA-based CA, *signing_algorithm* can be **SHA256withRSA**, **SHA384withRSA**, or **SHA512withRSA**. If the installed CA is an EC-based CA, *signing_algorithm* can be **SHA256withEC**, **SHA384withEC**, **SHA512withEC**.

      token_name is the name of the token used for generating the key pair and the certificate. If the internal/software token is used, use **Internal Key Storage Token** as the value.

      For example, the entries might look like this:

      ```
      ca.crl_signing.cacertnickname=crlSigningCert cert-pki-ca
      ca.crl_signing.defaultSigningAlgorithm=SHAMD512withRSA
      ca.crl_signing.tokenname=Internal Key Storage Token
      ```

   3. Save the changes, and close the file.

5. Restart the Certificate Manager.

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

   Now the Certificate Manager is ready to use the CRL signing certificate to sign the CRLs it generates.

9.2.3.11. Configuring CRL Generation from Cache in CS.cfg

The CRL cache is a simple mechanism that allows cert revocation information to be taken from a collection of revocation information maintained in memory. For best performance, it is recommended that this feature be enabled, which already represents the default behavior. The following configuration

information (which is the default) is presented for information purposes or if changes are desired.

1. Stop the CA server.

   > systemctl stop pki-tomcatd-nuxwdog@*instance_name*.service

2. Open the CA configuration directory.

   > # cd /var/lib/*instance_name*/conf/

3. Edit the **CS.cfg** file, setting the ***enableCRLCache*** and ***enableCacheRecovery*** parameters to true:

   > ca.crl.MasterCRL.enableCRLCache=true
   > ca.crl.MasterCRL.enableCacheRecovery=true

4. Start the CA server.

   > systemctl start pki-tomcatd-nuxwdog@*instance_name*.service

9.2.3.12. Configuring Update Intervals for CRLs in CS.cfg

The following describes how to configure the CRL system flexibly to reflect desired behavior. The goal is to configure CRL updates according to some schedule of two types. One type allows for a list of explicit times and the other consists of a length of time interval between updates. There is also a hybrid scenario where both are enabled to account for drift. The Note entry just below actually represents the default out of the box scenario.

The default scenario is listed as follows:

> ca.crl.MasterCRL.updateSchema=3
> ca.crl.MasterCRL.enableDailyUpdates=true
> ca.crl.MasterCRL.enableUpdateInterval=true
> ca.crl.MasterCRL.autoUpdateInterval=240
> ca.crl.MasterCRL.dailyUpdates=1:00
> ca.crl.MasterCRL.nextUpdateGracePeriod=0

Deviate from this only when a more detailed and specific update schedule is desired. The rest of the section will talk about how that is accomplished.

Configuring the settings for full and delta CRLs in the **CS.cfg** file involves editing parameters.

Table 9.9. CRL Extended Interval Parameters

Parameter	Description	Accepted Values
updateSchema	Sets the ratio for how many delta CRLs are generated per full CRL.	An integer value
enableDailyUpdates	Enables and disables setting CRL updates based on set times.	true or false

Parameter	Description	Accepted Values
enableUpdateInterval	Enables and disables setting CRL updates based on set intervals.	true or false
dailyUpdates	Sets the times the CRLs should be updated	A comma-delimited list of times
autoUpdateInterval	Sets the interval in minutes to update the CRLs.	An integer value
nextUpdateGracePeriod	Adds the time in minutes to the CRL validity period to ensure that CRLs remain valid throughout the publishing or replication period.	An integer value
refreshInSec	Sets the periodicity in seconds of the thread on the clone OCSP to check LDAP for any updates of the CRL.	An integer value

Procedure 9.1. How to configure CRL update intervals in CS.cfg

1. Stop the CA server.

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

2. Change to the CA configuration directory.

   ```
   # cd /var/lib/instance_name/conf/
   ```

3. Edit the **CS.cfg** file, and add the following line to set the update interval:

   ```
   ca.crl.MasterCRL.updateSchema=3
   ```

   The default interval is 1, meaning a full CRL is generated every time a CRL is generated. The ***updateSchema*** interval can be set to any integer.

4. Set the update frequency, either by specifying a cyclical interval or set times for the updates to occur:

   ○ Specify set times by enabling the ***enableDailyUpdates*** parameter, and add the desired times to the ***dailyUpdates*** parameter:

   ```
   ca.crl.MasterCRL.enableDailyUpdates=true
   ca.crl.MasterCRL.enableUpdateInterval=false
   ca.crl.MasterCRL.dailyUpdates=0:50,04:55,06:55
   ```

   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, set
01:50,04:55,06:55;02:00,05:00,17:00 to configure revocation on Day 1 of the cycle at
1:50am, 4:55am, and 6:55am and then Day 2 at 2am, 5am, and 5pm.

Specify intervals by enabling the *enableUpdateInterval* parameter, and add the required
interval in minutes to the *autoUpdateInterval* parameter:

```
ca.crl.MasterCRL.enableDailyUpdates=false
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
```

5. Set the following parameters depending on your environment:

   ○ If you run a CA without an OCSP subsystem, set:

   ```
   ca.crl.MasterCRL.nextUpdateGracePeriod=0
   ```

   ○ If you run a CA with an OCSP subsystem, set:

   ```
   ca.crl.MasterCRL.nextUpdateGracePeriod=time_in_minutes
   ```

   The *ca.crl.MasterCRL.nextUpdateGracePeriod* parameter defines the time in minutes,
   and the value must be big enough to enable the CA to propagate the new CRL to the
   OCSP. You must set the parameter to a non-zero value.

   If you additionally have OCSP clones in your environment, also set:

   ```
   ocsp.store.defStore.refreshInSec=time_in_seconds
   ```

   The *ocsp.store.defStore.refreshInSec* parameter sets the frequency in seconds with
   which the clone OCSP instances are informed of CRL updates through LDAP replication
   updates from the master OCSP instance.

   See Table 9.9, "CRL Extended Interval Parameters" for details on the parameters.

6. Restart the CA server.

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

> **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, set both *enableDailyUpdates* and *enableUpdateInterval* parameters to true, and add the required values to *autoUpdateInterval* and *dailyUpdates*:
>
> ```
> ca.crl.MasterCRL.enableDailyUpdates=true
> ca.crl.MasterCRL.enableUpdateInterval=true
> ca.crl.MasterCRL.autoUpdateInterval=240
> ca.crl.MasterCRL.dailyUpdates=1:00
> ```
>
> Only one *dailyUpdates* value will be accepted when updating CRLs by interval.
>
> The interval updates will resynchronize with the *dailyUpdates* value every 24 hours preventing schedule drift.

9.2.3.13. Changing the Access Control Settings for the Subsystem

By default, access control rules are applied by evaluating deny rules first and then by evaluating allow rules. To change the order, change the *authz.evaluateOrder* parameter in the **CS.cfg**.

```
authz.evaluateOrder=deny,allow
```

Additionally, access control rules can be evaluated from the local **web.xml** file (basic ACLs) or more complex ACLs can be accessed by checking the LDAP database. The *authz.sourceType* parameter identifies what type of authorization to use.

```
authz.sourceType=web.xml
```

> **NOTE**
>
> Always restart the subsystem after editing the **CS.cfg** file to load the updated settings.

9.2.3.14. Setting Requirement for **pkiconsole** to use TLS Client Certificate Authentication

Edit the **CS.cfg** file of each subsystem, search for the *authType* parameter and set it as follows:

```
authType=sslclientauth
```

9.3. MANAGING SYSTEM PASSWORDS

As explained in Section 2.3.11, "Passwords and Watchdog (nuxwdog)", Certificate System uses passwords bind to servers or to unlock tokens when the server starts.

The **password.conf** file stores system passwords in plain text. However, some administrators prefer to remove the password file entirely to allow **nuxwdog** to prompt for manual entry of each password initially and store for auto-restart in case of an unplanned shutdown.

When a Certificate System instance starts, the subsystem automatically checks for the **password.conf**

file. If the file exists, then it uses those passwords to connect to other services, such as the internal LDAP database. If that file does not exist, then the watchdog daemon prompts for all of the passwords required by the PKI server to start.

> **NOTE**
>
> If the **password.conf** file is present, the subsystem assumes that all the required passwords are present and properly formatted in clear text. If any passwords are missing or wrongly formatted, then the system fails to start correctly.

The required passwords are listed in the *cms.passwordlist* parameter in the **CS.cfg** file:

```
cms.passwordlist=internaldb,replicationdb,CA LDAP Publishing
cms.password.ignore.publishing.failure=true
```

> **NOTE**
>
> The *cms.password.ignore.publishing.failure* parameter allows a CA subsystem to start up successfully even if it has a failed connection to one of its LDAP publishing directories.

For the CA, KRA, OCSP, and TKS subsystems, the default expected passwords are:

- **internal** for the NSS database
- **internaldb** for the internal LDAP database
- **replicationdb** for the replication password
- Any passwords to access external LDAP databases for publishing (CA only)
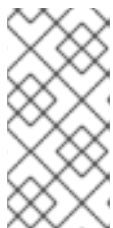
> **NOTE**
>
> If a publisher is configured after the **password.conf** file is removed, nothing is written to the **password.conf** file. Unless **nuxwdog** is configured, the server will not have access to the prompts for the new publishing password the next time that the instance restarts.

- Any external hardware token passwords

For the TPS, this prompts for three passwords:

- **internal** for the NSS database
- **tokendbpass** for the internal LDAP database
- Any external hardware token passwords

This section describes the two mechanisms provided for Certificate System to retrieve these passwords:

- **password.conf** file (the default)
- nuxwdog (watchdog)

9.3.1. Configuring the password.conf File

NOTE

This section is here for reference only. Correct and secure operation must involve using the **nuxwdog** watchdog. Please refer to Section 9.3.2, "Using the Certificate System Watchdog Service" to enable **nuxwdog**, as it is required for full compliance.

By default, passwords are stored in a plain text file, **password.conf**, in the subsystem **conf/** directory. Therefore, it is possible to modify them simply through a text editor.

The list of passwords stored in this file includes the following:

- The bind password used by the Certificate System instance to access and update the internal database.

- The password to the HSM

- The bind password used by the Certificate System instance to access the authentication directory, in case of CMC Shared Token.

- The bind password used by the subsystem to access and update the LDAP publishing directory; this is required only if the Certificate System instance is configured for publishing certificates and CRLs to an LDAP-compliant directory.

- the bind password used by the subsystem to access its replication database.

- For a TPS instance, the bind password used to access and update the token database.

The **password.conf** file also contains the token passwords needed to open the private keys of the subsystem.

The name and location password file to use for the subsystem is configured in the **CS.cfg** file:

```
passwordFile=/var/lib/pki/instance_name/conf/password.conf
```

The internal password store and replication database have randomly-generated PINs which were set when the subsystem was installed and configured; the internal LDAP database password was defined by the administrator when the instance was configured.

The password entries in the **password.conf** file are in the following format:

```
name=password
```

For example:

```
internal=413691159497
```

In cases where an HSM token is required, use the following format:

```
hardware-name=password
```

For example:

```
hardware-NHSM6000=MyHSM$S8cret
```

Example content of a **password.conf** file:

```
internal=376577078151
internaldb=secret12
replicationdb=1535106826
hardware-NHSM6000=MyHSM$S8cret
```

9.3.2. Using the Certificate System Watchdog Service

In Certificate System, the watchdog service is used to start services which require passwords to access the security database in order to start. In case there is a requirement not to store the unencrypted passwords on the system, the watchdog service:

- prompts for the relevant passwords during server startup and caches them.

- uses cached passwords in case of a failure when the server is automatically restarted due to a crash.

9.3.2.1. Enabling the Watchdog Service

To enable the watchdog service:

1. If you also want to use the Shared Secret feature on this host, enable the Shared Secret feature as described in in the *Enabling the CMC Shared Secret Feature* section in the *Certificate System Administration Guide (Common Criteria Edition)*.

2. Backup the **server.xml** and **password.conf** files from the **/var/lib/pki/*instance_name*/conf/** directory. For example:

   ```
   # cp -p /var/lib/pki/instance_name/conf/server.xml /root/
   # cp -p /var/lib/pki/instance_name/conf/password.conf /root/
   ```

3. Stop and disable the Certificate System instance's service:

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

4. If you use a Hardware Security Module (HSM), enable the watchdog service to prompt for the password of the hardware token:

   a. Display the name of the hardware token:

      ```
      # egrep "^hardware-" /var/lib/pki/instance_name/conf/password.conf
      hardware-HSM_token_name=password
      ```

      The highlighted string in the previous example is the hardware token name.

   b. Add the **cms.tokenList** parameter to the **/var/lib/pki/*instance_name*/conf/ca/CS.cfg** file and set it to the name of the hardware token. For example:

      ```
      cms.tokenList=HMS_token_name
      ```

5. Enable the watchdog configuration for the instance:

   ```
   # pki-server instance-nuxwdog-enable instance_name
   ```

   Alternatively, enable the watchdog for all instances:

   ```
   # pki-server nuxwdog-enable
   ```

   For further details, see the pki–server–nuxwdog(8) man page.

6. By default, **nuxwdog** starts the server as the user configured in the *TOMCAT_USER* variable in the **/etc/sysconfig/pki-tomcat** file. Optionally, to modify the user and group:

   a. Copy the watchdog **systemd** unit file of the instance to the **/etc/systemd/system/** directory:

      ```
      # cp -p /usr/lib/systemd/system/instance_name-nuxwdog@.service /etc/systemd/system/
      ```

      NOTE

      Unit files in the **/etc/systemd/system/** directory have a higher priority and are not replaced during updates.

   b. Add the following entries to the *[Service]* section in the **/etc/pki/**_instance_name_**/nuxwdog.conf** file:

      ```
      User new_user_name
      ```

   c. Reload the **systemd** configuration:

      ```
      # systemctl daemon-reload
      ```

7. Enable the Certificate System service that uses the watchdog:

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

8. Optionally: See Section 9.3.2.3, "Verifying That the Certificate System Watchdog Service is Enabled".

9. To start the Certificate System instance, run the following command and enter the prompted passwords:

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

9.3.2.2. Starting and Stopping Certificate System with the Watchdog Enabled

For information how to manage a Certificate System instance refer to Section 2.2.3, "Execution Management (systemctl)".

9.3.2.3. Verifying That the Certificate System Watchdog Service is Enabled

To verify that the watchdog service is enabled:

1. Verify that the **pki-tomcatd-nuxwdog** service is enabled:

   ```
   # systemctl is-enabled pki-tomcatd-nuxwdog@instance_name.service
   enabled
   ```

2. Verify that the **pki-tomcatd** service is disabled:

   ```
   # systemctl is-disabled pki-tomcatd@instance_name.service
   disabled
   ```

3. In the **/etc/pki/*instance_name*/server.xml** file:

   a. verify that the *passwordFile* parameter refers to the **CS.cfg** file. For example:

      ```
      passwordFile="/var/lib/pki/instance_name/ca/CS.cfg"
      ```

   b. verify that the *passwordClass* parameter is set to **com.netscape.cms.tomcat.NuxwdogPasswordStore**:

      ```
      passwordClass="com.netscape.cms.tomcat.NuxwdogPasswordStore"
      ```

9.3.2.4. Disabling the Watchdog Service

To disable the watchdog service:

1. Stop and disable the Certificate System instance's service that uses the watchdog:

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

2. Enable the regular service without watch dog for the instance:

   ```
   # pki-server instance-nuxwdog-disable instance_name
   ```

3. Disable the watchdog configuration for the instance:

   ```
   # systemctl enable pki-tomcatd@instance_name.service
   ```

   For further details, see the pki-server-nuxwdog(8) man page.

4. Restore the **password.conf** file to its original location. For example:

   ```
   # cp /root/password.conf.bak /var/lib/pki/instance_name/conf/password.conf
   ```

5. Start the Certificate System instance:

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

9.4. CONFIGURATION FILES FOR THE TOMCAT ENGINE AND WEB SERVICES

All of the user and administrative (administrators, agents, and auditors) services for the subsystems are accessed over web protocols.

