Red Hat Certificate System 8.1 Deployment, Planning, and Installation

preparing for a PKI infrastructure
Edition 8.1.1

Ella Deon Ballard
Abstract

This guide covers the major PKI concepts and decisions areas for planning a PKI infrastructure. This guide was updated for Errata RHSA-2012:1103.
# Table of Contents

## About This Guide
- 1. Examples and Formatting
  - 1.1. Formatting for Examples and Commands
  - 1.2. Tool Locations
  - 1.3. Text Formatting
  - 1.4. Recommended and Required Boxes
- 2. Additional Reading
- 3. Giving Feedback
- 4. Document History

## Part I. Planning How to Deploy Red Hat Certificate System

### Chapter 1. Introduction to Public-Key Cryptography
- 1.1. Encryption and Decryption
  - 1.1.1. Symmetric-Key Encryption
  - 1.1.2. Public-Key Encryption
  - 1.1.3. Key Length and Encryption Strength
- 1.2. Digital Signatures
- 1.3. Certificates and Authentication
  - 1.3.1. A Certificate Identifies Someone or Something
  - 1.3.2. Authentication Confirms an Identity
    - 1.3.2.1. Password-Based Authentication
    - 1.3.2.2. Certificate-Based Authentication
  - 1.3.3. How Certificates Are Used
    - 1.3.3.1. Uses for Certificates
      - 1.3.3.1.1. SSL
      - 1.3.3.1.2. Signed and Encrypted Email
      - 1.3.3.1.3. Single Sign-on
      - 1.3.3.1.4. Object Signing
    - 1.3.3.2. Types of Certificates
      - 1.3.3.2.1. CA Signing Certificates
      - 1.3.3.2.2. Other Signing Certificates
      - 1.3.3.2.3. SSL Server and Client Certificates
      - 1.3.3.2.4. User Certificates
      - 1.3.3.2.5. Dual-Key Pairs
      - 1.3.3.2.6. Cross-Pair Certificates
  - 1.3.4. Contents of a Certificate
    - 1.3.4.1. Certificate Data Formats
      - 1.3.4.1.1. Binary
      - 1.3.4.1.2. Text
    - 1.3.4.2. Distinguished Names
    - 1.3.4.3. A Typical Certificate
- 1.5. How CA Certificates Establish Trust
  - 1.5.1. CA Hierarchies
  - 1.5.2. Certificate Chains
  - 1.5.3. Verifying a Certificate Chain
- 1.6. Managing Certificates
  - 1.6.1. Issuing Certificates
  - 1.6.2. Key Management
  - 1.6.3. Renewing and Revoking Certificates

### Chapter 2. Introduction to Red Hat Certificate System

---

1

---

38
2.1. A Review of Certificate System Subsystems
2.2. How Certificate System Creates PKI (Non-TMS Environment)
   2.2.1. Issuing Certificates
      2.2.1.1. The Enrollment Process
      2.2.1.2. Certificate Profiles
      2.2.1.3. Authentication for Certificate Enrollment
      2.2.1.4. The Registration Manager
      2.2.1.5. Dual Key Pairs
      2.2.1.6. Cross-Pair Certificates
   2.2.2. Renewing Certificates
   2.2.3. Publishing Certificates and CRLs
   2.2.4. Revoking Certificates and Checking Status
      2.2.4.1. CRLs
      2.2.4.2. OCSP Services
         2.2.4.2.1. OCSP Response Signing
         2.2.4.2.2. OCSP Responses
         2.2.4.2.3. OCSP Services
      2.2.5. Archiving and Recovering Keys
         2.2.5.1. Archiving Keys
         2.2.5.2. Recovering Keys
   2.3. Working with Smart Cards (TMS)
      2.3.1. The TKS and Secure Channels
      2.3.2. TPS Operations
   2.4. Management and Security for Subsystems
      2.4.1. Notifications
      2.4.2. Jobs
      2.4.3. Logging
      2.4.4. Auditing
      2.4.5. Self-Tests
      2.4.6. Users, Authorization, and Access Controls
      2.4.7. Security-Enhanced Linux
   2.5. Red Hat Certificate System Services
      2.5.1. Administrative Consoles
         2.5.1.1. The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems
         2.5.1.2. The Administrative Interface for the RA and TPS
      2.5.2. Agent Interfaces
      2.5.3. End User Pages
      2.5.4. Enterprise Security Client

Chapter 3. Supported Standards and Protocols .................................................. 63
   3.1. PKCS #11 ....................................................... 63
   3.2. SSL/TLS, ECC, and RSA ........................................ 64
      3.2.1. Supported Cipher Suites for RSA .......................... 65
      3.2.2. Using ECC .............................................. 65
   3.3. IPv4 and IPv6 Addresses ........................................ 66
   3.4. Supported PKIX Formats and Protocols .......................... 67
   3.5. Supported Security and Directory Protocols .................... 67

Chapter 4. Planning the Certificate System ...................................................... 71
   4.1. Deciding on the Required Subsystems ................................. 71
      4.1.1. Using a Single Certificate Manager ......................... 71
      4.1.2. Planning for Lost Keys: Key Archival and Recovery ........ 72
      4.1.3. Balancing Certificate Request Processing ................. 73
      4.1.4. Balancing Client OCSP Requests .......................... 74
      4.1.5. Using Smart Cards ...................................... 74
4.2. Defining the Certificate Authority Hierarchy
4.2.1. Subordination to a Public CA
4.2.2. Subordination to a Certificate System CA
4.2.3. Linked CA
4.2.4. CA Cloning
4.3. Planning Security Domains
4.4. Determining the Requirements for Subsystem Certificates
4.4.1. Determining Which Certificates to Install
4.4.2. Planning the CA Distinguished Name
4.4.3. Setting the CA Signing Certificate Validity Period
4.4.4. Choosing the Signing Key Type and Length
4.4.5. Using Certificate Extensions
4.4.5.1. Structure of Certificate Extensions
4.4.6. Using and Customizing Certificate Profiles
4.4.7. Planning Authentication Methods
4.4.8. Publishing Certificates and CRLs
4.4.9. Renewing or Reissuing CA Signing Certificates
4.5. Planning for Network and Physical Security
4.5.1. Considering Firewalls
4.5.2. Considering Physical Security and Location
4.5.3. Planning Ports
4.6. Tokens for Storing Certificate System Subsystem Keys and Certificates
4.7. Implementing a Common Criteria Environment
4.8. A Checklist for Planning the PKI

Part II. Installing Red Hat Certificate System ........................................ 96

Chapter 5. Prerequisites and Preparation for Installation .............................. 97
5.1. Supported Platforms, Hardware, and Programs
5.1.1. Supported Platforms
5.1.2. Supported Web Browsers
5.1.3. Supported Smart Cards
5.1.4. Supported HSM
5.1.5. Supported Character Sets
5.1.6. Summary of Requirements for Common Criteria
5.2. Packages Installed on Red Hat Enterprise Linux
5.3. Before Installation: Setting up the Operating Environment
5.3.1. Installing the Required Java Development Kit (JDK)
5.3.2. Installing Apache (for the TPS)
5.3.3. Installing Red Hat Directory Server
5.3.4. Installing Additional Operating System Packages
5.3.5. Verifying Firewall Configuration and iptables
5.3.6. Enabling SELinux
5.3.7. Setting up Operating System Users and Groups
5.3.7.1. Creating Operating System Groups
5.3.7.2. Creating Operating System Users
5.3.7.2.1. Checking the pkuser System Account
5.3.7.2.2. Creating New User Accounts
5.3.7.2.3. Associating Existing User Accounts with PKI Groups
5.3.8. Using a Java Security Manager

Chapter 6. Installing and Configuring Certificate System ............................. 111
6.1. About pkicreate
6.2. Basic Installation
6.2.1. Installing and Configuring a CA
6.2.2. Installing and Configuring a DRM
B.1.5.3. Password and Certificate Storage
B.1.5.4. Hardware Token
B.1.5.5. Protection of Private and Secret Keys
B.1.6. Red Hat Certificate System 8.1 Privileged Users and Groups (Roles)
B.1.7. Understanding Setup of Common Criteria Evaluated Red Hat Certificate System 8.1
B.1.7.1. Understanding the Common Criteria Environment
B.1.7.1.1. Secure Environment
B.1.7.1.1.1. Network Environment
B.1.7.1.1.2. Operating System Environment
B.1.7.1.2. Verifying Package Bits
B.1.7.1.3. Red Hat Certificate System 8.1 Roles Assignment
B.1.7.1.4. Who Needs to Be Present
B.1.8. Common Criteria Deployment Scenarios
B.1.9. Understanding Subsystem Setup
B.1.9.1. Red Hat Certificate System 8.1 Role Users and Authorization
B.1.9.2. Audit Logs
B.1.9.3. Certificate Profiles
B.1.9.4. Authentication
B.1.9.5. CRLs
B.1.9.6. Jobs
B.1.9.7. Notifications
B.1.9.8. Publishing
B.1.9.9. Self Tests
B.1.9.10. Key Archival and Recovery
B.1.9.11. OCSP Responder Revocation Information Store
B.1.9.12. Token Management System
B.1.9.13. Features That Are Not Part of the Common Criteria Environment
B.1.10. Reporting Security Flaws
B.1.11. Relevant Links
B.2. Example Common Criteria Installations
B.2.1. Non-TMS Common Criteria Setup Procedures
B.2.1.1. Prerequisites: RHEL 5.6 Operating System Installation and Configuration
B.2.1.1.1. Non-TMS Inventory
B.2.1.1.2. Hardware Security Module Installation
B.2.1.1.3. SELinux Enforcement
B.2.1.1.4. Operating System User and Group Creation
B.2.1.1.5. Operating System User and Group Association
B.2.1.2. Red Hat Certificate System 8.1 Subsystem Packages
B.2.1.2.1. Directory Server Subsystem Package Installation
B.2.1.2.2. Certificate System Subsystem Package Installation
B.2.1.3. Hardware Security Module SELinux Context
B.2.1.4. Directory Server Instance Installation and Configuration
B.2.1.5. CA Instance Creation and Configuration
B.2.1.5.1. CA Instance Creation
B.2.1.5.2. Directory Server SSL Server Authentication Configuration (Temporary)
B.2.1.5.3. Hardware Security Module CA Configuration
B.2.1.5.4. CA Instance Configuration
B.2.1.5.5. Directory Server SSL Server Authentication Configuration (Trusted)
B.2.1.6. DRM Instance Creation and Configuration
B.2.1.6.1. DRM Instance Creation
B.2.1.6.2. Hardware Security Module DRM Configuration
B.2.1.6.3. DRM Instance Configuration
B.2.1.7. OCSP Instance Creation and Configuration
B.2.1.7.1. OCSP Instance Creation
B.2.1.7.2. Hardware Security Module OCSP Configuration
B.2.1.7.3. OCSP Instance Configuration
B.2.1.8. Post Configuration Instructions
B.2.1.8.1. Agent, Administrator, and Audit User Creation
B.2.1.8.2. Import of Certificate Chain into Browser
B.2.1.8.3. Sudo Permissions for PKI Services
B.2.1.8.4. Backup and Restore Procedures
B.2.1.8.5. Signed Audit
B.2.1.8.6. Revocation Checking and OCSP Responders
B.2.1.8.7. SSL Session Timeouts
B.2.1.8.8. Self-Tests Execution
B.2.1.8.9. Removal of password.conf
B.2.1.8.10. Removal of Unused Interfaces from web.xml
B.2.1.8.11. Downloading dumpans1 tool
B.2.1.8.12. Disable User/Group "multiroles" Support for Java Subsystems
B.2.1.8.13. Configuring POSIX System ACLs

B.2.2. TMS Common Criteria Setup Procedures
B.2.2.1. Prerequisites: RHEL 5.6 Operating System Installation and Configuration
B.2.2.1.1. TMS Inventory
B.2.2.1.2. Hardware Security Module Installation
B.2.2.1.3. SELinux Enforcement
B.2.2.1.4. Operating System User and Group Creation
B.2.2.1.5. Operating System User and Group Association
B.2.2.2. Red Hat Certificate System 8.1 Subsystem Packages
B.2.2.2.1. Directory Server Subsystem Package Installation
B.2.2.2.2. Certificate System Subsystem Package Installation
B.2.2.2.3. Hardware Security Module SELinux Context
B.2.2.4. Directory Server Instance Installation and Configuration
B.2.2.5. CA Instance Creation and Configuration
B.2.2.5.1. CA Instance Creation
B.2.2.5.2. Directory Server SSL Server Authentication Configuration (Temporary)
B.2.2.5.3. Hardware Security Module CA Configuration
B.2.2.5.4. CA Instance Configuration
B.2.2.5.5. Directory Server SSL Server Authentication Configuration (Trusted)
B.2.2.5.6. Configuring POSIX System ACLs
B.2.2.6. DRM Instance Creation and Configuration
B.2.2.6.1. DRM Instance Creation
B.2.2.6.2. Hardware Security Module DRM Configuration
B.2.2.6.3. DRM Instance Configuration
B.2.2.6.4. Configuring POSIX System ACLs
B.2.2.7. OCSP Instance Creation and Configuration
B.2.2.7.1. OCSP Instance Creation
B.2.2.7.2. Hardware Security Module OCSP Configuration
B.2.2.7.3. OCSP Instance Configuration
B.2.2.7.4. Configuring POSIX System ACLs
B.2.2.8. TKS Instance Creation and Configuration
B.2.2.8.1. TKS Instance Creation
B.2.2.8.2. Hardware Security Module TKS Configuration
B.2.2.8.3. TKS Instance Configuration
B.2.2.8.4. Configuring POSIX System ACLs
B.2.2.9. TPS Instance Creation and Configuration
B.2.2.9.1. TPS Instance Creation
B.2.2.9.2. Hardware Security Module TPS Configuration
B.2.2.9.3. TPS Instance Configuration
B.2.2.9.4. Configuring POSIX System ACLs
B.2.2.10. Post Configuration Instructions
B.2.2.10.1. Agent, Administrator, and Audit User Creation 390
B.2.2.10.2. Import of Certificate Chain into Browser 393
B.2.2.10.3. Sudo Permissions for PKI Services 394
B.2.2.10.4. Backup and Restore Procedures 394
B.2.2.10.5. Signed Audit 394
B.2.2.10.6. Revocation Checking and OCSP Responders 395
B.2.2.10.7. SSL Session Timeouts 395
B.2.2.10.8. Self-Tests Execution 395
B.2.2.10.9. Removal of password.conf 395
B.2.2.10.10. Removal of Unused Interfaces from web.xml 397
B.2.2.10.11. Downloading dumpans1 tool 397
B.2.2.10.12. Disable User/Group "multiroles" Support for Java Subsystems 398

B.3.1. Secure Usage Assumptions 398
B.3.1.1. Personnel Assumptions 398
B.3.1.2. Physical Assumptions 399
B.3.1.3. Connectivity Assumptions 399
B.3.1.4. Threats 400
  B.3.1.4.1. Authorized Users 400
  B.3.1.4.2. System 400
  B.3.1.4.3. Cryptography 400
  B.3.1.4.4. External Attacks 401
B.3.2. Organization Security Policies 401
B.4.1. Security Objectives for the Target of Evaluation 401
  B.4.1.1. Authorized Users 401
  B.4.1.2. System 401
  B.4.1.3. Cryptography 402
  B.4.1.4. External Attacks 402
B.4.2. Security Objectives for the Environment 402
B.4.3. Security Objectives for Both the Target of Evaluation and the Environment 404
B.5.1. Security Requirements for the IT Environment 406
B.5.2. Target of Evaluation Security Functional Requirements 406
  B.5.2.1. Security Audit (FAU) 408
  B.5.2.2. Communication (FCO) 412
  B.5.2.3. Cryptographic Support (FCS) 413
  B.5.2.4. User Data Protection (FDP) 413
  B.5.2.5. Identification and Authentication (FIA) 420
  B.5.2.6. Security Management (FMT) 420
  B.5.2.7. Protection of the Target Security Functions (FPT) 425
B.5.3. Target of Evaluation Security Assurance Requirements 426
  B.5.3.1. Development (ADV) 427
  B.5.3.2. Guidance Documents (AGD) 431
  B.5.3.3. Life-Cycle Support (ALC) 432
  B.5.3.4. Tests (ATE) 437
  B.5.3.5. Vulnerability Assessment (AVA) 439

Glossary ........................................................................................................... 440
A ................................................................................................................. 440
B ................................................................................................................. 442
C ................................................................................................................. 442
D ................................................................................................................. 446
E ................................................................................................................. 447
F ................................................................................................................. 448
I ................................................................................................................. 448
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>457</td>
</tr>
<tr>
<td>C</td>
<td>457</td>
</tr>
<tr>
<td>D</td>
<td>459</td>
</tr>
<tr>
<td>E</td>
<td>460</td>
</tr>
<tr>
<td>F</td>
<td>460</td>
</tr>
<tr>
<td>H</td>
<td>460</td>
</tr>
<tr>
<td>I</td>
<td>460</td>
</tr>
<tr>
<td>K</td>
<td>461</td>
</tr>
<tr>
<td>L</td>
<td>461</td>
</tr>
<tr>
<td>O</td>
<td>461</td>
</tr>
<tr>
<td>P</td>
<td>461</td>
</tr>
<tr>
<td>R</td>
<td>462</td>
</tr>
<tr>
<td>S</td>
<td>462</td>
</tr>
<tr>
<td>T</td>
<td>462</td>
</tr>
<tr>
<td>U</td>
<td>463</td>
</tr>
</tbody>
</table>

Table of Contents

J 449
K 450
L 450
M 450
N 451
O 451
P 452
R 453
S 454
T 456
V 457
About This Guide

This guide explains how to install and configure Red Hat Certificate System subsystems.

This guide is intended for experienced system administrators planning to deploy the Certificate System. Certificate System agents should refer to the Certificate System Agent’s Guide for information on how to perform agent tasks, such as handling certificate requests and revoking certificates. For information on using Certificate System to manage smart cards and security tokens, see Managing Smart Cards with the Enterprise Security Client.

Before using Certificate System, become familiar with the following concepts:

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

1. Examples and Formatting

1.1. Formatting for Examples and Commands

All of the examples for Red Hat Certificate System commands, file locations, and other usage are given for Red Hat Enterprise Linux 5.6 (32-bit) systems. Be certain to use the appropriate commands and files for your platform.

Example 1. Example Command

To start the Red Hat Certificate System:

```
service pki-ca start
```

The directory names are usually instance-specific. For example, the instance names are assumed to be `pki-subsystem_type`, such as `pki-ca`.

The port numbers used in the examples for the example subsystem instances are listed in Table 1, “Example Port Assignments for Certificate System 8.1”. The real port numbers used in a deployment depend on the values given when the subsystem was created.
Table 1. Example Port Assignments for Certificate System 8.1

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Standard</th>
<th>End-Entity SSL</th>
<th>Agent SSL</th>
<th>Admin SSL</th>
<th>Tomcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>9180</td>
<td>9444</td>
<td>9443</td>
<td>9445</td>
<td>9701</td>
</tr>
<tr>
<td>RA</td>
<td>12888</td>
<td></td>
<td>12889</td>
<td>12889</td>
<td></td>
</tr>
<tr>
<td>OCSP</td>
<td>11180</td>
<td>11444</td>
<td>11443</td>
<td>11445</td>
<td>11701</td>
</tr>
<tr>
<td>DRM</td>
<td>10180</td>
<td>10444</td>
<td>10443</td>
<td>10445</td>
<td>10701</td>
</tr>
<tr>
<td>TKS</td>
<td>13180</td>
<td>13444</td>
<td>13443</td>
<td>13445</td>
<td>13701</td>
</tr>
<tr>
<td>TPS</td>
<td>7888</td>
<td></td>
<td>7889</td>
<td>7889</td>
<td></td>
</tr>
</tbody>
</table>

1.2. Tool Locations
All of the tools for Red Hat Certificate System are located in the `/usr/bin` directory. These tools can be run from any location without specifying the tool location.

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

<table>
<thead>
<tr>
<th>Formatting Style</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monospace font</strong></td>
<td>Monospace is used for commands, package names, files and directory paths, and any text displayed in a prompt.</td>
</tr>
<tr>
<td>Monospace with a background</td>
<td>This type of formatting is used for anything entered or returned in a command prompt.</td>
</tr>
<tr>
<td><strong>Italicized text</strong></td>
<td>Any text which is italicized is a variable, such as <code>instance_name</code> or <code>hostname</code>. Occasionally, this is also used to emphasize a new term or other phrase.</td>
</tr>
<tr>
<td><strong>Bolded text</strong></td>
<td>Most phrases which are in bold are application names, such as Cygwin, or are fields or options in a user interface, such as a User Name Here: field or Save button.</td>
</tr>
</tbody>
</table>

Other formatting styles draw attention to important text.

![NOTE](image)  
A note provides additional information that can help illustrate the behavior of the system or provide more detail for a specific issue.

![IMPORTANT](image)  
Important information is necessary, but possibly unexpected, such as a configuration change that will not persist after a reboot.
1.4. Recommended and Required Boxes

Tasks may be available, recommended, or required to configure a Certificate System deployment. Additionally, some tasks may be recommended for a regular environment but required for a Common Criteria-certificated environment. Whether a task is required and what subsystems it is applied to are listed in recommended/required graphics at the beginning of major sections.

![Recommended and Required Boxes]

2. Additional Reading

The documentation for Certificate System includes the following guides:


- **Certificate System Administrator’s Guide** explains all administrative functions for the Certificate System. Administrators maintain the subsystems themselves, so this manual details backend configuration for certificate profiles, publishing, and issuing certificates and CRLs. It also covers managing subsystem settings like port numbers, users, and subsystem certificates. This manual is intended for Certificate System administrators.

- **Certificate System Agent’s Guide** describes how agents — users responsible for processing certificate requests and managing other aspects of certificate management — can use the Certificate System subsystems web services pages to process certificate requests, key recovery, OCSP requests and CRLs, and other functions. This manual is intended for Certificate System agents.

- **Managing Smart Cards with the Enterprise Security Client** explains how to install, configure, and use
the Enterprise Security Client, the user client application for managing smart cards, user certificates, and user keys.

This manual is intended for Certificate System administrators, agents, privileged users (such as security officers), and regular end users.

» Using End User Services is a quick overview of the end-user services in Certificate System, a simple way for users to learn how to access Certificate System services.

This manual is intended for regular end users.


This manual is intended for Certificate System administrators.


This manual is intended for Certificate System administrators.

» Release Notes contains important information on new features, fixed bugs, known issues and workarounds, and other important deployment information for Red Hat Certificate System 8.1.


3. Giving Feedback

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

» Select the Red Hat Certificate System product.
» Set the component to Doc - deployment-guide.
» Set the version number to 8.1.
» For errors, give the page number (for the PDF) or URL (for the HTML), and give a succinct description of the problem, such as incorrect procedure or typo.
» For enhancements, put in what information needs to be added and why.
» Give a clear title for the bug. For example, "Incorrect command example for setup script options" is better than "Bad example".

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

4. Document History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1.1-5</td>
<td>December 20, 2013</td>
<td>Ella Deon Ballard</td>
</tr>
<tr>
<td></td>
<td>Errata updates.</td>
<td></td>
</tr>
<tr>
<td>8.1-25</td>
<td>July 17, 2012</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td>Errata updates.</td>
<td></td>
</tr>
<tr>
<td>8.1-8</td>
<td>February 1, 2012</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td>Updating the subsystem overview image to clarify key recovery work flows.</td>
<td></td>
</tr>
<tr>
<td>Revision</td>
<td>Date</td>
<td>Author</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>8.1-7</td>
<td>January 24, 2012</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updating some links and adding a note to the Common Criteria examples in appendix B, per comments from reviewers.</td>
</tr>
<tr>
<td>8.1-6</td>
<td>January 6, 2012</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updating Common Criteria setup examples to correct some typos and clarify machine topology. Adding information on required multi-master replication setup for internal databases with cloning, per Bugzilla #743897. Adding step to change SELinux context for ECC libraries to prevent AVC denials, per Bugzilla #746756. Adding information on replication sequence numbers and cloning, per Bugzilla #750358.</td>
</tr>
<tr>
<td>8.1-5</td>
<td>September 28, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adding information on using setfacl for Posix system ACLs, per Bugzilla #723080. Adding procedures and Common Criteria examples for enabling FIPS mode on HSMs. Adding a section on configuring an HSM to work with ECC, per Bugzilla #677416. Adding a section on cloning a CA which has been re-keyed, per Bugzilla #741281.</td>
</tr>
<tr>
<td>8.1-4</td>
<td>July 25, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edits for comments from the ETR.</td>
</tr>
<tr>
<td>8.1-3</td>
<td>June 27, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edits for comments from the ETR.</td>
</tr>
<tr>
<td>8.1-2</td>
<td>May 26, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adding required LDAP database parameters to cloning example, from Bugzilla #704622. Fixed a typo in the LunaSA setup section, from Bugzilla #537529.</td>
</tr>
<tr>
<td>8.1-1</td>
<td>April 21, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adding new pkiremove parameter, from Bugzilla #696175.</td>
</tr>
<tr>
<td>8.1-0</td>
<td>March 21, 2011</td>
<td>Ella Deon Lackey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initial draft for Certificate System 8.1 documentation for review.</td>
</tr>
</tbody>
</table>
Part I. Planning How to Deploy Red Hat Certificate System

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. Introduction to Public-Key Cryptography

Public-key cryptography and related standards underlie the security features of many products such as signed and encrypted email, single sign-on, and Secure Sockets Layer (SSL) communications. This chapter covers the basic concepts of public-key cryptography.

Internet traffic, which passes information through intermediate computers, can be intercepted by a third party:

- **Eavesdropping.** Information remains intact, but its privacy is compromised. For example, someone could gather credit card numbers, record a sensitive conversation, or intercept classified information.
- **Tampering.** Information in transit is changed or replaced and then sent to the recipient. For example, someone could alter an order for goods or change a person's resume.
- **Impersonation.** Information passes to a person who poses as the intended recipient. Impersonation can take two forms:
  - **Spoofing.** A person can pretend to be someone else. For example, a person can pretend to have the email address `jdoe@example.net` or a computer can falsely identify itself as a site called `www.example.net`.
  - **Misrepresentation.** A person or organization can misrepresent itself. For example, a site called `www.example.net` can purport to be an on-line furniture store when it really receives credit-card payments but never sends any goods.

**Public-key cryptography** provides protection against Internet-based attacks through:

- **Encryption and decryption** allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.
- **Tamper detection** allows the recipient of information to verify that it has not been modified in transit. Any attempts to modify or substitute data are detected.
- **Authentication** allows the recipient of information to determine its origin by confirming the sender's identity.
- **Nonrepudiation** prevents the sender of information from claiming at a later date that the information was never sent.

### 1.1. Encryption and Decryption

**Encryption** is the process of transforming information so it is unintelligible to anyone but the intended recipient. **Decryption** is the process of decoding encrypted information. A cryptographic algorithm, also called a cipher, is a mathematical function used for encryption or decryption. Usually, two related functions are used, one for encryption and the other for decryption.

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a **key** that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, if not impossible.

#### 1.1.1. Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 1.1, “Symmetric-Key Encryption”.

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the legitimate parties using the key.

Symmetric-key encryption plays an important role in SSL communication, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described in the next section.

1.1.2. Public-Key Encryption

Public-key encryption (also called asymmetric encryption) involves a pair of keys, a public key and a private key, associated with an entity. Each public key is published, and the corresponding private key is kept secret. (For more information about the way public keys are published, see Section 1.3, “Certificates and Authentication”.) Data encrypted with a public key can be decrypted only with the corresponding private key. Figure 1.2, “Public-Key Encryption” shows a simplified view of the way public-key encryption works.
approach used by the SSL/TLS protocols.

The reverse of the scheme shown in Figure 1.2, "Public-Key Encryption" also works: data encrypted with a private key can be decrypted only with the corresponding public key. This is not a recommended practice to encrypt sensitive data, however, because it means that anyone with the public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful because it means the private key can be used to sign data with a digital signature, an important requirement for electronic commerce and other commercial applications of cryptography. Client software such as Mozilla Firefox can then use the public key to confirm that the message was signed with the appropriate private key and that it has not been tampered with since being signed. Section 1.2, "Digital Signatures" illustrates how this confirmation process works.

1.1.3. Key Length and Encryption Strength

Breaking an encryption algorithm is basically finding the key to the access the encrypted data in plain text. For symmetric algorithms, breaking the algorithm usually means trying to determine the key used to encrypt the text. For a public key algorithm, breaking the algorithm usually means acquiring the shared secret information between two recipients.

One method of breaking a symmetric algorithm is to simply try every key within the full algorithm until the right key is found. For public key algorithms, since half of the key pair is publicly known, the other half (private key) can be derived using published, though complex, mathematical calculations. Manually finding the key to break an algorithm is called a brute force attack.

Breaking an algorithm introduces the risk of intercepting, or even impersonating and fraudulently verifying, private information.

The key strength of an algorithm is determined by finding the fastest method to break the algorithm and comparing it to a brute force attack.

For symmetric keys, encryption strength is often described in terms of the size or length of the keys used to perform the encryption: longer keys generally provide stronger encryption. Key length is measured in bits. For example, 128-bit keys with the RC4 symmetric-key cipher supported by SSL provide significantly better cryptographic protection than 40-bit keys used with the same cipher. The 128-bit RC4 encryption is $3 \times 10^{26}$ times stronger than 40-bit RC4 encryption.

An encryption key is considered full strength if the best known attack to break the key is no faster than a brute force attempt to test every key possibility.

Different types of algorithms — particularly public key algorithms — may require different key lengths to achieve the same level of encryption strength as a symmetric-key cipher. The RSA cipher can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric-key encryption, can use all possible values for a key of a given length. More possible matching options means more security.

Because it is relatively trivial to break an RSA key, an RSA public-key encryption cipher must have a very long key — at least 1024 bits — to be considered cryptographically strong. On the other hand, symmetric-key ciphers are reckoned to be equivalently strong using a much shorter key length, as little as 80 bits for most algorithms.

1.2. Digital Signatures

Tamper detection relies on a mathematical function called a one-way hash (also called a message digest). A one-way hash is a number of fixed length with the following characteristics:
The value of the hash is unique for the hashed data. Any change in the data, even deleting or altering a single character, results in a different value.

The content of the hashed data cannot be deduced from the hash.

As mentioned in Section 1.1.2, “Public-Key Encryption”, it is possible to use a private key for encryption and the corresponding public key for decryption. Although not recommended when encrypting sensitive information, it is a crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, then uses the private key to encrypt the hash. The encrypted hash, along with other information such as the hashing algorithm, is known as a digital signature.

Figure 1.3, “Using a Digital Signature to Validate Data Integrity” illustrates the way a digital signature can be used to validate the integrity of signed data.

Figure 1.3. Using a Digital Signature to Validate Data Integrity

Figure 1.3, “Using a Digital Signature to Validate Data Integrity” shows two items transferred to the recipient of some signed data: the original data and the digital signature, which is a one-way hash of the original data encrypted with the signer’s private key. To validate the integrity of the data, the receiving software first uses the public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash to generate a new one-way hash of the same data. (Information about the hashing algorithm used is sent with the digital signature.) Finally, the receiving software compares the new hash against the original hash. If the two hashes match, the data has not changed since it was signed. If they do not match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that does not correspond to the public key presented by the signer.

If the two hashes match, the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. Confirming the identity of the signer also requires some way of confirming that the public key belongs to a particular entity. For more information on authenticating users, see Section 1.3, “Certificates and Authentication”.

A digital signature is similar to a handwritten signature. Once data have been signed, it is difficult to deny doing so later, assuming the private key has not been compromised. This quality of digital signatures provides a high degree of nonrepudiation; digital signatures make it difficult for the signer to deny having signed the data. In some situations, a digital signature is as legally binding as a handwritten signature.

1.3. Certificates and Authentication
1.3.1. A Certificate Identifies Someone or Something

A certificate is an electronic document used to identify an individual, a server, a company, or other entity and to associate that identity with a public key. Like a driver's license or passport, a certificate provides generally recognized proof of a person's identity. Public-key cryptography uses certificates to address the problem of impersonation.

To get personal ID such as a driver's license, a person has to present some other form of identification which confirms that the person is who he claims to be. Certificates work much the same way. Certificate authorities (CAs) validate identities and issue certificates. CAs can be either independent third parties or organizations running their own certificate-issuing server software, such as Certificate System. The methods used to validate an identity vary depending on the policies of a given CA for the type of certificate being requested. Before issuing a certificate, a CA must confirm the user's identity with its standard verification procedures.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies, such as the name of an employee or a server. Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, and a serial number. Most importantly, a certificate always includes the digital signature of the issuing CA. The CA's digital signature allows the certificate to serve as a valid credential for users who know and trust the CA but do not know the entity identified by the certificate.

For more information about the role of CAs, see Section 1.3.5, "How CA Certificates Establish Trust".

1.3.2. Authentication Confirms an Identity

Authentication is the process of confirming an identity. For network interactions, authentication involves the identification of one party by another party. There are many ways to use authentication over networks. Certificates are one of those way.

Network interactions typically take place between a client, such as a web browser, and a server. Client authentication refers to the identification of a client (the person assumed to be using the software) by a server. Server authentication refers to the identification of a server (the organization assumed to be running the server at the network address) by a client.

Client and server authentication are not the only forms of authentication that certificates support. For example, the digital signature on an email message, combined with the certificate that identifies the sender, can authenticate the sender of the message. Similarly, a digital signature on an HTML form, combined with a certificate that identifies the signer, can provide evidence that the person identified by that certificate agreed to the contents of the form. In addition to authentication, the digital signature in both cases ensures a degree of nonrepudiation; a digital signature makes it difficult for the signer to claim later not to have sent the email or the form.

Client authentication is an essential element of network security within most intranets or extranets. There are two main forms of client authentication:
Password-based authentication. Almost all server software permits client authentication by requiring a recognized name and password before granting access to the server.

Certificate-based authentication. Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server validates the signature and confirms the validity of the certificate.

1.3.2.1. Password-Based Authentication

Figure 1.4, “Using a Password to Authenticate a Client to a Server” shows the process of authenticating a user using a username and password. This example assumes the following:

> The user has already trusted the server, either without authentication or on the basis of server authentication over SSL.
> The user has requested a resource controlled by the server.
> The server requires client authentication before permitting access to the requested resource.

These are the steps in this authentication process:

1. When the server requests authentication from the client, the client displays a dialog box requesting the username and password for that server.
2. The client sends the name and password across the network, either in plain text or over an encrypted SSL connection.
3. The server looks up the name and password in its local password database and, if they match, accepts them as evidence authenticating the user's identity.
4. The server determines whether the identified user is permitted to access the requested resource and, if so, allows the client to access it.

With this arrangement, the user must supply a new password for each server accessed, and the administrator must keep track of the name and password for each user.

1.3.2.2. Certificate-Based Authentication

One of the advantages of certificate-based authentication is that it can be used to replace the first three steps in authentication with a mechanism that allows the user to supply one password, which is not sent across the network, and allows the administrator to control user authentication centrally. This is called single sign-on.

Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” shows how client authentication works using certificates and SSL. To authenticate a user to a server, a client digitally signs a randomly
generated piece of data and sends both the certificate and the signed data across the network. The server authenticates the user's identity based on the data in the certificate and signed data.

Like Figure 1.4, “Using a Password to Authenticate a Client to a Server”, Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” assumes that the user has already trusted the server and requested a resource and that the server has requested client authentication before granting access to the requested resource.

Unlike the authentication process in Figure 1.4, “Using a Password to Authenticate a Client to a Server”, the authentication process in Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” requires SSL. Figure 1.5, “Using a Certificate to Authenticate a Client to a Server” also assumes that the client has a valid certificate that can be used to identify the client to the server. Certificate-based authentication is preferred to password-based authentication because it is based on the user both possessing the private key and knowing the password. However, these two assumptions are true only if unauthorized personnel have not gained access to the user’s machine or password, the password for the client software’s private key database has been set, and the software is set up to request the password at reasonably frequent intervals.

NOTE

Neither password-based authentication nor certificate-based authentication address security issues related to physical access to individual machines or passwords. Public-key cryptography can only verify that a private key used to sign some data corresponds to the public key in a certificate. It is the user’s responsibility to protect a machine’s physical security and to keep the private-key password secret.

These are the authentication steps shown in Figure 1.5, “Using a Certificate to Authenticate a Client to a Server”:

1. The client software maintains a database of the private keys that correspond to the public keys published in any certificates issued for that client. The client asks for the password to this database the first time the client needs to access it during a given session, such as the first time the user attempts to access an SSL-enabled server that requires certificate-based client authentication.

   After entering this password once, the user does not need to enter it again for the rest of the
session, even when accessing other SSL-enabled servers.

2. The client unlocks the private-key database, retrieves the private key for the user's certificate, and uses that private key to sign data randomly-generated from input from both the client and the server. This data and the digital signature are evidence of the private key's validity. The digital signature can be created only with that private key and can be validated with the corresponding public key against the signed data, which is unique to the SSL session.

3. The client sends both the user's certificate and the randomly-generated data across the network.

4. The server uses the certificate and the signed data to authenticate the user's identity.

5. The server may perform other authentication tasks, such as checking that the certificate presented by the client is stored in the user's entry in an LDAP directory. The server then evaluates whether the identified user is permitted to access the requested resource. This evaluation process can employ a variety of standard authorization mechanisms, potentially using additional information in an LDAP directory or company databases. If the result of the evaluation is positive, the server allows the client to access the requested resource.

Certificates replace the authentication portion of the interaction between the client and the server. Instead of requiring a user to send passwords across the network continually, single sign-on requires the user to enter the private-key database password once, without sending it across the network. For the rest of the session, the client presents the user's certificate to authenticate the user to each new server it encounters. Existing authorization mechanisms based on the authenticated user identity are not affected.

### 1.3.3. How Certificates Are Used

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

#### 1.3.3.1. Uses for Certificates

- [Section 1.3.3.1.1, "SSL"](#)
- [Section 1.3.3.1.2, "Signed and Encrypted Email"](#)
- [Section 1.3.3.1.3, "Single Sign-on"](#)
- [Section 1.3.3.1.4, "Object Signing"](#)

#### 1.3.3.1.1. SSL

The Secure Sockets Layer (SSL) protocol governs server authentication, client authentication, and encrypted communication between servers and clients. SSL is widely used on the Internet, especially for interactions that involve exchanging confidential information such as credit card numbers.

SSL requires an SSL server certificate. As part of the initial SSL handshake, the server presents its certificate to the client to authenticate the server's identity. The authentication uses public-key encryption and digital signatures to confirm that the server is the server it claims to be. Once the server has been authenticated, the client and server use symmetric-key encryption, which is very fast, to encrypt all the information exchanged for the remainder of the session and to detect any tampering.

Servers may be configured to require client authentication as well as server authentication. In this case, after server authentication is successfully completed, the client must also present its certificate to the server to authenticate the client's identity before the encrypted SSL session can be established.

For an overview of client authentication over SSL and how it differs from password-based authentication, see [Section 1.3.2, "Authentication Confirms an Identity"](#).
1.3.3.1.2. Signed and Encrypted Email

Some email programs support digitally signed and encrypted email using a widely accepted protocol known as Secure Multipurpose Internet Mail Extension (S/MIME). Using S/MIME to sign or encrypt email messages requires the sender of the message to have an S/MIME certificate.

An email message that includes a digital signature provides some assurance that it was sent by the person whose name appears in the message header, thus authenticating the sender. If the digital signature cannot be validated by the email software, the user is alerted.

The digital signature is unique to the message it accompanies. If the message received differs in any way from the message that was sent, even by adding or deleting a single character, the digital signature cannot be validated. Therefore, signed email also provides assurance that the email has not been tampered with. This kind of assurance is known as nonrepudiation, which makes it difficult for the sender to deny having sent the message. This is important for business communication. For information about the way digital signatures work, see Section 1.2, “Digital Signatures”.

S/MIME also makes it possible to encrypt email messages, which is important for some business users. However, using encryption for email requires careful planning. If the recipient of encrypted email messages loses the private key and does not have access to a backup copy of the key, the encrypted messages can never be decrypted.

1.3.3.1.3. Single Sign-on

Network users are frequently required to remember multiple passwords for the various services they use. For example, a user might have to type a different password to log into the network, collect email, use directory services, use the corporate calendar program, and access various servers. Multiple passwords are an ongoing headache for both users and system administrators. Users have difficulty keeping track of different passwords, tend to choose poor ones, and tend to write them down in obvious places. Administrators must keep track of a separate password database on each server and deal with potential security problems related to the fact that passwords are sent over the network routinely and frequently.

Solving this problem requires some way for a user to log in once, using a single password, and get authenticated access to all network resources that user is authorized to use—without sending any passwords over the network. This capability is known as single sign-on.

Both client SSL certificates and S/MIME certificates can play a significant role in a comprehensive single sign-on solution. For example, one form of single sign-on supported by Red Hat products relies on SSL client authentication. A user can log in once, using a single password to the local client’s private-key database, and get authenticated access to all SSL-enabled servers that user is authorized to use—without sending any passwords over the network. This approach simplifies access for users, because they don’t need to enter passwords for each new server. It also simplifies network management, since administrators can control access by controlling lists of certificate authorities (CAs) rather than much longer lists of users and passwords.

In addition to using certificates, a complete single-sign on solution must address the need to interoperate with enterprise systems, such as the underlying operating system, that rely on passwords or other forms of authentication.

1.3.3.1.4. Object Signing

Many software technologies support a set of tools called object signing. Object signing uses standard techniques of public-key cryptography to let users get reliable information about code they download in much the same way they can get reliable information about shrink-wrapped software.

Most important, object signing helps users and network administrators implement decisions about
software distributed over intranets or the Internet—for example, whether to allow Java applets signed by a
given entity to use specific computer capabilities on specific users’ machines.

The objects signed with object signing technology can be applets or other Java code, JavaScript scripts,
plug-ins, or any kind of file. The signature is a digital signature. Signed objects and their signatures are
typically stored in a special file called a JAR file.

Software developers and others who wish to sign files using object-signing technology must first obtain
an object-signing certificate.

### 1.3.3.2. Types of Certificates

The Certificate System is capable of generating different types of certificates for different uses and in
different formats. Planning which certificates are required and planning how to manage them, including
determining what formats are needed and how to plan for renewal, are important to manage both the PKI
and the Certificate System instances.

This list is not exhaustive; there are certificate enrollment forms for dual-use certificates for LDAP
directories, file-signing certificates, and other subsystem certificates. These forms are available through
the Certificate Manager's end-entities page, at https://server.example.com:9444/ca/ee/ca.

For more detailed information about the different certificates that can be created, see the Certificate
System Agent's Guide.

When the different Certificate System subsystems are installed, the basic required certificates and keys
are generated; for example, configuring the Certificate Manager generates the CA signing certificate for
the self-signed root CA and the internal OCSP signing, audit signing, SSL server, and agent user
certificates. During the DRM configuration, the Certificate Manager generates the storage, transport,
audit signing, and agent certificates. Additional certificates can be created and installed separately.
<table>
<thead>
<tr>
<th>Certificate Type</th>
<th>Use</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client SSL certificates</td>
<td>Used for client authentication to servers over SSL. Typically, the identity of the client is assumed to be the same as the identity of a person, such as an employee. See Section 1.3.2.2, “Certificate-Based Authentication” for a description of the way SSL client certificates are used for client authentication. Client SSL certificates can also be used as part of single sign-on.</td>
<td>A bank gives a customer an SSL client certificate that allows the bank's servers to identify that customer and authorize access to the customer's accounts. A company gives a new employee an SSL client certificate that allows the company's servers to identify that employee and authorize access to the company's servers.</td>
</tr>
<tr>
<td>Server SSL certificates</td>
<td>Used for server authentication to clients over SSL. Server authentication may be used without client authentication. Server authentication is required for an encrypted SSL session. For more information, see Section 1.3.3.1.1, “SSL”.</td>
<td>Internet sites that engage in electronic commerce usually support certificate-based server authentication to establish an encrypted SSL session and to assure customers that they are dealing with the web site identified with the company. The encrypted SSL session ensures that personal information sent over the network, such as credit card numbers, cannot easily be intercepted.</td>
</tr>
<tr>
<td>S/MIME certificates</td>
<td>Used for signed and encrypted email. As with SSL client certificates, the identity of the client is assumed to be the same as the identity of a person, such as an employee. A single certificate may be used as both an S/MIME certificate and an SSL certificate; see Section 1.3.3.1.2, “Signed and Encrypted Email”. S/MIME certificates can also be used as part of single sign-on.</td>
<td>A company deploys combined S/MIME and SSL certificates solely to authenticate employee identities, thus permitting signed email and SSL client authentication but not encrypted email. Another company issues S/MIME certificates solely to sign and encrypt email that deals with sensitive financial or legal matters.</td>
</tr>
<tr>
<td>CA certificates</td>
<td>Used to identify CAs. Client and server software use CA certificates to determine what other certificates can be trusted. For more information, see Section 1.3.5, “How CA Certificates Establish Trust”.</td>
<td>The CA certificates stored in Mozilla Firefox determine what other certificates can be authenticated. An administrator can implement corporate security policies by controlling the CA certificates stored in each user's copy of Firefox.</td>
</tr>
<tr>
<td>Object-signing certificates</td>
<td>Used to identify signers of Java code, JavaScript scripts, or</td>
<td>Software companies frequently sign software distributed over</td>
</tr>
</tbody>
</table>
other signed files. the Internet to provide users with some assurance that the software is a legitimate product of that company. Using certificates and digital signatures can also make it possible for users to identify and control the kind of access downloaded software has to their computers.

<table>
<thead>
<tr>
<th>1.3.3.2.1. CA Signing Certificates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every Certificate Manager has a CA signing certificate with a public/private key pair it uses to sign the certificates and CRLs it issues. This certificate is created and installed when the Certificate Manager is installed.</td>
</tr>
<tr>
<td>The Certificate Manager's status as a root or subordinate CA is determined by whether its CA signing certificate is self-signed or is signed by another CA. Self-signed root CAs set the policies they use to issue certificates, such as the subject names, types of certificates that can be issued, and to whom certificates can be issued. A subordinate CA has a CA signing certificate signed by another CA, usually the one that is a level above in the CA hierarchy (which may or may not be a root CA). If the Certificate Manager is a subordinate CA in a CA hierarchy, the root CA's signing certificate must be imported into individual clients and servers before the Certificate Manager can be used to issue certificates to them.</td>
</tr>
<tr>
<td>The CA certificate must be installed in a client if a server or user certificate issued by that CA is installed on that client. The CA certificate confirms that the server certificate can be trusted. Ideally, the certificate chain is installed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.3.3.2.2. Other Signing Certificates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other services, such as the OCSP responder service and CRL publishing, can use signing certificates other than the CA certificate. For example, a separate CRL signing certificate can be used to sign the revocation lists that are published by a CA instead of using the CA signing certificate.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.3.3.2.3. SSL Server and Client Certificates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server certificates are used for secure communications, such as SSL, and other secure functions. Server certificates are used to authenticate themselves during operations and to encrypt data; client certificates authenticate the client to the server.</td>
</tr>
</tbody>
</table>

- Section 1.3.3.2.1, “CA Signing Certificates”
- Section 1.3.3.2.2, “Other Signing Certificates”
- Section 1.3.3.2.3, “SSL Server and Client Certificates”
- Section 1.3.3.2.4, “User Certificates”
- Section 1.3.3.2.5, “Dual-Key Pairs”
- Section 1.3.3.2.6, “Cross-Pair Certificates”
NOTE

CAs which have a signing certificate issued by a third-party may not be able to issue server certificates. The third-party CA may have rules in place which prohibit its subordinates from issuing server certificates.

1.3.3.2.4. User Certificates

End user certificates are a subset of client certificates that are used to identify users to a server or system. Users can be assigned certificates to use for secure communications, such as SSL, and other functions such as encrypting email or for single sign-on. Special users, such as Certificate System agents, can be given client certificates to access special services.

1.3.3.2.5. Dual-Key Pairs

Dual-key pairs are a set of two private and public keys, where one set is used for signing and one for encryption. These dual keys are used to create dual certificates. The dual certificate enrollment form is one of the standard forms listed in the end-entities page of the Certificate Manager.

When generating dual-key pairs, set the certificate profiles to work correctly when generating separate certificates for signing and encryption.

1.3.3.2.6. Cross-Pair Certificates

The Certificate System can issue, import, and publish cross-pair CA certificates. With cross-pair certificates, one CA signs and issues a cross-pair certificate to a second CA, and the second CA signs and issues a cross-pair certificate to the first CA. Both CAs then store or publish both certificates as a crossCertificatePair entry.

Bridging certificates can be done to honor certificates issued by a CA that is not chained to the root CA. By establishing a trust between the Certificate System CA and another CA through a cross-pair CA certificate, the cross-pair certificate can be downloaded and used to trust the certificates issued by the other CA.

1.3.4. Contents of a Certificate

The contents of certificates are organized according to the X.509 v3 certificate specification, which has been recommended by the International Telecommunications Union (ITU), an international standards body.

Users do not usually need to be concerned about the exact contents of a certificate. However, system administrators working with certificates may need some familiarity with the information contained in them.

1.3.4.1. Certificate Data Formats

Certificate requests and certificates can be created, stored, and installed in several different formats. All of these formats conform to X.509 standards.

1.3.4.1.1. Binary

The following binary formats are recognized:

- DER-encoded certificate. This is a single binary DER-encoded certificate.
- PKCS #7 certificate chain. This is a PKCS #7 SignedData object. The only significant field in the SignedData object is the certificates; the signature and the contents, for example, are ignored. The PKCS #7 format allows multiple certificates to be downloaded at a single time.
Netscape Certificate Sequence. This is a simpler format for downloading certificate chains in a PKCS #7 ContentInfo structure, wrapping a sequence of certificates. The value of the contentType field should be netscape-cert-sequence, while the content field has the following structure:

```
CertificateSequence ::= SEQUENCE OF Certificate
```

This format allows multiple certificates to be downloaded at the same time.

1.3.4.1.2. Text

Any of the binary formats can be imported in text form. The text form begins with the following line:

```
-----BEGIN CERTIFICATE-----
```

Following this line is the certificate data, which can be in any of the binary formats described. This data should be base-64 encoded, as described by RFC 1113. The certificate information is followed by this line:

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

1.3.4.2. Distinguished Names

An X.509 v3 certificate binds a distinguished name (DN) to a public key. A DN is a series of name-value pairs, such as uid=doe, that uniquely identify an entity. This is also called the certificate subject name.

This is an example DN of an employee for Example Corp.:

```
uuid=doe, cn=John Doe, o=Example Corp., c=US
```

In this DN, uid is the username, cn is the user’s common name, o is the organization or company name, and c is the country.

DNs may include a variety of other name-value pairs. They are used to identify both certificate subjects and entries in directories that support the Lightweight Directory Access Protocol (LDAP).

The rules governing the construction of DNs can be complex; for comprehensive information about DNs, see A String Representation of Distinguished Names at http://www.ietf.org/rfc/rfc4514.txt.

1.3.4.3. A Typical Certificate

Every X.509 certificate consists of two sections:

- The data section includes the following information:
  - The version number of the X.509 standard supported by the certificate.
  - The certificate’s serial number. Every certificate issued by a CA has a serial number that is unique among the certificates issued by that CA.
  - Information about the user’s public key, including the algorithm used and a representation of the key itself.
  - The DN of the CA that issued the certificate.
  - The period during which the certificate is valid; for example, between 1:00 p.m. on November 15, 2004, and 1:00 p.m. November 15, 2012.
  - The DN of the certificate subject, which is also called the subject name; for example, in an SSL client certificate, this is the user’s DN.
  - Optional certificate extensions, which may provide additional data used by the client or server. For
example, the Netscape Certificate Type extension indicates the type of certificate, such as an SSL client certificate, an SSL server certificate, or a certificate for signing email. Certificate extensions can also be used for other purposes.

The signature section includes the following information:
- The cryptographic algorithm, or cipher, used by the issuing CA to create its own digital signature.
- The CA's digital signature, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

Here are the data and signature sections of a certificate shown in the readable pretty-print format:

```
Certificate:
Data:
  Version: v3 (0x2)
  Serial Number: 3 (0x3)
  Signature Algorithm: PKCS #1 MD5 With RSA Encryption
  Issuer: OU=Example Certificate Authority, O=Example Corp, C=US
  Validity:
    Not After: Sun Oct 17 18:36:25 1999
  Subject: CN=Jane Doe, OU=Finance, O=Example Corp, C=US
  Subject Public Key Info:
    Algorithm: PKCS #1 RSA Encryption
    Public Key:
      Modulus:
      43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:1a:00:
      91:f4:15
      Public Exponent: 65537 (0x10001)
  Extensions:
    Identifier: Certificate Type
    Critical: no
    Certified Usage:
      SSL Client
    Identifier: Authority Key Identifier
    Critical: no
    Key Identifier:
      26:c9
  Signature:
    Algorithm: PKCS #1 MD5 With RSA Encryption
    Signature:
      d:c4
```

Here is the same certificate in the base-64 encoded format:

```
Certificate:
Data:
  Version: v3 (0x2)
  Serial Number: 3 (0x3)
  Signature Algorithm: PKCS #1 MD5 With RSA Encryption
  Issuer: OU=Example Certificate Authority, O=Example Corp, C=US
  Validity:
    Not After: Sun Oct 17 18:36:25 1999
  Subject: CN=Jane Doe, OU=Finance, O=Example Corp, C=US
  Subject Public Key Info:
    Algorithm: PKCS #1 RSA Encryption
    Public Key:
      Modulus:
      43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:1a:00:
      91:f4:15
      Public Exponent: 65537 (0x10001)
  Extensions:
    Identifier: Certificate Type
    Critical: no
    Certified Usage:
      SSL Client
    Identifier: Authority Key Identifier
    Critical: no
    Key Identifier:
      26:c9
  Signature:
    Algorithm: PKCS #1 MD5 With RSA Encryption
    Signature:
      d:c4
```
1.3.5. How CA Certificates Establish Trust

CAs validate identities and issue certificates. They can be either independent third parties or organizations running their own certificate-issuing server software, such as the Certificate System.

Any client or server software that supports certificates maintains a collection of trusted CA certificates. These CA certificates determine which issuers of certificates the software can trust, or validate. In the simplest case, the software can validate only certificates issued by one of the CAs for which it has a certificate. It is also possible for a trusted CA certificate to be part of a chain of CA certificates, each issued by the CA above it in a certificate hierarchy.

The sections that follow explains how certificate hierarchies and certificate chains determine what certificates software can trust.

- [Section 1.3.5.1, “CA Hierarchies”](#)
- [Section 1.3.5.2, “Certificate Chains”](#)
- [Section 1.3.5.3, “Verifying a Certificate Chain”](#)

1.3.5.1. CA Hierarchies

In large organizations, responsibility for issuing certificates can be delegated to several different CAs. For example, the number of certificates required may be too large for a single CA to maintain; different organizational units may have different policy requirements; or a CA may need to be physically located in the same geographic area as the people to whom it is issuing certificates.

These certificate-issuing responsibilities can be divided among subordinate CAs. The X.509 standard includes a model for setting up a hierarchy of CAs, shown in [Figure 1.6, “Example of a Hierarchy of Certificate Authorities”](#).
Figure 1.6. Example of a Hierarchy of Certificate Authorities

The root CA is at the top of the hierarchy. The root CA's certificate is a *self-signed certificate*; that is, the certificate is digitally signed by the same entity that the certificate identifies. The CAs that are directly subordinate to the root CA have CA certificates signed by the root CA. CAs under the subordinate CAs in the hierarchy have their CA certificates signed by the higher-level subordinate CAs.

Organizations have a great deal of flexibility in how CA hierarchies are set up; Figure 1.6, “Example of a Hierarchy of Certificate Authorities” shows just one example.

1.3.5.2. Certificate Chains

CA hierarchies are reflected in certificate chains. A *certificate chain* is series of certificates issued by successive CAs. Figure 1.7, “Example of a Certificate Chain” shows a certificate chain leading from a certificate that identifies an entity through two subordinate CA certificates to the CA certificate for the root CA, based on the CA hierarchy shown in Figure 1.6, “Example of a Hierarchy of Certificate Authorities”. 
A certificate chain traces a path of certificates from a branch in the hierarchy to the root of the hierarchy. In a certificate chain, the following occur:

- Each certificate is followed by the certificate of its issuer.
- Each certificate contains the name (DN) of that certificate’s issuer, which is the same as the subject name of the next certificate in the chain.

In Figure 1.7, “Example of a Certificate Chain”, the Engineering CA certificate contains the DN of the CA, USA CA, that issued that certificate. USA CA’s DN is also the subject name of the next certificate in the chain.

- Each certificate is signed with the private key of its issuer. The signature can be verified with the public key in the issuer’s certificate, which is the next certificate in the chain.

In Figure 1.7, “Example of a Certificate Chain”, the public key in the certificate for the USA CA can be used to verify the USA CA’s digital signature on the certificate for the Engineering CA.

### 1.3.5.3. Verifying a Certificate Chain

Certificate chain verification makes sure a given certificate chain is well-formed, valid, properly signed, and trustworthy. The following procedure is used to form and verify a certificate chain, starting with the certificate being presented for authentication:

1. The certificate validity period is checked against the current time provided by the verifier’s system clock.
2. The issuer’s certificate is located. The source can be either the verifier’s local certificate database on that client or server or the certificate chain provided by the subject, as with an SSL connection.
3. The certificate signature is verified using the public key in the issuer’s certificate.
4. If the issuer’s certificate is trusted by the verifier in the verifier’s certificate database, verification stops successfully here. Otherwise, the issuer’s certificate is checked to make sure it contains the appropriate subordinate CA indication in the certificate type extension, and chain verification starts over with this new certificate. **Figure 1.8, “Verifying a Certificate Chain to the Root CA”** presents an example of this process.

**Figure 1.8. Verifying a Certificate Chain to the Root CA**

**Figure 1.8, “Verifying a Certificate Chain to the Root CA”** illustrates what happens when only the root CA is included in the verifier's local database. If a certificate for one of the intermediate CAs, such as **Engineering CA**, is found in the verifier's local database, verification stops with that certificate, as shown in **Figure 1.9, “Verifying a Certificate Chain to an Intermediate CA”**.

**Figure 1.9. Verifying a Certificate Chain to an Intermediate CA**
Expired validity dates, an invalid signature, or the absence of a certificate for the issuing CA at any point in the certificate chain causes authentication to fail. Figure 1.10, “A Certificate Chain That Cannot Be Verified” shows how verification fails if neither the root CA certificate nor any of the intermediate CA certificates are included in the verifier's local database.

1.4. Managing Certificates

Certificates are used in many applications, from encrypting email to accessing websites. There are two major stages in the lifecycle of the certificate: the point when it is issued (issuance and enrollment) and the period when the certificates are no longer valid (renewal or revocation). There are also ways to manage the certificate during its cycle. Making information about the certificate available to other applications is publishing the certificate and then backing up the key pairs so that the certificate can be recovered if it is lost.

- Section 1.4.1, “Issuing Certificates”
- Section 1.4.2, “Key Management”
- Section 1.4.3, “Renewing and Revoking Certificates”

1.4.1. Issuing Certificates

The process for issuing a certificate depends on the CA that issues it and the purpose for which it will be used. Issuing non-digital forms of identification varies in similar ways. The requirements to get a library card are different than the ones to get a driver's license. Similarly, different CAs have different procedures for issuing different kinds of certificates. Requirements for receiving a certificate can be as simple as an email address or username and password to notarized documents, a background check, and a personal interview.
Depending on an organization's policies, the process of issuing certificates can range from being completely transparent for the user to requiring significant user participation and complex procedures. In general, processes for issuing certificates should be flexible, so organizations can tailor them to their changing needs.

### 1.4.2. Key Management

Before a certificate can be issued, the public key it contains and the corresponding private key must be generated. Sometimes it may be useful to issue a single person one certificate and key pair for signing operations and another certificate and key pair for encryption operations. Separate signing and encryption certificates keep the private signing key only on the local machine, providing maximum nonrepudiation. This also aids in backing up the private encryption key in some central location where it can be retrieved in case the user loses the original key or leaves the company.

Keys can be generated by client software or generated centrally by the CA and distributed to users through an LDAP directory. There are costs associated with either method. Local key generation provides maximum nonrepudiation but may involve more participation by the user in the issuing process. Flexible key management capabilities are essential for most organizations.

*Key recovery*, or the ability to retrieve backups of encryption keys under carefully defined conditions, can be a crucial part of certificate management, depending on how an organization uses certificates. In some PKI setups, several authorized personnel must agree before an encryption key can be recovered to ensure that the key is only recovered to the legitimate owner in authorized circumstance. It can be necessary to recover a key when information is encrypted and can only be decrypted by the lost key.

### 1.4.3. Renewing and Revoking Certificates

Like a driver’s license, a certificate specifies a period of time during which it is valid. Attempts to use a certificate for authentication before or after its validity period will fail. Managing certificate expirations and renewals are an essential part of the certificate management strategy. For example, an administrator may wish to be notified automatically when a certificate is about to expire so that an appropriate renewal process can be completed without disrupting the system operation. The renewal process may involve reusing the same public-private key pair or issuing a new one.

Additionally, it may be necessary to revoke a certificate before it has expired, such as when an employee leaves a company or moves to a new job in a different unit within the company.

Certificate revocation can be handled in several different ways. Servers can be configured so that the authentication process checks the directory for the presence of the certificate being presented. When an administrator revokes a certificate, the certificate can be automatically removed from the directory, and subsequent authentication attempts with that certificate will fail, even though the certificate remains valid in every other respect. Alternatively, a list of revoked certificates, a certificate revocation list (CRL), can be published to the directory at regular intervals. The CRL can be checked as part of the authentication process. The issuing CA can also be checked directly each time a certificate is presented for authentication. This procedure is sometimes called real-time status checking.
Chapter 2. Introduction to Red Hat Certificate System

Every common PKI operation — issuing, renewing and revoking certificates; archiving and recovering keys; publishing CRLs and verifying certificate status — is carried out by interooperating 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 six different subsystems, each focusing on different aspects of a PKI deployment:

- **A certificate authority called a 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 called a data recovery manager (DRM).** 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 DRM 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.

- **An online certificate status responder (OCSP).** 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 eases the load off of the issuing CA.

- **A registration authority (RA).** An RA accepts certificate requests and verifies, independently, whether that request should be approved. It then forwards approved requests to the CA to issue the certificate. Like the OCSP, this is a function that can be performed by the CA, but using a separate subsystem reduces the load on the 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, or process, 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. The Enterprise Security Client contacts the TPS when there is a token operation, and the TPS interacts with the CA, DRM, 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 like 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 DRM 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, registration authorities, and DRM instances as well.

### 2.2. How Certificate System Creates PKI (Non-TMS Environment)

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, registration authorities, and DRM instances as well.

2.2.1. 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.

2.2.1.1. The Enrollment Process

An end entity enrolls in the PKI by submitting an enrollment request through the end-entity interface. There can be many kinds of enrollment that use different enrollment methods or require different authentication methods. For each enrollment, there is a separate enrollment page created that is specific to the type of enrollment, type of authentication, and the certificate profiles associated with the type of certificate. The forms associated with enrollment can be customized for both appearance and content. Alternatively, the enrollment process can be customized by creating certificate profiles for each enrollment type. Certificate profiles dynamically-generate forms which are customized by configuring the inputs associated with the certificate profile.

When an end entity enrolls in a PKI by requesting a certificate, the following events can occur, depending on the configuration of the PKI and the subsystems installed:

1. The end entity provides the information in one of the enrollment forms and submits a request.
   - The information gathered from the end entity is customizable in the form depending on the information collected to store in the certificate or to authenticate against the authentication method associated with the form. The form creates a request that is then submitted to the Certificate Manager.
2. The enrollment form triggers the creation of the public and private keys or for dual-key pairs for the request.
3. The end entity provides authentication credentials before submitting the request, depending on the authentication type. This can be LDAP authentication, PIN-based authentication, or certificate-based authentication.
4. The request is submitted either to an agent-approved enrollment process or an automated process.
   - The agent-approved process, which involves no end-entity authentication, sends the request to the request queue in the agent services interface, where an agent must processes the request. An agent can then modify parts of the request, change the status of the request, reject the request, or approve the request.
     - Automatic notification can be set up so an email is sent to an agent any time a request appears in the queue. Also, an automated job can be set to send a list of the contents of the queue to agents on a pre-configured schedule.
   - The automated process, which involves end-entity authentication, processes the certificate request as soon as the end entity successfully authenticates.
5. The form collects information about the end entity from an LDAP directory when the form is submitted. For certificate profile-based enrollment, the defaults for the form can be used to collect the user LDAP ID and password.

6. The certificate profile associated with the form determine aspects of the certificate that is issued. Depending on the certificate profile, the request is evaluated to determine if the request meets the constraints set, if the required information is provided, and the contents of the new certificate.

7. The form can also request that the user export the private encryption key. If the DRM subsystem is set up with this CA, the end entity's key is requested, and an archival request is sent to the DRM. This process generally requires no interaction from the end entity.

8. The certificate request is either rejected because it did not meet the certificate profile or authentication requirements, or a certificate is issued.

9. The certificate is delivered to the end entity.
   - In automated enrollment, the certificate is delivered to the user immediately. Since the enrollment is normally through an HTML page, the certificate is returned as a response on another HTML page.
   - In agent-approved enrollment, the certificate can be retrieved by serial number or request ID in the end-entity interface.
   - If the notification feature is set up, the link where the certificate can be obtained is sent to the end user.

10. An automatic notice can be sent to the end entity when the certificate is issued or rejected.

11. The new certificate is stored in the Certificate Manager's internal database.

12. If publishing is set up for the Certificate Manager, the certificate is published to a file or an LDAP directory.

13. The internal OCSP service checks the status of certificates in the internal database when a certificate status request is received.

The end-entity interface has a search form for certificates that have been issued and for the CA certificate chain.

### 2.2.1.2. 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. 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. 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
2.2.1.3. 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.

2.2.1.4. The Registration Manager
A registration authority (RA) is an intermediary between a user and a CA. It accepts enrollment requests and then authenticates them locally. If the request is approved, the RA sends the request to the CA to issue the certificate and, once the certificate is issued, sends the certificate back to the user.

RAs remove some of the load from CAs by handling the validation part of a certificate request. For example, offices or organizations can validate requests locally, according to their predefined standards, using RA agents. This requires fewer CAs and allows the organization to group all of the CAs in a separate, secure location.

The kinds of certificates that can generated in the Certificate System RA are limited to the most common types of certificates: user certificates, SSL client certificates, RA agent certificates, and SCEP certificates for local routers. Users can check the RA services pages to view their certificate request status, retrieve their issued certificates, and renew their certificates.

Certificate System RAs can perform either automatic approval and manual approval, depending on the configuration in the enrollment profiles. Like the CA, the RA can also be configured to send a notification when the certificate request is processed.

The RA is normally set up outside of the firewall, and the CA is set up behind the firewall. This enables requests to be made from outside the protected environment (for example, the Internet), while the CA remains under the protection of the site's security measures.

2.2.1.5. Dual Key Pairs
The Certificate System supports generating dual key pairs, separate key pairs for signing and encrypting email messages and other data. To support separate key pairs for signing and encrypting data, dual certificates are generated for end entities, and the encryption keys are archived. If a client makes a certificate request for dual key pairs, the server issues two separate certificates.

2.2.1.6. 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.

2.2.2. 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.
2.2.3. 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.

2.2.4. 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.

2.2.4.1. 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.

2.2.4.2. 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.

2.2.4.2.1. 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 that's 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.

2.2.4.2.2. 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.

2.2.4.2.3. 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.

### 2.2.5. Archiving and Recovering Keys

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

- Clients that can generate dual keys and that support the key archival option (using the CRMF/CMMF protocol).
- An installed and configured DRM.
- HTML forms with which end entities can request dual certificates (based on dual keys) and key recovery agents can request key recovery.

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.

Clients that generate single key pairs use the same private key for both signing and encrypting data, so a private key derived from a single key pair cannot be archived and recovered. Clients that can generate dual key pairs use one private key for encrypting data and the other for signing data. Since the private encryption key is separate, it can be archived.

In addition to generating dual key pairs, the clients must also support archiving the encryption key in certificate requests. This option archives keys at the time the private encryption keys are generated as a part of issuing the certificate.

#### 2.2.5.1. Archiving Keys

The DRM 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 DRM 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 DRM'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 DRM) agents' certificates authorizes the DRM to complete the key recovery to retrieve its private storage key and use it to decrypt/recover an archived private key.

The DRM 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 DRM to archive the encryption key. The private key is encrypted by the transport key, and the DRM receives the encrypted copy and stores the key in its key repository. To archive the key, the DRM uses two special key pairs:

» A transport key pair and corresponding certificate.
» A storage key pair.

Figure 2.1, “How the Key Archival Process Works” illustrates how the key archival process occurs when an end entity requests a certificate.

Figure 2.1. How the Key Archival Process Works

1. The client requests and generates a dual key pair.
   a. The end entity, using a client which can generate dual key pairs, submits a request through the Certificate Manager enrollment form.
b. The client detects the JavaScript in the enrollment form and exports only the private encryption key, not the private signing key.

c. The Certificate Manager detects the key archival option in the request and asks the client for the private encryption key.

d. The client encrypts the private encryption key with the public key from the DRM's transport certificate embedded in the enrollment form.

2. After approving the certificate request and issuing the certificate, the Certificate Manager sends it to the DRM for storage, along with the public key. The Certificate Manager waits for verification from the DRM that the private key has been received and stored and that it corresponds to the public encryption key.

3. The DRM decrypts it with the private key. After confirming that the private encryption key corresponds to the public encryption key, the DRM 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 DRM uses the private key of its transport key pair to sign a token confirming that the key has been successfully stored; the DRM 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.

2.2.5.2. Recovering Keys

The DRM supports agent-initiated key recovery. Agent-initiated recovery is when designated recovery agents use the key recovery form on the DRM 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 DRM 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 DRM key recovery page. One of the agents initiates the key recovery process. The DRM 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.

There are two different paths for key recovery.

- **Synchronous recovery** means that when the first agent initiates the key recovery process, the process persists and the original browser must remain open until the entire process is complete. When the agent starts the recovery process, the DRM returns a reference number. All subsequent agents use the Authorize Recovery area and that referral link to access the thread. Continuous updates on the approval status are sent to the initiating agent so they can check the status.

**NOTE**

With synchronous recovery, the page that the first agent used to initiate the key recovery request keeps refreshing until all agents required to authorize have performed the authorization. It is important that the first agent does not close this browser session until the authorization is complete. Otherwise, the key recovery request needs to be started again.
Asynchronous recovery means that each step of the recovery process — the initial request and each subsequent approval or rejection — is stored in the DRM's internal database, under the key entry. The data for the recovery process can be retrieved even if the original browser session is closed or the DRM is shut down. Agents search for the key to recover, without using a reference number.

These two recovery options are illustrated in Figure 2.2, “Async and Sync Recovery, Side by Side”.

The DRM informs the agent who initiated the key recovery process of the status of the authorizations. When all of the authorizations are entered, the DRM 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 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 DRM 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 8.1 uses an *m-of-n ACL-based recovery scheme*, rather than a secret-splitting-based recovery scheme. In versions of Certificate System older than 7.1, the password for the storage token was split and protected by individual recovery agent passwords. Now, Certificate System uses its existing access control scheme to ensure recovery agents are appropriately authenticated over SSL and requires that the agent belong to a specific recovery agent group, by default the Data Recovery Manager Agents Group. The recovery request is executed only when *m-of-n* (a required number of) recovery agents have granted authorization to the request.

### 2.3. Working with Smart Cards (TMS)

Most certificates are enrolled through the CA. When certificates will be enrolled through an application such as a web browser or web server and will only be used by that application, then a simple CA or RA configuration is all that's necessary.

For security and for broader uses of the certificates, many organizations use smart cards (also called tokens). For managing smart cards, or tokens, Certificate System uses interlocking subsystems to create a *token management system* (TMS) which handles operations for certificates and keys stored on smart cards.

A TMS environment requires a CA, TKS, and TPS, with an optional DRM for server-side key generation:

- The Token Processing System (TPS) interacts with 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 Certificate Authority to generate certificates or the Data Recovery Manager to archive and recover keys.
- The Token Key Service (TKS) generates, or derives, symmetric keys used for communication between the TPS and smart card. Each set of keys generated by the TKS is unique because they are based on the card's unique ID. The keys are formatted on the smart card and are used to encrypt communications, or provide authentication, between the smart card and TPS.
- The Certificate Authority (CA) creates and revokes user certificates stored on the smart card.
- Optionally, the Data Recovery Manager (DRM) archives and recovers keys for the smart card.

The **Enterprise Security Client** is the conduit through which TPS communicates with each token over a secure HTTP channel (HTTPS), and, through the TPS, with the Certificate System. As with a non-TMS environment, an OCSP responder can be used to check the revocation status of certificates.
To use the tokens, the Token Processing System must be able to recognize and communicate with them. The tokens must first be enrolled to format the tokens with required keys and certificates and add the tokens to the Certificate System. The Enterprise Security Client provides the user interface for end entities to enroll tokens.

### 2.3.1. The TKS and Secure Channels

The Token Key Service (TKS) generates the (token) keys the TPS uses to communicate with the Enterprise Security Client. The TPS communicates with the TKS over SSL, but the TPS communicates with the smart cards themselves over a secure channel. The secure channel is made from an agreed on algorithm which is used by the TPS and smart card to independently derive three keys. The TKS provides the security between tokens and the TPS since the security relies on the relationship between the master key and the token keys.

