Red Hat Certificate System 10

规划、安装和部署指南

为 Red Hat Certificate System 10.4 更新

Last Updated: 2023-09-10

Red Hat Certificate System 10 规划、安装和部署指南

为 Red Hat Certificate System 10.4 更新

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

Marc Muehlfeld
Red Hat Customer Content Services

Petr Bokoč
Red Hat Customer Content Services

Filip Hanzelka
Red Hat Customer Content Services

Tomáš Čapek
Red Hat Customer Content Services

Aneta Petrová
Red Hat Customer Content Services

Ella Deon Ballard
Red Hat Customer Content Services

法律通告

摘要

本指南涵盖了规划 PKI 基础架构的主要 PKI （公钥基础架构）概念和决策方面。

目录

部分 I. 规划如何部署 RED HAT CERTIFICATE SYSTEM

本节概述证书系统，包括证书系统及其子系统的常规 PKI 原则和特定功能。规划部署对于设计符合您组织需求的 PKI 基础架构至关重要。

第 1 章 PUBLIC-KEY CRYPTOGRAPHY 简介

公钥加密和相关标准取决于许多产品的安全功能，如签名和加密的电子邮件、单点登录和传输层安全/安全套接字层(SSL/TLS)通信。本章论述了公钥加密的基本概念。

Internet 流量（通过中间计算机传递信息）可以被第三方拦截：

窃听

信息保持不变，但其隐私泄露。例如，某人可以收集信用卡号码、记录敏感对话或拦截分类信息。

篡改

传输中的信息会被更改或替换，然后发送到接收者。例如，某人可以更改好或改变个人恢复的订单。

模拟（Impersonation）

信息传递给作为预期接收者的人员。模拟可以采用两种形式：

- *欺骗*个人可以假定成为其他人。例如：一个人可以假定使用电子邮件地址 **jdoe@example.net**，或者计算机可以假地将自己识别为名为 **www.example.net** 的站点。
- *Misrepresentation*。个人或组织可以代表自己。例如：当名为 **www.example.net** 的网站实际上收到信用卡付款但从未发送任何良好时，可以将其作为在线电影存储。

公钥加密通过以下方式 保护基于互联网的攻击：

加密和解密

加密和解密允许两个通信方互相发送的信息。在发送方之前，发送方加密或 scrambles 信息。接收器在收到信息后解密或取消评分。在传输过程中，加密的信息不适用于入侵者。

篡改检测

篡改检测允许接收方验证信息在传输中没有被修改。检测到任何修改或替换数据的尝试。

身份验证

身份验证允许接收者通过确认发件人的身份来确定其来源。

nonRepudiation

nonRepudiation 可防止发送信息的发送者在以后不会发送信息。

1.1. 加密和解密

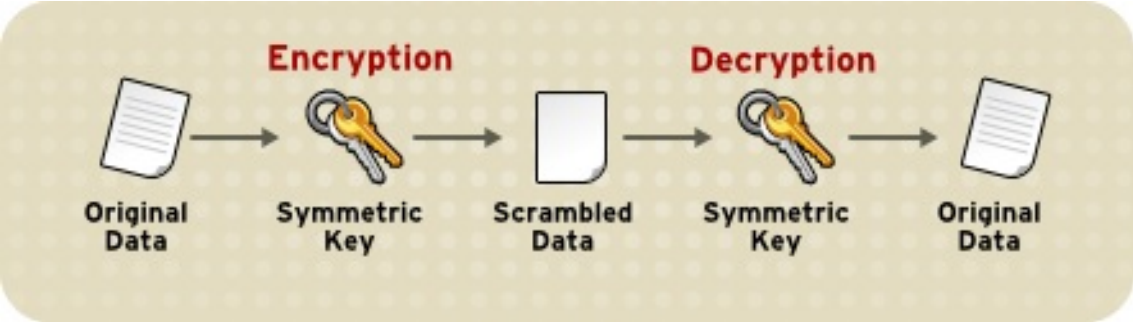
加密 是转换信息的过程，使其对除预期接收者以外的任何人都不适合。*解密* 是解码加密信息的过程。加密算法也称为 *密码*，是用于加密或解密的数学函数。通常，使用了两个相关功能，一个用于加密，另一个用于解密。

使用大多数现代加密时，保持加密信息 secret 的功能不基于加密算法，这并非广泛已知的，而是使用名为 *的密钥* 的数字来生成加密结果或解密之前加密的信息。使用正确密钥进行解密非常简单。如果无法进行，则在没有正确密钥的情况下进行解密非常困难。

1.1.1. symmetric-Key 加密

使用对称密钥加密，可以从解密密钥计算加密密钥，反之亦然。使用大多数对称算法时，相同的密钥用于加密和解密，如 图 1.1 “symmetric-Key 加密” 所示。

图 1.1. symmetric-Key 加密



对称密钥加密的实现可能非常高效，因此，由于加密和解密，用户不会遇到任何显著的延迟。

仅当对称密钥被涉及的双方持有机密时，对称密钥加密才有效。如果其他人发现密钥，它会同时影响机密性和身份验证。具有未授权的对称密钥的人员无法解密与该密钥发送的消息，但可以加密新消息，并将其发送，就像使用该密钥来自其中一个合法方一样。

对称密钥加密在 SSL/TLS 通信中扮演重要角色，它广泛用于通过 TCP/IP 网络进行身份验证、篡改检测和加密。SSL/TLS 还使用公钥加密的技术，在下一节中进行了描述。

1.1.2. 公钥加密

公钥加密（也称为非对称加密）涉及一对密钥、公钥和私钥，与实体相关联。每个公钥都已发布，并且对应的私钥保密。（有关发布公钥的方式的更多信息，请参阅 第 1.3 节 “证书和验证”。） 使用公钥加密的数据只能使用对应的私钥解密。图 1.2 “公钥加密” 显示公钥加密的工作方式的简化视图。