This section discusses the two major sets of configuration files that apply to all Red Hat Certificate System subsystems (CA, KRA, OCSP, TKS, and TPS):

- **/var/lib/pki/*instance_name*/conf/server.xml** provides the configuration for the Tomcat engine.

- **/usr/share/pki/*subsystem_type*/webapps/WEB-INF/web.xml** provides the configuration for the web services offered by this instance.

9.4.1. Tomcatjss
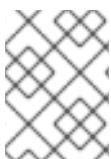
> **NOTE**
>
> The later subsections include important configuration information on required changes to parameter values. Ensure they are followed for strict compliance.

The following configuration in the **server.xml** file found in the example **pki-tomcat/conf** directory can be used to explain how **Tomcatjss** fits into the entire Certificate System ecosystem. Portions of the Connector entry for the secret port are shown below.

```
<Connector name="Secure"

# Info about the socket itself
port="8443"
protocol="org.apache.coyote.http11.Http11Protocol"
SSLEnabled="true"
sslProtocol="SSL"
scheme="https"
secure="true"
connectionTimeout="80000"
maxHttpHeaderSize="8192"
acceptCount="100" maxThreads="150" minSpareThreads="25"
enableLookups="false" disableUploadTimeout="true"
# Points to our tomcat jss implementation
sslImplementationName="org.apache.tomcat.util.net.jss.JSSImplementation"

# OCSP responder configuration can be enabled here
enableOCSP="true"
ocspCacheSize="1000"
ocspMinCacheEntryDuration="60"
ocspMaxCacheEntryDuration="120"
ocspTimeout="10"

# A collection of cipher related settings that make sure connections are secure.
strictCiphers="true"
# The "clientAuth" parameter configures the client authentication scheme
# for this server socket. If you set "clientAuth" to "want", the client
# authentication certificate is optional. Alternatively, set the
# parameter to "required" to configure that the certificate is is mandatory.
```

```
clientAuth="want"
sslVersionRangeStream="tls1_1:tls1_2"
sslVersionRangeDatagram="tls1_1:tls1_2"
sslRangeCiphers="+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_RSA_WITH_AES
_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA,+TLS_DHE_RSA_WITH_AES_256_
CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_DHE_RSA_WITH_AES_256_CB
C_SHA256,+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_ECDHE_RSA_WITH_AES_2
56_CBC_SHA384,+TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,+TLS_ECDHE_RSA_WITH_AE
S_128_GCM_SHA256,+TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,+TLS_ECDHE_RSA_WITH
_AES_256_GCM_SHA384
serverCertNickFile="/var/lib/pki/pki-tomcat/conf/serverCertNick.conf"
passwordFile="/var/lib/pki/pki-tomcat/conf/password.conf"
passwordClass="org.apache.tomcat.util.net.jss.PlainPasswordFile"
certdbDir="/var/lib/pki/pki-tomcat/alias"

/>
```

In the **server.xml** configuration file for the Tomcat engine, there is this Connector configuration element that contains the pointer to the tomcatjss implementation, which can be plugged into the **sslImplementation** property of this Connector object.

Each key parameter element is explained in the subsections below.

9.4.1.1. TLS Cipher Configuration

The TLS ciphers configured in the **server.xml** file provide system-wide defaults when Red Hat Certificate system is acting as a client and as a server. This includes when acting as a server (for example, when serving HTTPS connections from Tomcat) and as a client (for example, when communicating with the LDAP server or when communicating with another Certificate System instance).

The configuration for server TLS ciphers is in the Red Hat Certificate System instance-specific **/var/lib/pki/***instance_name***/conf/server.xml** file. The following parameters control the ciphers offered:

- *strictCiphers*, when set to **true**, ensures that only ciphers with a **+** sign in the *sslRangeCiphers* are enabled.

  ```
  strictCiphers="true"
  ```

  Do not change the default value (**true**).

- *sslVersionRangeStream* and *sslVersionRangeDatagram* sets the TLS version the server supports. The following are the defaults of the parameters:

  ```
  sslVersionRangeStream="tls1_1:tls1_2"
  sslVersionRangeDatagram="tls1_1:tls1_2"
  ```

  Do not change the default value of the parameters.

- *sslRangeCiphers* sets which ciphers are enabled and disabled. Ciphers with a **+** sign are enabled, ciphers with a **-** sign disabled.

  Set RSA ciphers as below:

  ```
  sslRangeCiphers="+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_RSA_WI
  TH_AES_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA,+TLS_DHE_RSA_W
  ```

> ITH_AES_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_DHE_RS
> A_WITH_AES_256_CBC_SHA256,+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,+TL
> S_ECDHE_RSA_WITH_AES_256_CBC_SHA384,+TLS_DHE_RSA_WITH_AES_128_GCM_
> SHA256,+TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,+TLS_DHE_RSA_WITH_AES
> _256_GCM_SHA384,+TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Set EC ciphers as below:

> sslRangeCiphers="+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,+TLS_ECDHE_ECD
> SA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,+T
> LS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,+TLS_ECDHE_ECDSA_WITH_AES_1
> 28_GCM_SHA256,+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,+TLS_ECDHE_E
> CDSA_WITH_AES_256_GCM_SHA384

For a list of allowed ciphers, see Section 3.1, "Allowed TLS Cipher Suites" .

- If you install Certificate System with either LunaSA or nCipher Hardware Security Module (HSM) on systems with FIPS mode enabled for RSA, disable the following ciphers, as they are unsupported on HSMs in FIPS mode:

  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

  - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

9.4.1.2. Enabling Certificate Revocation Checking for Subsystems

The Certificate System subsystems do not have OCSP checking enabled by default. OCSP checking can be enabled for all subsystems by editing the **server.xml** file.

There is one allowed method to configure OCSP checking:

- Allowed Method: Enabling a Certificate System subsystem to use the OCSP responder URL specified in the peer certificate's Authority Information Access (AIA) extension: This method allows for a more dynamic environment where peer certificates can be issued by more than one CA. For example, if the certificate of the LDAP server is issued by a different CA than other certificates. For details, see Section 9.4.1.2.2, "Enabling a Certificate System Subsystem to use the OCSP Responder URL Specified in the Peer Certificate's Authority Information Access (AIA) Extension".

In the following sections, the configuration of the allowed OCSP checking method will be discussed, where various OCSP parameters in the **server.xml** file might be used.

The following table provides information on each parameter relevant to OCSP in the **server.xml** file.

Table 9.10. OCSP Parameters for server.xml

Parameter	Description
enableOCSP	Enables (or disables) OCSP checking for the subsystem.

Parameter	Description
ocspResponderURL	N/A
ocspResponderCertNickname	N/A

ocspCacheSize	Sets the maximum number of OCSP response cache entries kept by the Network Security Services (NSS).
	
	There are two special values:
	
	    • **-1**: Turns off the OCSP response cache.
	
	    • **0**: Allows an unlimited number of OCSP responses in the cache.

ocspMinCacheEntryDuration	Sets minimum seconds before another fetch attempt can be made. For example, if this is set to 120, then the validity of a certificate cannot be checked again until at least 2 minutes after the last validity check.
ocspMaxCacheEntryDuration	Sets the maximum number of seconds to wait before making the next fetch attempt. This prevents having too large a window between validity checks.
ocspTimeout	Sets the timeout period, in seconds, for the OCSP request.

NOTE

If a **nextUpdate** field is sent with the OCSP response, it can affect the next fetch time with *ocspMinCacheEntryDuration* and *ocspMaxCacheEntryDuration* like following:

- If the value in **nextUpdate** has been reached before the value set in *ocspMinCacheEntryDuration*, the fetch will not be started until the value set in *ocspMinCacheEntryDuration* has been reached.

- If *ocspMinCacheEntryDuration* has been reached, the server checks if the value in **nextUpdate** has been reached. If the value has been reached, the fetch will happen.

- Regardless of the value in **nextUpdate**, if the setting in *ocspMaxCacheEntryDuration* has been reached, the fetch will happen.

9.4.1.2.1. Setting Trust of the OCSP Signing Certificate

Each OCSP signing certificate must chain up to a trusted root in the Certificate System's NSS database.

Once the OCSP signing certificate is imported and trusted in the subsystem's NSS database, the following consideration is required. If the OCSP responder being used has been configured to provide the entire certificate chain of the OCSP signing certificate with each OCSP response, then no further action is required. NSS knows how to validate this chain from the request information. If on the other hand the OCSP is known to not return the full chain, the chain can be imported manually. For details, see

the section called "Importing an OCSP Responder" . The Certificate System OCSP responder already includes the chain with every response. This includes the Certificate System outboard OCSP responder and the internal OCSP service that comes with each CA.

This behavior is by default and cannot be changed.

9.4.1.2.2. Enabling a Certificate System Subsystem to use the OCSP Responder URL Specified in the Peer Certificate's Authority Information Access (AIA) Extension

To configure that Certificate System uses the OCSP responder URL specified in the peer certificate's Authority Information Access (AIA) extension, edit the **server.xml** file and

- Set the *enableOCSP* to **true**.

- Remove the *ocspResponderURL* parameter.

- Remove the *ocspResponderCertNickname* parameter.

For a full list of parameters you can set, see Table 9.10, "OCSP Parameters for server.xml".
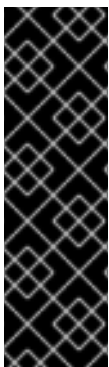
> **IMPORTANT**
>
> In addition to these settings, all peer certificates must contain the AIA extension if the desired result is an OCSP verification for that particular certificate. Otherwise, if the extension is missing, the verification step will be skipped.
>
> Also make sure that all included AIA extension URLs are correct and reachable. This is due to the fact that when the system tries to contact an unreachable OCSP server, the system will report back that the certificate in question has been rejected as a bad certificate.

Example 9.4. server.xml

```
<Connector name="Secure"
    enableOCSP="true"
    ocspCacheSize="1000"
    ocspMinCacheEntryDuration="60"
    ocspMaxCacheEntryDuration="120"
    ocspTimeout="10"
    ...
/>
```

Additionally, set the trust of the OCSP signing certificate. For details, see Section 9.4.1.2.1, "Setting Trust of the OCSP Signing Certificate".

9.4.1.2.3. Adding an AIA Extension to an Enrollment Profile

To set the AIA URL in the profile when using an external OCSP, add the correct URL to the certificate profile. For example:

```
policyset.cmcUserCertSet.5.default.params.authInfoAccessADLocation_0=http://example.com:8080/
ocsp/ee/ocsp
```

9.4.1.3. Session Timeout

When a user connects to PKI server through a client application, the server will create a session to keep track of the user. As long as the user remains active, the user can execute multiple operations over the same session without having to re-authenticate.

Session timeout determines how long the server will wait since the last operation before terminating the session due to inactivity. Once the session is terminated, the user will be required to re-authenticate to continue accessing the server, and the server will create a new session.

There are two types of timeouts:

- TLS session timeout

- HTTP session timeout

Due to differences in the way clients work, the clients will be affected differently by these timeouts.

> **NOTE**
>
> Certain clients have their own timeout configuration. For example, Firefox has a keep-alive timeout setting. For details, see http://kb.mozillazine.org/Network.http.keep-alive.timeout. If the value is different from the server's setting for TLS Session Timeout or HTTP Session Timeout, different behavior can be observed.

9.4.1.3.1. TLS Session Timeout

A TLS session is a secure communication channel over a TLS connection established through TLS handshake protocol.

PKI server generates audit events for TLS session activities. The server generates an **ACCESS_SESSION_ESTABLISH** audit event with **Outcome=Success** when the connection is created. If the connection fails to be created, the server will generate an **ACCESS_SESSION_ESTABLISH** audit event with **Outcome=Failure**. When the connection is closed, the server will generate an **ACCESS_SESSION_TERMINATED** audit event.

TLS session timeout (that is TLS connection timeout) is configured in the **keepAliveTimeout** parameter in the **Secure** <Connector> element in the **/etc/pki/<instance>/server.xml** file:

```
...
<Server>
 <Service>
  <Connector name="Secure"
   ...
   keepAliveTimeout="300000"
   ...
   />
 </Service>
</Server>
...
```

By default the timeout value is set to 300000 milliseconds (that is 5 minutes). To change this value, edit the **/etc/pki/<instance>/server.xml** file and then restart the server.

NOTE

Note that this value will affect all TLS connections to the server. A large value may improve the efficiency of the clients since they can reuse existing connections that have not expired. However, it may also increase the number of connections that the server has to support simultaneously since it takes longer for abandoned connections to expire.

9.4.1.3.2. HTTP Session Timeout

An HTTP session is a mechanism to track a user across multiple HTTP requests using HTTP cookies. PKI server does not generate audit events for the HTTP sessions.

NOTE

For the purpose of auditing consistency, set the **<session-timeout>** value in this section to match the **keepAliveTimeout** value in Section 9.4.1.3.1, "TLS Session Timeout". For example if **keepAliveTimeout** was set to *300000* (5 minutes), then set **<session-timeout>** to *30*.

The HTTP session timeout can be configured in the **<session-timeout>** element in the **/etc/pki/<instance>/web.xml** file:

```
...
<web-app>
 <session-config>
  <session-timeout>30</session-timeout>
 </session-config>
</web-app>
...
```

By default the timeout value is set to 30 minutes. To change the value, edit the **/etc/pki/<instance>/web.xml** file and then restart the server.

NOTE

Note that this value affects all sessions in all web applications on the server. A large value may improve the experience of the users since they will not be required to re-authenticate or view the access banner again so often. However, it may also increase the security risk since it takes longer for abandoned HTTP sessions to expire.

9.4.1.3.3. Session Timeout for PKI Web UI

PKI Web UI is an interactive web-based client that runs in a browser. Currently it only supports client certificate authentication.

When the Web UI is opened, the browser may create multiple TLS connections to a server. These connections are associated to a single HTTP session.

To configure a timeout for the Web UI, see Section 9.4.1.3.2, "HTTP Session Timeout". The TLS session timeout is normally irrelevant since the browser caches the client certificate so it can recreate the TLS session automatically.

When the HTTP session expires, the Web UI does not provide any immediate indication. However, the Web UI will display an access banner (if enabled) before a user executes an operation.

9.4.1.3.4. Session Timeout for PKI Console

PKI Console is an interactive standalone graphical UI client. It supports username/password and client certificate authentication.

When the console is started, it will create a single TLS connection to the server. The console will display an access banner (if enabled) before opening the graphical interface. Unlike the Web UI, the console does not maintain an HTTP session with the server.

To configure a timeout for the console, see Section 9.4.1.3.1, "TLS Session Timeout". The HTTP session timeout is irrelevant since the console does not use HTTP session.

When the TLS session expires, the TLS connection will close, and the console will exit immediately to the system. If the user wants to continue, the user will need to restart the console.

9.4.1.3.5. Session Timeout for PKI CLI

PKI CLI is a command-line client that executes a series of operations. It supports username/password and client certificate authentication.

When the CLI is started, it will create a single TLS connection to the server and an HTTP session. The CLI will display an access banner (if enabled) before executing operations.

Both timeouts are generally irrelevant to PKI CLI since the operations are executed in sequence without delay and the CLI exits immediately upon completion. However, if the CLI waits for user inputs, is slow, or becomes unresponsive, the TLS session or the HTTP session may expire and the remaining operations fail. If such delay is expected, see Section 9.4.1.3.1, "TLS Session Timeout" and Section 9.4.1.3.2, "HTTP Session Timeout" to accommodate the expected delay.

9.4.2. web.xml

9.4.2.1. Removing Unused Interfaces from web.xml (CA Only)

Several legacy interfaces (for features like bulk issuance or the policy framework) are still included in the CA's **web.xml** file. However, since these features are deprecated and no longer in use, then they can be removed from the CA configuration to increase security.