Every smart card has three keys stored on it:

- **An auth key** which is used for encryption and authentication; a session key derived from the auth key each time a secure channel is opened.
- **A MAC key** which is used for message authentication; like the auth keys, a session key is derived from the MAC key each time a secure channel is opened.
- **A key encryption key** (KEK) which is to encrypt the session keys.

The TPS contacts the smart card to initialize a secure channel, sending key set information and a host challenge. The smart card creates a card challenge, generates the session keys, and generates a cryptogram, then sends its initialization response. The TKS — using the card UID number, an agreed on algorithm, and the key information — independently derives the session keys and the host cryptogram. The TKS uses the host/server cryptogram to verify the card cryptogram, and then TPS establishes the
secure channel with the smart card.

Having the TKS independently derive the session keys effectively shares secrets between the TPS and the smart card without having to store these symmetric keys on the server.

The TKS manages the master and transport keys required to generate and distribute keys for smart cards or tokens. A master key is a Triple DES symmetric key stored either in software or hardware token. The transport key wraps the other keys derived and used by the TKS and TPS.

The TKS actually carries out the key-related functions that the TPS needs for its communications:

- Helps establish a secure channel (signed and encrypted) between the token and TPS.
- Provides proof of presence for the security token during enrollment.
- Supports key changeover when the master key changes on the TKS. Tokens with older keys get new token keys.
- Helps generate a symmetric session key for the DRM to wrap (encrypt) the entity's private key for (optional) server-side key generation, where the entity's encryption keys are generated on the DRM

**NOTE**

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

### 2.3.2. TPS Operations

The TPS is the conduit between the Enterprise Security Client and the other subsystems (CA, TKS, DRM). The token operations are posted to the TPS using SSL and the URL of the management interface:

```
```

There are a number of available operations, and each operation is limited so that only certain types of TPS users can initiate it (Section 2.4.6, “Users, Authorization, and Access Controls”). In general, TPS operations include:

- Formatting smart cards
- Resetting the PIN on smart card tokens
- Upgrading the applet for smart card tokens
- Enrolling smart cards through the Enterprise Security Client
- Performing LDAP authentication
- Managing the token database
- Logging token events

Operations may have additional parameters that are passed with them. For example, changing the status of a smart card is the `do_token` operation, but what that status is changed to is set in the `question=` parameter (where 1 through 6 set the new token status) and the token being changed is identified in the `tid=` parameter. To change a token with the ID of 1234 to be temporarily lost (3) posts the following URL:

```
```
Operations are always submitted to the token database or TUS interface of the TPS, which is its administrative interface. This is the interface used by administrators and agents who access the administrative web UI for the TPS and for services. The Enterprise Security Client and other end-user functions access the smart card or TPS interface.

2.3.3. Token Profiles

Just like certificate profiles (Section 2.2.1.2, “Certificate Profiles”), there are different token profiles to format different kinds of tokens. A token profile defines two areas:

1. The steps to format and enroll the token (sort of like the forms used for certificate profiles)
2. The configuration of the final enrolled token

The profile configuration to format a smart card identify the authentication mechanisms to use, the LDAP database connection, the CA to use, and which entity generates the keys and the key settings. This also identifies the certificate profile on the CA to use to submit the token request. The profile also includes a mapping entry which provides a mechanism to filter the tokens to identify automatically which profile to use to enroll a token.
Example 2.1. Token Profile for DevKey

```
# Profile for DevKey
op.format.mapping.0.filter.tokenCUID.start=1000000000000000
op.format.mapping.0.filter.tokenCUID.end=1000000000000100
op.format.mapping.0.filter.tokenType=DevKey
op.format.mapping.0.target.tokenType=DevKey

# LDAP Connection settings for devKey
auth.instance.0.type=LDAP_Authentication
auth.instance.0.libraryName=/usr/lib/libldapauth.so
auth.instance.0.libraryFactory=GetAuthentication
auth.instance.0.authId=ldap-dev
auth.instance.0.hostport=ldap-dev.example.com:1111
auth.instance.0.SSLOn=false
auth.instance.0.retries=1
auth.instance.0.retryConnect=3
auth.instance.0.baseDN=o=dev
auth.instance.0.ui.title.en=LDAP Authentication
auth.instance.0.ui.description.en=This authenticates user against the DEV LDAP directory.
auth.instance.0.ui.id.UID.name.en=LDAP User ID
auth.instance.0.ui.id.PASSWORD.name.en=LDAP Password
auth.instance.0.ui.id.UID.description.en=DEV LDAP User ID
auth.instance.0.ui.id.PASSWORD.description.en=DEV LDAP Password
```

There are a handful of profile defined for tokens already. New and custom tokens can be created.
Table 2.1. Default Token Types

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleanToken</td>
<td>For operations for any blank token, without any other applied token types.</td>
</tr>
<tr>
<td>soKey</td>
<td>For operations for generating keys for security officer stations.</td>
</tr>
<tr>
<td>soCleanSOToken</td>
<td>For operations for blank tokens for security officer stations.</td>
</tr>
<tr>
<td>soKeyTemporary</td>
<td>For operations for temporary security officer tokens.</td>
</tr>
<tr>
<td>soCleanUserToken</td>
<td>For operations for blank user tokens for security officers.</td>
</tr>
<tr>
<td>soUserKey</td>
<td>For operations for security officer user tokens.</td>
</tr>
<tr>
<td>tokenKey</td>
<td>For operations for generating keys for uses with servers or devices.</td>
</tr>
<tr>
<td>userKey</td>
<td>For operations for regular user tokens.</td>
</tr>
<tr>
<td>userKeyTemporary</td>
<td>For operations for temporary user tokens.</td>
</tr>
</tbody>
</table>

2.4. Management and Security for Subsystems

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.

2.4.1. 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.

2.4.2. Jobs

Automated jobs run at defined intervals.

2.4.3. 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 desired 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.

2.4.4. 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.
2.4.5. 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.

2.4.6. Users, Authorization, and Access Controls

Certificate System users can be assigned to groups, 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 authentication, including simple username/password, SSL 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.

The Certificate System is configured by default with three user types 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.

Additionally, when a security domain is created, the CA subsystem which hosts the domain is automatically granted the 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.

2.4.7. Security-Enhanced Linux

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 the SELinux section in the Red Hat Enterprise Linux Deployment 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.

The Certificate System has a separate RPM of SELinux policies installed by default. 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.

![SELinux Administration](image)

Figure 2.4. 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 `pkicreate` is run, new SELinux policies are automatically configured for the instance. All SELinux policies are updated every time a subsystem is added with `pkicreate` or removed with `pkiremove`.

The central definition for each instance is its SELinux domain. Each Certificate System subsystem runs in a single subsystem-specific SELinux domain, no matter how many subsystems are installed on a host. For example, if there are three CAs installed on a server, all three belong to the `pki_ca_t` SELinux domain.
Each SELinux policy sets rules on what actions the instance is allowed to perform on the system, based on the domain to which the instance belongs. For example, instances in the CA domain (pki_ca_t) are limited to write access for files with the CA context (pki_ca_var_log_t) and to access ports that match the CA type (pki_ca_port). When each Certificate System process is started, it initially runs in an unconfined domain (unconfined_t) and then transitions into the appropriate subsystem-specific domain.

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

2.5. Red Hat Certificate System Services

There are three different interfaces for managing certificates and subsystems, depending on the user type: administrators, agents, and end users.

2.5.1. Administrative Consoles

The administrative interface is used to manage the subsystem itself. This includes adding users, configuring logs, managing profiles and plug-ins, and the internal database, among many other functions. This interface is also the only interface that does not directly deal with certificates, tokens, or keys, meaning it is not used for managing the PKI, only the servers.

There are two types of administrative consoles, Java-based and HTML-based. Although the interface is different, both are accessed using a server URL and the administrative port number.

2.5.1.1. The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems

The Java console is used by four subsystems: the CA, OCSP, DRM, and TKS. The console is accessed using a locally-installed pkiconsole utility. It can access any subsystem because the command requires the hostname, the subsystem’s administrative SSL port, and the specific subsystem type.

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

This opens a console, as in Figure 2.5, “Certificate System Console”.

![Figure 2.5. Certificate System Console](image)

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

All subsystems have four basic options:

- Users and groups
- Access control lists
- Log configuration
- Subsystem certificates (meaning the certificates issued to the subsystem for use, for example, in the security domain or audit signing)

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

### 2.5.1.2. The Administrative Interface for the RA and TPS

The RA and TPS subsystems use HTML-based administrative interfaces. These are accessed by entering the hostname and secure port as the URL, authenticating with the administrator's certificate, and clicking the appropriate **Administrators** link.

**NOTE**

There is a single SSL port for RA and TPS subsystems which is used for both administrator and agent services. Access to those services is restricted by certificate-based authentication. The other subsystems used separate SSL ports for the agent and administrative services, along with certificate-based authentication.

The HTML admin interface is much more limited than the Java console; the primary administrative function is managing the subsystem users; all other administrative tasks are done by manually editing the **CS.cfg** file.

The RA allows administrators to create and edit users and groups for the subsystem.

**Figure 2.6. RA Admin Page**

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

Figure 2.7. TPS Admin Page

2.5.2. Agent Interfaces

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

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

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

### 2.5.3. End User Pages

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

The end-user services are accessed over standard HTTP using the server’s hostname and the standard port number; they can also be accessed over HTTPS using the server’s hostname and the specific end-entities SSL port.

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

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

The RA is a more lightweight subsystem, so it only processes four common certificate profiles. Like the CA, the enrollment forms are accessed through the End Entities URL. Users can submit certificate requests and retrieve their certificates through the RA.

### 2.5.4. Enterprise Security Client

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

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

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

- Supports JavaCard 2.1 or higher cards and Global Platform 2.01-compliant smart cards like Safenet's 330J smart card
- Supports Global Platform 2.01-compliant smart cards like Gemalto e-gate 32K and Gemalto TOP IM FIPS CY2 tokens, both the smart card and GemPCKey USB form factor key.
Enrolls security tokens so they are recognized by TPS.
Maintains the security token, such as re-enrolling a token with TPS.
Provides information about the current status of the token or tokens being managed.
Supports server-side key generation so that keys can be archived and recovered on a separate token if a token is lost.

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

NOTE
For more information on using the Enterprise Security Client, see the Certificate System Enterprise Security Client Guide.

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

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

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

- Allows the user to enroll security tokens so they are recognized by the TPS.
- Allows the user to maintain the security token. For example, Enterprise Security Client makes it possible to re-enroll a token with the TPS.
- Provides support for several different kinds of tokens through default and custom token profiles. By default, the TPS can automatically enroll user keys, device keys, and security officer keys; additional profiles can be added so that tokens for different uses (recognized by attributes such as the token CUID) can automatically be enrolled according to the appropriate profile.
- Provides information about the current status of the tokens being managed.
Chapter 3. Supported 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 supported by Certificate System 8.1 are outlined in this chapter, to help administrators plan their client services effectively.

3.1. 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 the hardware or software device that actually provides cryptographic services and optionally stores certificates and keys.

Two cryptographic modules 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.

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. This file can be modified using the `modutil` tool, and it should be modified when there are changes to the system like installing hardware accelerators to use for signing operations. For more information on `modutil`, see [http://www.mozilla.org/projects/security/pki/nss/tools/](http://www.mozilla.org/projects/security/pki/nss/tools/).

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

### 3.2. SSL/TLS, ECC, and RSA

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols are universally accepted standards for authenticated and encrypted communication between clients and servers. Both client and server authentication occur over SSL/TLS.

SSL/TLS uses a combination of public key and symmetric-key encryption. Symmetric-key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL/TLS session always begins with an exchange of messages called the SSL handshake, initial communication between the server and client. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows.

Both of these protocols support using a variety of different cryptographic algorithms, or ciphers, for operations such as authenticating the server and client, transmitting certificates, and establishing session keys. Clients and servers may support different cipher suites, or sets of ciphers. Among other functions, the SSL handshake determines how the server and client negotiate which cipher suite they will use to authenticate each other, to transmit certificates, and to establish session keys.

Key-exchange algorithms like RSA and Elliptic Curve Cryptography (ECC) govern the way the server and client determine the symmetric keys to use during an SSL session. The most common SSL cipher suites use RSA key exchange, while TLS supports ECC cipher suites as well as RSA. The Certificate System [1] supports both RSA and ECC public-key cryptographic systems.

NOTE

Longer RSA keys are required to provide security as computing capabilities increase. The recommended RSA key-length is 2048 bits. Though many web servers continue to use 1024-bit keys, web servers should migrate to at least 2048 bits. For 64-bit machines, consider using stronger keys. All CAs should use at least 2048-bit keys, and stronger keys (such as 3072 or 4096 bits) if possible.

As PKIs using RSA keys and certificates transition to other cryptographic systems like ECC, servers should continue to support RSA. Certificate System supports using both RSA- and ECC-based certificates in the same subsystem.
3.2.1. Supported Cipher Suites for RSA

Certificate System supports several different cipher suites with the RSA key exchange:

- **AES and SHA Message Authentication.** Advanced Encryption Standard (AES) ciphers have a fixed block size of 128-bits, and the keys can be either 128-bit or 256-bit. There are $3.4 \times 10^{38}$ possible 128-bit keys and $1.1 \times 10^{77}$ possible 256-bit keys. There are more possible keys than any other cipher, making AES the strongest cipher supported by SSL. These cipher suites are FIPS-compliant.

- **Triple DES and SHA Message Authentication.** Triple DES (Data Encryption Standard) is the second-strongest cipher supported by SSL, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are approximately $3.7 \times 10^{50}$ possible keys. This cipher suite is FIPS-compliant.

- **RC4 and RC2 and MD5 Message Authentication.** The RC4 and RC2 ciphers have 128-bit encryption, which permits approximately $3.4 \times 10^{38}$ possible keys. This makes RC4 or RC2 keys very difficult to crack. RC4 ciphers are faster than RC2 ciphers.

- **DES and SHA Message Authentication.** DES 56-bit encryption permits approximately $7.2 \times 10^{16}$ possible keys. This cipher suite is no longer FIPS-compliant because it is too weak cryptographically.

3.2.2. Using ECC

*Elliptic Curve Cryptography (ECC)* is a cryptographic system that uses elliptic curves to create keys for encrypting data. ECC creates cryptographically-stronger keys with shorter key lengths than RSA, which makes it faster and more efficient to implement.

ECC has several advantages over RSA, since it is faster and requires shorter key lengths for stronger keys. The drawback to using ECC is that it is not as widely supported as RSA.

**Table 3.1. Comparison of RSA and ECC Cipher Strength**

<table>
<thead>
<tr>
<th>Bits of Security</th>
<th>RSA Key Length</th>
<th>ECC Key Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1024</td>
<td>160-223</td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>224-255</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256-383</td>
</tr>
<tr>
<td>192</td>
<td>7860</td>
<td>384-511</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>512+</td>
</tr>
</tbody>
</table>


Certificate System supports using ECC with all of its subsystems, so ECC certificate requests can be submitted to CAs through any of the enrollment profiles and ECC keys can be archived and restored in the DRM. However, Certificate System does not include ECC support natively, so using ECC is slightly different than using RSA:

- An external PKCS#11 module must be loaded before any subsystems are installed so that they can be configured with ECC subsystem certificates.

- The CA profile pages can process ECC certificate requests, but they cannot generate ECC keys. Some of the profile forms, like manual user certificates, then cannot be used for ECC certificates. In those cases, a different or custom profile needs to be used, and certificate requests have to be generated using **certutil**.

For a CA to issue ECC certificates, the CA must be configured with an ECC CA signing certificate. This is...
best done by loading an ECC PKCS#11 module before the CA is installed, and then configuring the CA using ECC keys.

**NOTE**

A CA with an ECC CA signing certificate can issue both ECC and RSA certificates. A CA with an RSA CA signing certificate can only issue RSA certificates.

Only the CA signing certificate is required; if for support purposes it is better to use RSA client certificates with the CA, simply delete the ECC subsystem certificates (except for the signing certificate) and replace them with RSA certificates.

For more information on ECC, see [RFC 4492](https://tools.ietf.org/html/rfc4492), Section 5.6.1, Table 2.

### 3.3. IPv4 and IPv6 Addresses

Certificate System supports both IPv4 addresses and IPv6 addresses. In a very wide variety of circumstances, Certificate System subsystems or operations references a hostname or IP address; supporting both IPv4- and IPv6-style addresses ensures forward compatibility with network protocols. The operations that support IPv6 connections include the following:

- Communications between subsystems, including between the RA and CA and between the TPS, TKS, and CA and for joining security domains
- Token operations between the TPS and Enterprise Security Client
- Subsystem logging
- Access control instructions
- Operations performed with Certificate System tools, including the Subject Alt Name Extension tool, HttpClient, and the Bulk Issuance Tool
- Client communications, including both the `pkiconsole` and IPv6-enabled browsers for web services
- Certificate request names and certificate subject names, including user, server, and router certificates
- Publishing
- Connecting to LDAP databases for internal databases and authentication directories

Any time a hostname or URL is referenced, an IP address can be used:

- An IPv4 address must be in the format `n.n.n.n` or `n.n.n.n,m.m.m.m`. For example, `128.21.39.40` or `128.21.39.40,255.255.255.00`.

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the web services pages and to the subsystem Java consoles. The most common method is to use fully-qualified domain names:

- `https://ipv6host.example.com:9445/ca/services`
- `pkiconsole https://ipv6host.example.com:9445/ca`

To use IPv6 numeric addresses, then replace the fully-qualified domain name in the URL with the IPv6
address, enclosed in brackets ([ ]). For example:

```
https://[00:00:00:00:123:456:789:00:]:9445/ca/services
pkiconsole https://[00:00:00:00:123:456:789:00:]:9445/ca
```

### 3.4. Supported PKIX Formats and Protocols

The Certificate System supports many of the protocols and formats defined in Public-Key Infrastructure (X.509) by the IETF. Along with the PKIX standards listed here, other PKIX-listed standards are available at [http://www.ietf.org/html.charters/pkix-charter.html](http://www.ietf.org/html.charters/pkix-charter.html) under the **Internet Drafts** section.

**Table 3.2. PKIX Standards Supported in Certificate System 8.1**

<table>
<thead>
<tr>
<th>Format or Protocol</th>
<th>RFC or Draft</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X.509 version 1 and version 3</td>
<td></td>
<td>Digital certificate formats recommended by the International Telecommunications Union (ITU).</td>
</tr>
<tr>
<td>Certificate Request Message Format (CRMF)</td>
<td>RFC 4211</td>
<td>A message format to send a certificate request to a CA.</td>
</tr>
<tr>
<td>Certificate Management Message Formats (CMMF)</td>
<td></td>
<td>Message formats to send certificate requests and revocation requests from end entities to a CA and to return information to end entities. CMMF has been subsumed by another standard, CMC.</td>
</tr>
<tr>
<td>Certificate Management Messages over CS (CMC)</td>
<td>RFC 5274</td>
<td>A general interface to public-key certification products based on CS and PKCS #10, including a certificate enrollment protocol for RSA-signed certificates with Diffie-Hellman public-keys. CMC incorporates CRMF and CMMF.</td>
</tr>
<tr>
<td>Cryptographic Message Syntax (CMS)</td>
<td>RFC 2630</td>
<td>A superset of PKCS #7 syntax used for digital signatures and encryption.</td>
</tr>
<tr>
<td>PKIX Certificate and CRL Profile</td>
<td>RFC 5280</td>
<td>A standard developed by the IETF for a public-key infrastructure for the Internet. It specifies profiles for certificates and CRLs. For more information about certificate and CRL profiles, see <a href="http://www.ietf.org/rfc/rfc5280.txt">http://www.ietf.org/rfc/rfc5280.txt</a>.</td>
</tr>
</tbody>
</table>

### 3.5. Supported Security and Directory Protocols
The Certificate System supports several common Internet and network protocols.
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIPS PUBS 140</td>
<td>Federal Information Standards Publications (FIPS PUBS) 140 is a US government standard for implementing cryptographic modules such as hardware or software that encrypts and decrypts data, creates and verifies digital signatures, and other cryptographic functions. More information is available at <a href="http://csrc.nist.gov/publications/PubsFIPS.html">http://csrc.nist.gov/publications/PubsFIPS.html</a>.</td>
</tr>
<tr>
<td>Hypertext Transport Protocol (HTTP) and Hypertext Transport Protocol Secure (HTTPS)</td>
<td>Protocols used to communicate with web servers.</td>
</tr>
<tr>
<td>KEYGEN tag</td>
<td>An HTML tag that generates a key pair for use with a certificate.</td>
</tr>
<tr>
<td>Lightweight Directory Access Protocol (LDAP) v2, v3</td>
<td>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.</td>
</tr>
<tr>
<td>Public-Key Cryptography Standard (PKCS) #7</td>
<td>An encrypted data and message format developed by RSA Data Security to represent digital signatures, certificate chains, and encrypted data. This format is used to deliver certificates to end entities.</td>
</tr>
<tr>
<td>Public-Key Cryptography Standard (PKCS) #10</td>
<td>A message format developed by RSA Data Security for certificate requests. This format is supported by many server products.</td>
</tr>
<tr>
<td>Public-Key Cryptography Standard (PKCS) #11</td>
<td>Specifies an API used to communicate with devices such as hardware tokens that hold cryptographic information and perform cryptographic operations.</td>
</tr>
<tr>
<td>Secure Sockets Layer (SSL) 2.0 and 3.0 and Transport Layer Security (TLS)</td>
<td>A set of rules governing server authentication, client authentication, and encrypted communication between servers and clients.</td>
</tr>
<tr>
<td>Security-Enhanced Linux</td>
<td>Security-enhanced Linux, or SELinux, is a set of security protocols enforcing mandatory access control on Linux system kernels. This was developed by the United States National Security Agency to keep applications from accessing confidential or protected files through lenient or flawed access controls.</td>
</tr>
<tr>
<td>Simple Certificate Enrollment Protocol (SCEP)</td>
<td>A protocol designed by Cisco to specify a way for a router to communicate with an RA or CA for router certificate enrollment. SCEP defines two modes of operation: RA mode and CA mode. Certificate System supports CA mode, where the request is encrypted with the CA signing certificate.</td>
</tr>
</tbody>
</table>
| UTF-8                                                                   | The certificate enrollment pages support all UTF-
8 characters for specific fields (common name, organizational unit, requester name, and additional notes). The certificates will be generated with the UTF-8 strings correctly used in the subject names and other fields, and the UTF-8 strings are searchable and correctly display in the CA, OCSP, and DRM end user and agents services pages. This UTF-8 support does not extended to internationalized domain names, like in email addresses.

| IPv4 and IPv6 | Certificate System supports both IPv4 and IPv6 address namespaces for communications and operations with all subsystems and tools, as well as for clients, subsystem creation, and token and certificate enrollment, |

[1] Only RSA is supported natively. External PKCS#11 modules with ECC support must be loaded for subsystems to use ECC.
Chapter 4. Planning the Certificate System

Each Red Hat Certificate System subsystem is installed and configured separately. They can all be installed on the same 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.

4.1. 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?

4.1.1. 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.
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.

4.1.2. 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 DRM, the subsystem which specially archives and retrieves keys.

The DRM 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.

**4.1.3. 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 4.2, “Defining the Certificate Authority Hierarchy”.

Another option, though is to distribute some of the tasks of a single CA to another subsystem. For example, Example Corp. has a manageable number of people requesting certificates for a single CA to issue. However, because of their security policies, each certificate request has to be verified in person by an agent, with supporting documentation. This creates a bottleneck for the CA agents to approve requests. A registration authority (RA) is installed at each local office; the requests are processed and approved locally, and then a central CA issues all of the certificates.
The Registration Manager takes the load of processing certificate requests; the CA then only has to issue the requests. For network environments where there are strict, and possibly time-consuming, rules for issuing certificates, an RA can speed the process while also give control to local managers and administrators.

RA managers are also good for certain network demands. CAs require a very high degree of both physical security and network security because of the sensitive nature of the information they contain. RAs can be placed outside of a firewall so that regular users can connect to them and can be stored in less secure locations because they do not process or contain sensitive data.

### 4.1.4. Balancing Client OCSP Requests

If a certificate is within its validity period but needs to 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 SSL 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.

### 4.1.5. Using Smart Cards

Smart cards usually require in-person enrollment and approval processes, since they use a physical
medium to store key and certificate data. That means that multiple agents need to have access to the TPS and, possibly, multiple TPS subsystems in different offices or geographic locations.

The token management system is very scalable. Multiple TPSs can be configured to work with a single CA, TKS, or DRM instance, while multiple Enterprise Security Clients can communicate with a single TPS. As additional clients are installed, they can point back to the TPS instance without having to reconfigure the TPS; likewise, as TPSs are added, they can point to the same CA, TKS, and DRM instances without having to reconfigure those subsystems.

After installation, the TPS configuration can be edited to use additional CA, DRM, and TKS instances for failover support, so if the primary subsystem is unavailable, the TPS can switch to the next available system without interrupting its token services.

4.2. 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.

4.2.1. 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 SSL client authentication certificates, but not SSL 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.

4.2.2. 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.

4.2.3. 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.

4.2.4. CA Cloning

Instead of creating a hierarchy of root and subordinate CAs, it is possible to create multiple clones of a Certificate Manager and configure each clone to issue certificates within a range of serial numbers.

A cloned Certificate Manager uses the same CA signing key and certificate as another Certificate Manager, the master Certificate Manager.
If there is a chance that a subsystem will be cloned, then it is easiest to export its key pairs during the configuration process and save them to a secure location. The key pairs for the original Certificate Manager have to be available when the clone instance is configured, so that the clone can generate its certificates from the original Certificate Manager's keys. It is also possible to export the keys from the security databases at a later time, using the `pk12util` or the `PKCS12Export` commands.

Because clone CAs and original CAs use the same CA signing key and certificate to sign the certificates they issue, the *issuer name* in all the certificates is the same. Clone CAs and the original Certificate Managers issue certificates as if they are a single CA. These servers can be placed on different hosts for high availability failover support.

The advantage of cloning is that it distributes the Certificate Manager's load across several processes or even several physical machines. For a CA with a high enrollment demand, the distribution gained from cloning allows more certificates to be signed and issued in a given time interval.

A cloned Certificate Manager has the same features, such as agent and end-entity gateway functions, of a regular Certificate Manager.

The serial numbers for certificates issued by clones are distributed dynamically. The databases for each clone and master are replicated, so all of the certificate requests and issued certificates, both, are also replicated. This ensures that there are no serial number conflicts while serial number ranges do not have to be manually assigned to the cloned Certificate Managers.

### 4.3. 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 — RA, DRM, 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, dc=server.example.com-pki-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=drm.example.com:10444,cn=KRAList, ou=Security Domain, dc=server.example.com-pki-ca
objectClass: top
objectClass: pkiSubsystem
cn: drm.example.com:10444
host: server.example.com
SecurePort: 10444
DomainManager: false
Clone: false
SubsystemName: Data Recovery Manager

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 DRM; the DRM 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.
4.4. 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.

4.4.1. 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 4.1, "Initial Subsystem Certificates".
### Table 4.1. Initial Subsystem Certificates

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Certificates</th>
</tr>
</thead>
</table>
| Certificate Manager | - CA signing certificate  
- OCSP signing certificate  
- SSL server certificate  
- Subsystem certificate  
- User's (agent/administrator) certificate  
- Audit log signing certificate |
| RA | - SSL server certificate  
- User's (agent/administrator) certificate |
| OCSP | - OCSP signing certificate  
- SSL server certificate  
- Subsystem certificate  
- User's (agent/administrator) certificate  
- Audit log signing certificate |
| DRM | - Transport certificate  
- Storage certificate  
- SSL server certificate  
- Subsystem certificate  
- User's (agent/administrator) certificate  
- Audit log signing certificate |
| TKS | - SSL server certificate  
- User's (agent/administrator) certificate  
- Audit log signing certificate |
| TPS | - SSL 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, DRMs, OCSPs, TKSs, and TPSs will no longer function, and agents can no longer 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.
4.4.2. 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.

4.4.3. 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.

4.4.4. 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. Certificate System supports six signing algorithms, by default, two in the MD family, four in the SHA family, and one for ECC encryption:

- MD2withRSA
- MD5withRSA
- SHA1withRSA
- SHA256withRSA
- SHA512withRSA
SHA1withEC

SHA1withRSA is the default signing algorithm for CAs for RSA certificates. SHA1withEC is the default signing algorithm for CAs for ECC certificates.

**NOTE**

Certificate System does not include a module natively to enable ECC, but it is possible to load and use a third-party PKCS #11 module with ECC-enabled. This is covered in Chapter 9, *Installing an Instance 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 in the configuration wizard 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.

### 4.4.5. 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 3280, 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.

Before the X.509 v3 standard was finalized, Netscape and other companies had to address some of the most pressing issues with their own extension definitions. For example, applications such as Netscape Navigator and Enterprise Server supported an extension known as the Netscape Certificate Type Extension that specifies the type of certificate issued, such as client, server, or email. To maintain compatibility with older versions of browsers that were released before the X.509 v3 specification was finalized, certain kinds of certificates should include some of these Netscape extensions.

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](http://www.rsa.com).

### 4.4.5.1. Structure of Certificate Extensions

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

```plaintext
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.

### 4.4.6. 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.

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.
The parameters of the default certificate profiles - 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 - can be modified. 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 SSL 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 desired. The input values are set as values in the certificate. The default inputs are not configurable in the Certificate System.

An output specifies how the response page to a successful enrollment is presented. It usually displays the certificate in a user-readable format. The default output shows 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.

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. A profile handles one certificate request, but a single request can contain information for multiple certificates. A PKCS#10 request contains a single public key. One CRMF request can contain multiple public keys, meaning multiple certificate requests. A profile may contain multiple sets of policies, with each set specifying how to handle one certificate request within a CRMF request.

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.

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.

### 4.4.7. Planning Authentication Methods

As implied in Section 4.4.6, “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. There are three ways that the Certificate System can authenticate an entity:

- In **agent-approved** enrollment, end-entity requests are sent to an agent for approval. The agent approves the certificate request.
- In **automatic** enrollment, end-entity requests are authenticated using a plug-in, and then the certificate request is processed; an agent is not involved in the enrollment process.
- In **CMC enrollment**, a third party application can create a request that is signed by an agent and then automatically processed.

A Certificate Manager is initially configured for agent-approved enrollment and for CMC authentication. Automated enrollment is enabled by configuring one of the authentication plug-in modules. More than one authentication method can be configured in a single instance of a subsystem. The HTML registration pages contain hidden values specifying the method used. With certificate profiles, the end-entity enrollment pages are dynamically-generated for each enabled profile. The authentication method
associated with this certificate profile is specified in the dynamically-generated enrollment page.

The authentication process is simple.

1. An end entity submits a request for enrollment. The form used to submit the request identifies the method of authentication and enrollment. All HTML forms are dynamically-generated by the profiles, which automatically associate the appropriate authentication method with the form.

2. If the authentication method is an agent-approved enrollment, the request is sent to the request queue of the CA agent. If the automated notification for a request in queue is set, an email is sent to the appropriate agent that a new request has been received. The agent can modify the request as allowed for that form and the profile constraints. Once approved, the request must pass the certificate profiles set for the Certificate Manager, and then the certificate is issued. When the certificate is issued, it is stored in the internal database and can be retrieved by the end entity from the end-entities page by serial number or by request ID.

3. If the authentication method is automated, the end entity submits the request along with required information to authenticate the user, such as an LDAP username and password. When the user is successfully authenticated, the request is processed without being sent to an agent’s queue. If the request passes the certificate profile configuration of the Certificate Manager, the certificate is issued and stored in the internal database. It is delivered to the end entity immediately through the HTML forms.

The requirements for how a certificate request is authenticated can have a direct impact on the necessary subsystems and profile settings. For example, if an agent-approved enrollment requires that an agent meet the requester in person and verify their identity through supported documentation, the authentication process can be time-intensive, as well as constrained by the physical availability of both the agent and the requester. In that case, having numerous local RAs may be preferable to centralized CAs.

4.4.8. 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 SSL, the directory service has to be configured in SSL 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.

4.4.9. 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.

### 4.5. 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.

#### 4.5.1. 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, DRM, and TKS are always placed inside a firewall because they contain critical information that can cause devastating security consequences if they are compromised.

The RA is frequently placed outside the firewall and the TPS and OCSP can be. 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 Enterprise Linux *Deployment Guide*, such as "Using iptables."
4.5.2. Considering Physical Security and Location

Because of the sensitive data they hold, consider keeping the CA, DRM, 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.

4.5.3. Planning Ports

The subsystems are installed and configured with a separate TCP/IP port for each service, called port separation. Each Certificate System subsystem uses up to five ports:

- A standard port
- An SSL port for end users services
- An SSL port for agent services
- An SSL port for the administrative console or admin services
- A web server port (Tomcat for CA, DRM, OCSP, and TKS subsystems, Apache for the TPS and RA subsystems)

All of the service pages and interfaces described in Section 2.5, “Red Hat Certificate System Services” are connected to using the instance’s URL and the corresponding port number. For example:

https://server.example.com:9444/ca/ee/ca

To access the admin console, the URL specifies the admin port:

pkiconsole https://server.example.com:9445/ca

All agent and admin functions require SSL client authentication. For requests from end entities, the Certificate System listens on both the SSL (encrypted) port and non-SSL ports.

The ports for the different services to use are defined in the server.xml file for the CA, OCSP, DRM, and TKS and in the httpd.conf and nss.conf files for the RA and TPS. If a port is not used, it can be disabled in that file. For example:

```xml
<Service name="Catalina">
    <!-Connector port="9180" ... /-->
    <Connector port="9443" ... /> unused standard port
</Service>
```

Whenever a new instance is installed, it can be configured to have a single SSL port or to use three separate SSL ports for the different interface. Whichever way you choose, 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.
4.6. Tokens for Storing Certificate System Subsystem Keys and Certificates

A token is a hardware or software device that performs cryptographic functions and stores public-key certificates, cryptographic keys, and other data. The Certificate System defines two types of tokens, internal and external, for storing key pairs and certificates that belong to the Certificate System subsystems.

An internal (software) token is a pair of files, usually called the certificate database (cert8.db) and key database (key3.db), that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System automatically generates these files in the filesystem of its host machine when first using the internal token. These files are created during the Certificate System subsystem configuration if the internal token was selected for key-pair generation.

These security databases are located in the /var/lib/instance_name/alias directory.

An external token refers to an external hardware device, such as a smart card or hardware security module (HSM), that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System supports any hardware tokens that are compliant with PKCS #11.

PKCS #11 is a standard set of APIs and shared libraries which isolate an application from the details of the cryptographic device. This enables the application to provide a unified interface for PKCS #11-compliant cryptographic devices.

The PKCS #11 module implemented in the Certificate System supports cryptographic devices supplied by many different manufacturers. This module allows the Certificate System to plug in shared libraries supplied by manufacturers of external encryption devices and use them for generating and storing keys and certificates for the Certificate System managers.

Consider using external tokens for generating and storing the key pairs and certificates used by Certificate System. These devices are another security measure to safeguard private keys because hardware tokens are sometimes considered more secure than software tokens.

Before using external tokens, plan how the external token is going to be used with the subsystem:

- All system keys for a subsystem must be generated on the same token.
- The subsystem must be installed in an empty HSM slot. If the HSM slot has previously been used to store other keys, then use the HSM vendor’s utilities to delete the contents of the slot. The Certificate System has to be able to create certificates and keys on the slot with default nicknames. If not properly cleaned up, the names of these objects may collide with previous instances.

The Certificate System can also use hardware cryptographic accelerators with external tokens. Many of the accelerators provide the following security features:

- Fast SSL connections. Speed is important to accommodate a high number of simultaneous enrollment or service requests.
- Hardware protection of private keys. These devices behave like smart cards by not allowing private keys to be copied or removed from the hardware token. This is important as a precaution against key theft from an active attack of an online Certificate Manager.

The Certificate System supports the nCipher netHSM hardware security module (HSM), by default. Certificate System-supported HSMs are automatically added to the secmod.db database with modutil during the pre-configuration stage of the installation, if the PKCS #11 library modules are in
the default installation paths.

During configuration, the **Security Modules** panel displays the supported modules, along with the NSS internal software PKCS #11 module. All supported modules that are detected show a status of **Found** and is individually marked as either **Logged in** or **Not logged in**. If a token is found but not logged in, it is possible to log in using the **Login** under **Operations**. If the administrator can log into a token successfully, the password is stored in a configuration file. At the next start or restart of the Certificate System instance, the passwords in the password store are used to attempt a login for each corresponding token.

Administrators are allowed to select any of the tokens that are logged in as the default token, which is used to generate system keys.

### 4.7. Implementing a Common Criteria Environment

The **Common Criteria for Information Technology Security Evaluation** is an international standard that helps to define the security aspects and secure implementations of software and hardware. To receive certification, software is evaluated for security in a defined and controlled environment with clearly delineated configuration and environment parameters. This environment is called the **evaluated configuration**.


More information on Common Criteria, evaluation guidelines, and technology security is available at the [National Information Assurance Partnership: Common Criteria Evaluation and Validation Scheme website](https://commoncriteriaportal.nist.gov/).

**Some configuration is required.**

As part of the Common Criteria environment, several otherwise optional features in Certificate System are required to be configured:

- FIPS mode for any hardware security modules used to store key and certification information
- Both secure client connections and secure server connections with the internal LDAP database (meaning all connections to the Red Hat Directory Server are over SSL)
- SSL session timeouts for all secure connections
- Signed audit logging
- Established backup and restore procedures
- CRL checking (OCSP validation) for clients
- **sudo** permissions to run Certificate System scripts and processes
- Defined operating systems users and groups for Certificate System subsystems
- SELinux in enforcing mode
- Removing unused CA interfaces from the `web.xml` file
- Self-test diagnostics, which are enabled by default

These features are listed as part of the installation process and in **Chapter 15, After Configuration: Checklist of Configuration Areas for Deploying Certificate System**.

**The RA is not part of the Common Criteria environment.**

All of the Red Hat Certificate System subsystems (CA, DRM, OCSP, TKS, and TPS) were targets of evaluation for the Common Criteria certification process, with the exception of the RA. The RA is not considered part of a Common Criteria-certified environment. If your environment requires only Common
Criteria-certified components, then the RA cannot be deployed.

4.8. 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 SSL port, or is it better to have port separation for extra security?

A: The most secure solution is to use separate ports for each SSL interface. However, the feasibility of implementing multiple ports may depend 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 machines which a subsystem should be installed on depend on the network design. The RA and OCSP subsystems are specifically designed to operate outside a firewall for user convenience, while the CA, DRM, and TPS should all be secured behind a firewall for increased security.

When deciding what machines to install the subsystems on, 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 DRM 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 DRM 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 RA and 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 DRMs, 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 netHSM 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 be 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 4.4.6, “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 SSL 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 local offices need to process their own certificate requests? Will there be a large number of certificate requests?**

A: Even if there are centralized CAs, a degree of local control can be given by using local RAs to process user certificate requests. For geographically diverse PKIs, for organizations that require local control over certificate issuance, and for high-security areas which require in-person enrollments, using an RA can make the certificate enrollment process faster and more adaptable. It can also take the load off CAs by separating the approval step from certificate issuance, which helps for high-volume PKI deployments.

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 doesn't 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 DRM 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 DRM to archive the keys so that the token's certificates can be regenerated.

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 can define the publishing policy.
Part II. Installing Red Hat Certificate System

This section describes the requirements and procedures for installing Red Hat Certificate System, both for normal operating environments and Common Criteria-certified environments.
Chapter 5. Prerequisites and Preparation for Installation

Before installing the Red Hat Certificate System subsystems, check out the requirements and dependencies for the specific platform, as well as looking at the installed packages.

5.1. Supported Platforms, Hardware, and Programs

5.1.1. Supported Platforms

The Certificate System subsystems (CA, RA, DRM, OCSP, TKS, and TPS) are supported on the following platforms:

- Red Hat Enterprise Linux 5.6 and later (x86, 32-bit)
- Red Hat Enterprise Linux 5.6 and later (x86_64, 64-bit)

The Enterprise Security Client, which manages smart cards for end users, is supported on the following platforms:

- Red Hat Enterprise Linux 5.6 and later (x86, 32-bit)
- Red Hat Enterprise Linux 5.6 and later (x86_64, 64-bit)
- Microsoft Windows Vista 32-bit
- Microsoft Windows Vista 64-bit
- Microsoft Windows XP 32-bit
- Microsoft Windows XP 64-bit
- Apple Mac OS X 10.5.x (Leopard)

5.1.2. Supported Web Browsers

The services pages for the subsystems require a web browser that supports SSL. It is strongly recommended that users such as agents or administrators use Mozilla Firefox to access the agent services pages. Regular users should use Mozilla Firefox or Microsoft Internet Explorer.

NOTE

The only browser that is fully-supported for the HTML-based instance configuration wizard is Mozilla Firefox.
Table 5.1. Supported Web Browsers by Platform

<table>
<thead>
<tr>
<th>Platform</th>
<th>Agent Services</th>
<th>End User Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>Firefox 10 and later</td>
<td>Firefox 10 and later</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>Firefox 10 and later</td>
<td>Internet Explorer 7 and higher</td>
</tr>
<tr>
<td>Windows XP</td>
<td>Firefox 10 and later</td>
<td>Internet Explorer 6 and higher</td>
</tr>
<tr>
<td>Mac OS 10.5.x</td>
<td>Agent services are not supported for Mac</td>
<td>Firefox 10 and later</td>
</tr>
</tbody>
</table>

5.1.3. Supported Smart Cards

The Enterprise Security Client supports Global Platform 2.01-compliant smart cards and JavaCard 2.1 or higher.

The Certificate System subsystems have been tested using the following tokens:

- Gemalto TOP IM FIPS CY2 64K token, both as a smart card and GemPCKey USB form factor key
- Gemalto Cyberflex e-gate 32K token
- Safenet 330J Java smart card

Smart card testing was conducted using the SCM SCR331 CCID reader.

The only card manager applet supported with Certificate System is the CoolKey applet which ships with Red Hat Enterprise Linux 5.6.

5.1.4. Supported HSM

Red Hat Certificate System supports three hardware security modules (HSM): nCipher netHSM, nCipher sShield, and Chrysalis-IT LunaSA.

<table>
<thead>
<tr>
<th>HSM</th>
<th>Firmware</th>
<th>Appliance Software</th>
<th>Client Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safenet Chrysalis-ITS</td>
<td>4.5.2</td>
<td>3.2.4</td>
<td>3.2.4</td>
</tr>
<tr>
<td>LunaSA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nCipher netHSM 2000</td>
<td>2.33.60</td>
<td>11.10</td>
<td></td>
</tr>
<tr>
<td>nCipher sShield</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1.5. Supported Charactersets

Red Hat Certificate System fully supports UTF-8 characters in the CA end users forms for specific fields. This means that end users can submit certificate requests with UTF-8 characters in those fields and can search for and retrieve certificates and CRLs in the CA and retrieve keys in the DRM when using those field values as the search parameters.

Four fields fully-support UTF-8 characters:

- Common name (used in the subject name of the certificate)
- Organizational unit (used in the subject name of the certificate)
5.1.6. Summary of Requirements for Common Criteria

Red Hat Certificate System 8.1 is certified for Common Criteria on a defined environment. It is possible to install, configure, and use Certificate System in other environments, but to have a Common Criteria-certified environment, it must meet these requirements for software and hardware.

Table 5.2. Common Criteria Environment

<table>
<thead>
<tr>
<th>Requirement Area</th>
<th>Certified Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystems</td>
<td>CA</td>
</tr>
<tr>
<td></td>
<td>DRM</td>
</tr>
<tr>
<td></td>
<td>OCSP</td>
</tr>
<tr>
<td></td>
<td>TKS</td>
</tr>
<tr>
<td></td>
<td>TPS</td>
</tr>
<tr>
<td>Operating System</td>
<td>Red Hat Enterprise Linux 5.6 and later, 32-bit</td>
</tr>
<tr>
<td></td>
<td>Red Hat Enterprise Linux 5.6 and later, 64-bit</td>
</tr>
<tr>
<td>JDK/JRE</td>
<td>OpenJDK Runtime Environment 1.6.0.0</td>
</tr>
<tr>
<td>Internal Database</td>
<td>Red Hat Directory Server 8.2</td>
</tr>
<tr>
<td></td>
<td>Red Hat Directory Server 9.0</td>
</tr>
<tr>
<td>Web Server</td>
<td>Tomcat 5 (CA, DRM, OCSP, TKS)</td>
</tr>
<tr>
<td></td>
<td>Apache 2.2 (TPS)</td>
</tr>
<tr>
<td>Hardware Security Modules or Tokens</td>
<td>Any properly-certified HSM, running in FIPS 140 Level 3 mode</td>
</tr>
<tr>
<td>Web Browser [a]</td>
<td>Firefox 10 and later</td>
</tr>
</tbody>
</table>

[a] To access the configuration wizard and agent and administrative interfaces.

NOTE

This support does not include supporting internationalized domain names, like in email addresses.
5.2. Packages Installed on Red Hat Enterprise Linux

Multiple packages are installed with the Certificate System, in addition to the core Certificate System components.

### RPMs for Certificate System Subsystems and Components

<table>
<thead>
<tr>
<th>Package</th>
<th>Subsystem/Component</th>
<th>RPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>osutil</td>
<td></td>
<td>pki-ocsp</td>
</tr>
<tr>
<td>pki-ca</td>
<td></td>
<td>pki-ra</td>
</tr>
<tr>
<td>pki-common</td>
<td></td>
<td>pki-selininux</td>
</tr>
<tr>
<td>pki-common-javadoc</td>
<td></td>
<td>pki-setup</td>
</tr>
<tr>
<td>pki-console</td>
<td></td>
<td>pki-silent</td>
</tr>
<tr>
<td>pki-java-tools</td>
<td></td>
<td>pki-tks</td>
</tr>
<tr>
<td>pki-java-tools-javadoc</td>
<td></td>
<td>pki-tps</td>
</tr>
<tr>
<td>pki-kra</td>
<td></td>
<td>pki-util</td>
</tr>
<tr>
<td>pki-migrate</td>
<td></td>
<td>pki-util-javadoc</td>
</tr>
<tr>
<td>pki-native-tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMs for Tomcat Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ant</td>
<td></td>
<td>jakarta-commons-discovery</td>
</tr>
<tr>
<td>avalon-framework</td>
<td></td>
<td>jakarta-commons-el</td>
</tr>
<tr>
<td>avalon-logkit</td>
<td></td>
<td>jakarta-commons-fileupload</td>
</tr>
<tr>
<td>axis</td>
<td></td>
<td>jakarta-commons-httpclient</td>
</tr>
<tr>
<td>bcel</td>
<td></td>
<td>jakarta-commons-launcher</td>
</tr>
<tr>
<td>classpathx-jaf</td>
<td></td>
<td>jakarta-commons-logging</td>
</tr>
<tr>
<td>classpathx-mail</td>
<td></td>
<td>jakarta-commons-modeler</td>
</tr>
<tr>
<td>eclipse-ecj</td>
<td></td>
<td>jakarta-commons-pool</td>
</tr>
<tr>
<td>geronimo-specs</td>
<td></td>
<td>jdom</td>
</tr>
<tr>
<td>xalan-j2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>geronimo-specs-compat</td>
<td></td>
<td>jakarta-commons-beanutils</td>
</tr>
<tr>
<td>jakarta-commons-collections</td>
<td></td>
<td>ldapjdk</td>
</tr>
<tr>
<td>classpathx-jaf</td>
<td></td>
<td>jakarta-commons-logging</td>
</tr>
<tr>
<td>classpathx-mail</td>
<td></td>
<td>jakarta-commons-modeler</td>
</tr>
<tr>
<td>eclipse-ecj</td>
<td></td>
<td>jakarta-commons-pool</td>
</tr>
<tr>
<td>geronimo-specs</td>
<td></td>
<td>jdom</td>
</tr>
<tr>
<td>xalan-j2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>geronimo-specs-compat</td>
<td></td>
<td>jakarta-commons-beanutils</td>
</tr>
<tr>
<td>jakarta-commons-collections</td>
<td></td>
<td>ldapjdk</td>
</tr>
<tr>
<td>geronimo-specs</td>
<td></td>
<td>jdom</td>
</tr>
<tr>
<td>xalan-j2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jakarta-commons-logging</td>
<td></td>
<td>tomcat5-common-server</td>
</tr>
<tr>
<td>classpathx-jaf</td>
<td></td>
<td>jakarta-commons-logging</td>
</tr>
<tr>
<td>classpathx-mail</td>
<td></td>
<td>jakarta-commons-modeler</td>
</tr>
<tr>
<td>eclipse-ecj</td>
<td></td>
<td>jakarta-commons-pool</td>
</tr>
<tr>
<td>rpm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mod_nss</td>
<td></td>
<td>perl-XML-SAX</td>
</tr>
<tr>
<td>mod_perl</td>
<td></td>
<td>perl-XML-Simple</td>
</tr>
<tr>
<td>classpathx-jaf</td>
<td></td>
<td>perl-XML-NamespaceSupport</td>
</tr>
<tr>
<td>classpathx-mail</td>
<td></td>
<td>perl-XML-Parser</td>
</tr>
<tr>
<td>cyrus-sasl</td>
<td></td>
<td>moz1dap-devel</td>
</tr>
<tr>
<td>RPMs for LDAP Support</td>
<td></td>
<td>moz1dap-tools</td>
</tr>
<tr>
<td>RPMs for SQL Lite Support for the RA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.3. Before Installation: Setting up the Operating Environment

To install any Red Hat Certificate System subsystems on Red Hat Enterprise Linux, three programs are required: OpenJDK, Apache or Tomcat (depending on the subsystem), and Red Hat Directory Server. All other required packages should be present as part of the base Red Hat Enterprise Linux operating system packages.

The system itself must be configured in certain ways to ensure that the packages can be properly installed and the instances created. There are several factors that must be in place, depending on the planned deployment:

- SELinux should be enabled.
- A system user must be created. The Certificate System instance will run as this user.
- The system should have a Java Security Manager running to manage the Java-based subsystems.
- Any external hardware tokens that will be used to store subsystem certificates and keys must be installed, configured, and running before the subsystems are created.
- Check for any Fedora EPEL repos in /etc/yum.repos.d/ directory; these are usually named epel.repo or epel-testing.repo. Either remove these repo files or disable the repos (setting the enabled line to zero, enabled=0). Disabling the EPEL repos prevents any EPEL packages from overriding the official Red Hat Enterprise Linux packages.

5.3.1. Installing the Required Java Development Kit (JDK)

Certificate System requires OpenJDK 1.6.0. On Red Hat Enterprise Linux systems, this must be installed separately. The OpenJDK can be installed by using yum or by downloading the packages directly from http://openjdk.java.net/install/. For example:

```
yum install java-1.6.0-openjdk
```

After installing the JDK, run /usr/sbin/alternatives as root to insure that the proper JDK is available:

```
/usr/sbin/alternatives --config java
```

<table>
<thead>
<tr>
<th>Selection</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/usr/lib/jvm/jre-1.4.2-gcj/bin/java</td>
</tr>
<tr>
<td>+ 2</td>
<td>/usr/lib/jvm/jre-1.6.0-openjdk/bin/java</td>
</tr>
<tr>
<td>* 3</td>
<td>/usr/lib/jvm/jre-1.6.0-sun.x86_64/bin/java</td>
</tr>
</tbody>
</table>

5.3.2. Installing Apache (for the TPS)

Apache 2.x must be installed in order to install the TPS subsystem. Check that the appropriate version of Apache is installed.
yum info httpd
Installed Packages
Name   : httpd
Arch   : x86_64
Version: 2.2.3
Release: 22.el5_3.2
Size   : 2.9 M
Repo   : installed
...

Install Apache if it is not already available. For example:

yum install httpd

5.3.3. Installing Red Hat Directory Server

All subsystems require access to Red Hat Directory Server. This Directory Server instance is used by the subsystems to store their system certificates and user data.

» Either Red Hat Directory Server 8.2 or 9.0 can be used.
» The Directory Server instance can be installed on the local system or on a remote system.
» Directory Server 8.2 instances can be installed on Red Hat Enterprise Linux 5 32-bit, Red Hat Enterprise Linux 5 64-bit, or Solaris 9 Sparc 64-bit.
» Directory Server 9.0 instances can be installed on Red Hat Enterprise Linux 6 32-bit or 64-bit systems.
  The Directory Server can be installed on any supported platform for its version, regardless of what platform the Certificate System is installed on.

Check that the Red Hat Directory Server is already installed. For example:

[root@server ~]# yum info redhat-ds
Installed Packages
Name       : redhat-ds
Arch       : x86_64
Version    : 8.2.0
Release    : 0.14el5dsrv
Size       : 136M
Repo       : installed
...

Install and configure Red Hat Directory Server, if a directory service is not already available. For example:

[root@server ~]# yum install redhat-ds
[root@server ~]# setup-ds-admin.pl

Go through the configuration wizard; the default settings are fine for the Certificate System needs.

Installing Red Hat Directory Server is described in more detail in the Red Hat Directory Server Installation Guide.
The Certificate System SELinux policies assume that the Red Hat Directory Server is listening over the standard LDAP/LDAPS ports, 389 and 636, respectively. If the Directory Server is using non-standard ports, then edit the SELinux policy using `semanage` to relabel the LDAP/LDAPS ports and allow the subsystem to access the Directory Server.

Syntax checking is enabled by default in Directory Server 9.0, but it must be disabled for TPS enrollments to work properly.

```
[jsmith@server ~]$ ldapmodify -D "cn=Directory Manager" -w secret -h ds-server.example.com -p 389 -x

dn: cn=config
changetype: modify
modify: nsslapd-syntaxcheck
nsslapd-syntaxcheck: off
```

### 5.3.4. Installing Additional Operating System Packages

The following package groups and packages must be installed on all Red Hat Enterprise Linux systems:

- gnome-desktop (package group)
- compat-arch-support (package group)
- web-server (package group)
- kernel-smp (package)
- e2fsprogs (package)
- firefox (package)

To verify that the packages are installed, just run `rpm -qi` For example:

```
rpm -qi gnome-desktop
gnome-desktop-2.16.0-1.el5
```

On 64-bit Red Hat Enterprise Linux platforms, be certain that the 64-bit (x86_64) `compat-libstdc++` libraries are installed, and not only the 32-bit (i386) libraries. To confirm this, run the following as `root`:

```
rpm -qi compat-libstdc++ --queryformat '%{NAME}-%{VERSION}-%{RELEASE}.%{ARCH}.rpm\n' | grep x86_64
```

Numerous libraries should be displayed.

### 5.3.5. Verifying Firewall Configuration and iptables

Any firewalls must be configured to allow access to the Certificate System ports and to any other applications, like Red Hat Directory Server, which are required for the operation of the subsystems. Use caution when configuring the firewall, so that the system remains secure.
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 Enterprise Linux Deployment Guide, such as "Using iptables." Installing the subsystems will fail unless iptables is turned on and properly configured.

### 5.3.6. Enabling SELinux

SELinux policies for Certificate System subsystems are installed as a dependency for Certificate System 8.1, in the **pki-selinux** package. The SELinux policies are automatically configured whenever a new instance is created by the **pkicreate** command.

Red Hat recommends running Certificate System with SELinux in **enforcing** mode, to make the most of the security policies.

The Certificate System SELinux policies assume that the Red Hat Directory Server is listening over the standard LDAP/LDAPS ports, 389 and 636, respectively. If the Directory Server is using non-standard ports, then edit the SELinux policy using **semanage** to relabel the LDAP/LDAPS ports and allow the subsystem to access the Directory Server.

If SELinux is set to **enforcing**, then any external modules or hardware which interact with the subsystems must be configured with the proper SELinux settings to proceed with subsystem installation:

- Third-party modules, such as for ECC or HSM, must have an SELinux policy configured for them, or SELinux needs to be changed from **enforcing** mode to **permissive** mode to allow the module to function. Otherwise, any subsystem operations which require the ECC module will fail.

  Changing SELinux policies is covered in the **SELinux section in the Red Hat Enterprise Linux Deployment Guide**.

- SELinux policies must be set for any nCipher netHSM modules, as described in **Section 8.1.2.4, “Setting up SELinux on nCiper.netHSM 2000”**.

**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 the **SELinux section in the Red Hat Enterprise Linux Deployment Guide**.

Red Hat Certificate System has its own SELinux policies defined in a special package, **pki-selinux**, which is a required dependency for each subsystem. This policy defines objects, domains, and rules for each subsystem type, and those policies apply equally to every instance of that type on that machine. The rules and definitions for all the subsystems comprise the overall Certificate System SELinux policy.

The Certificate System subsystems run with SELinux set in enforcing mode, meaning that Certificate System operations can be successfully performed even when all SELinux rules are required to be
followed. All SELinux policies are updated every time a subsystem is added with `pkicreate` or removed with `pkiremove`.

To make sure SELinux is in the proper mode:

1. Open the **Systems** menu.
2. Open the **Administration** menu, and select the **SELinux Management** item.

3. In the **Status** area, set the system default and current enforcing modes to the desired setting. **Enforcing** mode is recommended.

5.3.7. Setting up Operating System Users and Groups
Certificate System uses operating system user and groups to run the subsystem processes. The groups used by Certificate System must be created on the operating systems before the packages are installed and any operating system users must be created or associated with those groups.

**NOTE**

The administrator who creates these groups and users must have the required access to the operating system and any associated programs (like NIS).

### 5.3.7.1. Creating Operating System Groups

Certificate System uses three and possibly four operating system groups:

- pkiuser
- pkiaudit
- A hardware token group, such as nfast (optional)

The first group, **pkiuser**, is used by the Certificate System subsystems; this is the user which the subsystem daemons run as. The other two groups, **pkiaudit** and **pkiaudit**, are used by Certificate System users who manage the subsystem instances. If the subsystem uses a hardware token, then the PKI administrator users must also belong to that group, such as **nfast** for an nCipher token.

All of the PKI groups are system accounts. They must meet certain criteria as system accounts:

- They must have a GID and UID lower than 500. It is strongly recommended that the **pkiuser** group has a GID and UID of 17. On Red Hat Enterprise Linux 5.6 and later systems, the **pkiuser** group is already configured and has a GID of 17.
- The groups must not have a login shell, meaning that the login shell has a value of `/sbin/nologin`.
- All PKI groups must be created before attempting to install any subsystem. This account must be present in `/etc/group`.
- The PKI user should be create before installing any subsystems. This account must be present in...
Both the pkiadmin and pkiaudit groups must be created for Certificate System. This is done using the groupadd tool, which is described in the SELinux section in the Red Hat Enterprise Linux Deployment Guide.

1. For Red Hat Enterprise Linux 5.6 (and later) systems, the pkiuser group is already created. This can be verified by checking the /etc/group file:

```
    grep pkiuser /etc/group
    pkiuser:x:17:
```

If the pkiuser group does not exist, then make sure that the appropriate tool packages are installed:

```
    # rpm -q setup
    setup-2.5.58-7.el5
    # rpm -q shadow-utils
    shadow-utils-4.0.17-15.el5
```

Then, if the pkiuser group does not exist or if it has a GID other than 17, then create the pkiuser group. This group must have a GID value of 17; this can be specified using the -g option.

```
    # userdel pkiuser
    # groupdel pkiuser
    # groupadd -g 17 -r pkiuser
```

2. Create the pkiadmin group. This group can have any randomly assigned GID for a system account. Use the -r option to create a system group.

```
    # groupadd -r pkiadmin
```

3. Create the pkiaudit group. This group can have any randomly assigned GID for a system account. Use the -r option to create a system group.

```
    # groupadd -r pkiaudit
```

4. Assign user accounts to the group so that users can perform the administrative and audit tasks for the subsystems. (If necessary, also create users for the groups.) This is described in Section 5.3.7.2, “Creating Operating System Users”.

```
    # usermod -a -G pkiadmin bjensen
```

Along with assigning regular users to the pkiadmin and pkiaudit groups, be sure to add the pkiuser system user account.

TIP

Using groupadd or the Red Hat Enterprise Linux UI tools updates all of the group files on the system, including /etc/group, /etc/gshadow, and /etc/login.defs.
5.3.7.2. Creating Operating System Users

As with system groups, Certificate System uses a system user account for its subsystem process. This is the **pkiuser** account, which is associated with the **pkiuser** system group.

The other Certificate System groups — **pkiadmin** and **pkiaudit** — allow real users (not system users) to be members so that those users can carry out normal administrative and auditing functions. These user accounts simply need to be added to the PKI groups, as described in Section 5.3.7.2.3, “Associating Existing User Accounts with PKI Groups”.

5.3.7.2.1. Checking the pkiuser System Account

On Red Hat Enterprise Linux 5.6 (and later) machines, the **pkiuser** account already exists; this can be verified by checking the `/etc/passwd` file:

```
# grep pkiuser /etc/passwd
pkiuser:x:17:
```

As with the **pkiuser** group, the **pkiuser** account must have a UID number of 17. If the **pkiuser** account does not exist or if it does not have a UID of 17, then check that the appropriate setup packages are installed:

```
# rpm -q setup
setup-2.5.58-7.el5
# rpm -q shadow-utils
shadow-utils-4.0.17-15.el5
```

Then create the **pkiuser** user. Use the `-g` option to give the group to add the user to, and use the `-r` option to create a system account. To set the UID explicitly, use the `-u` option.

```
# userdel pkiuser
# useradd -g pkiuser -d /usr/share/pki -s /sbin/nologin -c "Red Hat Certificate System" -u 17 -r pkiuser
```

5.3.7.2.2. Creating New User Accounts

Other PKI Users, associated with the administrator and auditor groups, can be regular users rather than system accounts. When creating new users, always use the system tools, like **useradd** or the UI tools, because those automatically update all system files related to users, including `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/gshadow`, `/etc/default/useradd`, `/etc/skel`, and `/etc/login.defs`.

The process and options for adding users is described in the Red Hat Enterprise Linux 5 documentation at [http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/s2-users-add.html](http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/s2-users-add.html). When users are created, they can simultaneously be associated with a group using the `-g` option. For example, this creates a **jsmith** user who belongs to the **pkiadmin** group, and then creates the user password:
5.3.7.2.3. Associating Existing User Accounts with PKI Groups

Existing users can be added to a PKI group using the `usermod` command. As with `useradd`, `usermod` updates all of the related user account files, including `/etc/passwd`, `/etc/shadow`, and `/etc/group`.

```bash
# usermod -a -G pkiadmin bjensen
```

**NOTE**

The users added to the admin and audit groups are regular user accounts, not a system account like `pkiuser`.

Add the user accounts to the appropriate PKI management group, *and only to that one group*. Users are either an administrator or an auditor; the same user cannot be in both groups.

- PKI auditors only need to be added to the `pkiaudit` group.
- PKI administrators need to be added to the `pkiadmin` group and any group uses by a hardware token used by the subsystem, such as `nfast` for an nCipher hardware token.

The `pkiuser` user should be added to both the `pkiadmin` and `pkiaudit` groups:

```bash
# usermod -a -G pkiadmin pkiuser
# usermod -a -G pkiaudit pkiuser
```

### 5.3.8. Using a Java Security Manager

<table>
<thead>
<tr>
<th>Deployment Suggestions</th>
<th>Relevant PKI Subsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional</td>
<td>CA, DRM</td>
</tr>
<tr>
<td>Recommended</td>
<td>OCSP</td>
</tr>
<tr>
<td>Mandatory</td>
<td>RA, TKS</td>
</tr>
<tr>
<td>Common Criteria Required</td>
<td>TPS</td>
</tr>
</tbody>
</table>

**Exceptions**

The Java Security Manager is configured to be on by default unless an instance is specifically created by `pkicreate` using the `-sans_security_manager` option.
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 `-secure` flag is set in Tomcat).

The Security Manager is disabled if the instance is created with the `-sans_security_manager` option.
Chapter 6. Installing and Configuring Certificate System

The Certificate System is comprised of subsystems which can be independently installed on different servers, multiple instances installed on a single server, and other flexible configurations for availability, scalability, and failover support. The procedures for downloading, installing, and configuring instances of Certificate System subsystems are described in this chapter.

There are different paths for the installation process, depending on the planning decisions that you made and the needs of your environment.

The Certificate System servers include six subsystems:

- Certificate Authority (CA)
- Registration Authority (RA)
- Data Recovery Manager (DRM), sometimes referred to as a Key Recovery Authority (KRA)
- Online Certificate Status Protocol (OCSP) Responder
- Token Key Service (TKS)
- Token Processing System (TPS)

Each subsystem is installed and then configured individually. The order in which subsystems are configured is very important because of the basic relationships which are established between subsystems at the time they are installed. For example, every subsystem depends on a certificate authority; the TPS also depends on a TKS and (optionally) DRM.
The installation process includes not only setting up the individual subsystems but also setting up the environment. The environment configuration is flexible and largely optional; the configuration that you select should depend on the existing network environment and security requirements.

The complete subsystem setup process includes the preparation for the environment, the instance creation and setup, and then the configuration of major features for each subsystem.

This chapter covers the basic installation procedures for general PKI subsystems and the token management system. Other installation options are covered in other chapters:

- Chapter 7, Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server
- Chapter 8, Using Hardware Security Modules for Subsystem Security Databases
- Chapter 9, Installing an Instance with ECC Enabled
- Chapter 10, Cloning Subsystems
- Chapter 11, Silently Configuring Instances
- Chapter 12, Additional Installation Options
6.1. About pkicreate

Certificate System subsystem instances are created and defined using a script called `pkicreate`. This script creates individual subsystem instances, with user-defined settings like the configuration and log directories and port numbers. After the instance is created, it is then configured through the HTML-based configuration wizard or by using the `pkisilent` script.

The syntax for `pkicreate` is slightly different between subsystems because of the different port and groups configurations. **Table 6.1, “pkicreate Parameters”**

---

**TIP**

To get full usage examples and syntax for the `pkicreate` command, run `pkicreate --help`.  

---

113
### Table 6.1. pkicreate Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pki_instance_root</td>
<td>Gives the full path to the new instance configuration directory.</td>
</tr>
<tr>
<td>subsystem_type</td>
<td>Gives the type of subsystem being created.</td>
</tr>
<tr>
<td>pki_instance_name</td>
<td>Gives the name of the new instance. Instance names must be unique on a single machine, but do not have to be unique within the security domain. (since instances are identified by hostname and port, not instance name).</td>
</tr>
<tr>
<td>secure_port[a]</td>
<td>Sets a single SSL port number for the subsystem. This parameter is required if port separation is not configured, meaning that separate ports are not assigned for the administrator, agent, and end-entities services.</td>
</tr>
<tr>
<td>agent_secure_port[a]</td>
<td>Sets the SSL port for the agent web services. If this is specified, then both ee_secure_port and admin_secure_port must be specified. For CAs only, an end-entities client authentication port is also required with the ee_secure_client_auth_port option.</td>
</tr>
<tr>
<td>ee_secure_port[a]</td>
<td>Sets the SSL port for the end-entities web services. If this is specified, then both agent_secure_port and admin_secure_port must be specified. For CAs only, an end-entities client authentication port is also required with the ee_secure_client_auth_port option.</td>
</tr>
<tr>
<td>ee_secure_client_auth_port[a]</td>
<td>For CAs only. Sets the SSL port for the end-entity client authentication. If this is specified, then ee_secure_port, agent_secure_port, and admin_secure_port must be specified.</td>
</tr>
<tr>
<td>admin_secure_port[a]</td>
<td>Sets the SSL port number for the administrator services, usually the pkiconsole. If this is specified, then both agent_secure_port and ee_secure_port must be specified. For CAs only, an end-entities client authentication port is also required with the ee_secure_client_auth_port option.</td>
</tr>
<tr>
<td>non_clientauth_secure_port[a]</td>
<td>Sets the end entities SSL port for RA and TPS subsystems.</td>
</tr>
<tr>
<td>unsecure_port[a]</td>
<td>Sets the regular port number. If this is not set, the number is randomly generated. Still, it is recommended that administrators set this value to make sure there are no conflicts with SELinux labels for other services.</td>
</tr>
<tr>
<td>tomcat_server_port[a]</td>
<td>Sets the port number for the Tomcat web server for CA, OCSP, TKS, and DRM instances.</td>
</tr>
<tr>
<td>redirect conf</td>
<td>Optional. Sets the location for the configuration</td>
</tr>
</tbody>
</table>
files for the new instance. This should include an instance-specific directory name in the path. For example, for the `pki-ca` instance, this should be something like `/etc/pki-ca`.

### redirect logs

*Optional.* Sets the location for the log files for the new instance. This should include an instance-specific directory name in the path. For example, for the `pki-ca` instance, this should be something like `/var/log/pki-ca`.

### user

*Optional.* Sets the user as which the Certificate System instance will run.

### group

*Optional.* Sets the group as which the Certificate System instance will run.

### audit_group

*Optional.* Gives the name of the group for auditors for the TPS instance. The default is `pkiaudit`, if this option is not given.

### sans_security_manager

*Optional.* For the CA, OCSP, DRM, and TKS. Configures the new instance to run without a Java Security Manager. This option should not be used for subsystems in a Common Criteria environment.

---

For more information on the `pkicreate` tool options, see the *Certificate System Command-Line Tools Guide*.

### 6.2. Basic Installation

There are three major steps to be taken when setting up Certificate System. The first (covered in Chapter 5, *Prerequisites and Preparation for Installation*) goes through setting up the machine and the environment that will host the subsystem instance — making sure that the platform meets requirements, installing necessary applications and packages, and setting up the operating system. The next step involves creating and configuring the subsystem instance itself. The last step involves customizing the instance by setting up recommended features ([Chapter 15, *After Configuration: Checklist of Configuration Areas for Deploying Certificate System*]).

This walk-through simply shows, at a very high level, the major steps for setting up a functional PKI. The exact configuration, like what subsystem types are installed and the desired post-installation configuration, are dependent on the specific PKI design that you developed as part of planning your Certificate System deployment.

1. **Install a Red Hat Directory Server**, as described in Section 5.3.3, “Installing Red Hat Directory Server”. This can be on a different machine from the Certificate System, which is the recommended scenario for most deployments.
2. **Create new, specific operating system groups for the Certificate System subsystems to run as**. This is described in Section 5.3.7.1, “Creating Operating System Groups”.
3. **Assign users to the operating system groups to perform the subsystem administrative tasks**. This is described in Section 5.3.7.2.3, “Associating Existing User Accounts with PKI Groups”.
4. **Download the Certificate System packages from the Red Hat Network channel**.
5. **Install the packages**.
6. Run **pkicreate** to create the subsystem instances.

7. Configure the Certificate System CA subsystem. At least one CA subsystem *must* be installed and fully configured before any other type of subsystem can be configured.

8. Configure the RA, OCSP, and DRM subsystems. Once the CA is installed, the other subsystems can be installed and configured in any order.

### 6.2.1. Installing and Configuring a CA

1. Set up the required **yum** repositories.
   a. Log into the Customer Portal.
   b. Open the **Downloads** tab, and click the **Downloads** box in the main window.
   c. Under any product group, click the **Download Software** button.
   d. Search for the product name **Certificate System** or select the **Red Hat Certificate System** product from the drop-down menu.
   e. Click the link for the appropriate architecture.
   f. On the **Downloads** tab for the Certificate System release, click the **Binary Disc** link and save the ISO image to media.
g. On the machine which hosts the repository, if necessary, install the VSFTP daemon. For example:

```
[root@server ~]# yum install vsftpd
```

h. On the machine which hosts the repository, mount the media with the ISO. For example:

```
[root@server ~]# mount /dev/cdrom /mnt
```

i. On the machine which hosts the repository, create a directory for the Certificate System packages.

```
[root@server ~]# mkdir /var/pub/rhcsrepo
```

j. Copy the Certificate System ISO into the new repository directory.

k. Create the `yum` repository, pointing to the RPM directory in the ISO.

```
[root@server ~]# createrepo -g /var/pub/rhcsrepo/RedHat/RPMS
```

l. Start the `vsftpd` service.

```
[root@server ~]# service vsftpd start
[root@server ~]# chkconfig vsftpd on
```

m. On each client machine, create a `.repo` file with the repository information. For example:
To use an IPv6 hostname for configuration, set the hostname in the `PKI_HOSTNAME` environment variable before installing the packages. This is described in Section 12.3, “Enabling IPv6 for a Subsystem”.

3. Run `yum` to install the CA packages. Optionally, include the console packages.

```bash
[root@server ~]# yum install pki-ca pki-console
```

4. Run the `pkicreate` command to create the CA instance. For example:

```bash
[root@server ~]# pkicreate -pki_instance_root=/var/lib
   -pki_instance_name=pki-ca
   -subsystem_type=ca
   -agent_secure_port=9443
   -ee_secure_port=9444
   -ee_secure_client_auth_port=9446
   -admin_secure_port=9445
   -unsecure_port=9180
   -tomcat_server_port=9701
   -audit_group=pkiaudit
   -redirect logs=/var/log/pki-instance_name/logs

PKI instance creation Utility ...

PKI instance creation completed ...

Starting `instance_name`: [ OK ]

instance_name (pid 17990) is running ...

'instance_name' must still be CONFIGURED!
(see /var/log/pki-instance_name-install.log)

Before proceeding with the configuration, make sure the firewall settings of this machine permit proper access to this subsystem.

Please start the configuration by accessing:

http://server.example.com:9180/ca/admin/console/config/login?pin=kI7E1MByNIUcPJ6RKHmH

After configuration, the server can be operated by the command:

```
service `instance_name` start | stop | restart
```

This example uses the recommended port separation configuration, specifies an auditor group, and uses a Java Security Manager. Other options could be specified to set user-defined log and
configuration directories and a user-defined operating system user and group. For other `pkicreate` options, see Table 6.1, "pkicreate Parameters".

The command options here are on separate lines to make it clear what options are used. All options should be on a single line.

5. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

6. When the `pkicreate` command completes, it returns a URL to use to access the web-based configuration wizard and a PIN to use to authenticate. Open the configuration wizard using the URL returned from the package installation.

Alternatively, log into the setup wizard through the admin link on the services page and supply the `preop.pin` value from the `/var/lib/pki-ca/conf/CS.cfg` file when prompted.

```
https://server.example.com:9444/ca/services
```

7. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given.

Any hardware tokens used with the instance must be configured before configuring the subsystem instance.

---

**Key Store**

Two lists of security modules are provided below. The **Supported Security Modules** list consists of both software-based and hardware-based security modules that this PKI solution supports, while the **Other Security Modules** list consists of any other security modules found by this PKI subsystem that are not recognized as one of the supported security modules. [Details]

### Supported Security Modules

<table>
<thead>
<tr>
<th>Module/Token</th>
<th>Status</th>
<th>Default</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSS Internal PKCS #11 Module</td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Internal Key Storage Token</td>
<td>Logged In</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nCipher's nFast Token Hardware Module</td>
<td>Found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- accelerator</td>
<td>Not logged In</td>
<td></td>
<td>Login</td>
</tr>
<tr>
<td>- nethsm</td>
<td>Not logged In</td>
<td></td>
<td>Login</td>
</tr>
<tr>
<td>SafeNet's LunaSA Token Hardware Module</td>
<td>Found</td>
<td></td>
<td>Login</td>
</tr>
<tr>
<td>- lunasa1-ca</td>
<td>Not logged In</td>
<td></td>
<td>Login</td>
</tr>
<tr>
<td>- lunasa2-ca</td>
<td>Not logged In</td>
<td></td>
<td>Login</td>
</tr>
</tbody>
</table>

### Other Security Modules

The security modules listed below are modules found by the server but not recognized as one of the supported modules. If the user believes that any listed modules below should have been supported, please check the "CS.cfg" configuration file to see if there is a name mismatch and adjust this accordingly.

---

The Certificate System automatically discovers Safenet's LunaSA and nCipher's netHSM hardware security modules. The discovery process assumes that the client software installations for these modules are local to the Certificate System subsystem and are in the following locations:

- LunaSA: `/usr/lunasa/lib/libCryptoki2.so`
- LunaSA: `/usr/lunasa/lib/libCryptoki2_64.so`
- nCipher: `/opt/nfast/toolkits/pkcs11/libcknfast.so`

8. Create a new security domain.
The first CA instance must create a new security domain. Subsequent CAs can create a new domain or join an existing security domain, but it is recommended that each CA have its own security domain.

**TIP**

If a CA which is a security domain master is cloned, then that cloned CA is also a security domain master. In that case, both the original CA and its clone share the same security domain configuration.

9. Enter a name for the new instance.
10. Set up the PKI hierarchy. Commonly, the first CA is a root, or self-signed, CA, meaning that it signs its own CA signing certificate rather than submitting its certificates to a third-party CA for issuance. Subsequent CAs can be subordinate CAs to that root. There are many other options, depending on the PKI environment.

For a CA, there are two possible configuration options:

- **Root CA**. A root CA signs its own CA signing certificate and, therefore, can set its own certificate issuance rules.
- **Subordinate CA**. A subordinate CA receives its CA signing certificate from a root CA. The root CA must be referenced here; it can be another Certificate System CA, but this can be an external root CA. The certificate requests generated in this process must be submitted to the external CA and be approved before configuration can be completed.

11. Fill in the information for the LDAP server which will be used for the instance's internal database. This requires connection information for the Directory Server instance, such as the hostname, port number, bind DN (username), and password. This step also creates a database in the Directory
Server and a corresponding base directory entry (base DN) to use for the subsystem's entries. To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, *Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server*.

The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address, if IPv6 was configured before the packages were installed.

**NOTE**

One thing that can derail subsystem configuration or function is having services that are unable to connect with each other. If servers that need to communicate with each other are on different servers or networks, when the firewalls and iptables must be configured to give the required access.

If the Red Hat Directory Server instances is on a different server or network than the Certificate System subsystem, then make sure that the Certificate System host's firewall allows access to whatever LDAP port was set in the previous configuration panel. Installation will not complete if iptables is not configured properly. To configure iptables, see the Red Hat Enterprise Linux Deployment Guide, such as “Using iptables.” It is also possible to simply turn iptables off.

12. Set the key size and the hashing algorithm (RSA) or curve (ECC) to use for the subsystem instance keys. A root CA has the additional option of selecting the algorithm to use to sign its certificates.

By default, the settings for the signing key are applied to the keys for every certificate for the CA. To set different key types, sizes, or hashing algorithms (RSA) or curves (ECC) for each certificate, click the [Advanced] link to expand the form so each key pair is listed.
The default RSA key size is 2048 and for ECC, 256.

**Key Pairs**

Select the key pair type(s), associated key pair size(s) or curve name(s), and signature algorithm(s) from the pull-down menus. [Details]

- **Key Type**: RSA
- **Signed With**: SHA256withRSA
- **Signing Algorithm**: SHA256withRSA
- **Use the default key size (2048 bits).**
- **Use the following custom key size:**
  - **Key Size**: 2048

**IMPORTANT**

ECC can be used for any keys for the subsystem, with one exception: only RSA can be used for audit signing keys.

**NOTE**

An ECC CA signing certificate can be used to sign both ECC and RSA certificates. If you do not want to use the ECC client certificate that is generated at installation, simply replace the client certificate after configuration, and keep the ECC CA signing certificate.

Any ECC-enabled PKCS#11 module must be loaded before beginning to configure the CA.

13. Optionally, change the subject names for the certificates.
NOTE

Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.

14. The next panels generate and show certificate requests, certificates, and key pairs.
15. If the subsystem will ever be cloned, or as a protection if keys or certificates are ever lost, back up the keys and certificates when prompted. It is also possible to extract these keys later.
16. The configuration wizard will prompt to import the new CA certificate. Set all of the trust flags (web, email, and software) and then import the certificate.

![Import CA's Certificate Chain](image)

The CA's certificate chain needs to be imported into your browser. Also, you must trust the CA. Once this is done, click Next.

17. Provide the information for the new subsystem administrator.

![Administrator](image)

The administrator is the privileged user who manages this subsystem. Please enter the following relevant information. A certificate request will be automatically generated and submitted. The administrator entry will be created in the internal database and his certificate will be imported into this browser automatically in the next panel. This administrator certificate is used to access the agent interface of this subsystem.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID:</td>
<td>admin</td>
</tr>
<tr>
<td>Name:</td>
<td>CA Administrator of Instance rlpki-example</td>
</tr>
<tr>
<td>Email:</td>
<td><a href="mailto:email@example.com">email@example.com</a></td>
</tr>
<tr>
<td>Password:</td>
<td>password</td>
</tr>
<tr>
<td>Password (Again):</td>
<td>password</td>
</tr>
</tbody>
</table>

18. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

19. When the configuration is complete, restart the subsystem.

```
service pki-ca restart
```

**IMPORTANT**

The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

6.2.2. Installing and Configuring a DRM
A Data Recovery Manager (DRM) is also known as a Key Recovery Authority (KRA). All command-line tools and many files for the DRM use the abbreviation `kra` for this reason. In the documentation, the subsystem used to archive and recover keys is called the DRM or KRA interchangeably.

1. A CA must be configured and running somewhere on the network. A DRM depends on the CA to issue their certificates and to create a security domain. If the security domain CA is not available, then the configuration process fails.

2. Set up the required `yum` repositories.
   Create a `.repo` file with the repository information (this was likely configured in Section 6.2.1, “Installing and Configuring a CA”). For example:

   ```bash
   [root@client ~]# touch /etc/yum.repos.d/rhcs.repo
   [root@client ~]# vim /etc/yum.repos.d/rhcs.repo
   [rhcs]
   name=rhcs
   baseurl=ftp://repo_ip_address/pub/rhcsrepo
   enabled=1
   gpgcheck=0
   ```

3. Run `yum` to install the DRM packages. Optionally, include the console packages.

   ```bash
   [root@server ~]# yum install pki-kra pki-console
   ```

4. Run the `pkicreate` command to create the DRM instance. For example:
pkicreate -pki_instance_root=/var/lib
  -pki_instance_name=pki-kra
  -subsystem_type=kra
  -agent_secure_port=10443
  -ee_secure_port=10444
  -admin_secure_port=10445
  -unsecure_port=10180
  -tomcat_server_port=10701
  -audit_group=pkiaudit
  -redirect logs=/var/log/pki- name/logs

PKI instance creation Utility ...
PKI instance creation completed ...

Starting instance_name: [ OK ]

instance_name (pid 17990) is running ...

'instance_name' must still be CONFIGURED!
(see /var/log/pki-name-install.log)

Before proceeding with the configuration, make sure the firewall settings of this machine permit proper access to this subsystem.

Please start the configuration by accessing:

http://server.example.com:10180/kra/admin/console/config/login?
pin=kI7E1MByNIUcPJ6RKHmH

After configuration, the server can be operated by the command:

  service instance_name start | stop | restart

This example uses the recommended port separation configuration, specifies an auditor group, and uses a Java Security Manager. Other options could be specified to set user-defined log and configuration directories and a user-defined operating system user and group. For other pkicreate options, see Table 6.1, “pkicreate Parameters”.

The command options here are on separate lines to make it clear what options are used. All options should be on a single line.

5. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

6. Download the CA certificate chain for the CA which will issue the CA certificate, and import the CA chain into the browser.
   a. Open the CA web services page.
      https://server.example.com:9444/ca/ee/ca
   b. Click the Retrieval tab.
   c. Click the Import CA Certificate Chain link.
   d. Select the radio button to import the CA certificate into the browser.
e. Click **Submit**.

7. When the `pkicreate` command completes, it returns a URL to use to access the web-based configuration wizard and a PIN to use to authenticate. Open the configuration wizard using the URL returned from the package installation.

   Alternatively, log into the setup wizard through the admin link on the services page and supply the `preop.pin` value from the `/var/lib/pki-kra/conf/CS.cfg` file when prompted.

   ```
   https://server.example.com:10445/kra/services
   ```

8. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given.

   Any hardware tokens used with the instance must be configured *before* configuring the subsystem instance.
9. Join an existing security domain by entering the CA information. This URL can be identified by running `service pki-ca status` on the CA's host; the security domain URL is returned with the other configuration settings. For example:

```
https://server.example.com:9445
```

When the CA is successfully contacted, then supply the admin username and password for the CA so that it can be properly accessed.
10. Enter a name for the new instance.

11. Fill in the information for the LDAP server which will be used for the instance's internal database. This requires connection information for the Directory Server instance, such as the hostname, port number, bind DN (username), and password. This step also creates a database in the Directory Server and a corresponding base directory entry (base DN) to use for the subsystem's entries. To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server.
The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address.

**NOTE**

One thing that can derail subsystem configuration or function is having services that are unable to connect with each other. If servers that need to communicate with each other are on different servers or networks, when the firewalls and iptables must be configured to give the required access.

If the Red Hat Directory Server instances is on a different server or network than the Certificate System subsystem, then make sure that the Certificate System host's firewall allows access to whatever LDAP port was set in the previous configuration panel.

Installation will not complete if iptables is not configured properly. To configure iptables, see the Red Hat Enterprise Linux *Deployment Guide*, such as “Using iptables.” It is also possible to simply turn iptables off.

12. Set the key size and the algorithm (RSA) or curve (ECC) to use for the subsystem instance keys.

To set different key types, sizes, or algorithms (RSA) or curves (ECC) for each certificate, click the [Advanced] link to expand the form so each key pair is listed.

The default RSA key size is 2048 and for ECC, 256.
Key Pairs

Select the key pair type(s), associated key pair size(s) or curve name(s), and signature algorithm(s) from the pulldown menus. [Details]

Common Key Settings

- **Key Type:** RSA
- **Signing Algorithm:** SHA256withRSA

- Use the default key size (2048 bits).
- Use the following custom key size:

  - **Key Size:** 2048

**IMPORTANT**

ECC can be used for any keys for the subsystem, with one exception: only RSA can be used for audit signing keys.

Any ECC-enabled PKCS#11 module must be loaded before beginning to configure the CA.

13. Optionally, change subject names to the listed certificates.
NOTE

Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.

14. The next panels generate and show certificate requests, certificates, and key pairs.
15. If the subsystem will ever be cloned, or as a protection if keys or certificates are ever lost, back up the keys and certificates when prompted. It is also possible to extract these keys later.

16. Provide the information for the new subsystem administrator.
17. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

18. When the configuration is complete, restart the subsystem.

```
service pki-kra restart
```

**IMPORTANT**

The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

### 6.2.3. Installing and Configuring an OCSP Responder

1. A CA must be configured and running somewhere on the network. An OCSP responder depends on the CA to issue their certificates and to create a security domain. If the security domain CA is not available, then the configuration process fails.

2. Set up the required **yum** repositories.

   Create a `.repo` file with the repository information (this was likely configured in **Section 6.2.1. “Installing and Configuring a CA”**). For example:

   ```
   [root@client ~]# touch /etc/yum.repos.d/rhcs.repo
   [root@client ~]# vim /etc/yum.repos.d/rhcs.repo
   
   [rhcs]
   name=rhcs
   baseurl=ftp://repo_ip_address/pub/rhcsrepo
   enabled=1
   gpgcheck=0
   
   ```

3. Run **yum** to install the OCSP packages. Optionally, include the console packages.

   ```
   [root@server ~]# yum install pki-ocsp pki-console
   ```

4. Run the **pkicreate** command to create the OCSP instance. For example:
**pkicreate**

```
pkicreate -pki_instance_root=/var/lib
   -pki_instance_name=pki-ocsp
   -subsystem_type=ocsp
   -agent_secure_port=11443
   -ee_secure_port=11444
   -admin_secure_port=11445
   -unsecure_port=11180
   -tomcat_server_port=11701
   -audit_group=pkiaudit
   -redirect logs=/var/log/pki-name/logs
```

PKI instance creation Utility ...

PKI instance creation completed ...

Starting *instance_name*: [ OK ]

```
instance_name (pid 17990) is running ...

'instance_name' must still be CONFIGURED!
(see /var/log/pki-name-install.log)
```

Before proceeding with the configuration, make sure the firewall settings of this machine permit proper access to this subsystem.

Please start the configuration by accessing:

```
http://server.example.com:11180/ocsp/admin/console/config/login?
pin=IOjh7fIOjkld90kkI7E1MByNIUcPJ6RKHmH
```

After configuration, the server can be operated by the command:

```
service instance_name start | stop | restart
```

This example uses the recommended port separation configuration, specifies an auditor group, and uses a Java Security Manager. Other options could be specified to set user-defined log and configuration directories and a user-defined operating system user and group. For other **pkicreate** options, see Table 6.1, "pkicreate Parameters".

The command options here are on separate lines to make it clear what options are used. All options should be on a single line.

5. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

6. Download the CA certificate chain for the CA which will issue the CA certificate, and import the CA chain into the browser.
   
a. Open the CA web services page.

   ![https://server.example.com:9444/ca/ee/ca](https://server.example.com:9444/ca/ee/ca)

b. Click the **Retrieval** tab.

c. Click the **Import CA Certificate Chain** link.

d. Select the radio button to import the CA certificate into the browser.
e. Click **Submit**.

7. When the `pkicreate` command completes, it returns a URL to use to access the web-based configuration wizard and a PIN to use to authenticate. Open the configuration wizard using the URL returned from the package installation.

   Alternatively, log into the setup wizard through the admin link on the services page and supply the `preop.pin` value from the `/var/lib/pki-ocsp/conf/CS.cfg` file when prompted.

   ```plaintext
   https://server.example.com:11444/ocsp/services
   ```

8. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given.

   Any hardware tokens used with the instance must be configured **before** configuring the subsystem instance.
9. Join an existing security domain by entering the CA information. This URL can be identified by running `service pki-ca status` on the CA's host; the security domain URL is returned with the other configuration settings. For example:

```
https://server.example.com:9445
```

When the CA is successfully contacted, then supply the admin username and password for the CA so that it can be properly accessed.
10. Enter a name for the new instance.

11. Fill in the information for the LDAP server which will be used for the instance's internal database. This requires connection information for the Directory Server instance, such as the hostname, port number, bind DN (username), and password. This step also creates a database in the Directory Server and a corresponding base directory entry (base DN) to use for the subsystem's entries. To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, *Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server*. 
The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address.

**NOTE**

One thing that can derail subsystem configuration or function is having services that are unable to connect with each other. If servers that need to communicate with each other are on different servers or networks, when the firewalls and iptables must be configured to give the required access.

If the Red Hat Directory Server instances is on a different server or network than the Certificate System subsystem, then make sure that the Certificate System host's firewall allows access to whatever LDAP port was set in the previous configuration panel. Installation will not complete if iptables is not configured properly. To configure iptables, see the Red Hat Enterprise Linux *Deployment Guide*, such as "Using iptables." It is also possible to simply turn iptables off.

12. Set the key size and the algorithm (RSA) or curve (ECC) to use for the subsystem instance keys.

To set different key types, sizes, or algorithms (RSA) or curves (ECC) for each certificate, click the [Advanced] link to expand the form so each key pair is listed.

The default RSA key size is 2048 and for ECC, 256.
IMPORTANT

ECC can be used for any keys for the subsystem, with one exception: only RSA can be used for audit signing keys.

Any ECC-enabled PKCS#11 module must be loaded before beginning to configure the CA.

13. Optionally, change subject names to the listed certificates.
Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.

14. The next panels generate and show certificate requests, certificates, and key pairs.
15. If the subsystem will ever be cloned, or as a protection if keys or certificates are ever lost, back up the keys and certificates when prompted. It is also possible to extract these keys later.

16. Provide the information for the new subsystem administrator.
17. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

18. When the configuration is complete, restart the subsystem.

   ```
   service pki-ocsp restart
   ```

   **IMPORTANT**

   The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

19. Restart the CA instance. Restarting the CA instance loads the configuration for the new OCSP responder.

   ```
   service pki-ca restart
   ```

### 6.2.4. Installing and Configuring an RA

1. A CA must be configured and running somewhere on the network. An RA depends on the CA to issue their certificates and to create a security domain. If the security domain CA is not available, then the configuration process fails.

2. Set up the required **yum** repositories.

   Create a `.repo` file with the repository information (this was likely configured in Section 6.2.1, “Installing and Configuring a CA”). For example:
3. Run `yum` to install the RA packages.

```
[root@server ~]# yum install pki-ra
```

4. Run the `pkicreate` command to create the RA instance. For example:

```
pkicreate -pki_instance_root=/var/lib
           -pki_instance_name=pki-ra
           -subsystem_type=ra
           -secure_port=12889
           -non_clientauth_secure_port=12890
           -unsecure_port=12888
           -redirect logs=/var/log/pki-
                          name/logs
```

When the `pkicreate` command completes, it returns a URL to use to access the web-based configuration wizard and a PIN to use to authenticate. This PIN is also contained in the install logs (`/var/log/pki-name-install.log`) and in the `CS.cfg` file for the instance.

```
PKI instance creation Utility ...
PKI instance creation completed ...
Starting `instance_name:` [ OK ]
instance_name (pid 17990) is running ...
   `instance_name` must still be CONFIGURED!
   (see /var/log/pki-name-install.log)

Before proceeding with the configuration, make sure the firewall settings of this machine permit proper access to this subsystem.

Please start the configuration by accessing:

http://server.example.com:12888/ra/admin/console/config/login?
    pin=kI7E1MByNIUCPJ6RKHmH

After configuration, the server can be operated by the command:

   service `instance_name` start | stop | restart
```

This example uses the recommended port separation configuration, specifies an auditor group, and uses a Java Security Manager. Other options could be specified to set user-defined log and configuration directories and a user-defined operating system user and group. For other `pkicreate` options, see Table 6.1, "pkicreate Parameters".

The command options here are on separate lines to make it clear what options are used. All
options should be on a single line.

5. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

6. Download the CA certificate chain for the CA which will issue the CA certificate, and import the CA chain into the browser.
   a. Open the CA web services page.

   ```plaintext
   https://server.example.com:9444/ca/ee/ca
   ```

   b. Click the Retrieval tab.
   c. Click the Import CA Certificate Chain link.
   d. Select the radio button to import the CA certificate into the browser.

   ![Red Hat Certificate Manager](image)

   e. Click Submit.

7. If the CA which will be used to configure the RA is configured to prefer client authentication (`sslClientAuth = want` is set in the `server.xml` file), then this setting must be disabled before the RA can be configured. Otherwise, the CA requests client authentication when the RA attempts to connect with it during configuration, which the RA cannot perform, and the configuration process hangs.

   The procedure for changing the client authentication settings is in the *Administrator's Guide*.

8. Open the configuration wizard using the URL returned by running `pkicreate`.

   ```plaintext
   https://server.example.com:12889/ra/admin/console/config/login?pin=kI7E1MByNIUcPJ6RKHmH
   ```

9. Join an existing security domain by entering the CA information. This URL can be identified by running `service pki-ca status` on the CA's host; the security domain URL is returned with the other configuration settings. For example:

   ```plaintext
   https://server.example.com:9445
   ```
When the CA is successfully contacted, then supply the admin username and password for the CA so that it can be properly accessed.

The hostname for the security domain CA can be the fully-qualified domain name or an IPv4 or IPv6 address, if IPv6 was configured before the packages were installed.

10. Enter a name for the new instance.
11. Select the CA which will issue, renew, and revoke certificates for certificates processed through the RA. All of the CAs configured in the security domain are listed in a dropdown menu.

12. Click **Next** on the **Internal Database** panel; the SQLite database is created automatically.
The RA uses a SQLite database to store its configuration and user data rather than an LDAP database, as the other subsystems do.

13. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given. Any hardware tokens used with the instance must be configured before configuring the subsystem instance.

14. Set the key size and type (RSA or ECC) to use for the subsystem instance keys. By default, the settings for the signing key are applied to the keys for every certificate for the CA. To set different key types, sizes, or hashing algorithms (RSA) or curves (ECC) for each certificate, click the [Advanced] link to expand the form so each key pair is listed.

The default RSA key size is 2048 and for ECC, 256.
Any ECC-enabled PKCS#11 module must be loaded before beginning to configure the RA.

15. Optionally, change the subject names for the certificates.
Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.

16. The next panels generate and show certificate requests, certificates, and key pairs.
17. Provide the information for the new subsystem administrator.

**Administrator**

The administrator is the privileged user who manages this subsystem. Please enter the following relevant information. A certificate request will be automatically generated and submitted. The administrator entry will be created in the internal database and his certificate will be imported into this browser automatically in the next panel. This administrator certificate is used to access the agent interface of this subsystem.

- **UID:** admin
- **Name:** RA Administrator of Instance pki-ra
- **Email:** email@example.com
- **Password:**
- **Password (Again):**

18. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

19. When the configuration is complete, restart the subsystem.

```
service pki-ra restart
```
The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

6.3. Configuring a Token Management System

As covered in Section 2.3, “Working with Smart Cards (TMS),” there are two subsystems that are dedicated to managing smart cards and tokens: the Token Key Service and the Token Processing System. The TPS actually performs operations on smart cards; the TKS derives the keys used to TPS-smart card communication.

The order of installation is important when creating a token management system. Three subsystems are required — CA, TKS, and TPS — and they must be installed in that order. The TKS requires a CA for configuration, and the TPS requires a CA and TPS.

This walk-through simply shows, at a very high level, the major steps for setting up a functional token management system.

1. Install a Red Hat Directory Server, as described in Section 5.3.3, “Installing Red Hat Directory Server”. This can be on a different machine from the Certificate System, which is the recommended scenario for most deployments.
2. Create new, specific operating system groups for the Certificate System subsystems to run as. This is described in Section 5.3.7.1, “Creating Operating System Groups”.
3. Assign users to the operating system groups to perform the subsystem administrative tasks. This is described in Section 5.3.7.2.3, “Associating Existing User Accounts with PKI Groups”.
4. Download the Certificate System packages from the Red Hat Network channel.
5. Install the packages.
6. Run `pkicreate` to create the subsystem instances.
7. Configure the Certificate System CA subsystem.
8. Optionally, configure the DRM subsystem.
9. Configure the TKS subsystem.
10. Configure the TPS subsystem. The TPS requires having an existing TKS and DRM available when it is configured, so this is the last subsystem to set up.

6.3.1. Installing and Configuring a TKS

1. A CA must be configured and running somewhere on the network. A TKS depends on the CA to issue their certificates and to create a security domain. If the security domain CA is not available, then the configuration process fails.
2. Set up the required `yum` repositories.

   Create a `.repo` file with the repository information (this was likely configured in Section 6.2.1, “Installing and Configuring a CA”). For example:
3. Run `yum` to install the TKS packages. Optionally, include the console packages.

```bash
[root@server ~]# yum install pki-tks pki-console
```

4. Run the `pkicreate` command to create the TKS instance. For example:

```bash
pkicreate -pki_instance_root=/var/lib
  -pki_instance_name=pki-tks
  -subsystem_type=tks
  -agent_secure_port=13443
  -ee_secure_port=13444
  -admin_secure_port=13445
  -unsecure_port=13180
  -tomcat_server_port=13701
  -audit_group=pkiaudit
  -redirect logs=/var/log/pki-
    name/logs
PKI instance creation Utility ...
PKI instance creation completed ...
Starting instance_name: [ OK ]
instance_name (pid 17990) is running ...

'instance_name' must still be CONFIGURED!
(see /var/log/pki-name-install.log)

Before proceeding with the configuration, make sure
the firewall settings of this machine permit proper
access to this subsystem.

Please start the configuration by accessing:
http://server.example.com:13180/tks/admin/console/config/login?
pin=kI7E1MByNIucPj6RKHmH

After configuration, the server can be operated by the command:

```
  service instance_name start | stop | restart
```

This example uses the recommended port separation configuration, specifies an auditor group,
and uses a Java Security Manager. Other options could be specified to set user-defined log and
configuration directories and a user-defined operating system user and group. For other
pkicreate options, see Table 6.1, “pkicreate Parameters”.

The command options here are on separate lines to make it clear what options are used. All
options should be on a single line.

5. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because
of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

6. Download the CA certificate chain for the CA which will issue the CA certificate, and import the CA chain into the browser.
   a. Open the CA web services page.
      
      \[ \text{https://server.example.com:9444/ca/ee/ca} \]
      
   b. Click the Retrieval tab.
   c. Click the Import CA Certificate Chain link.
   d. Select the radio button to import the CA certificate into the browser.
   e. Click Submit.

7. When the `pkicreate` command completes, it returns a URL to use to access the web-based configuration wizard and a PIN to use to authenticate. Open the configuration wizard using the URL returned from the package installation.

8. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given.

   Any hardware tokens used with the instance must be configured \textit{before} configuring the subsystem instance.
9. Join an existing security domain by entering the CA information. This URL can be identified by running `service pki-ca status` on the CA's host; the security domain URL is returned with the other configuration settings. For example:

```
https://server.example.com:9445
```

When the CA is successfully contacted, then supply the admin username and password for the CA so that it can be properly accessed.
10. Enter a name for the new instance.

11. Fill in the information for the LDAP server which will be used for the instance's internal database. This requires connection information for the Directory Server instance, such as the hostname, port number, bind DN (username), and password. This step also creates a database in the Directory Server and a corresponding base directory entry (base DN) to use for the subsystem's entries. To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, *Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server*. 

---

**Security Domain (Example Domain) Login**

The Enterprise KRA Administrator will register this KRA Subsystem located at example.com under this Security Domain located at example.com. The credential information will be provided to the Security Domain for authentication.

- **Uid:** admin
- **Password:** **********
The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address.

**NOTE**

One thing that can derail subsystem configuration or function is having services that are unable to connect with each other. If servers that need to communicate with each other are on different servers or networks, when the firewalls and iptables must be configured to give the required access.

If the Red Hat Directory Server instances is on a different server or network than the Certificate System subsystem, then make sure that the Certificate System host's firewall allows access to whatever LDAP port was set in the previous configuration panel. Installation will not complete if iptables is not configured properly. To configure iptables, see the Red Hat Enterprise Linux Deployment Guide, such as "Using iptables." It is also possible to simply turn iptables off.

12. Set the key size and the algorithm (RSA) or curve (ECC) to use for the subsystem instance keys. To set different key types, sizes, or algorithms (RSA) or curves (ECC) for each certificate, click the **Advanced** link to expand the form so each key pair is listed.

The default RSA key size is 2048 and for ECC, 256.
ECC can be used for any keys for the subsystem, with one exception: only RSA can be used for audit signing keys.

Any ECC-enabled PKCS#11 module must be loaded before beginning to configure the CA.

13. Optionally, change subject names to the listed certificates.
Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.

14. The next panels generate and show certificate requests, certificates, and key pairs.
15. If the subsystem will ever be cloned, or as a protection if keys or certificates are ever lost, back up the keys and certificates when prompted. It is also possible to extract these keys later.

16. Provide the information for the new subsystem administrator.
17. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

18. **Before restarting the new TKS instance**, create a shared secret key.

The `tkstool` script prints information about the shared secret key as it creates the key. These session key shares are required to import the shared key onto the TPS, so record this information.

```
tkstool -T -d /var/lib/pki-tks/alias -n sharedSecret
Generating the first session key share . . .
   first session key share: 792F AB89 8989 D902 9429 6137 8632 7CC4
   first session key share KCV: D1B6 14FD
Generating the second session key share . . .
   second session key share: 4CDF C8E0 B385 68EC 380B 6D5E 1C19 3E5D
   second session key share KCV: 1EC7 8D4B
Generating the third session key share . . .
   third session key share: CD32 3140 25B3 C789 854F 2C94 26C4 9752
   third session key share KCV: 73D6 8633
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .
transport key KCV: A8D0 97A2
Storing transport key on final specified token . . .
Naming transport key "sharedSecret" . . .
Successfully generated, stored, and named the transport key!
```
NOTE

If you are using a hardware token, the **tkstool** script could return an error requiring you to set environment variables before running the tool. Set the environment variables as directed, and then re-run the tool.

List the keys to make sure the shared key was properly imported.

```
tkstool -d /var/lib/pki-tks/alias -L
```

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

Enter Password or Pin for "NSS Certificate DB": xxxxx

<0> sharedSecret

The shared key is **sharedSecret**, which is the default name.

19. When the configuration is complete, restart the subsystem.

```
service pki-tks restart
```

IMPORTANT

The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

### 6.3.2. Installing and Configuring a TPS

1. A CA must be configured and running somewhere on the network.
2. A TKS must be configured and running somewhere on the network.
3. If a DRM will be used to store keys, then it must be configured and running somewhere on the network.
4. On the TPS host machine, set up the required **yum** repositories. Create a **.repo** file with the repository information (this was likely configured in **Section 6.2.1, “Installing and Configuring a CA”**). For example:

   ```
   [root@client ~]# touch /etc/yum.repos.d/rhcs.repo
   [root@client ~]# vim /etc/yum.repos.d/rhcs.repo
   
   [rhcs]
   name=rhcs
   baseurl=ftp://repo_ip_address/pub/rhcsrepo
   enabled=1
   gpgcheck=0
   
   [root@client ~]# yum install pki-tps
   ```

5. Run **yum** to install the TPS packages.

   ```
   [root@server ~]# yum install pki-tps
   ```

6. Run the **pkicreate** command to create the TPS instance. For example:
pkicreate -pki_instance_root=/var/lib
    -pki_instance_name=pki-tps
    -subsystem_type=tps
    -secure_port=7889
    -non_clientauth_secure_port=7890
    -unsecure_port=7888
    -audit_group=pkiaudit
    -redirect logs=/var/log/pki-name/logs

PKI instance creation Utility ...

PKI instance creation completed ...

Starting instance_name:                                         [  OK  ]

instance_name (pid 17990) is running ...

    'instance_name' must still be CONFIGURED!
    (see /var/log/pki-name-install.log)

Before proceeding with the configuration, make sure
the firewall settings of this machine permit proper
access to this subsystem.

Please start the configuration by accessing:

http://server.example.com:7888/tps/admin/console/config/login?
    pin=kI7E1MByNIUcPJ6RKHmH

After configuration, the server can be operated by the command:

    service instance_name start | stop | restart

This example uses the recommended port separation configuration, specifies an auditor group,
and uses a Java Security Manager. Other options could be specified to set user-defined log and
configuration directories and a user-defined operating system user and group. For other
pkicreate options, see Table 6.1, “pkicreate Parameters”.

The command options here are on separate lines to make it clear what options are used. All
options should be on a single line.

When the pkicreate command completes, it returns a URL to use to access the web-based
configuration wizard and a PIN to use to authenticate. This PIN is also contained in the install logs
(/var/log/pki-name-install.log) and in the CS.cfg file for the instance.

7. Create a new Firefox browser profile to use for configuring and accessing subsystems. Because
   of the certificates that are loaded, it is simpler and cleaner to use a fresh profile.

8. Download the CA certificate chain for the CA which will issue the CA certificate, and import the CA
   chain into the browser.
   a. Open the CA web services page.

   https://server.example.com:9444/ca/ee/ca

   b. Click the Retrieval tab.

   c. Click the Import CA Certificate Chain link.

   d. Select the radio button to import the CA certificate into the browser.
e. Click **Submit**.

9. If the CA, TKS, or DRM which will be used to work with the TPS is configured to prefer client authentication (**sslClientAuth = want** is set in the **server.xml** file), then this setting must be disabled before the TPS can be configured. Otherwise, the CA, TKS, or DRM will request client authentication when the TPS attempts to connect with it during configuration, which the TPS cannot perform.

   The procedure for changing the client authentication settings is in the *Administrator’s Guide*.

10. Open the configuration wizard using the URL returned by running **pkicreate**.

    ```
    http://server.example.com:7888/tps/admin/console/config/login?
    pin=kI7E1MByNIUcPJ6RKHmH
    ```

11. Select the token which will store the Certificate System certificates and keys; a list of detected hardware tokens and databases is given.

    Any hardware tokens used with the instance must be configured *before* configuring the subsystem instance.
12. Join an existing security domain by entering the CA information. This URL can be identified by running `service pki-ca status` on the CA's host; the security domain URL is returned with the other configuration settings. For example:

```
https://server.example.com:9445
```

When the CA is successfully contacted, then supply the admin username and password for the CA so that it can be properly accessed.
13. Enter a name for the new instance.

14. Select the CA which will issue, renew, and revoke certificates for token operations requested through the TPS subsystem.
15. Supply information about the TKS which will manage the TPS keys. Select the TKS from the drop-down menu of TKS subsystems within the security domain.

16. There is an option for server-side key generation for tokens enrolled through the TPS. If server-side key generation is selected, supply information about the DRM which will generate keys and archive encryption keys. Key and certificate recovery is initiated automatically through the TPS, which is a DRM agent. Select the DRM from the drop-down menu of DRM subsystems within the security domain.

17. Fill in the Directory Server authentication directory. This directory is used by the TPS to authenticate users which access the Enterprise Security Client and as an additional database for certificates and keys.
This Directory Server instance must not be the same Directory Server instance used as the TPS's internal database. This is a general user directory, which may be accessed by other applications or users, whereas the TPS's internal database is used exclusively by the TPS and is created on the fly as the TPS is configured.

To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, *Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server*.

The hostname of the LDAP server can be the fully-qualified domain name or an IPv4 or IPv6 address.

18. Fill in the information for the LDAP server which will be used to authenticate the identity of end users. [Details]

<table>
<thead>
<tr>
<th>Host:</th>
<th>localhost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port:</td>
<td>389</td>
</tr>
<tr>
<td>Base DN:</td>
<td>dc=example,dc=com</td>
</tr>
</tbody>
</table>

To configure SSL client authentication, make sure that the SSL port is set and the SSL checkbox is selected. The Directory Server must be configured to run in SSL, as described in Chapter 7, *Installing Red Hat Certificate System with SSL Connections to Red Hat Directory Server*.
The hostname can be the fully-qualified domain name or an IPv4 or IPv6 address.

**NOTE**

One thing that can derail subsystem configuration or function is having services that are unable to connect with each other. If servers that need to communicate with each other are on different servers or networks, when the firewalls and iptables must be configured to give the required access.

If the Red Hat Directory Server instances is on a different server or network than the Certificate System subsystem, then make sure that the Certificate System host's firewall allows access to whatever LDAP port was set in the previous configuration panel.

Installation will not complete if iptables is not configured properly. To configure iptables, see the Red Hat Enterprise Linux *Deployment Guide*, such as "Using iptables." It is also possible to simply turn iptables off.

19. Set the key size and type (RSA or ECC) to use for the subsystem instance keys.

By default, the settings for the signing key are applied to the keys for every certificate for the CA. To set different key types, sizes, or hashing algorithms for each certificate, click the [Advanced] link to expand the form so each key pair is listed.

The default RSA key size is 2048. The available algorithms are listed in *Section A.1, “RSA Hashing Algorithms”*. The default size for ECC is 256, and the only supported curve is nistp256.
20. Optionally, change subject names to the listed certificates.

21. The next panels generate and show certificate requests, certificates, and key pairs.

NOTE

Certificate nicknames must be unique, and changing the default nicknames is one way to ensure that. Having unique certificate nicknames is vital for using an HSM, since any nickname conflicts (even for subsystems on different servers) will cause configuration to fail.
22. Provide the information for the new subsystem administrator.

23. Click **Next** through the remaining panels to import the agent certificate into the browser and complete the configuration.

24. When the configuration is complete, restart the subsystem.
IMPORTANT

The new instance is not active until it is restarted, and weird behaviors can occur if you try to use the instance without restarting it first.

25. Stop the TPS.

   service pki-tps stop

26. Use the `tkstool` script to import the shared key into the NSS software token.

   `tkstool -I -d /var/lib/pki-tps/alias -n sharedSecret`

   The script will prompts for the session key shares that were printed when the key was created in the TKS. Enter the information from the TKS.

   NOTE

   If you are using a hardware token, the `tkstool` script could return an error requiring you to set environment variables before running the tool. Set the environment variables as directed, and then re-run the tool.

   List the keys to make sure the shared key was properly imported.

   `tkstool -d /var/lib/pki-tps/alias -L`

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

   Enter Password or Pin for "NSS Certificate DB": xxxxx
   <0> sharedSecret

   The shared key is `sharedSecret`, which is the default name. The TPS can be configured to set a different name for the shared key by changing the value of the `conn.tks1.tksSharedSymKeyName` value in the `CS.cfg`. This value must be the same as the nickname for the key imported into the token, or the TPS cannot locate the key.

27. Start the TPS instance.

   service pki-tps start

28. Edit the TPS configuration file to use the appropriate profile name for the `delegateISEtoken` profile.

   When a new TPS is installed using 8.1 packages, the `delegateISEtoken` profile is created with the wrong CA profile ID. The ID is incorrectly is set to `caTokenUserAuthenticationKeyEnrollment`.

   Open the TPS configuration file:

   `[root@server ~]# vim /var/lib/pki-tps/conf/CS.cfg`
Change the profile ID name to `caTokenUserDelegateAuthKeyEnrollment`:

```
op.enroll.delegateISEtoken.keyGen.encryption.ca.profileId=caTokenUserDelegateAuthKeyEnrollment
```

The CA, DRM, OCSP, TKS, and TPS all use a backend Red Hat Directory Server instance to store their certificate and configuration information. SSL connections can be configured between the Directory Server instance and the Certificate System subsystem instance by configuring SSL in the Directory Server and then enabling the Certificate System instance to use that connection.

**NOTE**

There are three parts to using SSL server connections with the Directory Server. The Directory Server must be configured before the Certificate System instance is configured. The next part imports the Directory Server’s CA certificate into the new Certificate System database after installation but before configuration. The last part configures the Certificate System instance to connect to the Directory Server over SSL as part of the Certificate System setup.

### 7.1. Using an External CA to Issue Directory Server Certificates

When first setting up Certificate System, a local CA may not be available. An external CA or a CA in a different PKI environment can be used to issue the certificates that are used to enabled SSL on the Directory Server instance. In that case, the only configuration necessary on the Certificate System side is to trust the CA which issued the Directory Server certificates.

**NOTE**

The CA certificate for the external CA which issued the Directory Server’s certificates must be imported into the security database for every subsystem instance which connects to that Directory Server.

   a. Obtain and install CA and server certificates for the Directory Server from the external authority. Each CA will have its own path for requesting and receiving certificates.
      When importing the CA certificate into the Directory Server security databases in the Directory Server Console, make sure to allow the CA certificate to be trusted for both client and server authentication.
   b. In the Directory Server Console, open the Configuration tab and the Encryption subtab. Check the Enable SSL checkbox and select all the ciphers and certificates to use.
   c. At the bottom of the window, select the Allow client authentication radio button. Do not require client authentication. Requiring client authentication will prevent the Certificate System server from connecting to the Directory Server instance.
   d. Restart the Directory Server instance.

   ```
   service dirsrv restart
   ```

2. If necessary, export the Directory Server's CA certificate so it can be imported into the Certificate System security database. The CA certificate had to be imported into the Directory Server, so a
copy should be available. If not, the CA certificate can be exported from the Directory Server Console or by using `certutil`.

```
certutil -L -d /ldap/alias/directory -n "DS CA certificate" -A > cacert.crt
```

3. Import that Directory Server’s CA certificate into the Certificate System security database. Importing the CA certificate allows the Certificate System instance to connect to the Directory Server over the secure port during its setup process.

```
# service instance_name stop
# certutil -A -i cacert.crt -t "CT,C,C" -n "CA_cert_nickname" -a -d /var/lib/instance_name/alias
# service instance_name start
```

4. For the TPS only. After the CA is configured, and after the TPS is created but before it is configured, import the Directory Server’s CA certificate into the TPS’s security databases.

```
# certutil -A -i cacert.crt -t "CT,C,C" -n "CA_cert_nickname" -a -d /var/lib/pki-tps/alias
```

5. Begin the instance setup. When the wizard comes to the section to configure the LDAP instance to use, supply the SSL port for the Directory Server instance and select the SSL checkbox.

6. Optional. Configure SSL client authentication between the Certificate System and LDAP server. This is done after the instance is set up and is covered in the section in the Certificate System Administrator’s Guide for configuring the LDAP database.

### 7.2. Using Temporary Self-Signed Directory Server Certificates

When first setting up Certificate System, a local CA may not be available. The Directory Server can be initially configured to use SSL based on temporary self-signed certificates which are generated using `certutil`. Once a new Certificate System CA is fully setup and configured, the Directory Server can then request permanent SSL certificates from the CA. The Directory Server instance and the CA must then be reconfigured to use the new, permanent certificates.

**NOTE**

The CA certificate for the external CA which issued the Directory Server’s certificates only needs to be imported into the security database for the first CA configured. Once the temporary certificates are replaced by the ones issued by the Certificate System CA, every subsystem instance in the security domain will automatically trust the Directory Server because they will already trust the issuing Certificate System CA.

1. Configure the Red Hat Directory Server instance to run over SSL. This is described in detail in the SSL configuration chapter in the Directory Server 8.2 Administrator’s Guide.

   a. Open the Directory Server instance's security directory.

```
cd /etc/dirsrv/slapd-instance
```

The Directory Server instance should have its security databases already set up in the `/etc/dirsrv/slapd-instance` directory. If these databases are missing for some reason,
they can be created using the certutil command.

```
certutil -N -d .
```

b. Generate temporary self-signed certificates for the Directory Server using certutil. For example:

```
certutil -S -n "Temporary CA certificate" -s "cn=Temporary CA
cert,dc=example,dc=com" -2 -x -t "CT,CT," -m 1000 -v 120 -d . -k rsa

certutil -S -n "Server-Cert" -s "cn=ldap.example.com" -c "Temporary CA
certificate" -t "u,u,u" -m 1001 -v 120 -d . -k rsa
```

c. Import the temporary server and CA certificates into the Directory Server using the Directory Server Console. When importing the CA certificate into the Directory Server security databases in the Directory Server Console, make sure to allow the CA certificate to be trusted for both client and server authentication.

d. In the Directory Server Console, open the Configuration tab and the Encryption subtab. Check the Enable SSL checkbox and select all the ciphers and certificates to use. The only server certificate listed should be the temporary server certificate, Server-Cert.

e. At the bottom of the window, select the Allow client authentication radio button. **Do not** require client authentication. Requiring client authentication will prevent the Certificate System server from connecting to the Directory Server instance.


```
service dirsrv restart
```

2. Export that Directory Server’s temporary CA certificate from its security database.

```
certutil -L -d /ldap/alias/directory -n "Temporary CA certificate" -A >
tempcacert.crt
```

3. Import that Directory Server’s temporary CA certificate into the Certificate System security database. Importing the CA certificate allows the Certificate System instance to connect to the Directory Server over the secure port during its setup process.

```
# service instance_name stop

# certutil -A -i tempcacert.crt -t "CT,C,C" -n "Temporary CA certificate" -a -
-d /var/lib/instance_name/alias

# service instance_name start
```

4. Begin the CA instance setup. When the wizard comes to the section to configure the LDAP instance to use, supply the SSL port for the Directory Server instance and select the SSL checkbox.

5. Once the CA is configured, it can be used to issue new certificates to the Directory Server instance.

   a. Generate a new certificate request for the Directory Server. This **must** have the same certificate nickname as the original, temporary certificate.

   A certificate request can be generated in the Directory Server Console or using certutil. For example:
b. Submit the generated certificate request through the CA's end-entities forms:

https://server.example.com:9444/ca/ee/ca

c. Log into the CA's agent forms as an agent, and approve the request. The process of approving certificates is covered in the Agent's Guide.

d. When the request is approved, the agent form returns the base 64-encoded version of the new certificate. Copy and save this certificate, including the header and footer lines, to a file.

e. Export the CA certificate so that it can be imported into the Directory Server.

f. Stop the Directory Server.

g. Stop the CA.

h. Delete the temporary Server-Cert SSL certificate from the Directory Server's security database:

i. Delete the temporary CA certificate from the Directory Server's security database:

j. Import the new, permanent Server-Cert SSL certificate into the Directory Server's security database:

k. Import the new, permanent CA signing certificate into the Directory Server's security database:

l. Start the Directory Server.

m. Start the CA.

6. For the TPS only. After the CA is configured, and after the TPS is created but before it is
configured, import the Directory Server's CA certificate into the TPS's security databases. The TPS instance must be stopped before the certificates can be imported.

```
# service pki-tps stop
# certutil -A -i cacert.crt -t "CT,C,C" -n "CA_certNickname" -a -d /var/lib/pki-tps/alias
# service pki-tps start
```

7. **Optional.** Configure SSL client authentication between each Certificate System subsystem instance and the LDAP server. This is done after the instance is set up and is covered in the section in the Certificate System Administrator's Guide for configuring the LDAP database.

A subsystem instance generates and stores its key information in a key store, called a security module or a token. A subsystem instance can be configured for the keys to be generated and stored using the internal NSS token or on a separate cryptographic device, a hardware token.

8.1. Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates

8.1.1. Types of Hardware Tokens

A token is a hardware or software device that performs cryptographic functions and stores public-key certificates, cryptographic keys, and other data. The Certificate System defines two types of tokens, internal and external, for storing key pairs and certificates that belong to the Certificate System subsystems.

8.1.1.1. Internal Tokens

An internal (software) token is a pair of files, usually called the certificate database and key database, that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System automatically generates these files in the filesystem of its host machine when pkicreate is run to create the subsystem instance.

In the Certificate System, the certificate database is named cert8.db; the key database is named key3.db. These files are located in the instanceID/alias directory.

8.1.1.2. External Tokens

An external token refers to an external hardware device, such as a smart card or hardware security module (HSM), that the Certificate System uses to generate and store its key pairs and certificates. The Certificate System supports any hardware tokens that are compliant with PKCS #11.

NOTE

See your specific HSM vendor documentation for installation and configuration instructions for using the HSM.

PKCS #11 is a standard set of APIs and shared libraries which isolate an application from the details of the cryptographic device. This enables the application to provide a unified interface for PKCS #11-compliant cryptographic devices.

The PKCS #11 module implemented in the Certificate System supports cryptographic devices supplied by many different manufacturers. This module allows the Certificate System to plug in shared libraries supplied by manufacturers of external encryption devices and use them for generating and storing keys and certificates for the Certificate System managers.

Consider using external tokens for generating and storing the key pairs and certificates used by Certificate System. These devices are another security measure to safeguard private keys because hardware tokens are sometimes considered more secure than software tokens.

Before using external tokens, plan how the external token is going to be used with the subsystem:
All system keys for a subsystem must be generated on the same token. The subsystem keys must be installed in an empty HSM slot. If the HSM slot has previously been used to store other keys, then use the HSM vendor's utilities to delete the contents of the slot. The Certificate System has to be able to create certificates and keys on the slot with default nicknames. If not properly cleaned up, the names of these objects may collide with previous instances. A single HSM can be used to store certificates and keys for multiple subsystem instances, which may be installed on multiple hosts. When an HSM is used, any certificate nickname for a subsystem must be unique for every subsystem instance managed on the HSM.

The Certificate System automatically discovers Safenet's LunaSA and nCipher's netHSM hardware security modules. The discovery process assumes that the client software installations for these modules are local to the Certificate System subsystem and are in the following locations:

- LunaSA: /usr/lunasa/lib/libCryptoki2.so
- LunaSA: /usr/lunasa/lib/libCryptoki2_64.so
- nCipher: /opt/nfast/toolkits/pkcs11/libcknfast.so

8.1.1.3. Hardware Cryptographic Accelerators
The Certificate System can use hardware cryptographic accelerators with external tokens. Many of the accelerators provide the following security features:

- Fast SSL connections. Speed is important to accommodate a high number of simultaneous enrollment or service requests.
- Hardware protection of private keys. These devices behave like smart cards by not allowing private keys to be copied or removed from the hardware token. This is important as a precaution against key theft from an active attack of an online Certificate Manager.

8.1.2. Using Hardware Security Modules with Subsystems
The Certificate System supports the nCipher netHSM hardware security module (HSM) by default. Certificate System-supported HSMs are automatically added to the secmod.db database with modutil during the pre-configuration stage of the installation, if the PKCS #11 library modules are in the default installation paths.

During configuration, the Key Store panel displays the supported modules, along with the NSS internal software PKCS #11 module. All supported modules that are detected show a status of Found and is individually marked as either Logged in or Not logged in. If a token is found but not logged in, it is possible to log in using the Login under Operations. If the administrator can log into a token successfully, the password is stored in a configuration file. At the next start or restart of the Certificate System instance, the passwords in the password store are used to attempt a login for each corresponding token.

Administrators are allowed to select any of the tokens that are logged in as the default token, which is used to generate system keys.

**IMPORTANT**
For information on configuring FIPS mode, see the hardware vendor documentation.

8.1.2.1. Adding or Managing the HSM Entry for a Subsystem
When an HSM is selected as the default token, some parameters are added to instance's CS.cfg file to
configure the HSM. For example:

```
#RHCS supported modules
preop.configModules.module0.commonName=NSS Internal PKCS #11 Module
preop.configModules.module0.imagePath=../img/mozilla.png
preop.configModules.module0.userFriendlyName=NSS Internal PKCS #11 Module
preop.configModules.module1.commonName=nfast
preop.configModules.module1.imagePath=../img/ncipher.png
preop.configModules.module1.userFriendlyName=nCipher's nFast Token Hardware Module
preop.configModules.module2.commonName=lunasa
preop.configModules.module2.imagePath=../img/safenet.png
preop.configModules.module2.userFriendlyName=SafeNet's LunaSA Token Hardware Module
#selected token
preop.module.token=Internal Key Storage Token
```

In addition, a parameter is added to the `password.conf` file for the HSM password. For example:

```
hardware-nethsm=caPassword
```

### 8.1.2.2. Using Chrysalis LunaSA HSM

To make sure that a LunaSA HSM works with Certificate System, edit the configuration files for the HSM before configuring the subsystems:

1. Check that the LunaSA module has been properly installed:

   ```
   modutil -dbdir /var/lib/instance_name/alias -list
   
   Listing of PKCS #11 Modules
   -----------------------------------------------
   1. NSS Internal PKCS #11 Module
      slots: 2 slots attached
      status: loaded
         slot: NSS Internal Cryptographic Services
            token: NSS Generic Crypto Services
               slot: NSS User Private Key and Certificate Services
                  token: NSS Certificate DB
   2. lunasa
      library name: /usr/lunasa/lib/libCryptoki2_64.so
      slots: 1 slot attached
      status: loaded
         slot: LunaNet Slot
            token: lunasa3-ca
   
   If the LunaSA module isn't listed, then install the module manually:
   a. Stop the subsystem.
      ```
      service instance_name stop
      ```
   b. Load the module.
c. Verify that the module has been loaded.

```
modutil -dbdir /var/lib/instance_name/alias -nocertdb -add lunasa
libfile /usr/lunasa/lib/libCryptoki2_64.so
```

d. Start the subsystem.

```
# Start the subsystem.
```

2. Open the `/etc/Chrystoki.conf` configuration file.

3. Add this configuration parameter.

```
# Misc { NetscapeCustomize=1023; }
```

4. If they are there, remove these two configuration lines for the applet version.

```
# AppIdMajor=2;
# AppIdMinor=4;
```

Then, after going through the subsystem configuration, but before restarting the server when completing the configuration wizard, edit the subsystem configuration to recognize the token:

1. Stop the server.

```
# Stop the server.
```

2. Edit the instance's `serverCertNick.conf` file in the `/var/lib/instance_name/conf` directory. Add the HSM token name to the `serverCert` parameter. The original value only points to the server:

```
Server-Cert instanceID
```

The new value includes a reference to the LunaSA HSM:

```
lunasa3-ca:Server-Cert instanceID
```

3. Start the server.

```
# Start the server.
```

### 8.1.2.3. Installing External Tokens and Unsupported HSM

To use HSMs which are not officially supported by the Certificate System, add the module to the subsystem database manually. If the desired HSM does not appear in the `Security Modules` panel during the subsystem configuration, check that the HSM is installed and activated correctly. Then run `modutil` manually to add the module to the `secmod.db` database.

1. Install the cryptographic device, using the manufacturer's instructions. Be sure to name the token something that will help identify it easily later.

2. Install the PKCS #11 module using the `modutil` command-line utility.
   a. Open the `alias` directory for the subsystem which is being configured with the PKCS #11
module. For example:

```
cd /var/lib/pki-ca/alias
```

b. The required security module database file, `secmod.db`, should be created by default when the subsystem is created. If it does not exist, use the `modutil` utility to create `secmod.db`.

```
modutil -dbdir . -nocertdb -create
```

c. Use the `modutil` utility to set the library information.

```
modutil -dbdir . -nocertdb / -add module_name -libfile library_file
```

`library_file` specifies the path to the library file containing the PKCS #11 interface module and `module_name` gives the name of the PKCS #11 module which was set when the drivers were installed.

- For the LunaSA HSM, the library can be `/usr/lunasa/lib/libCryptoki2_64.so` or `/usr/lunasa/lib/libCryptoki2.so`:

```
modutil -dbdir . -nocertdb -add lunasa -libfile /usr/lunasa/lib/libCryptoki2.so
```

- For an nCipher HSM:

```
modutil -dbdir . -nocertdb -add nethsm -libfile /opt/nfast/toolkits/pkcs11/libcknfast.so
```

**IMPORTANT**

To enable FIPS mode on an nCipher HSM, open the security UI:

```
/opt/nfast/bin/ksafe
```

In the 'Security World' tab, make sure that "Strict FIPS 140-2 Level III" is set to 'yes'. For more information on configuring FIPS mode, see the hardware vendor documentation.

### 8.1.2.4. Setting up SELinux on nCipher netHSM 2000

SELinux policies are created and configured automatically for all Certificate System instances, so Certificate System can run with SELinux in enforcing or permissive modes.

If SELinux is in enforcing mode, then any hardware tokens to be used with the Certificate System instances must also be configured to run with SELinux in enforcing mode, or the HSM will not be available during subsystem installation.

**IMPORTANT**

SELinux must be configured for the HSM before installing any Certificate System instances.
1. Install the SELinux packages for Certificate System.
   
   ```
   yum install pki-selinux
   ```

2. Reset the context of files in `/dev/nfast` and `/opt/nfast` to match the newly-installed policy.
   
   ```
   /sbin/restorecon -R /dev/nfast
   /sbin/restorecon -R /opt/nfast
   ```

3. Restart the netHSM software.

### 8.1.3. Viewing Tokens

To view a list of the tokens currently installed for a Certificate System instance, use the `modutil` utility.

1. Open the instance `alias` directory. For example:
   
   ```
   cd /var/lib/pki-ca/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
   ```

### 8.1.4. Detecting Tokens

To see if a token can be detected by Certificate System, 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-ca/alias
```

This utility will return all tokens which can be detected by the Certificate System, not only tokens which are installed in the Certificate System.

### 8.2. Configuring Subsystems with an HSM in FIPS Mode

If a subsystem is using a hardware security module (HSM) to store its key and certificate information, then this HSM can be enabled to use FIPS mode. This increases the security of the HSM and is recommended for most environments, especially Common Criteria environments.

#### 8.2.1. Configuring a CA with an HSM in FIPS Mode

1. Set up the HSM, as described in Section 8.1.2, “Using Hardware Security Modules with Subsystems” and the vendor documentation.
2. Install and configure the CA instance.
3. Stop the CA instance. The instance must be stopped to protect the information stored in its security databases.

   ```
   service pki-ca stop
   ```
4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on
the hardware token, but it should be placed on the software FIPS token.
   a. Open the CA's security database directory.

   ```bash
   cd /var/lib/pki-ca/alias
   ```

   b. Using `certutil`, create a request for a new SSL server certificate.

   ```bash
   certutil -d . -R -s "CN=ca.example.com,OU=pki-ca,O=Example Domain pki-ca" -o sslfips.req -h "NSS Certificate DB" -a
   ```

c. Restart the CA.

   ```bash
   service pki-ca start
   ```

d. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and
use the SSL Server Cert Profile to submit the request.

e. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the
request.

f. Copy the base 64-encoded certificate on the approval page and save it to a file, such as
`sslfips.cert`.

g. Stop the CA again.

   ```bash
   service pki-ca stop
   ```

h. Check the CA's certificate database to see if an SSL server certificate is already listed.

   ```bash
   certutil -d /var/lib/pki-ca/alias -L
   ```

i. If the certificate exists, then delete it.

   ```bash
   certutil -d /var/lib/pki-ca/alias -D -n "ServerCert nickname"
   ```

j. Import the new SSL server certificate.

   ```bash
   certutil -d /var/lib/pki-ca/alias -A -t "u,u,u" -n "ServerCert ca.example.com - Example Domain pki-ca" -i sslfips.cert -a
   ```

k. Edit the `/var/lib/pki-ca/conf/serverCertNick.conf` file to contain the nickname
of the new certificate, such as `ServerCert ca.example.com - Example Domain pki-ca`.

l. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.

   ```bash
   vim /var/lib/pki-ca/conf/CS/cfg
   ```

   ```bash
   ca.cert.sslserver.nickname= ServerCert ca.example.com - Example Domain pki-ca
   ca.sslserver.nickname= ServerCert ca.example.com - Example Domain pki-ca
   ```

m. In the `CS.cfg` file, add a line to verify signatures from the token. The value is the token
name, which depends on the vendor and version of the HSM. For example, for a NetHSM
token:
n. Edi the `server.xml` file to enable FIPS mode for each SSL-enabled connector. Set `strictCiphers` to true and add or set `ssl3` to false.

```xml
vim /var/lib/pki-ca/conf/server.xml

<Connector name="Agent" port="9443" maxHttpHeaderSize="8192"
  ... ...
  ss10ptions="ssl2=false,ssl3=false,tl1s=true"
  strictCiphers="true"
  ...>
```

o. Enable FIPS mode in the NSS software database.

```bash
modutil -dbdir /var/lib/pki-ca/alias -fips true
```

p. Verify that FIPS mode has been enabled. The command will return the current FIPS status.

```bash
modutil -dbdir /var/lib/pki-ca/alias modutil -dbdir . -chkfips true
FIPS mode enabled.
```

q. Start the CA.

```bash
service pki-ca start
```

### 8.2.2. Configuring a DRM, OCSP, or TKS with an HSM in FIPS Mode

1. Set up the HSM, as described in [Section 8.1.2, “Using Hardware Security Modules with Subsystems”](Red%20Hat%20Certificate%20System%208.1%20Deployment%2C%20Planning%2C%20and%20Installation_188) and the vendor documentation.

2. Install and configure the instance.

3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

   ```bash
   service instance_name stop
   ```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.

   a. Open the instance's security database directory.

   ```bash
cd /var/lib/instance_name/alias
   ```

   b. Using `certutil`, create a request for a new SSL server certificate.

   ```bash
certutil -d . -R -s "CN=server.example.com,OU=instance_name,O=Example Domain instance_name" -o sslfips.req -h "NSS Certificate DB" -a
   ```

   c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

   d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the
request.
e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as sslfips.cert.
f. Check the instance's certificate database to see if an SSL server certificate is already listed.

certutil -d /var/lib/instance_name/alias -L

g. If the certificate exists, then delete it.

certutil -d /var/lib/instance_name/alias -D -n "ServerCert nickname"

h. Import the new SSL server certificate.

certutil -d /var/lib/instance_name/alias -A -t "u,u,u" -n "ServerCert server.example.com - Example Domain instance_name" -i sslfips.cert -a

i. Edit the /var/lib/instance_name/conf/serverCertNick.conf file to contain the nickname of the new certificate, such as ServerCert server.example.com - Example Domain instance_name.

j. Edit the CS.cfg file to replace both references to the SSL server certificate nickname.

vim /var/lib/instance_name/conf/CS/cfg
type.cert.sslserver.nickname= ServerCert server.example.com - Example Domain instance_name
type.sslserver.nickname= ServerCert server.example.com - Example Domain instance_name

k. Edit the server.xml file to enable FIPS mode for each SSL-enabled connector. Set strictCiphers to true and add or set ssl3 to false. For example:

vim /var/lib/instance_name/conf/server.xml

<Connector name="Agent" port="11443" maxHttpHeaderSize="8192"
  ... ...
  sslOptions="ssl2=false,ssl3=false,tls=true"
  strictCiphers="true"
  ... >

l. Enable FIPS mode in the NSS software database.

modutil -dbdir /var/lib/instance_name/alias -fips true

m. Verify that FIPS mode has been enabled. The command will return the current FIPS status.

modutil -dbdir /var/lib/instance_name/alias modutil -dbdir . -chkfips true

FIPS mode enabled.

n. Start the instance.
8.2.3. Configuring a TPS with an HSM in FIPS Mode

1. Set up the HSM, as described in Section 8.1.2, "Using Hardware Security Modules with Subsystems" and the vendor documentation.

2. Install and configure the TPS.

3. Stop the TPS. The instance must be stopped to protect the information stored in its security databases.

   ```
   service pki-tps stop
   ```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance’s security database directory.

   ```
   cd /var/lib/pki-tps/alias
   ```

   b. Using `certutil`, create a request for a new SSL server certificate.

   ```
   certutil -d . -R -s "CN=tps.example.com,OU=pki-tps,O=Example Domain pki-tps" -o sslfips.req -h "NSS Certificate DB" -a
   ```

   c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

   d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

   e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as `sslfips.cert`.

   f. Check the instance's certificate database to see if an SSL server certificate is already listed.

   ```
   certutil -d /var/lib/pki-tps/alias -L
   ```

   g. If the certificate exists, then delete it.

   ```
   certutil -d /var/lib/pki-tps/alias -D -n "ServerCert nickname"
   ```

   h. Import the new SSL server certificate.

   ```
   certutil -d /var/lib/pki-tps/alias -A -t "u,u,u" -n "ServerCert tps.example.com - Example Domain pki-tps" -i sslfips.cert -a
   ```

   i. Edit the `/var/lib/pki-tps/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert tps.example.com - Example Domain pki-tps`.

   j. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.
5. The **nss.conf** file for the TPS contains a list of virtual servers for the different SSL listening ports for the TPS. Every SSL port must be configured to run under FIPS:
   a. Turn on FIPS.

   ```
   # FIPS Switch:
   # Enable/Disable FIPS mode
   NSSFIPS on
   ```

   b. Uncomment the FIPS cipher suite directive and comment out the standard directive.

   ```
   # SSL Cipher Suite:
   # List the ciphers that the client is permitted to negotiate.
   # See the mod_nss documentation for a complete list.
   #NSSCipherSuite -des,-desede3,-rc2,-rc2export,-rc4,-rc4export,+rsa_3des_sha,-rsa_des_56_sha,-rsa_null_md5,-rsa_null_sha,-rsa_rc4_40_md5,+rsa_rc4_128_md5,-rsa_rc4_128_sha,-rsa_rc4_40_md5,-rsa_rc4_56_sha,-fortezza,-fortezza_rc4_128_sha,-fortezza_null,-fips_des_sha,+fips_3des_sha,-rsa_aes_128_sha,-rsa_aes_256_sha,+ecdhe_ecdsa_aes_256_sha
   
   # SSL cipher suite in FIPS mode:
   NSSCipherSuite +rsa_3des_sha,-rsa_des_sha,-rsa_des_56_sha,-rsa_rc4_40_md5,-rsa_rc4_40_md5,-rsa_null_md5,-rsa_null_sha,+fips_3des_sha,-fips_des_sha,-fortezza,-fortezza_rc4_128_sha,-fortezza_null,-fips_des_sha,+fips_3des_sha,-rsa_aes_128_sha,-rsa_aes_256_sha
   ```

   c. For each SSL virtual host, turn off the setting to enforce valid certificates:

   ```
   NSSEnforceValidCerts off
   ```

6. Add or edit the line for the NSS FIPS database to the TPS's **password.conf** file.

   ```
   vim /var/lib/pki-tps/conf/password.conf
   NSS FIPS 140-2 Certificate DB:mypassword
   ```

7. Enable FIPS mode in the NSS software database.

   ```
   modutil -dbdir /var/lib/pki-tps/alias -fips true
   ```

8. Verify that FIPS mode has been enabled. The command will return the current FIPS status.

   ```
   modutil -dbdir /var/lib/pki-tps/alias modutil -dbdir . -chkfips true
   FIPS mode enabled.
   ```

9. Start the TPS.
8.3. About Retrieving Keys from an HSM

It is not possible to export keys and certificates stored on an HSM to a .p12 file. If this instance will be cloned, use the HSM tools to export the keys when necessary.
Chapter 9. Installing an Instance with ECC Enabled

Elliptic curve cryptography (ECC) is much more secure than the more common RSA-style encryption, which allows it to use much shorter key lengths and makes it faster to generate certificates. CAs which are ECC-enabled can issue both RSA and ECC certificates, using their ECC signing certificate.

Certificate System does not include a module natively to enable ECC, but it is possible to load and use a third-party PKCS #11 module with ECC-enabled.

To use the ECC module, it must be loaded before the subsystem instance is configured.

**IMPORTANT**

Third-party ECC modules must have an SELinux policy configured for them, or SELinux needs to be changed from **enforcing** mode to **permissive** mode to allow the module to function. Otherwise, any subsystem operations which require the ECC module will fail.

### 9.1. Loading a Third-Party ECC Module

1. Copy the third-party module to a common directory, like `/usr/lib` for 32-bit systems or `/usr/lib64` for 64-bit systems.
2. Create a new instance by running `pkicreate`, but do **not** go through the configuration wizard.
3. Stop the instance.

   ```
   service instance_name stop
   ```

4. The subsystem user runs as the `pkiuser` user. As `root`, create a home directory for `pkiuser`.

   ```
   /usr/sbin/usermod --home /usr/share/pki/pkiuser pkiuser
   cd /usr/share/pki
   mkdir pkiuser
   HOME=/usr/share/pki/pkiuser
   export HOME
   ```

5. Install the third-party module in the instance's security databases so it is available for the configuration.

   ```
   cd /var/lib/instance_name/alias
   modutil -dbdir . -nocertdb -add THIRD_PARTY_MODULE -libfile
   /usr/lib/libYourNewModule.so
   ```

   This creates a directory called `THIRD_PARTY_MODULE` in the new home directory created for `root` (the new `pkiuser` home directory). For example, if the module's name is EccForPki, then the directory is named `.EccForPki/`

6. Using `modutil`, set the password for the new ECC module token.

   ```
   modutil -dbdir . -nocertdb -changepw "THIRD_PARTY_MODULE_TOKEN"
   ```

7. Change the ownership of the new home directory from `root` to `pkiuser`.

   ```
   cd /usr/share/pki/pkiuser
   chown pkiuser pkiuser
   ```
8. Add the password for the ECC token to the instance's password file.

```bash
cd /usr/share/pki
chown -R pkiuser:pkiuser pkiuser
```

Add the password for the ECC token to the instance's password file.

```bash
vim /etc/instance_name/password.conf
```

```bash
hardware-THIRD_PARTY_MODULE_TOKEN=secret
```

The `hardware-` prefix is required.

9. Edit the instance configuration and add a line to require signature verification. For example:

```bash
cache.requestVerify.token=THIRD_PARTY_MODULE_TOKEN
```

10. Start the instance.

```bash
service instance_name start
```

11. Continue with the instance configuration, with two important configuration settings:
   - In the Key Store panel, the ECC module should be listed as an available token. Select that module for the key store.
   - In the Key Pairs panel, ECC should be listed as an option to use to generate the keys used for the CA's certificates. Select the ECC key type.

12. After completing the configuration for the instance, assuming it is a Java subsystem, try to log into the console.

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

This fails, because the console is not yet configured to run with ECC enabled. However, this does create the security databases for the console, so the ECC module can be loaded.

13. Load the ECC module into the console security databases.