图 1.2. 公钥加密

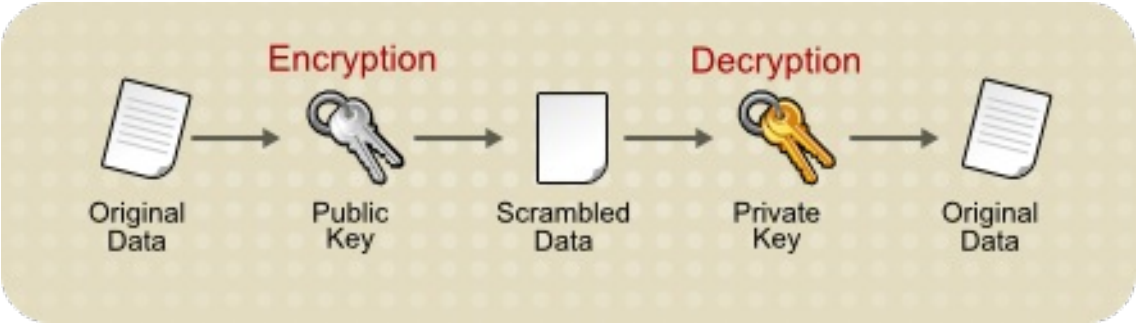


图 1.2 “公钥加密” 中显示的方案允许公钥自由分发，同时只有授权人员能够读取使用此密钥加密的数据。通常，要发送加密数据，数据会与该人的公钥进行加密，并且接收加密数据的个人使用对应的私钥解密数据。

与对称密钥加密相比，公钥加密需要更多处理，且可能不适用于加密和解密大量数据。但是，可以使用公钥加密来发送对称密钥，然后可用于加密其他数据。这是 SSL/TLS 协议使用的方法。

图 1.2 “公钥加密” 中显示的方案的反向也可以正常工作：使用私钥加密的数据只能使用对应的公钥解密。但不建议对敏感数据进行加密，因为这意味着具有公钥的任何人（由发布的定义发布）都可以解密数据。然而，私钥加密非常有用，因为它意味着私钥可用于使用数字签名为数据签名，对电子商业和其他加密商业应用程序的一项重要要求。然后 Mozilla Firefox 等客户端软件可以使用公钥确认消息已用适当的私钥签名，并且自自签名以来还没有篡改。第 1.2 节 “数字签名” 演示了此确认过程的工作方式。

1.1.3. Key Length 和 Encryption Strength

破坏 加密算法以纯文本形式查找访问加密数据的密钥。对于对称算法，破坏算法通常意味着尝试确定用于加密文本的密钥。对于公钥算法，破坏算法通常意味着在两个接收方之间获取共享机密信息。

破坏对称算法的一种方法是，只需在找到正确密钥之前，只需尝试完整算法中的每个键即可。对于公钥算法，由于半数密钥对是公开已知的，其他半（私钥）可以使用已发布、但数学的计算而衍生。手动查找用于中断算法的密钥被称为 brute 强制攻击。

破坏算法可能会带来拦截的风险，甚至可以模仿和欺诈地验证专用信息。

算法 *的主要强度* 是通过找到最快的方法破坏算法并将其与暴力攻击相比较来确定的。

对于对称密钥，通常会在用于执行加密的密钥的大小或 *长度* 上描述加密密钥：较长的密钥通常提供更强大的加密。密钥长度以位为单位。

如果最已知的攻击中断密钥的速度不快于 brute 强制测试每个密钥的可能性，则加密密钥被视为完全强度。

不同类型的算法 - 特别是公钥算法 - 可能需要不同的密钥长度才能获得与对称密钥密码相同的加密强度。由于给定长度的性质，RSA 密码只能将所有可能值的子集用于给定长度的键。其他密码（如用于对称密钥加密的密码）可以将所有可能的值用于给定长度的键。更多可能匹配的选项意味着更高的安全性。

因为破坏 RSA 密钥相对简单，所以 RSA 公钥加密密码必须具有非常长的密钥 - 至少 2048 位，才被视为非常加密。另一方面，对称密钥密码的补救措施是一样的，使用较短的密钥长度比大多数算法的 80 位小。同样，基于 elliptic curve 加密(ECC)的公钥密码，如 Elliptic Curve Digital Signature Algorithm (ECDSA)密码，还需要比 RSA 密码少。

1.2. 数字签名

篡改检测依赖于称为 *单向哈希*（也称为 *消息摘要*）的数学函数。单向哈希是固定长度的多个固定长度，具有以下特征：

- 哈希的值对于散列数据是唯一的。对数据的任何更改（甚至删除或更改单个字符）都会生成不同的值。
- 散列数据的内容无法从哈希中推断出来。

如 第 1.1.2 节 “公钥加密” 所述，可以使用私钥加密以及对应的解密公钥。虽然在加密敏感信息时不建议使用，但它是数字签名任何数据的关键部分。签名软件不加密数据本身，而是创建数据的单向哈希，然后使用私钥加密哈希。加密的哈希以及其他信息（如哈希算法）被称为数字签名。