1. Stop the CA.

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

   OR

   ```
   systemctl stop pki-tomcatd-nuxwdog@instance_name.service (if using nuxwdog watchdog)
   ```

2. Open the web files directory for the CA. For example:

   ```
   cd /var/lib/pki/pki-tomcat/ca/webapps/ca/WEB-INF
   ```

3. Back up the current **web.xml** file.

   ```
   cp web.xml web.xml.servlets
   ```

4. Edit the **web.xml** file and remove the entire **<servlet>** entries for each of the following deprecated servlets:

   - caadminEnroll

   - cabulkissuance

   - cacertbasedenrollment

   - caenrollment

   - caProxyBulkIssuance

   For example, remove the **caadminEnroll** servlet entry:

   ```
   <servlet>
       <servlet-name> caadminEnroll </servlet-name>
       <servlet-class> com.netscape.cms.servlet.cert.EnrollServlet
   </servlet-class>
           <init-param><param-name> GetClientCert </param-name>
               <param-value> false      </param-value> </init-param>
           <init-param><param-name> successTemplate </param-name>
               <param-value> /admin/ca/EnrollSuccess.template
   </param-value> </init-param>
           <init-param><param-name> AuthzMgr   </param-name>
               <param-value> BasicAclAuthz </param-value>
   </init-param>
           <init-param><param-name> authority  </param-name>
               <param-value> ca        </param-value> </init-param>
           <init-param><param-name> interface  </param-name>
               <param-value> admin       </param-value>
   </init-param>
           <init-param><param-name> ID       </param-name>
               <param-value> caadminEnroll </param-value>
   </init-param>
           <init-param><param-name> resourceID </param-name>
               <param-value> certServer.admin.request.enrollment
   </param-value> </init-param>
           <init-param><param-name> AuthMgr    </param-name>
               <param-value> passwdUserDBAuthMgr </param-value>
   </init-param>
     </servlet>
   ```

5. After removing the servlet entries, remove the corresponding **<servlet-mapping>** entries.

   ```
   <servlet-mapping>
       <servlet-name> caadminEnroll </servlet-name>
       <url-pattern>  /admin/ca/adminEnroll </url-pattern>
   </servlet-mapping>
   ```

6. Remove three **<filter-mapping>** entries for an end-entity request interface.

   ```
    <filter-mapping>
       <filter-name> EERequestFilter           </filter-name>
       <url-pattern> /certbasedenrollment        </url-pattern>
   ```

```
</filter-mapping>

<filter-mapping>
  <filter-name> EERequestFilter          </filter-name>
  <url-pattern> /enrollment              </url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name> EERequestFilter          </filter-name>
  <url-pattern> /profileSubmit           </url-pattern>
</filter-mapping>
```

7. Start the CA again.

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

OR

```
systemctl start pki-tomcatd-nuxwdog@instance_name.service (if using nuxwdog watchdog)
```

9.4.3. Customizing Web Services

All of the subsystems (with the exception of the TKS) have some kind of a web-based services page for agents and some for other roles, like administrators or end entities. These web-based services pages use basic HTML and JavaScript, which can be customized to use different colors, logos, and other design elements to fit in with an existing site or intranet.

9.4.3.1. Customizing Subsystem Web Applications

Each PKI subsystem has a corresponding web application, which contains:

- HTML pages containing texts, JavaScript codes, page layout, CSS formatting, and so on

- A **web.xml** file, which defines servlets, paths, security constraints, and other

- Links to PKI libraries.

The subsystem web applications are deployed using context files located in the **/var/lib/pki/pki-tomcat/conf/Catalina/localhost/** directory, for example, the **ca.xml** file:

```
<Context docBase="/usr/share/pki/ca/webapps/ca" crossContext="true" allowLinking="true">
  ...
</Context>
```

The **docBase** points to the location of the default web application directory, **/usr/share/pki/**.

To customize the web application, copy the web application directory into the instance's **webapps** directory:

```
$ cp -r /usr/share/pki/ca/webapps/ca /var/lib/pki/pki-tomcat/webapps
```

Then change the **docBase** to point to the custom web application directory relative from the **webapps** directory:

```
<Context docBase="ca" crossContext="true" allowLinking="true">
   ...
</Context>
```

The change will be effective immediately without the need to restart the server.

To remove the custom web application, simply revert the **docBase** and delete the custom web application directory:

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/ca
```

9.4.3.2. Customizing the Web UI Theme

The subsystem web applications in the same instance share the same theme, which contains:

- CSS files, which determine the global appearance

- Image files including logo, icons, and other

- Branding properties, which determine the page title, logo link, title color, and other.

The Web UI theme is deployed using the **pki.xml** context file in the **/var/lib/pki/pki-tomcat/conf/Catalina/localhost/** directory:

```
<Context docBase="/usr/share/pki/common-ui" crossContext="true" allowLinking="true">
   ...
</Context>
```

The docBase points to the location of the default theme directory, **/usr/share/pki/**.

To customize the theme, copy the default theme directory into the **pki** directory in the instance's **webapps** directory:

```
$ cp -r /usr/share/pki/common-ui /var/lib/pki/pki-tomcat/webapps/pki
```

Then change the **docBase** to point to the custom theme directory relative from the **webapps** directory:

```
<Context docBase="pki" crossContext="true" allowLinking="true">
   ...
</Context>
```

The change will be effective immediately without the need to restart the server.

To remove the custom theme, simply revert the **docBase** and delete the custom theme directory:

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/pki
```

9.4.3.3. Customizing TPS Token State Labels

The default token state labels are stored in the **/usr/share/pki/tps/conf/token-states.properties** file and described in Section 2.5.2.4.1.4, "Token State and Transition Labels" .

To customize the labels, copy the file into the instance directory:

```
$ cp /usr/share/pki/tps/conf/token-states.properties /var/lib/pki/pki-tomcat/tps/conf
```

The change will be effective immediately without the need to restart the server.

To remove the customized labels, simply delete the customized file:

```
$ rm /var/lib/pki/pki-tomcat/tps/conf/token-states.properties
```

9.5. USING AN ACCESS BANNER

In Certificate System, Administrators can configure a banner with customizable text. The banner will be displayed in the following situations:

Application	When the banner is displayed
PKI Console	- Before the console is displayed. - After the session has expired. [a]
Web interface	- When you connect to the web interface. - After the session expired.[a]
pki command-line utility	Before the actual operation proceeds.

[a] For details about changing the session timeout, see Section 9.4.1.3, "Session Timeout".

You can use the banner to display important information to the users before they can use Certificate System. The user must agree to the displayed text to continue.

> **Example 9.5. When the Access Banner is Displayed**
>
> The following example shows when the access banner is displayed if you are using the **pki** utility:
>
> ```
> # $ pki cert-show 0x1
> WARNING! Access to this service is restricted to those individuals with specific permissions. If you
> are not an authorized user, disconnect now. Any attempts to gain unauthorized access will be
> prosecuted to the fullest extent of the law. Do you want to proceed (y/N)? y
> -----------------
> Certificate "0x1"
> -----------------
>   Serial Number: 0x1
>   Issuer: CN=CA Signing Certificate,OU=instance_name,O=EXAMPLE
>   Subject: CN=CA Signing Certificate,OU=instance_name,O=EXAMPLE
>   Status: VALID
>   Not Before: Mon Feb 20 18:21:03 CET 2017
>   Not After: Fri Feb 20 18:21:03 CET 2037
> ```

9.5.1. Enabling an Access Banner

To enable the access banner, create the **/etc/pki/*instance_name*/banner.txt** file and enter the text to displayed.
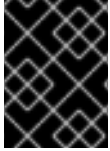
> **IMPORTANT**
>
> The text in the **/etc/pki/*instance_name*/banner.txt** file must use the UTF-8 format. To validate, see Section 9.5.3, "Validating the Banner".

9.5.2. Displaying the Banner

To display the currently configured banner:

```
# pki-server banner-show -i instance_name
```

9.5.3. Validating the Banner

To validate that the banner does not contain invalid characters:

```
# pki-server banner-validate -i instance_name
---------------
Banner is valid
---------------
```

9.6. CONFIGURATION FOR CMC

This section describes how to configure Certificate System for Certificate Management over CMS (CMC).

9.6.1. Understanding How CMC Works

Before configuring CMC, read the following documentation to learn more about the subject:

- Section 2.4.1.1.1, "Enrolling with CMC"

- *Requesting and Receiving Certificates Using CMC* in the *Certificate System Administration Guide (Common Criteria Edition)*.

- *Making Rules for Issuing Certificates (Certificate Profiles)* in the *Certificate System Administration Guide (Common Criteria Edition)*.

9.6.2. Enabling the `PopLinkWittnessV2` Feature

For a high-level security on the Certificate Authority (CA), enable the following option in the **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** file:

```
cmc.popLinkWitnessRequired=true
```

9.6.3. Enabling the CMC Shared Secret Feature

To enable the shared token feature in a Certificate Authority (CA):

1. If the watchdog service is enabled on the host, temporarily disable it. See Section 9.3.2.4, "Disabling the Watchdog Service".

2. Add the **shrTok** attribute to Directory Server's schema:

   ```
   # ldapmodify -D "cn=Directory Manager" -H ldaps://server.example.com:636 -W -x

   dn: cn=schema
   changetype: modify
   add: attributetypes
   attributetypes: ( 2.16.840.1.117370.3.1.123 NAME 'shrTok' DESC 'User
    Defined ObjectClass for SharedToken' SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
    SINGLE-VALUE X-ORIGIN 'custom for sharedToken')
   ```

3. If the system keys are stored on a Hardware Security Module (HSM), set the ***cmc.token*** parameter in the **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** file. For example:

   ```
   cmc.token=NHSM6000
   ```

4. Enable the shared token authentication plug-in by using one of the following methods:

   - To enable the plug-in using the **pkiconsole** utility:

     1. Log into the system using the **pkiconsole** utility. For example:

        ```
        # pkiconsole https:host.example.com:8443/ca
        ```

     2. On the **Configuration** tab, select **Authentication**.

     3. Click **Add** and select **SharedToken**.

     4. Click **Next**.

     5. 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
        ```

     6. Click **OK**.

   - To manually enable the plug-in, add the following settings into the **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** file:

```
auths.impl.SharedToken.class=com.netscape.cms.authentication.SharedSecret
auths.instance.SharedToken.dnpattern=
auths.instance.SharedToken.ldap.basedn=ou=People,dc=example,dc=org
auths.instance.SharedToken.ldap.ldapauth.authtype=BasicAuth
auths.instance.SharedToken.ldap.ldapauth.bindDN=cn=Directory Manager
auths.instance.SharedToken.ldap.ldapauth.bindPWPrompt=Rule SharedToken
auths.instance.SharedToken.ldap.ldapauth.clientCertNickname=
auths.instance.SharedToken.ldap.ldapconn.host=server.example.com
auths.instance.SharedToken.ldap.ldapconn.port=636
auths.instance.SharedToken.ldap.ldapconn.secureConn=true
auths.instance.SharedToken.ldap.ldapconn.version=3
auths.instance.SharedToken.ldap.maxConns=
auths.instance.SharedToken.ldap.minConns=
auths.instance.SharedToken.ldapByteAttributes=
auths.instance.SharedToken.ldapStringAttributes=
auths.instance.SharedToken.pluginName=SharedToken
auths.instance.SharedToken.shrTokAttr=shrTok
```

5. Set the nickname of an RSA issuance protection certificate in the **ca.cert.issuance_protection.nickname** parameter in the **/var/lib/pki/instance_name/ca/conf/CS.cfg** file. For example:

```
ca.cert.issuance_protection.nickname=issuance_protection_certificate
```

This step is:

- Optional if you use an RSA certificate in the **ca.cert.subsystem.nickname** parameter.

- Required if you use an ECC certificate in the **ca.cert.subsystem.nickname** parameter.
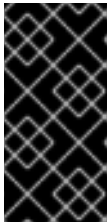
> **IMPORTANT**
>
> If the **ca.cert.issuance_protection.nickname** parameter is not set, Certificate System automatically uses the certificate of the subsystem specified in the **ca.cert.subsystem.nickname**. However, the issuance protection certificate must be an RSA certificate.

6. Restart Certificate System:

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

When the CA starts, Certificate System prompts for the LDAP password used by the Shared Token plug-in.

7. If you temporarily disabled the watchdog service at the beginning of this procedure, re-enable it. See Section 9.3.2.1, "Enabling the Watchdog Service".

9.6.4. Enabling CMCRevoke for the Web User Interface

As described in the *Performing a CMC Revocation* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*, there are two ways to submit CMC revocation requests.

In case when you use the **CMCRevoke** utility to create revocation requests to be submitted through the web UI, add the following setting to the **/var/lib/pki/instance_name/ca/conf/CS.cfg** file:

cmc.bypassClientAuth=true

CHAPTER 10. 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 Section 2.3.8.1, "NSS Soft Token (internal token)".

For information about HSM, please see Section 2.3.8.2, "Hardware Security Module (HSM, external token)".

10.1. 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.

10.1.1. 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.

10.1.2. PKICertImport Basic Usage

PKICertImport [options]

Each **PKICertImport** invocation takes 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 10.1.4, "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.

10.1.3. 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 10.1. 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.

10.1.4. 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.

10.2. 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 10.1, "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.

10.3. 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 10.2, "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 10.1, "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.

10.4. IMPORTING A CERTIFICATE INTO AN HSM

This procedure describes how to import a certificate into an HSM after gaining a newly issued certificate (such as when a system certificate is renewed) whose keys were generated on the same HSM token as the process of creating a CSR.

Before beginning, please change directories into the NSS DB:

- **cd** */path/to/nssdb* for example **cd /var/lib/pki/pki-ca/alias**

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.

If you have not imported and trusted the root certificate, see Section 10.2, "Importing a Root Certificate". If you have not imported and validated the intermediate certificates, see Section 10.3, "Importing an Intermediate Certificate Chain".

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

- For TLS server certs for all PKI substems, follow the *server certificate steps*.

- For any subsystem's audit signing cert, follow the steps below for validating an *object Signing certificate*.

- For the CA subsystem's signing cert, follow the steps above for importing and validating an *intermediate certificate chain*, but do so only with the **caSigningCert**.

- For the CA subsystem's OCSP signing cert, follow the steps below for validating an *OCSP certificate*.

- For all other system certs of the PKI subsystems, follow the *Client Certificate steps*.

For more information about the **certutil** and **PKICertImport** options used below, see Section 10.1, "About **certutil** and **PKICertImport**".

To import a server certificate on the HSM:

- Execute **PKICertImport -d . -h HSM -n "host.name.example.com" -t ",," -a -i service.crt -u V**

  This command validates and imports the server certificate onto the HSM. 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 due to expiry of a parent certificate or a missing CA trust chain (such as a missing intermediate certificate or a missing CA Root). If the validation does not succeed, contact the issuer and ensure that all intermediate and root certificates are present on your system.

To import a client certificate on the HSM:

- Execute **PKICertImport -d . -h HSM -n "client name" -t ",," -a -i client.crt -u C**

  This command validates and imports the client certificate onto the HSM. 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.

To import an object signing certificate on the HSM:

- Execute **PKICertImport -d . -h HSM -n "certificate name" -t ",,P" -a -i objectsigning.crt -u J**

  This command validates and imports the object signing certificate onto the HSM. 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.

To import an OCSP response signing certificate on the HSM:

- Execute **PKICertImport -d . -h HSM -n "certificate name" -t ",," -a -i ocsp.crt -u O**

  This command validates and imports the OCSP responder certificate onto the HSM. 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.

10.5. 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 10.1, "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 10.2, "Importing a Root Certificate".

3. Import and validate the intermediate certificates, if not already imported and validated. For details, see Section 10.3, "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 10.2, "Importing a Root Certificate" .

3. Import and validate the intermediate certificates, if not already imported and validated. For details, see Section 10.3, "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 10.2, "Importing a Root Certificate" .

3. Import and validate the intermediate certificates, if not already imported and validated. For details, see Section 10.3, "Importing an Intermediate Certificate Chain" .