```bash
cd ~/.redhat-idm-console/
modutil -dbdir . -nocertdb -add THIRD_PARTY_MODULE -libfile
/usr/lib/libYourNewModule.so
```

Now, logging into the console succeeds.

### 9.2. Loading the Certicom ECC Module

Certicom's ECC module has a slightly different configuration process than the procedure for loading a general ECC module.

1. Copy the third-party libraries to a common directory, like `/usr/lib` for 32-bit systems or `/usr/lib64` for 64-bit systems.

   There are two library files for the Certicom ECC modules, `libsbcpgse.so` and `libbgse2.so`.

2. Cache the recent shared libraries.

   ```bash
   ldconfig
   ```

3. Install the instance, but do not go through the configuration wizard.
4. **Stop the instance.**

   ```
   service instance_name stop
   ```

5. **The instance runs as the `pkiuser` user. As `root`, create a home directory for `pkiuser`.**

   ```
   /sbin/usermod --home /usr/share/pki/pkiuser pkiuser
cd /usr/share/pki
mkdir pkiuser
HOME=/usr/share/pki/pkiuser
export HOME
   ```

6. **Open the subsystem's `alias` directory. For example:**

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

7. **Install the third-party module in the CA's security databases so it is available for the configuration.**

   ```
   modutil -dbdir . -nocertdb -add certicom -libfile /usr/lib/libsbcpgse.so
   ```

   This creates a `.certicom` directory in the new `pkiuser` home directory.

8. **Certicom's ECC module includes an initpin file; copy this into the new `pkiuser` directory and give it execute permissions. For example:**

   ```
   cp /tmp/initpin /usr/share/pki/pkiuser
chmod +x initpin
   ```

9. **Run Certicom's `initpin` file from the `/usr/share/pki/pkiuser` directory. This first prompts for the directory to use for the Certicom token databases; use the `pkiuser` home directory, `/usr/share/pki/pkiuser`. This also prompts to set a password for the module, and then proceed with configuring the module.**
Security Builder API for PKCS #11 Samples
   CryptoAes() success
   CryptoArc4() success
   CryptoDes() success
   CryptoDh() success
   CryptoDsa() success
   CryptoEcdh() success
   CryptoEcdsa() success
   CryptoEcmqV() success
   CryptoPkcs1Enc() success
   CryptoPkcs1Sig() success
   CryptoRsaEnc() success
   CryptoRsaSig() success
   CryptoSha1() success

Token() samples starting
Slot info for Slot 0
Desc: FIPS Generic Crypto Services V2.0.1d
manufacturerID:  Certicom Corp.
flags:           0x1
                  CKF_TOKEN_PRESENT
hardwareVersion: 1.0
...
umask 00002
NSS_USE_DECODED_CKA_EC_POINT=1
export NSS_USE_DECODED_CKA_EC_POINT

15. Start the instance.

    service instance_name start

16. Continue with the instance configuration, with two important configuration settings:
   - In the Key Store panel, the ECC module should be listed as an available token. Select that module for the key store.
   - In the Key Pairs panel, ECC should be listed as an option to use to generate the keys used for the CA’s certificates. Select the ECC key type.

17. After completing the configuration, assuming this is a Java subsystem, try to log into the subsystem console.

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

This fails, because the console is not yet configure to run in ECC. However, this does create the security databases for the console, so the ECC module can be loaded.

Load the ECC module into the console security databases.

    cd ~/.redhat-idm-console/
    modutil -dbdir . -nocertdb -add certicom -libfile /usr/lib/libsbcpgse.so

Now, logging into the console succeeds.

18. The web browser used to access administrative and agent services pages also needs to be configured to support ECC.
   a. Create a user for the browser profile, such as agent-pki.
   b. Launch Firefox and create a profile for this user; this automatically creates the required security databases and directory.
   c. Set the root home directory to /home/agent-pki, and make sure the directory is owned by root.

    chown -R root:root /home/agent-pki

d. Copy the ECC module libraries and initpin file to the /home/agent-pki directory. All these files should be owned by root.

e. Load the ECC module.

    modutil -dbdir /home/agent-pki/.mozilla/profile.default -nocertdb -add certicom -libfile /usr/lib/libsbcpgse.so

f. Run the initpin file. When prompted, enter the Certicom token database directory, /usr/share/pki/pkiuser, and enter the PIN configured for those databases.

    ./initpin

g. Change the ownership of the new user’s home directory from root to the user. For example:
chown -R agent-pki:agent-pki /home/agent-pki

h. In the terminal with the /home/agent-pki directory open, export the environment variable that allows ECC support.

    export NSS_USE_DECODED_CKA_EC_POINT=1

i. Open Firefox again. The Certicom module should be available and you should be able to log into it successfully.

j. Then, import the agent certificate and root CA certificate or certificate chain into Firefox so that the user profile can access the agent services pages.

19. The NSS_USE_DECODED_CKA_EC_POINT environment variable also needs to be set to access the subsystem Java console with an ECC certificate. This can be set in the .bashrc file for the user who uses the console. For example:

    [root@server ~]# vim /home/jsmith/.bashrc

    # User specific aliases and functions
    NSS_USE_DECODED_CKA_EC_POINT=1
    export NSS_USE_DECODED_CKA_EC_POINT

20. Configure the appropriate SE Linux policies and settings.

a. The Certicom ECC library stores some of its data in the user's home directory. However, this directory is not defined in the Certificate System SE Linux file contexts, so some operations could be prevented from accessing the libraries. To avoid this, relabel the files to allow the appropriate SE Linux context so that the subsystem processes can access the libraries. For example:

    [root@server ~]# /usr/sbin/semanage fcontext -a -t pki_ca_t /usr/share/pki/pki.db

b. Update the contexts to allow the Certicom client to have write access to the Certificate System user directory, so it can maintain the Certicom libraries.

    [root@server ~]# /usr/sbin/semanage fcontext -a -t pki_common_t /usr/share/pki/.certicom\([^/]*\)?
    [root@server ~]# restorecon -r -v /usr/share/pki/.certicom

c. Then, enable enforcing mode by setting the SELINUX parameter in the SE Linux configuration file.

    [root@server ~]# vim /etc/selinux/config

    SELINUX=enforcing

9.3. Using ECC with an HSM

The HSMs supported by Certificate System (LunaSA and nCipher) support their own native ECC modules. This means that it is not necessary to load an independent ECC module for use with an HSM, but it is still necessary to configure the subsystem to use the ECC module with the token.

1. Install the cryptographic device, using the manufacturer's instructions. Be sure to name the token
something that will help identify it easily later.

2. Install the PKCS #11 module on the subsystem using the `modutil` command-line utility.
   a. Open the `alias` directory for the subsystem which is being configured with the PKCS #11 module:
      ```bash
      cd /var/lib/instance_name/alias
      ```
   b. The required security module database file, `secmod.db`, should be created by default when the subsystem is created. If it does not exist, use the `modutil` utility to create `secmod.db`.
      ```bash
      modutil -dbdir . -nocertdb -create
      ```
   c. Use the `modutil` utility to set the library information.
      ```bash
      modutil -dbdir . -nocertdb / -add module_name -libfile library_file
      ```
      `library_file` specifies the path to the library file containing the PKCS #11 interface module and `module_name` gives the name of the PKCS #11 module which was set when the drivers were installed.
      For the LunaSA HSM, the library can be `/usr/lunasa/lib/libCryptoki2_64.so` or `/usr/lunasa/lib/libCryptoki2.so`:
      ```bash
      modutil -dbdir . -nocertdb -add lunasa -libfile
      /usr/lunasa/lib/libCryptoki2.so
      ```
      For an nCipher HSM:
      ```bash
      modutil -dbdir . -nocertdb -add nethsm -libfile
      /opt/nfast/toolkits/pkcs11/libcknfast.so
      ```

3. Install the instance, but do not go through the configuration wizard.

4. Stop the instance.
   ```bash
   service instance_name stop
   ```

5. Edit the `CS.cfg` configuration and add a line to require signature verification. In this file, spaces and special characters do not need to be escaped. For example:
   ```bash
   ca.requestVerify.token=module name
   ```

6. Start the instance.
   ```bash
   service instance_name start
   ```

7. Continue with the instance configuration, with two important configuration settings:
   - In the **Key Store** panel, the ECC module should be listed as an available token. Select that module for the key store.
   - In the **Key Pairs** panel, ECC should be listed as an option to use to generate the keys used for the CA's certificates. Select the ECC key type.

Section 8.1, “Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates” describes
how to set up a hardware token to use with a subsystem.
Chapter 10. Cloning Subsystems

When a new subsystem instance is first configured, the Red Hat Certificate System allows subsystems to be cloned, or duplicated, for high availability of the Certificate System. The cloned instances run on different machines to avoid a single point of failure and their databases are synchronized through replication.

10.1. About Cloning

Planning for *high availability* reduces unplanned outages and other problems by making one or more subsystem clones available. When a host machine goes down, the cloned subsystems can handle requests and perform services, taking over from the master (original) subsystem seamlessly and keeping uninterrupted service.

Using cloned subsystems also allows systems to be taken offline for repair, troubleshooting, or other administrative tasks without interrupting the services of the overall PKI system.

**NOTE**

All of the subsystems except the TPS and RA can be cloned.

Cloning is one method of providing scalability to the PKI by assigning the same task, such as handling certificate requests, to separate instances on different machines. The internal databases for the master and its clones are replicated between each other, so the information about certificate requests or archived keys on one subsystem is available on all the others.

Typically, master and cloned instances are installed on different machines, and those machines are placed behind a *load balancer*. The load balancer accepts HTTP and HTTPS requests made to the Certificate System subsystems and directs those requests appropriately between the master and cloned instances. In the event that one machine fails, the load balancer transparently redirects all requests to the machine that is still running until the other machine is brought back online.
Figure 10.1. Cloning Example

The load balancer in front of a Certificate System subsystem is what provides the actual failover support in a high availability system. A load balancer can also provide the following advantages as part of a Certificate System subsystem:

- DNS round-robin, a feature for managing network congestion that distributes load across several different servers.
- Sticky SSL, which makes it possible for a user returning to the system to be routed the same host used previously.

10.1.1. Cloning for CAs

Cloned instances have the exact same private keys as the master, so their certificates are identical. For CAs, that means that the CA signing certificates are identical for the original master CA and its cloned CAs. From the perspectives of clients, these look like a single CA.

Every CA, both cloned and master, can issue certificates and process revocation requests.

The main issue with managing cloned CAs is how to assign serial numbers to the certificates they issue. Different CAs can have different levels of traffic, using serial numbers at different rates, and it is imperative that no CA issue the certificates with the same serial number. These serial number ranges are assigned and managed dynamically by using a shared, replicated entry that defines the ranges for
each CA and the next available range to reassign when one CA range runs low.

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

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

```plaintext
dbs.beginRequestNumber=1
dbs.beginSerialNumber=1
dbs.enableSerialManagement=true
dbs.endRequestNumber=9980000
dbs.endSerialNumber=ffe0000
dbs.replicaCloneTransferNumber=5
```

Cloned CAs do have limits on what operations they can perform. Most important, cloned CAs cannot generate or publish CRLs. Any CRL requests submitted to a cloned CA are immediately redirected to the master CA. Anything related to generating or caching CRLs is disabled in the `CS.cfg` file for the clone. Clones can revoke, display, import, and download CRLs previously generated by master CAs, but having them generate new CRLs may cause synchronization problems. Only a single CA should generate CRLs, and this task is always left to the master CA, which also maintains the CRL cache.

Master CAs also manage the relationships and information sharing between the cloned CAs by monitoring replication changes to the internal databases.

**TIP**

If a CA which is a security domain master is cloned, then that cloned CA is also a security domain master. In that case, both the original CA and its clone share the same security domain configuration.

**IMPORTANT**

As covered in Section 10.1.7, "Custom Configuration and Clones", the data within the LDAP database is replicated among a master and clones, but the `configuration files` for the different instances are not replicated. This means that any changes which affect the `CS.cfg` file — such as adding a DRM connection or creating a custom profile — are not copied into a clone’s configuration if the change occurs after the clone is created.

Any changes to the CA configuration should either be made to the master before any clones are created (so the custom changes are included in the cloning process) or any configuration changes must be copied over manually to the clone instances after they are created.

### 10.1.2. Cloning for DRMs

With DRMs, all keys archived in one DRM are replicated to the internal databases of the other DRMs. This allows a key recovery to be initiated on any clone DRM, regardless of which DRM the key was originally archived on.

After a key recovery is processed, the record of the recovery is stored in the internal database of all of
the cloned DRMs.

Although key recovery can be initiated on any clone, once the recovery is initiated, it must be completed on the same single DRM. This is because a recovery operation is recorded in the replicated database only after the appropriate number of approvals have been obtained from the DRM agents. Until then, the DRM on which the recovery is initiated is the only one which knows about the recovery operation.

### 10.1.3. Cloning for Other Subsystems

There is no real operational difference between masters and clones for TKSs; the information created or maintained on one is replicated along the other servers.

For OCSPs, only the master OCSP receives CRL updates, and then the published CRLs are replicated to the clones.

### 10.1.4. Cloning and Key Stores

Cloning a subsystem creates two server processes performing the same functions: another new instance of the subsystem is created and configured to use the same keys and certificates to perform its operations. Depending on where the keys are stored for the master clone, the method for the clone to access the keys is very different.

If the keys and certificates are stored in the internal software token, then the keys must be exported from the master subsystem when it is first configured. When configuring the master instance, there is an option in the Export Keys and Certificates panel to back up the keys and certificates to a PKCS#12 file. Before the clone instance is configured, the PKCS#12 file is copied to the alias/ directory for the clone instance. Then, the PKCS#12 filename is given in the Restore Keys and Certificates screen during the clone’s configuration.

If the keys and certificates are stored on a hardware token, then the keys and certificates can be copied or referenced directly in the token:

- Duplicate all the required keys and certificates, except the SSL server key and certificate to the clone instance. Keep the nicknames for those certificates the same. Additionally, copy all the necessary trusted root certificates from the master instance to the clone instance, such as chains or cross-pair certificates.
- If the token is network-based, then the keys and certificates simply need to be available to the token; the keys and certificates do not need to be copied.
- When using a network-based hardware token, make sure the high-availability feature is enabled on the hardware token to avoid single point of failure.

### 10.1.5. LDAP and Port Considerations

As mentioned in Section 10.1, “About Cloning”, part of the behavior of cloning is to replication information between the master and the clone, so that they work from an identical set of data and records. This means that the LDAP servers for the master and clones need to be able to communicate.

If the Directory Server instances are on different hosts, then make sure that there is appropriate firewall access to allow the Directory Server instances to connect with each other.

---

**NOTE**

Cloned subsystems and their masters must use separate LDAP servers.
A subsystem can connect to its internal database using either SSL over an LDAPS port or over a standard connection over an LDAP port. When a subsystem is cloned, the clone instance uses the same connection method (SSL or standard) as its master (subsystem => database). With cloning, there is an additional database connection though: the master Directory Server database to the clone Directory Server database. For that connection, there are three connection options:

- If the master uses SSL to connect to its database, then the clone uses SSL, and the master/clone Directory Server databases use SSL connections for replication.
- If the master uses a standard connection to its database, then the clone must use a standard connection, and the Directory Server databases can use unencrypted connections for replication.
- If the master uses a standard connection to its database, then the clone must use a standard connection, but there is an option to use Start TLS for the master/clone Directory Server databases for replication. Start TLS opens a secure connection over a standard port.

**NOTE**

To use Start TLS, the Directory Server must still be configured to accept SSL connections, so it must have a server certificate and a CA certificate installed on the Directory Server and SSL must be enabled.

Whatever connection method (secure or standard) used by the master must be used by the clone and must be properly configured for the Directory Server databases.

**IMPORTANT**

Even if the clone connects to the master over a secure connection, the standard LDAP port (389 by default) must still be open and enabled on the LDAP server while cloning is configured. For secure environments, the standard LDAP port can be disabled on the master's Directory Server instance once the clone is configured.

### 10.1.6. Replica ID Numbers

Cloning is based on setting up a replication agreement between the Directory Server for the master instance and the Directory Server for the cloned instance.

Servers involved together with replication are in the same replication topology. Every time a subsystem instance is cloned, it is added to the overall topology. Directory Server discerns between different servers in the topology based on their replica ID number. This replica ID must be unique among all of the servers in the topology.

As with the serial number ranges used for requests and certificates (covered in [Section 10.1.1, “Cloning for CAs”](#)), every subsystem is assigned a range of allowed replica IDs. When the subsystem is cloned, it assigns one of the replica IDs from its range to the new clone instance.

```
dbs.beginReplicaNumber=1
dbs.endReplicaNumber=95
```

The replica ID range can be refreshed with new numbers if an instance begins to exhaust its current range.

### 10.1.7. Custom Configuration and Clones
After a clone is created, configuration changes are not replicated between clones or between a master and a clone. The instance configuration is in the CS.cfg file — outside the replicated database.

For example, there are two CAs, a master and a clone. A new DRM is installed which is associated, at its configuration, with the master CA. The CA-DRM connector information is stored in the master CA's CS.cfg file, but this connector information is not added to the clone CA configuration. If the DRM sends a key archival request to the master CA, the DRM is recognized and the request is approved. If the same DRM sends its key archival request to the clone CA, the DRM is not recognized and the key archival request is ignored.

**Changes made to the configuration of a master server or to a clone server are not replicated to other cloned instances.** Any critical settings must be added manually to clones.

### TIP

Set all desired, custom configuration for a master server before configuring any clones. For example, install all DRMs so that all the connector information is in the master CA configuration file, create any custom profiles, or configure all publishing points for a master OCSP responder. Any custom settings in the master instance will be included in the cloned instances **at the time they are cloned** (but not after).

#### 10.2. Exporting Keys from a Software Database

Ideally, the keys for the master instance are exported when the instance is first created. If the keys were not exported then or if the backup file is lost, then it is possible to extract the keys from the internal software database for the subsystem instance using the **PKCS12Export** command. For example:

```
PKCS12Export -debug -d /var/lib/instance_name/alias -w p12pwd.txt -p internal.txt -o master.p12
```

The PKCS#12 file (in this example, **master.p12**) can then be copied to the clone instance's **alias/** directory and imported during the clone configuration.

### NOTE

Keys and certificates do not need to be exported from an HSM, so long as the clone instance is installed using the same HSM as the master. If both instances use the same key store, then the keys are naturally available to the clone.

#### 10.3. Cloning a CA

1. Configure the master CA, and back up the keys.
2. In the **CS.cfg** file for the master CA, enable the master CA to monitor replication database changes by adding the **ca.listenToCloneModifications** parameter:

```
ca.listenToCloneModifications=true
```
3. Create the clone subsystem instance.

**IMPORTANT**

Do not go through the setup wizard for the instance yet.

4. Copy the exported PKCS#12 file containing the master instance's keys to the clone's alias/directory.

The keys for the master instance could have been exported to a .p12 file when the instance was configured. Alternatively, the keys can be exported using the PKCS12Export command, as in Section 10.2, “Exporting Keys from a Software Database”.

5. Make sure the PKCS#12 file is accessible by the Certificate System user. If necessary, change the file owner to pkiuser and reset the permissions to allow the correct read/write access. For example:

```bash
chown pkiuser:pkiuser example.p12
chmod 00644 example.p12
```

6. It may be necessary to reset the SELinux permissions for the exported file so that the setup program can use it. First, check what context is assigned:

```bash
ls -lZ *
```

If it does not match the other security files, then reset the SELinux context to that of the other objects using chcon:

```bash
chcon "system_u:object_r:pki_ca_var_lib_t:s0" example.p12
ls -lZ *
```

7. Open the setup wizard URL, which was returned when the instance was created. For example:

```bash
http://server.example.com:9180/ca/admin/console/config/login?pin=HIsd90RJSioDK==
```

8. In the Security Domain panel, add the clone to the same security domain to which the master belongs.

9. The Subsystem Type panel sets whether to create a new instance or a clone; select the clone radio button.
10. Give the path and filename of the PKCS #12 backup file which was saved when the master instance was created or that were exported in 4. If the keys are stored on an HSM that is accessible to the clone, then they are picked up automatically.

NOTE
When cloning a CA, the master and clone instances have the same CA signing key.

11. The subsystem information is automatically supplied from the master instance to the clone instance once the keys are successfully restored. Complete the configuration process. When configuring the LDAP database, there are three critical configuration changes that must be made:
   - The clone must use a different Directory Server instance but must have the same suffix name.
By default, the instance configuration wizard uses localhost as the location for the internal LDAP database for a new instance. However, with cloning, the configuration process will spin endlessly and never complete if localhost is used for the internal database location, even if the LDAP database is indeed installed on the localhost.

Use the fully-qualified domain name for the LDAP database in the Internal Database panel when configuring a clone.

The subsystem can connect to its database over a special secure port using SSL or over the regular port. However, the clone can only use a secure connection if the master was first set up to use a secure connection to the database, and it must use the same type of connection (SSL or unencrypted) as the master. If the master and clone use a regular, unencrypted connection, then the clone has the option to use Start TLS (a secure connection over an unsecure port) for replication between the master Directory Server database and the clone database.

**IMPORTANT**

Even if the clone connects to the master over a secure connection, the standard LDAP port (389 by default) must still be open and enabled on the LDAP server while cloning is configured.

For secure environments, the standard LDAP port can be disabled on the master’s Directory Server instance once the clone is configured.

12. Edit the Cs.cfg file for the clone. Certain parameters must be added to the clone configuration to disable caching and generating CRLs.

   - Disable control of the database maintenance thread:
     
     \[
     \text{ca.certStatusUpdateInterval}=0
     \]

   - Disable monitoring database replication changes:
     
     \[
     \text{ca.listenToCloneModifications}=false
     \]

   - Disable maintenance of the CRL cache:
     
     \[
     \text{ca.crl.\textit{IssuingPointId}}.enableCRLCache=false
     \]

   - Disable CRL generation:
     
     \[
     \text{ca.crl.\textit{IssuingPointId}}.enableCRLUpdates=false
     \]

   - Enable the clone to redirect CRL requests to the master clone:
     
     \[
     \text{master.ca.agent.host}=\text{\textit{master_hostname}}
     \text{master.ca.agent.port}=\text{\textit{master_port}}
     \]

13. Restart the Directory Server instance used by the clone.

    \[
    \text{service instance\_name restart drm-clone-ds-instance}
    \]
14. Restart the clone instance.

```
service instance_name restart
```

After configuring the clone, test to make sure that the master-clone relationship is functioning:

1. Request a certificate from the cloned CA.
2. Approve the request.
3. Download the certificate to the browser.
4. Revoke the certificate.
5. Check master CA's CRL for the revoked certificate. In the master Certificate Manager's agent services page, click **Update Certificate Revocation List**. Find the CRL in the list.
   The CRL should show the certificate revoked by the cloned Certificate Manager. If that certificate is not listed, check logs to resolve the problem.

### 10.4. Updating CA-DRM Connector Information After Cloning

As covered in Section 10.1.7, “Custom Configuration and Clones”, configuration information is not updated in clone instances if it is made after the clone is created. Likewise, changes made to a clone are not copied back to the master instance.

If a new DRM is installed or cloned after a clone CA is created, then the clone CA does not have the new DRM connector information in its configuration. This means that the clone CA ignores any archival requests from the DRM because it does not recognize it as a legitimate client.

Whenever a new DRM is created or cloned, copy its connector information into all of the cloned CAs in the deployment.

1. On the master clone machine, open the master CA's `CS.cfg` file, and copy all of the `ca.connector.KRA.*` lines for the new DRM connector.
   ```
   [root@master ~]# vim /var/lib/pki-ca/conf/CS.cfg
   ```
2. Stop the clone CA instance. For example:
   ```
   [root@clone-ca ~] service pki-ca stop
   ```
3. Open the clone CA's `CS.cfg` file.
   ```
   [root@clone-ca ~]# vim /var/lib/pki-ca/conf/CS.cfg
   ```
4. Copy in the connector information for the new DRM instance or clone.
5. Start the clone CA.

    [root@clone-ca ~] service pki-ca start

### 10.5. Cloning OCSP Subsystems

1. Configure the master OCSP, and back up the keys.

2. In the CS.cfg file for the master OCSP, set the `OCSP.Responder.store.defStore.refreshInSec` parameter to any non-zero number other than 21600; 21600 is the setting for a clone.

    
    ```
    vim /etc/instance_name/CS.cfg
    OCSP.Responder.store.defStore.refreshInSec=15000
    ```

3. Create the clone subsystem instance.

   **IMPORTANT**

   Do not go through the setup wizard for the instance yet.

4. Copy the exported PKCS#12 file containing the master instance's keys to the clone's alias/ directory.

   The keys for the master instance could have been exported to a .p12 file when the instance was configured. Alternatively, the keys can be exported using the `PKCS12Export` command, as in Section 10.2, “Exporting Keys from a Software Database”.

5. Make sure the PKCS#12 file is accessible by the Certificate System user. If necessary, change the file owner to `pkiuser` and reset the permissions to allow the correct read/write access. For example:

    ```
    chown pkiuser:pkiuser example.p12
    chmod 06644 example.p12
    ```

6. It may be necessary to reset the SELinux permissions for the exported file so that the setup program can use it. First, check what context is assigned:

    ```
    ls -1Z *
    -rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 cert8.db
    -rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 key3.db
    -rw-r--r--. pkiuser pkiuser system_u:object_r:nfs_t:s0 example.p12
    -rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 secmod.db
    ```
If it does not match the other security files, then reset the SELinux context to that of the other objects using `chcon`:

```
chcon "system_u:object_r:pki_ocsp_var_lib_t:s0" example.p12
```

```
ls -lZ *
```

```
-rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 cert8.db
-rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 key3.db
-rw-r--r--. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 example.p12
-rw-------. pkiuser pkiuser system_u:object_r:pki_ocsp_var_lib_t:s0 secmod.db
```

7. Open the setup wizard URL, which was returned when the instance was created. For example:

```
http://server.example.com:11180/ocsp/admin/console/config/login?pin=IOjh7fIOjkld90k
```

8. In the **Security Domain** panel, add the clone to the same security domain to which the master belongs.

9. The **Subsystem Type** panel sets whether to create a new instance or a clone; select the clone radio button.

10. Give the path and filename of the PKCS #12 backup file which was saved when the master instance was created or that were exported in 3.
    If the keys are stored on an HSM that is accessible to the clone, then they are picked up automatically.
11. The subsystem information is automatically supplied from the master instance to the clone instance once the keys are successfully restored. Complete the configuration process.

When configuring the LDAP database, there are three critical configuration changes that must be made:

- The clone must use a different Directory Server instance but must have the same suffix name.
- By default, the instance configuration wizard uses localhost as the location for the internal LDAP database for a new instance. However, with cloning, the configuration process will spin endlessly and never complete if localhost is used for the internal database location, even if the LDAP database is indeed installed on the localhost.

Use the fully-qualified domain name for the LDAP database in the Internal Database panel when configuring a clone.

- The subsystem can connect to its database over a special secure port using SSL or over the regular port. However, the clone can only use a secure connection if the master was first set up to use a secure connection to the database, and it must use the same type of connection (SSL or unencrypted) as the master. If the master and clone use a regular, unencrypted connection, then the clone has the option to use Start TLS (a secure connection over an unsecure port) for replication between the master Directory Server database and the clone database.

**IMPORTANT**

Even if the clone connects to the master over a secure connection, the standard LDAP port (389 by default) must still be open and enabled on the LDAP server while cloning is configured.

For secure environments, the standard LDAP port can be disabled on the master's Directory Server instance once the clone is configured.

12. Edit the CS.cfg file for the clone. Set the `OCSP.Responder.store.defStore.refreshInSec` parameter in the clone instance to 21600.

```bash
vim /etc/instance_name/CS.cfg
OCSP.Responder.store.defStore.refreshInSec=21600
```

13. Restart the Directory Server instance used by the clone.
NOTE

Restarting the Directory Server reloads the updated schema, which is required for proper performance.

14. Restart the clone instance.

    service instance_name start

After configuring the clone, test to make sure that the master-clone relationship is functioning:

1. Set up OCSP publishing in the master CA so that the CRL is published to the master OCSP.
2. Once the CRL is successfully published, check both the master and cloned OCSP’s List Certificate Authorities link in the agent pages. The list should be identical.
3. Use the OCSPClient tool to submit OCSP requests to the master and the cloned Online Certificate Status Manager. The tool should receive identical OCSP responses from both managers.

10.6. Cloning DRM Subsystems

1. Configure the master subsystem, and back up the keys.
2. Create the clone subsystem instance.

IMPORTANT

Do not go through the setup wizard for the instance yet.

3. Copy the exported PKCS#12 file containing the master instance's keys to the clone's alias/ directory.
   
   The keys for the master instance could have been exported to a .p12 file when the instance was configured. Alternatively, the keys can be exported using the PKCS12Export command, as in Section 10.2, “Exporting Keys from a Software Database”.
4. Make sure the PKCS#12 file is accessible by the Certificate System user. If necessary, change the file owner to pkiuser and reset the permissions to allow the correct read/write access. For example:

   chown pkiuser:pkiuser example.p12
   chmod 00644 example.p12

5. It may be necessary to reset the SELinux permissions for the exported file so that the setup program can use it. First, check what context is assigned:
If it does not match the other security files, then reset the SELinux context to that of the other objects using `chcon`:

```
chcon "system_u:object_r:pki_kra_var_lib_t:s0" example.p12
```

6. Open the setup wizard URL, which was returned when the instance was created. For example:

```
http://server.example.com:10180/kra/admin/console/config/login?pin=Jhu7HBJiodljnw3oijs
```

7. In the **Security Domain** panel, add the clone to the same security domain to which the master belongs.

8. The **Subsystem Type** panel sets whether to create a new instance or a clone; select the clone radio button.

9. Give the path and filename of the PKCS #12 backup file which was saved when the master instance was created or that were exported in 3.

   If the keys are stored on an HSM that is accessible to the clone, then they are picked up.
automatically.

![Restore Keys and Certificates](image)

### NOTE

When cloning a DRM, the master and clone instances have the same storage and transport keys.

10. The subsystem information is automatically supplied from the master instance to the clone instance once the keys are successfully restored. Complete the configuration process.

When configuring the LDAP database, there are three critical configuration changes that must be made:

- The clone must use a different Directory Server instance but must have the same suffix name.
- By default, the instance configuration wizard uses `localhost` as the location for the internal LDAP database for a new instance. However, with cloning, the configuration process will spin endlessly and never complete if localhost is used for the internal database location, even if the LDAP database is indeed installed on the localhost.

Use the fully-qualified domain name for the LDAP database in the Internal Database panel when configuring a clone.

- The subsystem can connect to its database over a special secure port using SSL or over the regular port. However, the clone can only use a secure connection if the master was first set up to use a secure connection to the database, and it must use the same type of connection (SSL or unencrypted) as the master. If the master and clone use a regular, unencrypted connection, then the clone has the option to use Start TLS (a secure connection over an unsecure port) for replication between the master Directory Server database and the clone database.

### IMPORTANT

Even if the clone connects to the master over a secure connection, the standard LDAP port (389 by default) must still be open and enabled on the LDAP server while cloning is configured.

For secure environments, the standard LDAP port can be disabled on the master's Directory Server instance once the clone is configured.

11. Restart the Directory Server instance used by the clone.
service instance_name restart

NOTE

Restarting the Directory Server reloads the updated schema, which is required for proper performance.

12. Restart the clone instance.

service instance_name restart

For the DRM clone, test to make sure that the master-clone relationship is functioning:

1. Go to the DRM agent's page.
2. Click List Requests.
3. Select Show all requests for the request type and status.
4. Click Submit.
5. Compare the results from the cloned DRM and the master DRM. The results ought to be identical.

10.7. Cloning TKS Subsystems

1. Configure the master subsystem, and back up the keys.
2. Create the clone subsystem instance.

IMPORTANT

Do not go through the setup wizard for the instance yet.

3. Copy the exported PKCS#12 file containing the master instance's keys to the clone's alias/directory.
   The keys for the master instance could have been exported to a .p12 file when the instance was configured. Alternatively, the keys can be exported using the PKCS12Export command, as in Section 10.2, “Exporting Keys from a Software Database”.
4. Make sure the PKCS#12 file is accessible by the Certificate System user. If necessary, change the file owner to pkiuser and reset the permissions to allow the correct read/write access. For example:

   chown pkiuser:pkiuser example.p12
   chmod 00644 example.p12

5. It may be necessary to reset the SELinux permissions for the exported file so that the setup program can use it. First, check what context is assigned:

   ls -1Z *
   -rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 cert8.db
   -rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 key3.db
   -rw-r--r--. pkiuser pkiuser system_u:object_r:nfs_t:s0 example.p12
   -rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 secmod.db
If it does not match the other security files, then reset the SELinux context to that of the other objects using **chcon**:

```
chcon "system_u:object_r:pki_tks_var_lib_t:s0" example.p12
```

```
ls -lZ *
-rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 cert8.db
-rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 key3.db
-rw-r--r--. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 example.p12
-rw-------. pkiuser pkiuser system_u:object_r:pki_tks_var_lib_t:s0 secmod.db
```

6. Open the setup wizard URL, which was returned when the instance was created. For example:

```
http://server.example.com:13180/tks/admin/console/config/login?pin=Jhu7HBJiodljnw3oijs
```

7. In the **Security Domain** panel, add the clone to the same security domain to which the master belongs.

8. The **Subsystem Type** panel sets whether to create a new instance or a clone; select the clone radio button.

9. Give the path and filename of the PKCS #12 backup file which was saved when the master instance was created or that were exported in step 3.

   If the keys are stored on an HSM that is accessible to the clone, then they are picked up automatically.
10. The subsystem information is automatically supplied from the master instance to the clone instance once the keys are successfully restored. Complete the configuration process.

When configuring the LDAP database, there are three critical configuration changes that must be made:

- The clone must use a different Directory Server instance but must have the same suffix name.
- By default, the instance configuration wizard uses localhost as the location for the internal LDAP database for a new instance. However, with cloning, the configuration process will spin endlessly and never complete if localhost is used for the internal database location, even if the LDAP database is indeed installed on the localhost.
  Use the fully-qualified domain name for the LDAP database in the Internal Database panel when configuring a clone.
- The subsystem can connect to its database over a special secure port using SSL or over the regular port. However, the clone can only use a secure connection if the master was first set up to use a secure connection to the database, and it must use the same type of connection (SSL or unencrypted) as the master. If the master and clone use a regular, unencrypted connection, then the clone has the option to use Start TLS (a secure connection over an unsecure port) for replication between the master Directory Server database and the clone database.

**IMPORTANT**

Even if the clone connects to the master over a secure connection, the standard LDAP port (389 by default) must still be open and enabled on the LDAP server while cloning is configured.

For secure environments, the standard LDAP port can be disabled on the master’s Directory Server instance once the clone is configured.

11. Restart the clone instance.

```
service instance_name restart
```

For the TKS, enroll a smart card and then run an `ldapsearch` to make sure that the same key information is contained in both databases.

### 10.8. Converting Masters and Clones

There can be any number of clones, but there can only be a single configured master. For DRMs and
TKSs, there is no configuration difference between masters and clones, but CAs and OCSPs do have some configuration differences. This means that when a master is taken offline — because of a failure or for maintenance or to change the function of the subsystem in the PKI — then the existing master must be reconfigured to be a clone, and one of the clones promoted to be the master.

10.8.1. Converting CA Clones and Masters

1. Stop the master CA if it is still running.

2. Open the existing master CA configuration directory:

   ```
   cd /var/lib/pki-ca/conf
   ```

3. Edit the `CS.cfg` file for the master, and change the CRL and maintenance thread settings so that it is set as a clone:

   - Disable control of the database maintenance thread:
     ```
     ca.certStatusUpdateInterval=0
     ```
   - Disable monitoring database replication changes:
     ```
     ca.listenToCloneModifications=false
     ```
   - Disable maintenance of the CRL cache:
     ```
     ca.crl.IssuingPointId.enableCRLCache=false
     ```
   - Disable CRL generation:
     ```
     ca.crl.IssuingPointId.enableCRLUpdates=false
     ```
   - Set the CA to redirect CRL requests to the new master:
     ```
     master.ca.agent.host=new_master_hostname
     master.ca.agent.port=new_master_port
     ```

4. Stop the cloned CA server.

   ```
   service instance_name stop
   ```

5. Open the cloned CA's configuration directory.

   ```
   cd /etc/instance_name
   ```

6. Edit the `CS.cfg` file to configure the clone as the new master.

   a. Delete each line which begins with the `ca.crl.` prefix.
   b. Copy each line beginning with the `ca.crl.` prefix from the former master CA `CS.cfg` file into the cloned CA's `CS.cfg` file.
   c. Enable control of the database maintenance thread; the default value for a master CA is 600.

   ```
   ca.certStatusUpdateInterval=600
   ```
d. Enable monitoring database replication:

```
calistenToCloneModifications=true
```

e. Enable maintenance of the CRL cache:

```
cacr.IssuingPointId.enableCRLCache=true
```

f. Enable CRL generation:

```
cacr.IssuingPointId.enableCRLUpdates=true
```

g. Disable the redirect settings for CRL generation requests:

```
master.ca.agent.host=hostname
master.ca.agent.port=port number
```

7. Start the new master CA server.

```
service instance_name start
```

## 10.8.2. Converting OCSP Clones

1. Stop the OCSP master, if it is still running.
2. Open the existing master OCSP configuration directory.

```
cd /etc/instance_name
```

3. Edit the CS.cfg, and reset the `OCSP.Responder.store.defStore.refreshInSec` parameter to 21600:

```
OCSP.Responder.store.defStore.refreshInSec=21600
```

4. Stop the online cloned OCSP server.

```
service instance_name stop
```

5. Open the cloned OCSP responder's configuration directory.

```
cd /etc/instance_name
```

6. Open the CS.cfg file, and delete the `OCSP.Responder.store.defStore.refreshInSec` parameter or change its value to any non-zero number:

```
OCSP.Responder.store.defStore.refreshInSec=15000
```

7. Start the new master OCSP responder server.

```
service instance_name start
```

## 10.9. Cloning a CA That Has Been Re-Keyed
When a certificate expires, it has to be replaced. This can either be done by renewing the certificate, which re-uses the original keypair to generate a new certificate, or it can be done by generating a new keypair and certificate. The second method is called **re-keying**.

When a CA is re-keyed, new keypairs are stored in its certificate database, and these are the keys references for normal operations. However, for cloning a subsystem, the cloning process checks for the CA private keys as stored in its `CS.cfg` configuration file — and those keys are not updated when the certificate database keys change.

If a CA has been re-keyed and then an administrator attempts to clone it, the cloned CA fails to generate any certificates for the certificates which were re-keyed, and it shows up in the error logs with this error:

```
CertUtil::createSelfSignedCert() - CA private key is null!
```

To clone a CA that has been re-keyed:

1. Find all of the private keys in the `CS.cfg` file.

   ```
   # grep privkey.id /var/lib/pki-ca/conf/CS.cfg
   cloning.signing.privkey.id       =-4d798441a7236910d4e1c39fa132ea228d5d1bc
   cloning.ocsp_signing.privkey.id  =-3e23e743e0dd88f2a7c6f69fa9f9cbebfa1a60
   cloning.subsystem.privkey.id     =-c3cb3b4d8f5dd6d2bdef07581c80b1552536
   cloning.sslserver.privkey.id     =3023d30245804afab42be09eb0d683423a8f
   cloning.audit_signing.privkey.id=2fe35d9d46b373efabe9ef01b8436667a70df96
   ```

2. Print all of the current private keys stored in the NSS database and compare them to the private keys stored in the `CS.cfg` file:

   ```
   # certutil -K -d alias
   certutil: Checking token "NSS Certificate DB" in slot "NSS User Private Key and Certificate Services"
   Enter Password or Pin for "NSS Certificate DB": 
   < 0> rsa a7b0944b7b8397729a4c8c9af3a9c2b96f49c6f3   caSigningCert cert-
            ca4-test-master
   < 1> rsa 6006094af3e5d02aaa91426594ca66cb53e73ac0   ocspSigningCert
            cert-ca4-test-master
   < 2> rsa d684da39bf42789a3fc9d42204596f4578ad2d9   subsystemCert
            cert-ca4-test-master
   < 3> rsa a8edd7c2b5c94f13144cad699624578ae30b7e43   sslserverCert
            cert-ca4-test1
   < 4> rsa 2fe35d9d46b373efabe9ef01b8436667a70df96   auditSigningCert
            cert-ca4-test1
   ```

   In this example, only the audit signing key is the same; the others have been changed.

3. Take the keys returned in step 2 and convert them from unsigned values (which is what `certutil` returns) to signed Java Bigintegers (which is how the keys are stored in the Certificate System database).

   This can be done with a calculator or by using the script in [Example 10.1, “Certutil to BigInteger Conversion Program”](#).

4. Copy the new key values into the `CS.cfg` file.
### Example 10.1. Certutil to BigInteger Conversion Program

This Java program can convert the key output from `certutil` to the required BigInteger format.

Save this as a `.java` file, such as `Test.java`.

```java
import java.math.BigInteger;

public class Test {
    public static byte[] hexStringToByteArray(String s) {
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4) + Character.digit(s.charAt(i+1), 16));
        }
        return data;
    }

    public static void main(String[] args) {
        byte[] bytes = hexStringToByteArray(args[0]);
        BigInteger big = new BigInteger (bytes);
        System.out.println("Result is ==> " + big.toString(16));
    }
}
```

Then, compile the file:

```
# javac Test.java
```

### 10.10. Updating CA Clones

When a CA is cloned, any configuration in its `CS.cfg` is also copied to the clone CA. This includes any DRM's which are configured for the CA to use for key archival. However, if a DRM is configured for a master CA after a clone is created, then the new DRM configuration must be copied over to the clone CAs manually.

To update any clone CAs with new DRM configuration in the master CA:

1. Stop the clone CA.
Always stop a subsystem instance before editing its configuration files.

2. Copy all of the `ca.connector.KRA.*` parameters for the new DRM connection in the `CS.cfg` for the master CA over to the clone CA `CS.cfg` file.

3. Restart the clone CA.
Chapter 11. Silently Configuring Instances

The Certificate System includes a tool, `pkisilent`, which configures an instance in a single step. Normally, instances are configured by accessing the subsystem HTML page and going through the setup wizard. `pkisilent` can be used to pass all of the configuration parameters to a new instance simply from the command line.

**NOTE**

The `pkisilent` script is downloaded and installed in its own package.

11.1. About pkisilent

*Silent configuration* sets up a new subsystem instance in a single pass, by sending all of the configuration parameters through the command line. For Certificate System subsystems, this is done using the `pkisilent` command.

The `pkisilent` command can configure the subsystem instance the same as if it were configured using the HTML-based configuration wizard, so it can create a new security domain or use an existing one, back up keys, create a clone, or use certificates issued by an external CA.

From a high level, the `pkisilent` command has groups of parameters that define major areas of the subsystem's default settings and users.

There are two template files that are shell scripts for silent configuration: `/usr/share/pki/silent/pki_silent.template` and `/usr/share/pki/silent/subca_silent.template`. Both of these templates have detailed information on parameters and usage options for `pkisilent`.

**Example 11.1. pkisilent Command**

```
pkisilent Configure typeregister -parameters to configure the subsystem URL ... -parameters to configure the admin user ... -parameters to configure the domain ... -parameters to configure the agent ... -parameters to configure the internal database ... -parameters to configure the subsystem keys, certificates, and key store
```

The options available to use with the `pkisilent` command are listed in *Table 11.1, “Parameters for pkisilent”*. 
There are two template files that are shell scripts for silent configuration:

```
/usr/share/pki/silent/pki_silent.template
```

and

```
/usr/share/pki/silent/subca_silent.template
```

Both of these templates have detailed information on parameters and usage options for `pkisilent`. To check the specific options for any `ConfigureType` option, just run the `pkisilent` command with the `ConfigureType` option and the `-help` flag. For example, to get the help for configuring a subordinate CA:

```
pkisilent ConfigureSubCA -help
```

The `ConfigureType` option sets what kind of subsystem is being configured. This can be any of the following:

- ConfigureCA (for a root CA) or ConfigureSubCA (for a subordinate CA)
- ConfigureRA
- ConfigureDRM
- ConfigureOCSP
- ConfigureTKS
- ConfigureTPS
### Table 11.1. Parameters for pkisilent

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Instance Configuration</strong></td>
<td></td>
</tr>
<tr>
<td>cs_hostname</td>
<td>The hostname for the Certificate System machine.</td>
</tr>
<tr>
<td>cs_port</td>
<td>The administrative SSL port number of the Certificate System instance.</td>
</tr>
<tr>
<td>subsystem_name</td>
<td>Sets the name of the new subsystem instance.</td>
</tr>
<tr>
<td>client_certdb_dir</td>
<td>The directory for the subsystem certificate databases.</td>
</tr>
<tr>
<td>client_certdb_pwd</td>
<td>The password to protect the certificate database.</td>
</tr>
<tr>
<td>preop_pin</td>
<td>The preoperation PIN number used for the initial configuration. This PIN is part of the output of <code>pkicreate</code>, at the end of the configuration URL. It can also be found in the URL in the installation file for the instance (<code>/var/log/pki-name/logs-install.log</code>).</td>
</tr>
<tr>
<td>token_name</td>
<td>Gives the name of the HSM token used to store the subsystem certificates. This is only required for hardware tokens; if this parameter is not given, then the script automatically uses the local software token.</td>
</tr>
<tr>
<td>token_pwd</td>
<td>Gives the password for the HSM.</td>
</tr>
<tr>
<td><strong>Agent and Admin User Configuration</strong></td>
<td></td>
</tr>
<tr>
<td>admin_user</td>
<td>The new admin user for the new subsystem.</td>
</tr>
<tr>
<td>admin_email</td>
<td>The email address of the admin user.</td>
</tr>
<tr>
<td>admin_password</td>
<td>The password for the admin user.</td>
</tr>
<tr>
<td>agent_key_size</td>
<td>The key size to use for generating the agent certificate and key pair.</td>
</tr>
<tr>
<td>agent_key_type</td>
<td>The key type to use for generating the agent certificate and key pair.</td>
</tr>
<tr>
<td>agent_cert_subject</td>
<td>The subject name for the agent certificate.</td>
</tr>
<tr>
<td><strong>Security Domain Configuration</strong></td>
<td></td>
</tr>
<tr>
<td>domain_name</td>
<td>The name of the security domain to which the subsystem will be added.</td>
</tr>
<tr>
<td>sd_hostname</td>
<td>The hostname of the CA which hosts security domain.</td>
</tr>
<tr>
<td>sd_admin_port</td>
<td>The administrative SSL port of the CA which hosts security domain.</td>
</tr>
<tr>
<td>sd_agent_port</td>
<td>The agent SSL port of the CA which hosts security domain.</td>
</tr>
<tr>
<td>sd_ssl_port</td>
<td>The end-entities SSL port of the CA which hosts security domain.</td>
</tr>
<tr>
<td>sd_admin_name</td>
<td>The username of the administrative user for the CA hosting the security domain.</td>
</tr>
<tr>
<td>sd_admin_password</td>
<td>The password of the administrative user for the CA hosting the security domain.</td>
</tr>
<tr>
<td><strong>Internal Database Configuration</strong></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ldap_host</td>
<td>The hostname of the Directory Server machine.</td>
</tr>
<tr>
<td>ldap_port</td>
<td>The non-SSL port of the Directory Server.</td>
</tr>
<tr>
<td>bind_dn</td>
<td>The bind DN which will access the Directory Server; this is normally the Directory Manager ID.</td>
</tr>
<tr>
<td>bind_password</td>
<td>The bind DN password.</td>
</tr>
<tr>
<td>base_dn</td>
<td>The entry DN under which to create all of the subsystem entries.</td>
</tr>
<tr>
<td>db_name</td>
<td>The database name.</td>
</tr>
<tr>
<td>secure_conn</td>
<td>Whether to use SSL to connect to the internal database. This is either true or false.</td>
</tr>
<tr>
<td>remove_data</td>
<td>Whether to overwrite the data if a database of the same name exists.</td>
</tr>
</tbody>
</table>

### Subsystem Certificates and Keys Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_size</td>
<td>The size of the key to generate. The recommended size for an RSA key is 1048 bits for regular operations and 2048 bits for sensitive operations.</td>
</tr>
<tr>
<td>key_type</td>
<td>The type of key to generate; the only option is RSA.</td>
</tr>
<tr>
<td>key_algorithm</td>
<td>The hashing algorithm to use for the key pair. This is only used for root CA subsystems; hashing algorithms for other subsystems and sub CAs are set by editing the certificate profile. For RSA:</td>
</tr>
</tbody>
</table>
|              | ‣ SHA256withRSA  
|              | ‣ SHA1withRSA    
|              | ‣ SHA256withRSA  
|              | ‣ SHA512withRSA  
|              | ‣ MD5withRSA     
|              | ‣ MD2withRSA     

For ECC:

|              | ‣ SHA256withEC (the default)  
|              | ‣ SHA1withEC                
|              | ‣ SHA384withEC              
|              | ‣ SHA512withEC              |

| key_curvename | For ECC keys. The curve to use for the key. The default is nistp256.                                                                    |

<p>| signing_key_type | For CA signing certificates. CAs only. Sets the specific settings to generate a CA signing key and certificate. |
| signing_key_size | The key_type, key_size, key_algorithm, and key_curvename parameters apply to every key and certificate generated by a subsystem. However, each individual key can have its own parameters set separately. The parameters available to key_type, key_size, |
| signing_key_algorithm |                                                                                     |
| signing_key_curvename |                                                                                      |
| signing_key_signingalgorithm |                                                                                     |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ocsp_signing_key_type</code></td>
<td>Sets the specific settings to generate an OCSP signing key and certificate.</td>
</tr>
<tr>
<td><code>ocsp_signing_key_size</code></td>
<td>The <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> parameters apply to every key and certificate generated by a subsystem. However, each individual key can have its own parameters set separately. The parameters available to <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> apply to the OCSP signing key parameters.</td>
</tr>
<tr>
<td><code>ocsp_signing_key_algorithm</code></td>
<td></td>
</tr>
<tr>
<td><code>ocsp_signing_key_curvename</code></td>
<td></td>
</tr>
<tr>
<td><code>ocsp_signing_key_signingalgorithm</code></td>
<td></td>
</tr>
<tr>
<td><code>audit_signing_key_type</code></td>
<td>For audit signing certificates. For CA, DRM, OCSP, TKS, and TPS. Sets the specific settings to generate an audit log signing key and certificate. The only supported key type for audit certificates is RSA.</td>
</tr>
<tr>
<td><code>audit_signing_key_size</code></td>
<td>The <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> parameters apply to every key and certificate generated by a subsystem. However, each individual key can have its own parameters set separately. The parameters available to <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> apply to the audit log signing key parameters.</td>
</tr>
<tr>
<td><code>audit_signing_key_algorithm</code></td>
<td></td>
</tr>
<tr>
<td><code>audit_signing_key_curvename</code></td>
<td></td>
</tr>
<tr>
<td><code>subsystem_key_type</code></td>
<td>For subsystem client certificates. For all subsystems. Sets the specific settings to generate an SSL client key and certificate.</td>
</tr>
<tr>
<td><code>subsystem_key_size</code></td>
<td>The <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> parameters apply to every key and certificate generated by a subsystem. However, each individual key can have its own parameters set separately. The parameters available to <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> apply to the SSL client key parameters.</td>
</tr>
<tr>
<td><code>subsystem_key_algorithm</code></td>
<td></td>
</tr>
<tr>
<td><code>subsystem_key_curvename</code></td>
<td></td>
</tr>
<tr>
<td><code>sslserver_key_type</code></td>
<td>For server certificates. For all subsystems. Sets the specific settings to generate an SSL server key and certificate.</td>
</tr>
<tr>
<td><code>sslserver_key_size</code></td>
<td>The <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> parameters apply to every key and certificate generated by a subsystem. However, each individual key can have its own parameters set separately. The parameters available to <code>key_type</code>, <code>key_size</code>, <code>key_algorithm</code>, and <code>key_curvename</code> apply to the SSL server key parameters.</td>
</tr>
<tr>
<td><code>sslserver_key_algorithm</code></td>
<td></td>
</tr>
<tr>
<td><code>sslserver_key_curvename</code></td>
<td></td>
</tr>
</tbody>
</table>
**key_algorithm**, and **key_curvename** apply to the SSL server key parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_p12</td>
<td>Sets whether to export the keys and certificate information to a backup PKCS #12 file. <strong>true</strong> backs up the information; <strong>false</strong> does not back up the information. <strong>Only for the CA subsystem.</strong></td>
</tr>
<tr>
<td>backup_pwd</td>
<td>The password to protect the PKCS #12 backup file containing the subsystem keys and certificates. <strong>Not for use with TPS installation.</strong></td>
</tr>
<tr>
<td>backup_fname</td>
<td>The file to which to export the the PKCS #12 backup file.</td>
</tr>
<tr>
<td>ca_subsystem_cert_subject_name</td>
<td>The subject names for the CA subsystem certificates.</td>
</tr>
<tr>
<td>ca_ocsp_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ca_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ca_sign_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ca_audit_signing_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ra_subsystem_cert_subject_name</td>
<td>The subject names and nicknames for the RA subsystem certificates.</td>
</tr>
<tr>
<td>ra_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ra_subsystem_certNickname</td>
<td></td>
</tr>
<tr>
<td>ra_server_certNickname</td>
<td></td>
</tr>
<tr>
<td>ocsp_ocsp_cert_subject_name</td>
<td>The subject names for the OCSP subsystem certificates.</td>
</tr>
<tr>
<td>ocsp_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ocsp_subsystem_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>ocsp_audit_signing_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>drm_storage_cert_subject_name</td>
<td>The subject names for the DRM subsystem certificates.</td>
</tr>
<tr>
<td>drm_transport_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>drm_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>drm_subsystem_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>drm_audit_signing_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>tks_subsystem_cert_subject_name</td>
<td>The subject names for the TKS subsystem certificates.</td>
</tr>
<tr>
<td>tks_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>tks_audit_signing_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>tps_subsystem_cert_subject_name</td>
<td>The subject names and nicknames for the TPS subsystem certificates.</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>tps_server_cert_subject_name</td>
<td></td>
</tr>
<tr>
<td>tps_subsystem_certNickname</td>
<td></td>
</tr>
<tr>
<td>tps_server_certNickname</td>
<td></td>
</tr>
</tbody>
</table>

**Required Subsystem Configuration**

<table>
<thead>
<tr>
<th>ca_hostname</th>
<th>The hostname for the CA subsystem which will issue the certificates for a subordinate CA, RA, DRM, OCSP, TKS, or TPS subsystem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca_port</td>
<td>The non-SSL port number of the CA.</td>
</tr>
<tr>
<td>ca_ssl_port</td>
<td>The SSL end entities port number of the CA.</td>
</tr>
<tr>
<td>drm_hostname</td>
<td>The hostname for the DRM subsystem to use to archive keys. <em>For the TPS only.</em></td>
</tr>
<tr>
<td>drm_ssl_port</td>
<td>The SSL agent port number of the DRM. <em>For the TPS only.</em></td>
</tr>
<tr>
<td>tks_hostname</td>
<td>The hostname for the TKS subsystem to use to derive keys. <em>For the TPS only.</em></td>
</tr>
<tr>
<td>tks_ssl_port</td>
<td>The SSL agent port number of the TKS. <em>For the TPS only.</em></td>
</tr>
</tbody>
</table>

**Authentication Database Configuration** *(TPS only)*

| ldap_auth_host       | Gives the hostname of the LDAP directory database to use for the TPS subsystem token database. *Only for the TPS subsystem.* | |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------| |
| ldap_auth_port       | Gives the port number of the LDAP directory database to use for the TPS subsystem token database. *Only for the TPS subsystem.* | |
| ldap_auth_base_dn    | Gives the base DN in the LDAP directory tree of the TPS token database under which to create token entries. *Only for the TPS subsystem.* | |

**External CA for Issuing Certificates**

<table>
<thead>
<tr>
<th>external</th>
<th>Sets whether to submit the subsystem certificates to the configured CA or to an external CA. The options are true or false. If this is not set, then the default is false.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_csr_file</td>
<td>The output file to which to write the generated certificate requests for the subsystem certificates. <em>Step one of the silent configuration process.</em></td>
</tr>
<tr>
<td>ext_ca_cert_file</td>
<td>The input file for the certificates issued by the external CA. <em>Step two of the silent configuration process.</em></td>
</tr>
<tr>
<td>ext_ca_cert_chain_file</td>
<td>The input file for the CA certificate chain for the external CA issuing the certificate. <em>Step two of the silent configuration process.</em></td>
</tr>
</tbody>
</table>

**Cloning Configuration**

| clone                | Sets whether the new instance is a clone. Its                                                                                                                                       |
11.2. Silently Configuring Subsystems

NOTE

Before running **pkisilent**, first run **pkicreate** to create the instance.

There are slight differences in the options used to configure the different subsystem types:

- Different security domain settings. A CA can host a security domain, so it has special configuration options to create a security domain. All other subsystems (as well as CAs) must join an existing security domain.

  **TIP**

  It is recommended that every CA have its own security domain, because each system within the security domain depends on having the security domain running and accessible. However, subordinate CAs can only be configured within the root CA’s security domain using the **pkisilent** script.

- Different numbers and types of SSL ports. The CA, DRM, OCSP, and TKS each have three SSL ports (admin, agents, and users), while the RA and TPS both have two SSL ports (client and non-client).

- Different numbers and types of certificates.

- Different required subsystems. Every subsystem must, at a minimum, specify which CA will sign and issue its certificates, while a CA has the option of self-signing its certificates. The TPS also relies on a TKS and optional DRM, which can also be specified at configuration.

- Different database configuration. The RA uses a SQLite database as its internal databases, while all other subsystems use an LDAP directory. The TPS uses two separate LDAP directories, one as its internal database and the other as an authentication directory to help manage its users.

For all of that, the usage of **pkisilent** is still pretty similar between the subsystems. They use the same options to identify the instance to configure, back up their keys, and configure their users, and even though the parameters are slightly different in name, the configuration concepts (like cloning or generating certificates) are the same.
NOTE

Any spaces in the arguments used with `pkisilent` must be escaped.

Example 11.2, “Configuring a Root CA” configures a CA, creates a new security domain, backs up its keys, and self-signs its certificates. All of the parameters should be on a single line.

**Example 11.2. Configuring a Root CA**

```bash
pkisilent ConfigureCA -cs_hostname localhost
   -cs_port 9445
   -subsystem_name "pki-ca2"
   -client_certdb_dir /tmp/
   -client_certdb_pwd password
   -preop_pin sY8er834FG9793fsef7et5
   -domain_name "testca"
   -admin_user admin
   -admin_email "admin@example.com"
   -admin_password secret
   -agent_name "jsmith"
   -agent_key_size 2048
   -agent_key_type rsa
   -agent_cert_subject "cn=ca\ agent\ cert"
   -ldap_host server
   -ldap_port 389
   -secure_conn false
   -remove_data true
   -bind_dn "cn=directory\ manager"
   -bind_password secret
   -base_dn "o=pki-ca2"
   -db_name "server.example.com-pki-ca2"
   -key_size 2048
   -key_type rsa
   -key_algorithm SHA256withRSA
   -backup_pwd password
   -backup_fname /export/backup.p12
   -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca"
   -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
   -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
   -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
   -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca"
   -token_name "internal"
```

A subordinate CA — along with the DRM, OCSP, and TKS — is configured to join an existing security domain and to have its certificates signed by an existing Certificate System CA (by default; it is also possible to use an external CA, as in Section 11.5, “Performing Silent Configuration Using an External CA”). All of the parameters should be on a single line.
Example 11.3. Configuring a Subordinate CA

```
pkisilent ConfigureCA -cs_hostname localhost
  -cs_port 9445
  -subsystem_name "pki-ca2"
  -client_certdb_dir /tmp/
  -client_certdb_pwd password
  -preop_pin sYY8er834FG9793fsef7et5
  -sd_hostname "domain.example.com"
  -sd_admin_port 9445
  -sd_agent_port 9443
  -sd_ssl_port 9444
  -sd_admin_name admin
  -sd_admin_password secret
  -admin_user admin
  -admin_email "admin@example.com"
  -admin_password secret
  -agent_name "jismith"
  -agent_key_size 2048
  -agent_key_type rsa
  -agent_cert_subject "cn=ca\ agent\ cert"
  -ldap_host server
  -ldap_port 389
  -secure_conn false
  -remove_data true
  -bind_dn "cn=directory\ manager"
  -bind_password secret
  -base_dn "o=pki-ca2"
  -db_name "server.example.com-pki-ca2"
  -key_size 2048
  -key_type rsa
  -save_p12 true
  -backup_pwd password
  -backup_fname /export/backup.p12
  -ca_hostname server.example.com
  -ca_port 9180
  -ca_ssl_port 9443
  -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\ domain"
  -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
  -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
  -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
  -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\ domain"
  -token_name "internal"
```

A DRM, TKS, and OCSP subsystem is largely the same as a subordinate CA, but without the `-save_p12` option. All of the parameters should be on a single line.
Example 11.4. Configuring a DRM

```
pkisilent ConfigureDRM -cs_hostname localhost
    -cs_port 9445
    -subsystem_name "pki-kra"
    -client_certdb_dir /tmp/
    -client_certdb_pwd password
    -preop_pin sYY8er834FG9793fsef7et5
    -domain_name "example\ domain"
    -sd_hostname "domain.example.com"
    -sd_admin_port 9445
    -sd_agent_port 9443
    -sd_ssl_port 9444
    -sd_admin_name admin
    -sd_admin_password secret
    -admin_user admin
    -admin_email "admin@example.com"
    -admin_password secret
    -agent_key_size 2048
    -agent_key_type rsa
    -agent_cert_subject "cn=drm\ agent\ cert"
    -agent_name "jsmith"
    -ldap_host server
    -ldap_port 389
    -secure_conn false
    -remove_data true
    -bind_dn "cn=directory\ manager"
    -bind_password secret
    -base_dn "o=pki-kra"
    -db_name "server.example.com-pki-kra"
    -key_size 2048
    -key_type rsa
    -backup_pwd password
    -ca_hostname server.example.com
    -ca_port 9180
    -ca_ssl_port 9443
    -drm_subsystem_cert_subject_name "cn=drm\ subsystem\ cert,o=example\ domain"
    -drm_transport_cert_subject_name "cn=drm\ transport\ cert,o=example\ domain"
    -drm_server_cert_subject_name "cn=drm\ client\ cert,o=example\ domain"
    -drm_storage_cert_subject_name "cn=drm\ storage\ cert,o=example\ domain"
    -drm_audit_signing_cert_subject_name "cn=drm\ audit\ signing\ cert,o=example\ domain"
    -token_name "internal"
```

The RA, unlike the other subsystems, does not use an LDAP database, so it does not specify the same database parameters as the other subsystems. In this example, the keys for the RA are not automatically backed up and there is no audit log signing certificate, since the RA is the only subsystem which does not support signed audit logs.
Example 11.5. Configuring an RA

```
pkisilent ConfigureRA -cs_hostname localhost
    -cs_port 9445
    -subsystem_name "pki-ra2"
    -client_certdb_dir /tmp/
    -client_certdb_pwd password
    -preop_pin sYY8er834FG9793fsef7et5
    -domain_name "example\ domain"
    -sd_hostname "domain.example.com"
    -sd_admin_port 9445
    -sd_agent_port 9443
    -sd_ssl_port 9444
    -sd_admin_name admin
    -sd_admin_password secret
    -admin_user admin
    -admin_email "admin@example.com"
    -admin_password secret
    -agent_name "jsmith"
    -agent_key_size 2048
    -agent_key_type rsa
    -agent_cert_subject "cn=ra\ agent\ cert"
    -ca_hostname server.example.com
    -ca_port 9180
    -ca_ssl_port 9443
    -key_size 2048
    -key_type rsa
    -ra_subsystem_cert_subject_name "cn=ra subsystem\ cert,o=testca\ domain"
    -ra_server_cert_subject_name "cn=ra client\ cert,o=testca\ domain"
    -token_name ="internal"
```

A TPS requires the most parameters, since it depends on having a CA, DRM, and TKS configured and uses two LDAP databases, along with joining an existing security domain. However, since the TPS cannot be cloned, it is not required to back up its keys to a PKCS #12 file.
Example 11.6. Configuring a TPS

```bash
pkisilent ConfigureTPS -cs_hostname localhost
   -cs_port 9445
   -subsystem_name "pki-tps2"
   -client_certdb_dir /tmp/
   -client_certdb_pwd password
   -preop_pin sYY8er834FG9793fsef7et5
   -domain_name "example\ domain"
   -sd_hostname "domain.example.com"
   -sd_admin_port 9445
   -sd_agent_port 9443
   -sd_ssl_port 9444
   -sd_admin_name admin
   -sd_admin_password secret
   -admin_user admin
   -admin_email "admin@example.com"
   -admin_password secret
   -agent_name "jsmith"
   -agent_key_size 2048
   -agent_key_type rsa
   -agent_cert_subject "cn=tps\ agent\ cert"
   -ldap_host server
   -ldap_port 389
   -secure_conn false
   -remove_data true
   -bind_dn "cn=directory\ manager"
   -bind_password secret
   -base_dn "o=pki-tps2"
   -db_name "server.example.com-pki-tps2"
   -ca_hostname server.example.com
   -ca_port 9180
   -ca_ssl_port 9443
   -tks_hostname server.example.com
   -tks_ssl_port 13443
   -drm_hostname server.example.com
   -drm_ssl_port 10443
   -key_size 2048
   -key_type rsa
   -tps_subsystem_cert_subject_name "cn=tps\ subsystem\ cert,o=testca\ domain"
   -tps_server_cert_subject_name "cn=tps\ client\ cert,o=testca\ domain"
   -tps_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\ domain"
   -ldap_auth_host auth.example.com
   -ldap_auth_port 389
   -ldap_auth_base_dn "ou=tsp,ou=People,dc=example,dc=com"
   -token_name "internal"
```

11.3. Using Different Key Settings

Generally, the key settings are applied to all keys generated for a subsystem.

```bash
... -key_size 2048 -key_type rsa -key_algorithm SHA256withRSA ...
```
However, each individual key can have its own parameters set separately, meaning each key could be of a different type, of a different size, or use different algorithms or curves. As with the certificate subject names, the types of keys that are configured differ depending on the subsystem.

Every key can be given a unique setting, or only the specified keys can be given unique settings, while all other keys use the default.

For example, this sets different settings for every key for a CA:

```
pkisilent ConfigureCA -cs_hostname localhost
    -cs_port 9445
    -subsystem_name "pki-ca2"
    -client_certdb_dir /tmp/
    -client_certdb_pwd password
    -preop_pin sYY8er834FG9793fsef7et5
    -domain_name "testca"
    -signing_key_type ec
    -signing_key_size 256
    -signing_key_curve nist256
    -signing_key_signingalgorithm SHA256withEC
    -ocsp_signing_key_type ec
    -ocsp_signing_key_size 256
    -ocsp_signing_key_curve nist256
    -ocsp_signing_key_signingalgorithm SHA256withEC
    -audit_signing_key_type rsa
    -audit_signing_key_size 2048
    -audit_signing_key_algorithm SHA256withRSA
    -subsystem_key_type rsa
    -subsystem_key_size 2048
    -subsystem_key_algorithm SHA512withRSA
    -sslserver_key_type rsa
    -sslserver_key_size 2048
    -sslserver_key_algorithm SHA256withRSA
```

This only sets values for the CA signing key and uses the defaults for the other keys:

```
pkisilent ConfigureCA -cs_hostname localhost
    -cs_port 9445
    -subsystem_name "pki-ca2"
    -signing_key_type ec
    -signing_key_size 256
    -signing_key_curve nist256
    -signing_key_signingalgorithm SHA256withEC
    -key_size 2048
    -key_type rsa
    -key_algorithm SHA256withRSA
```

### 11.4. Cloning a Subsystem Silently

**IMPORTANT**

Only CA and DRM instances can be cloned using `pkisilent`. The other subsystem clones must be configured using the HTML-based configuration wizard.
When creating a new subsystem, there are options to set the type of keys to generate and to back up the keys to a PKCS #12 file. For cloning a subsystem, there are no key generation options. Instead, the parameters contain information pointing to the PKCS #12 file for the master subsystem and the URL for the subsystem to clone:

- `clone` true (which sets that the new instance will be a clone)
- `clone_p12_file` and `clone_p12_password`, which gives the name of the master's PKCS #12 key file in the clone's `/var/lib/instance_name/alias` directory and the password to access it
- `clone_start_tls`, which sets whether to use Start TLS for replication between clones

Additionally, a clone must have some configuration in common with its master:

- The same security domain, set in the `-sd_*` parameters
- The same LDAP base DN and database name, set in the `-ldap_*` parameters (either the hostname or the port must be different, since the clone does require a separate Directory Server instance)
- The same issuing CA for its certificates, set in either the `-ca_*` parameters or possibly self-signed, for a CA

Aside from the differences in creating the subsystem certificates, the configuration for the clone (joining the security domain, creating the admin user, setting up the internal LDAP directories) is the same as with any other subsystem configuration.

For example, this clones a CA instance (all parameters should be on one line):
11.5. Performing Silent Configuration Using an External CA

As described in Section 12.1, “Requesting Subsystem Certificates from an External CA”, a CA outside of the security domain can be used to generate a subsystem's certificates. It is also possible to request and submit certificates issued by an external CA using `pkisilent`.

By default, the `pkisilent` command assumes that you will request a certificate from a CA within the security domain, and this CA is identified in the `-ca_hostname` and other `ca_` options. This assumes that the `-external` option is false.

To submit the subsystem certificate requests to an external CA, explicitly set the `-external` option to true. The generated certificate requests are exported to a file, and then can be submitted to the external
CA. Once they are issued, files which contain the subsystem certificates and the CA certificate chain for the issuing external CA can be passed using the `pkisilent` command. This is set in four parameters:

- `-external`, which explicitly sets whether to use an external CA
- `-ext_csr_file`, which gives the path and name of the output file to which to write the certificate requests for the subsystem
- `-ext_ca_cert_file`, which gives the input file to use which contains the certificates issued by the external CA
- `-ext_ca_cert_file`, which gives the input file to use which contains the CA certificate chain for the external CA which issued the certificates

Whether it is performed through the HTML wizard or using `pkisilent`, submitting certificates to an external CA is a three-step process, two of them involving `pkisilent`:

1. In the first step, much of the preliminary information is configured for the instance.
   Along with the subsystem configuration settings, the subsystem's certificate requests are generated and written to the file specified in `-ext_csr_file`. These certificate requests must be submitted to the external CA.

For example (in real life, these options should be on a single line):

```
pkisilent ConfigureCA
  -cs_hostname server.example.com
  -cs_port 9445
  -subsystem_name "pki-ca2"
  -client_certdb_dir /tmp/
  -client_certdb_pwd password
  -preop_pin sYY8er834FG9793fsef7et5
  -domain_name "testca"
  -agent_name jsmith
  -agent_key_size 2048
  -agent_key_type rsa
  -agent_cert_subject "cn=ca\ agent\ cert"
  -ldap_host ldapserver.example.com
  -ldap_port 389
  -secure_conn false
  -remove_data true
  -bind_dn "cn=directory\ manager"
  -bind_password password -base_dn "o=pki-ca2"
  -db_name "server.example.com-pki-ca2"
  -key_size 2048
  -key_type rsa
  -key_algorithm SHA512withRSA
  -token_name internal
  -token_pwd 242986083911
  -save_p12 true
  -backup_pwd password
  -backup_fname /export/backup.p12
  -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\ domain"
  -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
  -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
  -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
  -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\ domain"
  -external true
  -ext_csr_file /tmp/cert.req
```
2. The certificate requests are submitted to the external CA, and the issued certificates are retrieved and saved to file.

3. The newly issued subsystem certificates are installed in the instance by referencing the saved certificate file in the `ext_cert_file` parameter and the issuing CA's certificate chain in the `ext_cert_chain_file` parameter.

   For example (in real life, these options should be on a single line):

   ```
   pkisilent ConfigureCA
   -cs_hostname server.example.com
   -cs_port 9445
   -subsystem_name "pki-ca2"
   -client_certdb_dir /tmp/
   -client_certdb_pwd password
   -preop_pin sYY8er834FG9793fsef7et5
   -domain_name "testca"
   -admin_user admin
   -admin_password_secret
   -admin_email "admin@example.com"
   -agent_name jsmith
   -agent_key_size 2048
   -agent_key_type rsa
   -agent_cert_subject "cn=ca\ agent\ cert"
   -ldap_host ldapserver.example.com
   -ldap_port 389
   -secure_conn false
   -remove_data true
   -bind_dn "cn=directory\ manager"
   -bind_password password
   -base_dn "o=pki-ca2"
   -db_name "server.example.com-pki-ca2"
   -key_size 2048
   -key_type rsa
   -key_algorithm SHA512withRSA
   -token_name internal
   -token_pwd 242986083911
   -save_p12 true
   -backup_pwd password
   -backup_fname /export/backup.p12
   -ca_subsystem_cert_subject_name "cn=ca\ subsystem\ cert,o=testca\ domain"
   -ca_ocsp_cert_subject_name "cn=ocsp\ signing\ cert,o=testca\ domain"
   -ca_server_cert_subject_name "cn=ca\ client\ cert,o=testca\ domain"
   -ca_sign_cert_subject_name "cn=ca\ signing\ cert,o=testca\ domain"
   -ca_audit_signing_cert_subject_name "cn=audit\ signing\ cert,o=testca\ domain"
   -external true
   -ext_cert_file /tmp/cert.cer
   -ext_cert_chain_file /tmp/cachain.cer
   ```

   This is also when the final configuration to create the administrator user is performed.

   **NOTE**

   All of the previous parameters must be included the second time that `pkisilent` is invoked.
Chapter 12. Additional Installation Options

All Red Hat Certificate System instances created with **pkicreate** make certain assumptions about the instances being installed, such as the default signing algorithm to use for CA signing certificates and whether to allow IPv6 addresses for hosts.

This chapter describes additional configuration options that impact the installation and configuration for new instances, so many of these procedures occur before the instance is created.

### 12.1. Requesting Subsystem Certificates from an External CA

Most of the time, it is easier and simpler to have a CA within your PKI be the root CA, since this affords a lot of flexibility for defining the rules and settings of the PKI deployment. However, for public-facing networks, this can be a difficult thing to implement, because web administrators have to find some way to propagate and update CA certificate chains to their clients so that any site is trusted. For this reason, people often use public CAs, hosted by companies like VeriSign or Thawte, to issue CA signing certificates and make all of their CAs subordinate to the public CA. This is one of the planning considerations covered in the *Certificate System Deployment Guide*.

All subsystem certificates can be submitted to an external CA when the subsystem is configured. When the certificates are generated from a CA outside the Certificate System deployment (or from a Certificate System CA in a different security domain), then the configuration process does not occur in one sitting. The configuration process is on hold until the certificates can be retrieved. Aside from that delay, the process is more or less the same as in Chapter 6, *Installing and Configuring Certificate System*.

1. Install the subsystem packages, and open the configuration URL.
2. Join a security domain. For a CA, it is also possible to create a new security domain.
3. Set the subsystem information, like its name.
4. For a CA, set the CA to be a subordinate CA. This is required in order to submit CA subsystem certificates to an external CA; making a root CA automatically generates self-signed certificates.
5. Set up the internal database.
6. Select the key store, and generate the key pairs.
7. Set the subsystem certificate names; you can set these to whatever you want, but make sure that they conform to any requirements that the external CA sets for the subject names of certificates.
   At the bottom of the screen is the list of CAs which are available to accept the submitted certificate requests. Choose **External CA** from the drop-down menu.

   ![External CA dropdown menu](image)

8. In the **Requests and Certificates** panel, the CA signing certificate is listed, with an **action required** label. Once that certificate is generated, the other certificates for the CA will be automatically generated.
   For other subsystems, each subsystem certificate has an **action required** label and must be submitted to the external CA.
9. Click the Step 1: Copy the certificate request link, and copy the certificate request to a file or the clipboard.

```
-----BEGIN CERTIFICATE REQUEST-----
MIIeCjCCoCAQAwH/ETMGEAUEChNHKzVjIERkWVpbjEeHe8gAUEAxWVG2Yd3tMaWNwGUGg
QXV0aG9yakR5clB1JANBkghk1go5w90BAeQFAA0CCQSM1IEBCgKCMQAnSpOgyxuU6Y70E/6dEml
CDeRaFxQ1existentCA23ExETY/Z4nCT7RoCXMW/sMFvJDFkC190jUxOg
CN=Red Hat Certificate System 8.1 Deployment, Planning, and Installation
N=Red Hat Certificate Authority 0=New Domain
-----END CERTIFICATE REQUEST-----
```

10. Submit the request to the external CA. Leave the browser with the configuration wizard open as
you wait for the certificates to be generated, so that it is easier to pick up the process.

11. Click **Step 2**, and import the certificate chain for the issuing CA. This certificate chain must **not** contain the leaf certificate (the certificate being requested).

12. After retrieving the issued certificates, click the **Step 3: Paste in the base 64-encoded certificate** link, and paste in the new certificate (this should be only the new certificate, not a certificate chain).

![Certificate System](image)

Copy the resulting base64-encoded certificate (NOTE: PKCS #7 not accepted) into the text box below and press 'X':

...paste certificate here...

13. For the CA, this only has to be done for the signing certificate. For the other subsystems, repeat steps 9, 10, and 12 for each certificate.

When all the certificates have been submitted, click **Next** to resume the configuration process.

14. Export the keys for the certificates, if the subsystem will be cloned.

15. Create the administrator user.

16. When the configuration is done, restart the subsystem.