图 1.3 “使用数字签名来验证数据完整性” 演示了使用数字签名来验证已签名数据的完整性。

图 1.3. 使用数字签名来验证数据完整性

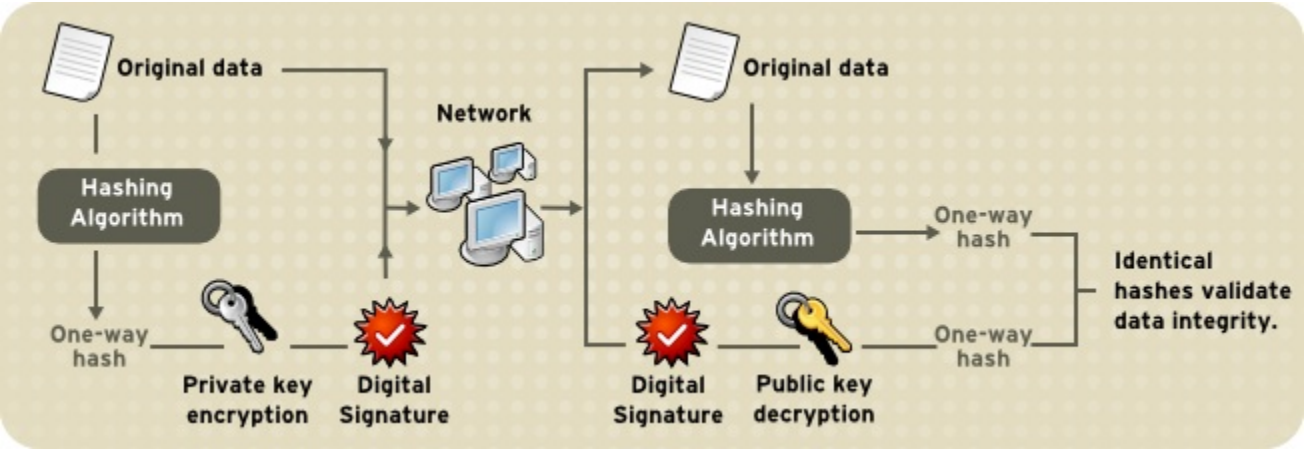


图 1.3 “使用数字签名来验证数据完整性” 显示转移到某些已签名数据的接收者的两个项目：原始数据和数字签名，这是通过签名人的私钥加密的原始数据的单向哈希。要验证数据的完整性，接收软件首先使用公钥解密哈希。然后，它使用生成原始哈希的同一哈希算法来生成同一数据的新单向哈希。（有关使用的哈希算法的信息与数字签名一起发送。） 最后，接收软件将新哈希与原始哈希进行比较。如果两个哈希匹配，则数据自签名以来没有改变。如果不匹配，则数据可能是因为签名以来已被篡改，或者可能会使用与签名者提供的公钥不匹配的私钥创建签名。

如果两个哈希匹配，则接收方可以确定用于解密数字签名的公钥与用于创建数字签名的私钥对应。确认签名人的身份还需要某种方式确认公钥属于特定实体。有关验证用户的详情，请参考 第 1.3 节 “证书和验证”。

数字签名与手写签名类似。在数据签名后，以后很难拒绝这样做，假设私钥没有被破坏。这种数字签名的质量提供了高度的非缓解；数字签名使签名人难以拒绝对数据进行签名。在某些情况下，数字签名是作为手写签名的法律绑定。

1.3. 证书和验证

1.3.1. 一个证书标识了 someone 或 something

证书是一个电子文件，用于识别个人、服务器、公司或其他实体并将该身份与公钥关联。与驱动程序的许可证或护照一样，证书可提供个人身份的可识别验证。公钥加密使用证书来解决身份模拟问题。

要获得个人 ID，如驱动程序的许可证，个人必须提供某种其他形式的识别方式，确认个人声称的身份是谁。证书的工作方式与相同。证书颁发机构(CA)验证身份和发布证书。CA 可以是独立的第三方，也可以是运行自己的证书的组织，如证书系统。用于验证身份的方法因所请求的证书类型的给定 CA 策略而异。在发布证书前，CA 必须通过其标准验证流程确认用户的身份。

CA 发布的证书将特定公钥绑定到证书标识的实体名称，如员工或服务器的名称。证书有助于防止使用假的公钥模拟。只有证书认证的公钥将与证书标识的实体拥有的对应私钥一起使用。

除了公钥外，证书始终包括它标识的实体的名称、到期日期、签发证书的 CA 的名称以及序列号。最重要的是，证书总是包括发布 CA 的数字签名。CA 的数字签名允许证书充当知道和信任 CA 但不知道证书标识的实体的用户的有效凭证。

有关 CA 角色的更多信息，请参阅 第 1.3.6 节 “CA 证书如何建立信任”。

1.3.2. 身份验证确认一个身份

身份验证 是确认身份的过程。对于网络交互，身份验证涉及由另一方识别方。可以通过多种方式通过网络使用身份验证。证书是这些方法之一。

网络交互通常在客户端（如 Web 浏览器和服务器）之间发生。*客户端身份验证* 指的是服务器识别客户端（假定使用软件的人员）。*服务器身份验证* 指的是客户端在网络地址中运行服务器（假定在网络地址中运行的服务器）的标识。

客户端和服务器身份验证不是证书支持的唯一验证形式。例如，电子邮件消息上的数字签名与标识发送者的证书相结合，可以验证邮件的发送者。同样，HTML 表单上的数字签名与标识签名者的证书相结合，可以提供由该证书标识给表单内容的人员的证据。除了身份验证之外，两个情况下的数字签名还确保了一定程度的非缓解；数字签名使签名者难以在以后不会发送电子邮件或表单。