4. Validate and import the OCSP responder certificate:

   ```
   # PKICertImport -d . -n "certificate name" -t ",," -a -i ocsp.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 11. CERTIFICATE PROFILES CONFIGURATION

11.1. CREATING AND EDITING CERTIFICATE PROFILES DIRECTLY ON THE FILE SYSTEM

As part of the installation process of a CA, the certificate enrollment profiles can be modified directly on the file system by modifying the profiles' configuration files. Default files exist for the default profiles at installation; when new profiles are needed, new profile configuration files are to be created. The configuration files are stored in the CA profile directory, *instance_directory*/**ca**/**profiles**/**ca**/, such as /**var**/**lib**/**pki**/**pki-ca**/**ca**/**profiles**/**ca**/. The file is named *profile_name*.**cfg**. All of the parameters for profile rules can be set or modified in those profile configuration files. Profile rules can be inputs, outputs, authentication, authorization, defaults, and constraints.

The enrollment profiles for the CA certificates are located in the /**var**/**lib**/**pki**/**instance_name**/**ca**/**conf** directory with the name *** .profile**.

> **NOTE**
>
> For audit reasons, use this method only during the CA installation prior to deployment.
>
> Restart the server after editing the profile configuration file for the changes to take effect.

- Section 11.1.1.1, "Profile Configuration Parameters"

- Section 11.1.1.2, "Modifying Certificate Extensions Directly on the File System"

- Section 11.1.1.3, "Adding Profile Inputs Directly on the File System"

11.1.1. Configuring non-CA System Certificate Profiles

11.1.1.1. Profile Configuration Parameters

All of the parameters for a profile rule – defaults, inputs, outputs, and constraints – are configured within a single policy set. A policy set for a profile has the name **policyset.***policyName.policyNumber*. For example:

```
policyset.cmcUserCertSet.6.constraint.class_id=noConstraintImpl
policyset.cmcUserCertSet.6.constraint.name=No Constraint
policyset.cmcUserCertSet.6.default.class_id=userExtensionDefaultImpl
policyset.cmcUserCertSet.6.default.name=User Supplied Key Default
policyset.cmcUserCertSet.6.default.params.userExtOID=2.5.29.15
```

The common profile configuration parameters are described in Table 11.1, "Profile Configuration File Parameters".

Table 11.1. Profile Configuration File Parameters

Parameter	Description

Parameter	Description
desc	Gives a free text description of the certificate profile, which is shown on the end-entities page. For example, *desc=This certificate profile is for enrolling server certificates with agent authentication.*
enable	Sets whether the profile is enabled, and therefore accessible through the end-entities page. For example, *enable=true*.
auth.instance_id	Sets which authentication manager plug-in to use to authenticate the certificate request submitted through the profile. For automatic enrollment, the CA issues a certificate immediately if the authentication is successful. If authentication fails or there is no authentication plug-in specified, the request is queued to be manually approved by an agent. For example, *auth.instance_id=CMCAuth*. The authentication method must be one of the registered authentication instances from **CS.cfg**.
authz.acl	Specifies the authorization constraint. Most commonly, this us used to set the group evaluation ACL. For example, this caCMCUserCert parameter requires that the signer of the CMC request belong to the Certificate Manager Agents group: *authz.acl=group="Certificate Manager Agents"* In directory-based user certificate renewal, this option is used to ensure that the original requester and the currently-authenticated user are the same. An entity must authenticate (bind or, essentially, log into the system) before authorization can be evaluated. The authorization method specified must be one of the registered authorization instances from **CS.cfg**.
name	Gives the name of the profile. For example, *name=Agent-Authenticated Server Certificate Enrollment*. This name is displayed in the end users enrollment or renewal page.
input.list	Lists the allowed inputs for the profile by name. For example, *input.list=i1,i2*.
input.*input_id*.class_id	Gives the java class name for the input by input ID (the name of the input listed in *input.list*). For example, *input.i1.class_id=cmcCertReqInputImpl*.
output.list	Lists the possible output formats for the profile by name. For example, *output.list=o1*.

Parameter	Description
output.*output_id*.class_id	Gives the java class name for the output format named in *output.list*. For example, *output.o1.class_id=certOutputImpl*.
policyset.list	Lists the configured profile rules. For dual certificates, one set of rules applies to the signing key and the other to the encryption key. Single certificates use only one set of profile rules. For example, *policyset.list=serverCertSet*.
policyset.*policyset_id*.list	Lists the policies within the policy set configured for the profile by policy ID number in the order in which they should be evaluated. For example, *policyset.serverCertSet.list=1,2,3,4,5,6,7,8*.
policyset.*policyset_id.policy_number*.constraint.class_id	Gives the java class name of the constraint plug-in set for the default configured in the profile rule. For example, *policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl*.
policyset.*policyset_id.policy_number*.constraint.name	Gives the user-defined name of the constraint. For example, *policyset.serverCertSet.1.constraint.name=Subject Name Constraint*.
policyset.*policyset_id.policy_number*.constraint.params.*attribute*	Specifies a value for an allowed attribute for the constraint. The possible attributes vary depending on the type of constraint. For example, *policyset.serverCertSet.1.constraint.params.pattern=CN=.**.
policyset.*policyset_id.policy_number*.default.class_id	Gives the java class name for the default set in the profile rule. For example, *policyset.serverCertSet.1.default.class_id=userSubjectNameDefaultImpl*
policyset.*policyset_id.policy_number*.default.name	Gives the user-defined name of the default. For example, *policyset.serverCertSet.1.default.name=Subject Name Default*
policyset.*policyset_id.policy_number*.default.params.*attribute*	Specifies a value for an allowed attribute for the default. The possible attributes vary depending on the type of default. For example, *policyset.serverCertSet.1.default.params.name=CN=(Name)$request.requestor_name$*.

11.1.1.2. Modifying Certificate Extensions Directly on the File System

Changing constraints changes the restrictions on the type of information which can be supplied. Changing the defaults and constraints can also add, delete, or modify the extensions which are accepted or required from a certificate request.

For example, the default caFullCMCUserCert profile is set to create a Key Usage extension from information in the request.

```
 policyset.cmcUserCertSet.6.constraint.class_id=keyUsageExtConstraintImpl
policyset.cmcUserCertSet.6.constraint.name=Key Usage Extension Constraint
policyset.cmcUserCertSet.6.constraint.params.keyUsageCritical=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageCrlSign=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDataEncipherment=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDigitalSignature=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageEncipherOnly=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyAgreement=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyCertSign=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyEncipherment=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageNonRepudiation=true
policyset.cmcUserCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.cmcUserCertSet.6.default.name=Key Usage Default
policyset.cmcUserCertSet.6.default.params.keyUsageCritical=true
policyset.cmcUserCertSet.6.default.params.keyUsageCrlSign=false
policyset.cmcUserCertSet.6.default.params.keyUsageDataEncipherment=false
policyset.cmcUserCertSet.6.default.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.6.default.params.keyUsageDigitalSignature=true
policyset.cmcUserCertSet.6.default.params.keyUsageEncipherOnly=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyAgreement=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyEncipherment=true
policyset.cmcUserCertSet.6.default.params.keyUsageNonRepudiation=true
```

The default is updated to allow user-supplied key extensions:

```
policyset.cmcUserCertSet.6.default.class_id=userExtensionDefaultImpl
policyset.cmcUserCertSet.6.default.name=User Supplied Key Default
policyset.cmcUserCertSet.6.default.params.userExtOID=2.5.29.15
```

This sets the server to accept the extension OID **2.5.29.15** in the certificate request.

Other constraints and defaults can be changed similarly. Make sure that any required constraints are included with appropriate defaults, that defaults are changed when a different constraint is required, and that only allowed constraints are used with the default. For more information, see the Defaults Reference and Constraints Reference sections in Red Hat Certificate System Administration Guide.

11.1.1.2.1. Key Usage and Extended Key Usage Consistency

Red Hat Certificate System provides a flexible infrastructure for administrators to create customized enrollment profiles to meet the requirements of their environment. However, it is important that profiles do not allow issuing certificates that violate the requirements defined in RFC 5280. When creating an enrollment profile where both Key Usage (KU) and Extended Key Usage (EKU) extensions are present, it is important to make sure that the consistency between the two extensions is maintained as per section 4.2.1.12. Extended Key Usage of RFC 5280.

For details about the KU extension, see:

The following table provides the guidelines that maps consistent Key Usage bits to the Extended Key Usage Extension for each purpose:

Purpose / Extended Key Usages	Key Usages
TLS Server Authentication command **id-kp-serverAuth**	**digitalSignature**, **keyEncipherment**, or **KeyAgreement**
TLS Client (Mutual) Authentication **id-kp-clientAuth**	**digitalSignature**, **keyEncipherment**, and/or **KeyAgreement**
Code Signing **id-kp-codeSigning**	**digitalSignature**
Email Protection **id-kp-emailProtection**	**digitalSignature**, **nonRepudiation**, and/or (**keyEncipherment** or **keyAgreement**)
OCSP Response Signing **id-kp-OCSPSigning**	**KeyAgreement** and/or **nonRepudiation**

The following shows two examples of inconsistent EKU/KU:

- An enrollment profile that is intended for purpose of OCSP response signing contains Extended key usage **id-kp-OCSPSigning** but with **keyEncipherment** key usage bit:

```
policyset.ocspCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.ocspCertSet..6.default.name=Key Usage Default
policyset.ocspCertSet..6.default.params.keyUsageCritical=true
policyset.ocspCertSet..6.default.params.keyUsageCrlSign=false
policyset.ocspCertSet..6.default.params.keyUsageDataEncipherment=false
policyset.ocspCertSet..6.default.params.keyUsageDecipherOnly=false
policyset.ocspCertSet..6.default.params.keyUsageDigitalSignature=true
policyset.ocspCertSet..6.default.params.keyUsageEncipherOnly=false
policyset.ocspCertSet..6.default.params.keyUsageKeyAgreement=false
policyset.ocspCertSet..6.default.params.keyUsageKeyCertSign=false
policyset.ocspCertSet..6.default.params.keyUsageKeyEncipherment=true
policyset.ocspCertSet..6.default.params.keyUsageNonRepudiation=true
policyset.ocspCertSet.7.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.9
policyset.ocspCertSet.7.default.class_id=extendedKeyUsageExtDefaultImpl
policyset.ocspCertSet.7.default.name=Extended Key Usage Default
policyset.ocspCertSet.7.default.params.exKeyUsageCritical=false
policyset.ocspCertSet.7.default.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.9
```

- An enrollment profile that is intended for the purpose of TLS server authentication contains Extended key **usage id-kp-serverAuth** but with CRL signing key usage bit:

```
policyset.serverCertSet.6.default.name=Key Usage Default
policyset.serverCertSet.6.default.params.keyUsageCritical=true
policyset.serverCertSet.6.default.params.keyUsageDigitalSignature=true
```

```
policyset.serverCertSet.6.default.params.keyUsageNonRepudiation=false
policyset.serverCertSet.6.default.params.keyUsageDataEncipherment=true
policyset.serverCertSet.6.default.params.keyUsageKeyEncipherment=false
policyset.serverCertSet.6.default.params.keyUsageKeyAgreement=true
policyset.serverCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.serverCertSet.6.default.params.keyUsageCrlSign=true
policyset.serverCertSet.6.default.params.keyUsageEncipherOnly=false
policyset.serverCertSet.6.default.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.7.default.class_id=extendedKeyUsageExtDefaultImpl
policyset.cmcUserCertSet.7.default.name=Extended Key Usage Extension Default
policyset.cmcUserCertSet.7.default.params.exKeyUsageCritical=false
policyset.serverCertSet.7.default.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.1
```

- The *Extended Key Usage Extension Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

- The *keyUsage* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

For details about the EKU extension, see:

- The *Extended Key Usage Extension Constraint* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

- The *extKeyUsage* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

11.1.1.3. Adding Profile Inputs Directly on the File System

The certificate profile configuration file in the CA's **profiles/ca** directory contains the input information for that particular certificate profile form. Inputs are the fields in the end-entities page enrollment forms. There is a parameter, **input.list**, which lists the inputs included in that profile. Other parameters define the inputs; these are identified by the format **input.***ID*. For example, this adds a generic input to a profile:

```
input.list=i1,i2,i3,i4
...
input.i4.class_id=genericInputImpl
input.i4.params.gi_display_name0=Name0
input.i4.params.gi_display_name1=Name1
input.i4.params.gi_display_name2=Name2
input.i4.params.gi_display_name3=Name3
input.i4.params.gi_param_enable0=true
input.i4.params.gi_param_enable1=true
input.i4.params.gi_param_enable2=true
input.i4.params.gi_param_enable3=true
input.i4.params.gi_param_name0=gname0
input.i4.params.gi_param_name1=gname1
input.i4.params.gi_param_name2=gname2
input.i4.params.gi_param_name3=gname3
input.i4.params.gi_num=4
```

For more information on what inputs, or form fields, are available, see the Input Reference section in Red Hat Certificate System Administration Guide.

11.1.2. Changing the Default Validity Time of Certificates

In each profile on a Certificate Authority (CA), you can set how long certificates issued using a profile are valid. You can change this value for security reasons.

For example, to set the validity of the generated Certificate Authority (CA) signing certificate to **825** days (approximately 27 months), open the **/var/lib/pki/*instance_name*/ca/profiles/ca/caCACert.cfg** file in an editor and set:

> policyset.caCertSet.2.default.params.range=825

11.1.3. Configuring CA System Certificate Profiles

Unlike the non-CA subsystems, the enrollment profiles for CA's own system certificates are kept in the **/var/lib/pki/[instance name]/ca/conf** file. Those profiles are:

- **caAuditSigningCert.profile**

- **eccAdminCert.profile**

- **rsaAdminCert.profile**

- **caCert.profile**

- **eccServerCert.profile**

- **saServerCert.profile**

- **caOCSPCert.profile**

- **eccSubsystemCert.profile**

- **rsaSubsystemCert.profile**

If you wish to change the default values in the profiles above, make changes to the profiles before Section 7.3.2.4, "Starting the **pkispawn** Step Two Installation" procedure is performed. The following is an example that demonstrates:

- How to change validity to CA signing certificate.

- How to add extensions (e.g. Certificate policies extension).

1. Back up the original CA certificate profile used by **pkispawn**.

   > cp -p /usr/share/pki/ca/conf/caCert.profile /usr/share/pki/ca/conf/caCert.profile.orig

2. Open the CA certificate profile used by the configuration wizard.

   > vim /usr/share/pki/ca/conf/caCert.profile

3. Reset the validity period in the Validity Default to whatever you want. For example, to change the period to two years:

```
2.default.class=com.netscape.cms.profile.def.ValidityDefault
2.default.name=Validity Default
2.default.params.range=7200
```

4. Add any extensions by creating a new default entry in the profile and adding it to the list. For example, to add the Certificate Policies Extension, add the default (which, in this example, is default #9):

```
9.default.class_id=certificatePoliciesExtDefaultImpl
9.default.name=Certificate Policies Extension Default
9.default.params.Critical=false
9.default.params.PoliciesExt.certPolicy0.enable=false
9.default.params.PoliciesExt.certPolicy0.policyId=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.enable=true
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.value=CertificatePolicies.exa
mple.com
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.enable=false
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.explicitText.value=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeReference.noticeNu
mbers=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeReference.organizat
on=
```

Then, add the default number to the list of defaults to use the new default:

```
list=2,4,5,6,7,8,9
```

11.1.4. Managing Smart Card CA Profiles
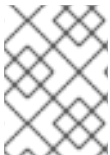
NOTE

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

The TPS does not generate or approve certificate requests; it sends any requests approved through the Enterprise Security Client to the configured CA to issue the certificate. This means that the CA actually contains the profiles to use for tokens and smart cards. The profiles to use can be automatically assigned, based on the card type.

The profile configuration files are in the **/var/lib/pki/***instance_name***/profiles/ca/** directory with the other CA profiles. The default profiles are listed in Table 11.2, "Default Token Certificate Profiles" .

Table 11.2. Default Token Certificate Profiles

Profile Name	Configuration File	Description
Regular Enrollment Profiles		
Token Device Key Enrollment	caTokenDeviceKeyEnrollment.cfg	For enrolling tokens used for devices or servers.

Profile Name	Configuration File	Description
Token User Encryption Certificate Enrollment	caTokenUserEncryptionKeyEnrollment.cfg	For enrolling encryption certificates on the token for a user.
Token User Signing Certificate Enrollment	caTokenUserSigningKeyEnrollment.cfg	For enrolling signing certificates on the token for a user.
Token User MS Login Certificate Enrollment	caTokenMSLoginEnrollment.cfg	For enrolling user certificates to use for single sign-on to a Windows domain or PC.
Temporary Token Profiles		
Temporary Device Certificate Enrollment	caTempTokenDeviceKeyEnrollment.cfg	For enrolling certificates for a device on a temporary token.
Temporary Token User Encryption Certificate Enrollment	caTempTokenUserEncryptionKeyEnrollment.cfg	For enrolling an encryption certificate on a temporary token for a user.
Temporary Token User Signing Certificate Enrollment	caTempTokenUserSigningKeyEnrollment.cfg	For enrolling a signing certificates on a temporary token for a user.
Renewal Profiles[a]		
Token User Encryption Certificate Enrollment (Renewal)	caTokenUserEncryptionKeyRenewal.cfg	For renewing encryption certificates on the token for a user, if renewal is allowed.
Token User Signing Certificate Enrollment (Renewal)	caTokenUserSigningKeyRenewal.cfg	For renewing signing certificates on the token for a user, if renewal is allowed.

[a] Renewal profiles can only be used in conjunction with the profile that issued the original certificate. There are two settings that are beneficial:

- It is important the original enrollment profile name does not change.

- The Renew Grace Period Constraint should be set in the original enrollment profile. This defines the amount of time before and after the certificate's expiration date when the user is allowed to renew the certificate. There are only a few examples of these in the default profiles, and they are mostly not enabled by default.

11.1.4.1. Editing Enrollment Profiles for the TPS

Administrators have the ability to customize the default smart card enrollment profiles, used with the TPS. For instance, a profile could be edited to include the user's email address in the Subject Alternative Name extension. The email address for the user is retrieved from the authentication directory. To configure the CA for LDAP access, change the following parameters in the profile files, with the appropriate directory information:

```
policyset.set1.p1.default.params.dnpattern=UID=$request.uid$, O=Token Key User
   policyset.set1.p1.default.params.ldap.enable=true
   policyset.set1.p1.default.params.ldap.basedn=ou=people,dc=host,dc=example,dc=com
   policyset.set1.p1.default.params.ldapStringAttributes=uid,mail
   policyset.set1.p1.default.params.ldap.ldapconn.host=localhost.example.com
   policyset.set1.p1.default.params.ldap.ldapconn.port=389
```

These CA profiles come with LDAP lookup disabled by default. The **ldapStringAttributes** parameter tells the CA which LDAP attributes to retrieve from the company directory. For example, if the directory contains **uid** as an LDAP attribute name, and this will be used in the subject name of the certificate, then **uid** must be listed in the **ldapStringAttributes** parameter, and *request.uid* listed as one of the components in the **dnpattern**.

Editing certificate profiles is covered in 3.2. Setting up Certificate Profiles in Red Hat Certificate System's Administration Guide.

The format for the **dnpattern** parameter is covered in B.2.11. Subject Name Constraint and B.1.28. Subject Name Default in Red Hat Certificate System Administration Guide.

11.1.4.2. Creating Custom TPS Profiles

Certificate profiles are created as normal in the CA, but they also have to be configured in the TPS for it to be available for token enrollments.

> **NOTE**
>
> New profiles are added with new releases of Red Hat Certificate System. If an instance is migrated to Certificate System 9.0, then the new profiles need to be added to the migrated instance as if they are custom profiles.

1. Create a new token profile for the issuing CA. Setting up profiles is covered in 3.2. Setting up Certificate Profiles in Red Hat Certificate System's Administration Guide.

2. Copy the profile into the CA's profiles directory, **/var/lib/*instance_name*/ca/profiles/ca/**.

3. Edit the CA's **CS.cfg** file, and add the new profile references and the profile name to the CA's list of profiles. For example:

   ```
   vim etc/pki/instance_name/ca/CS.cfg

      profile.list=caUserCert,...,caManualRenewal,tpsExampleEnrollProfile
      ...
      profile.caTokenMSLoginEnrollment.class_id=caUserCertEnrollImpl

   profile.caTokenMSLoginEnrollment.config=/var/lib/pki/instance_name/profiles/ca/tpsExampleE
   nrollProfile.cfg
   ```

4. Edit the TPS **CS.cfg** file, and add a line to point to the new CA enrollment profile. For example:

   ```
   vim /etc/pki/instance_name/tps/CS.cfg

      op.enroll.userKey.keyGen.signing.ca.profileId=tpsExampleEnrollProfile
   ```

5. Restart the instance after editing the smart card profiles:

> systemctl restart pki-tomcatd-nuxwdog@*instance_name*.service

If the CA and TPS are in separate instances, restart both instances.

> **NOTE**
>
> Enrollment profiles for the External Registration (**externalReg**) setting are configured in the user LDAP entry.

11.1.4.3. Using the Windows Smart Card Logon Profile

The TPS uses a profile to generate certificates to use for single sign-on to a Windows domain or PC; this is the Token User MS Login Certificate Enrollment profile (**caTokenMSLoginEnrollment.cfg**).

However, there are some special considerations that administrators *must* account for when configuring Windows smart card login.

- Issue a certificate to the domain controller, if it is not already configured for TLS.

- Configure the smart card login per user, rather than as a global policy, to prevent locking out the domain administrator.

- Enable CRL publishing to the Active Directory server because the domain controller checks the CRL at every login.

11.1.5. Disabling Certificate Enrolment Profiles

This section provides instructions on how to disable all non-Certificate Management over CMS (non-CMC) profiles.

To disable a certificate profile, edit the corresponding ***.cfg** file in the **/var**/**lib**/**pki**/*instance_name*/**ca**/**profiles**/**ca**/ directory and set the *visible* and *enable* parameters to **false**.

To disable all non-CMC profiles:

1. List all non-CMC profiles:

   > # ls -l /var/lib/pki/*instance_name*/ca/profiles/ca/ | grep -v "CMC"

2. In each of the displayed files, set the following parameters to **false**:

   > visible=false
   > enable=false

Additionally, set **visible=false** in all CMC profiles to make them invisible on the end entity page:

1. List all CMC profiles:

   > # ls -l /var/lib/pki/*instance_name*/ca/profiles/ca/*CMC*

2. In each of the displayed files, set:

> visible=false

CHAPTER 12. CONFIGURING THE KEY RECOVERY AUTHORITY

12.1. MANUALLY SETTING UP KEY ARCHIVAL

> **IMPORTANT**
>
> This procedure is unnecessary if the CA and KRA are in the same security domain. This procedure is only required for CAs *outside* the security domain.

Configuring key archival manually demands two things:

- Having a trusted relationship between a CA and a KRA.

- Having the enrollment form enabled for key archival, meaning it has key archival configured and the KRA transport certificate stored in the form.

In the same security domain, both of these configuration steps are done *automatically* when the KRA is configured because it is configured to have a trusted relationship with any CA in its security domain. It is possible to create that trusted relationship with Certificate Managers outside its security domain by manually configuring the trust relationships and profile enrollment forms.

1. If necessary, create a trusted manager to establish a relationship between the Certificate Manager and the KRA.

   For the CA to be able to request key archival of the KRA, the two subsystems must be configured to recognize, trust, and communicate with each other. Verify that the Certificate Manager has been set up as a privileged user, with an appropriate TLS client authentication certificate, in the internal database of the KRA. By default, the Certificate Manager uses its subsystem certificate for TLS client authentication to the KRA.

2. Copy the base-64 encoded transport certificate for the KRA.

   The transport certificate is stored in the KRA's certificate database, which can be retrieved using the **certutil** utility. If the transport certificate is signed by a Certificate Manager, then a copy of the certificate is available through the Certificate Manager end-entities page in the **Retrieval** tab.

   Alternatively, download the transport certificate using the **pki** ultility:

   ```
   # pki cert-find --name "KRA Transport certificate's subject common name"
   # pki cert-show serial_number --output transport.pem
   ```

3. Add the transport certificate to the CA's **CS.cfg** file.

   ```
   ca.connector.KRA.enable=true
   ca.connector.KRA.host=server.example.com
   ca.connector.KRA.local=false
   ca.connector.KRA.nickName=subsystemCert cert-pki-ca
   ca.connector.KRA.port=8443
   ca.connector.KRA.timeout=30
   ca.connector.KRA.transportCert=MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUFADA6
   MRgwFgYDVQQKEw9Eb21haW4gc28gbmFtZWQxHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1d
   ```

```
GhvcmI0eTAeFw0wNjExMTQxODI2NDdaFw0wODEwMTQxNzQwNThaMD4xGDAWBgNVB
AoTD0RvbWFpbiBzbyBuYW1lZDEiMCAGA1UEAxMZRFJNIFRyYW5zcG9ydCBDZXJ0aWZpY
2F0ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKnMGB3WkznueouwZjr
WLFZBLpKt6TimNKV9iz5s0zrGUlpdt81/BTsU5A2sRUwNfoZSMs/d5KLuXOHPyGtmC6yVvaY
719hr9EGYuv0Sw6jb3WnEKHpjbUO/vhFwTufJHWKXFN3V4pMbHTkqW/x5fu/3QyyUre/5IhG0
fcEmfvYxIyvZUJx+aQBW437ATD99Kuh+I+FuYdW+SqYHznHY8BqOdJwJ1JiJMNceXYAuAdk+
9t70RztfAhBmkK0OOP0vH5BZ7RCwE3Y/6ycUdSyPZGGc76a0HrKOz+lwVFulFStiuZIaG1pv0
NNivzcj0hEYq6AfJ3hgxcC1h87LmCxgRWUCAwEAAaN5MHcwHwYDVR0jBBgwFoAURShCYt
Sg+Oh4rrgmLFB/Fg7X3qcwRAYIKwYBBQUHAQEEODA2MDQGCCsGAQUFBzABhihodHR
wOi8vY2x5ZGUucmR1LnJlZGhhdC5jb206OTE4MC9jYS9vY3NwMA4GA1UdDwEB/wQEAwIE
8DANBgkqhkiG9w0BAQUFAAOCAQEAFYz5ibujdIXgnJCbHSPWdKG0T+FmR67YqiOtoNlGyI
gJ42fi5lsDPfCbIAe3YFqmF3wU472h8LDLGyBjy9RJxBj+aCizwHkuoH26KmPGntlayqWDH/UG
sIL0mvTSOeLqI3KM0IuH7bxGXjiION83xWbxumW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWy
YdfEOjpp/UQZOhOM1d8GFiw8N8ClWBGc3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwI
FhC82zqeQ1PbxQDL8QLjzca+IUzq6Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVlskXJw
5ZR0Y2zA==
ca.connector.KRA.uri=/kra/agent/kra/connector
```

4. Then edit the enrollment form and add or replace the transport certificate value in the **keyTransportCert** method.

```
vim /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/ca/ProfileSelect.template

var keyTransportCert =
MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUFADA6MRgwFgYDVQQKEw9Eb21haW4
gc28gbmFtZWQxHjAcBgNVBAMTFUNIcnRpZmljYXRlIEF1dGhvcml0eTAeFw0wNjExMTQxO
DI2NDdaFw0wODEwMTQxNzQwNThaMD4xGDAWBgNVBAoTD0RvbWFpbiBzbyBuYW1lZD
EiMCAGA1UEAxMZRFJNIFRyYW5zcG9ydCBDZXJ0aWZpY2F0ZTCCASIwDQYJKoZIhvcNA
QEBBQADggEPADCCAQoCggEBAKnMGB3WkznueouwZjrWLFZBLpKt6TimNKV9iz5s0zrGUl
pdt81/BTsU5A2sRUwNfoZSMs/d5KLuXOHPyGtmC6yVvaY719hr9EGYuv0Sw6jb3WnEKHpjb
UO/vhFwTufJHWKXFN3V4pMbHTkqW/x5fu/3QyyUre/5IhG0fcEmfvYxIyvZUJx+aQBW437ATD
99Kuh+I+FuYdW+SqYHznHY8BqOdJwJ1JiJMNceXYAuAdk+9t70RztfAhBmkK0OOP0vH5BZ7
RCwE3Y/6ycUdSyPZGGc76a0HrKOz+lwVFulFStiuZIaG1pv0NNivzcj0hEYq6AfJ3hgxcC1h87L
mCxgRWUCAwEAAaN5MHcwHwYDVR0jBBgwFoAURShCYtSg+Oh4rrgmLFB/Fg7X3qcwRA
YIKwYBBQUHAQEEODA2MDQGCCsGAQUFBzABhihodHRwOi8vY2x5ZGUucmR1LnJlZGh
hdC5jb206OTE4MC9jYS9vY3NwMA4GA1UdDwEB/wQEAwIE8DANBgkqhkiG9w0BAQUFAA
OCAQEAFYz5ibujdIXgnJCbHSPWdKG0T+FmR67YqiOtoNlGyIgJ42fi5lsDPfCbIAe3YFqmF3w
U472h8LDLGyBjy9RJxBj+aCizwHkuoH26KmPGntlayqWDH/UGsIL0mvTSOeLqI3KM0IuH7bx
GXjiION83xWbxumW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWyYdfEOjpp/UQZOhOM1d8GFi
w8N8ClWBGc3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwIFhC82zqeQ1PbxQDL8QLjzca
+IUzq6Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVlskXJw5ZR0Y2zA==;
```

12.2. ENCRYPTION OF KRA OPERATIONS

Certificate System encrypts the following key operations in the Key Recovery Authority (KRA):

- Archival:

  - Encryption of keys to archive in a Certificate Request Message Format (CRMF) package for transport to the KRA.

  - Encryption of the key for storage in the KRA LDAP database.

- Recovery:

- ○ Encryption of a user-provided session key for transport to the key.

  - ○ Decryption of the secret and re-encryption using the user provided session key or creation of a PKCS#12 package.

- Generation:

  - ○ Encryption of a generated key for storage.

12.2.1. How Clients Manage Key Operation Encryption

The Certificate System client automatically uses the encryption algorithm set in the KRA's configuration and no further actions are required.

12.2.2. Configuring the Encryption Algorithm in the KRA

> **NOTE**
>
> Only AES CBC (in case when **kra.allowEncDecrypt.archive=true** and **kra.allowEncDecrypt.recovery=true**) and AES Key Wrap (in case when **kra.allowEncDecrypt.archive=false** and **kra.allowEncDecrypt.recovery=false**) are allowed in the following configuration. Any FIPS140-2 validated HSM that supports either algorithm is allowed for the key archival and recovery feature provided by KRA.

Certificate System defines groups of configuration parameters related to key operation encryption in the **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** file. We recommend the following set of parameters (see note above for other option):