```
service instance_name restart
```

### 12.2. Installing with Shared Port Assignments

**WARNING**

Using shared SSL ports is deprecated is and not recommended.

The recommended and most secure configuration for subsystems is to use **port separation** for secure connections, so that each type of secure connection (end entities, agents, and administrators) uses its own SSL port. Older versions of Certificate System used a single port for all SSL connections, and it is still possible to create an instance with a single SSL port.

An instance must be created with either a single SSL port or all three separate SSL ports. It is not possible to share a port for some interfaces and then have others separate when running `pkicreate`, though this can be configured manually after the instance is configured.

To create an instance with three separate ports for the different subsystem services, run `pkicreate` with three options which specify the services ports: `-admin_secure_port`, `-agent_secure_port`, and `-ee_secure_port`. For CAs only, there is an additional port for end-entity client authentication, `-ee_secure_client_auth_port`.

1. Run the `pkicreate` command, specifying the type of subsystem being created, the configuration directory, instance name, and port numbers. For example, this created a second DRM instance:
2. When the instance is successfully created, the process returns a URL for the HTML configuration page. For example:

http://server.example.com:10180/kra/admin/console/config/login?pin=nt2z2keqcqAZiBRBGLf

**TIP**

The configuration URL is written to the end of the instance's installation file, 
/var/log/pki-name/logs-install.log. This log is also useful for debugging an instance.

3. Open the new instance URL, and go through the configuration wizard as described in Chapter 6, *Installing and Configuring Certificate System*. Supply the security domain, CA, instance ID, internal LDAP database, and agent information.

4. When the configuration is complete, restart the subsystem.

```
service instance_ID restart
```

For more information on the pkicreate tool options, see the *Certificate System Command-Line Tools Guide*.

### 12.3. Enabling IPv6 for a Subsystem

Certificate System automatically configures and manages connections between subsystems. Every subsystem must interact with a CA as members of a security domain and to perform their PKI operations.

For these connections, Certificate System subsystems can be recognized by their host's fully-qualified domain name or an IP address. By default, Certificate System resolves IPv4 addresses and hostnames automatically, but Certificate System can also use IPv6 for their connections. IPv6 is supported for all server connections: to other subsystems, to the administrative console (pkiconsole), or through command-line scripts such as tpsclient:

```
op=var_set name=ca_host value=IPv6 address
```

1. Install the Red Hat Certificate System packages.
2. Set the IPv4 and IPv6 addresses in the /etc/hosts file. For example:

```
vim /etc/hosts

192.0.0.0 server.example.com IPv4 address
```
3. Then, export the environment variable to use the IPv6 address for the server. For example:
4. Run `pkicreate` to create the new instance. The values for the server hostname in the `CS.cfg` file will be set to the IPv6 address.

### 12.4. Configuring Separate RA Instances

When an RA is installed or created, it is automatically added to a default Registration Managers Group on the CA. This means that all RA managers belong to the same group, by default.

However, a particular site might require more than one RA instance, each having its own set of RA agents. If the site policy disallows cross-management between the RA instances, then extra configuration is needed to create separate RA groups.

1. Install and configure the first RA instance.
2. Add the new RA group to the Certificate Manager.
   a. Start the Console. For example:

   ```bash
   pkiconsole https://server.example.com:9445/ca
   ```
   
   b. Click **Users and Groups**, and then click **Groups**.
   
   c. Click **Add** to open the **Edit Group Information** dialog box.
   
   d. Enter the group name and description, such as **Registration Manager2 Agents**.
   
   e. Click **OK**.

3. Add the new RA authentication instance to the CA:
   a. Open the CA configuration directory, and edit the `CS.cfg` file

   ```bash
   cd /etc/pki-ca
   vi CS.cfg
   ```

   b. Search for the string `raCertAuth`.
   
   c. Copy those lines for the first RA instance, paste them, and edit them for the second RA instance's information. For example:

   ```
   auths.instance.raCertAuth.agentGroup=Registration Manager Agents
   auths.instance.raCertAuth.plug-inName=AgentCertAuth
   auths.instance.ra2CertAuth.agentGroup=Registration Manager2 Agents
   auths.instance.ra2CertAuth.plug-inName=AgentCertAuth
   ```

   4. Add the new RA user enrollment profile to the Certificate Manager's certificate profiles list to utilize the new RA authentication instance.

   a. Open the CA profiles directory.

   ```bash
   cd /var/lib/pki-ca/profiles/ca
   ```

   b. Copy the current RA profile to create the new profile. For example:

   ```bash
   cp caDualRAuserCert.cfg caDualRA2userCert.cfg
   ```
c. Edit the new file to contain the second RA instance's information. Change `raCertAuth` to `ra2CertAuth`.

5. Open the CA configuration directory, and edit the `CS.cfg` file.

```
cd /var/lib/pki-ca/conf
vi CS.cfg
```

a. Add `caDualRA2userCert` to the profiles list. For example:

```
profile.list=...[snip]...caRAserverCert,caRA2userCert
```

Make sure to use a comma to separate the entries.

b. Search for the lines for the `caDualRAuserCert` profile configuration, copy them, and edit them for the second RA instance's information.

```
profile.caDualRAuserCert.class_id=caEnrollImpl
profile.caDualRAuserCert.config=/var/lib/pki-ca/profiles/ca/caDualRAuserCert.cfg
profile.caDualRA2userCert.class_id=caEnrollImpl
profile.caDualRA2userCert.config=/var/lib/pki-ca/profiles/ca/caDualRA2userCert.cfg
```

6. Add a new URI mapping to allow the new RA agent to be registered in the new RA group.

a. Open the CA web applications directory, and edit the `web.xml` file:

```
cd /var/lib/pki-ca/webapps/ca/WEB-INF
vi web.xml
```

b. At about line 288 in the `web.xml` file is the `servlet` setting for the first RA's user. Copy the entire entry, including the opening and closing `<servlet>` tags, and edit the information to match the second RA's user. For example:
<servlet>
    <servlet-name> caRegisterRa2User </servlet-name>
    <servlet-class> com.netscape.cms.servlet.csadmin.RegisterUser </servlet-class>
    <init-param>
        <param-name> GetClientCert </param-name>
        <param-value> false </param-value>
    </init-param>
    <init-param>
        <param-name> authority </param-name>
        <param-value> ca </param-value>
    </init-param>
    <init-param>
        <param-name> ID </param-name>
        <param-value> caRegisterRaUser </param-value>
    </init-param>
    <init-param>
        <param-name> AuthMgr </param-name>
        <param-value> TokenAuth </param-value>
    </init-param>
    <init-param>
        <param-name> GroupName </param-name>
        <param-value> Registration Manager2 Agents </param-value>
    </init-param>
    <init-param>
        <param-name> AuthzMgr </param-name>
        <param-value> BasicAclAuthz </param-value>
    </init-param>
    <init-param>
        <param-name> resourceID </param-name>
        <param-value> certServer.ca.registerUser </param-value>
    </init-param>
</servlet>

c. At about line 2510 in the web.xml file is the servlet-mapping setting for the first RA's user mapping. Copy the entire entry, including the opening and closing <servlet-mapping> tags, and edit the information to match the second RA's user. For example:

    <servlet-mapping>
        <servlet-name> caRegisterRa2User </servlet-name>
        <url-pattern> /admin/ca/registerRa2User </url-pattern>
    </servlet-mapping>

7. Restart the CA. For example:

    service pki-ca restartt

8. Create the new RA instance using the pkicreate.

    pkicreate -pki_instance_root=/var/lib -subsystem_type=ra -pki_instance_name=pki-ra2 -secure_port=12899 -unsecure_port=12898 -verbose -user=pkiuser -group=pkiuser

9. Open the configuration file for the new RA instance, and edit its parameters to reflect the second RA instance information.

    cd /var/lib/pki-ra2/conf/
    vi CS.cfg

10. Change the registerRaUser setting to registerRa2User.

    conn.ca1.servlet.addagent=/ca/admin/ca/registerRa2User
11. Change the `caDualRAuserCert` setting to `caDualRA2userCert`.

   ```
   request.renewal.approve_request.0.profileId=caDualRAuser2Cert
   ...
   request.user.approve_request.0.profileId=caDualRA2userCert
   ```

12. Restart the new RA instance. For example:

   ```
   # service pki-ra2 restart
   ```

13. A URL was generated at the end of the `pkicreate` command; go to that URL to configure the second RA. For example:

   ```
   http://server.example.com:12898/ra/admin/console/config/login?
   pin=bFyAk9nWPfgLZXffRBT9
   ```

14. When the new RA is completely configured, restart the instance.

   ```
   # service pki-ra2 restart
   ```
Chapter 13. Updating and Removing Subsystem Packages

Certificate System is installed using individual packages for each of its subsystems and its supporting systems, like Red Hat Directory Server and NSS. This makes the Certificate System modular, and each individual subsystem and its packages can be installed, updated, and removed independently.

13.1. Updating Certificate System Packages

There are many packages, listed in Section 5.2, "Packages Installed on Red Hat Enterprise Linux", installed with Certificate System for related applications and dependencies, not just the subsystem packages. For all supported platforms, individual Certificate System packages may be updated through the native package utility, `yum`.

NOTE

All Certificate System instances must be stopped before beginning any updates.

The recommended way to update packages is to use `yum` to manage the updates.

1. Stop all Certificate System instances.

   ```
   service instance_ID stop
   ```

2. Log in as `root`.

3. Run `yum` for the package. For example:

   ```
   yum update pki-java-tools-8.1.0-4.noarch
   ```

   Or simply:

   ```
   yum update pki-java-tools
   ```

4. Restart the Certificate System instances.

   ```
   service instance_ID start
   ```

Alternatively, any updated RPMs can be downloaded and installed manually.

1. Stop all Certificate System instances.

   ```
   service instance_ID stop
   ```

2. Log in as `root`.

3. Install the updated package.

   ```
   rpm -Uvh package_name
   ```

   For example:

   ```
   rpm -Uvh pki-java-tools-8.1.0-4.noarch.rpm
   ```
4. Restart the Certificate System instances.

```
service instance_ID start
```

### 13.2. Uninstalling Certificate System Subsystems

It is possible to remove individual subsystem instances or to uninstall all packages associated with an entire subsystem. Instances and subsystems are installed and uninstalled individually. For example, it is possible to uninstall a DRM subsystem while leaving an installed and configured CA subsystem. It is also possible to remove a single CA instance while leaving other CA instances on the machine.

#### 13.2.1. Removing a Subsystem Instance

Removing an instance requires specifying the instance directory and the instance name. This command removes all files associated with the instance (without removing the subsystem packages).

```
pkiremove -pki_instance_root=instance_root -pki_instance_name=instance_ID -token_pwd=password -force
```

The `instance_root` is the directory path of the instance, such as `/var/lib/instance_name`. The `instance_name` is the instance name, such as `pki-ca`. The `password` is the password used to access the NSS database for the instance being removed. If the password isn't given with the command, then the script assumes that it is in the `password.conf` file; otherwise, the script prompts for the password.

**TIP**

Use `-force` with `pkiremove` to remove the instance without prompting for confirmation.

#### Example 13.1. Removing a CA Instance

```
pkiremove -pki_instance_root=/var/lib -pki_instance_name=pki-ca1 -force -token_pwd=secret
```

PKI instance Deletion Utility ...

PKI instance Deletion Utility cleaning up instance ...

Stopping pki-ca1:
process already stopped

Removing dir /var/lib/pki-ca1
Removing file /var/log/pki-ca1-install.log
Removing file /etc/init.d/pki-ca1
Removing file /usr/share/applications/pki-ca1-config.desktop
Removing file /usr/bin/dtomcat5-pki-ca1

`pkiremove` removes the instance and any related files, such as the certificate databases, certificates, keys, and associated users. It does not uninstall the subsystem packages.

#### 13.2.2. 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 5.2, “Packages Installed on Red Hat Enterprise Linux”. Removing a subsystem instance removes only the files and directories associated with that specific instance. It does not remove the actual installed packages that 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 subsystem instances using `pkiremove`. For example:

   ```bash
   pkiremove -pki_instance_root=/var/lib -pki_instance_name=pki-ca -force -
   token_pwd=secret
   ```

2. Run the uninstall utility. For example:

   ```bash
   yum remove pki-*subsystem_type*
   ```

   The subsystem type can be ca, ra, drm, 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 5.2, “Packages Installed on Red Hat Enterprise Linux”.

Chapter 13. Updating and Removing Subsystem Packages
Chapter 14. Troubleshooting Installation, Cloning, and Upgrade

This chapter covers some of the more common installation and migration issues that are encountered when installing Certificate System.

**IMPORTANT**

For instance creation errors (related to running `pkicreate`), try the `/var/log/pki-name/logs/instance-install.log` file. If you have configuration errors (meaning, setting up the subsystem after running `pkicreate`), check the `catalina.out` and `debug` files in the instance’s log directory.

14.1. Installation

**Q:** I can't see any Certificate System packages or updates.

**A:** Certificate System packages are delivered through Red Hat Network, so you need to have your `yum` repositories configured to point to Red Hat Network or a local Satellite and then to use and account with the appropriate subscriptions to access the Red Hat Certificate System channels.

**Q:** The init script returned an OK status, but my CA instance does not respond. Why?

**A:** This should not happen. Usually (but not always), this indicates a listener problem with the CA, but it can have many different causes. Check in the `catalina.out`, `system`, and `debug` log files for the instance to see what errors have occurred. This lists a couple of common errors.

One situation is when there is a PID for the CA, indicating the process is running, but that no listeners have been opened for the server. This would return Java invocation class errors in the `catalina.out` file:

```
INFO: Initializing Coyote HTTP/1.1 on http-9080
java.lang.reflect.InvocationTargetException
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:615)
  at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:243)
  at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:408)
Caused by: java.lang.UnsatisfiedLinkError: jss4
```

This could mean that you have the wrong version of JSS or NSS. The process requires `libnss3.so` in the path. Check this with this command:

```
ldd /usr/lib64/libjss4.so
```

If `libnss3.so` is not found, try unsetting the `LD_LIBRARY_PATH` variable and restart the CA.
Q: I am having trouble running through the configuration wizard for a new instance. It's giving me errors about already having the certificate for the subsystem installed.

A: This error occurs when the same browser profile is used to configure multiple instances of the same type of subsystem. If a subsystem (or the entire Certificate System) is removed and reinstalled, then the previous CA certificate and subsystem certificates need to be deleted from the browser profile. Otherwise, when the replacement subsystem is created, the CA certificate has the same subject name and serial number as the previous instance, which creates a conflict in the browser NSS databases.

The easiest solution is to create new browser profiles when reinstalling instances of Certificate System.

Q: I want to set different certificate validity periods and extensions for my root certificate authority — but I don't see a way to set it in the configuration wizard.

A: There isn't currently a way to do this in the configuration wizard. However, there is a way to edit the certificate profiles used by the configuration wizard to generate the root CA certificates.

**IMPORTANT**

This must be done before running `pkicreate` to create a new CA instance.

1. Back up the original CA certificate profile used by the configuration wizard.

   ```bash
   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.

   ```bash
   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:

   ```bash
   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 Basic Constraint Extension, add the default (which, in this example, is default #9):

   ```bash
   9.default.class=com.netscape.cms.profile.def.BasicConstraintsExtDefault
   9.default.name=Basic Constraint Extension Constraint
   9.default.params.basicConstraintsCritical=true
   9.default.params.basicConstraintsIsCA=true
   9.default.params.basicConstraintsPathLen=2
   ```
Then, add the default number to the list of defaults to use the new default:

`list=2,4,5,6,7,8,9`

5. Once the new profile is set up, then run `pkicreate` to create the new CA instance and go through the configuration wizard.

Q: I'm 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 generif error which can have many different causes. Check in the `catalina.out`, `system` and `debug` log files for the instance to see what errors have occurred. This lists a couple of common errors, 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 `catalina.out` 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:

```
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
cal port 389 Error netscape.ldap.LDAPException: failed to connect to
server ldap://ca1.example.com: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 `catalina.out` log file for the instance’s Tomcat service shows a series of connection errors that result in the HTTP 500 error:
Q: I keep getting errors in when I try to configure the LDAP internal database for my instance. It says the database has already been used. Why?

A: The configuration process creates a new suffix and database in the Directory Server. Even an instance that was removed still has its database in the Directory Server so its information is preserved. If a new instance is created with the same name, then either the Directory Server suffix in the configuration wizard needs to be given a new, unique name or you need to select the checkbox to overwrite any existing data in the database.

If the Directory Server database for the subsystem instance isn't given a unique name, then configuration will not move past that point and the debug file will have these errors about the database already being used:
14.2. Java Console

Q: I can't open the pkiconsole and I'm seeing Java exceptions in stdout.

A: This probably means that you have the wrong JRE installed or the wrong JRE set as the default. Run `alternatives --config java` to see what JRE is selected. Red Hat Certificate System requires OpenJDK 1.6.

Q: I tried to run pkiconsole, and I got Socket exceptions in stdout. Why?

A: This means that there is a port problem. Either there are incorrect SSL settings for the administrative port (meaning there is bad configuration in the `server.xml`) or the wrong port was given to access the admin interface.

Port errors will look like the following:
14.3. Cloning

Q: I created a new clone instance, but when I tried to restart it, I got an error that said 
*Configuration Definitions not found.*

A: This is an intermittent error that means it had a problem accessing the new clone information in the security domain. Just restart the instance again, and it will restart successfully.

14.4. Upgrade and Migration

Q: I'm seeing an authentication error when I try to run the migration tools to upgrade my instance.

A: The migration tools are out of date. The new migration tools are included with the new Certificate System packages. To upgrade to 8.1, use the upgrade tools supplied with 8.1.

When trying to upgrade by running *TxtTo73* instead of *TxtTo81*, then the new instance cannot import the previous information, which means it cannot recognize and confirm the old certificates. This results in an authentication error for the wrong admin certificate in the *debug* logs:

```java
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:24)
    at com.netscape.admin.certsrv.connection.JSSConnection.readHeader(JSSConnection.java:439)
    at com.netscape.admin.certsrv.connection.JSSConnection.initReadResponse(JSSConnection.java:430)
    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:623)
    at com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:590)
    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 ran the** Old_version ToTxt **script and then** TxtTo73, **and I'm seeing AUTH_TOKEN errors in my debug log.**

A: The migration tools are out of date. The upgrade script go to the new version of Certificate System, meaning that to upgrade to 8.1, you need to run **TxtTo81.** This script is available with the new Certificate System 8.1 pacakges.

If the wrong script is used to convert the output text file, then you may see errors like this:

```
ERROR AuthToken type -
  AUTH_TOKEN=com.netscape.certsrv.authentication.AuthToken=uid:[Ljava.lang.String;=
  =[Ljava.lang.String;@108ca1
```
Part III. After Installing Red Hat Certificate System

This part has basic information for finding subsystem services pages, configuration files, and related scripts.
Chapter 15. After Configuration: Checklist of Configuration Areas for Deploying Certificate System

The create-and-configure process sets up the subsystem instances so that they are running and fully functional. However, the details of deploying a PKI — like defining what kind of certificates to issue — must still be done.

The features and performance of Certificate System subsystems can be matched to the requirements of your specific deployment. The most common (and recommended) features to configure are listed in Table 15.1, "Certificate System Features to Configure after Setup".
<table>
<thead>
<tr>
<th>Feature</th>
<th>Why It's Useful</th>
<th>Admin Guide Link</th>
<th>Recommended or Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create agents and administrative users</td>
<td>The setup process only creates a single administrative user account. To manage a PKI environment in real life requires a number more administrators and agents, which are created after the instance is set up.</td>
<td>CA, OCSP, DRM, and TKS users, RA users, TPS users</td>
<td>Mandatory (agents only) Recommended (administrators only) Common Criteria required</td>
</tr>
<tr>
<td>Import the certificate chain into the browser.</td>
<td>Any browser which will be used to access agent or admin services pages must have the client certificate and the CA certificate chain imported into its trust store.</td>
<td>Importing certs</td>
<td>Optional Common Criteria required</td>
</tr>
<tr>
<td>Set sudo permissions on PKI services for PKI administrators</td>
<td>Granting sudo permissions for Certificate System and Directory Server services to PKI administrators makes it easier to manage the instances.</td>
<td>Setting sudo permissions</td>
<td>Optional Common Criteria required</td>
</tr>
<tr>
<td>Set up backup and restore procedures</td>
<td>A reliable backup and restore strategy is vital for disaster and data recovery.</td>
<td>Backup and restore</td>
<td>Recommended Common Criteria required</td>
</tr>
<tr>
<td>Enable signed audit logs</td>
<td>This can be done when an instance is first created by using the audit_group option, but it can also be done post-installation.</td>
<td>Audit logs for CA, DRM, OCSP, and TKS, Audit logs for TPS</td>
<td>Recommended Common Criteria required</td>
</tr>
<tr>
<td>Configure LDAP publishing</td>
<td>LDAP publishing publishes the certificates to specific entries within an LDAP database, so other clients can access the entries.</td>
<td>LDAP publishing</td>
<td>Recommended</td>
</tr>
<tr>
<td>Configure client authentication with the backend Directory</td>
<td>Client authentication allows the Certificate System instance to bind</td>
<td>Client authentication</td>
<td>Recommended</td>
</tr>
<tr>
<td>Server</td>
<td>automatically to its backend Directory Server over SSL by presenting a recognized SSL client certificate (similar to the way agents connect to a subsystem instance).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customize certificate profiles</td>
<td>A certificate profile defines everything associated with issuing a particular type of certificate, including the authentication method, the authorization method, the certificate content (defaults), constraints for the values of the content, and the contents of the input and output for the certificate profile. Existing certificate profiles can be edited and new profiles added to allow deployment-specific certificates and tokens to be issued.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enable revocation checking and OCSP responses</td>
<td>There are two methods for checking whether a certificate is active (and not revoked): CRL checking and OCSP services. Checking the revocation status helps ensure that a revoked certificate isn't wrongly accepted just because it is within its validity period.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configure a timeout period for SSL sessions</td>
<td>Using session timeouts prevents unauthorized users from accessing an open, but otherwise inactive, session with a subsystem.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use port forwarding</td>
<td>The default URLs used by Certificate System subsystems can be very long, which makes it possible for users to mistype the URL. Port forwarding allows users</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Recommended

Revocation checking

OCSP responders

Session timeouts

Recommended

Common Criteria required

Optional
<table>
<thead>
<tr>
<th>Set up cross-pair certificates (also called FBCA or bridge certificates).</th>
<th>Cross-pair certificates set up a trusted environment between two separate PKI environments, so that certificates issued in one PKI are automatically recognized and trusted by the other PKI environment.</th>
<th>Cross-pair certificate profiles</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run self-tests</td>
<td>Self-tests are subsystem type-specific diagnostics. These are usually run when the server starts, but they can also be run on demand. Additionally, self-test can be customized by creating new self-test plug-ins or to accommodate changes to the subsystem configuration like new certificates or changed certificate nicknames.</td>
<td>Self-tests</td>
<td>Recommended</td>
</tr>
<tr>
<td></td>
<td>The password.conf file stores system passwords in plaintext.</td>
<td>Running without the password.conf file</td>
<td>Common Criteria required</td>
</tr>
<tr>
<td></td>
<td>Remove the password.conf file.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove unused CA connection interfaces. Several legacy interfaces are included in the CA's web.xml. While these can be used for custom or legacy applications to connect to the CA, for secure environments, these deprecated interfaces should be removed from the CA configuration to prevent unauthorized access.</td>
<td>Remove unused CA connection interfaces</td>
<td>Common Criteria required</td>
</tr>
<tr>
<td></td>
<td>Assign POSIX system ACLs for PKI subsystem users. This provides finer control over the ACLs used for the subsystems' system</td>
<td>Configuring POSIX system ACLs</td>
<td>Common Criteria required</td>
</tr>
</tbody>
</table>

Chapter 15. After Configuration: Checklist of Configuration Areas for Deploying Certificate System
users.
Chapter 16. Basic Information for Using Certificate System

Using any Certificate System has basic tasks such as editing the configuration file, starting and stopping the server instance and Console, opening web services, and locating logs. This is explained in more detail in the Certificate System Administrator's Guide.

16.1. Starting the Certificate System Console

The CA, DRM, OCSP, and TKS subsystems have a Java interface which can be accessed to perform administrative functions. For the DRM, 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 SSL port using the `pkiconsole` command. This command has the format:

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

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

```
pkiconsole https://server.example.com:10445/kra
```

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the console. For example:

```
https://1.2.3.4:9445/ca
https://[00:00:00:00:123:456:789:00]:9445/ca
```

16.2. Starting, Stopping, and Restarting an Instance

The Certificate System subsystem instances can be stopped and started using system tools on Red Hat Enterprise Linux. For example:

```
service instance-name {start|stop|restart}
```

The instance name for subsystem instances is usually has the format `pki-instance-id`, such as `pki-ca`.

16.3. Starting the Subsystem Automatically

Red Hat Enterprise Linux 5.6 has a tool called `chkconfig` which 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. `chkconfig` also defines startup settings for different run levels of the server. `chkconfig` is explained more in the Red Hat Enterprise Linux documentation, such as the Deployment Guide.

Certificate System subsystems can be managed by `chkconfig`, so this tool can set whether to restart subsystems automatically. By default, every Certificate System subsystem instance is turned off at every run level in the system, meaning instances must be started and stopped manually. This can be changed by resetting the configuration in `chkconfig` to `on`. For example, this automatically restarts Red Hat Directory Server, Administration Server, and the CA:
Make sure the subsystem is listed with the other services.

```
chkconfig --list | grep subsystem_name
```

To remove the subsystem from the start list, simply turn the level to off:

```
chkconfig --level 35 subsystem_name off
```

Red Hat Enterprise Linux also has a GUI console that can manage `chkconfig` settings.

![Figure 16.1. chkconfig Settings](image)

The start order of services is extremely important, or the subsystems will not function. The Directory Server and Administration Server instances used by the subsystems must be running before the subsystems can be started, and their web services (Tomcat or Apache) must be running before the subsystems are started or their web services will not function.

The default Certificate System `chkconfig` settings set a start and stop priority for all of the subsystems and their dependent services so that they start and stop in the proper order, as listed in Table 16.1, “Certificate System Processes and Their chkconfig Start Priority”. Processes with a low
number for their start priority are started first, so Directory Server, Administration Server, and Tomcat are started before any of the subsystem instances. Likewise, processes with a low number for their shutdown priority are shut down first, so the subsystem processes are stopped before the processes they depend on are stopped.

Table 16.1. Certificate System Processes and Their chkconfig Start Priority

<table>
<thead>
<tr>
<th>Server</th>
<th>Process Name</th>
<th>Start Priority</th>
<th>Shutdown Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration Server</td>
<td>dirsrv-admin</td>
<td>21</td>
<td>79</td>
</tr>
<tr>
<td>Directory Server</td>
<td>dirsrv</td>
<td>21</td>
<td>79</td>
</tr>
<tr>
<td>Tomcat Server</td>
<td>tomcat5</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>CA</td>
<td>pki-ca</td>
<td>81</td>
<td>19</td>
</tr>
<tr>
<td>DRM</td>
<td>pki-kra</td>
<td>82</td>
<td>18</td>
</tr>
<tr>
<td>OCSP</td>
<td>pki-ocsp</td>
<td>83</td>
<td>17</td>
</tr>
<tr>
<td>TKS</td>
<td>pki-tks</td>
<td>84</td>
<td>16</td>
</tr>
<tr>
<td>Apache</td>
<td>httpd</td>
<td>85</td>
<td>15</td>
</tr>
<tr>
<td>RA</td>
<td>86</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>TPS</td>
<td>pki-tps</td>
<td>87</td>
<td>13</td>
</tr>
</tbody>
</table>

16.4. Finding the Subsystem Web Services Pages

The CA, RA, DRM, OCSP, TKS, and TPS subsystems have web services pages for agents, as well as potentially regular users and administrators. 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:9444/ca/services

TIP

To get a complete list of all of the interfaces, URLs, and ports for a subsystem, check the service's status:

service instance-name status

The main web services page for each subsystem has a list of available services pages; these are summarized in Table 16.2, “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:9444/ca/ee/ca

If DNS is properly configured, then an IPv4 or IPv6 address can be used to connect to the services pages. For example:

https://1.2.3.4:9444/ca/services
https://[00:00:00:00:123:456:789:00]:9444/ca/services
Anyone can access the end user pages for a subsystem, but accessing agent or admin web services pages requires that an agent or administrator certificate be issued and installed in the web browser, or authentication to the web services fails.
Table 16.2. Default Web Services Pages

<table>
<thead>
<tr>
<th>Port</th>
<th>Used for SSL</th>
<th>Used for Client Authentication</th>
<th>Web Services</th>
<th>Web Service Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Certificate Manager</td>
</tr>
<tr>
<td>9180</td>
<td>No</td>
<td></td>
<td>End Entities</td>
<td>ca/ee/ca/</td>
</tr>
<tr>
<td>9444</td>
<td>Yes</td>
<td>No</td>
<td>End Entities</td>
<td>ca/ee/ca</td>
</tr>
<tr>
<td>9443</td>
<td>Yes</td>
<td>Yes</td>
<td>Agents</td>
<td>ca/admin/console/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>config/login?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pin=pin</td>
</tr>
<tr>
<td>9445</td>
<td>Yes</td>
<td>No</td>
<td>Configuration</td>
<td>ra/admin/console/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>config/login?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pin=pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Services</td>
<td>ca/services</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Console</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pkiconsole</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><a href="https://host:port/ca">https://host:port/ca</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Registration Manager</td>
</tr>
<tr>
<td>12888</td>
<td>No</td>
<td></td>
<td>End Entities</td>
<td>ee/index.cgi</td>
</tr>
<tr>
<td>12889</td>
<td>Yes</td>
<td>Yes</td>
<td>Agents</td>
<td>agent/index.cgi</td>
</tr>
<tr>
<td>12889</td>
<td>Yes</td>
<td>Yes</td>
<td>Admin</td>
<td>admin/index.cgi</td>
</tr>
<tr>
<td>12890</td>
<td>Yes</td>
<td></td>
<td>Configuration</td>
<td>ra/admin/console/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>config/login?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pin=pin</td>
</tr>
<tr>
<td>12890</td>
<td>Yes</td>
<td></td>
<td>End Entities</td>
<td>ee/index.cgi</td>
</tr>
<tr>
<td>12890</td>
<td>Yes</td>
<td></td>
<td>Services</td>
<td>index.cgi</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Recovery</td>
</tr>
<tr>
<td>10180</td>
<td>No</td>
<td></td>
<td>End Entities</td>
<td>kra/ee/kra</td>
</tr>
<tr>
<td>10444</td>
<td>Yes</td>
<td>No</td>
<td>End Entities</td>
<td>kra/ee/kra</td>
</tr>
<tr>
<td>10443</td>
<td>Yes</td>
<td>Yes</td>
<td>Agents</td>
<td>kra/admin/console/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>config/login?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pin=pin</td>
</tr>
<tr>
<td>10445</td>
<td>Yes</td>
<td>No</td>
<td>Configuration</td>
<td>kra/services</td>
</tr>
<tr>
<td>10445</td>
<td>Yes</td>
<td>No</td>
<td>Console</td>
<td>pkiconsole</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><a href="https://host:port/kr">https://host:port/kr</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Online Certificate</td>
</tr>
<tr>
<td>11180</td>
<td>No</td>
<td></td>
<td>End Entities</td>
<td>ocsp/ee/ocsp</td>
</tr>
<tr>
<td>11444</td>
<td>Yes</td>
<td>No</td>
<td>End Entities</td>
<td>ocsp/ee/ocsp</td>
</tr>
<tr>
<td>11443</td>
<td>Yes</td>
<td>Yes</td>
<td>Agents</td>
<td>ocsp/admin/ocsp</td>
</tr>
<tr>
<td>11445</td>
<td>Yes</td>
<td></td>
<td>Configuration</td>
<td>ocsp/admin/console/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>config/login?</td>
</tr>
</tbody>
</table>

[a] No authentication available 
[b] Obtains OCSP response 
[c] Obtains OCSP status information
<table>
<thead>
<tr>
<th>Service Type</th>
<th>Yes/No</th>
<th>End Entities</th>
<th>URL/Service Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Key Service</td>
<td>11445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>11445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>11445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>13180</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13444</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>13443</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>13445</td>
<td><strong>Yes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>13445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>13445</td>
<td><strong>Yes</strong></td>
<td>No</td>
</tr>
<tr>
<td>Token Processing System</td>
<td>7888</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7890</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7888</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7890</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7890</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7889</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7889</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7889</td>
<td><strong>Yes</strong></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7890</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7890</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
16.5. File and Directory Locations for Certificate System

Certificate System servers consist of subsystems and instances. Server subsystems are servers for a specific type of PKI function. 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.

Server instances are somewhat relocatable and have user-specific default and customized forms and data.

16.5.1. CA Instance Information

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

[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 entity 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 entity 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 has a different tab on that page. The role-specific tab is visible to every user who is a member of that role.
### Table 16.3. CA Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ports</strong></td>
<td>Standard port</td>
</tr>
<tr>
<td></td>
<td>End users port</td>
</tr>
<tr>
<td></td>
<td>End users client authentication port</td>
</tr>
<tr>
<td></td>
<td>Agents port</td>
</tr>
<tr>
<td></td>
<td>Admin port</td>
</tr>
<tr>
<td></td>
<td>Tomcat port</td>
</tr>
<tr>
<td><strong>Main Directory</strong></td>
<td>/var/lib/pki-ca</td>
</tr>
<tr>
<td><strong>Configuration Directory</strong></td>
<td>/etc/pki-ca</td>
</tr>
<tr>
<td><strong>Configuration File</strong></td>
<td>/etc/pki-ca/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-ca/password.conf</td>
</tr>
<tr>
<td><strong>Subsystem Certificates</strong></td>
<td>CA signing certificate</td>
</tr>
<tr>
<td></td>
<td>OCSP signing certificate (for the CA's internal OCSP service)</td>
</tr>
<tr>
<td></td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Audit log signing certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate [a]</td>
</tr>
<tr>
<td><strong>Security Databases</strong></td>
<td>/var/lib/pki-ca/alias</td>
</tr>
<tr>
<td><strong>Log Files</strong></td>
<td>/var/log/pki-ca/logs</td>
</tr>
<tr>
<td><strong>Install Logs</strong></td>
<td>/var/log/pki-ca/logs-install.log</td>
</tr>
<tr>
<td><strong>Process File</strong></td>
<td>/var/run/pki-ca.pid</td>
</tr>
<tr>
<td><strong>Profile Files</strong></td>
<td>/var/lib/pki-ca/profiles/ca</td>
</tr>
<tr>
<td><strong>Email Notification Templates</strong></td>
<td>/var/lib/pki-ca/emails</td>
</tr>
<tr>
<td><strong>Web Services Files</strong></td>
<td>/var/lib/pki-ca/webapps - Agent services</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-ca/webapps.admin - Admin services</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-ca/webapps.ee - End user services</td>
</tr>
</tbody>
</table>

[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.

### 16.5.2. RA Instance Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-ra**; the true value is whatever is specified at the time the instance is created with **pkicreate**.
### Table 16.4. RA Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Standard port (for end users)</td>
</tr>
<tr>
<td></td>
<td>SSL port (for end users)</td>
</tr>
<tr>
<td></td>
<td>SSL port (for agents and administrators)</td>
</tr>
<tr>
<td>Instance Name</td>
<td>pki-ra</td>
</tr>
<tr>
<td>Main Directory</td>
<td>/var/lib/pki-ra</td>
</tr>
<tr>
<td>Configuration Directory</td>
<td>/etc/pki-ra</td>
</tr>
<tr>
<td>Configuration File</td>
<td>/etc/pki-ra/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-ra/nss.conf</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-ra/password.conf</td>
</tr>
<tr>
<td>Subsystem Certificates</td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate [a]</td>
</tr>
<tr>
<td>Security Databases</td>
<td>/var/lib/pki-ra/alias</td>
</tr>
<tr>
<td>Log Files</td>
<td>/var/log/pki-ra/logs</td>
</tr>
<tr>
<td>Install Logs</td>
<td>/var/log/pki-ra/logs Install.log</td>
</tr>
<tr>
<td>Web Services Files</td>
<td>/var/lib/pki-ra/docroot</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-ra/lib</td>
</tr>
</tbody>
</table>

[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.

### 16.5.3. DRM Instance Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-kra**; the true value is whatever is specified at the time the instance is created with **pkicreate**.
### Table 16.5. KRA Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Standard port</td>
</tr>
<tr>
<td></td>
<td>End users secure port</td>
</tr>
<tr>
<td></td>
<td>Agents port</td>
</tr>
<tr>
<td></td>
<td>Admin port</td>
</tr>
<tr>
<td></td>
<td>Tomcat port</td>
</tr>
<tr>
<td>Instance Name</td>
<td>pki-kra</td>
</tr>
<tr>
<td>Main Directory</td>
<td>/var/lib/pki-kra</td>
</tr>
<tr>
<td>Configuration Directory</td>
<td>/etc/pki-kra</td>
</tr>
<tr>
<td>Configuration File</td>
<td>/etc/pki-kra/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-kra/password.conf</td>
</tr>
<tr>
<td>Subsystem Certificates</td>
<td>Transport certificate</td>
</tr>
<tr>
<td></td>
<td>Storage certificate</td>
</tr>
<tr>
<td></td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Audit log signing certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate [a]</td>
</tr>
<tr>
<td>Security Databases</td>
<td>/var/lib/pki-kra/alias</td>
</tr>
<tr>
<td>Log Files</td>
<td>/var/log/pki-kra/logs</td>
</tr>
<tr>
<td>Install Logs</td>
<td>/var/log/pki-kra/logs-install.log</td>
</tr>
<tr>
<td>Process File</td>
<td>/var/run/pki-kra.pid</td>
</tr>
<tr>
<td>Web Services Files</td>
<td>/var/lib/pki-kra/webapps - Agent services</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-kra/webapps.admin - Admin services</td>
</tr>
</tbody>
</table>

[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.

### 16.5.4. OCSP Instance Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is `pki-ocsp`; the true value is whatever is specified at the time the instance is created with `pkicreate`.
### Table 16.6. OCSP Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Standard port</td>
</tr>
<tr>
<td></td>
<td>End users SSL port</td>
</tr>
<tr>
<td></td>
<td>Agents port</td>
</tr>
<tr>
<td></td>
<td>Admin port</td>
</tr>
<tr>
<td></td>
<td>Tomcat port</td>
</tr>
<tr>
<td>Instance Name</td>
<td>pki-ocsp</td>
</tr>
<tr>
<td>Main Directory</td>
<td>/var/lib/pki-ocsp</td>
</tr>
<tr>
<td>Configuration Directory</td>
<td>/etc/pki-ocsp</td>
</tr>
<tr>
<td>Configuration File</td>
<td>/etc/pki-ocsp/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-ocsp/password.conf</td>
</tr>
<tr>
<td>Subsystem Certificates</td>
<td>OCSP signing certificate</td>
</tr>
<tr>
<td></td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Audit log signing certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate [a]</td>
</tr>
<tr>
<td>Security Databases</td>
<td>/var/lib/pki-ocsp/alias</td>
</tr>
<tr>
<td>Log Files</td>
<td>/var/log/pki-ocsp/logs</td>
</tr>
<tr>
<td>Install Logs</td>
<td>/var/log/pki-ocsp/logs-install.log</td>
</tr>
<tr>
<td>Process File</td>
<td>/var/run/pki-ocspocsp.pid</td>
</tr>
<tr>
<td>Web Services Files</td>
<td>/var/lib/pki-ocsp/webapps - Agent services</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-ocsp/webapps.admin - Admin services</td>
</tr>
</tbody>
</table>

[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.

### 16.5.5. TKS Instance Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tks**; the true value is whatever is specified at the time the instance is created with **pkicreate**.
### Table 16.7. TKS Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Standard port</td>
</tr>
<tr>
<td></td>
<td>End users SSL port</td>
</tr>
<tr>
<td></td>
<td>Agents port</td>
</tr>
<tr>
<td></td>
<td>Admin port</td>
</tr>
<tr>
<td></td>
<td>Tomcat port</td>
</tr>
<tr>
<td>Instance Name</td>
<td>pki-tks</td>
</tr>
<tr>
<td>Main Directory</td>
<td>/var/lib/pki-tks</td>
</tr>
<tr>
<td>Configuration Directory</td>
<td>/etc/pki-tks</td>
</tr>
<tr>
<td>Configuration File</td>
<td>/etc/pki-tks/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-tks/password.conf</td>
</tr>
<tr>
<td>Subsystem Certificates</td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Audit log signing certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate [a]</td>
</tr>
<tr>
<td>Security Databases</td>
<td>/var/lib/pki-tks/alias</td>
</tr>
<tr>
<td>Log Files</td>
<td>/var/log/pki-tks/logs</td>
</tr>
<tr>
<td>Install Logs</td>
<td>/var/log/pki-tks/logs-install.log</td>
</tr>
<tr>
<td>Process File</td>
<td>/var/run/pki-tks.pid</td>
</tr>
</tbody>
</table>

[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.

### 16.5.6. TPS Instance Information

The directories are instance specific, tied to the instance name. In these examples, the instance name is **pki-tps**; the true value is whatever is specified at the time the instance is created with **pkicreate**.
### Table 16.8. TPS Instance Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>Standard port (for end users)</td>
</tr>
<tr>
<td></td>
<td>SSL port (for end users)</td>
</tr>
<tr>
<td></td>
<td>SSL port (for agents and administrators)</td>
</tr>
<tr>
<td>Main Directory</td>
<td>/var/lib/pki-tps</td>
</tr>
<tr>
<td>Configuration Directory</td>
<td>/etc/pki-tps</td>
</tr>
<tr>
<td>Configuration File</td>
<td>/etc/pki-tps/CS.cfg</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-tps/nss.conf</td>
</tr>
<tr>
<td></td>
<td>/etc/pki-tps/password.conf</td>
</tr>
<tr>
<td>Subsystem Certificates</td>
<td>SSL server certificate</td>
</tr>
<tr>
<td></td>
<td>Subsystem certificate</td>
</tr>
<tr>
<td>Security Databases</td>
<td>/var/lib/pki-tps/alias</td>
</tr>
<tr>
<td>Log Files</td>
<td>/var/log/pki-tps/logs</td>
</tr>
<tr>
<td>Install Logs</td>
<td>/var/log/pki-tps/logs-install.log</td>
</tr>
<tr>
<td>Web Services Files</td>
<td>/var/lib/pki-tps/docroot</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-tps/cgi-bin</td>
</tr>
<tr>
<td></td>
<td>/var/lib/pki-tps/lib</td>
</tr>
</tbody>
</table>

### 16.5.7. 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 16.9, “Subsystem File Locations”. 
### Table 16.9. Subsystem File Locations

<table>
<thead>
<tr>
<th>Directory Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( /var/lib/instance_name )</td>
<td>Contains the main instance directory, which is the location for user-specific default and customized configuration files, profiles, certificate databases, web files, and other files for the subsystem instance.</td>
</tr>
<tr>
<td>( /usr/share/java/pki )</td>
<td>Contains Java archive files shared by the Certificate System subsystems. Along with shared files for all subsystems, there are subsystem-specific files in subfolders:</td>
</tr>
<tr>
<td></td>
<td>pki/ca/ (CA)</td>
</tr>
<tr>
<td></td>
<td>pki/kra/ (DRM)</td>
</tr>
<tr>
<td></td>
<td>pki/ocsp/ (OCSP)</td>
</tr>
<tr>
<td></td>
<td>pki/tks/ (TKS)</td>
</tr>
<tr>
<td></td>
<td><em>Not used by the RA or TPS subsystems.</em></td>
</tr>
<tr>
<td>( /usr/share/pki )</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>pki/ca/ (CA)</td>
</tr>
<tr>
<td></td>
<td>pki/kra/ (DRM)</td>
</tr>
<tr>
<td></td>
<td>pki/ocsp/ (OCSP)</td>
</tr>
<tr>
<td></td>
<td>pki/ra/ (RA)</td>
</tr>
<tr>
<td></td>
<td>pki/tks/ (TKS)</td>
</tr>
<tr>
<td></td>
<td>pki/tps (TPS)</td>
</tr>
<tr>
<td>( /usr/bin )</td>
<td>Contains the <code>pkicreate</code> and <code>pkiremove</code> instance configuration scripts and tools (Java, native, and security) shared by the Certificate System subsystems.</td>
</tr>
<tr>
<td>( /var/lib/tomcat5/common/lib )</td>
<td>Contains Java archive files shared by local Tomcat web applications and shared by the Certificate System subsystems. <em>Not used by the TPS or RA subsystems.</em></td>
</tr>
<tr>
<td>( /var/lib/tomcat5/server/lib )</td>
<td>Contains Java archive files used by the local Tomcat web server and shared by the Certificate System subsystems. <em>Not used by the TPS or RA subsystems.</em></td>
</tr>
<tr>
<td>( /usr/lib/httpd/modules )</td>
<td>Contains Apache modules shared by TPS and RA subsystems. <em>Not used by the CA, DRM, OCSP, or TKS subsystems.</em></td>
</tr>
<tr>
<td>( /usr/lib64/httpd/modules )</td>
<td></td>
</tr>
<tr>
<td>/usr/lib/mozldap</td>
<td>Mozilla LDAP SDK tools shared by TPS and RA subsystems. Not used by the CA, DRM, OCSP, or TKS subsystems.</td>
</tr>
<tr>
<td>/usr/lib64/mozldap</td>
<td></td>
</tr>
</tbody>
</table>
Supported Algorithms and Curves

When a new subsystem instance is first configured, the Red Hat Certificate System allows subsystems to be cloned, or duplicated, for high availability of the Certificate System. The cloned instances run on different machines to avoid a single point of failure and their databases are synchronized through replication.

A.1. RSA Hashing Algorithms

The following algorithms are available for RSA keys:

- SHA256withRSA (the default)
- SHA1withRSA
- SHA256withRSA
- SHA512withRSA
- MD5withRSA
- MD2withRSA

A.2. ECC Algorithms and Curves

Certificate System does not include a module natively to enable ECC, but it is possible to load and use a third-party PKCS #11 module with ECC-enabled. This is covered in Chapter 9, Installing an Instance with ECC Enabled.

The following algorithms are available for ECC keys:

- SHA256withEC (the default)
- SHA1withEC
- SHA384withEC
- SHA512withEC

The curves available for ECC keys are listed in Table A.1, "ECC Curves".

**NOTE**

The only supported curve for the TPS is nistp256.

**IMPORTANT**

While Certificate System supports all of these curves, hardware security modules or servers may not support some of these curves. Check with the hardware vendor when determining what curves to use.
Table A.1. ECC Curves

<table>
<thead>
<tr>
<th>Curve Family</th>
<th>Supported Curves</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIST, SEC2 Prime</td>
<td>secp521r1 and nistp521, nistp521, secp384r1 and nistp384, nistp384, secp256r1 and nistp256, nistp256, secp256k1, secp224r1 and nistp224, nistp224, secp224k1, secp192r1 and nistp192, nistp192, secp192k1, secp160r2, secp160r1, secp160k1, secp128r2, secp128r1, secp112r2, secp112r1</td>
</tr>
<tr>
<td>NIST, SEC2 Binary</td>
<td>sect571r1 and nistb571, nistb571, sect571k1 and nistk571, nistk571, sect409r1 and nistb409, nistb409, sect409k1 and nistk409, nistk409, sect283r1 and nistb283, nistb283, sect283k1 and nistk283, nistk283, sect239k1, sect233r1 and nistb233, nistb233, sect233k1 and nistk233, nistk233, sect193r2, sect193r1, nistb163, sect163r2 and nistb163, sect163r1, sect163k1 and nistk163, nistk163, sect131r2, sect131r1, sect113r2</td>
</tr>
</tbody>
</table>
### ANSI X9.62 Prime

- prime239v3
- prime239v2
- prime239v1
- prime192v3
- prime192v2
- prime192v1 and nistp192
- prime256v1 and nistp256

### ANSI X9.62 Binary

- c2pnb163v1
- c2pnb163v2
- c2pnb163v3
- c2pnb163v1
- c2tnb191v1
- c2tnb191v2
- c2tnb191v3
- c2onb191v4
- c2onb191v5
- c2pnb208w1
- c2tnb239v1
- c2tnb239v2
- c2tnb239v3
- c2onb239v4
- c2onb239v5
- c2pnb272w1
- c2pnb304w1
- c2tnb359v1
- c2pnb368w1
- c2tnb431r1

## A.3. Key Size Limits and Internet Explorer

Microsoft uses certain cryptographic providers which support only a subset of potential key sizes for RSA and for ECC keys. These are listed in Table A.2, “Providers and Key Sizes”.

The key size support can impact the configuration of profiles that will be used with Internet Explorer and, possibly, the configuration of the CA.
### Table A.2. Providers and Key Sizes

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Provider</th>
<th>Supported Key Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC</td>
<td>Microsoft Software Key Storage Provider</td>
<td>nistp256, nistp384, nistp521</td>
</tr>
<tr>
<td>ECC</td>
<td>Microsoft Smart Card Key Storage Provider</td>
<td>nistp256, nistp384, nistp521</td>
</tr>
<tr>
<td>RSA</td>
<td>Microsoft Base Cryptographic Provider</td>
<td>1024</td>
</tr>
<tr>
<td>RSA</td>
<td>Microsoft Strong Cryptographic Provider</td>
<td>1024, 2048, 3072, 4096, 8192</td>
</tr>
<tr>
<td>RSA</td>
<td>Enhanced Cryptographic Provider</td>
<td>1024, 2048, 3072, 4096, 8192</td>
</tr>
<tr>
<td>RSA</td>
<td>Microsoft Software Key Storage Provider</td>
<td>1024, 2048, 3072, 4096, 8192</td>
</tr>
</tbody>
</table>
Defining the Common Criteria Environment

This appendix provides information about the configuration used to set up Red Hat Certificate System 8.1 in the Common Criteria environment, as well as information for the certification security target.


The purpose of this appendix is to give administrators a guideline on how to set up and maintain the Common Criteria environment. This is separate from the sections for the security target which defines how Red Hat Certificate System satisfies the requirement.

This section lists the configuration that is required or allowed for a Common Criteria environment. The actual procedures and instructions are contained in the Administrator's Guide or this Deployment Guide.

**IMPORTANT**

The Certificate System Registration Manager subsystem is not included in the target of evaluation, and therefore its configuration and features are not included in this discussion.

B.1.1. PKI Overview

Chapter 1, *Introduction to Public-Key Cryptography*

B.1.2. Security Objectives

Section B.4, "Common Criteria: Security Objectives"

B.1.3. Security Requirements

Section B.5, "Common Criteria: Security Requirements"

B.1.4. Target of Evaluation Security Environment Assumptions

Section B.3, "Common Criteria: Security Environment Assumptions"

B.1.5. IT Environment Assumptions

The assumptions about the target of evaluation's environment are that you have the ability to:

- Recover to a viable state after malicious code is introduced and damage occurs.
- Provide time stamps to ensure the sequencing of events can be verified.
- Implement automated notification or other responses to the target security functions-discovered attacks in order to identify attacks and create an attack deterrent.
- Require inspection for downloads.
- Respond to possible loss of stored audit records.
- Run the subsystem instances in an environment where the applications and services are restricted to a trusted path (SELinux).
- Run the subsystem instances in an environment that enforces data integrity and establishes policies for periodic integrity checks (SELinux).

B.1.5.1. Reliable Timestamp

Red Hat Certificate System 8.1 relies on the operating system to provide reliable timestamps. To ensure that the certificates signed by the CA contain accurate timestamps and the audit log events record...
accurate time of event occurrence, Red Hat Certificate System 8.1 administrators need to make sure the operating system has a time-syncing mechanism with a reliable source.

**B.1.5.2. Private and Secret Key Zeroization**

Red Hat Certificate System 8.1 relies on a FIPS 140 validated module accessed through NSS to perform critical key generation, key storage, and zeroization for key destruction. There are no explicit calls from Red Hat Certificate System 8.1 code to do private and secret key zeroization. NSS automatically handles zeroization for Red Hat Certificate System 8.1 by invoking the zeroization routines provided by the cryptographic hardware, so there isn't anything the administrator needs to do specifically to activate this feature.

**B.1.5.3. Password and Certificate Storage**

By default, Red Hat Certificate System 8.1 stores passwords on a per instance basis in 
/var/lib/instance_name/conf/password.conf (protected by Linux operating system permissions). However, for Common Criteria purposes, Red Hat Certificate System 8.1 utilizes the nuxwdog executable to store unencrypted passwords in the kernel keyring through sockets, and the associated instance-specific password.conf file is deleted for security reasons.

Red Hat Certificate System system certificates are stored in an NSS database on a FIPS-validated HSM (Section B.1.5.4, “Hardware Token”).

**B.1.5.4. Hardware Token**

To meet Common Criteria requirements, Red Hat Certificate System 8.1 protects security critical keys and other information by either encrypting it or storing it within a hardware cryptographic module that has been certified to FIPS 140 Level 3 operating in a FIPS-approved or recommended mode of operation. Red Hat Certificate System 8.1 uses the PKCS#11 module provided by the cryptographic hardware vendors to access the hardware cryptographic modules. Supported cryptographic hardware components are listed in Section 5.1.4, “Supported HSM”.

**B.1.5.5. Protection of Private and Secret Keys**

Within a Common Criteria certified environment, Red Hat Certificate System 8.1 certificate private keys and secret keys are to be generated and stored in a FIPS 140-1 level 3 certified hardware cryptographic token. Administrators are required during the installation process to select an HSM which conforms to this requirement in order to generate its CIMC keys.

The Red Hat Certificate System 8.1 private (asymmetric) keys for a CA are:

- Private key associated with the CA server certificate.
- Private key associated with the CA subsystem certificate.
- Private key associated with the CA audit log signing certificate.
- Private key associated with the CA signing certificate.
- Private key associated with the OCSP Responder signing certificate.

The Red Hat Certificate System 8.1 private (asymmetric) keys for a DRM are:

- Private key associated with the DRM server certificate.
- Private key associated with the DRM subsystem certificate.
- Private key associated with the DRM audit log signing certificate.
- Private key associated with the DRM transport certificate.
- Private key associated with the DRM storage certificate (for DRM key archival).

The Red Hat Certificate System 8.1 private (asymmetric) keys for an OCSP are:
Private key associated with the OCSP server certificate.
Private key associated with the OCSP subsystem certificate.
Private key associated with the OCSP audit log signing certificate.
Private key associated with the OCSP signing certificate.

The Red Hat Certificate System 8.1 private (asymmetric) keys for a TKS are:

Private key associated with the TKS server certificate.
Private key associated with the TKS subsystem certificate.
Private key associated with the TKS audit log signing certificate.

The Red Hat Certificate System 8.1 private (asymmetric) keys for a TPS are:

Private key associated with the TPS server certificate.
Private key associated with the TPS subsystem certificate.
Private key associated with the TPS audit log signing certificate.

The sole use of secret (symmetric) keys by Red Hat Certificate System 8.1 is to generate a DES3 key on-the-fly which is used to encrypt a user's private (asymmetric) key before this generated secret key and encrypted user's private key are wrapped together by the DRM's storage key and stored as archived data in the DRM's internal Red Hat Directory Server 8.2 LDAP database.

B.1.6. Red Hat Certificate System 8.1 Privileged Users and Groups (Roles)
The procedures for creating Certificate System subsystem users are covered in the Certificate System Administrator's Guide:

- CA, OCSP, DRM, and TKS users
- TPS users

Of all privileged roles supported by Red Hat Certificate System 8.1, the Certificate Manager agents role and the DRM agent role are the ones that map directly to the officer role defined in the security target and the CIMC PP. The Online Certificate Status Manager agents are a sub-group of the administrator role defined in the CIMC PP.

Table B.1, “Mappings for Default Roles” shows the mapping between Certificate System roles and CIMC PP roles.
### Table B.1. Mappings for Default Roles

<table>
<thead>
<tr>
<th>CIMC PP Role</th>
<th>Certificate System Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>Administrators for the CA, DRM, OCSP, and TKS subsystems</td>
</tr>
<tr>
<td></td>
<td>Online Certificate Status Manager agents</td>
</tr>
<tr>
<td></td>
<td>Administrators for the TPS</td>
</tr>
<tr>
<td>Officer</td>
<td>Certificate Manager agents</td>
</tr>
<tr>
<td></td>
<td>Data Recovery Manager agents</td>
</tr>
<tr>
<td></td>
<td>TPS agents</td>
</tr>
<tr>
<td>Auditor</td>
<td>Auditors from CA, DRM, OCSP, TKS, and TPS</td>
</tr>
</tbody>
</table>

#### B.1.7. Understanding Setup of Common Criteria Evaluated Red Hat Certificate System 8.1

This document describes at a high level the steps for setup, installation, and configuration of Red Hat Certificate System 8.1 in an IT environment of the kind described in Section 4.7, "Implementing a Common Criteria Environment". It gives administrators an idea of what's ahead before starting them on the exact setup steps involved in installation and setup.

**B.1.7.1. Understanding the Common Criteria Environment**

This section describes the environment before Red Hat Certificate System 8.1 is installed and configured.

**B.1.7.1.1. Secure Environment**

This section describes the secure environment you will be instructed to set up before installing and configuring Red Hat Certificate System 8.1.

**B.1.7.1.1.1. Network Environment**

It is important to make sure that only those users that are part of the Red Hat Certificate System 8.1 installation have access to the system that is about to be installed; unauthorized personnel should not have access. It is recommended that you carry out the installation/configuration procedures in an isolated, secure network and re-establish the full network access only when the configuration is complete.

**B.1.7.1.1.2. Operating System Environment**

Because Red Hat Certificate System 8.1 relies on the IT environment to provide the basic operating system file system security, inter-process communication, and process space protection, Red Hat Certificate System 8.1 must be installed and run on an operation system certified at a Common Criteria assurance level no less than the level of Red Hat Certificate System 8.1 itself, such as Red Hat Enterprise Linux 5.6 with SELinux in enforcing mode.

**B.1.7.1.2. Verifying Package Bits**

All Red Hat Certificate System packages are GPG signed by Red Hat, for security. Our key is available at [https://access.redhat.com/security/team/key/#package](https://access.redhat.com/security/team/key/#package). You can verify each package with the following
command:

```
rpm --checksig -v filename
```

If you only wish to verify that each package has not been corrupted or tampered with, examine only the md5sum with the following command:

```
md5sum filename
```

**B.1.7.1.3. Red Hat Certificate System 8.1 Roles Assignment**

In order to maintain accountability, it is prudent to require individual users to log into their individual accounts for regular Red Hat Certificate System 8.1 operations and maintenance. To achieve this, you first have to assign Red Hat Certificate System 8.1 privilege roles to users. It is also recommended that the user ID at the operating system level is the same user ID that is used in Red Hat Certificate System 8.1. Red Hat Certificate System 8.1 allows more than one user to have the same role (for example, you can have two CA agents); however, Red Hat Certificate System 8.1 does not allow one person to have more than one role within the same subsystem (for example, the user Joe cannot be both the CA administrator and agent for the same ca subsystem). See Section B.1.6, "Red Hat Certificate System 8.1 Privileged Users and Groups (Roles)" for a description of the various Red Hat Certificate System 8.1 privileged roles.

**B.1.7.1.4. Who Needs to Be Present**

During the installation and configuration, the Red Hat Certificate System 8.1 audit function is not operational, so it is crucial that all Red Hat Certificate System 8.1 roles be present to witness the installation and make necessary operations and decisions. Section B.1.6, "Red Hat Certificate System 8.1 Privileged Users and Groups (Roles)" lists the default Certificate System user types, while the procedures for creating the required operating system accounts are in Section 5.3.7, "Setting up Operating System Users and Groups".

Required configuration files like `CS.cfg` are protected and audited within the operating system environment. Therefore, only operating system administrators can directly edit these files. During installation, an operating system administrator must be present to help set any custom configuration in the `CS.cfg` file that is required.

**B.1.8. Common Criteria Deployment Scenarios**

As long as the subsystems are installed and configured following the Common Criteria environment rules and guidelines outlined in this book, Red Hat Certificate System 8.1 can be deployed in any desired scenario. A deployment can, for example, have a root CA, a CA subordinate to an Red Hat Certificate System 8.1 CA, a CA subordinate to a public third-party CA, or have any number of CAs in vertical or horizontal chains as long as they follow the constraints contained in the CA signing certificate.

Likewise, Data Recovery Manager can work with any CA within the Red Hat Certificate System environment. An OCSP responder can work with any CA within the environment, and one OCSP responder can work with multiple CAs.

Similarly, multiple TKS and TPS instances may be set up to deploy a token management system.

A Common Criteria environment can be set up with all subsystems installed on the same host, or with some or all subsystems on separate hosts.

**B.1.9. Understanding Subsystem Setup**

This section describes at a high-level what to expect when configuring an Red Hat Certificate System
8.1 subsystem. Links to detailed procedures for each feature are given in Chapter 15, After Configuration: Checklist of Configuration Areas for Deploying Certificate System.

### B.1.9.1. Red Hat Certificate System 8.1 Role Users and Authorization

In Red Hat Certificate System 8.1, you create users and then assign them to groups (also called *roles*) to give them the privileges of the role represented by the group membership. You need to set up at least one auditor role user, one agent role user, and one administrator role user for each subsystem. You specify the first administrator role user when you install the subsystem. You will be setting up the administrative interface (Red Hat Certificate System 8.1 console) for SSL authentication; all agent role users, auditor role users, and administrator role users you set up will need to obtain a certificate, and the certificates for those role users will need to be stored with their role user entries. It is recommended that you have the auditor role users, administrator role users, and agent role users use their hardware tokens to submit requests to the end-entity interface of the Certificate Manager that will process the request.

Role users must follow the site policy on changing passwords and renewing certificates at appropriate intervals and to appropriate values. The site policy will govern the required settings for the password policy, such as proper password lengths, the password history, and password strength.

You can also configure new groups and assign them privileges other than the default privileges assigned to the default groups, thus creating new roles in the subsystem. You do this by creating a group, setting up ACIs for this group in the ACLs pertinent to the privileges you want to define for this group.

For complete information on creating users, assigning them to groups, creating groups, and changing the ACLs, see section 13.1. About Authorization in the Administrator’s Guide and Section 2.4.6, “Users, Authorization, and Access Controls”.

**NOTE**

While there is flexibility to add groups and change the ACLs under the Common Criteria environment, be cautious about creating scenarios that are not secure, such as allowing anyone access to the agent services interface. Also, be careful when changing the default roles or when adding roles so that you do not create security holes or vulnerabilities.

Any custom plug-ins for the access control feature are not part of the Common Criteria environment.

### B.1.9.2. Audit Logs

The Common Criteria environment requires that the signed audit log file feature be enabled and configured. Sections 14.2.5. Managing Signed Audit Logs and 14.3.3. Managing Audit Logs in the Administrator’s Guide provide complete information about how to set up the signed audit logs.

Auditors must comply with the site policy to run the AuditVerify with sufficient frequency to verify the audit log signatures, along with reviewing the audit logs. Although the Certificate System subsystems are designed to deny operations that violate defined security policies, it is up to the site to define operational policies on the audit log review frequency to detect any type of attack.

If a subsystem is unable to write to its audit logs for any reason, that Certificate System subsystem will shut down to prevent possible loss of audit records.

### B.1.9.3. Certificate Profiles

In the Common Criteria environment, you must set up the certificate profiles feature for certificate...
enrollment in a CA subsystem. You can set up and enable any or all of the pre-built certificate profiles. You can also create other certificate profiles in the Red Hat Certificate System administrative console using the defaults, constraints, inputs, and outputs that are defined. Custom plug-ins for any of the components of the certificate profile feature are not supported as part of the Common Criteria environment. It is important to note that only the Red Hat Certificate System 8.1 CA administrators are allowed to configure the certificate enrollment profiles, such as setting ranges for fields and enabling extensions. It is the Red Hat Certificate System CA agents’ responsibility to approve the fields and extensions in the certificate profiles enabled by the administrators. You will be instructed on how to perform these operations.

See Section 4.4.6, “Using and Customizing Certificate Profiles” and section 2.2 in the Administrator’s Guide for complete information about certificate profiles.

B.1.9.4. Authentication

In the Common Criteria environment, you can enable and configure the agent-approved authentication method or any of the authentication plug-ins in conjunction with a certificate profile. You cannot set up certificate-based enrollment or in-person enrollment. See Section 4.4.7, “Planning Authentication Methods” and chapter 8 in the Administrator’s Guide for complete information about authentication.

B.1.9.5. CRLs

In a CA subsystem, you can set up the CRL feature and any of the three issuing points for CRLs under the Common Criteria environment. When setting up the CRL feature, you cannot set up a CRL that does not have an update frequency specified in the Update at this frequency field. Compliant CRLs must contain the nextUpdateTime extension, which will not be filled in correctly if an update frequency is not specified. Note that you can specify an update frequency and also specify that the CRL is updated every time a certificate is revoked; both settings will be respected. For complete information on CRLs, see chapter 6 in the Administrator’s Guide.

B.1.9.6. Jobs

Jobs are events that can be scheduled to be activated periodically. You can set up any of the available job plug-ins in the Common Criteria environment. Any custom plug-ins for any of the jobs feature are not part of the Common Criteria environment, however. Although you will not specifically be instructed to configure jobs, you can turn on and configure any jobs that are provided by default. If you want notifications to be sent, be careful when selecting the email addresses to use, and make sure they belong to appropriate roles. For complete information on jobs, see chapter 10 in the Administrator’s Guide.

B.1.9.7. Notifications

Automated email notifications are event-driven tasks that send out an email through SMTP when a specified event occurs. You can set up any of the available Notification plug-ins in the Common Criteria environment. Custom plug-ins for the Notification feature are not part of the Common Criteria environment, however. Although you will not specifically be instructed to configure notifications, you can turn on and configure any Notification task that is provided by default. Be careful when selecting the email addresses to use, and make sure they belong to appropriate roles. For complete information on notifications, see chapter 9 in the Administrator’s Guide.

B.1.9.8. Publishing

You can set up any of the methods of publishing in any of the configurations available in Red Hat Certificate System 8.1 in the Common Criteria environment. If you set up LDAP publishing, it is highly recommended that you set it up using SSL client authentication and that you set up the Directory Server in SSL mode as well. For information about publishing, see chapter 7 in the Administrator’s Guide.
B.1.9.9. Self Tests

Red Hat Certificate System 8.1 provides a self-diagnostic feature that checks the sanity of a few key items during an Red Hat Certificate System 8.1 subsystem startup. Any failed self test is an indication of a fatal error to the subsystem. Technically, Red Hat Certificate System 8.1 allows users to add their own self tests as plug-ins; however, only the self-tests provided by default are Common Criteria evaluated. You will not be instructed to configure self tests during the Red Hat Certificate System 8.1 Common Criteria setup, but this is because the self tests are turned on and operational by default. For more information, see section 12.11 in the Administrator's Guide.

B.1.9.10. Key Archival and Recovery

The DRM subsystem provides features to archive user private keys for encryption-only certificates. It also provides features to recover the user private keys that it has archived. Key recovery requires a minimum of two Data Recovery Manager agents to work to approve a recovery request cooperatively; the DRM settings can be configured to require even more than two recovery agents to approve a recovery request. You will be instructed to configure the key recovery and key archival settings for the Data Recovery Manager and establish trust with the Certificate Manager. For complete information on key archival and recovery, see chapter 3 in the Administrator's Guide.

B.1.9.11. OCSP Responder Revocation Information Store

The OCSP Responder revocation store contains information about where the CRLs can be retrieved for serving OCSP requests. You will be instructed to configure the Online Certificate Status Manager revocation store. If you set up the Online Certificate Status Manager to use the ldapstore option, the LDAP store you use must be configured for SSL authentication. For complete information about the OCSP responder, see section 6.6 in the Administrator's Guide.

B.1.9.12. Token Management System

Certificate System creates, manages, renews, and revokes certificates, as well as archiving and recovering keys. For organizations which use smart cards, the Certificate System has a token management system — a collection of subsystems with established relationships — to generate keys and requests and receive certificates to be used for smart cards. For details, see section 1.4 in the Administrator's Guide.

To understand how to set up and use smart cards through the Token Key Service (TKS) and Token Processing System (TPS), see section 5.1 in the Administrator's Guide.

B.1.9.13. Features That Are Not Part of the Common Criteria Environment

Some features or ways of configuring Red Hat Certificate System 8.1 are not part of the Common Criteria environment:

- Using anything other than hardware tokens to create and store CIMC keys and certificates is not supported for Common Criteria.
- The watchdog daemon prompts for passwords to ensure that passwords are never entered or stored in the clear. Using the plain-text password file, `password.conf` is not supported for Common Criteria.
- For security, all services should run using secure connections. Using the administrative interface (the subsystem Java console) in non-SSL client authentication mode is not supported for Common Criteria.
- Using a Registration Manager is not supported for Common Criteria.
- For security, all services should run using secure connections. Running the internal database or any publishing LDAP database in non-SSL client authentication mode is not supported for Common Criteria.
Enrollment is supported using certificate profiles. Using the deprecated policy feature is not supported.

Certificate System services use the operating system settings to provide an additional layer of security. SELinux must be in enforcing mode for the Common Criteria environment, not disabled or permissive.

For security, all services should run using secure connections. Configuring OCSP publishing to a non-SSL port on the OCSP is not supported for Common Criteria.

Customized plug-ins for jobs, notifications, or other extensible areas allow Certificate System performance to be extended, but they cannot be evaluated as part of the target of evaluation. Adding a custom plug-in in essence breaks the Common Criteria assurance. If adding custom plug-ins is unavoidable, it is the responsibility of all role users to carefully evaluate these plug-ins before making them part of the system.

As with Certificate System plug-ins, the Certificate System ACLs provide comprehensive security for subsystem/subsystem and subsystem/user interactions. Any changed default ACLs for a custom environment cannot be evaluated and is therefore not supported for Common Criteria. The default ACLs contain access control enforcement requirements specified in the CIMC Security Level 3 protection profile. Caution must be taken when making changes to them.

Instructions for disabling these features, where necessary, are included within the administration documentation to conform to the defined Common Criteria environment.

B.1.10. Reporting Security Flaws

Although a problem may reported through email by contacting pki-users@redhat.com, more serious issues should be reported to Red Hat support and entered as a software flaw. All software flaw reporting utilizes the Red Hat bug-tracking system Bugzilla. Bugzilla allows software flaws to be reported against specific versions of Red Hat Certificate System.

Updates are delivered through the Red Hat Errata system and can be installed using local or remote content repositories and the yum system tools or by downloading packages from Red Hat Network. Applying updates and new packages is described at https://access.redhat.com/kb/docs/DOC-11259.

Support for a specific issue may be obtained through several different avenues:

- The Red Hat Knowledgebase. Many technical issues can be resolved the Knowledgebase articles. Before contacting technical support, search the Knowledgebase to see if the question or issue is already a known case.
- Production support. Red Hat Technical Support can be contacted through web or phone depending upon the level of service:
  - Basic support is for non-essential or prototype systems.
  - Standard support is for most business systems that are not mission critical and only require support during normal business hours.
  - Premium support is for mission critical services, which require rapid response.
  This is described in more detail on the Red Hat Customer Portal as part of explaining the service-level agreements, https://access.redhat.com/support/offerings/production/sla.html.
- Technical account manager (TAM). A service provided by supper service highly personalized support relationships. This service offers a highly-skilled, proactive support engineer who knows the client personally and understands their IT infrastructure, internal processes and overall business needs. The TAM is accountable for driving technical issues like bugs. This level of support provides access to a separate issue tracker for reporting and tracking all technical requests.
  This is described in more detail on the Red Hat Customer Portal at https://access.redhat.com/support/offerings/tam/.
B.1.11. Relevant Links

The regular installation guide contains the information necessary to select Common Criteria-required platforms and for the Common Criteria installation process:

- Section B.2, “Example Common Criteria Installations”
- Section 5.1, “Supported Platforms, Hardware, and Programs”
- Section 5.1.2, “Supported Web Browsers”

B.2. Example Common Criteria Installations

These example provide explicit and detailed configuration examples for both token management and non-token management deployments. These use example server names, users, and other configuration options.

B.2.1. Non-TMS Common Criteria Setup Procedures

These are the instructions for establishing a Common Criteria test environment for a Non-TMS (Token Management System). Throughout this document, reference links to the Red Hat Certificate System 8.1 Guidance documents have been provided for users as either a well-documented stand-alone procedure, or as an optional source of additional information, and will be clearly denoted as to their purpose.

**NOTE**

For the purposes of this test, everything is installed/configured on a single machine.

B.2.1.1. Prerequisites: RHEL 5.6 Operating System Installation and Configuration

B.2.1.1.1. Non-TMS Inventory

This test requires a single 64-bit x86_64 machine containing a 64-bit version of the Red Hat Enterprise Linux (RHEL) Server release 5.6 operating system for the x86_64 architecture. The machine, which for the purposes of this document will be referred to with a hostname of server.example.com, should be registered with Red Hat Network (RHN) so that it may be fully updated with the latest packages for this version of the operating system. For example, once the operating system has been installed, this can be accomplished by becoming the root user, and executing the following command:

```
[root@server]# yum -y update
```

**NOTE**

In the command listed above, [root@server]# is merely an example command-line prompt. This convention is used throughout this document.