客户端身份验证是大多数 Inets 或 extranets 中网络安全性的基本元素。客户端身份验证有两种形式：

基于密码的身份验证

几乎所有服务器软件都允许通过在授予服务器访问权限之前要求识别的名称和密码进行客户端身份验证。

基于证书的验证

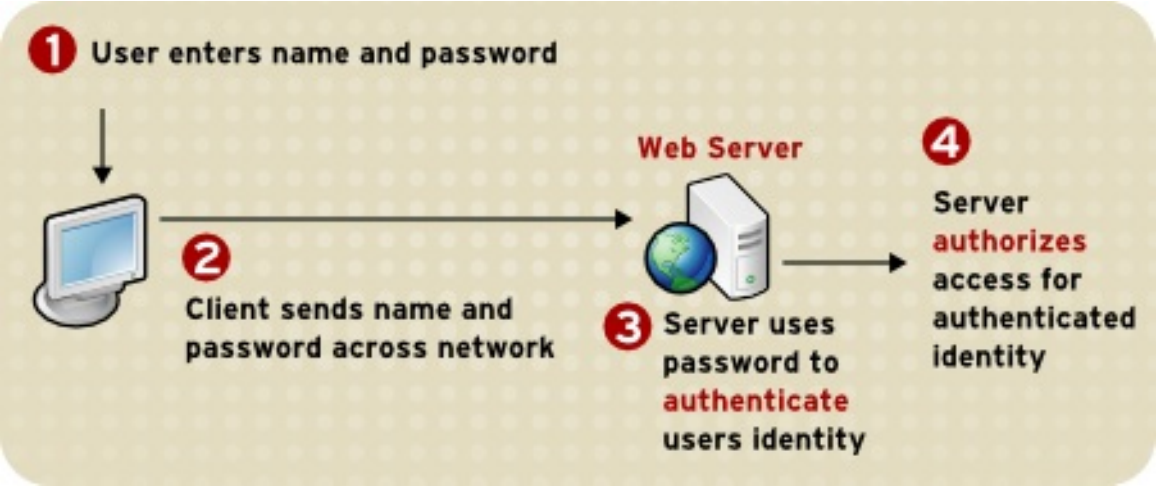
基于证书的客户端身份验证是 SSL/TLS 协议的一部分。客户端以数字方式签署随机生成的数据，并通过网络发送证书和签名数据。服务器验证签名并确认证书的有效性。

1.3.2.1. 基于密码的身份验证

图 1.4 "使用密码向服务器验证客户端" 显示使用用户名和密码验证用户的过程。这个示例假设如下：

- 用户已经信任服务器，无需身份验证，或基于 SSL/TLS 服务器身份验证。
- 用户请求由服务器控制的资源。
- 服务器需要客户端身份验证，然后允许访问所请求的资源。

图 1.4. 使用密码向服务器验证客户端



以下是此身份验证过程中的步骤：

1. 当服务器从客户端请求身份验证时，客户端会显示一个对话框，请求该服务器的用户名和密码。
2. 客户端在网络上发送名称和密码，可以是纯文本或加密的 SSL/TLS 连接。
3. 服务器在其本地密码数据库中查找名称和密码（如果匹配），则接受它们作为验证用户身份的证据。
4. 服务器决定识别的用户是否允许访问所请求的资源，如果允许客户端访问它。

通过此安排，用户必须为访问的每个服务器提供新密码，并且管理员必须跟踪每个用户的名称和密码。

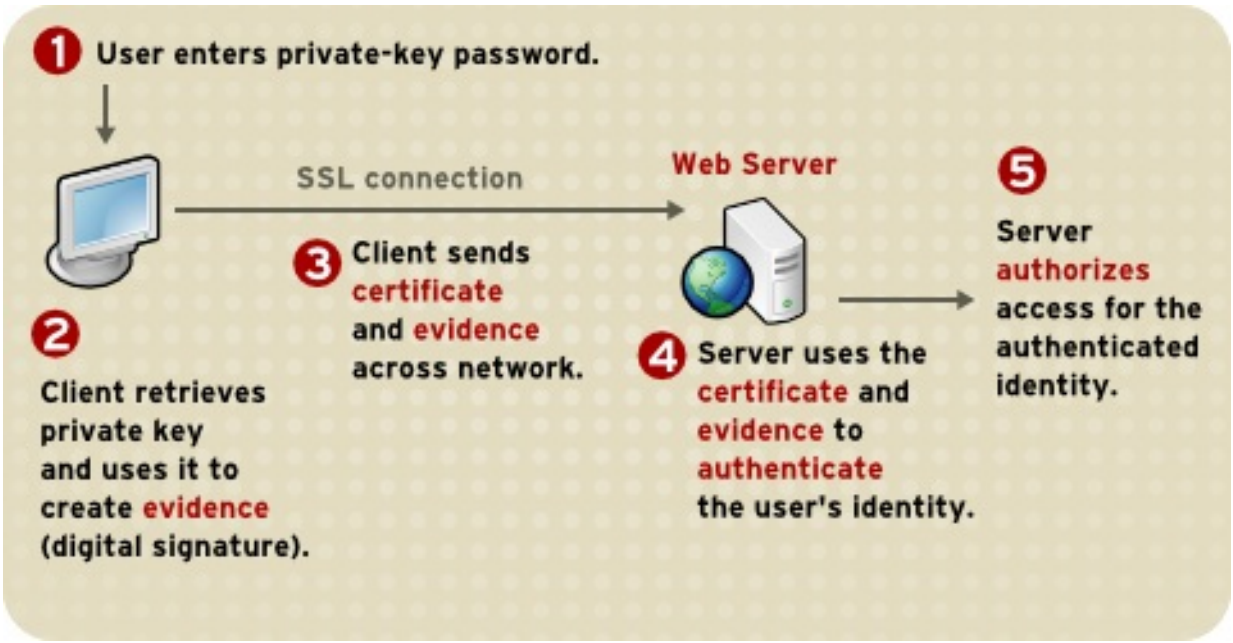
1.3.2.2. 基于证书的身份验证

基于证书的身份验证的一个优点是，它可用于用一种机制替换身份验证中的前三个步骤，允许用户提供一个密码（不通过网络发送），并允许管理员集中控制用户身份验证。这称为 *单点登录*。