```
kra.allowEncDecrypt.archive=false
kra.allowEncDecrypt.recovery=false
kra.storageUnit.wrapping.1.sessionKeyLength=256
kra.storageUnit.wrapping.1.sessionKeyWrapAlgorithm=RSA
kra.storageUnit.wrapping.1.payloadEncryptionPadding=PKCS5Padding
kra.storageUnit.wrapping.1.sessionKeyKeyGenAlgorithm=AES
kra.storageUnit.wrapping.1.payloadEncryptionAlgorithm=AES
kra.storageUnit.wrapping.1.payloadEncryptionMode=CBC
kra.storageUnit.wrapping.1.payloadWrapAlgorithm=AES KeyWrap
kra.storageUnit.wrapping.1.sessionKeyType=AES
kra.storageUnit.wrapping.1.payloadWrapIVLen=16
kra.storageUnit.wrapping.choice=1
```

Each group (**kra.storageUnit.wrapping.0.*** vs **kra.storageUnit.wrapping.1.***) has individual settings, and the number defines which settings belong to the same configuration group. The current configuration group is set in the ***kra.storageUnit.wrapping.choice*** parameter in the **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** file.

Ensure that **kra.storageUnit.wrapping.choice=1** is set in the configuration file before continuing.

> **NOTE**
>
> Certificate System adds the information required to decrypt the data to the record in the KRA database. Therefore, even after changing the encryption algorithm, Certificate System is still able to decrypt data previously stored in the KRA using a different encryption algorithm.

12.2.2.1. Explanation of Parameters and their Values

Each secret (a "payload") is encrypted with a session key. Parameters controlling this encryption are prefixed with **payload**. The set of parameters to be used depends on the value of **kra.allowEncDecrypt.archive** and **kra.allowEncDecrypt.recovery**. By default, both of these are false. See Section 12.2.2.2, "Solving Limitations of HSMs When Using AES Encryption in KRAs" for the effect of these two parameters on HSMs.

When **kra.allowEncDecrypt.archive** and **kra.allowEncDecrypt.recovery** are both false:

- **payloadWrapAlgorithm** determines the wrapping algorithm used. The only one valid choice is **AES KeyWrap**.

- When **payloadWrapAlgorithm=AES/CBC/PKCS5Padding**, then **payloadWrapIVLength=16** has to be specified to tell PKI that we need to generate an IV (as CBC requires one).

When **kra.allowEncDecrypt.archive** and **kra.allowEncDecrypt.recovery** are both true:

- **payloadEncryptionAlgorithm** determines the encryption algorithm used. The only valid choice is *AES*.

- **payloadEncryptionMode** determines the block chaining mode. The only valid choice is *CBC*.

- **payloadEncryptionPadding** determines the padding scheme. The only valid choice is *PKCS5Padding*.

The session key is then wrapped with the KRA Storage Certificate, an RSA token. Parameters controlling the session key and its encryption are prefixed with **sessionKey**.

- **sessionKeyType** is the type of key to generate. The only valid choice is *AES*.

- **sessionKeyLength** is the length of the generated session key. Valid choices are *128* and *256*, to encrypt the payload with 128-bit AES or 256-bit AES respectively.

- **sessionKeyWrapAlgorithm** is the type of key the KRA Storage Certificate is. The only valid choice in this guide is *RSA*.

12.2.2.2. Solving Limitations of HSMs When Using AES Encryption in KRAs

If you run Certificate System with AES enabled in the KRA, but the Hardware Security Module (HSM) does not support the AES key wrapping feature, key archival fails. To solve the problem, the following solutions are supported:

- the section called "Selecting a Different Algorithm for the Key Wrapping"

- Section 7.3.2.3.2, "Setting the KRA into Encryption Mode"

Selecting a Different Algorithm for the Key Wrapping

Sometimes the KRA does not support the default key wrapping algorithm, but it supports other algorithms. For example, to use **AES-128-CBC** as the key wrapping algorithm:

1. Set the following parameters in the **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** file:

   ```
   kra.storageUnit.wrapping.1.payloadWrapAlgorithm=AES KeyWrap
   kra.storageUnit.wrapping.1.payloadWrapIVLen=16
   kra.storageUnit.wrapping.1.sessionKeyLength=128
   ```

2. Restart the instance:

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

   OR

   ```
   # systemctl restart pki-tomcatd-nuxwdog@instance_name.service (if using nuxwdog
   watchdog)
   ```

   If the KRA runs in a difference instance then the CA, you need to restart both instances.

Selecting a different algorithm for the key wrapping has the benefit that if the HSM later adds support for AES key wrapping, you can revert the settings because the key records have the relevant information set.

This configuration uses the **kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen}** and **kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding}** parameters.

Setting the KRA into Encryption Mode

If the HSM does not support any KeyWrap algorithms, on some occassions it is necessary to place the KRA into Encryption Mode. When setting the KRA into encryption mode, all keys will be stored using encryption algorithms rather than key wrapping algorithms.

To set the KRA into encryption mode:

1. Set the following parameters in the **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** file to **true**:

   ```
   kra.allowEncDecrypt.archive=true
   kra.allowEncDecrypt.recovery=true
   ```

2. Restart the service:

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

   OR

   ```
   # systemctl restart pki-tomcatd-nuxwdog@instance_name.service (if using nuxwdog
   watchdog)
   ```

   If the KRA runs in a difference instance than the CA, you need to restart both instances.

This configuration uses **kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding}** and **kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen}** parameters.

> **NOTE**
>
> If you later switch to a different algorithm for the key wrapping according to the section called "Selecting a Different Algorithm for the Key Wrapping", you must manually add the appropriate meta data to records created before you set the KRA into encryption mode.

12.3. SETTING UP AGENT-APPROVED KEY RECOVERY SCHEMES

Key recovery agents collectively authorize and retrieve private encryption keys and associated certificates in a PKCS #12 package. To authorize key recovery, the required number of recovery agents access the KRA agent services page and use the **Authorize Recovery** area to enter each authorization separately.

One of the agents initiates the key recovery process. For a synchronous recovery, each approving agent uses the reference number (which was returned with the initial request) to open the request and then authorizes key recovery separately. For an asynchronous recovery, the approving agents all search for the key recovery request and then authorize the key recovery. Either way, when all of the authorizations are entered, the KRA checks the information. If the information presented is correct, it retrieves the requested key and returns it along with the corresponding certificate in the form of a PKCS #12 package to the agent who initiated the key recovery process.

The *key recovery agent scheme* configures the KRA to recognize to which group the key recovery agents belong and specifies how many of these agents are required to authorize a key recovery request before the archived key is restored.

12.3.1. Configuring Agent-Approved Key Recovery in the Console

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

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

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

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

3. Enter the number of agents to use to approve key recover in the **Required Number of Agents** field.
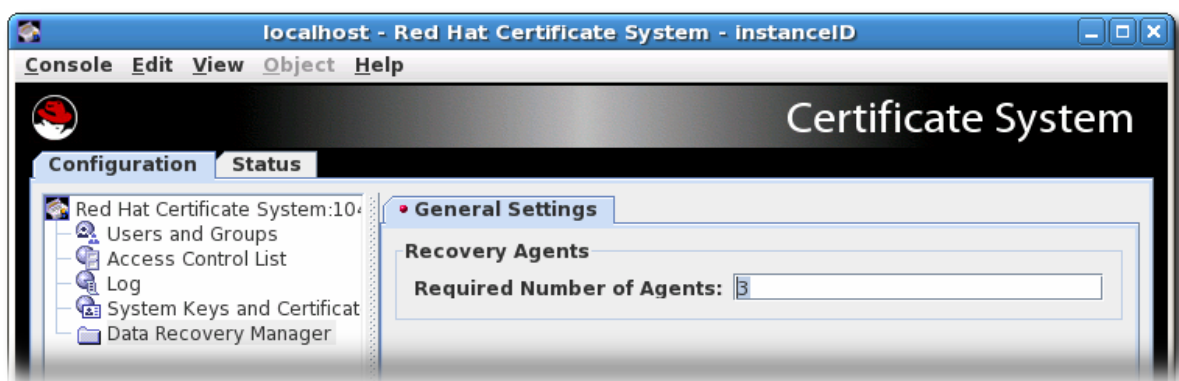
12.3.2. Configuring Agent–Approved Key Recovery in the Command Line

To set up agent–initiated key recovery, edit two parameters in the KRA configuration:

- Set the number of recovery managers to require to approve a recovery.

- Set the group to which these users must belong.

These parameters are set in the KRA's **CS.cfg** configuration file.

1. Stop the server before editing the configuration file.

   systemctl stop pki-tomcatd@*instance_name*.service

   OR

   systemctl stop pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

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

   vim /var/lib/pki/pki-tomcat/kra/conf/CS.cfg

3. Edit the two recovery scheme parameters.

   kra.noOfRequiredRecoveryAgents=3
   kra.recoveryAgentGroup=Key Recovery Authority Agents

4. Restart the server.

   systemctl start pki-tomcatd@*instance_name*.service

   OR

   systemctl start pki-tomcatd-nuxwdog@*instance_name*.service

12.3.3. Customizing the Key Recovery Form

The default key agent scheme requires a single agent from the Key Recovery Authority Agents group to be in charge of authorizing key recovery.

It is also possible to customize the appearance of the key recovery form. Key recovery agents need an appropriate page to initiate the key recovery process. By default, the KRA's agent services page includes the appropriate HTML form to allow key recovery agents to initiate key recovery, authorize key recovery requests, and retrieve the encryption keys. This form is located in the **/var/lib/pki/pki-tomcat/kra/webapps/kra/agent/kra/** directory, called **confirmRecover.html**.
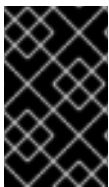
> **IMPORTANT**
>
> If the key recovery confirmation form is customized, do not to delete any of the information for generating the response. This is vital to the functioning of the form. Restrict any changes to the content in and appearance of the form.

CHAPTER 13. CONFIGURING 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.

For details about log administration after the installation, see the *Configuring Subsystem Logs* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

For an overview on logs, see Section 2.3.15, "Logs".

13.1. CERTIFICATE SYSTEM LOG SETTINGS

The way that logs are configured can affect Certificate System performance. For example, log file rotation keeps logs from becoming too large, which slows down subsystem performance. This section explains the different kinds of logs recorded by Certificate System subsystems and covers important concepts such as log file rotation, buffered logging, and available log levels.

13.1.1. Services That Are Logged

All major components and protocols of Certificate System log messages to log files. Table 13.1, "Services Logged" lists services that are logged by default. To view messages logged by a specific service, customize log settings accordingly.

Table 13.1. Services Logged

Service	Description
ACLs	Logs events related to access control lists.
Administration	Logs events related to administration activities, such as HTTPS communication between the Console and the instance.
All	Logs events related to all the services.
Authentication	Logs events related to activity with the authentication module.
Certificate Authority	Logs events related to the Certificate Manager.
Database	Logs events related to activity with the internal database.
HTTP	Logs events related to the HTTP activity of the server. Note that HTTP events are actually logged to the errors log belonging to the Apache server incorporated with the Certificate System to provide HTTP services.
Key Recovery Authority	Logs events related to the KRA.
LDAP	Logs events related to activity with the LDAP directory, which is used for publishing certificates and CRLs.

Service	Description
OCSP	Logs events related to OCSP, such as OCSP status GET requests.
Others	Logs events related to other activities, such as command-line utilities and other processes.
Request Queue	Logs events related to the request queue activity.
User and Group	Logs events related to users and groups of the instance.

13.1.2. Log Levels (Message Categories)

The different events logged by Certificate System services are determined by the log levels, which makes identifying and filtering events simpler. The different Certificate System log levels are listed in Table 13.2, "Log Levels and Corresponding Log Messages" .

Log levels are represented by numbers **0** to **10**, each number indicating the level of logging to be performed by the server. The level sets how detailed the logging should be.

A higher priority level means less detail because only events of high priority are logged.

> **NOTE**
>
> The default log level is **1** and this value should not be changed. To enable debug logging, see Section 13.3.3, "Additional Configuration for Debug Log" .
>
> Table 13.2, "Log Levels and Corresponding Log Messages" is provided for reference to better understand log messages.

Table 13.2. Log Levels and Corresponding Log Messages

Log level	Message category	Description
0	Debugging	These messages contain debugging information. This level is not recommended for regular use because it generates too much information.
1	All logging levels	This log level enables all logging levels.
2	Warning	These messages are warnings only and do not indicate any failure in the normal operation of the server.

Log level	Message category	Description
3	Failure; the default selection for system and error logs	These messages indicate errors and failures that prevent the server from operating normally, including failures to perform a certificate service operation (*User authentication failed* or *Certificate revoked*) and unexpected situations that can cause irrevocable errors (*The server cannot send back the request it processed for a client through the same channel the request came from the client*).
4	Misconfiguration	These messages indicate that a misconfiguration in the server is causing an error.
5	Catastrophic failure	These messages indicate that, because of an error, the service cannot continue running.
6	Security-related events	These messages identify occurrences that affect the security of the server. For example, **Privileged access attempted by user with revoked or unlisted certificate**.
7	PDU-related events (debugging)	These messages contain debugging information for PDU events. This level is not recommended for regular use because it generates more information than is normally useful.
8	PDU-related events	These messages relate transactions and rules processed on a PDU, such as creating MAC tokens.
9	PDU-related events	This log level provides verbose log messages for events processed on a PDU, such as creating MAC tokens.
10	Informational (default selection for audit log)	These messages provide general information about the state of the Certificate System, including status messages such as *Certificate System initialization complete* and *Request for operation succeeded.*

Log levels can be used to filter log entries based on the severity of an event. By default, log level 3 (Failure) is set for all services.

The log level is successive; specifying a value of 3 causes levels 4, 5, and 6 to be logged. Log data can be extensive, especially at lower (more verbose) logging levels. Make sure that the host machine has sufficient disk space for all the log files. It is also important to define the logging level, log rotation, and server-backup policies appropriately so that all the log files are backed up and the host system does not get overloaded; otherwise, information can be lost.

13.1.3. Buffered and Unbuffered Logging

The Java subsystems support buffered logging for all types of logs. The server can be configured for either buffered or unbuffered logging.

If buffered logging is configured, the server creates buffers for the corresponding logs and holds the messages in the buffers for as long as possible. The server flushes out the messages to the log files only when one of the following conditions occurs:

- The buffer gets full. The buffer is full when the buffer size is equal to or greater than the value specified by the **bufferSize** configuration parameter. The default value for this parameter is 512 KB.

- The flush interval for the buffer is reached. The flush interval is reached when the time interval since the last buffer flush is equal to or greater than the value specified by the **flushInterval** configuration parameter. The default value for this parameter is 5 seconds.

- When current logs are read from Console. The server retrieves the latest log when it is queried for current logs.

If the server is configured for unbuffered logging, the server flushes out messages as they are generated to the log files. Because the server performs an I/O operation (writing to the log file) each time a message is generated, configuring the server for unbuffered logging decreases performance.

Setting log parameters is described in the *Configuring Logs in the Console* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

13.1.4. Log File Rotation

The subsystem logs have an optional log setting that allows them to be rotated and start a new log file instead of letting log files grow indefinitely. Log files are rotated when either of the following occur:

- The size limit for the corresponding file is reached. The size of the corresponding log file is equal to or greater than the value specified by the **maxFileSize** configuration parameter. The default value for this parameter is 100 KB.

- The age limit for the corresponding file is reached. The corresponding log file is equal to or older than the interval specified by the **rolloverInterval** configuration parameter. The default value for this parameter is 2592000 seconds (every thirty days).

When a log file is rotated, the old file is named using the name of the file with an appended time stamp. The appended time stamp is an integer that indicates the date and time the corresponding active log file was rotated. The date and time have the forms YYYYMMDD (year, month, day) and HHMMSS (hour, minute, second).

Log files, especially the audit log file, contain critical information. These files should be periodically archived to some backup medium by copying the entire **log** directory to an archive medium.

> **NOTE**
>
> The Certificate System does not provide any tool or utility for archiving log files.

The Certificate System provides a command-line utility, **signtool**, that signs log files before archiving them as a means of tamper detection.

Signing log files is an alternative to the signed audit logs feature. Signed audit logs create audit logs that are automatically signed with a subsystem signing certificate.

Rotated log files are not deleted.

13.2. OPERATING SYSTEM (EXTERNAL TO RHCS) LOG SETTINGS

13.2.1. Enabling OS-level Audit Logs

> **WARNING**
>
> All operations in the following sections have to be performed as root or a privileged user via **sudo**.

The **auditd** logging framework provides many additional audit capabilities. These OS-level audit logs complement functionality provided by Certificate System directly. Before performing any of the following steps in this section, make sure the **audit** package is installed:

```
# sudo yum install audit
```

Auditing of system package updates (using **yum** and **rpm** and including Certificate System) is automatically performed and requires no additional configuration.

> **NOTE**
>
> After adding each audit rule and restarting the **auditd** service, validate the new rules were added by running:
>
> ```
> # auditctl -l
> ```
>
> The contents of the new rules should be visible in the output.

For instructions on viewing the resulting audit logs, see the *Displaying Operating System-level Audit Logs* in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

13.2.1.1. Auditing Certificate System Audit Log Deletion

To receive audit events for when audit logs are deleted, you need to audit system calls whose targets are Certificate System logs.

Create the file **/etc/audit/rules.d/rhcs-audit-log-deletion.rules** with the following contents:

```
-a always,exit -F arch=b32 -S unlink -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S rename -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S rmdir -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S unlinkat -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S renameat -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S unlink -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S rename -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S rmdir -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S unlinkat -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S renameat -F dir=/var/log/pki -F key=rhcs_audit_deletion
```

Then restart **auditd**:

```
# service auditd restart
```

13.2.1.2. Auditing Unauthorized Certificate System Use of Secret Keys

To receive audit events for all access to Certificate System Secret or Private keys, you need to audit the file system access to the NSS DB.

Create the **/etc/audit/rules.d/rhcs-audit-nssdb-access.rules** file with the following contents:

```
-w /etc/pki/<instance name>/alias -p warx -k rhcs_audit_nssdb
```

<instance name> is the name of the current instance. For each file (`<file>`) in **/etc/pki/<instance name>/alias**, add to **/etc/audit/rules.d/rhcs-audit-nssdb-access.rules** the following line :

```
-w /etc/pki/<instance name>/alias/<file> -p warx -k rhcs_audit_nssdb
```

For example, if the instance name is **pki-ca121318ec** and **cert8.db**, **key3.db**, **NHSM6000-OCScert8.db**, **NHSM6000-OCSkey3.db**, and **secmod.db** are files, then the configuration file would contain:

```
-w /etc/pki/pki-ca121318ec/alias -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/cert8.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/key3.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/NHSM6000-OCScert8.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/NHSM6000-OCSkey3.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/secmod.db -p warx -k rhcs_audit_nssdb
```

Then restart **auditd**:

```
# service auditd restart
```

13.2.1.3. Auditing Time Change Events

To receive audit events for time changes, you need to audit a system call access which could modify the system time.

Create the **/etc/audit/rules.d/rhcs-audit-rhcs_audit_time_change.rules** file with the following contents:

```
-a always,exit -F arch=b32 -S adjtimex,settimeofday,stime -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S adjtimex,settimeofday -F key=rhcs_audit_time_change
-a always,exit -F arch=b32 -S clock_settime -F a0=0x0 -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S clock_settime -F a0=0x0 -F key=rhcs_audit_time_change
-a always,exit -F arch=b32 -S clock_adjtime -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S clock_adjtime -F key=rhcs_audit_time_change
-w /etc/localtime -p wa -k rhcs_audit_time_change
```

Then restart **auditd**:

```
# service auditd restart
```

For instructions on how to set time, see Chapter 15. Setting Time and Date in Red Hat Enterprise Linux 7.6 in Red Hat Certificate System's Administration Guide.

13.2.1.4. Auditing Access to Certificate System Configuration

To receive audit events for all modifications to the Certificate System instance configuration files, audit the file system access for these files.

Create the **/etc/audit/rules.d/rhcs-audit-config-access.rules** file with the following contents:

```
-w /etc/pki/instance_name/server.xml -p wax -k rhcs_audit_config
```

Additionally, add for each subsystem in the **/etc/pki/instance_name/** directory the following contents:

```
-w /etc/pki/instance_name/subsystem/CS.cfg -p wax -k rhcs_audit_config
```

Example 13.1. rhcs-audit-config-access.rules Configuration File

For example, if the instance name is **pki-ca121318ec** and only a CA is installed, the **/etc/audit/rules.d/rhcs-audit-config-access.rules** file would contain:

```
-w /etc/pki/pki-ca121318ec/server.xml -p wax -k rhcs_audit_config
-w /etc/pki/pki-ca121318ec/ca/CS.cfg -p wax -k rhcs_audit_config
```

Note that access to the PKI NSS database is already audited under **rhcs_audit_nssdb**.

13.3. CONFIGURING LOGS IN THE CS.CFG FILE

During the installation configuration, logging can be configured by directly editing the **CS.cfg** for the instance.

1. Stop the subsystem instance.

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

2. Open the **CS.cfg** file in the **/var/lib/pki/instance_name/subsystem_name/conf/** directory.

3. To create a new log, copy all of the entries for either the system or transactions log. These are the parameters that begin with **log.instance.Transactions** or **log.instance.System**. Paste all entries at the bottom of the logging section and change the name of this instance by changing the word **Transactions** or **System** in each parameter to the new name.

4. To configure a log instance, modify the parameters associated with that log. These parameters begin with **log.instance**.

    Table 13.3. Log Entry Parameters

Parameter	Description
type	The type of log file. **system** creates error and system logs; **transaction** records audit logs.

Parameter	Description
enable	Sets whether the log is active. Only enabled logs actually record events.
level	Sets the log level in the text field. The level must be manually entered in the field; there is no selection menu. The log level setting is a numeric value, as listed in Section 13.1.2, "Log Levels (Message Categories)".
fileName	The full path, including the file name, to the log file. The subsystem user should have read/write permission to the file.
bufferSize	The buffer size in kilobytes (KB) for the log. Once the buffer reaches this size, the contents of the buffer are flushed out and copied to the log file. The default size is 512 KB. For more information on buffered logging, see Section 13.1.3, "Buffered and Unbuffered Logging".
flushInterval	The amount of time, in seconds, before the contents of the buffer are flushed out and added to the log file. The default interval is 5 seconds.
maxFileSize	The size, kilobytes (KB), a log file can become before it is rotated. Once it reaches this size, the file is copied to a rotated file, and the log file is started new. For more information on log file rotation, see Section 13.1.4, "Log File Rotation". The default size is 2000 KB.
rolloverInterval	The frequency which the server rotates the active log file. The available choices are hourly, daily, weekly, monthly, and yearly. The default selection is monthly. For more information, see Section 13.1.4, "Log File Rotation".
register[a]	If this variable is set to **false** (the default value), the self-test messages are only logged to the log file specified by **selftests.container.logger.fileName**. If this variable is set to **true**, then the self-test messages are written to both the log file specified by **selftests.container.logger.fileName** as well as to the log file specified by **log.** *instance* **.Transactions.** *fileName*.
logSigning[b]	Enables signed logging. When this parameter is enabled, provide a value for the **signedAuditCertNickname** parameter. This option means the log can only be viewed by an auditor. The value is either **true** or **false**.
signedAuditCertNickname[b]	The nickname of the certificate used to sign audit logs. The private key for this certificate must be accessible to the subsystem in order for it to sign the log.
events[b]	Specifies which events are logged to the audit log. Log events are separated by commas with no spaces.

[a] This is for self-test logs only.

[b] This is for audit logs only.

5. Save the file.

6. Start the subsystem instance.

> systemctl start pki-tomcatd@*instance_name.service*

OR

> systemctl start pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

13.3.1. Enabling and Configuring Signed Audit Log

13.3.1.1. Enabling Signed Audit Logging

By default, audit logging is enabled upon installation. However, log signing needs to be enabled manually after installation.

To display the current audit logging configuration:

```
# pki-server subsystem-audit-config-show
  Enabled: True
  Log File: audit_signing_log_file
  Buffer Size (bytes): 512
  Flush Interval (seconds): 5
  Max File Size (bytes): 2000
  Rollover Interval (seconds): 2592000
  Expiration Time (seconds): 0
  Log Signing: False
  Signing Certificate: audit_signing_certificate
```

To enable signed audit logging:

1. Use the **pki-server** utility to set the **--logSigning** option to **true**:

   ```
   # pki-server subsystem-audit-config-mod --logSigning True
     ...
     Log Signing: True
     ...
   ```

2. Restart the instance:

   > # systemctl restart pki-tomcatd@*instance_name*.service

   OR

   > systemctl start pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

13.3.1.2. Configuring Audit Events

13.3.1.2.1. Enabling and Disabling Audit Events

For details about enabling and disabling audit events, see the *Configuring a Signed Audit Log in the Console* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

> **IMPORTANT**
>
> All the audit events required by the Common Criteria specification, are enabled by default. For the list of required audit events, see Appendix E of Red Hat Certificate System's Administration Guide.

In addition, audit event filters can be set to finer grained selection. See Section 13.3.1.2.2, "Filtering Audit Events".

For a complete list of auditable events in Certificate System, see the *Audit Events* appendix in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

13.3.1.2.2. Filtering Audit Events

In Certificate System administrators can set filters to configure which audit events will be logged in the audit file based on the event attributes.

The format of the filters is the same as for LDAP filters. However, Certificate System only supports the following filters:

Table 13.4. Supported Audit Event Filters

Type	Format	Example
Presence	(*attribute*=*)	(ReqID=*)
Equality	(*attribute*=*value*)	(Outcome=Failure)
Substring	(*attribute*=*initial*any*...*any*final*)	(SubjectID=*admin*)
AND operation	(&(*filter_1*)(*filter_2*)...(*filter_n*))	(&(SubjectID=admin)(Outcome=Failure))
OR operation	(|(*filter_1*)(*filter_2*)...(*filter_n*))	(|(SubjectID=admin)(Outcome=Failure))
NOT operation	(!(*filter*))	(!(SubjectID=admin))

For further details on LDAP filters, see the *Using Compound Search Filters* section in the *Red Hat Directory Server Administration Guide*.

> **Example 13.2. Filtering Audit Events**
>
> To display the current settings for profile certificate requests and processed certificates:
>
> ```
> $ pki-server ca-audit-event-show PROFILE_CERT_REQUEST
>   Event Name: PROFILE_CERT_REQUEST
>   Enabled: True
>   Filter: None
>
> $ pki-server ca-audit-event-show CERT_REQUEST_PROCESSED
> ```

> Event Name: CERT_REQUEST_PROCESSED
> Enabled: True
> Filter: None

To log only failed events for profile certificate requests and events for processed certificate requests that have the **InfoName** field set to **rejectReason** or **cancelReason**:

1. Execute the following commands:

   ```
   $ pki-server ca-audit-event-update PROFILE_CERT_REQUEST --filter "
   (Outcome=Failure)"

     ...
     Filter: (Outcome=Failure)

   $ pki-server ca-audit-event-update CERT_REQUEST_PROCESSED --filter "(|
   (InfoName=rejectReason)(InfoName=cancelReason))"

     ...
     Filter: (|(InfoName=rejectReason)(InfoName=cancelReason))
   ```

2. Restart Certificate System:

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

   OR

   ```
   systemctl start pki-tomcatd-nuxwdog@instance_name.service (if using nuxwdog
   watchdog)
   ```

13.3.2. Configuring Self-Tests

The self-tests feature and individual self-tests are registered and configured in the **CS.cfg** file. If a self-test is enabled, that self-test is listed for either on-demand or start up and is either empty or set as **critical**.

Critical self-tests have a colon and the word **critical** after the name of the self-test. Otherwise, nothing is in this place. The server shuts down when a critical self-test fails during on demand self-tests; the server will not start when a critical self-test fails during start up.

The implemented self-tests are automatically registered and configured when the instance was installed. The self-tests that are registered and configured are those associated with the subsystem type.

A self-test's criticality is changed by changing the respective settings in the **CS.cfg** file.

13.3.2.1. Default Self-Tests at Startup

The following self-tests are enabled by default at startup.

For the CA subsystem, the following self-tests are enabled by default at startup:

- **CAPresence** - used to verify the presence of the CA subsystem.

- **CAValidity** - used to determine that the CA subsystem is currently valid and has not expired. This involves checking the expiration of the CA certificate.

- **SystemCertsVerification** – used to determine that the system certificates are currently valid and have not expired or been revoked. For the CA subsystem, only validity tests for each certificate are done, leaving out certificate verification tests which could result in an OCSP request. This behavior can be overridden with the following config parameter:

    selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true

    By default, this config parameter is considered **false** if not present at all.

For the KRA subsystem, the following-self-tests are enabled:

- **KRAPresence** – used to verify the presence of the KRA subsystem.

- **KRAValidity** – used to determine that the KRA subsystem is currently valid and has not expired. This involves checking the expiration of the KRA certificate.

- **SystemCertsVerification** – used to determine that the system certificates are currently valid and have not expired or been revoked.

For the OCSP subsystem, the following self-tests are enabled:

- **OCSPPresence** – used to verify the presence of the OCSP subsystem.

- **OCSPValidity** – used to determine that the OCSP subsystem is currently valid and has not expired. This involves checking the expiration of the OCSP certificate.

- **SystemCertsVerification** – used to determine that the system certificates are currently valid and have not expired or been revoked. For the OCSP subsystem, only validity tests for each certificate are done, leaving out certificate verification tests which could result in an OCSP request. This behavior can be overridden with the following config parameter:

    selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true

    By default, this config parameter is considered **false** if not present at all.

For the TKS subsystem, the following-self-tests are enabled:

- **TKSKnownSessionKey** – used to verify the known session key of a TKS subsystem. This verifies the TKS is able to properly assist in creating keys on behalf of the TPS and supported smart cards.

- **SystemCertsVerification** – used to determine that the system certificates are currently valid and have not expired or been revoked.

For the TPS subsystem, the following-self-tests are enabled:

- **TPSPresence** – used to verify the presence of the TPS subsystem.

- **TPSValidity** – used to determine that the TPS subsystem is currently valid and has not expired. This involves checking the expiration of the TPS certificate.

- **SystemCertsVerification** – used to determine that the system certificates are currently valid and have not expired or been revoked.

13.3.2.2. Modifying Self-Test Configuration

By default, the self-test configuration is compliant. However, some settings can change the visibility of self-test logging or improve performance. To modify the configuration settings for self-tests:

1. Stop the subsystem instance.

2. Open the **CS.cfg** file located in the instance's **conf/** directory.

3. To edit the settings for the self-test log, edit the entries that begin with **selftests.container.logger**. Unless otherwise specified, these parameters do not affect compliance. These include the following parameters:

   - **bufferSize** – Specify the buffer size in kilobytes (KB) for the log. The default size is 512 KB. Once the buffer reaches this size, the contents of the buffer are flushed out and copied to the log file.

   - **enable** – Specify **true** to enable. Only enabled logs actually record events. This value *must* be enabled for compliance.

   - **fileName** – Specify the full path, including the filename, to the file to write messages. The server must have read/write permission to the file. By default, the self-test log file is **/var/lib/pki/***instance_name***/logs/***subsystem_type***/selftests.log**

   - **flushInterval** – Specify the interval, in seconds, to flush the buffer to the file. The default interval is 5 seconds. The **flushInterval** is the amount of time before the contents of the buffer are flushed out and added to the log file.

   - **level** – The default selection is 1; this log is not set up for any level beside 1.

   - **maxFileSize** – Specify the file size in kilobytes (KB) for the error log. The default size is 100 KB. The **maxFileSize** determines how large a log file can become before it is rotated. Once it reaches this size, the file is copied to a rotated file, and a new log file is started.

   - **register** – If this variable is set to **false** (the default value), the self-test messages are only logged to the log file specified by **selftests.container.logger.fileName**. If this variable is set to **true**, then the self-test messages are written to both the log file specified by **selftests.container.logger.fileName** and the log file specified by **log.instance.Transactions.fileName**.

   - **rolloverInterval** – Specify the frequency at which the server rotates the active error log file. The choices are hourly, daily, weekly, monthly, and yearly. The default selection is monthly.

   - **type** – Set to **transaction**; do not change this.

4. To edit the order in which the self-test are run, specify the order by listing any of the self-test as the value of the following parameters separated by a comma and a space.

   To mark a self-test critical, add a colon and the word critical to the name of the self-test in the list.

   To disable a self-test, remove it as the value of either the **selftests.container.order.onDemand** or **selftests.container.order.startup** parameters.

5. Save the file.

6. Start the subsystem.

13.3.3. Additional Configuration for Debug Log

13.3.3.1. Enabling and Disabling Debug Logging

By default, debug logging is enabled in Certificate System. However, in certain situations, Administrators want to disable or re-enable this feature:

1. Edit the **/var/lib/pki/***instance_name***/***subsystem***/conf/CS.cfg** file and set the ***debug.enabled*** parameter:

   - To disable debug logging, set:

     ```
     debug.enabled=false
     ```

     > **NOTE**
     >
     > Debug logs are *not* part of audit logging. Debug logs are helpful when trying to debug specific failures in Certificate System or a failing installation.
     >
     > By default, debug logs are enabled. If it is not desired, the administrator can safely disable debug logging to turn down verbosity.

   - To enable debug logging, set:

     ```
     debug.enabled=true
     ```

2. Restart the Certificate System instance:

   ```
   # systemctl restart pki-tomcatd@instance-name.service
   ```

   OR

   ```
   # systemctl restart pki-tomcatd-nuxwdog@instance-name.service (if using nuxwdog watchdog)
   ```

13.3.3.2. Setting up Rotation of Debug Log Files

Certificate System is not able to rotate debug logs. Debug logging is enabled by default and these logs grow until the file system is full. Use an external utility, such as **logrotate**, to rotate the logs.

> **Example 13.3. Using logrotate to Rotate Debug Logs**
>
> Create a configuration file, such as **/etc/logrotate.d/***rhcs_debug* with the following content:
>
> ```
> /var/log/pki/instance_name/subsystem/debug {
>     copytruncate
>     weekly
>     rotate 5
>     notifempty
>     missingok
> }
> ```
>
> To rotate debug logs for multiple subsystems in one configuration file, list the paths to the logs, each separated by white space, before the opening curly bracket. For example:

```
/var/log/pki/instance_name/ca/debug /var/log/pki/instance_name/kra/debug {
    ...
}
```

For further details about **logrotate** and the parameters used in the example, see the logrotate(8) man page.

13.4. AUDIT RETENTION

Audit data are required to be retained in a way according to their retention categories:

- Extended Audit Retention: Audit data that is retained for necessary maintenance for a certificate's lifetime (from issuance to its expiration or revocation date). In Certificate System, they appear in the following areas:

  - Signed audit logs: All events defined in Appendix E. Audit Events of Red Hat Certificate System's Administration Guide.

  - In the CA's internal LDAP server, certificate request records received by the CA and the certificate records as the requests are approved.

- Normal Audit Retention: Audit data that is typically retained only to support normal operation. This includes all events that do not fall under the *extended audit retention* category.

> **NOTE**
>
> Certificate System does not provide any interface to modify or delete audit data.

13.4.1. Location of Audit Data

This section explains where Certificate System stores audit data and where to find the expiration date which plays a crucial role to determine the retention category.

13.4.1.1. Location of Audit Logs

Certificate System stores audit logs in the **/var/lib/pki/***instance_name***/***subsystem_type***/logs/signedAudit/** directory. For example, the audit logs of a CA are stored in the **/var/lib/pki/***instance_name***/ca/logs/signedAudit/** directory. Normal users cannot access files in this directory. See

For a list of audit log events that need to follow the extended audit retention period, see *Audit Events* appendix in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)* ..

> **IMPORTANT**
>
> Do not delete any audit logs that contain any events listed in for certificate requests or certificates that have not yet expired.
>
> These audit logs will consume storage space potentially up to all space available in the disk partition.

13.4.1.2. Location of Certificate Requests and Certificate Records

When certificate signing requests (CSR) are submitted, the CA stores the CSRs in the request repository provided by the CA's internal directory server. When these requests are approved, each certificate issued successfully, will result in an LDAP record being created in the certificate repository by the same internal directory server.

The CA's internal directory server was specified in the following parameters when the CA was created using the **pkispawn** utility:

- *pki_ds_hostname*

- *pki_ds_ldap_port*

- *pki_ds_database*

- *pki_ds_base_dn*

If a certificate request has been approved successfully, the validity of the certificate can be viewed by accessing the CA EE portal either by request ID or by serial number.

To display the validity for a certificate request record:

1. Log into the CA EE portal under **https://*host_name*:*port*/ca/ee/ca/**.

2. Click **Check Request Status**.

3. Enter the Request Identifier.

4. Click **Issued Certificate**.

5. Search for **Validity**.

To display the validity from a certificate record:

1. Log into the CA EE portal under **https://*host_name*:*port*/ca/ee/ca/**.

2. Enter the serial number range. If you search for one specific record, enter the record's serial number in both the lowest and highest serial number field.

3. Click on the search result.

4. Search for **Validity**.

> **IMPORTANT**
>
> Do not delete the request of the certificate records of the certificates that have not yet expired.

CHAPTER 14. CREATING A ROLE USER

As explained in Section 2.6.6.1, "Default Administrative Roles", a bootstrap user was created during the installation. After the installation, create real users and assign them proper system privileges. For compliance, each user must be a member of only one role (group).

This chapter instructs you how to:

- Create a Certificate System administrative user on the operating system

- Create a PKI role in Certificate System

14.1. CREATING A PKI ADMINISTRATIVE USER ON THE OPERATING SYSTEM

This section is for administrative role users. Agent and Auditor role users, see Section 14.2, "Creating a PKI Role User in Certificate System".

In general, administrators, agents, and auditors in Certificate System can manage the Certificate System instance remotely using client applications, such as command-line utilities, the Java Console, and browsers. For the majority of CS management tasks, a Certificate System role user does not need to log on to the host machine where the instance runs. For example, an auditor role user is allowed to retrieve signed audit logs remotely for verification, and an agent role user can use the agent interface to approve a certificate issuance, while an administrator role user can use command-line utilities to configure a profile.

In certain cases, however, a Certificate System administrator requires to log in to the host system to modify configuration files directly, or to start or stop a Certificate System instance. Therefore, on the operating system, the administrator role user should be someone who is allowed to make changes to the configuration files and read various logs associated with Red Hat Certificate System.

> **NOTE**
>
> Do not allow the Certificate System administrators or anyone other than the auditors to access the audit log files.

1. Create the **pkiadmin** group on the operating system.

   ```
   # groupadd -r pkiadmin
   ```

2. Add the **pkiuser** to the **pkiadmin** group:

   ```
   # usermod -a -G pkiadmin pkiuser
   ```

3. Create a user on the operating system. For example, to create the **jsmith** account:

   ```
   # useradd -g pkiadmin -d /home/jsmith -s /bin/bash -c "Red Hat Certificate System
   Administrator John Smith" -m jsmith
   ```

   For details, see the useradd(8) man page.

4. Add the user **jsmith** to the **pkiadmin** group:

```
# usermod -a -G pkiadmin jsmith
```

For details, see the usermod(8) man page.

If you are using a nCipher hardware security module (HSM), add the user who manages the HSM device to the **nfast** group:

```
# usermod -a -G nfast pkiuser
# usermod -a -G nfast jsmith
```

5. Add proper **sudo** rules to allow the **pkiadmin** group to Certificate System and other system services.

   For both simplicity of administration and security, the Certificate System and Directory Server processes can be configured so that PKI administrators (instead of only root) can start and stop the services.

   A recommended option when setting up subsystems is to use a **pkiadmin** system group. (Details are Section 6.7, "Certificate System Operating System Users and Groups"). All of the operating system users which will be Certificate System administrators are then added to this group. If this **pkiadmin** system group exists, then it can be granted sudo access to perform certain tasks.

   1. Edit the /**etc**/**sudoers** file; on Red Hat Enterprise Linux, this can be done using the **visudo** command:

      ```
      # visudo
      ```

   2. Depending on what is installed on the machine, add a line for the Directory Server, the Administration Server, PKI management tools, and each PKI subsystem instance, granting **sudo** rights to the **pkiadmin** group:

      ```
      # For Directory Server services
      %pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv.target
      %pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv-admin.service

      # For PKI instance management
      %pkiadmin ALL = PASSWD: /usr/sbin/pkispawn *
      %pkiadmin ALL = PASSWD: /usr/sbin/pkidestroy *

      # For PKI instance services
      %pkiadmin ALL = PASSWD: /usr/bin/systemctl * pki-tomcatd@instance_name.service
      ```
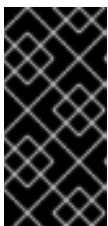
   IMPORTANT

   Make sure to set sudo permissions for every Certificate System, Directory Server, and Administration Server on the machine — and *only* for those instances on the machine. There could be multiple instances of the same subsystem type on a machine or no instance of a subsystem type. It depends on the deployment.

6. Set the group on the following files to **pkiadmin**:

   ```
   # chgrp pkiadmin /etc/pki/instance_name/server.xml
   ```

```
# chgrp -R pkiadmin /etc/pki/instance_name/alias
# chgrp pkiadmin /etc/pki/instance_name/subsystem/CS.cfg
# chgrp pkiadmin /var/log/pki/instance_name/subsystem/debug
```

After creating the administrative user on the operating system, follow Section 14.2, "Creating a PKI Role User in Certificate System".

14.2. CREATING A PKI ROLE USER IN CERTIFICATE SYSTEM

To create a PKI role user, see the *Managing Certificate System Users and Groups* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

CHAPTER 15. DELETING THE BOOTSTRAP USER
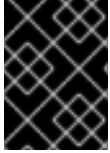
> **IMPORTANT**
>
> Before you delete the bootstrap user, create a real PKI administrative user as described in
> Chapter 14, *Creating a Role User* .

To delete the bootstrap user, follow the procedure described in the *Deleting a Certificate System User* section in the *Red Hat Certificate System Administration Guide (Common Criteria Edition)*.

15.1. DISABLING MULTI-ROLES SUPPORT

By the default, users can belong to more than one subsystem group at once, allowing the user to act as more than one role. For example, John Smith could belong to both an agent and an administrator group. However, for highly secure environments, the subsystem roles should be restricted so that a user can only belong to one role. This can be done by disabling the **multirole** attribute in the instance's configuration.

For all subsystems:

1. Stop the server:

   systemctl stop pki-tomcatd@*instance_name*.service

   OR

   systemctl stop pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

2. Open the **CS.cfg** file:

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

3. Change the **multiroles.enable** parameter value from **true** to **false**.

4. Add or edit the list of default roles in Certificate System that are affected by the multi-roles setting. If multi-roles is disabled and a user belongs to *one* of the roles listed in the **multiroles.false.groupEnforceList** parameter, then the user cannot be added to any group for any of the other roles in the list.

   multiroles.false.groupEnforceList=Administrators,Auditors,Trusted Managers,Certificate Manager Agents,Registration Manager Agents,Key Recovery Authority Agents,Online Certificate Status Manager Agents,Token Key Service Manager Agents,Enterprise CA Administrators,Enterprise KRA Adminstrators,Enterprise OCSP Administrators,Enterprise TKS Administrators,Enterprise TPS Administrators,Security Domain Administrators,Subsystem Group

5. Restart the server:

   systemctl start pki-tomcatd@*instance_name*.service

   OR

systemctl start pki-tomcatd-nuxwdog@*instance_name*.service (if using nuxwdog watchdog)

PART IV. UNINSTALLING CERTIFICATE SYSTEM SUBSYSTEMS

It is possible to remove individual subsystems or to uninstall all packages associated with an entire subsystem. Subsystems are installed and uninstalled individually. For example, it is possible to uninstall a KRA subsystem while leaving an installed and configured CA subsystem. It is also possible to remove a single CA subsystem while leaving other CA subsystems on the machine.

CHAPTER 16. REMOVING A SUBSYSTEM

Removing a subsystem requires specifying the subsystem type and the name of the server in which the subsystem is running. This command removes all files associated with the subsystem (without removing the subsystem packages).

> pkidestroy -s *subsystem_type* -i *instance_name*

The **-s** option specifies the subsystem to be removed (such as CA, KRA, OCSP, TKS, or TPS). The **-i** option specifies the instance name, such as **pki-tomcat**.

> **Example 16.1. Removing a CA Subsystem**
>
> ```
> $ pkidestroy -s CA -i pki-tomcat
> Loading deployment configuration from /var/lib/pki/pki-tomcat/ca/registry/ca/deployment.cfg.
> Uninstalling CA from /var/lib/pki/pki-tomcat.
> Removed symlink /etc/systemd/system/multi-user.target.wants/pki-tomcatd.target.
>
> Uninstallation complete.
> ```

The **pkidestroy** utility removes the subsystem and any related files, such as the certificate databases, certificates, keys, and associated users. It does not uninstall the subsystem packages. If the subsystem is the last subsystem on the server instance, the server instance is removed as well.

CHAPTER 17. REMOVING CERTIFICATE SYSTEM SUBSYSTEM PACKAGES

A number of subsystem-related packages and dependencies are installed with Red Hat Certificate System; these are listed in Section 7.2, "Certificate System Packages". Removing a subsystem removes only the files and directories associated with that specific subsystem. It does not remove the actual installed packages that are used by that instance. Completely uninstalling Red Hat Certificate System or one of its subsystems requires using package management tools, like **yum**, to remove each package individually.

To uninstall an individual Certificate System subsystem packages:

1. Remove all the associated subsystems. For example:

   ```
   pkidestroy -s CA -i pki-tomcat
   ```

2. Run the uninstall utility. For example:

   ```
   yum remove pki-subsystem_type
   ```

   The subsystem type can be **ca**, **kra**, **ocsp**, **tks**, or **tps**.

3. To remove other packages and dependencies, remove the packages specifically, using **yum**. The complete list of installed packages is at Section 7.2, "Certificate System Packages".

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 http://www.ietf.org/rfc/rfc2511.txt.

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. For more information, see https://tools.ietf.org/html/draft-ietf-tls-ecc-12.

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 that is 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 communciation 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 that is public key corresponds to a private key used to create digital signatures. For example, a Certificate Manager must have a signing certificate that is 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 that is 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

active logs

default file location, Logs

message categories, Services That Are Logged

adding new directory attributes, Adding New or Custom Attributes

agents

authorizing key recovery, Recovering Keys

port used for operations, Planning Ports

archiving

rotated log files, Log File Rotation

Audit log

defined, Transactions Log

B

buffered logging, Buffered and Unbuffered Logging

C

CA chaining, Linked CA

CA decisions for deployment

CA renewal, Renewing or Reissuing CA Signing Certificates

distinguished name, Planning the CA Distinguished Name

root versus subordinate, Defining the Certificate Authority Hierarchy

signing certificate, Setting the CA Signing Certificate Validity Period

signing key, Choosing the Signing Key Type and Length

CA hierarchy, Subordination to a Certificate System CA

root CA, Subordination to a Certificate System CA

subordinate CA, Subordination to a Certificate System CA

CA signing certificate, Setting the CA Signing Certificate Validity Period

Certificate Manager

as root CA, Subordination to a Certificate System CA

as subordinate CA, Subordination to a Certificate System CA

CA hierarchy, Subordination to a Certificate System CA

chaining to third-party CAs, Linked CA

KRA and, Planning for Lost Keys: Key Archival and Recovery

certificate profiles

Windows smart card login, Using the Windows Smart Card Logon Profile

H

how to search for keys,Archiving Keys

I

installation, Installing and Configuring Certificate System

    planning, A Checklist for Planning the PKI

K

key archival, Archiving Keys

    how it works,Archiving Keys

    how keys are stored,Archiving Keys

    how to set up,Manually Setting up Key Archival

    PKI setup required,Archiving, Recovering, and Rotating Keys

    reasons to archive,Archiving Keys

    where keys are stored,Archiving Keys

key length, Choosing the Signing Key Type and Length

key recovery, Recovering Keys

    how to set up,Setting up Agent-Approved Key Recovery Schemes

Key Recovery Authority

    setting up

        key archival, Manually Setting up Key Archival

        key recovery, Setting up Agent-Approved Key Recovery Schemes

KRA

    Certificate Manager and, Planning for Lost Keys: Key Archival and Recovery

L

linked CA, Linked CA

location of

    active log files,Logs

logging

    buffered vs. unbuffered,Buffered and Unbuffered Logging

    log files

        archiving rotated files,Log File Rotation

        default location,Logs

        timing of rotation,Log File Rotation

    log levels, Log Levels (Message Categories)

        default selection,Log Levels (Message Categories)

APPENDIX A. 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**
   Corrections for setting up HSMs in FIPS mode and various minor corrections for 9.4 Common Criteria Maintenance Update.

Revision 9.4-0   **Wed Apr 02, 2019**
   Initial version of this guide.