Additionally, the user will need to have an nCipher nShield Hardware Security Module (HSM) attached to this machine for compliance with FIPS 140-2 (Level 3). Details for installation and configuration of this device are provided in Section B.2.1.2, “Hardware Security Module Installation”. For configuring FIPS mode, see the hardware vendor documentation.
Finally, for the purposes of Common Criteria certification of a non-TMS, the user will need to have the appropriate Red Hat subscriptions to download and install the following applications to this machine (detailed download, installation, and configuration instructions are provided below in various sections of this document):

- Tomcat 5.5.23
- Red Hat Directory Server 8.2
- Red Hat Certificate System 8.1 subsystems:
  - Certificate Authority (CA)
  - Data Recovery Management (DRM) - also called a Key Recovery Authority (KRA)
  - Online Certificate Status Protocol (OCSP) Responder
- Red Hat Certificate System (Red Hat Certificate System) 8.1 Console
- Firefox 10 and later
- Open JDK/JRE 1.6
- JSS 4.6
- NSS 3.12
- nss-tools 3.12
- mozldap-tools

B.2.1.1.2. Hardware Security Module Installation

If the nCipher nShield is not setup, insert the accompanying DVD, or mount the ISO and cd into the "Documents" folder. For example, assuming that the ISO is available in /tmp:

```
[root@server]# mkdir /mnt/iso
[root@server]# cd /mnt/iso/document
[root@server]# mount -o loop /tmp/nCSS_linux64_user_11_40.iso /mnt/iso/
```

```
ls
dr-xr-xr-x 2 root root 2048 Aug 2 2010 netHSM
-r-xr-xr-x 1 root root 967656 Aug 2 2010 nShield_Connect_Physical_Security_Checklist.pdf
-r-xr-xr-x 1 root root 310693 Aug 2 2010 nShield_Connect_Quick_Start_Guide.pdf
-r-xr-xr-x 1 root root 2409806 Aug 2 2010 nShield_Connect_User_Guide.pdf
-r-xr-xr-x 1 root root 610234 Aug 2 2010 nShield_Hardware_Installation.pdf
-r-xr-xr-x 1 root root 278386Aug 2 2010 nShield_Quick_Start_Guide.pdf
-r-xr-xr-x 1 root root 2449190 Aug 2 2010 nShield_User_Guide.pdf
-r-xr-xr-x 1 root root 255087 Aug 2 2010 nToken_Install_Guide.pdf
```

Follow the vendor's instructions to setup the nCipher nShield.

Once the nCipher nShield has been installed, ensure that the connection status is OK (by using grep):

```
[root@server]# /opt/nfast/bin/enquiry | grep -i status
connection status OK
```
To enable FIPS mode, open the security UI:

```
/opt/nfast/bin/ksafe
```

In the 'Security World' tab, make sure that "Strict FIPS 140-2 Level III" is set to yes.

For configuring FIPS mode, see the hardware vendor documentation.

Optional Reference: Section 8.1, “Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates”

### B.2.1.1.3. SELinux Enforcement

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering.

Optional Reference: SELinux section of the Red Hat Enterprise Linux Deployment Guide

Red Hat recommends running Certificate System with SELinux in enforcing mode, to make the most of the security policies.

The `usr/sbin/getenforce` command will give the status of SELinux. If running `usr/sbin/getenforce` returns a result of either disabled or permissive, then it needs to be enabled to enforcing. As the root user, edit the file `/etc/sysconfig/selinux` and set `SELINUX=Enforcing`, then reboot the system.

```
[root@server]# /usr/sbin/getenforce
Enforcing
```

```
[root@server]# /usr/sbin/sestatus
SELinux status: enabled
SELinuxfs mount: /selinux
Current mode: enforcing
Mode from config file: enforcing
Policy version: 21
Policy from config file: targeted
```

Optional Reference: Section 5.3.6, “Enabling SELinux”

### B.2.1.1.4. Operating System User and Group Creation

This test requires that certain operating system users and groups exist. Log into the RHEL 5.6 machine as super user and ensure that both the UID and the GID of pkiuser are 17:

```
[root@server]# grep pkiuser /etc/group
pkiuser:x:17:
```

```
[root@server]# grep pkiuser /etc/passwd
pkiuser:x:17:17:Red Hat Certificate System:/usr/share/pki:/sbin/nologin
```

In the event that the pkiuser group is not present, make certain that the appropriate tool packages are installed and recheck the pkiuser user and group:
If the pkiuser user or group are still not present, add them:

```
[root@server]# /usr/sbin/groupadd -g 17 -r pkiuser
[root@server]# /usr/sbin/useradd -g pkiuser -d /usr/share/pki -s /sbin/nologin -c
   "Red Hat Certificate System" -u 17 -r pkiuser
```

If the pkiuser user or group exist but do not have a uid:gid of 17:17, reset the values:

```
[root@server]# /usr/sbin/userdel pkiuser
[root@server]# /usr/sbin/groupdel pkiuser
[root@server]# /usr/sbin/groupadd -g 17 -r pkiuser
[root@server]# /usr/sbin/useradd -g pkiuser -d /usr/share/pki -s /sbin/nologin -c
   "Red Hat Certificate System" -u 17 -r pkiuser
```

Create an administration group:

```
[root@server]# /usr/sbin/groupadd -r pkiadmin
```

Create an auditor group:

```
[root@server]# /usr/sbin/groupadd -r pkiaudit
```

### Note

Linux distinguishes system users such as **pkiuser** (which are often referred to as daemons which have no real person associated with them) from standard users that comprise the PKI administrators, agents, and auditors which are associated with real people.

Optional Reference: *Users and Groups*

If they do not already exist, create standard user accounts for either pkiadmin or pkiaudit, but never allow any standard user access to both groups.

For example, to create an administrator called John Smith:

```
[root@server]# /usr/sbin/useradd -g pkiadmin -d /home/jsmith -s /bin/bash -c
   "Red Hat Certificate System Administrator" -m jsmith
```

For example, to create an auditor called Sally Jones:

```
"Red Hat Certificate System Administrator" -m jsmith
```
Optional Reference: [Section 5.3.7, “Setting up Operating System Users and Groups”](#)

**B.2.1.1.5. Operating System User and Group Association**

Once all operating system groups have been established (see [Section B.2.1.1.4, “Operating System User and Group Creation”](#)), the pkiuser system user must be associated with both the pkiadmin and the pkiaudit groups:

```
[root@server]# /usr/sbin/usermod -a -G pkiadmin pkiuser
[root@server]# /usr/sbin/usermod -a -G pkiaudit pkiuser
```

**Important**

The pkiuser user is allowed to be associated with both the pkiadmin and the pkiaudit groups because pkiuser is a system user. Regular users are never allowed to be associated with both groups because no regular user is ever allowed to be both an administrator and an auditor.

If a pre-existing user is re-assigned to become an administrator, they can be associated with the pkiadmin group:

```
[root@server]# grep mnelson /etc/group
mnelson:x:1099:

[root@server]# grep mnelson /etc/passwd
mnelson:x:1099:1099:Mary Nelson:/home/mnelson:/bin/bash

[root@server]# /usr/sbin/usermod -a -G pkiadmin mnelson
```

Similarly, if a pre-existing user is re-assigned to become an auditor, they can be associated with the pkiaudit group:

```
[root@server]# grep blarson /etc/group
blarson:x:1257:
[root@server]# grep blarson /etc/passwd
blarson:x:1257:1257:Bob Larson:/home/blarson:/bin/bash
[root@server]# /usr/sbin/usermod -a -G pkiaudit blarson
```

Additionally, since this test utilizes an nCipher nShield HSM, both the ‘pkiuser’ user and any individuals in charge of administration of this device must be associated with the ‘nfast’ group:
Once all associations have been generated, ensure that the groups for users 'pkiuser', 'jsmith', 'sjones', 'mnelson', and 'blarson' are correct:

```
[root@server]# groups pkiuser jsmith sjones mnelson blarson
pkiuser : pkiuser pkiadmin pkiaudit nfast
jsmith : pkiadmin nfast
sjones : pkiaudit
mnelson : mnelson pkiadmin nfast
blarson : blarson pkiaudit
```

Optional Reference: Section 5.3.7.2.3, “Associating Existing User Accounts with PKI Groups”

### B.2.1.2. Red Hat Certificate System 8.1 Subsystem Packages

#### B.2.1.2.1. Directory Server Subsystem Package Installation

This machine has been registered with Red Hat Network (RHN) to contain the appropriate Red Hat subscriptions to download and install Red Hat Directory Server 8.2 (see Section B.2.1.1, “Prerequisites: RHEL 5.6 Operating System Installation and Configuration”).

**NOTE**

Check the `/etc/yum.repos.d/` directory for the existence of either `epel.repo` or `epel-testing.repo`. If either file is present, make certain that ALL sections in both `epel.repo` and `epel-testing.repo` have been set to `enabled=0` so that no EPEL packages accidentally override the officially desired RHEL RPMS.

Install the OpenLDAP tools:

```
[root@server]# yum install openldap-clients
```

To install all of the Red Hat Directory Server 8.2 packages, perform the following command:

```
[root@server]# yum install redhat-ds
```

Optional Reference: Section 5.3.3, “Installing Red Hat Directory Server”

#### B.2.1.2.2. Certificate System Subsystem Package Installation

This machine has been registered with Red Hat Network (RHN) to contain the appropriate Red Hat subscriptions to download and install Red Hat Certificate System 8.1 (see Section B.2.1.1, “Prerequisites: RHEL 5.6 Operating System Installation and Configuration”).
NOTE

Check the `/etc/yum.repos.d/` directory for the existence of either ‘epel.repo’ or ‘epel-testing.repo’. If either file is present, make certain that ALL sections in both ‘epel.repo’ and ‘epel-testing.repo’ have been set to ‘enabled=0’ so that no EPEL packages accidentally override the officially desired RHEL RPMS.

To install all of the Red Hat Certificate System 8.1 packages necessary for this test, perform the following commands:

```
[root@server]# yum install pki-ca
[root@server]# yum install pki-kra
[root@server]# yum install pki-ocsp
[root@server]# yum install pki-console
[root@server]# yum -y update
```

B.2.1.3. Hardware Security Module SELinux Context

Prior to creating any Red Hat Certificate System 8.1 instances, since this test requires using an nCipher nShield HSM (see Section B.2.1.1.2, “Hardware Security Module Installation”), restore the SELinux context by running the 'restorecon' command as the super user and restart the nCipher nShield HSM:

```
[root@nontps]# restorecon -R /opt/nfast
[root@nontps]# restorecon -R /dev/nfast
[root@nontps]# /opt/nfast/sbin/init.d-ncipher restart
```

**Important**

This procedure **must** be executed **AFTER** installation of the ‘pki-selinux’ package but **BEFORE** any Red Hat Certificate System 8.1 instance is created, and **must** be executed once on every machine hosting an HSM-based Red Hat Certificate System 8.1 instance.

B.2.1.4. Directory Server Instance Installation and Configuration

Since Red Hat Directory Server 8.2 has been installed on this machine (see Section B.2.1.2.1, “Directory Server Instance Installation and Configuration”)

```
Server Subsystem Package Installation), the user should now install and configure a Red Hat Directory Server 8.2 instance appropriate for use by the Red Hat Certificate System 8.1 instances required for this test.

**Important**

The Certificate System SELinux policies assume that the Red Hat Directory Server is listening over the standard LDAP/LDAPS ports, 389 and 636, respectively.

Additionally, for the purposes of this test, this document uses a convention of all passwords being "Secret123".

To install a Red Hat Directory Server 8.2 instance appropriate for use by the Red Hat Certificate System 8.1 instances included in this test, perform the instructions in the following procedural reference choosing default values:

Procedural Reference: RHDS 8.2: Typical Setup

**B.2.1.5. CA Instance Creation and Configuration**

**B.2.1.5.1. CA Instance Creation**

Use the `pkicreate` tool to create an Red Hat Certificate System 8.1 instance of a CA.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the 'pkicreate' tool as shown below:

```
[root@server]# pkicreate -pki_instance_root=/var/lib  
   -pki_instance_name=pki-ca            
   -subsystem_type=ca                  
   -agent_secure_port=9443             
   -ee_secure_port=9444                
   -ee_secure_client_auth_port=9446    
   -admin_secure_port=9445             
   -unsecure_port=9180                 
   -tomcat_server_port=9701            
   -user=pkiuser                       
   -group=pkiuser                      
   -audit_group=pkiaudit               
   -redirect conf=/etc/pki-ca          
   -redirect logs=/var/log/pki-ca      
   -verbose
```

Optional Reference: Section 6.1, “About pkicreate”

**B.2.1.5.2. Directory Server SSL Server Authentication Configuration (Temporary)**
This section creates a temporary server certificate and temporary self-signed CA to establish SSL communication between the Red Hat Directory Server and the Red Hat Certificate System CA during its configuration. These certificates will be replaced in Section B.2.1.5.5, “Directory Server SSL Server Authentication Configuration (Trusted)”.

Stop the CA instance before modifying its NSS security databases:

```
[root@server]# /sbin/service pki-ca stop
Stopping pki-ca: ..                                        [  OK  ]
```

Stop the Red Hat Directory Server instance before modifying its NSS security databases:

```
[root@server]# /sbin/service dirsrv stop
Shutting down dirsrv: server...                                    [  OK  ]
```

Make a backup of Red Hat Directory Server contents:

```
[root@server]# cd /etc/dirsrv/slapd-server/

[root@server]# ls
cert8.db      dse.ldif      dse.ldif.startOK   key3.db  secmod.db
certmap.conf  dse.ldif.bak  dse_original.ldif  schema   slapd-collations.conf

[root@server]# tar -cf /tmp/db-backup.tar *
```

Using a password file, establish a password for the existing NSS security databases:

```
[root@server]# echo Secret123 > /tmp/pwdfile

[root@server]# /usr/bin/certutil -W -d . -f /tmp/pwdfile

[root@server]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server
Certificate Nickname                                         Trust Attributes
SSL, S/MIME, JAR/XPI
```

Create a temporary Self-Signed CA certificate for LDAP:
NOTE

In the command below, an unlimited path length is specified by a negative one (-1).

```
[root@server]# /usr/bin/certutil -S -n "LDAP Self Signed CA certificate" -s "cn=LDAPCA-cert,dc=example,dc=com" -2 -x -t "CT,," -m 1000 -v 120 -d . -k rsa -f /tmp/pwdfile
```

A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!

```
|***********************************************|
```

Continue typing until the progress meter is full:

```
|******************************************************|
```

Finished. Press enter to continue:

Generating key. This may take a few moments...

```
Is this a CA certificate [y/N]?
y
Enter the path length constraint, enter to skip [<0 for unlimited path]: > -1
```

```
Is this a critical extension [y/N]?
y
```

```
[root@server]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server
Certificate Nickname                           Trust Attributes
LDAP Self Signed CA certificate                SSL, S/MIME, JAR/XPI
```

Create a temporary Server Cert (and sign it using the temporary CA certificate created above):
A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

```
|************************************************************|
```

Finished. Press enter to continue:

Generating key. This may take a few moments...

Start the Red Hat Directory Server instance:

```
[root@server]# /sbin/service dirsrv start
Starting dirsrv: server... [ OK ]
```

Create an LDIF file by running the following command:

```
[root@server]# /sbin/service dirsrv start
Starting dirsrv: server... [ OK ]
```

In the command listed below, "> " is merely the command-line prompt; do NOT type these initial two characters.
Enable SSL in the LDAP Server by running the following command:

```
[root@server]# /usr/bin/ldapmodify -x -h server.example.com -p 389 -D "cn=Directory Manager" -w Secret123 -f /tmp/example.ldif
modifying entry "cn=config"
modifying entry "cn=encryption,cn=config"
adding new entry "cn=RSA,cn=encryption,cn=config"
```

Ensure that `etc/dirsrv/slapd-server/dse.ldif` has appropriate directives for SSL/TLS to work smoothly:

```
[root@server]# egrep -i 'nsslapd-security|nsslapd-securePort|nsslapd-securePort: 636|nsslapd-security|nsslapd-securePort: 636|nsslapd-securePort: 636
```

```
nsSSLClientAuth: allowed
nsSSL3: on
nsSSL3Ciphers: -rsa_null_md5,+rsa_rc4_128_md5,+rsa_rc4_40_md5,+rsa_rc2_40_md5,+rsa_fips_des_sha,+rsa_3des_sha,+rsa_fips_3des_sha,+fortezza,+fortezza_rc4_128_sha,+fortezza_null,+tls_rsa_export1024_with_rc4_56_sha,+tls_rsa_export1024_with_des_cbc_sha
```
Restart the Red Hat Directory Server instance (on a secure port):

```
[root@server]# /sbin/service dirsrv restart
Shutting down dirsrv:
    server...                                            [  OK  ]
Starting dirsrv:
    server...Enter PIN for Internal (Software) Token:
        [  OK  ]
```

Perform an LDAP search with SSL/TLS to ensure that everything is working smoothly:

```
[root@server]# /usr/lib64/mozldap/ldapsearch -h server.example.com -p 636 -Z -P /etc/dirsrv/slapd-server/cert8.db -D "cn=Directory Manager" -w Secret123 -b "cn=config" objectclass=* dn
version: 1
dn: cn=config
    dn: cn=encryption,cn=config
    dn: cn=features,cn=config
    dn: cn=mapping tree,cn=config
    .
    .
```

Export the CA certificate from dirsrv into base-64 pem format:

```
[root@server]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server -n "LDAP Self Signed CA certificate" -a > ldap-ca.crt
```

Import the temporary LDAP self-signed CA certificate into the pki-ca's NSS security database with the appropriate trust attributes set:

```
[root@server]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias
Certificate Nickname                                    Trust Attributes
Server-Cert cert-pki-ca                                SSL, S/MIME, JAR/XPI

[root@server]# /usr/bin/certutil -A -i ldap-ca.crt -t "CT,C,C" -d /var/lib/pki-ca/alias -n "LDAP Self Signed CA certificate"
```
Important

Configuring SSL between Red Hat Certificate System and Red Hat Directory Server won't work unless the appropriate trust attributes ('CT,C,C') are set.

List certificates in the pki-ca NSS security database after importing the LDAP Self Signed CA certificate:

```
[root@server]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias/

Certificate Nickname                     Trust Attributes
SSL,S/MIME,JAR/XPI                        
Server-Cert cert-pki-ca                   CTu,Cu,Cu
LDAP Self Signed CA certificate           CT,C,C
```

Start the CA instance:

```
[root@server]# /sbin/service pki-ca start
Starting pki-ca:
    Using Java Security Manager
    Constructing 'pki-ca.policy' Security Policy
Starting pki-ca:                               [  OK  ]

pki-ca (pid 26728) is running ...            

'pki-ca' must still be CONFIGURED!          
(see /var/log/pki-ca-install.log)
```

Clean up temporary files:

- Move `/tmp/db-backup.tar` to a secure location, or delete it from the system:
  
  ```
  [root@server]# rm -f /tmp/db-backup.tar
  ```

- Delete `/tmp/example.ldif` from the system:
  
  ```
  [root@server]# rm -f /tmp/example.ldif
  ```

- Delete `/tmp/pwdfile` from the system:
  
  ```
  [root@server]# rm -f /tmp/pwdfile
  ```


B.2.1.5.3. Hardware Security Module CA Configuration

Follow instructions in the following procedural reference:

Procedural Reference: Section 8.1.2.3, "Installing External Tokens and Unsupported HSM"
B.2.1.5.4. CA Instance Configuration

**NOTE**

This section involves configuring a CA instance using SSL with Directory Server

Since `pkicreate` has been executed in Section B.2.1.5.1, “CA Instance Creation”, obtain the URL to configure the CA by doing a `tail /var/log/pki-ca-install.log`:

```
[root@server]# tail /var/log/pki-ca-install.log
[2011-03-27 01:40:59] [debug] Restorecon file context for /etc/pki-ca/tomcat5.conf
[2011-03-27 01:40:59] [debug] Setting selinux context pki_ca_port_t for 9446
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' runlevel to '-'
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' start priority to '81'
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' stop priority to '19'
[2011-03-27 01:41:04] [debug] Registered 'pki-ca' with '/sbin/chkconfig'.
[2011-03-27 01:41:14] [log] After configuration, the server can be operated by the command:
/sbin/service pki-ca start | stop | restart
```

**Important**

For configuration purposes, a Firefox browser **must** be used to configure all PKI subsystems.

As an administrator, launch a Firefox browser using the profile manager:

```
[root@server]# /usr/bin/firefox -ProfileManager -no-remote
```

Create a new profile entitled **server**:

- Click **Next>**
- Enter new profile name: **server**
- Click **Finish**

Open the URL displayed in `/var/log/pki-ca-install.log` in the Firefox browser.

Within the Firefox browser, click on "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

Traverse the CA configuration panels:

- **Welcome** panel:
  - Click **Next>**
- **Key Store** panel:
  - Click on 'Login' for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the 'Security Module Token Password'
Click Next>
- Back in the **Key Store** panel, select the 'NHSM6000-OCS' Token
- Click Next>

**Security Domain** panel:
- Accept the defaults
- Click Next>

**Subsystem Type** panel:
- Accept the defaults
- Click Next>

**PKI Hierarchy** panel:
- Accept the defaults
- Click Next>

**Internal Database** panel:
- 'Host' -- `server.example.com`
- 'Port' -- **636** (SSL port)
- Check the 'SSL' box
- Provide the 'Bind Password' of the 'Directory Manager' (value which you gave while creating a Red Hat Directory Server instance)
- Click Next>

**Key Pairs** panel:
- Accept the defaults.
- Select the 'Use the default key size (2048 bits).' radio button
- Click Next>

**Subject Names** panel:
- Accept the defaults
- Click Next>

**Requests and Certificates** panel:
- Click on 'Apply'
- Once the screen redraws, click on **Next**>

---

**NOTE**

It is not possible to export keys and certificates stored on an HSM to a `.p12` file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

**Export Keys and Certificates** panel:
- Accept the defaults.
- Click Next>

**Import CA's Certificate Chain** panel:
- Click OK for the prompt 'You will now be asked to import and trust the Certificate Chain from the CA. Please do so.'
- Select the check-boxes for:
  - Check 'Trust this CA to identify web sites'
- Check 'Trust this CA to identify email users'
- Check 'Trust this CA to identify software developers'
- Click OK to dismiss this dialog box

Administrator panel:
- UID: admin
- Name: CA Administrator of Instance pki-ca
- Email: pki-ca-admin@example.com
- Password:
- Password (Again):
- Click Next>

Import Administrator's Certificate panel:
- Click OK on the alert which says 'Your personal certificate has been installed. You should keep a backup copy of this certificate.'
- Click Next>

Done panel:
- Restart the CA by running /sbin/service pki-ca restart

Enabling FIPS Mode

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, "Using Hardware Security Modules with Subsystems" and the vendor documentation.
2. Install and configure the CA instance.
3. Stop the CA instance. The instance must be stopped to protect the information stored in its security databases.

   service pki-ca stop

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the CA's security database directory.

   cd /var/lib/pki-ca/alias

   b. Using certutil, create a request for a new SSL server certificate.

   certutil -d . -R -s "CN=ca.example.com,OU=pki-ca,O=Example Domain pki-ca" -o sslfips.req -h "NSS Certificate DB" -a

   c. Restart the CA.

   service pki-ca start

   d. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

   e. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

   f. Copy the base 64-encoded certificate on the approval page and save it to a file, such as
sslfips.cert.
g. Stop the CA again.
```
service pki-ca stop
```
h. Check the CA's certificate database to see if an SSL server certificate is already listed.
```
certutil -d /var/lib/pki-ca/alias -L
```
i. If the certificate exists, then delete it.
```
certutil -d /var/lib/pki-ca/alias -D -n "ServerCert nickname"
```
j. Import the new SSL server certificate.
```
certutil -d /var/lib/pki-ca/alias -A -t "u,u,u" -n "ServerCert ca.example.com - Example Domain pki-ca" -i sslfips.cert -a
```
k. Edit the `/var/lib/pki-ca/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert ca.example.com - Example Domain pki-ca`.
l. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.
```
vim /var/lib/pki-ca/conf/CS/cfg
```
m. In the `CS.cfg` file, add a line to verify signatures from the token. The value is the token name, which depends on the vendor and version of the HSM. For example, for a NetHSM token:
```
ca.requestVerify.token=NHSM6000-0CS
```
n. Edit the `server.xml` file to enable FIPS mode for each SSL-enabled connector. Set `strictCiphers` to true and add or set `ssl3` to false.
```
vim /var/lib/pki-ca/conf/server.xml
```
```
<Connector name="Agent" port="9443" maxHttpHeaderSize="8192"
   ... ...
   sslOptions="ssl2=false,ssl3=false,tlsv1=true"
   strictCiphers="true"
   ...>
```
o. Enable FIPS mode in the NSS software database.
```
modutil -dbdir /var/lib/pki-ca/alias -fips true
```
p. Verify that FIPS mode has been enabled. The command will return the current FIPS status.
modutil -dbdir /var/lib/pki-ca/alias modutil -dbdir . -chkfips true
FIPS mode enabled.

q. Start the CA.

service pki-ca start

B.2.1.5.5. Directory Server SSL Server Authentication Configuration (Trusted)
Create a certificate signing request (csr) for 'Server-Cert-ldap' (ensure 'cn=server.example.com' [FQDN] with the same nickname 'Server-Cert-ldap'):

[root@server]# cd /etc/dirsrv/slapd-server/

[root@server]# /usr/bin/certutil -R -s "cn=server.example.com" -o Server-Cert-ldap.req -d . -n "Server-Cert-ldap" -a
Enter Password or Pin for "NSS Certificate DB":
A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

|************************************************************|

Finished. Press enter to continue:

Generating key. This may take a few moments...
Certificate request generated by Netscape certutil

Phone: (not specified)

Common Name: server.example.com
Email: (not specified)
Organization: (not specified)
State: (not specified)
Country: (not specified)

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBZjCBAIBADAnMSUwIwYDVQQDExtZWFWcGlKLMRzZGV2LnNqYy5yZWRoYXQg
Y29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDCiCR7kumfUozvkJYoD7w
1u8TFjwtpCpc1CA0UzGF7jqqHO4v1asLbJlTM0EFO6pqayQbi0ThzmzNcZueSP+
MEMgKq1pqo7rsGt9JBaYSJCE8sbCLih0oOieeSrSp1JIMuXY4a3mnZkkZrhiP
kmEsRY07ahftf61vbHM/rdwIDAQABoAAwDQYJKoZIhvcNAQEFBQADgYEAuUMC4r1Y
8qBmf9/40tY4n8dQ16460BMcgn5yQqTJ8fN0zZC1gydZo6P4ibm8txMBVCW/sVw
suDGFmSc+APufQQUUbG7c4I1Z2RO6Ynhy8SpPwavsVU8W++D25Yjfy6wHg8FYAgI
0YRTGzapYwo83mv5Pd96JYQXTWtqXS2SJUO=
-----END NEW CERTIFICATE REQUEST-----

As an administrator, launch a Firefox browser using the profile manager:

[root@server]# /usr/bin/firefox -ProfileManager -no-remote

Select the profile entitled server that was utilized for configuration of the CA in Section B.2.1.5.4, “CA Instance Configuration”.

Select the URL that points to the Secure Admin Port of the previously configured CA and open the browser to a new tab called CA Services.

Note

This URL can be identified by running /sbin/service pki-ca status.

Select the URL called SSL End Users Services to open a new tab called CA End-Entity within the browser.

Select the profile entitled Manual Server Certificate Enrollment, and perform the following procedures:

» Cut the certificate request contained in the file called /etc/dirsrv/slapd-server/Server-Cert-ldap.req and paste it into the area called ‘Certificate Request’ on the form:
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBZjCB0AIBADAnMSUwIwYDVQQDExtZWFW0cGl1LmRzZGV2LnNqYy5yZWRoYXQu
Y29tMIGfMA0GCSqGSIb3DQEBADQUAA4GNADCBiQKBgQDCiCR7kumfiU0zvKVYoD7w
1u0TFjwTCpc1CHa0UzGF7jqqhQ4v1asLBj1TMmE06pqayQbIt0DhzmzhNcZueSP+
1EMGKq1p0q7sGt9JbaRYSJCE8sbCLih0oC01eeSrSp1JIMuXY4a3mnZkKZrhiP
kmUsRY07ahtf61vbHM/rwdIDAQABoAAwDQYJKoZIhvcNAQEFBQADgYEauUMC4rY
8qBmf9//40ty4n8dQ16460BMcgn5yQqT0fN0zZg1gydZo6P41m8txMBVCw/sVw
sUpGFMsc+APufeQUUUbG7c4I1Z2RD8YnhysBpMwasvU8W++D25YjfGwJ8FyAgI
OYRTGzapYwo3mSPd9GJYXWtQzXS2SU0==
-----END NEW CERTIFICATE REQUEST-----

Note

The certificate request listed above is only an EXAMPLE of what the certificate request will look like. Do NOT use this EXAMPLE certificate request as a part of this test!

» Press the Submit button

» A page entitled Certificate Profile should appear noting a request ID.

Within the browser, click on the tab entitled CA Services, and select the URL called Agent Services to open a new tab called CA Agent.

Note

At this point, a certificate may be presented in a dialog box entitled 'User Identification Request' in order for the agent to confirm his/her identity. Select the proper certificate to present as identification, and accept this by clicking the OK button.

Select the browser tab labeled CA Agent, and perform the following tasks:

» Click on List Requests

» Click on Find

» Click on the request id # of the entry labeled CN=server.example.com

» Scroll to the bottom of this request, and click on the submit button to approve this request.

» Scroll to the bottom of the approved request and highlight the Base-64 Encoded Certificate:
Note

The certificate listed above is only an EXAMPLE of what the certificate will look like. Do NOT use this EXAMPLE certificate as a part of this test!

Copy this highlighted 'Base-64 Encoded' certificate and paste it into a file called /etc/dirsrv/slapd-server/ldap-server-b64.crt.

Obtain the CA's signing certificate into a base-64-encoded pem format:

```
[root@server]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias/ -n "caSigningCert cert-pki-ca" -a > /etc/dirsrv/slapd-server/casigning-b64.crt
```

Stop the CA instance before modifying its NSS security databases:

```
[root@server]# /sbin/service pki-ca stop
Stopping pki-ca: .. [ OK ]
```

Stop the Red Hat Directory Server instance before modifying its NSS security databases:

```
[root@server]# /sbin/service dirsrv stop
Shutting down dirsrv: .. [ OK ]
```

Delete the temporary 'Server-Cert-ldap' certificate issued earlier in Section B.2.1.5.2, "Directory Server SSL Server Authentication Configuration (Temporary)".
Import the new trusted 'ldap-server-b64.crt' (issued by pki-ca) into the Red Hat Directory Server NSS security database:

```
[root@server]# /usr/bin/certutil -A -i ldap-server-b64.crt -d /etc/dirsrv/slapd-server -n "Server-Cert-ldap" -t "u,u,u"
```

Delete the temporary LDAP self-signed CA certificate issued earlier in Section B.2.1.5.2, "Directory Server SSL Server Authentication Configuration (Temporary)":

```
[root@server]# /usr/bin/certutil -D -d /etc/dirsrv/slapd-server -n "LDAP Self Signed CA certificate"
```

Import the trusted CA signing certificate into the Red Hat Directory Server NSS security database and trust it:
[root@server]# /usr/bin/certutil -A -i casigning-b64.crt -d /etc/dirsrv/slapd-server -n "caSigningCert cert-pki-ca" -t "CTu,Cu,Cu"

[root@server]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server

<table>
<thead>
<tr>
<th>Certificate Nickname</th>
<th>Trust Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSL,S/MIME,JAR/XPI</td>
</tr>
<tr>
<td>Server-Cert-ldap</td>
<td>u,u,u</td>
</tr>
<tr>
<td>caSigningCert cert-pki-ca</td>
<td>CT,C,C</td>
</tr>
</tbody>
</table>

Delete the temporary LDAP self-signed CA certificate from the pki-ca's NSS security database:

[root@server]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias

<table>
<thead>
<tr>
<th>Certificate Nickname</th>
<th>Trust Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSL,S/MIME,JAR/XPI</td>
</tr>
<tr>
<td>caSigningCert cert-pki-ca</td>
<td>CTu,Cu,Cu</td>
</tr>
<tr>
<td>Server-Cert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
<tr>
<td>auditSigningCert cert-pki-ca</td>
<td>u,u,Pu</td>
</tr>
<tr>
<td>LDAP Self Signed CA certificate</td>
<td>CT,C,C</td>
</tr>
<tr>
<td>ocspSigningCert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
<tr>
<td>subsystemCert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
</tbody>
</table>

[root@server]# /usr/bin/certutil -D -d /var/lib/pki-ca/alias -n "LDAP Self Signed CA certificate"

[root@server]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias

<table>
<thead>
<tr>
<th>Certificate Nickname</th>
<th>Trust Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSL,S/MIME,JAR/XPI</td>
</tr>
<tr>
<td>caSigningCert cert-pki-ca</td>
<td>CTu,Cu,Cu</td>
</tr>
<tr>
<td>Server-Cert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
<tr>
<td>auditSigningCert cert-pki-ca</td>
<td>u,u,Pu</td>
</tr>
<tr>
<td>ocspSigningCert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
<tr>
<td>subsystemCert cert-pki-ca</td>
<td>u,u,u</td>
</tr>
</tbody>
</table>

Start the Red Hat Directory Server instance (on a secure port):

**NOTE**

The PIN being prompted for below is merely referring to the password used throughout this test.
B.2.1.6. DRM Instance Creation and Configuration

B.2.1.6.1. DRM Instance Creation
Use the `pkicreate` tool to create an Red Hat Certificate System 8.1 instance of a DRM.

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the 'pkicreate' tool as shown below:
Optional Reference: Section 6.1, "About pkicreate"

**B.2.1.6.2. Hardware Security Module DRM Configuration**

Follow instructions in the following procedure (substituting the name 'pki-kra' for 'pki-ca'):

Procedural Reference: Section 8.1.2.3, "Installing External Tokens and Unsupported HSM"

To enable FIPS mode, open the security UI:

```
/opt/nfast/bin/ksafe
```

In the 'Security World' tab, make sure that "Strict FIPS 140-2 Level III" is set to 'yes'.

For configuring FIPS mode, see the hardware vendor documentation.

**B.2.1.6.3. DRM Instance Configuration**

**NOTE**

This section involves configuring a DRM instance using SSL with Directory Server

Since **pkicreate** has been executed in Section B.2.1.6.1, "DRM Instance Creation", obtain the URL to configure the DRM by doing a tail `/var/log/pki-kra-install.log`:

```
[root@server]# tail /var/log/pki-kra-install.log
[2011-03-27 07:46:37] [debug] Restorecon /etc/pki-kra
[2011-03-27 07:46:37] [debug] Restorecon file context for /etc/pki-kra/tomcat5.conf
[2011-03-27 07:46:37] [debug] Setting 'pki-kra' runlevel to '-'
[2011-03-27 07:46:37] [debug] Setting 'pki-kra' start priority to '82'
[2011-03-27 07:46:37] [debug] Setting 'pki-kra' stop priority to '18'
[2011-03-27 07:46:37] [debug] Registered 'pki-kra' with '/sbin/chkconfig'.
[2011-03-27 07:46:47] [log] After configuration, the server can be operated by the command:
/sbin/service pki-kra start | stop | restart
```
As an administrator, launch a Firefox browser using the profile manager:

```
[root@server]# /usr/bin/firefox -ProfileManager -no-remote
```

Select the profile entitled `server` that was utilized for configuration of the CA in [Section B.2.1.5.4, "CA Instance Configuration".](#)

Open the URL displayed in `/var/log/pki-kra-install.log` in the Firefox browser.

Within the Firefox browser, click on "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

Traverse the DRM configuration panels:

- **Welcome** panel:
  - Click Next>

- **Key Store** panel:
  - Click on ‘Login’ for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the ‘Security Module Token Password’
  - Click Next>
  - Back in the Key Store panel, select the ‘NHSM6000-OCS’ Token
  - Click Next>

- **Security Domain** panel:
  - Select ‘Join an Existing Security Domain’ and enter the URL to an existing security domain. This URL can be identified by running `/sbin/service pki-ca status`.
  - Click Next>

- **Display Certificate Chain** panel:
  - Click Next>

- **Security Domain Login** panel:
  - Supply the admin uid name and password for the CA so that it can be properly accessed.
  - Click Login

- **Subsystem Type** panel:
  - Accept the defaults.
  - Click Next>

- **Internal Database** panel:
  - 'Host' -- `server.example.com`
  - 'Port' -- 636 (SSL port)
  - Check the ‘SSL’ box
  - Provide the ‘Bind Password’ of the 'Directory Manager' (value which you gave while creating a Red Hat Directory Server instance)
  - Click Next>

- **Key Pairs** panel:
  - Accept the defaults.
  - Select the ‘Use the default key size (2048 bits).’ radio button
  - Click Next>

- **Subject Names** panel:
Accept the defaults
Click Next>

Requests and Certificates panel:
- Click on 'Apply'
- Once the screen redraws, click on Next>

**NOTE**
It is not possible to export keys and certificates stored on an HSM to a .p12 file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

Export Keys and Certificates panel:
- Accept the defaults.
- Click Next>

Administrator panel:
- UID: admin
- Name: KRA Administrator of Instance pki-kra
- Email: pki-kra-admin@example.com
- Password:
- Password (Again):
- Click Next>

Import Administrator's Certificate panel:
- Click OK on the alert which says 'Your personal certificate has been installed. You should keep a backup copy of this certificate.'
- Click Next>

Done panel:
- Stop the DRM by running /sbin/service pki-kra stop.

**Enabling FIPS Mode**

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, "Using Hardware Security Modules with Subsystems" and the vendor documentation.
2. Install and configure the instance.
3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

```
    service instance_name stop
```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance's security database directory.

```
    cd /var/lib/instance_name/alias
```
   b. Using certutil, create a request for a new SSL server certificate.
c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.
d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.
e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as sslfips.cert.
f. Check the instance's certificate database to see if an SSL server certificate is already listed.

g. If the certificate exists, then delete it.

h. Import the new SSL server certificate.

i. Edit the /var/lib/instance_name/conf/serverCertNick.conf file to contain the nickname of the new certificate, such as ServerCert server.example.com - Example Domain instance_name.

j. Edit the CS.cfg file to replace both references to the SSL server certificate nickname.

k. Edit the server.xml file to enable FIPS mode for each SSL-enabled connector. Set strictCiphers to true and add or set ssl3 to false. For example:

l. Enable FIPS mode in the NSS software database.

m. Verify that FIPS mode has been enabled. The command will return the current FIPS status.
modutil -dbdir /var/lib/instance_name/alias modutil -dbdir . -chkfips true

FIPS mode enabled.

n. Start the instance.

```
service instance_name start
```

**B.2.1.7. OCSP Instance Creation and Configuration**

**B.2.1.7.1. OCSP Instance Creation**

Use the `pkicreate` tool to create a Red Hat Certificate System 8.1 instance of an OCSP Responder.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the 'pkicreate' tool as shown below:

```
[root@server]# pkicreate -pki_instance_root=/var/lib       
          -pki_instance_name=pki-ocsp               
          -subsystem_type=ocsp                 
          -agent_secure_port=11443           
          -ee_secure_port=11444            
          -admin_secure_port=11445           
          -unsecure_port=11180              
          -tomcat_server_port=11701         
          -user=pkiuser                    
          -group=pkiuser                   
          -audit_group=pkiaudit             
          -redirect conf=/etc/pki-ocsp     
          -redirect logs=/var/log/pki-ocsp  
          -verbose
```

Optional Reference: [Section 6.1, “About pkicreate”](#)

**B.2.1.7.2. Hardware Security Module OCSP Configuration**

Follow instructions in the following procedural reference (substituting the name 'pki-ocsp' for 'pki-ca'):

Procedural Reference: [Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”](#)

**B.2.1.7.3. OCSP Instance Configuration**

**NOTE**

This section involves configuring an OCSP instance using SSL with Directory Server

Since `pkicreate` has been executed in [Section B.2.1.7.1, “OCSP Instance Creation”](#), obtain the URL to configure the OCSP Responder by doing a `tail /var/log/pki-ocsp-install.log`:
As an administrator, launch a Firefox browser using the profile manager:

```
[root@server]# /usr/bin/firefox -ProfileManager -no-remote
```

Select the profile entitled `server` that was utilized for configuration of the CA in Section B.2.1.5.4, "CA Instance Configuration".

Open the URL displayed in `/var/log/pki-ocsp-install.log` in the Firefox browser.

Within the Firefox browser, click on "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> " Confirm security Exception".

 Traverse the OCSP configuration panels:

- **Welcome** panel:
  - Click Next>

- **Key Store** panel:
  - Click on 'Login' for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the 'Security Module Token Password'
  - Click Next>
  - Back in the **Key Store** panel, select the 'NHSM6000-OCS' Token
  - Click Next>

- **Security Domain** panel:
  - Select 'Join an Existing Security Domain' and enter the URL to an existing security domain. This URL can be identified by running `/sbin/service pki-ca status`.
  - Click Next>

- **Display Certificate Chain** panel:
  - Click Next>

- **Security Domain Login** panel:
  - Supply the admin uid name and password for the CA so that it can be properly accessed.
  - Click Login

- **Subsystem Type** panel:
  - Accept the defaults.
  - Click Next>
Internal Database panel:
- 'Host' -- server.example.com
- 'Port' -- 636 (SSL port)
- Check the ‘SSL’ box
- Provide the 'Bind Password' of the 'Directory Manager' (value which you gave while creating a Red Hat Directory Server instance)
- Click Next>

Key Pairs panel:
- Accept the defaults.
- Select the 'Use the default key size (2048 bits).’ radio button
- Click Next>

Subject Names panel:
- Accept the defaults
- Click Next>

Requests and Certificates panel:
- Click on 'Apply'
- Once the screen redraws, click on Next>

NOTE
It is not possible to export keys and certificates stored on an HSM to a .p12 file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

Export Keys and Certificates panel:
- Accept the defaults.
- Click Next>

Administrator panel:
- UID: admin
- Name: OCSP Administrator of Instance pki-ocsp
- Email: pki-ocsp-admin@example.com
- Password:
- Password (Again):
- Click Next>

Import Administrator's Certificate panel:
- Click OK on the alert which says ‘Your personal certificate has been installed. You should keep a backup copy of this certificate.’
- Click Next>

Done panel:
- Stop the OCSP Responder by running /sbin/service pki-ocsp stop.

Enabling FIPS Mode

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, “Using Hardware Security Modules with..."
Subsystems" and the vendor documentation.

2. Install and configure the instance.

3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

   ```bash
   service instance_name stop
   ```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance's security database directory.

   ```bash
   cd /var/lib/instance_name/alias
   ```

   b. Using `certutil`, create a request for a new SSL server certificate.

   ```bash
   certutil -d . -R -s "CN=server.example.com,OU=instance_name,O=Example Domain instance_name" -o sslfips.req -h "NSS Certificate DB" -a
   ```

c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as `sslfips.cert`.

f. Check the instance's certificate database to see if an SSL server certificate is already listed.

   ```bash
   certutil -d /var/lib/instance_name/alias -L
   ```

g. If the certificate exists, then delete it.

   ```bash
   certutil -d /var/lib/instance_name/alias -D -n "ServerCert nickname"
   ```

h. Import the new SSL server certificate.

   ```bash
   certutil -d /var/lib/instance_name/alias -A -t "u,u,u" -n "ServerCert server.example.com - Example Domain instance_name" -i sslfips.cert -a
   ```

i. Edit the `/var/lib/instance_name/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert server.example.com - Example Domain instance_name`.

j. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.

   ```bash
   vim /var/lib/instance_name/conf/CS/cfg
   ```

   ```bash
   type.cert.sslserver.nickname= ServerCert server.example.com - Example Domain instance_name
   type.sslserver.nickname= ServerCert server.example.com - Example Domain instance_name
   ```

k. Edit the `server.xml` file to enable FIPS mode for each SSL-enabled connector. Set `strictCiphers` to true and add or set `ssl3` to false. For example:
B.2.1.8. Post Configuration Instructions

B.2.1.8.1. Agent, Administrator, and Audit User Creation

The setup process only creates a single administrative user account. To manage a PKI environment in real life requires a larger number of administrators, agents, and auditors which are created after the instance is set up.

Log into server.example.com as an administrator, and launch the Console to create the users:

```
[root@server]# /usr/bin/pkiconsole https://server.example.com:9445/ca
```

These users are to be created as appropriate for relevant CS subsystems including CA, DRM, and OCSP.

- **For Admin users:**
  - adminV (this user will have a valid certificate throughout testing)
  - adminR (this user will have a revoked certificate)
  - adminE (this user will have an expired certificate)
  - adminTCA (this user's certificate will be signed by a trusted CA)
  - adminC (initially this user will be assigned to the administrator group; during the testing process this user will be assigned to a different group)
  - adminUTCA (this user's certificate will be issued by an untrusted CA) [Note: This requires a second CA setup!]

- **For Agent users:**
  - agentV (this user will have a valid certificate throughout testing)
  - agentR (the status of this user's certificate is Revoked)
- agentE (this user's certificate is in the expired state)
- adminTCA (this user's certificate will be signed by a trusted CA)
- agentTCA (this user's certificate will be issued by a trusted CA)
- agentUTCA (this user's certificate will be issued by an untrusted CA) [Note: This requires a second CA setup!]

» For Audit users:
  - auditV (this user will have a valid certificate throughout testing)

» To Revoke Certificates

» To Achieve Expired Certificates

### Expired Certificates

From CA EE pages, submit enrollment requests for all of these users.

From the CA Agent port, approve these requests. When approving expired certs, remember to change ‘notAfter’ to point to 1-day.

Optional Reference: [Managing Users and Groups for a CA, OCSP, DRM, or TKS](#)

---

**B.2.1.8.2. Import of Certificate Chain into Browser**

Any browser which will be used to access agent or admin services pages must have the client certificate and the CA certificate chain imported into its trust store.

### Note

Since this test uses a common browser profile for its CA, DRM, and OCSP instances, the following steps will **NOT** be necessary as the certificate chain was imported during creation of the CA as documented in Section B.2.1.5.4, “CA Instance Configuration”. However, if different browser profiles had been used for its DRM and OCSP instances, the following steps would have been necessary.

To import the certificate chain into the browser, perform the following steps:

» Select the URL that points to the Secure End-Entity Port of the previously configured CA and open the browser to this URL.

### Note

This URL can be identified by running /sbin/service pki-ca status.

» Click on the tab labeled **Retrieval**

» Select the URL called **Import CA Certificate Chain**

» Select the radio button for **Import the CA certificate chain into your browser**

» Click on the **submit** button.

» Click OK for the prompt ‘You will now be asked to import and trust the Certificate Chain from the CA. Please do so:’

» Select the check-boxes for:
Check 'Trust this CA to identify web sites'
Check 'Trust this CA to identify email users'
Check 'Trust this CA to identify software developers'
Click OK to dismiss this dialog box

Optional Reference: [Downloading CA Certificates and Certificate Chains](#)

### B.2.1.8.3. Sudo Permissions for PKI Services

Granting sudo permissions for Certificate System and Directory Server services to PKI administrators makes it easier to manage the instances.

To grant these sudo permissions, execute the following command:

```
[root@server]# /usr/sbin/visudo
```

Add the following lines to the `/etc/sudoers` file:

```
# For Directory Server services
%pkiadmin ALL = PASSWD: /sbin/service dirsrv *
%pkiadmin ALL = PASSWD: /sbin/service dirsrv-admin *

# For PKI instance management
%pkiadmin ALL = PASSWD: /usr/bin/pkicreate *
%pkiadmin ALL = PASSWD: /usr/bin/pkiremove *

# For PKI instance services
%pkiadmin ALL = PASSWD: /sbin/service pki-ca *
%pkiadmin ALL = PASSWD: /sbin/service pki-kra *
%pkiadmin ALL = PASSWD: /sbin/service pki-ocsp *
```

Optional Reference: [Resetting sudo Permissions for Certificate System Services](#)

### B.2.1.8.4. Backup and Restore Procedures

A reliable backup and restore strategy is vital for disaster and data recovery.

**Note**

For the purposes of this test, no backup and recovery mechanism will be utilized.

Optional Reference: [Backing up and Restoring Certificate System](#)

### B.2.1.8.5. Signed Audit

This can be done when an instance is first created by using the -audit_group option, but it can also be done post-installation.

The following procedural reference can be followed to enable signed auditing:

Procedural Reference: [Managing Signed Audit Logs](#)

### B.2.1.8.6. Revocation Checking and OCSP Responders
There are two methods for checking whether a certificate is active (and not revoked): CRL checking and OCSP services. Checking the revocation status helps ensure that a revoked certificate isn't wrongly accepted just because it is within its validity period.

The following procedural reference can be followed to enable revocation checking:

Procedural Reference: Enabling Revocation Checking

Similarly, the following procedural reference can be followed to establish an OCSP responder:


**B.2.1.8.7. SSL Session Timeouts**

Using session timeouts prevents unauthorized users from accessing an open, but otherwise inactive, session with a subsystem.

**Note**

For the purposes of this test, the default SSL Session Timeouts for the CA, DRM, and OCSP instances should be sufficient.

To alter the default behavior of this feature, follow the procedural reference below applying it to the CA, DRM, and OCSP instances:

Procedural Reference: Setting SSL Session Timeouts

**B.2.1.8.8. Self-Tests Execution**

Self-tests are subsystem type-specific diagnostics. These are usually run when the server starts, but they can also be run on demand. Additionally, self-test can be customized by creating new self-test plugins or to accommodate changes to the subsystem configuration like new certificates or changed certificate nicknames.

**Note**

For the purposes of this test, all required instance-specific self-tests were automatically registered and configured when each instance was installed.

Optional Reference: Running Self-Tests

**B.2.1.8.9. Removal of password.conf**

The password.conf file stores system passwords in plaintext.

Remove the `/var/lib/pki-ca/password.conf` file to ‘prompt for passwords’, restart the pki-ca server, and provide the passwords prompted. The following example is for the CA.

**Important**

Similar changes need to be done for all other PKI subsystem instances!
Apply this same process to both the DRM and OCSP instances.

Optional Reference: System Passwords

**B.2.1.8.10. Removal of Unused Interfaces from web.xml**

Several legacy interfaces are included in the CA's web.xml. While these can be used for custom or legacy applications to connect to the CA, for secure environments, these deprecated interfaces should be removed from the CA configuration to prevent unauthorized access.

To perform this step, please refer to the following procedural reference:

Procedural Reference: Removing Unused Interfaces from web.xml

**B.2.1.8.11. Downloading dumpasn1 tool**

Download dumpasn1 from the url below; compile it with gcc; then copy it to /usr/local/bin:
B.2.1.8.12. Disable User/Group "multiroles" Support for Java Subsystems

By the default, the Certificate System Console for Java Subsystems supports the notion of allowing given users the ability to reside within more than one group at a time. Ultimately this property makes it possible for a single user to assume more than one distinct role. This default behavior is fine for normal processing. The CC setup requires that we configure the systems to not allow this multirole behavior.

For CA, KRA, TKS, and OCSP subsystems:

1. Stop the server:
   ```bash
   service pki-ca stop
   ```

2. Open the `CS.cfg` file:
   ```bash
   vim /var/lib/pki-ca/conf/CS.cfg
   ```

3. Modify the current setting of `multiroles.enable=true` and change it to `multiroles.enable=false`.

4. Restart the server:
   ```bash
   service pki-ca restart
   ```

B.2.1.8.13. Configuring POSIX System ACLs

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

Steps 3 and 4 only need to be performed once. The rest of the procedure must be performed separately for every instance installed on the machine.

1. Log into the system as root.
2. Stop all Certificate System subsystems.
   ```bash
   service instance_name stop
   ```

3. Make sure that the `acl` package is installed on the system.
   ```bash
   # rpm -q acl
   acl-2.2.39-6.el5
   ```

4. Four filesystem paths must be configured to use the `acl` package:
   - `/`
   - `/etc` (including `/etc/init.d` and `/etc/sysconfig`)
   - `/usr` (including `/usr/bin` and `/usr/share`)
   - `/var` (including `/var/lock` and `/var/run`)
   
   First, check to see where the filesystems are mounted and whether the `acl` option is used:
In this example, /, /etc, and /var are mounted on /dev/sda1, while /usr is mounted on /dev/sdb1.

There are two ways to apply the acl option to the filesystems:

» Using tune2fs:

```bash
# /sbin/tune2fs -o +acl /dev/sda1
# /sbin/tune2fs -o +acl /dev/sdb1
```

» Editing the /etc/fstab file:

```bash
# vim /etc/fstab
LABEL=/           /             ext3    defaults,acl 1 1
LABEL=/usr        /usr          ext2    defaults,acl 1 2
tmpfs            /dev/shm      tmpfs   defaults        0 0
devpts           /dev/pts      devpts  gid=5,mode=620 0 0
sysfs            /sys          sysfs   defaults        0 0
proc             /proc         proc    defaults        0 0
LABEL=SWAP-sdb2  swap          swap    defaults        0 0
LABEL=SWAP-sda2  swap          swap    defaults        0 0
```

Then, remount the filesystems:

```bash
# mount -o remount /
# mount -o remount /usr
```

Confirm that the acl option has been applied to the filesystems:

```bash
# /sbin/dumpe2fs /dev/sda1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:    user_xattr acl

# /sbin/dumpe2fs /dev/sdb1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:    acl

# mount | grep acl
/dev/sda1 on / type ext3 (rw,acl)
/dev/sdb1 on /usr type ext2 (rw,acl)
```

5. **Stop the instance.**

```
    service instance_name stop
```

6. Set the group readability to the pkiadmin group for the instance's directories and files.
7. Apply execute (x) ACL permissions on all directories:

```bash
# setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/instance_name
```

8. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:

```bash
# setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/instance_name/logs/signedAudit
```

9. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

```bash
# setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/instance_name/logs/signedAudit
```

10. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:

```bash
# find -L /var/lib/instance_name/logs/signedAudit -type d -exec setfacl -L -n -m g:pkiaudit:rx,d:g:pkiaudit:rx {} \;
```

11. Start the instance.

```bash
service instance_name start
```

12. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:
B.2.2. TMS Common Criteria Setup Procedures

These are the instructions for establishing a Common Criteria test environment for a TMS (Token Management System). Throughout this document, reference links to the Red Hat Certificate System guidance documents have been provided for users as either a well-documented stand-alone procedure, or as an optional source of additional information, and will be clearly denoted as to their purpose.

NOTE

For the purposes of this test, Red Hat Directory Server 8.2 and Red Hat Certificate System subsystems are installed/configured on three different machines:

- **Machine-1**: Directory Server, Certificate Authority subsystem
- **Machine-2**: Data Recovery Management subsystem, Online Certificate Status Protocol Responder, Token Key Service subsystem
- **Machine-3**: Token Processing System subsystem

B.2.2.1. Prerequisites: RHEL 5.6 Operating System Installation and Configuration

B.2.2.1.1. TMS Inventory

This test requires three 64-bit x86_64 machines each containing a 64-bit version of the Red Hat Enterprise Linux server 5.6 operating system for the x86_64 architecture. The 3 machines, which for the purposes of this document will be referred to with hostnames of server1.example.com, server2.example.com, server3.example.com should be registered with Red Hat Network (RHN) so
that each machine may be fully updated with the latest packages for this version of the operating system. For example, once the operating system has been installed, this can be accomplished by becoming the root user, and executing the following command:

```
[root@server1]# yum -y update
```

**NOTE**

The command should be executed on all three machines.

In the command, `[root@server1]#` is merely an example command-line prompt for **Machine-1**. This convention is used throughout this document. A similar convention is used when referring to **Machine-2** and **Machine-3**.

Additionally, the user will need to have an nCipher nShield hardware security module (HSM) attached to this machine for compliance with FIPS 140-2 (Level 3). Details for installation and configuration of this device are provided in Section B.2.2.1.2, “Hardware Security Module Installation”.

Finally, for the purposes of Common Criteria certification of a TMS, the user will need to have the appropriate Red Hat subscriptions to download and install the following applications to all the three machines (server1.example.com,server2.example.com,server3.example.com) -- [detailed download, installation, and configuration instructions are provided below in various sections of this document]:

- Apache 2.2.3
- Tomcat 5.5.23
- Red Hat Directory Server 8.2
- Red Hat Certificate System subsystems:
  - Certificate Authority (CA)
  - Data Recovery Management (DRM) - also called a Key Recovery Authority (KRA)
  - Token Key Service (TKS)
  - Token Processing System (TPS)
  - Enterprise Security Client (ESC)
  - Online Certificate Status Protocol (OCSP) Responder
- Red Hat Certificate System Console
- Firefox 10 and later
- ESC (Enterprise Security Client)
- Open JDK/JRE 1.6
- JSS 4.6
- NSS 3.12
- nss-tools 3.12
- mozldap-tools

**B.2.2.1.2. Hardware Security Module Installation**

**NOTE**

HSM installation should be performed on **Machine-1**, **Machine-2**, and **Machine-3**.
If the nCipher nShield is not setup, insert the accompanying DVD, or mount the ISO and cd into the “Documents” folder. For example, assuming that the ISO is available in /tmp:

```
[root@server1]# mkdir /mnt/iso
[root@server1]# cd /mnt/iso/document
[root@server1]# mount -o loop /tmp/nCSS_linux64_user_11_40.iso /mnt/iso/
```

Follow the vendor's instructions to setup the nCipher nShield.

Once the nCipher nShield has been installed, run the command below to ensure that the connection status is OK (by using grep):

```
[root@server1]# /opt/nfast/bin/enquiry | grep -i status
connection status    OK
```

Optional Reference: Section 8.1, “Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates”

To enable FIPS mode, open the security UI:

```
/opt/nfast/bin/ksafe
```

In the Security World tab, make sure that “Strict FIPS 140-2 Level III” is set to yes.

For more information on configuring FIPS mode, see the hardware vendor documentation.

**B.2.2.1.3. SELinux Enforcement**

**NOTE**

SELinux Enforcement should be performed on Machine-1, Machine-2, and Machine-3.

SELinux is a collection of mandatory access control rules which are enforced across a system to restrict unauthorized access and tampering.

Optional Reference: SELinux section of the Red Hat Enterprise Linux Deployment Guide

Red Hat recommends running Certificate System with SELinux in enforcing mode, to make the most of the security policies.
The `/usr/sbin/getenforce` command will give the status of SELinux. If running `/usr/sbin/getenforce` returns a result of either Disabled or Permissive, then it needs to be enabled to Enforcing. This may be accomplished by becoming the root user, editing the file `/etc/sysconfig/selinux` and configuring SELINUX=Enforcing, and re-booting the system.

```
[root@server1]# /usr/sbin/getenforce
Enforcing
```

```
[root@server1]# /usr/sbin/sestatus
SELinux status: enabled
SELinuxfs mount: /selinux
Current mode: enforcing
Mode from config file: enforcing
Policy version: 21
Policy from config file: targeted
```

Optional Reference: Section 5.3.6, “Enabling SELinux”

**B.2.2.1.4. Operating System User and Group Creation**

**NOTE**

Procedures in this section should be performed on Machine-1, Machine-2, and Machine-3.

This test requires that certain operating system users and groups exist. Login to the RHEL 5.6 machine as super user and ensure that both the UID and the gid of pkiuser are 17 by running by the following commands:

```
[root@server1]# grep pkiuser /etc/group
pkiuser:x:17:
```

```
[root@server1]# grep pkiuser /etc/passwd
pkiuser:x:17:17:Red Hat Certificate System:/usr/share/pki:/sbin/nologin
```

If the pkiuser group is not present, make certain that the appropriate tool packages are installed and recheck the pkiuser user and group:

```
[root@server1]# rpm -q setup
setup-2.5.58-7.el5
```

```
[root@server1]# rpm -q shadow-utils
shadow-utils-4.0.17-15.el5
```

If the pkiuser user or group are still not present:

```
[root@server1]# /usr/sbin/groupadd -g 17 -r pkiuser
```

```
[root@server1]# /usr/sbin/useradd -g pkiuser -d /usr/share/pki -s /sbin/nologin -c "Red Hat Certificate System" -u 17 -r pkiuser
```

If the pkiuser user or group exist but do not have a uid:gid of 17:17:
Create an administration group:

```
[root@server1]# /usr/sbin/groupadd -r pkiadmin
```

Create an auditor group:

```
[root@server1]# /usr/sbin/groupadd -r pkiaudit
```

**Note**

Linux distinguishes system users such as `pkiuser` (which are often referred to as daemons which have no real person associated with them) from standard users that comprise the PKI administrators, agents, and auditors which are associated with real people.

Optional Reference: [Users and Groups](#)

If they do not already exist, create standard user accounts for either `pkiadmin` or `pkiaudit`, but never allow any standard user access to both groups.

For example, to create an administrator called John Smith:

```
[root@server1]# /usr/sbin/useradd -g pkiadmin -d /home/jsmith -s /bin/bash -c
"Red Hat Certificate System Administrator" -m jsmith
```

For example, to create an auditor called Sally Jones:

```
[root@server1]# /usr/sbin/useradd -g pkiaudit -d /home/sjones -s /bin/bash -c
"Red Hat Certificate System Auditor" -m sjones
```

Optional Reference: [Section 5.3.7, “Setting up Operating System Users and Groups”](#)

**B.2.2.1.5. Operating System User and Group Association**

**NOTE**

Procedures in this section should be performed on **Machine-1**, **Machine-2**, and **Machine-3**.

Once all operating system groups have been established (see [Section B.2.2.1.4, “Operating System User and Group Creation”](#)), the pkiuser system user must be associated with both the pkiadmin and the pkiaudit groups:
```
[root@server1]# /usr/sbin/usermod -a -G pkiadmin pkiuser
[root@server1]# /usr/sbin/usermod -a -G pkiaudit pkiuser
```

**Important**

The pkiuser user is allowed to be associated with both the pkiadmin and the pkiaudit groups because pkiuser is a "system" user. Regular users are never allowed to be associated with both groups because no regular user is ever allowed to be both an administrator and an auditor.

If a pre-existing user is re-assigned to become an administrator, they can be associated with the pkiadmin group:

```
[root@server1]# grep mnelson /etc/group
mnelson:x:1099:
[root@server1]# grep mnelson /etc/passwd
mnelson:x:1099:1099:Mary Nelson:/home/mnelson:/bin/bash
[root@server1]# /usr/sbin/usermod -a -G pkiadmin mnelson
```

Similarly, if a pre-existing user is re-assigned to become an auditor, they can be associated with the pkiaudit group:

```
[root@server1]# grep blarson /etc/group
blarson:x:1257:
[root@server1]# grep blarson /etc/passwd
blarson:x:1257:1257:Bob Larson:/home/blarson:/bin/bash
[root@server1]# /usr/sbin/usermod -a -G pkiaudit blarson
```

Additionally, since this test utilizes an nCipher nShield HSM, both the pkiuser user and any individuals in charge of administration of this device must be associated with the nfast group:

```
[root@server1]# /usr/sbin/usermod -a -G nfast pkiuser
[root@server1]# /usr/sbin/usermod -a -G nfast jsmith
[root@server1]# /usr/sbin/usermod -a -G nfast mnelson
```

Once all associations have been generated, ensure that the groups for users pkiuser, jsmith, sjones, mnelson, and blarson are correct:
[root@server1]# groups pkiuser jsmith sjones mnelson blarson
  pkiuser : pkiuser pkiadmin pkiaudit nfast
  jsmith : pkiadmin nfast
  sjones : pkiaudit
  mnelson : mnelson pkiadmin nfast
  blarson : blarson pkiaudit

Optional Reference: Section 5.3.7.2.3, “Associating Existing User Accounts with PKI Groups”

B.2.2.2. Red Hat Certificate System 8.1 Subsystem Packages

B.2.2.2.1. Directory Server Subsystem Package Installation

NOTE

Directory Server ONLY needs to be installed on Machine-1 (server1.example.com).

Machine-1 (server1.example.com) has been registered with Red Hat Network (RHN) to contain the appropriate Red Hat subscriptions to download and install Red Hat Directory Server 8.2 (see Section B.2.2.1, “Prerequisites: RHEL 5.6 Operating System Installation and Configuration”).

NOTE

Check the /etc/yum.repos.d/ directory for the existence of either epel.repo or epel-testing.repo. If either file is present, make certain that ALL sections in both epel.repo and epel-testing.repo have been set to enabled=0 so that no EPEL packages accidentally override the officially desired RHEL RPMS.

Install the OpenLDAP tools:

[root@server1]# yum install openldap-clients

To install all of the Red Hat Directory Server 8.2 packages, perform the following command:

[root@server1]# yum install redhat-ds

Optional Reference: Section 5.3.3, “Installing Red Hat Directory Server”

B.2.2.2.2. Certificate System Subsystem Package Installation

These machines have all been registered with Red Hat Network (RHN) to contain the appropriate Red Hat subscriptions to download and install Red Hat Certificate System 8.1 (see Section B.2.2.1, “Prerequisites: RHEL 5.6 Operating System Installation and Configuration”).
NOTE

Check the /etc/yum.repos.d/ directory for the existence of either epel.repo or epel-testing.repo. If either file is present, make certain that ALL sections in both epel.repo and epel-testing.repo have been set to enabled=0 so that no EPEL packages accidentally override the officially desired RHEL RPMS.

To install a CA subsystem, perform the following commands on **Machine-1 (server1.example.com)**:

```
[root@server1]# yum install pki-ca
[root@server1]# yum install pki-console
[root@server1]# yum -y update
```

To install DRM, OCSP, TKS subsystems, perform the following commands on **Machine-2 (server2.example.com)**:

```
[root@server2]# yum install pki-kra
[root@server2]# yum install pki-ocsp
[root@server2]# yum install pki-tks
[root@server2]# yum install pki-console
[root@server2]# yum -y update
```

To install a TPS subsystem perform the following commands on **Machine-3 (server3.example.com)**:

```
[root@server3]# yum install pki-tps
[root@server3]# yum -y update
```

**B.2.2.3. Hardware Security Module SELinux Context**

NOTE

Procedures in this section should be performed on **Machine-1**, **Machine-2**, and **Machine-3**.

Prior to creating any Red Hat Certificate System 8.1 instances, since this test requires using an nCipher nShield HSM (see Section B.2.2.1.2, “Hardware Security Module Installation”), restore the SELinux context by running the restorecon command as the super user and restart the nCipher nShield HSM:

```
[root@server1]# restorecon -R /opt/nfast
```
B.2.2.4. Directory Server Instance Installation and Configuration

**NOTE**

The "Directory Server Instance Installation and Configuration" section is **ONLY** performed on **Machine-1 (server1.example.com)**.

Since Red Hat Directory Server 8.2 has been installed on this machine (see Section B.2.2.2.1, “Directory Server Subsystem Package Installation”), the user should now install and configure a Red Hat Directory Server 8.2 instance appropriate for use by the Red Hat Certificate System 8.1 instances required for this test.

**Important**

The Certificate System SELinux policies assume that the Red Hat Directory Server is listening over the standard LDAP/LDAPS ports, 389 and 636, respectively.

To install a Red Hat Directory Server 8.2 instance appropriate for use by the Red Hat Certificate System instances included in this test, perform the instructions in the following procedure, **choosing default values**:

*Red Hat Directory Server 8.2: Typical Setup*

B.2.2.5. CA Instance Creation and Configuration

**NOTE**

This procedure is **only** performed on **Machine-1 (server1.example.com)**.

B.2.2.5.1. CA Instance Creation

Use the **pkicreate** tool to create a Red Hat Certificate System instance of a CA.
Note

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the pkicreate tool:

```
[root@server1]# pkicreate -pki_instance_root=/var/lib        
   -pki_instance_name=pki-ca          
   -subsystem_type=ca                 
   -agent_secure_port=9443            
   -ee_secure_port=9444               
   -ee_secure_client_auth_port=9446   
   -admin_secure_port=9445            
   -unsecure_port=9180                
   -tomcat_server_port=9701           
   -user=pkiuser                      
   -group=pkiuser                     
   -audit_group=pkiaudit              
   -redirect conf=/etc/pki-ca         
   -redirect logs=/var/log/pki-ca     
   -verbose
```

Optional Reference: Section 6.1, “About pkicreate”

B.2.2.5.2. Directory Server SSL Server Authentication Configuration (Temporary)

Note

This section creates a temporary server certificate and temporary self-signed CA to establish SSL communication between the Red Hat Directory Server and the Red Hat Certificate System CA during its configuration. These certificates will be replaced in Section B.2.2.5.5, “Directory Server SSL Server Authentication Configuration (Trusted)”.

Stop the CA instance before modifying its NSS security databases:

```
[root@server1]# /sbin/service pki-ca stop
Stopping pki-ca: .. [ OK ]
```

Stop the Red Hat Directory Server instance before modifying its NSS security databases:

```
[root@server1]# /sbin/service dirsrv stop
Shutting down dirsrv:
   server1... [ OK ]
```

Make a backup of Red Hat Directory Server contents:

```
[root@server1]# cd /etc/dirsrv/slapd-server1/
```

```
[root@server1]# ls
cert8.db  dse.ldif  dse.ldif.startOK  key3.db  secmod.db
certmap.conf  dse.ldif.bak  dse_original.ldif  schema  slapd-collations.conf
```
Using a password file, establish a password for the existing NSS security databases:

```
[root@server1]# tar -cf /tmp/db-backup.tar *
```

```
[root@server1]# echo Secret123 > /tmp/pwdfile
```

```
[root@server1]# /usr/bin/certutil -W -d . -f /tmp/pwdfile
```

Create a temporary Self-Signed CA certificate for LDAP:

```
[root@server1]# /usr/bin/certutil -S -n "LDAP self-signed CA certificate" -s "cn=LDAPCA-cert,dc=example,dc=com" -2 -x -t "CT,,” -m 1000 -v 120 -d . -k rsa -f /tmp/pwdfile
```

A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO not USE THE AUTORePEAT FUNCTION ON YOUR KEYBOARD!

```
|********************************************************|
```

```
Finished. Press enter to continue:
```

```
Generating key. This may take a few moments...
```

```
Is this a CA certificate [y/N]?
```

```
y
```

```
Enter the path length constraint, enter to skip [<0 for unlimited path]: > -1
```

```
Is this a critical extension [y/N]?
```

```
y
```

Create a temporary Server Cert (and sign it using the temporary CA certificate created above):
A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO not USE THE AUTO REPEAT FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

```
|************************************************************|
```

Finished. Press enter to continue:

Generating key. This may take a few moments...