图 1.5 "使用证书将客户端验证到服务器" 显示客户端身份验证如何使用证书和 SSL/TLS。为了验证用户到服务器，客户端以数字方式签署随机生成的数据，并通过网络发送证书和签名数据。服务器根据证书和签名数据中的数据验证用户身份。

与 图 1.4 "使用密码向服务器验证客户端" 一样，图 1.5 "使用证书将客户端验证到服务器" 假设用户已经信任服务器并请求资源，并且服务器在授予对请求资源的访问权限前已请求了客户端身份验证。

图 1.5. 使用证书将客户端验证到服务器



与 图 1.4 “使用密码向服务器验证客户端” 中的身份验证过程不同，图 1.5 “使用证书将客户端验证到服务器” 中的身份验证过程需要 SSL/TLS。图 1.5 “使用证书将客户端验证到服务器” 另外，也假定客户端具有可用于识别到服务器的客户端的有效证书。基于证书的验证是首选基于密码的身份验证，因为它基于具有私钥和知道密码的用户。但是，只有在未授权的人没有获得对用户的计算机或密码的访问权限、客户端软件的私钥数据库的密码时，才进行这两个假设，并且该软件设置为以合理频繁的间隔请求密码。

注意

无基于密码的身份验证或基于证书的身份验证地址与对各个机器或密码的物理访问相关的安全问题。公钥加密只能验证用于签署某些数据的私钥是否与证书中的公钥对应。用户负责保护计算机的物理安全性，并保持私钥密钥密码 secret。

这些是在 图 1.5 “使用证书将客户端验证到服务器” 中显示的身份验证步骤：

1. 客户端软件维护一个私钥数据库，该密钥对应于为该客户端发布的任何证书中的公钥。客户端在第一次在给定会话中需要访问这个数据库时（如首次尝试访问启用了证书的 SSL/TLS 服务器时），客户端会要求提供密码。

   输入此密码后，用户不需要为其余会话再次输入它，即使访问其他启用了 SSL/TLS 的服务器也是如此。

2. 客户端解锁 private-key 数据库，检索用户证书的私钥，并使用该私钥从客户端和服务器的输入随机生成数据。这个数据和数字签名是私钥的有效性的证据。数字签名只能使用该私钥创建，并可以根据签名数据使用对应的公钥进行验证，这对 SSL/TLS 会话是唯一的。

3. 客户端通过网络发送用户的证书和随机生成的数据。

4. 服务器使用证书和签名数据来验证用户的身份。

5. 服务器可以执行其他身份验证任务，例如检查客户端提供的证书存储在 LDAP 目录中的用户条目中。然后，服务器会评估识别的用户是否允许访问所请求的资源。此评估过程可以使用各种标准授权机制，可能在 LDAP 目录或公司数据库中使用其他信息。如果评估的结果是正数，服务器则允许客户端访问请求的资源。

证书替换客户端和服务器间交互的身份验证部分。单点登录不必要求用户不断在网络中发送密码，而是要求用户输入一次私钥数据库密码，而无需通过网络发送密码。对于会话的其余部分，客户端会显示用户的

证书，以向它遇到的每个新服务器验证用户。基于经过身份验证的用户身份的现有授权机制不会受到影响。

1.3.3. 使用证书

证书的目的是建立信任。其使用量因它们用来确保的信任类型而异。某些类型的证书用于验证 presenter 的身份；另一些的证书用于验证对象或项目是否已被篡改。

1.3.3.1. SSL/TLS

传输层安全性/安全套接字层(SSL/TLS)协议管理服务器和客户端之间的服务器身份验证、客户端身份验证和加密通信。SSL/TLS 在互联网上广泛使用，特别是涉及交换机密信息的交互，如信用卡号码。

SSL/TLS 需要 SSL/TLS 服务器证书。作为初始 SSL/TLS 握手的一部分，服务器向客户端提供证书以验证服务器的身份。身份验证使用公钥加密和数字签名来确认服务器是它声明为的服务器。服务器经过身份验证后，客户端和服务器使用对称密钥加密（非常快），加密会话剩余部分交换的所有信息，并检测任何篡改。

服务器可以配置为需要客户端身份验证以及服务器身份验证。在这种情况下，在服务器身份验证成功完成后，客户端还必须向服务器提供证书，以便在建立加密的 SSL/TLS 会话前验证客户端的身份。

有关通过 SSL/TLS 进行客户端验证的概述，以及如何与基于密码的身份验证的不同，请参阅 第 1.3.2 节 “身份验证确认一个身份”。

1.3.3.2. 签名和加密的电子邮件

有些电子邮件程序支持使用广泛接受的协议（称为安全多用途互联网邮件扩展(S/MIME)）进行数字签名和加密的电子邮件。使用 S/MIME 签名或加密电子邮件消息，要求消息的发件人具有 S/MIME 证书。

包含数字签名的电子邮件消息提供了一些保证，它是由在邮件标头中显示的名称的人员发送的，从而对发送者进行身份验证。如果电子邮件软件无法验证数字签名，则会警告用户。

数字签名对它所对应的消息是唯一的。如果收到的消息因发送的消息的任何方式不同，即使添加或删除单个字符，也无法验证数字签名。因此，签名的电子邮件还提供保证电子邮件没有被篡改。这种保证称为非缓解，这使得发件人难以拒绝发送邮件。这对业务通信非常重要。有关数字签名的工作方式的详情，请参考 第 1.2 节 “数字签名”。

s/MIME 还可以加密电子邮件消息，这对于某些业务用户来说非常重要。但是，对电子邮件使用加密需要仔细规划。如果加密的电子邮件邮件的接收者丢失了私钥并且无法访问密钥的备份副本，则加密的消息永远不会被解密。