```
[root@server1]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server1
Certificate Nickname                              Trust Attributes     
Server-Cert-ldap                                   ssl,s/mime,jar/xpi
LDAP self-signed CA certificate                   ct,ssl,s/mime,jar/xpi
```

Start the Red Hat Directory Server instance:

```
[root@server1]# /sbin/service dirsrv start
Starting dirsrv:
    server1... [  OK  ]
```

Create an LDIF file by running the following command:

```
NOTE

In the command listed below, "> " is merely the command-line prompt; do not type these initial two characters.
```
Enable SSL in the LDAP Server by running the following command:

```
[root@server1]# /usr/bin/ldapmodify -x -h server1.example.com -p 389 -D "cn=Directory Manager" -w Secret123 -f /tmp/example.ldif
```

```
modifying entry "cn=config"
modifying entry "cn=encryption,cn=config"
adding new entry "cn=RSA,cn=encryption,cn=config"
```

Ensure that `/etc/dirsrv/slapd-server1/dse.ldif` has appropriate directives for SSL/TLS to work smoothly:

```
[root@server1]# egrep -i 'nsslapd-security|nsssl3|nsssslclientauth|nsssslpersonalityssl|nsSSLActivation' /etc/dirsrv/slapd-server1/dse.ldif
```

```
nsslapd-security: on
nsSSLClientAuth: allowed
nsSSL3: on
nsSSL3Ciphers: -rsa_null_md5,+rsa_rc4_128_md5,+rsa_rc4_40_md5,+rsa_rc2_40_md5,+rsa_fips_des_sha,+rsa_3des_sha,+rsa_fips_3des_sha,+fortezza,+fortezza_rc4_128_sha,+fortezza_null,+tls_rsa_export1024_with_rc4_56_sha,+tls_rsa_export1024_with_des_cbc_sha
```

Restart the Red Hat Directory Server instance (on a secure port):
NOTE

The PIN being prompted for below is merely referring to the password used throughout this test.

[root@server1]# /sbin/service dirsrv restart
Shutting down dirsrv:
  server1... [ OK ]
Starting dirsrv:
  server1...Enter PIN for Internal (Software) Token: [ OK ]

Perform an LDAP search with SSL/TLS to ensure that everything is working smoothly:

[root@server1]# /usr/lib64/mozldap/ldapsearch -h server1.example.com -p 636 -Z -P
/etc/dirsrv/slapd-server1/cert8.db -D "cn=directory manager" -w Secret123 -b
"cn=config" objectclass=* dn
version: 1
dn: cn=config
dn: cn=encryption,cn=config
dn: cn=features,cn=config
dn: cn=mapping tree,cn=config
.
.
Export the CA certificate from dirsrv into base-64 pem format:

[root@server1]# /usr/bin/certutil -L -d /etc/dirsrv/slapd-server1 -n "LDAP self-
signed CA certificate" -a > ldap-ca.crt

Import the LDAP self-signed CA certificate into the pki-ca's NSS security database with the appropriate trust attributes set:

[root@server1]# /usr/bin/certutil -A -i ldap-ca.crt -t "CT,C,C" -d /var/lib/pki-
ca/alias -n "LDAP self-signed CA certificate"

IMPORTANT

Configuring SSL between Red Hat Certificate System and Red Hat Directory Server won't work unless the appropriate trust attributes ('CT,C,C') are set.

List certificates in the pki-ca NSS security database after importing the LDAP self-signed CA certificate:
[root@server1]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias/

<table>
<thead>
<tr>
<th>Certificate Nickname</th>
<th>Trust Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL, S/MIME, JAR/XPI</td>
<td>SSL, S/MIME, JAR/XPI</td>
</tr>
<tr>
<td>LDAP self-signed CA certificate</td>
<td>CT, C, C</td>
</tr>
<tr>
<td>Server-Cert cert-pki-ca</td>
<td>CTu, Cu, Cu</td>
</tr>
</tbody>
</table>

Start the CA instance:

[root@server1]# /sbin/service pki-ca start
Starting pki-ca:
Using Java Security Manager
Constructing 'pki-ca.policy' Security Policy
Starting pki-ca: [ OK ]
pki-ca (pid 26728) is running ...

'pki-ca' must still be CONFIGURED!
(see /var/log/pki-ca-install.log)


B.2.2.5.3. Hardware Security Module CA Configuration

Follow instructions in the following procedure:

Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”

B.2.2.5.4. CA Instance Configuration

Since pkicreate has been executed in Section B.2.2.5.1, “CA Instance Creation”, obtain the URL to configure the CA by doing a tail /var/log/pki-ca-install.log:

[root@server1]# tail /var/log/pki-ca-install.log
[2011-03-27 01:40:59] [debug] Restorecon file context for /etc/pki-ca/tomcat5.conf
[2011-03-27 01:40:59] [debug] Setting selinux context pki_ca_port_t for 9446
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' runlevel to '.-
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' start priority to '81'
[2011-03-27 01:41:04] [debug] Setting 'pki-ca' stop priority to '19'
[2011-03-27 01:41:04] [debug] Registered 'pki-ca' with '/sbin/chkconfig.
[2011-03-27 01:41:14] [log] Configuration Wizard listening on https://server1.example.com:9445/ca/admin/console/config/login?
pin=rc16U50F6eUt92X3HbHg
[2011-03-27 01:41:14] [log] After configuration, the server can be operated by the command:
/sbin/service pki-ca start | stop | restart

**Important**

For configuration purposes, a Firefox browser must be used to configure all PKI subsystems.

As an administrator, launch a Firefox browser using the profile manager:
[root@server1]# /usr/bin/firefox -ProfileManager -no-remote

Create a new profile entitled server:

- Click Next>
- Enter new profile name: server
- Click Finish

Open the URL displayed in /var/log/pki-ca-install.log in the Firefox browser.

Within the Firefox browser, click "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

 Traverse the CA configuration panels:

- Welcome panel:
  - Click Next>
- Key Store panel:
  - Click Login for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the security module token password.
  - Click Next>.
  - Back in the Key Store panel, select the NHSM6000-OCS Token.
  - Click Next>.
- Security Domain panel:
  - Accept the defaults.
  - Click Next>.
- Subsystem Type panel:
  - Accept the defaults.
  - Click Next>.
- PKI Hierarchy panel:
  - Accept the defaults.
  - Click Next>.
- Internal Database panel:
  - Host -- server1.example.com
  - Port -- 636 (SSL port)
  - Check the SSL box.
  - Provide the bind password of the Directory Manager (value which you gave while creating a Red Hat Directory Server instance).
  - Click Next>.
- Key Pairs panel:
  - Accept the defaults.
  - Select the Use the default key size (2048 bits). radio button.
  - Click Next>.
- Subject Names panel:
  - Accept the defaults.
  - Click Next>.
- Requests and Certificates panel:
Click Apply.
Once the screen redraws, click Next>.

**NOTE**

It is not possible to export keys and certificates stored on an HSM to a .p12 file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

- **Export Keys and Certificates** panel:
  - Accept the defaults.
  - Click Next>.
- **Import CA's Certificate Chain** panel:
  - Click OK for the prompt 'You will now be asked to import and trust the Certificate Chain from the CA. Please do so.'
  - Select the checkboxes for:
    - Check 'Trust this CA to identify web sites'.
    - Check 'Trust this CA to identify email users'.
    - Check 'Trust this CA to identify software developers'.
  - Click OK to dismiss this dialog box.
  - Click Next>.
- **Administrator** panel:
  - UID: admin.
  - Name: CA Administrator of Instance pki-ca.
  - Email: pki-ca-admin@example.com.
  - Password:
  - Password (Again):
  - Click Next>.
- **Import Administrator's Certificate** panel:
  - Click OK on the alert which says 'Your personal certificate has been installed. You should keep a backup copy of this certificate.'
  - Click Next>.
- **Done** panel:
  - Restart the CA by running /sbin/service pki-ca restart.

**Enabling FIPS Mode**

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, "Using Hardware Security Modules with Subsystems" and the vendor documentation.
2. Install and configure the CA instance.
3. Stop the CA instance. The instance must be stopped to protect the information stored in its security databases.

```
service pki-ca stop
```
4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   
a. Open the CA's security database directory.
   
```bash
cd /var/lib/pki-ca/alias
```

b. Using `certutil`, create a request for a new SSL server certificate.
   
```bash
certutil -d . -R -s "CN=server1.example.com,OU=pki-ca,O=Example Domain pki-ca" -o sslfips.req -h "NSS Certificate DB" -a
```

c. Restart the CA.
   
```bash
service pki-ca start
```

d. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

e. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

f. Copy the base 64-encoded certificate on the approval page and save it to a file, such as `sslfips.cert`.

g. Stop the CA again.
   
```bash
service pki-ca stop
```

h. Check the CA's certificate database to see if an SSL server certificate is already listed.
   
```bash
certutil -d /var/lib/pki-ca/alias -L
```

i. If the certificate exists, then delete it.
   
```bash
certutil -d /var/lib/pki-ca/alias -D -n "ServerCert nickname"
```

j. Import the new SSL server certificate.
   
```bash
certutil -d /var/lib/pki-ca/alias -A -t "u,u,u" -n "ServerCert nickname" server1.example.com - Example Domain pki-ca" -i sslfips.cert -a
```

k. Edit the `/var/lib/pki-ca/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert ca.example.com - Example Domain pki-ca`.

l. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.
   
```bash
vim /var/lib/pki-ca/conf/CS.cfg
```

```bash
c.a.cert.sslserver.nickname= ServerCert ca.example.com - Example Domain pki-ca
c.a.sslserver.nickname= ServerCert ca.example.com - Example Domain pki-ca
```

m. In the `CS.cfg` file, add a line to verify signatures from the token. The value is the token name, which depends on the vendor and version of the HSM. For example, for a NetHSM token:
n. Edit the `server.xml` file to enable FIPS mode for each SSL-enabled connector. Set `strictCiphers` to true and add or set `ssl3` to false.

```xml
vim /var/lib/pki-ca/conf/server.xml

<Connector name="Agent" port="9443" maxHttpHeaderSize="8192"
    ... 
    ...
    sslOptions="ssl2=false,ssl3=false,tls=true"
    strictCiphers="true"
    ...
>
```

o. Enable FIPS mode in the NSS software database.

```
modutil -dbdir /var/lib/pki-ca/alias -fips true
```

p. Verify that FIPS mode has been enabled. The command will return the current FIPS status.

```
modutil -dbdir /var/lib/pki-ca/alias modutil -dbdir . -chkfips true
FIPS mode enabled.
```

q. Start the CA.

```
service pki-ca start
```

B.2.2.5.5. Directory Server SSL Server Authentication Configuration (Trusted)

Create a certificate signing request for Server-Cert-ldap (ensure cn=server1.example.com [FQDN] with the same nickname Server-Cert-ldap):

```
[root@server1]# cd /etc/dirsrv/slapd-server1/
```
A random seed must be generated that will be used in the creation of your key. One of the easiest ways to create a random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter is full. DO not USE THE AUTORepeat FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

|****************************|  

Finished. Press enter to continue:

Generating key. This may take a few moments...

Certificate request generated by Netscape certutil
Phone: (not specified)
Common Name: server1.example.com
Email: (not specified)
Organization: (not specified)
State: (not specified)
Country: (not specified)

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBZjCB0AIBADAnMSUwIwYDVQQDExxtZWF0cGllLmRzZGV2LnNqYy5yZWRoYXQuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDCiCR7kumfiUozvkUYoD7w1u8TFjwtpCpc1ChAnUzGF7jqqHQ4vlasLbJlTMefO6pqayQbIt0DhzmNhCuzwSP+mEMgKq1qgo7rsGGt9JbaRYSJCE8sbCiih0cOieeSrSpjJIMuXY4a3mnZkKZrhiPkmEsRY07ahtf61bhVM/rwdIDAQABOAAwDQYJKoZIhvcNAQEFBQADgYEAuUMC4r1Y8qBm9f/40tY4n8dQ16466BMcgno5yQqTJf6fNqZC1gydZo6P4ibm8txMBVCw/sVw6UpGFmSc+APufeQUUUbG7c4i1Z2R08YnhySBpMwasvU8W++D25YjfyGwHg8FYAgI0YRTGzapYwo3mvSPd9GJYXUTwtqZXS2JU0=  
-----END NEW CERTIFICATE REQUEST-----

As an administrator, launch a Firefox browser using the profile manager:

```
[root@server1]# /usr/bin/firefox -ProfileManager -no-remote
```

Select the profile entitled server that was utilized for configuration of the CA in Section B.2.5.4, "CA Instance Configuration".

Select the URL that points to the Secure Admin Port of the previously configured CA and open the browser to a new tab called CA Services.
Select the URL called **SSL End Users Services** to open a new tab called **CA End-Entity** within the browser.

Select the profile entitled **Manual Server Certificate Enrollment**, and perform the following procedures:

- Cut the certificate request contained in the file called `/etc/dirsrv/slapd-server1/Server-Cert-ldap.req` and paste it into the area called Certificate Request on the form:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBZjCB0AIBADAmMSUwIwYDVQQDExtZWF0cGllLmRzZGV2LnNqYy5yZwRoYXQu
Y29tMIgFM0GCSqGSIb3DQEBAQAA4GNADCBiQKBgQDCiCR7kumfiU0zvkUYoD7w
1u0TfjwCp1CHaUMg7jqqH4v1asLbJ1TMeF06pqayQbItO0hzmzNhNcZueSP+
meMqKq1pqq7rsGt9JbaYSjC8sblh00CioeSrSp1JIMwXY4a3mnZkKzrhiP
kmEsRY07ahT61bhVHMrw/sQNoZJkoZIhvcNAQEBQADgYEaUMC4r1Y
8qBmf9///40tY4n8q016460BMcn5yQqTj6fN6zZC1gydZo6P41b8txMBvC/W/sVw
sUpGfMSc+APuFeQUUUbG7c4I1IZ2R28hnysBpMwasvU8W++d25YfjyGwHg8FYAgI
0YRTGzapyWo3mVSPd96JYXTwTqzXS2S3U0=
-----END NEW CERTIFICATE REQUEST-----
```

- Press the **Submit** button

A page entitled **Certificate Profile** should appear noting a request ID.

Within the browser, click the tab entitled **CA Services**, and select the URL called **Agent Services** to open a new tab called **CA Agent**.

Select the browser tab labeled **CA Agent**, and perform the following tasks:

- Click **List Requests**
- Click **Find**
- Click the request id labeled **CN=server1.example.com**
- Scroll to the bottom of this request, and click the **submit** button to approve this request.
- Scroll to the bottom of the approved request and highlight the Base-64 Encoded Certificate:
-----BEGIN CERTIFICATE-----
MIIDETCCAfmgAwIBAgIBBzANBgkqhkiG9w0BAQsFADBMR4wHAYDVQQKEExVEc2RldmQx
MjBEeTAeMwYwHAYDVQQKExVEc2RldNqY1JlZGhhdCBzeaA0bCBhcnkgbGF5IzAIIBA

Copy this highlighted Base-64 Encoded certificate and paste it into a file called `/etc/dirsrv/slapd-server1/ldap-server-b64.crt`.

Obtain the CA's signing certificate into a base-64-encoded pem format:

```
[root@server1]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias/ -n "caSigningCert cert-pki-ca" -a > /etc/dirsrv/slapd-server1/casigning-b64.crt
```

Stop the CA instance before modifying its NSS security databases:

```
[root@server1]# /sbin/service pki-ca stop
Stopping pki-ca: ..                                      [  OK  ]
```

Stop the Red Hat Directory Server instance before modifying its NSS security databases:

```
[root@server1]# /sbin/service dirsrv stop
Shutting down dirsrv: server1...                         [  OK  ]
```

Delete the Server-Cert-ldap Certificate Server instance issued earlier in Section B.2.2.5.2, "Directory Server SSL Server Authentication Configuration (Temporary)":

```
[root@server1]# /usr/bin/certutil -D -d . -n "Server-Cert-ldap"
```

Import the new ldap-server-b64.crt (issued by pki-ca) into the Red Hat Directory Server NSS security database:

```
[root@server1]# /usr/bin/certutil -A -i ldap-server-b64.crt -d . -n "Server-Cert-ldap" -t "u,u,u"
```

Import caSigningCert cert-pki-ca (extracted from the certificate-ca NSS security database) into the Red Hat Directory Server NSS database and trust it:

```
[root@server1]# /usr/bin/certutil -A -i casigning-b64.crt -d /etc/dirsrv/slapd-server1 -n "caSigningCert cert-pki-ca" -t "CTu,Cu,Cu"
```

--- END CERTIFICATE ---
Delete LDAP self-signed CA certificate from pki-ca's NSS security database:

```
[root@server1]# /usr/bin/certutil -D -d /var/lib/pki-ca/alias -n "LDAP self-signed CA certificate"
```

Start the Red Hat Directory Server instance (on a secure port):

```
NOTE

The PIN being prompted for below is merely referring to the password used throughout this test.
```

```
[root@server1]# /sbin/service dirsrv start
Starting dirsrv:
server1...Enter PIN for Internal (Software) Token: [ OK ]
```

Start the CA instance:

```
[root@server1]# /sbin/service pki-ca start
Starting pki-ca:
Using Java Security Manager
Constructing 'pki-ca.policy' Security Policy
Please provide password for internal:
Please provide password for hardware-nethsm2k:
Please provide password for internaldb:
Please provide password for replicationdb:
Starting pki-ca: [ OK ]
pki-ca (pid 26828) is running ...

Unsecure Port = http://server1.example.com:9180/ca/ee/ca
Secure Agent Port = https://server1.example.com:9443/ca/agent/ca
Secure EE Port = https://server1.example.com:9444/ca/ee/ca
Secure Admin Port = https://server1.example.com:9445/ca/services
EE Client Auth Port = https://server1.example.com:9446/ca/eeca/ca
PKI Console Port = pkiconsole https://server1.example.com:9445/ca
Tomcat Port = 9701 (for shutdown)

PKI Instance Name: pki-ca
PKI Subsystem Type: Root CA (Security Domain)

Registered PKI Security Domain Information:
==================================================================================
Name: Example Domain
URL: https://server1.example.com:9445
==================================================================================
```

**B.2.2.5.6. Configuring POSIX System ACLs**

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

1. Log into the system as root.
2. Make sure that the acl package is installed on the system.

```bash
# rpm -q acl
acl-2.2.39-6.el5
```

3. Four filesystem paths must be configured to use the acl package:
   - `/`
   - `/etc` (including `/etc/init.d` and `/etc/sysconfig`)
   - `/usr` (including `/usr/bin` and `/usr/share`)
   - `/var` (including `/var/lock` and `/var/run`)

First, check to see where the filesystems are mounted and whether the acl option is used:

```bash
# mount
/dev/sda1 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sdb1 on /usr type ext2 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

In this example, `/`, `/etc`, and `/var` are mounted on `/dev/sda1`, while `/usr` is mounted on `/dev/sdb1`.

There are two ways to apply the acl option to the filesystems:

- Using `tune2fs`:

  ```bash
  # /sbin/tune2fs -o +acl /dev/sda1
  # /sbin/tune2fs -o +acl /dev/sdb1
  
  # vim /etc/fstab
  
  LABEL=/        /            ext3          defaults,acl  1 1
  LABEL=/usr     /usr         ext2          defaults,acl  1 2
  tmpfs          /dev/shm     tmpfs         defaults        0 0
  devpts         /dev/pts     devpts        gid=5,mode=620 0 0
  sysfs          /sys         sysfs         defaults        0 0
  proc           /proc        proc           defaults        0 0
  LABEL=SWAP-sdb2 swap swap defaults 0 0
  LABEL=SWAP-sda2 swap swap defaults 0 0
  
  Then, remount the filesystems.
  
  ```bash
  # mount -o remount /
  # mount -o remount /usr
  ```

Confirm that the acl option has been applied to the filesystems:
# /sbin/dumpe2fs /dev/sda1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:    user_xattr acl

# /sbin/dumpe2fs /dev/sdb1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:    acl

# mount | grep acl
/dev/sda1 on / type ext3 (rw,acl)
/dev/sdb1 on /usr type ext2 (rw,acl)

4. Stop the CA instance.

   service pki-ca stop

5. Set the group readability to the pkiadmin group for the instance's directories and files.

   # setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki-ca

6. Apply execute (x) ACL permissions on all directories:

   # find -L /var/lib/pki-ca -type d -exec setfacl -L -n -m g:pkiadmin:rx,d:g:pkiadmin:rx {} \;

7. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:

   # setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/pki-ca/logs/signedAudit

8. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

   # setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki-ca/logs/signedAudit

9. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:

   # find -L /var/lib/pki-ca/logs/signedAudit -type d -exec setfacl -L -n -m g:pkiaudit:rx,d:g:pkiaudit:rx {} \;

10. Start the CA.

    service pki-ca start

11. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:
B.2.2.6. DRM Instance Creation and Configuration

**NOTE**

The "DRM Instance Creation and Configuration" section is **ONLY** performed on Machine-2 (server2.example.com).

### B.2.2.6.1. DRM Instance Creation

Use the `pkicreate` tool to create a Red Hat Certificate System instance of an DRM Subsystem.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the `pkicreate` tool:
Optional Reference: Section 6.1, “About pkicreate”

B.2.2.6.2. Hardware Security Module DRM Configuration
Follow instructions in the following procedure (substituting the name pki-kra for pki-ca):

Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”

B.2.2.6.3. DRM Instance Configuration

NOTE
This Section involves configuring an DRM instance using SSL with Directory Server

Since pkicreate has been executed in Section B.2.2.6.1, “DRM Instance Creation”, obtain the URL to configure the DRM Subsystem by doing a tail /var/log/pki-kra-install.log:

```
[root@server2]# tail /var/log/pki-kra-install.log
[2011-03-14 04:54:53] [debug] Setting 'pki-kra' runlevel to '-'
[2011-03-14 04:54:53] [debug] Setting 'pki-kra' start priority to '82'
[2011-03-14 04:54:53] [debug] Setting 'pki-kra' stop priority to '18'
[2011-03-14 04:54:53] [debug] Registered 'pki-kra' with '/sbin/chkconfig.
[2011-03-14 04:55:02] [log] Configuration Wizard listening on
https://server2.example.com:19245/kra/admin/console/config/login?
pin=0ID1d0JTQ5GgMhK8j0DK
[2011-03-14 04:55:02] [log] After configuration, the server can be operated by the command:
/sbin/service pki-kra start | stop | restart
```

Important

For the purposes of this test, the ONLY Firefox browser that will be used is the one installed on Machine-1 (server1.example.com).

Additionally, ALL subsystems mentioned in this test will ONLY use the Firefox browser profile called server created in Section B.2.2.5.4, “CA Instance Configuration”.

As an administrator, launch a Firefox browser using the profile manager:
Select the profile entitled `server` that was utilized for configuration of the CA in `Section B.2.2.5.4, "CA Instance Configuration"`.  

Open the URL displayed in `/var/log/pki-kra-install.log` in the Firefox browser.

Within the Firefox browser, click "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

Traverse the DRM configuration panels:

- **Welcome** panel:
  - Click Next.

- **Key Store** panel:
  - Click Login for NHSM6000-OCS (nCipher’s nFast Token Hardware Module).
  - Provide the security module token password.
  - Click Next.
  - Back in the Key Store panel, select the NHSM6000-OCS Token.
  - Click Next.

- **Security Domain** panel:
  - Select ‘Join an Existing Security Domain and enter the URL to an existing security domain. This URL can be identified by running `/sbin/service pki-ca status` on Machine-1 where CA subsystem is configured.
  - Click Next.

- **Display Certificate Chain** panel:
  - Click Next.

- **Security Domain Login** panel:
  - Supply the admin UID name and password for the CA so that it can be properly accessed.
  - Click Login

- **Subsystem Type** panel:
  - Accept the defaults.
  - Click Next.

- **Internal Database** panel:
  - Host -- `server1.example.com`
  - Port -- 636 (SSL port)
  - Check the SSL box.
  - Provide the bind password of the Directory Manager (value which you gave while creating a Red Hat Directory Server instance).
  - Click Next.

- **Key Pairs** panel:
  - Accept the defaults.
  - Select the **Use the default key size (2048 bits)**. radio button.
  - Click Next.

- **Subject Names** panel:
  - Accept the defaults.
  - Click Next.
» Requests and Certificates panel:
   - Click Apply.
   - Once the screen redraws, click Next>.

**NOTE**

It is not possible to export keys and certificates stored on an HSM to a .p12 file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

» Export Keys and Certificates panel:
   - Accept the defaults.
   - Click Next>.

» Administrator panel:
   - UID: admin
   - Name: KRA Administrator of Instance pki-kra
   - Email: pki-kra-admin@example.com
   - Password:
   - Password (Again):
   - Click Next>.

» Import Administrator's Certificate panel:
   - Click OK on the alert which says 'Your personal certificate has been installed. You should keep a backup copy of this certificate.'
   - Click Next>.

» Done panel:
   - Stop the DRM Subsystem by running /sbin/service pki-kra stop.

**Enabling FIPS Mode**

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, "Using Hardware Security Modules with Subsystems" and the vendor documentation.
2. Install and configure the instance.
3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

   service instance_name stop

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance's security database directory.

      cd /var/lib/instance_name/alias

   b. Using certutil, create a request for a new SSL server certificate.
c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as sslfips.cert.

f. Check the instance's certificate database to see if an SSL server certificate is already listed.

g. If the certificate exists, then delete it.

h. Import the new SSL server certificate.

i. Edit the /var/lib/instance_name/conf/serverCertNick.conf file to contain the nickname of the new certificate, such as ServerCert server2.example.com - Example Domain instance_name.

j. Edit the CS.cfg file to replace both references to the SSL server certificate nickname.

k. Edit the server.xml file to enable FIPS mode for each SSL-enabled connector. Set strictCiphers to true and add or set ssl3 to false. For example:

l. Enable FIPS mode in the NSS software database.

m. Verify that FIPS mode has been enabled. The command will return the current FIPS status.
modutil -dbdir /var/lib/instance_name/alias modutil -dbdir . -chkfips true

FIPS mode enabled.

n. Start the instance.

service instance_name start

B.2.2.6.4. Configuring POSIX System ACLs

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

1. Log into the system as root.
2. Make sure that the acl package is installed on the system.

   # rpm -q acl
   acl-2.2.39-6.el5

3. Four filesystem paths must be configured to use the acl package:

   » /
   » /etc (including /etc/init.d and /etc/sysconfig)
   » /usr (including /usr/bin and /usr/share)
   » /var (including /var/lock and /var/run)

First, check to see where the filesystems are mounted and whether the acl option is used:

   # mount
   /dev/sda1 on / type ext3 (rw)
   proc on /proc type proc (rw)
   devpts on /dev/pts type devpts (rw,gid=5,mode=620)
   /dev/sdb1 on /usr type ext2 (rw)
   tmpfs on /dev/shm type tmpfs (rw)
   none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
   sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)

In this example, /, /etc, and /var are mounted on /dev/sda1, while /usr is mounted on /dev/sdb1.

There are two ways to apply the acl option to the filesystems:

   » Using tune2fs:

      # /sbin/tune2fs -o +acl /dev/sda1
      # /sbin/tune2fs -o +acl /dev/sdb1

   » Editing the /etc/fstab file:
Then, remount the filesystems.

```
# mount -o remount /
# mount -o remount /usr
```

Confirm that the `acl` option has been applied to the filesystems:

```
# /sbin/dumpe2fs /dev/sda1 | grep acl
# /sbin/dumpe2fs /dev/sdb1 | grep acl
# mount | grep acl
```

4. Stop the DRM instance.

```
service pki-kra stop
```

5. Set the group readability to the pkiadmin group for the instance's directories and files.

```
# setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki-kra
```

6. Apply execute (x) ACL permissions on all directories:

```
# find -L /var/lib/pki-kra -type d -exec setfacl -L -n -m g:pkiadmin:rx,d:g:pkiadmin:rx {} \;
```

7. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/pki-kra/logs/signedAudit
```

8. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki-kra/logs/signedAudit
```

9. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:
10. Start the CA.

   service pki-kra start

11. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:

   ```
   # getfacl /var/lib/pki-kra /var/log/pki-kra/logs/signedAudit/
   getfacl: Removing leading '/' from absolute path names
   # file: var/lib/pki-kra
   # owner: pkiuser
   # group: pkiuser
   user::rwx
   group::rwx
   group:pkiadmin:r-x
   mask::rwx
   other::r-x
   default:user::rwx
   default:group::rwx
   default:group:pkiadmin:r-x
   default:mask::rwx
   default:other::r-x

   # file: var/lib/pki-kra/logs/signedAudit
   # owner: pkiuser
   # group: pkiaudit
   user::rwx
   group::rwx
   group:pkiaudit:r-x
   mask::rwx
   other::---
   default:user::rwx
   default:group::rwx
   default:group:pkiaudit:r-x
   default:mask::rwx
   default:other::---
   ```

### B.2.2.7. OCSP Instance Creation and Configuration

**NOTE**

The "OCSP Instance Creation and Configuration" section is ONLY performed on Machine-2 (server2.example.com).

#### B.2.2.7.1. OCSP Instance Creation

Use the `pkicreate` tool to create a Red Hat Certificate System instance of an OCSP Responder.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.
Use the pkicreate tool:

```
[root@server2]# pkicreate -pki_instance_root=/var/lib         
   -pki_instance_name=pki-ocsp         
   -subsystem_type=ocsp               
   -agent_secure_port=11443           
   -ee_secure_port=11444              
   -admin_secure_port=11445           
   -unsecure_port=11180               
   -tomcat_server_port=11701          
   -user=pkiuser                      
   -group=pkiuser                     
   -audit_group=pkiaudit              
   -redirect conf=/etc/pki-ocsp       
   -redirect logs=/var/log/pki-ocsp   
   -verbose
```

Optional Reference: Section 6.1, "About pkicreate"

**B.2.2.7.2. Hardware Security Module OCSP Configuration**

Follow instructions in the following procedure (substituting the name pki-ocsp for pki-ca):

Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”

**B.2.2.7.3. OCSP Instance Configuration**

**NOTE**

This Section involves configuring an OCSP instance using SSL with Directory Server

Since **pkicreate** has been executed in Section B.2.2.7.1, “OCSP Instance Creation”, obtain the URL to configure the OCSP Responder by doing a tail `/var/log/pki-ocsp-install.log`:

```
[root@server2]# tail /var/log/pki-ocsp-install.log
[2011-03-14 07:23:28] [debug] Setting 'pki-ocsp' runlevel to '-'
[2011-03-14 07:23:28] [debug] Setting 'pki-ocsp' start priority to '83'
[2011-03-14 07:23:28] [debug] Setting 'pki-ocsp' stop priority to '17'
[2011-03-14 07:23:37] [log] Configuration Wizard listening on
https://server2.example.com:11445/ocsp/admin/console/config/login?
pin=Q1R94Mwz5MPnNZlj7cnW
[2011-03-14 07:23:37] [log] After configuration, the server can be operated by the
command:
/sbin/service pki-ocsp start | stop | restart
```

**Important**

For the purposes of this test, the **ONLY** Firefox browser that will be used is the one installed on **Machine-1 (server1.example.com)**.

Additionally, **ALL** subsystems mentioned in this test will **ONLY** use the Firefox browser profile called **server** created in Section B.2.2.5.4, “CA Instance Configuration”.
As an administrator, launch a Firefox browser using the profile manager:

```
[root@server1]# /usr/bin/firefox -ProfileManager -no-remote
```

Select the profile entitled **server** that was utilized for configuration of the CA in Section B.2.5.2, "CA Instance Configuration".

Open the URL displayed in `/var/log/pki-ocsp-install.log` in the Firefox browser.

Within the Firefox browser, click "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

Traverse the OCSP configuration panels:

- **Welcome** panel:
  - Click Next>
- **Key Store** panel:
  - Click Login for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the 'Security Module Token Password'
  - Click Next>
- Back in the **Key Store** panel, select the NHSM6000-OCS Token
  - Click Next>
- **Security Domain** panel:
  - Select 'Join an Existing Security Domain and enter the URL to an existing security domain. This URL can be identified by running `/sbin/service pki-ca status` on Machine-1 where CA subsystem is configured.'
  - Click Next>
- **Display Certificate Chain** panel:
  - Click Next>
- **Security Domain Login** panel:
  - Supply the admin UID name and password for the CA so that it can be properly accessed.
  - Click Login.
- **Subsystem Type** panel:
  - Accept the defaults.
  - Click Next>
- **Internal Database** panel:
  - Host -- **server1.example.com**
  - Port -- **636** (SSL port)
  - Check the SSL box.
  - Provide the bind password of the Directory Manager (value which you gave while creating a Red Hat Directory Server instance).
  - Click Next>
- **Key Pairs** panel:
  - Accept the defaults.
  - Select the **Use the default key size (2048 bits)** radio button.
  - Click Next>
- **Subject Names** panel:
Accept the defaults.
- Click Next>.

**Requests and Certificates** panel:
- Click Apply.
- Once the screen redraws, click Next>.

**NOTE**

It is not possible to export keys and certificates stored on an HSM to a .p12 file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

**Export Keys and Certificates** panel:
- Accept the defaults.
- Click Next>.

**Administrator** panel:
- UID: admin
- Name: **OCSP Administrator of Instance pki-ocsp**
- Email: pki-ocsp-admin@example.com
- Password:
- Password (Again):
- Click Next>.

**Import Administrator's Certificate** panel:
- Click OK on the alert which says ‘Your personal certificate has been installed. You should keep a backup copy of this certificate.’
- Click Next>.

**Done** panel:
- Restart the OCSP Responder by running `/sbin/service pki-ocsp restart`.

**Enabling FIPS Mode**

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in [Section 8.1.2, “Using Hardware Security Modules with Subsystems”](#) and the vendor documentation.
2. Install and configure the instance.
3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

```
   service instance_name stop
```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance’s security database directory.
      ```
      cd /var/lib/instance_name/alias
      ```
   b. Using certutil, create a request for a new SSL server certificate.
certutil -d . -R -s "CN=server2.example.com,OU=instance_name,0=Example Domain instance_name" -o sslfips.req -h "NSS Certificate DB" -a

c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.
d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.
e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as sslfips.cert.
f. Check the instance's certificate database to see if an SSL server certificate is already listed.

certutil -d /var/lib/instance_name/alias -L

g. If the certificate exists, then delete it.

certutil -d /var/lib/instance_name/alias -D -n "ServerCert nickname"

h. Import the new SSL server certificate.

certutil -d /var/lib/instance_name/alias -A -t "u,u,u" -n "ServerCert server2.example.com - Example Domain instance_name" -i sslfips.cert -a

i. Edit the /var/lib/instance_name/conf/serverCertNick.conf file to contain the nickname of the new certificate, such as ServerCert server2.example.com - Example Domain instance_name.

j. Edit the CS.cfg file to replace both references to the SSL server certificate nickname.

vim /var/lib/instance_name/conf/CS/cfg

type.cert.sslserver.nickname= ServerCert server2.example.com - Example Domain instance_name
type.sslserver.nickname= ServerCert server2.example.com - Example Domain instance_name

k. Edit the server.xml file to enable FIPS mode for each SSL-enabled connector. Set strictCiphers to true and add or set ssl3 to false. For example:

vim /var/lib/instance_name/conf/server.xml

<Connector name="Agent" port="11443" maxHttpHeaderSize="8192"
  ... 
  ... 
  sslOptions="ssl2=false,ssl3=false,tlsv1=true"
  strictCiphers="true"
  ... 
>

l. Enable FIPS mode in the NSS software database.

modutil -dbdir /var/lib/instance_name/alias -fips true

m. Verify that FIPS mode has been enabled. The command will return the current FIPS status.
**B.2.2.7.4. Configuring POSIX System ACLs**

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

**NOTE**

It is not necessary to apply the `acl` option and remount the filesystem because that was done for the DRM, and the DRM, OCSP, and TKS are on the same machine.

1. Log into the system as root.
2. Stop the OCSP instance.
   
   ```bash
   service pki-ocsp stop
   ```
3. Set a group readability of pkiadmin for the instance's directories and files.
   
   ```bash
   # setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki-ocsp
   ```
4. Apply execute (x) ACL permissions on all directories:
   
   ```bash
   # find -L /var/lib/pki-ocsp -type d -exec setfacl -L -n -m g:pkiadmin:rx,d:g:pkiadmin:rx {} \;
   ```
5. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:
   
   ```bash
   # setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/pki-ocsp/logs/signedAudit
   ```
6. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:
   
   ```bash
   # setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki-ocsp/logs/signedAudit
   ```
7. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:
   
   ```bash
   # find -L /var/lib/pki-ocsp/logs/signedAudit -type d -exec setfacl -L -n -m g:pkiaudit:rx,d:g:pkiaudit:rx {} \;
   ```
8. Start the OCSP.
   
   ```bash
   service pki-ocsp start
   ```
9. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:

```bash
# getfacl /var/lib/pki-ocsp /var/log/pki-ocsp/logs/signedAudit/
getfacl: Removing leading '/' from absolute path names
# file: var/lib/pki-ocsp
# owner: pkiuser
# group: pkiuser
user::rwx
group::rwx
group:pkiadmin:r-x
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:pkiadmin:r-x
default:mask::rwx
default:other::r-x

# file: var/lib/pki-ocsp/logs/signedAudit
# owner: pkiuser
# group: pkiaudit
user::rwx
group::rwx
group:pkiaudit:r-x
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:pkiaudit:r-x
default:mask::rwx
default:other::---
```

**B.2.2.8. TKS Instance Creation and Configuration**

**NOTE**

The "TKS Instance Creation and Configuration" section is **ONLY** performed on **Machine-2** (server2.example.com).

**B.2.2.8.1. TKS Instance Creation**

Use the `pkicreate` tool to create a Red Hat Certificate System instance of a TKS Subsystem.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the `pkicreate` tool:
B.2.2.8.2. Hardware Security Module TKS Configuration

Follow instructions in the following procedure (substituting the name pki-tks for pki-ca):

Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”

B.2.2.8.3. TKS Instance Configuration

**NOTE**

This Section involves configuring an TKS instance using SSL with Directory Server

Since `pkicreate` has been executed in Section B.2.2.8.1, “TKS Instance Creation”, obtain the URL to configure the TKS Subsystem by doing a tail /var/log/pki-tks-install.log:

```
[root@server2]# tail /var/log/pki-tks-install.log
[2011-03-14 07:47:42] [log] Configuration Wizard listening on https://server2.example.com:13445/tks/admin/console/config/login?pin=CGEfq8Du10EvwjbaTb6h
[2011-03-14 07:47:42] [log] After configuration, the server can be operated by the command:
/sbin/service pki-tks start | stop | restart
```

**Important**

For the purposes of this test, the **ONLY** Firefox browser that will be used is the one installed on **Machine-1 (server1.example.com)**.

Additionally, **ALL** subsystems mentioned in this test will **ONLY** use the Firefox browser profile called **server** created in Section B.2.2.5.4, “CA Instance Configuration”.

As an administrator, launch a Firefox browser using the profile manager:
Select the profile entitled server that was utilized for configuration of the CA in Section B.2.2.5.4, "CA Instance Configuration".

Open the URL displayed in /var/log/pki-tks-install.log in the Firefox browser.

Within the Firefox browser, click "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

 Traverse the TKS configuration panels:

- **Welcome** panel:
  - Click Next>

- **Key Store** panel:
  - Click Login for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the 'Security Module Token Password'
  - Click Next>
  - Back in the Key Store panel, select the NHSM6000-OCS Token
  - Click Next>

- **Security Domain** panel:
  - Select 'Join an Existing Security Domain and enter the URL to an existing security domain. This URL can be identified by running /sbin/service pki-ca status on Machine-1 where CA subsystem is configured.'
  - Click Next>

- **Display Certificate Chain** panel:
  - Click Next>

- **Security Domain Login** panel:
  - Supply the admin UID name and password for the CA so that it can be properly accessed.
  - Click Login

- **Subsystem Type** panel:
  - Accept the defaults.
  - Click Next>.

- **Internal Database** panel:
  - Host -- server1.example.com
  - Port -- 636 (SSL port)
  - Check the SSL box.
  - Provide the bind password of the Directory Manager (value which you gave while creating a Red Hat Directory Server instance).
  - Click Next>.

- **Key Pairs** panel:
  - Accept the defaults.
  - Select the Use the default key size (2048 bits) radio button.
  - Click Next>.

- **Subject Names** panel:
  - Accept the defaults.
  - Click Next>.
Requests and Certificates panel:
- Click Apply.
- Once the screen redraws, click **Next**.

**NOTE**

It is not possible to export keys and certificates stored on an HSM to a `.p12` file. It is also not necessary to extract keys from the HSM to clone a subsystem. The keys are already stored on the HSM and accessible to any cloned instances.

Export Keys and Certificates panel:
- Accept the defaults.
- Click **Next**.

Administrator panel:
- **UID:** admin
- **Name:** TKS Administrator of Instance pki-tks
- **Email:** pki-tks-admin@example.com
- **Password:**
- **Password (Again):**
- Click **Next**.

Import Administrator's Certificate panel:
- Click OK on the alert which says 'Your personal certificate has been installed. You should keep a backup copy of this certificate.'
- Click **Next**.

Done panel:
- Stop the TKS Subsystem by running `/sbin/service pki-tks stop`.

Setting up a Shared Key to Use with the TPS

After the TKS is configured, it must be configured to use the same shared secret (symmetric) key for their communications.

**NOTE**

If you are using a hardware token, the **tkstool** script could return an error requiring you to set environment variables before running the tool. Set the environment variables as directed, and then re-run the tool.

1. On the TKS instance, create the shared secret key which will be used to communicate with the TPS.
   The **tkstool** script prints information about the shared secret key as it creates the key. These session key shares are required to import the shared key onto the TPS, so record this information.
NOTE

Creating shared keys for the TKS and TPS is covered in the Administrator’s Guide.

```bash
tkstool -T -d /var/lib/pki-tks/alias -n sharedSecret

Generating the first session key share . . .
  first session key share:    792F AB89 8989 D902
                              9429 6137 8632 7CC4

  first session key share KCV:  D1B6 14FD
Generating the second session key share . . .
  second session key share:    4CDF C8E0 B385 68EC
                              380B 6D5E 1C19 3E5D

  second session key share KCV:  1EC7 8D4B
Generating the third session key share . . .
  third session key share:     CD32 3140 25B3 C789
                              B54F 2C94 26C4 9752

  third session key share KCV:  73D6 8633
Generating first symmetric key . . .
Generating second symmetric key . . .
Generating third symmetric key . . .
Extracting transport key from operational token . . .
  transport key KCV:  A8D0 97A2
Storing transport key on final specified token . . .
Naming transport key "sharedSecret" . . .
Successfully generated, stored, and named the transport key!
```

2. List the shared key in the TKS database to make sure it was successfully created.

```bash
tkstool  -d /var/lib/pki-tks/alias -L

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

Enter Password or Pin for "NSS Certificate DB": xxxxxxx
<0> sharedSecret
```

The default key name is `sharedSecret`. The TKS can be configured to have a different name by changing the `tks.tksSharedSymKeyName` value in the `cs.cfg`. The name given in the TKS settings must be the same as the name of the key that is installed in the TKS database or the TKS cannot locate the key.

**Enabling FIPS Mode**

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, “Using Hardware Security Modules with Subsystems” and the vendor documentation.
2. Install and configure the instance.
3. Stop the instance. The instance must be stopped to protect the information stored in its security databases.

```bash
service instance_name stop
```
4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance's security database directory.
      ```
      cd /var/lib/instance_name/alias
      ```
   b. Using `certutil`, create a request for a new SSL server certificate.
      ```
      certutil -d . -R -s "CN=server2.example.com,OU=instance_name,O=Example Domain instance_name" -o sslfips.req -h "NSS Certificate DB" -a
      ```
   c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.
   d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.
   e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as `sslfips.cert`.
   f. Check the instance's certificate database to see if an SSL server certificate is already listed.
      ```
      certutil -d /var/lib/instance_name/alias -L
      ```
   g. If the certificate exists, then delete it.
      ```
      certutil -d /var/lib/instance_name/alias -D -n "ServerCert nickname"
      ```
   h. Import the new SSL server certificate.
      ```
      certutil -d /var/lib/instance_name/alias -A -t "u,u,u" -n "ServerCert nickname" -i sslfips.cert -a
      ```
   i. Edit the `/var/lib/instance_name/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert server2.example.com - Example Domain instance_name`.
   j. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.
      ```
      vim /var/lib/instance_name/conf/CS/cfg
      ```
      ```
      type.cert.sslserver.nickname= ServerCert server2.example.com - Example Domain instance_name
      type.sslserver.nickname= ServerCert server2.example.com - Example Domain instance_name
      ```
   k. Edit the `server.xml` file to enable FIPS mode for each SSL-enabled connector. Set `strictCiphers` to true and add or set `ssl3` to false. For example:
vim /var/lib/instance_name/conf/server.xml

<Connector name="Agent" port="12443" maxHttpHeaderSize="8192"
  ... 
  sslOptions="ssl2=false,ssl3=false,tlsv1.2=true"
  strictCiphers="true"
  ... 
>

1. Enable FIPS mode in the NSS software database.
   modutil -dbdir /var/lib/instance_name/alias -fips true

m. Verify that FIPS mode has been enabled. The command will return the current FIPS status.
   modutil -dbdir /var/lib/instance_name/alias modutil -dbdir . -chkfips true
   FIPS mode enabled.

n. Start the instance.
   service instance_name start

B.2.2.8.4. Configuring POSIX System ACLs

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

NOTE

It is not necessary to apply the acl option and remount the filesystem because that was done for the DRM, and the DRM, OCSP, and TKS are on the same machine.

1. Log into the system as root.
2. Stop the TKS instance.
   service pki-tks stop

3. Set a group readability of pkiadmin for the instance’s directories and files.
   # setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki-tks

4. Apply execute (x) ACL permissions on all directories:
   # find -L /var/lib/pki-tks -type d -exec setfacl -L -n -m 
   g:pkiadmin:rx,d:g:pkiadmin:rx {} \;

5. Remove group readability for the pkiadmin group from the instance’s signedAudit/ directory and its associated files:
6. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

```bash
# setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki-tks/logs/signedAudit
```

7. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:

```bash
# find -L /var/lib/pki-tks/logs/signedAudit -type d -exec setfacl -L -n -m g:pkiaudit:rx,d:g:pkiaudit:rx {} \;
```

8. Start the TKS.

```bash
service pki-tks start
```

9. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:

```bash
# getfacl /var/lib/pki-tks /var/log/pki-tks/logs/signedAudit/
getfacl: Removing leading '/' from absolute path names
# file: var/lib/pki-tks
# owner: pkiuser
# group: pkiuser
user::rwx
group::rwx
group:pkiadmin:r-x
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:pkiadmin:r-x
default:mask::rwx
default:other::r-x

# file: var/lib/pki-tks/logs/signedAudit
# owner: pkiuser
# group: pkiaudit
user::rwx
group::rwx
group:pkiaudit:r-x
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:pkiaudit:r-x
default:mask::rwx
default:other::---
```

**B.2.2.9. TPS Instance Creation and Configuration**

**NOTE**

The "TPS Instance Creation and Configuration" section is **ONLY** performed on **Machine-3** (server3.example.com).
B.2.2.9.1. TPS Instance Creation

Use the **pkicreate** tool to create a Red Hat Certificate System instance of a TPS.

**Note**

Make certain that the "-audit_group=pkiaudit" parameter is used.

Use the pkicreate tool:

```
[rootserver3]# pkicreate -pki_instance_root=/var/lib            
   -pki_instance_name=pki-tps             
   -subsystem_type=tps                    
   -secure_port=7889                      
   -non_clientauth_secure_port=7890       
   -unsecure_port=7888                    
   -user=pkiuser                          
   -group=pkiuser                         
   -audit_group=pkiaudit                  
   -redirect conf=/etc/pki-tps            
   -redirect logs=/var/log/pki-tps        
   -verbose
```

Optional Reference: [Section 6.1, “About pkicreate”](#)

B.2.2.9.2. Hardware Security Module TPS Configuration

Follow instructions in the following procedure (substituting the name pki-tps for pki-ca'):

**Section 8.1.2.3, “Installing External Tokens and Unsupported HSM”**

B.2.2.9.3. TPS Instance Configuration

Before configuring TPS, stop pki-tps:

```
[root@server3 alias]# /sbin/service pki-tps stop
```

Now, obtain CA’s Signing certificate (in base64 format) from Machine-1 (server1.example.com) where CA is configured.

```
[root@server1 alias]# /usr/bin/certutil -L -d /var/lib/pki-ca/alias -n "caSigningCert cert-pki-ca" -a > pki-casigningcert.crt
```

Now, import CA’s Signing certificate (in base64 format) into TPS alias database:

```
[root@server3 alias]# /usr/bin/certutil -A -d /var/lib/pki-tps/alias -i pki-casigningcert.crt -n "caSigningCert cert-pki-ca" -t "CT,C,C"
```

Now, start TPS:

```
[root@server3 alias]# /sbin/service pki-tps start
```

Since **pkicreate** has been executed in [Section B.2.2.9.1, “TPS Instance Creation”](#), obtain the URL to configure the TPS by doing a `tail /var/log/pki-tps-install.log`:
Important

For the purposes of this test, the ONLY Firefox browser that will be used is the one installed on Machine-1 (server1.example.com).

Additionally, ALL subsystems mentioned in this test will ONLY use the Firefox browser profile called server created in Section B.2.2.5.4, “CA Instance Configuration”.

As an administrator, launch a Firefox browser using the profile manager:

```
[root@server3 alias]# /usr/bin/firefox -ProfileManager -no-remote
```

Select the profile entitled server that was utilized for configuration of the CA in Section B.2.2.5.4, “CA Instance Configuration”.

Open the URL displayed in /var/log/pki-tps-install.log in the Firefox browser.

Within the Firefox browser, click "I Understand the Risks" -> "Add Exception..." -> "Get Certificate" -> "Confirm security Exception".

Traverse the TPS configuration panels:

- **Welcome** panel:
  - Click Next>

- **Security Modules** panel:
  - Click Login for NHSM6000-OCS (nCipher's nFast Token Hardware Module)
  - Provide the Security Module Token Password'
  - Click Next>
  - Back in the **Security Modules** panel, select the NHSM6000-OCS Token
  - Click Next>

- **Security Domain** panel:
  - Select 'Join an Existing Security Domain and enter the URL to an existing security domain. This URL can be identified by running /sbin/service pki-ca status on Machine-1 where CA subsystem is configured.
  - Click Next>

- **Display Certificate Chain** panel:
  - Click Next>
Security Domain Login panel:
- Supply the admin name and password for the CA so that it can be properly accessed.
- Click Login

Subsystem Type panel:
- Accept the defaults.
- Click Next>

CA Information panel:
- Accept the defaults.
- Click Next>

TKS Information panel:
- Accept the defaults.
- Click Next>

DRM Information panel:
- Accept the defaults.
- Click Next>

Authentication Directory panel:
- Host -- server1.example.com.
- Port -- 636 (SSL port).
- Check the SSL box.
- Accept the default for Base DN.
- Click Next>.

Internal Database panel:
- Host -- server1.example.com.
- Port -- 636 (SSL port).
- Check the SSL box.
- Provide the bind password of the Directory Manager (value which you gave while creating a Red Hat Directory Server instance).
- Click Next>.

Key Pairs panel:
- Accept the defaults.
- Select the Use the default key size (2048 bits). radio button.
- Click Next>.

Subject Names panel:
- Accept the defaults.
- Click Next>.

Certificate Requests panel:
- Click Apply.
- Once the screen redraws, click Next>.

Administrator panel:
- UID: admin
- Name: TPS Administrator of Instance pki-tps
- Email: pki-tps-admin@example.com
- Password:
Setting up a Shared Key to Use with the TPS

After the TKS is configured, it must be configured to use the same shared secret (symmetric) key for their communications.

NOTE

If you are using a hardware token, the tkstool script could return an error requiring you to set environment variables before running the tool. Set the environment variables as directed, and then re-run the tool.

1. Stop the TPS.
   
   service pki-tps stop

2. Use the tkstool script to import the shared key into the NSS software token.
   
   tkstool -I -d /var/lib/pki-tps/alias -n sharedSecret

   The script will prompts for the session key shares that were printed when the key was created in the TKS. Enter the information from the TKS.

3. List the keys to make sure the shared key was properly imported.
   
   tkstool -I -d /var/lib/pki-tps/alias -L
   
   slot:  NSS User Private Key and Certificate Services
tenon:  NSS Certificate DB

   Enter Password or Pin for "NSS Certificate DB": xxxxx
   <0> sharedSecret

   The shared key is sharedSecret, which is the default name. The TPS can be configured to set a different name for the shared key by changing the value of the conn.tks1.tksSharedSymKeyName value in the CS.cfg. This value must be the same as the nickname for the key imported into the token, or the TPS cannot locate the key.

Enabling FIPS Mode

Then, enable FIPS mode for the HSM.

1. Set up the HSM, as described in Section 8.1.2, “Using Hardware Security Modules with Subsystems” and the vendor documentation.

2. Install and configure the TPS.
3. Stop the TPS. The instance must be stopped to protect the information stored in its security databases.

```
service pki-tps stop
```

4. Replace the SSL subsystem certificate. By default, the installation process puts the certificate on the hardware token, but it should be placed on the software FIPS token.
   a. Open the instance's security database directory.

```
cd /var/lib/pki-tps/alias
```

b. Using `certutil`, create a request for a new SSL server certificate.

```
certutil -d . -R -s "CN=tps.example.com,OU=pki-tps,O=Example Domain pki-tps" -o sslfips.req -h "NSS Certificate DB" -a
```

c. Open the end entities pages for the CA (https://server.example.com:9444/ca/ee/ca), and use the SSL Server Cert Profile to submit the request.

d. Log into the agent pages (https://server.example.com:9443/ca/agent/ca), and approve the request.

e. Copy the base 64-encoded certificate on the approval page and save it to a file, such as `sslfips.cert`.

f. Check the instance's certificate database to see if an SSL server certificate is already listed.

```
certutil -d /var/lib/pki-tps/alias -L
```

g. If the certificate exists, then delete it.

```
certutil -d /var/lib/pki-tps/alias -D -n "ServerCert nickname"
```

h. Import the new SSL server certificate.

```
certutil -d /var/lib/pki-tps/alias -A -t "u,u,u" -n "ServerCert tps.example.com - Example Domain pki-tps" -i sslfips.cert -a
```

i. Edit the `/var/lib/pki-tps/conf/serverCertNick.conf` file to contain the nickname of the new certificate, such as `ServerCert tps.example.com - Example Domain pki-tps`.

j. Edit the `CS.cfg` file to replace both references to the SSL server certificate nickname.

```
 vim /var/lib/pki-tps/conf/CS/cfg

tps.cert.sslserver.nickname= ServerCert server3.example.com - Example Domain pki-tps

tps.sslserver.nickname= ServerCert server3.example.com - Example Domain pki-tps
```

5. The `nss.conf` file for the TPS contains a list of virtual servers for the different SSL listening ports for the TPS. Every SSL port must be configured to run under FIPS:
   a. Turn on FIPS.
# FIPS Switch:
# Enable/Disable FIPS mode
NSSFIPS on

b. Change the **NSSNickname** to reflect the new TPS server certificate. For example:

```bash
# SSL Certificate Nickname:
# The nickname of the server certificate you are going to use.
NSSNickname "ServerCert server3.example.com - Example Domain pki-tps"
```

c. Uncomment the FIPS cipher suite directive and comment out the standard directive.

```bash
# SSL Cipher Suite:
# List the ciphers that the client is permitted to negotiate.
# See the mod_nss documentation for a complete list.
#NSSCipherSuite -des,-desede3,-rc2,-rc2export,-rc4,-rc4export,+rsa_3des_sha,-rsa_des_56_sha,-rsa_null_md5,-rsa_null_sha,-rsa_rc2_40_md5,+rsa_rc4_128_md5,-rsa_rc4_128_sha,-rsa_rc4_40_md5,-rsa_rc4_56_sha,-fortezza,-fortezza_rc4_128_sha,-fortezza_null,-fips_des_sha,+fips_3des_sha,-rsa_des_128_sha,-rsa_aes_256_sha,+ecdhe_ecdsa_aes_256_sha

# SSL cipher suite in FIPS mode:
NSSCipherSuite +rsa_3des_sha,-rsa_des_128_sha,-rsa_rc2_40_md5,-rsa_rc4_40_md5,-rsa_null_md5,-rsa_null_sha,+fips_3des_sha,-fips_des_sha,-fortezza,-fortezza_rc4_128_sha,-fortezza_null,-rsa_des_56_sha,-rsa_rc4_56_sha,+rsa_aes_128_sha,+rsa_aes_256_sha
```

d. Turn SSLv3 off for every SSL port, so that only TLSv1 is enabled.

NSSProtocol TLSv1

e. For each SSL virtual host, turn off the setting to enforce valid certificates:

NSSEnforceValidCerts off

6. Add or edit the line for the NSS FIPS database to the TPS's **password.conf** file.

```bash
vim /var/lib/pki-tps/conf/password.conf

NSS FIPS 140-2 Certificate DB:mypassword
```

7. Enable FIPS mode in the NSS software database.

```bash
modutil -dbdir /var/lib/pki-tps/alias -fips true
```

8. Verify that FIPS mode has been enabled. The command will return the current FIPS status.

```bash
modutil -dbdir /var/lib/pki-tps/alias modutil -dbdir . -chkfips true
FIPS mode enabled.
```

9. Start the TPS.
service pki-tps start

**NOTE**

Hit the enter key to complete the startup process or the TPS will not start.

Optional reference to learn about the Enterprise Security Client: ESC Guide

### B.2.2.9.4. Configuring POSIX System ACLs

POSIX system access control rules provide finer granularity over system user permissions. These ACLs must be set for each instance after it is fully configured.

1. Log into the system as root.
2. Stop all Certificate System subsystems.

   ```bash
   service instance_name stop
   ```

3. Make sure that the `acl` package is installed on the system.

   ```bash
   # rpm -q acl
   acl-2.2.39-6.el5
   ```

4. Four filesystem paths must be configured to use the `acl` package:

   - `/#`
   - `/etc` (including `/etc/init.d` and `/etc/sysconfig`)
   - `/usr` (including `/usr/bin` and `/usr/share`)
   - `/var` (including `/var/lock` and `/var/run`)

   First, check to see where the filesystems are mounted and whether the `acl` option is used:

   ```bash
   # mount
   /dev/sda1 on / type ext3 (rw)
   proc on /proc type proc (rw)
   sysfs on /sys type sysfs (rw)
   devpts on /dev/pts type devpts (rw,gid=5,mode=620)
   /dev/sdb1 on /usr type ext2 (rw)
   tmpfs on /dev/shm type tmpfs (rw)
   none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
   sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
   ```

   In this example, `/`, `/etc`, and `/var` are mounted on `/dev/sda1`, while `/usr` is mounted on `/dev/sdb1`.

   There are two ways to apply the `acl` option to the filesystems:

   - **Using `tune2fs`:**
     ```bash
     # /sbin/tune2fs -o +acl /dev/sda1
     # /sbin/tune2fs -o +acl /dev/sdb1
     ```

   - **Editing the `/etc/fstab` file:**
# vim /etc/fstab

```
LABEL=/           /             ext3    defaults, acl  1 1
LABEL=/usr        /usr          ext2    defaults, acl  1 2
tmpfs             /dev/shm      tmpfs   defaults        0 0
devpts            /dev/pts      devpts  gid=5, mode=620  0 0
sysfs             /sys          sysfs   defaults        0 0
proc              /proc         proc    defaults        0 0
LABEL=SWAP-sdb2   swap          swap    defaults        0 0
LABEL=SWAP-sda2   swap          swap    defaults        0 0
```

Then, remount the filesystems.

```
# mount -o remount /
# mount -o remount /usr
```

Confirm that the **acl** option has been applied to the filesystems:

```
# /sbin/dumpe2fs /dev/sda1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:  user_xattr acl

# /sbin/dumpe2fs /dev/sdb1 | grep acl
dumpe2fs 1.39 (29-May-2006)
Default mount options:  acl

# mount | grep acl
/dev/sda1 on / type ext3 (rw,acl)
/dev/sdb1 on /usr type ext2 (rw,acl)
```

5. Stop the TPS instance.

```
service pki-tps stop
```

6. Set the group readability to the pkiadmin group for the instance's directories and files.

```
# setfacl -R -L -m g:pkiadmin:r,d:g:pkiadmin:r /var/lib/pki-tps
```

7. Apply execute (x) ACL permissions on all directories:

```
# find -L /var/lib/pki-tps -type d -exec setfacl -L -n -m g:pkiadmin:rx,d:g:pkiadmin:rx {} \;
```

8. Remove group readability for the pkiadmin group from the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -x g:pkiadmin,d:g:pkiadmin /var/lib/pki-tps/logs/signedAudit
```

9. Set group readability for the pkiaudit group for the instance's signedAudit/ directory and its associated files:

```
# setfacl -R -L -m g:pkiaudit:r,d:g:pkiaudit:r /var/lib/pki-tps/logs/signedAudit
```

10. Re-apply execute (x) ACL permissions on the signedAudit/ directory and all of its subdirectories:
11. Start the TPS.

```
service pki-tps start
```

12. Confirm that the file access controls were properly applied by using the `getfacl` command to show the current ACL settings:

```
# getfacl /var/lib/pki-tps/logs/signedAudit
getfacl: Removing leading '/' from absolute path names
# file: var/lib/pki-tps
# owner: pkiuser
# group: pkiuser
user::rwx
group::rwx
group:pkiadmin:r-x
mask::rwx
other::r-x
default:user::rwx
default:group::rwx
default:group:pkiadmin:r-x
default:mask::rwx
default:other::r-x

# file: var/lib/pki-tps/logs/signedAudit
# owner: pkiuser
# group: pkiaudit
user::rwx
group::rwx
group:pkiaudit:r-x
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:pkiaudit:r-x
default:mask::rwx
default:other::---
```

**B.2.2.10. Post Configuration Instructions**

**B.2.2.10.1. Agent, Administrator, and Audit User Creation**

The setup process only creates a single administrative user account. To manage a PKI environment in real life requires a larger number of administrators, agents, and auditors which are created after the instance is set up.

Login to `server1.example.com` as an administrator, and launch `pki-console` to create the users:

```
[root@server1]# /usr/bin/pkiconsole https://server1.example.com:9445/ca
```

Detailed info on Red Hat Certificate System users that needs to be created:

These users are to be created as appropriate for relevant Certificate System subsystems including CA, TKS, DRM, OCSP, and TPS.
» For **Admin** users for CA, TKS, OCSP, and DRM:
  - adminV (this user will have a valid certificate throughout testing)
  - adminR (this user will have a revoked certificate)
  - adminE (this user will have an expired certificate)
  - adminTCA (this user's certificate will be signed by a trusted CA)
  - adminC (initially this user will be assigned to the administrator group; during the testing process this user will be assigned to a different group)
  - adminUTCA (this user's certificate will be issued by an untrusted CA) [Note: This requires a second CA setup!]

General procedure for 'admin' certificates: (Example for user **adminV**)

- For CA, TKS, DRM and OCSP:
  - Create users and add them to the Administrator group
  - Create a user certificate
  - ADD_CERTIFICATE to ~/.redhat-idm-console

```bash
# Backup the role user certificate from the browser in adminV.p12 file
[root@server1]# cd ~/.redhat-idm-console
[root@server1]# /usr/bin/pk12util -d ~/.redhat-idm-console -i adminV.p12
```

- For TPS:
  - Create users with an Admin role
  - ADD_CERTIFICATE to TPS users:
    - Copy the base 64 encoded blob (without the header and footer) of the role user certificate

```
Visit TPS UI Admin tab with TPS admin credential.
List users and select user-name.
Paste the certificate base 64 blob.
Click update.
```

- Add profile access to TPS users:

```
Visit TPS UI Admin tab with TPS admin credential.
List users and select user-name.
Add new profile, select "All Profiles".
Click Add profile.
```

» For **Agent** users:
  - agentV (this user will have a valid certificate throughout testing)
  - agentR (the status of this user's certificate is Revoked)
  - agentE (this user's certificate is in the expired state)
  - adminTCA (this user's certificate will be signed by a trusted CA)
  - agentTCA (this user's certificate will be issued by a trusted CA))
  - agentUTCA (this user's certificate will be issued by an untrusted CA) [Note: This requires a second CA setup!]

General procedure for agent certificates:

- For CA, TKS, DRM, and OCSP:
Add a user to the Certificate Manager Agents group
Create a user certificate
ADD_CERTIFICATE to ~/.redhat-idm-console

# Backup the role user certificate from the browser in agentV.p12 file
[root@server1]# cd ~/.redhat-idm-console
[root@server1]# /usr/bin/pk12util -d ~/.redhat-idm-console -i agentV.p12

For TPS:
Create a user and assign the Agent role
ADD_CERTIFICATE to TPS users:
  Copy the base 64 encoded blob (without the header and footer) of the role user certificate

Visit TPS UI Admin tab with TPS admin credential.
List users and select user-name.
Paste the certificate base 64 blob.
Click update.

Add profile access to TPS users:

Visit TPS UI Admin tab with TPS admin credential.
List users and select user-name.
Add new profile, select "All Profiles".
Click Add profile.

For Audit users:
auditV (this user will have a valid certificate throughout testing)
General procedure for 'audit' certificates: (Example for user auditV) for CA, TKS, DRM and OCSP do the following steps:
ADD_CERTIFICATE to ~/.redhat-idm-console

# Backup the role user certificate from the browser in auditV.p12 file
[root@server1]# cd ~/.redhat-idm-console
[root@server1]# /usr/bin/pk12util -d ~/.redhat-idm-console -i auditV.p12

For TPS Operator users:
operatorV (this user will have a valid certificate throughout testing)
General procedure for 'operator' certificates: (Example for user operatorV)
Create a user with the Operator role
ADD_CERTIFICATE to TPS users:
  Copy the base 64 encoded blob (without the header and footer) of the role user certificate

Visit TPS UI Admin tab with TPS admin credential.
List users and select user-name.
Paste the certificate base 64 blob.
Click update.

Add profile access to TPS users:
Visit TPS UI Admin tab with TPS admin credential. List users and select user-name. Add new profile, select "All Profiles". Click Add profile.

To Revoke Certificates
To Achieve Expired Certificates

**Expired Certificates**

From CA EE pages, submit enrollment requests for all of these users. From the CA Agent port, approve these requests. When approving expired certs, remember to change notAfter to point to 1-day.

Optional Reference: [Managing Users and Groups for a CA, OCSP, DRM, or TKS](#)

### B.2.2.10.2. Import of Certificate Chain into Browser

Any browser which will be used to access agent or admin services pages must have the client certificate and the CA certificate chain imported into its trust store.

**Note**

Since this test uses a common browser profile for its CA, DRM, OCSP, TKS, and TPS instances, the following steps will *not* be necessary as the certificate chain was imported during creation of the CA as documented in [Section B.2.2.5.4, "CA Instance Configuration"](#). However, if different browser profiles had been used for its DRM, OCSP, TKS, or TPS instances, the following steps would have been necessary.

To import the certificate chain into the browser, perform the following steps:

- Select the URL that points to the Secure End-Entity Port of the previously configured CA and open the browser to this URL.

**Note**

This URL can be identified by running `/sbin/service pki-ca status`.

- Click the tab labeled **Retrieval**
- Select the URL called **Import CA Certificate Chain**
- Select the radio button for **Import the CA certificate chain into your browser**
- Click the **submit** button.
- Click OK for the prompt ‘You will now be asked to import and trust the Certificate Chain from the CA. Please do so.’
- Select the checkboxes for:
  - Check ‘Trust this CA to identify web sites’
  - Check ‘Trust this CA to identify email users’
Check ‘Trust this CA to identify software developers’

Click OK to dismiss this dialog box

Optional Reference: [Downloading CA Certificates and Certificate Chains](#)

B.2.2.10.3. Sudo Permissions for PKI Services

Granting sudo permissions for Certificate System and Directory Server services to PKI administrators makes it easier to manage the instances.

To grant these sudo permissions, execute the following command:

```
[root@server1]# /usr/sbin/visudo
```

Add the following lines to the `/etc/sudoers` file:

```
# For Directory Server services
%pkiadmin ALL = PASSWD: /sbin/service dirsrv *
%pkiadmin ALL = PASSWD: /sbin/service dirsrv-admin *

# For PKI instance management
%pkiadmin ALL = PASSWD: /usr/bin/pkicreate *
%pkiadmin ALL = PASSWD: /usr/bin/pkiremove *

# For PKI instance services
%pkiadmin ALL = PASSWD: /sbin/service pki-ca *
%pkiadmin ALL = PASSWD: /sbin/service pki-kra *
%pkiadmin ALL = PASSWD: /sbin/service pki-ocsp *
%pkiadmin ALL = PASSWD: /sbin/service pki-tks *
%pkiadmin ALL = PASSWD: /sbin/service pki-tps *
```

Optional Reference: [Resetting sudo Permissions for Certificate System Services](#)

B.2.2.10.4. Backup and Restore Procedures

A reliable backup and restore strategy is vital for disaster and data recovery.

**Note**

For the purposes of this test, no backup and recovery mechanism will be utilized.

Optional Reference: [Backing up and Restoring Certificate System](#)

B.2.2.10.5. Signed Audit

This can be done when an instance is first created by using the -audit_group option, but it can also be done post-installation.

For the CA, DRM, OCSP, or TKS instance, the following procedural reference can be followed to enable signed auditing:

Procedural Reference: [Managing Signed Audit Logs](#)

This procedure is mandatory to enable signed auditing for a TPS subsystem:
Procedural Reference: Managing Audit Logs

B.2.2.10.6. Revocation Checking and OCSP Responders
There are two methods for checking whether a certificate is active (and not revoked): CRL checking and OCSP services. Checking the revocation status helps ensure that a revoked certificate isn't wrongly accepted just because it is within its validity period.

The following procedural reference can be followed to enable revocation checking:

Procedural Reference: Enabling Revocation Checking

Similarly, the following procedural reference can be followed to establish an OCSP responder:


B.2.2.10.7. SSL Session Timeouts
Using session timeouts prevents unauthorized users from accessing an open, but otherwise inactive, session with a subsystem.

**Note**

For the purposes of this test, the default SSL session timeouts for the CA, DRM, OCSP, TKS, and TPS instances should be sufficient.

To alter the default behavior of this feature, follow the procedural reference for the CA, DRM, OCSP, TKS, or TPS instances:

Procedural Reference: Setting SSL Session Timeouts

B.2.2.10.8. Self-Tests Execution
Self-tests are subsystem type-specific diagnostics. These are usually run when the server starts, but they can also be run on demand. Additionally, self-test can be customized by creating new self-test plugins or to accommodate changes to the subsystem configuration like new certificates or changed certificate nicknames.

**Note**

For the purposes of this test, all required instance-specific self-tests were automatically registered and configured when each instance was installed.

Optional Reference: Running Self-Tests

B.2.2.10.9. Removal of password.conf
The password.conf file stores system passwords in plaintext.

Remove the /var/lib/pki-ca/password.conf file to prompt for passwords, restart the pki-ca server, and provide the passwords prompted. The following example is for the CA.
Important

Similar changes need to be done for all other PKI subsystem instances!

```
[root@server1]# pwd
/var/lib/pki-ca/conf

[root@server1]# rm -rf password.conf

[root@server1]# /sbin/service pki-ca restart
Stopping pki-ca: ..                                    [  OK  ]
Starting pki-ca:
    Using Java Security Manager
    Constructing 'pki-ca.policy' Security Policy
Please provide password for internal:
Please provide password for hardware-nethsm2k:
Please provide password for internaldb:
Please provide password for replicationdb:
Starting pki-ca:                                        [  OK  ]
pki-ca (pid 26928) is running ...

    Unsecure Port       = http://server1.example.com:9180/ca/ee/ca
    Secure Agent Port   = https://server1.example.com:9443/ca/agent/ca
    Secure EE Port      = https://server1.example.com:9444/ca/ee/ca
    Secure Admin Port   = https://server1.example.com:9445/ca/services
    EE Client Auth Port = https://server1.example.com:9446/ca/eeca/ca
    PKI Console Port    = pkiconsole https://server1.example.com:9445/ca
    Tomcat Port         = 9701 (for shutdown)

    PKI Instance Name:  pki-ca
    PKI Subsystem Type: Root CA (Security Domain)

    Registered PKI Security Domain Information:
    ==============================================================
    Name: Example Domain
    URL: https://server1.example.com:9445
    ==============================================================
```
Apply this same process to the DRM, OCSP, and TKS instances. For the TPS, after moving the password.conf file out of the way, there are a couple of more quick steps required.

- Edit the file (/var/lib/pki-tps/conf/nss.conf).
- Make the following changes to related settings as follows:

```plaintext
#   Pass Phrase Dialog:
#   Configure the pass phrase gathering process.
#   The filtering dialog program ("builtin" is a internal
terminal dialog) has to provide the pass phrase on stdout.
#NSSPassPhraseDialog  builtin
#NSSPassPhraseDialog  defer:/var/lib/pki-tps/conf/password.conf

#   Pass Phrase Helper:
#   This helper program stores the token password pins between
#   restarts of Apache.
#NSSPassPhraseHelper /usr/share/pki/tps/scripts/nss_pcache
```

Enabling the password dialog means the server not longer references the password.conf file. The system prompts the operator to manually type in the needed passwords.

Optional Reference: System Passwords

For TPS system in FIPS mode only, there is an issue with manually entering in an incorrect tokendb bind password. Unlike the other subsystems, which will continually ask for a new password if the one entered is incorrect, the TPS will only appear to start successfully after entering an incorrect password. The server in fact is only in limp mode and will not execute any commands successfully. After the TPS start, commands fail inexplicably, simply restart the TPS.

### B.2.2.10.10. Removal of Unused Interfaces from web.xml

Several legacy interfaces are included in the CA's web.xml. While these can be used for custom or legacy applications to connect to the CA, for secure environments, these deprecated interfaces should be removed from the CA configuration to prevent unauthorized access.

To perform this step, please refer to the following procedural reference:

Procedural Reference: Removing Unused Interfaces from web.xml

### B.2.2.10.11. Downloading dumpasn1 tool

Download dumpasn1 from the url below; compile it with gcc; then copy it to /usr/local/bin:
B.2.2.10.12. Disable User/Group "multiroles" Support for Java Subsystems

By default, the Certificate System Console for Java Subsystems supports the notion of allowing given users the ability to reside within more than one group at a time. Ultimately this property makes it possible for a single user to assume more than one distinct role. This default behavior is fine for normal processing. The CC setup requires that we configure the systems to not allow this multirole behavior.

For CA, KRA, TKS, and OCSP subsystems:

1. Stop the server:

   ```bash
dashboard@server1:~ # service pki-ca stop
   ```

2. Open the `CS.cfg` file:

   ```bash
dashboard@server1:~ # vi /var/lib/pki-ca/conf/CS.cfg
   ```

3. Modify the current setting of `multiroles.enable=true` and change it to `multiroles.enable=false`.

4. Restart the server:

   ```bash
dashboard@server1:~ # service pki-ca restart
   ```


This section is taken directly from the security target document. This section includes the following:

- Secure usage assumptions
- Threats
- Organizational security policies

This information provides the basis for the security objectives specified in Section B.4, “Common Criteria: Security Objectives” and the security functional requirements for the target of evaluation specified in Section B.5, “Common Criteria: Security Requirements”.

B.3.1. Secure Usage Assumptions

The usage assumptions are organized in three categories: personnel (assumptions about administrators and users of the system as well as any threat agents), physical (assumptions about the physical location of the target of evaluation or any attached peripheral devices), and connectivity (assumptions about other IT systems that are necessary for the secure operation of the target of evaluation).

B.3.1.1. Personnel Assumptions

**Auditors Review Audit Logs**

Audit logs are required for security-relevant events and must be reviewed by the auditors.
Authentication Data Management

An authentication data management policy is enforced to ensure that users change their authentication data at appropriate intervals and to appropriate values, such as proper lengths, histories, and variations. This assumption is not applicable to biometric authentication data.

Competent Administrators, Operators, Officers, and Auditors

Competent administrators, operators, officers, and auditors will be assigned to manage the target of evaluation and the security of the information it contains.


All administrators, operators, officers, and auditors are familiar with the certificate policy (CP) and certification practices statement (CPS) under which the target of evaluation is operated.

Disposal of Authentication Data

Proper disposal of authentication data and associated privileges is performed after access has been removed, such as for a job termination or a change in responsibility.

Malicious Code Not Signed

Malicious code destined for the target of evaluation is not signed by a trusted entity.

Notify Authorities of Security Issues

Administrators, operators, officers, auditors, and other users notify proper authorities of any security issues that impact their systems to minimize the potential for the loss or compromise of data.

Social Engineering Training

General users, administrators, operators, officers and auditors are trained in techniques to thwart social engineering attacks.

Cooperative Users

Users need to accomplish some task or group of tasks that require a secure IT environment. The users require access to at least some of the information managed by the target of evaluation and are expected to act in a cooperative manner.

B.3.1.2. Physical Assumptions

Communications Protection

The system is adequately physically protected against loss of communications i.e., availability of communications.

Physical Protection

The target of evaluation hardware, software, and firmware critical to security policy enforcement will be protected from unauthorized physical modification.

B.3.1.3. Connectivity Assumptions

Operating System

The operating system has been selected to provide the functions required by this CIMC to counter the
perceived threats for the appropriate Security Level identified in this family of PPs.

B.3.1.4. Threats
The threats are organized in four categories: authorized users, system, cryptography, and external attacks.

B.3.1.4.1. Authorized Users

Administrative Errors of Omission
Administrators, operators, officers, or auditors fail to perform some function essential to security.

User Abuses Authorization to Collect or Send Data
User abuses granted authorizations to improperly collect or send sensitive or security-critical data.

User Error Makes Data Inaccessible
User accidentally deletes user data rendering user data inaccessible.

Administrators, Operators, Officers, or Auditors Commit Errors or Hostile Actions
An administrator, operator, officer, or auditor commits errors that change the intended security policy of the system or application or maliciously modify the system's configuration to allow security violations to occur.

B.3.1.4.2. System

Critical System Component Fails
Failure of one or more system components results in the loss of system critical functionality.

Malicious Code Exploitation
An authorized user, IT system, or hacker downloads and executes malicious code, which causes abnormal processes that violate the integrity, availability, or confidentiality of the system assets.

Message Content Modification
A hacker modifies information that is intercepted from a communications link between two unsuspecting entities before passing it on to the intended recipient.

Flawed Code
A system or applications developer delivers code that does not perform according to specifications or contains security flaws.

B.3.1.4.3. Cryptography

Disclosure of Private and Secret Keys
A private or secret key is improperly disclosed.

Modification of Private/Secret Keys
A secret/private key is modified.
**Sender Denies Sending Information**

The sender of a message denies sending the message to avoid accountability for sending the message and for subsequent action or inaction.

**B.3.1.4.4. External Attacks**

**Hacker Gains Access**

A hacker masquerades as an authorized user to perform operations that will be attributed to the authorized user or a system process or gains undetected access to a system due to missing, weak, or incorrectly implemented access control causing potential violations of integrity, confidentiality, or availability.

**Hacker Gains Physical Access**

A hacker physically interacts with the system to exploit vulnerabilities in the physical environment, resulting in arbitrary security compromises.

**Social Engineering**

A hacker uses social engineering techniques to gain information about system entry, system use, system design, or system operation.

**B.3.2. Organization Security Policies**

**Authorized Use of Information**

Information shall be used only for its authorized purpose(s).

**Cryptography**

FIPS-approved or NIST-recommended cryptographic functions shall be used to perform all cryptographic operations.


This section is taken directly from the security target document. This section includes the security objectives including security objectives for the target of evaluation, security objectives for the environment, and security objectives for both the target of evaluation and environment.

**B.4.1. Security Objectives for the Target of Evaluation**

This section includes the security objectives for the target of evaluation, divided among four categories: authorized users, system, cryptography, and external attacks.

**B.4.1.1. Authorized Users**

**O. Certificates**

The target security functions must ensure that certificates, certificate revocation lists, and certificate status information are valid.

**B.4.1.2. System**

**O. Preservation/Trusted Recovery of Secure state**
Preserve the secure state of the system in the event of a secure component failure or recover to a secure state.

B.4.1.3. Cryptography

O. Non-Repudiation

Prevent user from avoiding accountability for sending a message by providing evidence that the user sent the message.

B.4.1.4. External Attacks

O. Control Unknown Source Communication Traffic

Control (such as reroute or discard) communication traffic from an unknown source to prevent potential damage.

B.4.2. Security Objectives for the Environment

This section specifies the security objectives for the environment.

O. Administrators, Operators, Officers, and Auditors Guidance Documentation

Deter administrator, operator, officer, or auditor errors by providing adequate documentation on securely configuring and operating the CIMC.

O. Auditors Review Audit Logs

Identify and monitor security-relevant events by requiring auditors to review audit logs on a frequency sufficient to address level of risk.

O. Authentication Data Management

Ensure that users change their authentication data at appropriate intervals and to appropriate values, such as proper lengths, histories, and variations through enforced authentication data management. This objective is not applicable to biometric authentication data.

O. Communications Protection

Protect the system against a physical attack on the communications capability by providing adequate physical security.

O. Competent Administrators, Operators, Officers and Auditors

Provide capable management of the target of evaluation by assigning competent administrators, operators, officers, and auditors to manage the target of evaluation and the security of the information it contains.


All administrators, operators, officers, and auditors shall be familiar with the certificate policy (CP) and the certification practices statement (CPS) under which the target of evaluation is operated.

O. Disposal of Authentication Data

Provide proper disposal of authentication data and associated privileges after access has been removed, such as for a job termination or a change in responsibility.
O. Installation

Those responsible for the target of evaluation must ensure that the target of evaluation is delivered, installed, managed, and operated in a manner which maintains IT security.

O. Malicious Code Not Signed

Protect the target of evaluation from malicious code by ensuring all code is signed by a trusted entity prior to loading it into the system.

O. Notify Authorities of Security Issues

Notify proper authorities of any security issues that impact their systems to minimize the potential for the loss or compromise of data.

O. Physical Protection

Those responsible for the target of evaluation must ensure that the security-relevant components of the target of evaluation are protected from physical attack that might compromise IT security.

O. Social Engineering Training

Provide training for general users, administrators, operators, officers, and auditors in techniques to thwart social engineering attacks.

O. Cooperative Users

Ensure that users are cooperative so that they can accomplish some task or group of tasks that require a secure IT environment and information managed by the target of evaluation.

O. Lifecycle Security

Provide tools and techniques used during the development phase to ensure security is designed into the CIMC. Detect and resolve flaws during the operational phase.

O. Repair Identified Security Flaws

The vendor repairs security flaws that have been identified by a user.

O. Cryptographic Functions

The target of evaluation must implement approved cryptographic algorithms for encryption/decryption, authentication, and signature generation/verification; approved key generation techniques and use validated cryptographic modules. (Validated is defined as FIPS 140-2 validated.)

O. Operating System

The operating system used is validated to provide adequate security, including domain separation and non-bypassability, in accordance with security requirements recommended by the National Institute of Standards and Technology.

O. Periodically Check Integrity

Provide periodic integrity checks on both system and software.

O. Security Roles

Maintain security-relevant roles and the association of users with those roles.
O. Social Engineering Training

Provide training for general users, administrators, operators, officers, and auditors in techniques to thwart social engineering attacks.

O. Sufficient Backup Storage and Effective Restoration

Provide sufficient backup storage and effective restoration to ensure that the system can be recreated.

O. Validation of Security Function

Ensure that security-relevant software, hardware, and firmware are correctly functioning through features and procedures.

O. Trusted Path

Provide a trusted path between the user and the system. Provide a trusted path to security-relevant (target security functions) data in which both end points have assured identities.

B.4.3. Security Objectives for Both the Target of Evaluation and the Environment

This section specifies the security objectives that are jointly addressed by the target of evaluation and the environment. While normally security objectives are assigned to the target of evaluation or the environment, but not both, the CIMC-BR-PP introduces this set of jointly addressed security objectives. That information serves to differentiate the specific expectations for the target of evaluation and its environment relative to these objectives.

O. Configuration Management

Implement a configuration management plan. Implement configuration management to assure identification of system connectivity (software, hardware, and firmware), and components (software, hardware, and firmware), auditing of configuration data, and controlling changes to configuration items.

O. Data Import/Export

Protect data assets when they are being transmitted to and from the target of evaluation, either through intervening untrusted components or directly to/from human users.

O. Detect Modifications of Firmware, Software, and Backup Data

Provide integrity protection to detect modifications to firmware, software, and backup data.

O. Individual Accountability and Audit Records

Provide individual accountability for audited events. Record in audit records: date and time of action and the entity responsible for the action.

O. Integrity Protection of User Data and Software

Provide appropriate integrity protection for user data and software.

O. Limitation of Administrative Access

Design administrative functions so that administrators, operators, officers, and auditors do not automatically have access to user objects, except for necessary exceptions. Control access to the system by operators and administrators who troubleshoot the system and perform system updates.

O. Maintain User Attributes
Maintain a set of security attributes — which may include role membership or access privileges — associated with individual users. This is in addition to user identity.

**O. Manage Behavior of Security Functions**

Provide management functions to configure, operate, and maintain the security mechanisms.

**O. Object and Data Recovery Free from Malicious Code**

Recover to a viable state after malicious code is introduced and damage occurs. That state must be free from the original malicious code.

**O. Procedures for Preventing Malicious Code**

Incorporate malicious code prevention procedures and mechanisms.

**O. Protect Stored Audit Records**

Protect audit records against unauthorized access, modification, or deletion to ensure accountability of user actions.

**O. Protect User and Target Security Functions Data During Internal Transfer**

Ensure the integrity of user and target security functions data transferred internally within the system.

**O. Require Inspection for Downloads**

Require inspection of downloads/transfers.

**O. Respond to Possible Loss of Stored Audit Records**

Respond to possible loss of audit records when audit trail storage is full or nearly full by restricting auditable events.

**O. Restrict Actions Before Authentication**

Restrict the actions a user may perform before the target of evaluation authenticates the identity of the user.

**O. Security-Relevant Configuration Management**

Manage and update system security policy data and enforcement functions, and other security-relevant configuration data, to ensure they are consistent with organizational security policies.

**O. Timestamps**

Provide timestamps to ensure that the sequencing of events can be verified.

**O. User Authorization Management**

Manage and update user authorization and privilege data to ensure they are consistent with organizational security and personnel policies.

**O. React to Detected Attacks**

Implement automated notification (or other responses) to the target security functions-discovered attacks in an effort to identify attacks and to create an attack deterrent.

This section is taken directly from the security target document. This section identifies the security functional and assurance requirements applicable to the target of evaluation. Among the security functional requirements are extended components that have been explicitly defined within the CIMC-BR-PP as indicated below.

B.5.1. Security Requirements for the IT Environment

This security target includes a number of explicitly stated requirements. Each of the explicitly stated requirements is defined in the CIMC-BR-PP and corresponding rationale immediately follows the statement of each such requirement. The explicitly stated requirements can be identified by the use of the keyword “CIMC” in the requirement component and element identifiers. These are identified in bold-italics in Table B.2, “CIMC Target of Evaluation Functional Security Requirements”.

The CIMC-BR-PP should be referenced for the initial definition and rationale for each applicable requirement, though some rationale has been reproduced for each extended component included in the subsequent section as well.

B.5.2. Target of Evaluation Security Functional Requirements

This section specifies the security requirements that are applicable to CIMC functionality, such as key management, certificate registration, and CIMC configuration and management functions.

Note that the iteration identifiers in this section are consistent with the CIMC-BR-PP and as such include reference to those CIMC-BR-PP requirements for the IT environment (as opposed to the target of evaluation) reproduced in section and not necessarily considered requirements in the context of this security target.
<table>
<thead>
<tr>
<th>Security Functional Class</th>
<th>Security Functional Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Audit (FAU)</td>
<td>FAU_GEN.1 Audit data generation (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FAU_GEN.2 User identity association (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FAU_SEL.1 Selective audit (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FAU_STG.1 Protected audit trail storage (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FAU_STG.4 Prevention of audit data loss (iteration 2)</td>
</tr>
<tr>
<td>Communication (FCO)</td>
<td>FCO_NRO_CIMC.3 Enforced proof of origin and verification of origin</td>
</tr>
<tr>
<td></td>
<td>FCO_NRO_CIMC.4 Advanced verification of origin</td>
</tr>
<tr>
<td>Cryptographic support (FCS)</td>
<td>FCS_CKM_CIMC.5 CIMC private and secret key zeroization</td>
</tr>
<tr>
<td></td>
<td>FCS_SOF_CIMC.1 CIMC Strength of Functions</td>
</tr>
<tr>
<td>User Data Protection (FDP)</td>
<td>FDP_ACC.1 Subset access control (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FDP_ACF.1 Security attribute based access control (iteration 2)</td>
</tr>
<tr>
<td></td>
<td>FDP_ACF_CIMC.2 User private key confidentiality protection</td>
</tr>
<tr>
<td></td>
<td>FDP_ACF_CIMC.3 User secret key confidentiality protection</td>
</tr>
<tr>
<td></td>
<td>FDP_CIMC_CER.1 Certificate Generation</td>
</tr>
<tr>
<td></td>
<td>FDP_CIMC_CRL.1 Certificate Revocation</td>
</tr>
<tr>
<td></td>
<td>FDP_CIMC_CSE.1 Certificate status export</td>
</tr>
<tr>
<td></td>
<td>FDP_CIMC_OCSP.1 Basic Response Validation</td>
</tr>
<tr>
<td></td>
<td>FDP_ETC_CIMC.5 Extended user private and secret key export</td>
</tr>
<tr>
<td></td>
<td>FDP_ITT.1 Basic internal transfer protection (iterations 3 and 4)</td>
</tr>
<tr>
<td></td>
<td>FDP_SD_CIMC.3 Stored public key integrity monitoring and action</td>
</tr>
<tr>
<td>FIA</td>
<td>Identification and authentication (FIA)</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>FIA_UAU.1</td>
<td>Timing of authentication (iteration 2)</td>
</tr>
<tr>
<td>FIA_UID.1</td>
<td>Timing of identification (iteration 2)</td>
</tr>
<tr>
<td>FIA_USB.1</td>
<td>User-subject binding (iteration 2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FMT</th>
<th>Security management (FMT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMT_MOF.1</td>
<td>Management of security functions behavior (iteration 2)</td>
</tr>
<tr>
<td><strong>FMT_MOF_CIMC.3</strong></td>
<td>Extended certificate profile management</td>
</tr>
<tr>
<td><strong>FMT_MOF_CIMC.5</strong></td>
<td>Extended certificate revocation list profile management</td>
</tr>
<tr>
<td><strong>FMT_MOF_CIMC.6</strong></td>
<td>OCSP Profile Management</td>
</tr>
<tr>
<td><strong>FMT_MTD_CIMC.4</strong></td>
<td>TSF private key confidentiality protection</td>
</tr>
<tr>
<td><strong>FMT_MTD_CIMC.5</strong></td>
<td>TSF secret key confidentiality protection</td>
</tr>
<tr>
<td><strong>FMT_MTD_CIMC.7</strong></td>
<td>Extended TSF private and secret key export</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FPT</th>
<th>Protection of the target security functions (FPT)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FPT_CIMC_TSP.1</strong></td>
<td>Audit log signing event</td>
</tr>
<tr>
<td>FPT_ITC.1</td>
<td>Inter-TSF confidentiality during transmission (iteration 2)</td>
</tr>
<tr>
<td>FPT_ITT.1</td>
<td>Basic internal TSF data transfer protection (iterations 3 and 4)</td>
</tr>
<tr>
<td>FPT_RVM.1</td>
<td>Non-bypassability of the TSP (iteration 2)</td>
</tr>
<tr>
<td>FPT_STM.1</td>
<td>Reliable time stamps (iteration 2)</td>
</tr>
</tbody>
</table>

**B.5.2.1. Security Audit (FAU)**

**FAU_GEN.1 Audit data generation (iteration 2)**

**FAU_GEN.1.1**

The target security functions shall be able to generate an audit record of the following auditable events:

- a. Startup and shutdown of the audit functions
- b. All auditable events for the [minimum] level of audit logging
- c. [The events listed in Table B.3, “Auditable Events and Audit Data”]
FAU_GEN.1.2

The target security functions shall record within each audit record at least the following information:

a. Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event

b. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [the information specified in the "Additional Details" column in Table B.3, “Auditable Events and Audit Data”]. Additionally, the audit shall not include plaintext private or secret keys or other critical security parameters.
<table>
<thead>
<tr>
<th>Section/Function</th>
<th>Component</th>
<th>Event</th>
<th>Additional Details</th>
</tr>
</thead>
</table>
| Security Audit           | FAU_GEN.1 Audit data generation  | Any changes to the audit parameters, such  | Digital signature, keyed hash, or authentication code shall be included in the audit log.
<p>|                          | (iteration 2)                    | as audit frequency, type of event audited, |                                                                                   |
|                          |                                  | or an attempt to delete the audit log      |                                                                                   |
| Local Data Entry         | FPT_CIMC_TSP.1                   | Audit log signing event                    |                                                                                   |
|                          |                                  | All security-relevant data that is entered | The identity of the data entry individual if the entered data is linked to any other data, such as clicking an accept button. This shall be included with the accepted data. |
|                          |                                  | in the system                              |                                                                                   |
| Remote Data Entry        |                                  | All security-relevant messages that are    |                                                                                   |
|                          |                                  | received by the system                     |                                                                                   |
| Data Export and Output   |                                  | All successful and unsuccessful requests   |                                                                                   |
|                          |                                  | for confidential and security-relevant     |                                                                                   |
|                          |                                  | information                               |                                                                                   |
| Key Generation           | FCS_CKM.1 Cryptographic Key      | Whenever the target security functions    | The public component of any asymmetric key pair generated                          |
|                          | Generation                       | requests generation of a cryptographic key. |                                                                                   |
|                          |                                  | (Not mandatory for single session or one-  |                                                                                   |
|                          |                                  | time use symmetric keys.)                 |                                                                                   |
| Private Key Load         |                                  | The loading of Component private keys      |                                                                                   |
| Private Key Storage      |                                  | All access to certificate subject private  |                                                                                   |
|                          |                                  | keys retained within the target of evaluation for key recovery purposes |                                                                                   |
| Trusted Public Key       |                                  | All changes to the trusted public keys,    | The public key and all information associated with the key                         |
| Entry, Deletion and      |                                  | including additions and deletions         |                                                                                   |
| Storage                  |                                  |                                            |                                                                                   |
| Secret Key Storage       |                                  | The manual entry of secret keys used for   |                                                                                   |</p>
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private and Secret Key Export</td>
<td>FDP_ETC_CIMC.5 / FMT_MTD_CIMC.7</td>
<td>The export of private and secret keys (keys used for a single session or message are excluded)</td>
</tr>
<tr>
<td>Certificate Registration</td>
<td>FDP_CIMC_CER.1</td>
<td>All certificate requests</td>
</tr>
<tr>
<td>Certificate Status Change Approval</td>
<td></td>
<td>All requests to change the status of a certificate</td>
</tr>
<tr>
<td>CIMC Configuration</td>
<td></td>
<td>Any security-relevant changes to the configuration of the target security functions.</td>
</tr>
<tr>
<td>Certificate Profile Management</td>
<td>FMT_MOF_CIMC.3</td>
<td>All changes to the certificate Profile</td>
</tr>
<tr>
<td>Revocation Profile Management</td>
<td></td>
<td>All changes to the revocation profile</td>
</tr>
<tr>
<td>Certificate Revocation List Profile Management</td>
<td>FMT_MOF_CIMC.5</td>
<td>All changes to the certificate revocation list profile</td>
</tr>
<tr>
<td>Online Certificate Status Protocol (OCSP) Profile Management</td>
<td>FMT_MOF_CIMC.6</td>
<td>All changes to the OCSP profile</td>
</tr>
</tbody>
</table>

**FAU_GEN.2 User identity association (iteration 2)**

**FAU_GEN.2.1**

For audit events resulting from actions of identified users, the target security functions shall be able to associate each auditable event with the identity of the user that caused the event.

**FAU_SEL.1 Selective audit (iteration 2)**

**FAU_SEL.1.1**

The target security functions shall be able to select the set of audited events from the set of all auditable events based on the following attributes:

a. [event type]
b. [no additional attributes]

FAU_STG.1 Protected audit trail storage (iteration 2)

FAU_STG.1.1
The target security functions shall protect the stored audit records in the audit trail from unauthorized deletion.

FAU_STG.1.2
The target security functions shall be able to [detect] unauthorized modifications to the stored audit records in the audit trail.

FAU_STG.4 Prevention of audit data loss (iteration 2)

FAU_STG.4.1
The target security functions shall [prevent audited events, except those taken by the auditor,] and [no other action], if the audit trail is full.

B.5.2.2. Communication (FCO)
FCO_NRO_CIMC.3 Enforced proof of origin and verification of origin

FCO_NRO_CIMC.3.1
The target security functions shall enforce the generation of evidence of origin for certificate status information and all other security-relevant information at all times.

FCO_NRO_CIMC.3.2
The target security functions shall be able to relate the identity and [the identity of the certificate issuer] of the originator of the information, and the security-relevant portions of the information to which the evidence applies.

FCO_NRO_CIMC.3.3
The target security functions shall verify the evidence of origin of information for all security-relevant information.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by existing Common Criteria requirements. It supports the security objective O.Non-repudiation and O.Control unknown source communication traffic.

FCO_NRO_CIMC.4 Advanced verification of origin

FCO_NRO_CIMC.4.1
The target security functions shall, for initial certificate registration messages sent by the certificate subject, only accept messages protected using an authentication code, keyed hash, or digital signature algorithm.
FCO_NRO_CIMC.4.2

The target security functions shall, for all other security-relevant information, only accept the information if it was signed using a digital signature algorithm.

*Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by existing Common Criteria requirements. It supports the security objective O.Non-repudiation.*

B.5.2.3. Cryptographic Support (FCS)

FCS_CKM_CIMC.5 CIMC private and secret key zeroization

FCS_CKM_CIMC.5.1

The target security functions shall provide the capability to zeroize plaintext secret and private keys within the FIPS 140-2 validated cryptographic module.

*Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.*

FCS_SOF_CIMC.1 CIMC Strength of Functions

FCS_SOF_CIMC.1.1

The target security functions shall provide cryptographic mechanisms that fulfill the specific Strength of Function requirements.

*Rationale: This component is necessary to require specific Strength of Function metrics for cryptographic mechanisms of the target security functions.*

B.5.2.4. User Data Protection (FDP)

FDP_ACC.1 Subset access control (iteration 2)

FDP_ACC.1.1

The target security functions shall enforce the [CIMC target of evaluation access control policy] on [users, services, and access to services].

FDP_ACF.1 Security attribute based access control (iteration 2)

FDP_ACF.1.1

The target security functions shall enforce the [CIMC target of evaluation access control policy] to objects based on [the identity of the subject and the set of roles that the subject is authorized to assume].

FDP_ACF.1.2

The target security functions shall enforce the rules [in Table B.4, “Access Controls”] to determine if an operation among controlled subjects and controlled objects is allowed.
FDP_ACF.1.3
The target security functions shall explicitly authorize access of subjects to objects based on [no additional rules].

FDP_ACF.1.4
The target security functions shall explicitly deny access of subjects to objects based on [no additional rules].
Table B.4. Access Controls

<table>
<thead>
<tr>
<th>Section/Function</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Request Remote and Local Data Entry</td>
<td>The entry of certificate request data shall be restricted to officers and the subject of the requested certificate.</td>
</tr>
<tr>
<td>Certificate Revocation Request Remote and Local Data Entry</td>
<td>The entry of certificate revocation request data shall be restricted to officers and the subject of the certificate to be revoked.</td>
</tr>
<tr>
<td>Data Export and Output</td>
<td>The export or output of confidential and security-relevant data shall only be at the request of authorized users.</td>
</tr>
<tr>
<td>Key Generation</td>
<td>The capability to request the generation of Component keys (used to protect data in more than a single session or message) shall be restricted to administrators.</td>
</tr>
<tr>
<td>Private Key Load</td>
<td>The capability to request the loading of Component private keys into cryptographic modules shall be restricted to administrators.</td>
</tr>
<tr>
<td>Private Key Storage</td>
<td>The capability to request the decryption of certificate subject private keys shall be restricted to officers. The target security functions shall not provide a capability to decrypt certificate subject private keys that may be used to generate digital signatures. At least two officers or one officer and an administrator, auditor, or operator shall be required to request the decryption of a certificate subject private key.</td>
</tr>
<tr>
<td>Trusted Public Key Entry, Deletion, and Storage</td>
<td>The capability to change (add, revise, delete) the trusted public keys shall be restricted to administrators.</td>
</tr>
<tr>
<td>Secret Key Storage</td>
<td>The capability to request the loading of CIMC secret keys into cryptographic modules shall be restricted to administrators.</td>
</tr>
<tr>
<td>Private and Secret Key Destruction</td>
<td>The capability to zeroize CIMC plaintext private and secret keys shall be restricted to administrators, auditors, officers, and operators.</td>
</tr>
<tr>
<td>Private and Secret Key Export</td>
<td>The capability to export a component private key shall be restricted to administrators. The capability to export certificate subject private keys shall be restricted to officers. The export of a certificate subject private key shall require the authorization of at least two officers or one officer and an administrator, auditor, or operator.</td>
</tr>
</tbody>
</table>
Certificate Status Change Approval

Only officers and the subject of the certificate shall be capable of requesting that a certificate be placed on hold.

Only officers shall be capable of removing a certificate from on hold status.

Only officers shall be capable of approving the placing of a certificate on hold.

Only officers and the subject of the certificate shall be capable of requesting the revocation of a certificate.

Only officers shall be capable of approving the revocation of a certificate and all information about the revocation of a certificate.

FDP_ACF_CIMC.2 User private key confidentiality protection

FDP_ACF_CIMC.2.1
CIMS personnel private keys shall be stored in a FIPS 140-2 validated cryptographic module or stored in encrypted form. If CIMS personnel private keys are stored in encrypted form, the encryption shall be performed by the FIPS 140-2 validated cryptographic module.

FDP_ACF_CIMC.2.2
If certificate subject private keys are stored in the target of evaluation, they shall be encrypted using a Long Term Private Key Protection Key. The encryption shall be performed by the FIPS 140-2 validated cryptographic module.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FDP_ACF_CIMC.3 User secret key confidentiality protection

FDP_ACF_CIMC.3.1
User secret keys stored within the CIMC, but not within a FIPS 140-2 validated cryptographic module, shall be stored in encrypted form. The encryption shall be performed by the FIPS 140-2 validated cryptographic module.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FDP_CIMC_BKP.2 Extended CIMC backup and recovery

FDP_CIMC_BKP.2.1
The backup data shall be protected against modification through the use of digital signatures, keyed hashes, or authentication codes.
Critical security parameters and other confidential information shall be stored in encrypted form only.

*Rationale: This component is necessary to specify a unique requirement of certificate issuing and management components that is not addressed by Common Criteria. It supports the security objectives O.Object and data recovery free from malicious code and O.Preservation/trusted recovery of secure state.*

**FDP_CIMC_CER.1 Certificate Generation**

**FDP_CIMC_CER.1.1**
The target security functions shall only generate certificates whose format complies with [the X.509 standard for public key certificates].

**FDP_CIMC_CER.1.2**
The target security functions shall only generate certificates that are consistent with the currently defined certificate profile.

**FDP_CIMC_CER.1.3**
The target security functions shall verify that the prospective certificate subject possesses the private key that corresponds to the public key in the certificate request before issuing a certificate, unless the public/private key pair was generated by the target security functions, whenever the private key may be used to generate digital signatures.

**FDP_CIMC_CER.1.4**
If the target security functions generates X.509 public key certificates, it shall only generate certificates that comply with requirements for certificates as specified in ITU-T Recommendation X.509. At a minimum, the target security functions shall ensure that:

- a. The version field shall contain the integer 0, 1, or 2.
- b. If the certificate contains an issuerUniqueID or subjectUniqueID then the version field shall contain the integer 1 or 2.
- c. If the certificate contains extensions then the version field shall contain the integer 2.
- d. The serialNumber shall be unique with respect to the issuing Certification Authority.
- e. The validity field shall specify a notBefore value that does not precede the current time and a notAfter value that does not precede the value specified in notBefore.
- f. If the issuer field contains a null name, such as a sequence of zero relative distinguished names, then the certificate shall contain a critical issuerAltName extension.
- g. If the subject field contains a null name, such as a sequence of zero relative distinguished names, then the certificate shall contain a critical subjectAltName extension.
- h. The signature field and the algorithm in the subjectPublicKeyInfo field shall contain the OID for a FIPS-approved or recommended algorithm.

*Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.*

**FDP_CIMC_CRL.1 Certificate revocation list validation**
FDP_CIMC_CRL.1.1
A target security function that issues CRLs shall verify that all mandatory fields in any CRL issued contain values in accordance with ITU-T Recommendation X.509. At a minimum, the following items shall be validated:

1. If the version field is present, then it shall contain a 1.
2. If the CRL contains any critical extensions, then the version field shall be present and contain the integer 1.
3. If the issuer field contains a null name, such as a sequence of zero relative distinguished names, then the CRL shall contain a critical issuerAltName extension.
4. The signature and signatureAlgorithm fields shall contain the OID for a FIPS-approved digital signature algorithm.
5. The thisUpdate field shall indicate the issue date of the CRL.
6. The time specified in the nextUpdate field (if populated) shall not precede the time specified in the thisUpdate field.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FDP_CIMC_CSE.1 Certificate status export
FDP_CIMC_CSE.1.1
Certificate status information shall be exported from the target of evaluation in messages whose format complies with [the X.509 standard for CRLs (RFC2459) and, the OCSP standard as defined by RFC 2560].

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FDP_CIMC_OCSP.1 OCSP basic response validation
FDP_CIMC_OCSP.1.1
If a target security function is configured to allow OCSP responses of the basic response type, the target security functions shall verify that all mandatory fields in the OCSP basic response contain values in accordance with IETF RFC 2560. At a minimum, the following items shall be validated:

1. The version field shall contain a zero (0).
2. If the issuer field contains a null name, such as a sequence of zero relative distinguished names, then the response shall contain a critical issuerAltName extension.
3. The signatureAlgorithm field shall contain the OID for a FIPS-approved digital signature algorithm.
4. The thisUpdate field shall indicate the time at which the status being indicated is known to be correct.
5. The producedAt field shall indicate the time at which the OCSP responder signed the response.
6. The time specified in the nextUpdate field (if populated) shall not precede the time specified in the thisUpdate field.
**Rationale**: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

**FDP_ETC_CIMC.5 Extended user private and secret key export**

**FDP_ETC_CIMC.5.1**

Private and secret keys shall only be exported from the target of evaluation in encrypted form or using split knowledge procedures. Electronically distributed secret and private keys shall only be exported from the target of evaluation in encrypted form.

**Rationale**: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

**FDP_ITT.1 Basic internal transfer protection (iteration 3)**

**FDP_ITT.1.1**

The target security functions shall enforce the [CIMC target of evaluation access control policy] to prevent the [modification] of user data when it is transmitted between physically-separated parts of the target of evaluation.

**FDP_ITT.1 Basic internal transfer protection (iteration 4)**

**FDP_ITT.1.1**

The target security functions shall enforce the [CIMC target of evaluation access control policy] to prevent the [disclosure] of user data when it is transmitted between physically separated parts of the target of evaluation.

**FDP_SDI_CIMC.3 Stored public key integrity monitoring and action**

**FDP_SDI_CIMC.3.1**

Public keys stored within the CIMC, but not within a FIPS 140-2 validated cryptographic module, shall be protected against undetected modification through the use of digital signatures, keyed hashes, or authentication codes.

**FDP_SDI_CIMC.3.2**

The digital signature, keyed hash, or authentication code used to protect a public key shall be verified upon each access to the key. If verification fails, the target security functions shall [audit the failure].

**Rationale**: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

**FDP_UCT.1 Basic data exchange confidentiality (iteration 2)**

**FDP_UCT.1.1**

The target security functions shall enforce the [CIMC target of evaluation access control policy]
to be able to transmit objects in a manner protected from unauthorized disclosure.

B.5.2.5. Identification and Authentication (FIA)
FIA_UAU.1 Timing of authentication (iteration 2)

FIA_UAU.1.1
The target security functions shall allow [certificate enrollment requests and certificate retrieval] on behalf of the user to be performed before the user is authenticated.

FIA_UAU.1.2
The target security functions shall require each user to be successfully authenticated before allowing any other target security functions-mediated actions on behalf of that user.

FIA_UID.1 Timing of identification (iteration 2)

FIA_UID.1.1
The target security functions shall allow [certificate enrollment requests and certificate retrieval] on behalf of the user to be performed before the user is identified.

FIA_UID.1.2
The target security functions shall require each user to be successfully identified before allowing any other target security functions-mediated actions on behalf of that user.

FIA_USB.1 User-subject binding (iteration 2)

FIA_USB.1.1
The target security functions shall associate the [authentication token] user security attributes with subjects acting on the behalf of that user.

FIA_USB.1.2
The target security functions shall enforce the rules that [the authentication token that is assigned to the user after a successful authentication] on the initial association of user security attributes with subjects acting on the behalf of users.

FIA_USB.1.3
The target security functions shall associate the user security attributes that [the authentication token of a subject cannot change] with subjects acting on the behalf of that user.

B.5.2.6. Security Management (FMT)
FMT_MOF.1 Management of security functions behavior (iteration 2)

FMT_MOF.1.1
The target security functions shall restrict the ability to [modify the behavior of] the functions
[listed in Table B.5, “Authorized Roles for Management of Security Functions Behavior”] to [the authorized roles as specified in Table B.5, “Authorized Roles for Management of Security Functions Behavior”].
<table>
<thead>
<tr>
<th>Section/Function</th>
<th>Component Function</th>
<th>Authorized Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Audit</td>
<td>The capability to configure the audit parameters shall be restricted to administrators.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The capability to change the frequency of the audit log signing event shall be restricted to administrators.</td>
<td></td>
</tr>
<tr>
<td>Certificate Registration</td>
<td>The capability to approve fields or extensions to be included in a certificate shall be restricted to officers.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If an automated process is used to approve fields or extensions to be included in a certificate, the capability to configure that process shall be restricted to officers.</td>
<td></td>
</tr>
<tr>
<td>Data Export and Output</td>
<td>The export of CIMC private keys shall require the authorization of at least two administrators or one administrator and one officer, auditor, or operator.</td>
<td></td>
</tr>
<tr>
<td>Certificate Status Change Approval</td>
<td>Only officers shall configure the automated process used to approve the revocation of a certificate or information about the revocation of a certificate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Only officers shall configure the automated process used to approve the placing of a certificate on hold or information about the on hold status of a certificate.</td>
<td></td>
</tr>
<tr>
<td>CIMC Configuration</td>
<td>The capability to configure any target security functions functionality shall be restricted to administrators. (This requirement applies to all configuration parameters unless the ability to configure that aspect of the target security functions functionality has been explicitly assigned to a different role.)</td>
<td></td>
</tr>
</tbody>
</table>
Certificate Profile Management

The capability to modify the certificate profile shall be restricted to administrators.

Revocation Profile Management

The capability to modify the revocation profile shall be restricted to administrators.

Certificate Revocation List Profile Management

The capability to modify the certificate revocation list profile shall be restricted to administrators.

Online Certificate Status Protocol (OCSP) Profile Management

The capability to modify the OCSP profile shall be restricted to administrators.

FMT_MOF_CIMC.3 Extended certificate profile management

FMT_MOF_CIMC.3.1
The target security functions shall implement a certificate profile and shall ensure that issued certificates are consistent with that profile.

FMT_MOF_CIMC.3.2
The target security functions shall require the administrator to specify the set of acceptable values for the following fields and extensions:

1. The key owner's identifier
2. The algorithm identifier for the subject's public/private key pair
3. The identifier of the certificate issuer
4. The length of time for which the certificate is valid

FMT_MOF_CIMC.3.3
If the certificates generated are X.509 public key certificates, the target security functions shall require the administrator to specify the set of acceptable values for the following fields and extensions:

1. keyUsage
2. basicConstraints
3. certificatePolicies

FMT_MOF_CIMC.3.4
The administrator shall specify the acceptable set of certificate extensions.

Rationale: This component is necessary to specify a unique requirement of certificate issuing and management components that is not addressed by Common Criteria. It supports the security objective O.Configuration management.

FMT_MOF_CIMC.5 Extended certificate revocation list profile management

FMT_MOF_CIMC.5.1
If the target security function issues CRLs, the target security functions must implement a certificate revocation list profile and ensure that issued CRLs are consistent with the certificate revocation list profile.

**FMT_MOF_CIMC.5.2**
If the target security function issues CRLs, the target security functions shall require the administrator to specify the set of acceptable values for the following fields and extensions:

1. issuer
2. issuerAltName [4]
3. nextUpdate (that is, the lifetime of a CRL)

**FMT_MOF_CIMC.5.3**
If the target security function issues CRLs, the administrator shall specify the acceptable set of CRL and CRL entry extensions.

*Rationale:* This component is necessary to specify a unique requirement of certificate issuing and management components that is not addressed by Common Criteria. It supports the security objective O.Configuration management.

**FMT_MOF_CIMC.6** OCSP profile management

**FMT_MOF_CIMC.6.1**
If the target security function issues OCSP responses, the target security functions shall implement an OCSP profile and ensure that issued OCSP responses are consistent with the OCSP profile.

**FMT_MOF_CIMC.6.2**
If the target security functions issues OCSP responses, the target security functions shall require the administrator to specify the set of acceptable values for the responsetype field (unless the CIMC can only issue responses of the basic response type).

**FMT_MOF_CIMC.6.3**
If the target security function is configured to allow OCSP responses of the basic response type, the target security functions shall require the administrator to specify the set of acceptable values for the ResponderID field within the basic response type.

*Rationale:* This component is necessary to specify a unique requirement of certificate issuing and management components that is not addressed by Common Criteria. It supports the security objective O.Configuration management.

**FMT_MTD_CIMC.4** TSF private key confidentiality protection

**FMT_MTD_CIMC.4.1**
CIMC private keys shall be stored in a FIPS 140-2 validated cryptographic module or stored in encrypted form. If CIMC private keys are stored in encrypted form, the encryption shall be performed by the FIPS 140-2 validated cryptographic module.
Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FMT_MTD_CIMC.5 TSF secret key confidentiality protection

FMT_MTD_CIMC.5.1
Target security function secret keys stored within the target of evaluation, but not within a FIPS 140-2 validated cryptographic module, shall be stored in encrypted form. The encryption shall be performed by the FIPS 140-2 validated cryptographic module.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

FMT_MTD_CIMC.7 Extended TSF private and secret key export

FMT_MTD_CIMC.7.1
Private and secret keys shall only be exported from the target of evaluation in encrypted form or using split knowledge procedures. Electronically distributed secret and private keys shall only be exported from the target of evaluation in encrypted form.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by Common Criteria.

B.5.2.7. Protection of the Target Security Functions (FPT)

FPT_CIMC_TSP.1 Audit log signing event

FPT_CIMC_TSP.1.1
The target security functions shall periodically create an audit log signing event in which it computes a digital signature, keyed hash, or authentication code over the entries in the audit log.

FPT_CIMC_TSP.1.2
The digital signature, keyed hash, or authentication code shall be computed over, at least, every entry that has been added to the audit log since the previous audit log signing event and the digital signature, keyed hash, or authentication code from the previous audit log signed event.

FPT_CIMC_TSP.1.3
The specified frequency at which the audit log signing event occurs shall be configurable.

FPT_CIMC_TSP.1.4
The digital signature, keyed hash, or authentication code from the audit log signing event shall be included in the audit log.

Rationale: This component is necessary to specify a unique requirement for certificate issuing and management components that is not addressed by existing Common Criteria requirements. It supports the security objective O.Protect stored audit records, by providing additional protection for stored audit records at Security Levels 2 and 3.
FPT_ITC.1 Inter-TSF confidentiality during transmission (iteration 2)

FPT_ITC.1.1
The target security functions shall protect all target security functions data transmitted from the target security functions to another trusted IT product from unauthorized disclosure during transmission.

FPT_ITT.1 Basic internal TSF data transfer protection (iteration 3)

FPT_ITT.1.1
The target security functions shall protect target security functions data from [modification] when it is transmitted between separate parts of the target of evaluation.

FPT_ITT.1 Basic internal target security functions data transfer protection (iteration 4)

FPT_ITT.1.1
The target security functions shall protect target security functions data from [disclosure] when it is transmitted between separate parts of the target of evaluation.

FPT_STM.1 Reliable time stamps (iteration 2)

FPT_STM.1.1
The target security functions shall be able to provide reliable time stamps.

B.5.3. Target of Evaluation Security Assurance Requirements

The security assurance requirements for the target of evaluation are the Evaluation Assurance Level 4 (EAL 4) components, as specified in Part 3 of Common Criteria version 3.1, augmented with ALC_FLR.2 as indicated in bold the following table.
<table>
<thead>
<tr>
<th>Requirement Class</th>
<th>Requirement Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV: Development</td>
<td>ADV_ARC.1: Security architecture description</td>
</tr>
<tr>
<td></td>
<td>ADV_FSP.4: Complete functional specification</td>
</tr>
<tr>
<td></td>
<td>ADV_IMP.1: Implementation representation of the TSF</td>
</tr>
<tr>
<td></td>
<td>ADV_TDS.3: Basic modular design</td>
</tr>
<tr>
<td>AGD: Guidance documents</td>
<td>AGD_OPE.1: Operational user guidance</td>
</tr>
<tr>
<td></td>
<td>AGD_PRE.1: Preparative procedures</td>
</tr>
<tr>
<td>ALC: Life-cycle support</td>
<td>ALC_CMC.4: Production support, acceptance procedures and automation</td>
</tr>
<tr>
<td></td>
<td>ALC_CMS.4: Problem tracking CM coverage</td>
</tr>
<tr>
<td></td>
<td>ALC_DEL.1: Delivery procedures</td>
</tr>
<tr>
<td></td>
<td>ALC_DVS.1: Identification of security measures</td>
</tr>
<tr>
<td></td>
<td>ALC_FLR.2: Flaw reporting procedures</td>
</tr>
<tr>
<td></td>
<td>ALC_LCD.1: Developer defined life-cycle model</td>
</tr>
<tr>
<td></td>
<td>ALC_TAT.1: Well-defined development tools</td>
</tr>
<tr>
<td>ATE: Tests</td>
<td>ATE_COV.2: Analysis of coverage</td>
</tr>
<tr>
<td></td>
<td>ATE_DPT.2: Testing: security enforcing modules</td>
</tr>
<tr>
<td></td>
<td>ATE_FUN.1: Functional testing</td>
</tr>
<tr>
<td></td>
<td>ATE_IND.2: Independent testing - sample</td>
</tr>
<tr>
<td>AVA: Vulnerability assessment</td>
<td>AVA_VAN.3: Focused vulnerability analysis</td>
</tr>
</tbody>
</table>

**B.5.3.1. Development (ADV)**

**ADV_ARC.1 Security architecture description**

**ADV_ARC.1.1c**

The security architecture description shall be at a level of detail commensurate with the description of the SFR-enforcing abstractions described in the target of evaluation design document.

**ADV_ARC.1.1d**

The developer shall design and implement the target of evaluation so that the security features of the target security functions cannot be bypassed.
ADV_ARC.1.1e
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_ARC.1.2c
The security architecture description shall describe the security domains maintained by the target security functions consistently with the SFRs.

ADV_ARC.1.2d
The developer shall design and implement the target security functions so that it is able to protect itself from tampering by untrusted active entities.

ADV_ARC.1.3c
The security architecture description shall describe how the target security functions initialization process is secure.

ADV_ARC.1.3d
The developer shall provide a security architecture description of the target security functions.

ADV_ARC.1.4c
The security architecture description shall demonstrate that the target security functions protects itself from tampering.

ADV_ARC.1.5c
The security architecture description shall demonstrate that the target security functions prevents bypass of the SFR-enforcing functionality.

ADV_FSP.4 Complete functional specification

ADV_FSP.4.1c
The functional specification shall completely represent the target security functions.

ADV_FSP.4.1d
The developer shall provide a functional specification.

ADV_FSP.4.1e
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.4.2c
The functional specification shall describe the purpose and method of use for all target security function instructions.
ADV_FSP.4.2d
The developer shall provide a tracing from the functional specification to the SFRs.

ADV_FSP.4.2e
The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

ADV_FSP.4.3c
The functional specification shall identify and describe all parameters associated with each target security function instructions.

ADV_FSP.4.4c
The functional specification shall describe all actions associated with each target security function instructions.

ADV_FSP.4.5c
The functional specification shall describe all direct error messages that may result from security enforcing effects and exceptions associated with an invocation of each target security function instructions.

ADV_FSP.4.6c
The tracing shall demonstrate that the SFRs trace to target security function instructions in the functional specification.

ADV_IMP.1 Implementation representation of the TSF

ADV_IMP.1.1c
The implementation representation shall define the target security functions to a level of detail such that the target security functions can be generated without further design decisions.

ADV_IMP.1.1d
The developer shall make available the implementation representation for the entire target security functions.

ADV_IMP.1.1e
The evaluator shall confirm that, for the selected sample of the implementation representation, the information provided meets all requirements for content and presentation of evidence.

ADV_IMP.1.2c
The implementation representation shall be in the form used by the development personnel.

ADV_IMP.1.2d
The developer shall provide a mapping between the target of evaluation design description and the sample of the implementation representation.
ADV_IMP.1.3c
The mapping between the target of evaluation design description and the sample of the implementation representation shall demonstrate their correspondence.

ADV_TDS.3 Basic modular design

ADV_TDS.3.10c
The mapping shall demonstrate that all behavior described in the target of evaluation design is mapped to the target security function instructions that invoke it.

ADV_TDS.3.1c
The design shall describe the structure of the target of evaluation in terms of subsystems.

ADV_TDS.3.1d
The developer shall provide the design of the target of evaluation.

ADV_TDS.3.1e
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_TDS.3.2c
The design shall describe the target security functions in terms of modules.

ADV_TDS.3.2d
The developer shall provide a mapping from the target security function instructions of the functional specification to the lowest level of decomposition available in the target of evaluation design.

ADV_TDS.3.2e
The evaluator shall determine that the design is an accurate and complete instantiation of all security functional requirements.

ADV_TDS.3.3c
The design shall identify all subsystems of the target security functions.

ADV_TDS.3.4c
The design shall provide a description of each subsystem of the target security functions.

ADV_TDS.3.5c
The design shall provide a description of the interactions among all subsystems of the target security functions.

ADV_TDS.3.6c
The design shall provide a mapping from the subsystems of the target security functions to the
modules of the target security functions.

**ADV_TDS.3.7c**
The design shall describe each SFR-enforcing module in terms of its purpose.

**ADV_TDS.3.8c**
The design shall describe each SFR-enforcing module in terms of its SFR-related interfaces, return values from those interfaces, and called interfaces to other modules.

**ADV_TDS.3.9c**
The design shall describe each SFR-supporting or SFR-non-interfering module in terms of its purpose and interaction with other modules.

### B.5.3.2. Guidance Documents (AGD)

**AGD_OPE.1 Operational user guidance**

**AGD_OPE.1.1c**
The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.

**AGD_OPE.1.1d**
The developer shall provide operational user guidance.

**AGD_OPE.1.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**AGD_OPE.1.2c**
The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the target of evaluation in a secure manner.

**AGD_OPE.1.3c**
The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate.

**AGD_OPE.1.4c**
The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the target security functions.

**AGD_OPE.1.5c**
The operational user guidance shall identify all possible modes of operation of the target of
evaluation (including operation following failure or operational error), their consequences and implications for maintaining secure operation.

**AGD_OPE.1.6c**
The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfill the security objectives for the operational environment as described in the security target.

**AGD_OPE.1.7c**
The operational user guidance shall be clear and reasonable.

**AGD_PRE.1 Preparative procedures**

**AGD_PRE.1.1c**
The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered target of evaluation in accordance with the developer’s delivery procedures.

**AGD_PRE.1.1d**
The developer shall provide the target of evaluation including its preparative procedures.

**AGD_PRE.1.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**AGD_PRE.1.2c**
The preparative procedures shall describe all the steps necessary for secure installation of the target of evaluation and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the security target.

**AGD_PRE.1.2e**
The evaluator shall apply the preparative procedures to confirm that the target of evaluation can be prepared securely for operation.

**B.5.3.3. Life-Cycle Support (ALC)**

**ALC_CMC.4 Production support, acceptance procedures and automation**

**ALC_CMC.4.10c**
The evidence shall demonstrate that the CM system is being operated in accordance with the CM plan.

**ALC_CMC.4.1c**
The target of evaluation shall be labeled with its unique reference.
**ALC_CMC.4.1d**
The developer shall provide the target of evaluation and a reference for the target of evaluation.

**ALC_CMC.4.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_CMC.4.2c**
The CM documentation shall describe the method used to uniquely identify the configuration items.

**ALC_CMC.4.2d**
The developer shall provide the CM documentation.

**ALC_CMC.4.3c**
The CM system shall uniquely identify all configuration items.

**ALC_CMC.4.4c**
The CM system shall provide automated measures such that only authorized changes are made to the configuration items.

**ALC_CMC.4.5c**
The CM system shall support the production of the target of evaluation by automated means.

**ALC_CMC.4.6c**
The CM documentation shall include a CM plan.

**ALC_CMC.4.7c**
The CM plan shall describe how the CM system is used for the development of the target of evaluation.

**ALC_CMC.4.8c**
The CM plan shall describe the procedures used to accept modified or newly created configuration items as part of the target of evaluation.

**ALC_CMC.4.9c**
The evidence shall demonstrate that all configuration items are being maintained under the CM system.

**ALC_CMS.4 Problem tracking CM coverage**

**ALC_CMS.4.1c**
The configuration list shall include the following: the target of evaluation itself; the evaluation evidence required by the SARs; the parts that comprise the target of evaluation; the
implementation representation; and security flaw reports and resolution status.

**ALC_CMS.4.1d**
The developer shall provide a configuration list for the target of evaluation.

**ALC_CMS.4.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_CMS.4.2c**
The configuration list shall uniquely identify the configuration items.

**ALC_CMS.4.3c**
For each target security functions relevant configuration item, the configuration list shall indicate the developer of the item.

**ALC_DEL.1 Delivery procedures**

**ALC_DEL.1.1c**
The delivery documentation shall describe all procedures that are necessary to maintain security when distributing versions of the target of evaluation to the consumer.

**ALC_DEL.1.1d**
The developer shall document procedures for delivery of the target of evaluation or parts of it to the consumer.

**ALC_DEL.1.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_DEL.1.2d**
The developer shall use the delivery procedures.

**ALC_DVS.1 Identification of security measures**

**ALC_DVS.1.1c**
The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the target of evaluation design and implementation in its development environment.

**ALC_DVS.1.1d**
The developer shall produce development security documentation.

**ALC_DVS.1.1e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_DVS.1.2e**

The evaluator shall confirm that the security measures are being applied.

**ALC_FLR.2 Flaw reporting procedures**

**ALC_FLR.2.1c**

The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the target of evaluation.

**ALC_FLR.2.1d**

The developer shall document flaw remediation procedures addressed to target of evaluation developers.

**ALC_FLR.2.1e**

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_FLR.2.2c**

The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.

**ALC_FLR.2.2d**

The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws.

**ALC_FLR.2.3c**

The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.

**ALC_FLR.2.4c**

The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to target of evaluation users.

**ALC_FLR.2.5c**

The flaw remediation procedures shall describe a means by which the developer receives from target of evaluation users reports and inquiries of suspected security flaws in the target of evaluation.

**ALC_FLR.2.6c**

The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to target of evaluation users.
**ALC_FLR.2.7c**

The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.

**ALC_FLR.2.8c**

The flaw remediation guidance shall describe a means by which target of evaluation users report to the developer any suspected security flaws in the target of evaluation.

**ALC_LCD.1 Developer defined life-cycle model**

**ALC_LCD.1.1c**

The life-cycle definition documentation shall describe the model used to develop and maintain the target of evaluation.

**ALC_LCD.1.1d**

The developer shall establish a life-cycle model to be used in the development and maintenance of the target of evaluation.

**ALC_LCD.1.1e**

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_LCD.1.2c**

The life-cycle model shall provide for the necessary control over the development and maintenance of the target of evaluation.

**ALC_LCD.1.2d**

The developer shall provide life-cycle definition documentation.

**ALC_TAT.1 Well-defined development tools**

**ALC_TAT.1.1c**

Each development tool used for implementation shall be well-defined.

**ALC_TAT.1.1d**

The developer shall identify each development tool being used for the target of evaluation.

**ALC_TAT.1.1e**

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ALC_TAT.1.2c**

The documentation of each development tool shall unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation.
ALC_TAT.1.2d
The developer shall document the selected implementation-dependent options of each
development tool.

ALC_TAT.1.3c
The documentation of each development tool shall unambiguously define the meaning of all
implementation-dependent options.

B.5.3.4. Tests (ATE)
ATE_COV.2 Analysis of coverage

ATE_COV.2.1c
The analysis of the test coverage shall demonstrate the correspondence between the tests in
the test documentation and the target security function instructions in the functional
specification.

ATE_COV.2.1d
The developer shall provide an analysis of the test coverage.

ATE_COV.2.1e
The evaluator shall confirm that the information provided meets all requirements for content and
presentation of evidence.

ATE_COV.2.2c
The analysis of the test coverage shall demonstrate that all target security function instructions
in the functional specification have been tested.

ATE_DPT.2 Testing: security enforcing modules

ATE_DPT.2.1c
The analysis of the depth of testing shall demonstrate the correspondence between the tests in
the test documentation and the target security functions subsystems and modules in the
target of evaluation design.

ATE_DPT.2.1d
The developer shall provide the analysis of the depth of testing.

ATE_DPT.2.1e
The evaluator shall confirm that the information provided meets all requirements for content and
presentation of evidence.

ATE_DPT.2.2c
The analysis of the depth of testing shall demonstrate that all target security functions
subsystems in the target of evaluation design have been tested.
ATE_DPT.2.3c
The analysis of the depth of testing shall demonstrate that the SFR-enforcing modules in the
target of evaluation design have been tested.

ATE_FUN.1 Functional testing

ATE_FUN.1.1c
The test documentation shall consist of test plans, expected test results and actual test results.

ATE_FUN.1.1d
The developer shall test the target security functions and document the results.

ATE_FUN.1.1e
The evaluator shall confirm that the information provided meets all requirements for content and
presentation of evidence.

ATE_FUN.1.2c
The test plans shall identify the tests to be performed and describe the scenarios for
performing each test.

ATE_FUN.1.2d
The developer shall provide test documentation.

ATE_FUN.1.3c
The expected test results shall show the anticipated outputs from a successful execution of the
tests.

ATE_FUN.1.4c
The actual test results shall be consistent with the expected test results.

ATE_IND.2 Independent testing – sample

ATE_IND.2.1c
The target of evaluation shall be suitable for testing.

ATE_IND.2.1d
The developer shall provide the target of evaluation for testing.

ATE_IND.2.1e
The evaluator shall confirm that the information provided meets all requirements for content and
presentation of evidence.
ATE_IND.2.2c
The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the target security functions.

ATE_IND.2.2e
The evaluator shall execute a sample of tests in the test documentation to verify the developer test results.

ATE_IND.2.3e
The evaluator shall test a subset of the target security functions interfaces to confirm that the target security functions operates as specified.

B.5.3.5. Vulnerability Assessment (AVA)
AVA_VAN.3 Focused vulnerability analysis

AVA_VAN.3.1c
The target of evaluation shall be suitable for testing.

AVA_VAN.3.1d
The developer shall provide the target of evaluation for testing.

AVA_VAN.3.1e
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.3.2e
The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the target of evaluation.

AVA_VAN.3.3e
The evaluator shall perform an independent vulnerability analysis of the target of evaluation using the guidance documentation, functional specification, target of evaluation design, security architecture description and implementation representation to identify potential vulnerabilities in the target of evaluation.

AVA_VAN.3.4e
The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the target of evaluation is resistant to attacks performed by an attacker possessing Enhanced-Basic attack potential.

[2] Certificate enrollment allows users to request various types of certificates. However, in order for a request to be fulfill the user must either be authenticated or an officer must approve the request.

[3] Certificate retrieval allows users to search, list and view certificates as well as download certificates and CRLs.
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, Data Recovery Manager 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.

[4] If a CIMC does not issue CRLs with this extension, then it is not required within the certificate revocation list profile.
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.
bind DN

A user ID, in the form of a distinguished name (DN), used with a password to authenticate to Red Hat Directory Server.

CA certificate

A certificate that identifies a certificate authority. See also certificate authority (CA), subordinate CA, root CA.

CA hierarchy

A hierarchy of CAs in which a root CA delegates the authority to issue certificates to subordinate CAs. Subordinate CAs can also expand the hierarchy by delegating issuing status to other CAs. See also certificate authority (CA), subordinate CA, root CA.

CA server key

The SSL server key of the server providing a CA service.

CA signing key

The private key that corresponds to the public key in the CA certificate. A CA uses its signing key to sign certificates and CRLs.

certificate

Digital data, formatted according to the X.509 standard, that specifies the name of an individual, company, or other entity (the subject name of the certificate) and certifies that a public key, which is also included in the certificate, belongs to that entity. A certificate is issued and digitally signed by a certificate authority (CA). A certificate’s validity can be verified by checking the CA’s digital signature through public-key cryptography techniques. To be trusted within a public-key infrastructure (PKI), a certificate must be issued and signed by a CA that is trusted by other entities enrolled in the PKI.

certificate authority (CA)

A trusted entity that issues a certificate after verifying the identity of the person or entity the certificate is intended to identify. A CA also renews and revokes certificates and generates CRLs. The entity named in the issuer field of a certificate is always a CA. Certificate authorities can be independent third parties or a person or organization using certificate-issuing server software, such as Red Hat Certificate System.

certificate chain

A hierarchical series of certificates signed by successive certificate authorities. A CA certificate identifies a certificate authority (CA) and is used to sign certificates issued by that authority. A CA certificate can in turn be signed by the CA certificate of a parent CA, and so on up to a root CA. Certificate System allows any end entity to retrieve all the certificates in a certificate chain.
**certificate extensions**

An X.509 v3 certificate contains an extensions field that permits any number of additional fields to be added to the certificate. Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates. A number of standard extensions have been defined by the PKIX working group.

**certificate fingerprint**

A [one-way hash](#) associated with a certificate. The number is not part of the certificate itself, but is produced by applying a hash function to the contents of the certificate. If the contents of the certificate changes, even by a single character, the same function produces a different number. Certificate fingerprints can therefore be used to verify that certificates have not been tampered with.

**Certificate Management Message Formats (CMMF)**

Message formats used to convey certificate requests and revocation requests from end entities to a Certificate Manager and to send a variety of information to end entities. A proposed standard from the Internet Engineering Task Force (IETF) PKIX working group. CMMF is subsumed by another proposed standard, [Certificate Management Messages over Cryptographic Message Syntax (CMC)](#). For detailed information, see [http://www.ietf.org/internet-drafts/draft-ietf-pkix-cmmf-02.txt](http://www.ietf.org/internet-drafts/draft-ietf-pkix-cmmf-02.txt).

**Certificate Management Messages over Cryptographic Message Syntax (CMC)**


**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 ftp://ftp.isi.edu/in-notes/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

Certificate System console

Certificate System subsystem
One of the five Certificate System managers: Certificate Manager, Online Certificate Status Manager, Data Recovery Manager, Token Key Service, or Token Processing System.

certificate-based authentication
Authentication based on certificates and public-key cryptography. See also password-based authentication.

chain of trust
See certificate chain.

chained CA
See linked CA.

cipher
See cryptographic algorithm.

client authentication
The process of identifying a client to a server, such as with a name and password or with a certificate and some digitally signed data. See certificate-based authentication, password-based authentication, server authentication.

client SSL certificate
A certificate used to identify a client to a server using the SSL protocol. See Secure Sockets Layer (SSL).
CMC

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

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

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).

Data Recovery Manager

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 Data Recovery Manager before issuing new certificates. The Data Recovery Manager 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.

Data Recovery Manager agent

A user who belongs to a group authorized to manage agent services for a Data Recovery Manager, including managing the request queue and authorizing recovery operation using HTML-based administration pages.

Data Recovery Manager recovery agent

One of the $m$ of $n$ people who own portions of the storage key for the Data Recovery Manager.

Data Recovery Manager storage key

Special key used by the Data Recovery Manager to encrypt the end entity’s encryption key after it has been decrypted with the Data Recovery Manager’s private transport key. The storage key never leaves the Data Recovery Manager.

Data Recovery Manager transport certificate

Certifies the public key used by an end entity to encrypt the entity’s encryption key for transport to the Data Recovery Manager. The Data Recovery Manager uses the private key corresponding to the certified public key to decrypt the end entity’s key before encrypting it with the storage key.

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.

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 http://ietfreport.isoc.org/idref/draft-ietf-tls-ecc/.
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.

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.

extension field
See certificate extensions.

enrollment
The process of requesting and receiving an X.509 certificate for use in a public-key infrastructure (PKI). Also known as registration.

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.

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)

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 Netscape Security Services (NSS).
K

KEA
See Key Exchange Algorithm (KEA).

key
A large number used by a cryptographic algorithm to encrypt or decrypt data. A person’s public key, for example, allows other people to encrypt messages intended for that person. The messages must then be decrypted by using the corresponding private key.

key exchange
A procedure followed by a client and server to determine the symmetric keys they will both use during an SSL session.

Key Exchange Algorithm (KEA)
An algorithm used for key exchange by the US Government.

L

Lightweight Directory Access Protocol (LDAP)
A directory service protocol designed to run over TCP/IP and across multiple platforms. LDAP is a simplified version of Directory Access Protocol (DAP), used to access X.500 directories. LDAP is under IETF change control and has evolved to meet Internet requirements.

linked CA
An internally deployed certificate authority (CA) whose certificate is signed by a public, third-party CA. The internal CA acts as the root CA for certificates it issues, and the third-party CA acts as the root CA for certificates issued by other CAs that are linked to the same third-party root CA. Also known as "chained CA" and by other terms used by different public CAs.

M

manual authentication
A way of configuring a Certificate System subsystem that requires human approval of each certificate request. With this form of authentication, a servlet forwards a certificate request to a request queue after successful authentication module processing. An agent with appropriate privileges must then approve each request individually before profile processing and certificate issuance can proceed.

MD5
A message digest algorithm that was developed by Ronald Rivest. See also one-way hash.

message digest
See one-way hash.
misrepresentation
The presentation of an entity as a person or organization that it is not. For example, a website might pretend to be a furniture store when it is really a site that takes credit-card payments but never sends any goods. Misrepresentation is one form of impersonation. See also spoofing.

N
Netscape Security Services (NSS)
A set of libraries designed to support cross-platform development of security-enabled communications applications. Applications built using the NSS libraries support the Secure Sockets Layer (SSL) protocol for authentication, tamper detection, and encryption, and the PKCS #11 protocol for cryptographic token interfaces. NSS is also available separately as a software development kit.

non-TMS
Non-token management system. Refers to a configuration of subsystems (the CA and, optionally, DRM, OCSP, and RA) 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's associated private key is used to sign objects; related to object signing.

OCSP

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.

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 Data Recovery Manager transport key that contains information about an archived end-entity key, including key serial number, name of the Data Recovery Manager, subject name of the corresponding certificate, and date of archival. The signed proof-of-archival data are the response returned by the Data Recovery Manager to the Certificate Manager after a successful key archival operation. See also Data Recovery Manager 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, Data Recovery Manager, Token Key Service, and Token Processing System.

registration

See enrollment.

root CA

The certificate authority (CA) with a self-signed certificate at the top of a certificate chain. See also CA certificate, subordinate CA.
RSA algorithm
Short for Rivest-Shamir-Adleman, a public-key algorithm for both encryption and authentication.
It was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman and introduced in 1978.

RSA key exchange
A key-exchange algorithm for SSL based on the RSA algorithm.

sandbox
A Java™ term for the carefully defined limits within which Java™ code must operate.

secure channel
A security association between the TPS and the smart card which allows encrypted communication based on a shared master key generated by the TKS and the smart card APDUs.

Secure Sockets Layer (SSL)
A protocol that allows mutual authentication between a client and server and the establishment of an authenticated and encrypted connection. SSL runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols.

security domain
A centralized repository or inventory of PKI subsystems. Its primary purpose is to facilitate the installation and configuration of new PKI services by automatically establishing trusted relationships between subsystems.

self tests
A feature that tests a Certificate System instance both when the instance starts up and on-demand.

server authentication
The process of identifying a server to a client. See also client authentication.

server SSL certificate
A certificate used to identify a server to a client using the Secure Sockets Layer (SSL) protocol.

servlet
Java™ code that handles a particular kind of interaction with end entities on behalf of a Certificate System subsystem. For example, certificate enrollment, revocation, and key recovery requests are each handled by separate servlets.
SHA-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’s public key corresponds to a private key used to create digital signatures. For example, a Certificate Manager must have a signing certificate that’s public key corresponds to the private key it uses to sign the certificates it issues.

signing key
A private key used for signing only. A signing key and its equivalent public key, plus an encryption key and its equivalent public key, constitute a dual key pair.

single sign-on
1. In Certificate System, a password that simplifies the way to sign on to Red Hat Certificate System by storing the passwords for the internal database and tokens. Each time a user logs on, he is required to enter this single password.

2. The ability for a user to log in once to a single computer and be authenticated automatically by a variety of servers within a network. Partial single sign-on solutions can take many forms, including mechanisms for automatically tracking passwords used with different servers. Certificates support single sign-on within a public-key infrastructure (PKI). A user can log in once to a local client's private-key database and, as long as the client software is running, rely on certificate-based authentication to access each server within an organization that the user is allowed to access.

slot
The portion of a PKCS #11 module, implemented in either hardware or software, that contains a token.

smart card
A small device that contains a microprocessor and stores cryptographic information, such as keys and certificates, and performs cryptographic operations. Smart cards implement some or all of the PKCS #11 interface.

spoofing
Pretending to be someone else. For example, a person can pretend to have the email address jdoe@example.com, or a computer can identify itself as a site called www.redhat.com.
when it is not. Spoofing is one form of impersonation. See also misrepresentation.

SSL
See Secure Sockets Layer (SSL).

subject
The entity identified by a certificate. In particular, the subject field of a certificate contains a subject name that uniquely describes the certified entity.

subject name
A distinguished name (DN) that uniquely describes the subject of a certificate.

subordinate CA
A certificate authority that's certificate is signed by another subordinate CA or by the root CA. See CA certificate, root CA.

symmetric encryption
An encryption method that uses the same cryptographic key to encrypt and decrypt a given message.

T

tamper detection
A mechanism ensuring that data received in electronic form entirely corresponds with the original version of the same data.

token
A hardware or software device that is associated with a slot in a PKCS #11 module. It provides cryptographic services and optionally stores certificates and keys.

token key service (TKS)
A subsystem in the token management system which derives specific, separate keys for every smart card based on the smart card APDUs and other shared information, like the token CUID.

token management system (TMS)
The interrelated subsystems — CA, TKS, TPS, and, optionally, the DRM — which are used to manage certificates on smart cards (tokens).

token processing system (TPS)
A subsystem which interacts directly the Enterprise Security Client and smart cards to manage the keys and certificates on those smart cards.
tree hierarchy
The hierarchical structure of an LDAP directory.

trust
Confident reliance on a person or other entity. In a public-key infrastructure (PKI), trust refers to the relationship between the user of a certificate and the certificate authority (CA) that issued the certificate. If a CA is trusted, then valid certificates issued by that CA can be trusted.

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
accelerators, Tokens for Storing Certificate System Subsystem Keys and Certificates, Hardware Cryptographic Accelerators
administrators
- tools provided
  - Certificate System console, The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems
agent certificate, User Certificates
agents
- authorizing key recovery, Recovering Keys
- port used for operations, Planning Ports
algorithm
- cryptographic, Encryption and Decryption
authentication
- certificate-based, Certificate-Based Authentication
- client and server, Authentication Confirms an Identity
- password-based, Password-Based Authentication
- See also client authentication, Certificate-Based Authentication
- See also server authentication, Certificate-Based Authentication

C
CA
- certificate, Types of Certificates
- defined, A Certificate Identifies Someone or Something
- hierarchies and root, CA Hierarchies
- trusted, How CA Certificates Establish Trust

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 scalability, CA Cloning

CA signing certificate, CA Signing Certificates, 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
- CA signing certificate, CA Signing Certificates
- chaining to third-party CAs, Linked CA
- cloning, CA Cloning
- DRM and, Planning for Lost Keys: Key Archival and Recovery

Certificate System
- Elliptic Curve Cryptography (ECC), Using ECC
- starting and stopping, Starting, Stopping, and Restarting an Instance

Certificate System console
- Configuration tab, The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems
- Status tab, The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems

Certificate-based authentication
- defined, Authentication Confirms an Identity

Certificates
- authentication using, Certificate-Based Authentication
- CA certificate, Types of Certificates
chains, Certificate Chains
- contents of, Contents of a Certificate
- issuing of, Issuing Certificates
- renewing, Renewing and Revoking Certificates
- revoking, Renewing and Revoking Certificates
- S/MIME, Types of Certificates
- self-signed, CA Hierarchies
- verifying a certificate chain, Verifying a Certificate Chain

ciphers
- defined, Encryption and Decryption

certification
- client authentication defined, Types of Certificates

certification, CA Cloning

Common Criteria
- configuring SSL client authentication, Installing and Configuring a CA, Installing and Configuring a DRM, Installing and Configuring an OCSP Responder
- configuring SSL client authentication, Installing and Configuring a TKS, Installing and Configuring a TPS

Configuration tab, The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems

CRL signing certificate, Other Signing Certificates

CRLs
- Certificate Manager support for, CRLs
- publishing to online validation authority, OCSP Services

deployment planning
- CA decisions
  - 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

- token management, Working with Smart Cards (TMS)

digital signatures
- defined, Digital Signatures

distinguished name (DN)
- for CA, Planning the CA Distinguished Name
DRM
- Certificate Manager and, Planning for Lost Keys: Key Archival and Recovery

E
Elliptic Curve Cryptography (ECC), Using ECC
email, signed and encrypted, Signed and Encrypted Email
encryption
- defined, Encryption and Decryption
- public-key, Public-Key Encryption
- symmetric-key, Symmetric-Key Encryption

Enterprise Security Client, Enterprise Security Client
extensions
- structure of, Structure of Certificate Extensions

external tokens
- defined, Tokens for Storing Certificate System Subsystem Keys and Certificates, External Tokens
- installing, Installing External Tokens and Unsupported HSM

F
Firefox profiles, Installing and Configuring a CA, Installing and Configuring a DRM,
Installing and Configuring an OCSP Responder, Installing and Configuring an RA,
Installing and Configuring a TKS, Installing and Configuring a TPS

H
hardware accelerators, Tokens for Storing Certificate System Subsystem Keys and Certificates, Hardware Cryptographic Accelerators
hardware tokens, Tokens for Storing Certificate System Subsystem Keys and Certificates, Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates
- See external tokens, Tokens for Storing Certificate System Subsystem Keys and Certificates, External Tokens

how to search for keys, Archiving Keys

I
installation, Installing and Configuring Certificate System
- planning, A Checklist for Planning the PKI
- prerequisites
  - Firefox profiles, Installing and Configuring a CA, Installing and Configuring a DRM,
    Installing and Configuring an OCSP Responder, Installing and Configuring an RA,
    Installing and Configuring a TKS, Installing and Configuring a TPS
installing external hardware tokens, Installing External Tokens and Unsupported HSM
internal tokens, Tokens for Storing Certificate System Subsystem Keys and Certificates, Internal Tokens

K

key archival, Archiving Keys
- how it works, Archiving Keys
- how keys are stored, Archiving Keys
- PKI setup required, Archiving and Recovering 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
keys
- defined, Encryption and Decryption
- management and recovery, Key Management

L

linked CA, Linked CA

O

OCSP responder, OCSP Services
OCSP server, OCSP Services
OCSP signing certificate, Other Signing Certificates

P

password
- using for authentication, Authentication Confirms an Identity

password-based authentication, defined, Password-Based Authentication
PKCS #11 support, Tokens for Storing Certificate System Subsystem Keys and Certificates, Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates, External Tokens
planning installation, A Checklist for Planning the PKI
ports
- for agent operations, Planning Ports
- how to choose numbers, Planning Ports

private key, defined, Public-Key Encryption
public key
- defined, Public-Key Encryption
- management, Key Management

publishing
- of CRLs
  - to online validation authority, OCSP Services

R
recovering users' private keys, Recovering Keys
root CA, Subordination to a Certificate System CA
root versus subordinate CA, Defining the Certificate Authority Hierarchy
RSA, Choosing the Signing Key Type and Length

S
S/MIME certificate, Types of Certificates
self-signed certificate, CA Hierarchies
signing certificate
  - CA, Setting the CA Signing Certificate Validity Period

signing key, for CA, Choosing the Signing Key Type and Length
SSL
  - client certificates, Types of Certificates

SSL client authentication, Installing and Configuring a CA, Installing and Configuring a DRM, Installing and Configuring an OCSP Responder, Installing and Configuring a TKS, Installing and Configuring a TPS
SSL client certificate, SSL Server and Client Certificates
SSL server certificate, SSL Server and Client Certificates
Status tab, The Java Administrative Console for CA, OCSP, DRM, and TKS Subsystems
subordinate CA, Subordination to a Certificate System CA

T
Token Key Service, Working with Smart Cards (TMS)
  - Token Processing System and, Working with Smart Cards (TMS)

Token Key Service (TKS), The TKS and Secure Channels
Token Management System
  - Enterprise Security Client, Enterprise Security Client
  - TKS, The TKS and Secure Channels
Token Processing System, **Working with Smart Cards (TMS)**
- scalability, **Using Smart Cards**
- Token Key Service and, **Working with Smart Cards (TMS)**

tokens
- defined, **Tokens for Storing Certificate System Subsystem Keys and Certificates**, **Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates**, **Types of Hardware Tokens**
- external, **Tokens for Storing Certificate System Subsystem Keys and Certificates**, **Setting up HSMs for Storing Certificate System Subsystem Keys and Certificates**, **External Tokens**
- internal, **Tokens for Storing Certificate System Subsystem Keys and Certificates**, **Internal Tokens**
- viewing which tokens are installed, **Viewing Tokens**

topology decisions, for deployment, **Working with Smart Cards (TMS)**
transport certificate
- when used, **Archiving Keys**

trusted CA, defined, **How CA Certificates Establish Trust**

user certificate, **User Certificates**