1.3.3.3. 单点登录

网络用户通常需要记住他们所使用的各种服务的多个密码。例如，用户可能必须键入不同的密码来登录网络、收集电子邮件、使用目录服务、使用公司日历程序并访问各种服务器。对于用户和系统管理员，多个密码都是持续的难题。用户难以跟踪不同密码，往往选择不良密码，并且往往会将它们写在明显的位置。管理员必须跟踪每个服务器上的独立密码数据库，并处理与密码定期通过网络发送的事实相关的潜在安全问题。

解决此问题需要一些方法让用户使用单一密码登录一次，并获得对用户被授权使用的所有网络资源的访问权限，而无需通过网络发送任何密码。这个功能被称为单点登录。

客户端 SSL/TLS 证书和 S/MIME 证书都可以在全面的单点登录解决方案中扮演重要角色。例如，红帽产品支持的一个形式的单点登录依赖于 SSL/TLS 客户端身份验证。用户可以一次登录，使用单个密码登录到本地客户端的私钥数据库，并获得对用户被授权使用的所有 SSL/TLS 服务器的验证访问权限，而无需

通过网络发送任何密码。这种方法简化了用户的访问权限，因为它们不需要为每个新的服务器输入密码。它还简化了网络管理，因为管理员可以通过控制证书颁发机构(CA)列表而不是更长的用户和密码列表来控制访问权限。

除了使用证书外，完整的单点登录解决方案还必须满足与企业系统（如底层操作系统）进行交互的需求，它们依赖于密码或其他形式的身份验证。

1.3.3.4. 对象签名

许多软件技术支持一组称为 *对象签名* 的工具。对象签名使用公钥加密的标准技术，让用户可以获得与其下载代码的可靠信息的方式与缩小软件相关的信息相同。

最重要的是，对象签名可帮助用户和网络管理员实施有关在内部网上分发的软件或互联网的决策，例如，是否允许由给定实体签名的 Java 小程序在特定用户机器上使用特定的计算机功能。

使用对象签名技术签名的对象可以是 小程序或其他 Java 代码、JavaScript 脚本、插件或任何类型的文件。签名是一个数字签名。签名的对象及其签名通常存储在名为 JAR 文件的特殊文件中。

软件开发人员和希望使用对象签名技术为文件签名，必须首先获取对象签名证书。

1.3.4. 证书类型

证书系统能够为不同的格式和不同的格式生成不同类型的证书。规划需要哪些证书，并计划如何管理它们，包括确定需要哪些格式以及如何规划续订，对于管理 PKI 和证书系统实例非常重要。

此列表并不完整，存在用于 LDAP 目录、文件签名证书和其他子系统证书的双用证书的证书注册表单。这些表单通过证书管理器的端点页面（位于 **https://server.example.com:8443/ca/ee/ca** ）提供。

安装不同的证书系统子系统时，会生成基本所需的证书和密钥；例如，配置证书管理器为自签名根 CA 和内部 OCSP 签名、审计签名、SSL/TLS 服务器和代理用户证书生成 CA 签名证书。在 KRA 配置过程中，证书管理器会生成存储、传输、审计签名和代理证书。可以单独创建和安装其他证书。

表 1.1. 常见证书

证书类型	使用	示例
客户端 SSL/TLS 证书	用于通过 SSL/TLS 向服务器进行客户端身份验证。通常，假设客户端的身份与个人的身份相同，如员工。有关 SSL/TLS 客户端证书用于客户端验证的方式的信息，请参阅 第 1.3.2.2 节 “基于证书的身份验证”。客户端 SSL/TLS 证书也可以用作单点登录的一部分。	银行为客户提供 SSL/TLS 客户端证书，允许银行的服务器识别客户帐户并授权访问客户帐户。 公司提供了一个新员工的 SSL/TLS 客户端证书，允许公司的服务器识别该员工并授权对公司服务器的访问权限。
服务器 SSL/TLS 证书	用于通过 SSL/TLS 向客户端进行服务器身份验证。在不进行客户端身份验证的情况下，可以使用服务器身份验证。加密 SSL/TLS 会话需要服务器身份验证。如需更多信息，请参阅 第 1.3.3.1 节 “SSL/TLS”。	参与电子商业业的互联网通常会支持基于证书的服务器身份验证，以建立加密的 SSL/TLS 会话，并确保客户处理公司标识的网站。加密的 SSL/TLS 会话可确保通过网络发送的个人信息（如信用卡号）无法轻松截获。

证书类型	使用	示例
s/MIME 证书	用于签名和加密的电子邮件。与 SSL/TLS 客户端证书一样，假设客户端的身份与个人的身份相同，如员工。单个证书可以同时用作 S/MIME 证书和 SSL/TLS 证书；请参阅 第 1.3.3.2 节 “签名和加密的电子邮件”。s/MIME 证书也可以用作单点登录的一部分。	公司部署组合 S/MIME 和 SSL/TLS 证书，以专门验证员工身份，从而允许签名的电子邮件和 SSL/TLS 客户端身份验证，但不加密电子邮件。另一个公司只发出 S/MIME 证书以签发和加密处理敏感财务或法律问题的电子邮件。
CA 证书	用于识别 CA。客户端和服务器软件使用 CA 证书来确定可信任哪些其他证书。如需更多信息，请参阅 第 1.3.6 节 “CA 证书如何建立信任”。	Mozilla Firefox 中存储的 CA 证书确定可以验证哪些其他证书。管理员可以通过控制存储在每个用户的 Firefox 副本中的 CA 证书来实施企业安全策略。
对象签名证书	用于识别 Java 代码、JavaScript 脚本或其他签名文件的签名者。	软件公司经常为通过互联网分发的软件签名，为用户提供一些保证，确保该软件是公司的合法产品。使用证书和数字签名也可以让用户识别和控制下载的软件对计算机的访问权限类型。

1.3.4.1. CA 签署证书

每个证书管理器都有一个带有公钥/私钥对的 CA 签名证书，用于签署证书和证书撤销列表(CRL)。安装证书管理器时，会创建并安装此证书。

注意

有关 CRL 的更多信息，请参阅 第 2.4.4 节 “撤销证书和检查状态”。

证书管理器的状态是根或从属 CA 的状态，由其 CA 签名证书是自签名的，还是由另一个 CA 签名。自签名 root CA 设置他们用来发布证书的策略，如主题名称、可发布的证书类型，以及可以向谁签发证书。从属 CA 具有由另一个 CA 签名的 CA 签名证书，通常是 CA 层次结构中高于级别的 CA 签名证书（可能也可能不是 root CA）。如果证书管理器是 CA 层次结构中的从属 CA，则必须将 root CA 的签名证书导入到单独的客户端和服务器中，然后才能使用证书管理器向它们发布证书。

如果该 CA 发布的服务器或用户证书已安装在该客户端上，则必须将 CA 证书安装到客户端中。CA 证书确认服务器证书可以被信任。理想情况下，会安装证书链。

1.3.4.2. 其他签名证书

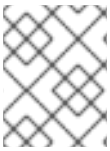
其他服务（如在线证书状态协议(OCSP)响应器服务和 CRL 发布）可以使用 CA 证书以外的签名证书。例如，单独的 CRL 签名证书可用于签署 CA 发布的撤销列表，而不使用 CA 签名证书。

注意

有关 OCSP 的详情，请参考 第 2.4.4 节 “撤销证书和检查状态”。

1.3.4.3. SSL/TLS 服务器和客户端证书

服务器证书用于安全通信，如 SSL/TLS 和其他安全功能。服务器证书用于在操作期间和加密数据期间验证自己；客户端证书向服务器验证客户端。



注意

具有第三方发布的签名证书的 CA 可能无法发布服务器证书。第三方 CA 可能有规则，以禁止其从属者发出服务器证书。

1.3.4.4. 用户证书

最终用户证书是客户端证书的子集，用于识别用户到服务器或系统。可以为用户分配用于安全通信的证书，如 SSL/TLS 和其他功能，如加密电子邮件或单点登录。特殊用户（如证书系统代理）可以被授予客户端证书来访问特殊服务。

1.3.4.5. dual-Key Pairs

双密钥对是一组两个私钥和公钥，其中一个集合用于签名，另一个用于加密。这些双密钥用于创建双证书。双证书注册表是证书管理器端点中列出的标准表单之一。

在生成双密钥对时，在为签名和加密生成单独的证书时，将证书配置文件设置为正常工作。

1.3.4.6. 跨证书

证书系统可以发布、导入和发布跨对 CA 证书。使用跨对证书时，一个 CA 签署并签发对第二个 CA 的跨对证书，第二个 CA 签署并向第一个 CA 发布跨对证书。然后，两个 CA 都存储或发布两个证书作为 **跨证书** 条目。

可以进行桥接证书来遵循不是由 CA 发布的证书，该证书没有串联到 root CA。通过跨对 CA 证书在证书系统 CA 和另一个 CA 之间建立信任，可以下载跨对证书并用来信任其他 CA 发布的证书。

1.3.5. 证书内容

证书内容根据 X.509 v3 证书规范进行组织，该规范由国际电信联合(ITU)推荐，这是一个国际标准正文。

用户通常不需要关注证书的确切内容。但是，使用证书的系统管理员可能需要一定了解它们中包含的信息。

1.3.5.1. 证书数据格式

证书请求和证书可以使用多种不同的格式创建、存储和安装。所有这些格式均符合 X.509 标准。

1.3.5.1.1. 二进制

可识别以下二进制格式：

- *DER 编码的证书*.这是单个二进制 DER 编码的证书。
- *PKCS #7 证书链*.这是一个 PKCS #7 **SignedData** 对象。**SignedData** 对象中的唯一重要字段是证书；例如，签名和内容将被忽略。PKCS #7 格式允许一次下载多个证书。
- *Netscape 证书序列*.这是在 PKCS #7 **ContentInfo** 结构中下载证书链的更简单格式，嵌套一系列证书。**contentType** 字段的值应该是 **netscape-cert-sequence**，而 content 字段具有以下结构：

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

  此格式允许同时下载多个证书。

1.3.5.1.2. 文本

任何二进制格式都可以以文本形式导入。文本表单以以下行开头：

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

在此行之后，证书数据可以采用描述的任何二进制格式。这个数据应该采用 base-64 编码，如 RFC 1113 所述。证书信息后跟这一行：

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

1.3.5.2. 区分名称

X.509 v3 证书将区分名称(DN)绑定到公钥。DN 是一系列名称值对，如 **uid=doe**，它唯一标识一个实体。这也称为证书 *主题名称*。

这是 Example Corp 的员工的 DN 示例：

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

在这个 DN 中，**uid** 是用户名，**cn** 是用户的通用名称，**o** 是机构或公司名称，**c** 是国家/地区。

DNS 可能包括各种其他名称值对。它们用于识别支持轻量级目录访问协议(LDAP)的目录中的证书主题和条目。

管理 DN 结构的规则可能比较复杂；有关 DN *的综合信息*，请参阅 http://www.ietf.org/rfc/rfc4514.txt。

1.3.5.3. Typical 证书

每个 X.509 证书由两个部分组成：

data 部分

本节包括以下信息：

- 证书支持的 X.509 标准的版本号。
- 证书的序列号。CA 发布的每个证书都有一个序列号，该序列号在该 CA 发布的证书之间是唯一的。
- 有关用户公钥的信息，包括所用的算法和密钥本身的表示形式。
- 发布证书的 CA 的 DN。
- 证书有效的期间；例如，2004 年 11 月 15 日下午 1:00 到 1:00 p.m 之间的时段。2022 年 11 月 15 日。
- 证书主题的 DN （也称为主题名称）；例如，在 SSL/TLS 客户端证书中，这是用户的 DN。
- 可选 *的证书扩展*，它可能会提供客户端或服务器使用的额外数据。例如：
  - Netscape 证书类型扩展表示证书类型，如 SSL/TLS 客户端证书、SSL/TLS 服务器证书或签名电子邮件的证书
  - 主题备用名称(SAN)扩展将证书链接到一个或多个主机名

证书扩展也可以用于其他目的。

签名部分

本节包括以下信息：

- 发布 CA 用来创建自己的数字签名的加密算法或密码。
- CA 的数字签名，将证书中的所有数据哈希到一起，并使用 CA 的私钥对其进行加密。

以下是以可读用户友善格式显示的证书的数据和签名部分：

```
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 Before: Fri Oct 17 18:36:25 1997
   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:
       00:ca:fa:79:98:8f:19:f8:d7:de:e4:49:80:48:e6:2a:2a:86:
       ed:27:40:4d:86:b3:05:c0:01:bb:50:15:c9:de:dc:85:19:22:
       43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8:51:a3:a1:00:
       98:ce:7f:47:50:2c:93:36:7c:01:6e:cb:89:06:41:72:b5:e9:
       73:49:38:76:ef:b6:8f:ac:49:bb:63:0f:9b:ff:16:2a:e3:0e:
       9d:3b:af:ce:9a:3e:48:65:de:96:61:d5:0a:11:2a:a2:80:b0:
       7d:d8:99:cb:0c:99:34:c9:ab:25:06:a8:31:ad:8c:4b:aa:54:
       91:f4:15
     Public Exponent: 65537 (0x10001)
  Extensions:
   Identifier: Certificate Type
     Critical: no
     Certified Usage:
     TLS Client
   Identifier: Authority Key Identifier
     Critical: no
     Key Identifier:
       f2:f2:06:59:90:18:47:51:f5:89:33:5a:31:7a:e6:5c:fb:36:
       26:c9
  Signature:
   Algorithm: PKCS #1 MD5 With RSA Encryption
  Signature:
6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c:01:69:8e:54:65:fc:06:
30:43:34:d1:63:1f:06:7d:c3:40:a8:2a:82:c1:a4:83:2a:fb:2e:8f:fb:
f0:6d:ff:75:a3:78:f7:52:47:46:62:97:1d:d9:c6:11:0a:02:a2:e0:cc:
2a:75:6c:8b:b6:9b:87:00:7d:7c:84:76:79:ba:f8:b4:d2:62:58:c3:c5:
b6:c1:43:ac:63:44:42:fd:af:c8:0f:2f:38:85:6d:d6:59:e8:41:42:a5:
4a:e5:26:38:ff:32:78:a1:38:f1:ed:dc:0d:31:d1:b0:6d:67:e9:46:a8:
d:c4
```

以下是 base-64 编码格式的相同证书：

```
-----BEGIN CERTIFICATE-----
MIICKzCCAZSgAwIBAgIBAzANBgkqhkiG9w0BAQQFADA3MQswCQYDVQQGEwJVUzER
MA8GA1UEChMITmV0c2NhcGUxFTATBgNVBAsTDFN1cHJpeWEncyBDQTAeFw05NzEw
MTgwMTM2MjVaFw05OTEwMTgwMTM2MjVaMEgxCzAJBgNVBAYTAlVTMREwDwYDVQQK
EwhOZXRzY2FwZTENMAsGA1UECxMEUHViczEXMBUGA1UEAxMOU3VwcmI5YSBTaGV0
dHkwgZ8wDQYJKoZIhvcNAQEFBQADgY0AMIGJAoGBAMr6eZiPGfjX3uRJgEjmKiqG
7SdATYazBcABu1AVyd7chRkiQ31FbXFOGD3wNktbf6hRo6EAmM5/R1AskzZ8AW7L
iQZBcrXpc0k4du+2Q6xJu2MPm/8WKuMOnTuvzpo+SGXelmHVChEqooCwfdiZywyZ
NMmrJgaoMa2MS6pUkfQVAgMBAAGjNjA0MBEGCWCGSAGG+EIBAQQEAwIAgDAfBgNV
HSMEGDAWgBTy8gZZkBhHUfWJM1oxeuZc+zYmyTANBgkqhkiG9w0BAQQFAAOBgQBt
I6/z07Z635DfzX4XbAFpjlRl/AYwQzTSYx8GfcNAqCqCwaSDKvsuj/vwbf91o3j3
UkdGYpcd2cYRCgKi4MwqdWyLtpuHAH18hHZ5uvi00mJYw8W2wUOsY0RC/a/IDy84
hW3WWehBUqVK5SY4/zJ4oTjx7dwNMdGwbWfpRqjd1A==
-----END CERTIFICATE-----
```

1.3.6. CA 证书如何建立信任

CA 验证身份和发布证书。它们可以是独立的第三方，也可以是运行自己的证书的组织，如证书系统。

支持证书的任何客户端或服务器软件均维护可信 CA 证书的集合。这些 CA 证书决定了软件可以信任或验证哪些证书签发者。在最简单的情形中，软件只能验证其具有证书的 CA 发布的证书。可信 CA 证书也可以作为 CA 证书链的一部分，每个证书都由 CA 在证书层次结构中的上面发布的。

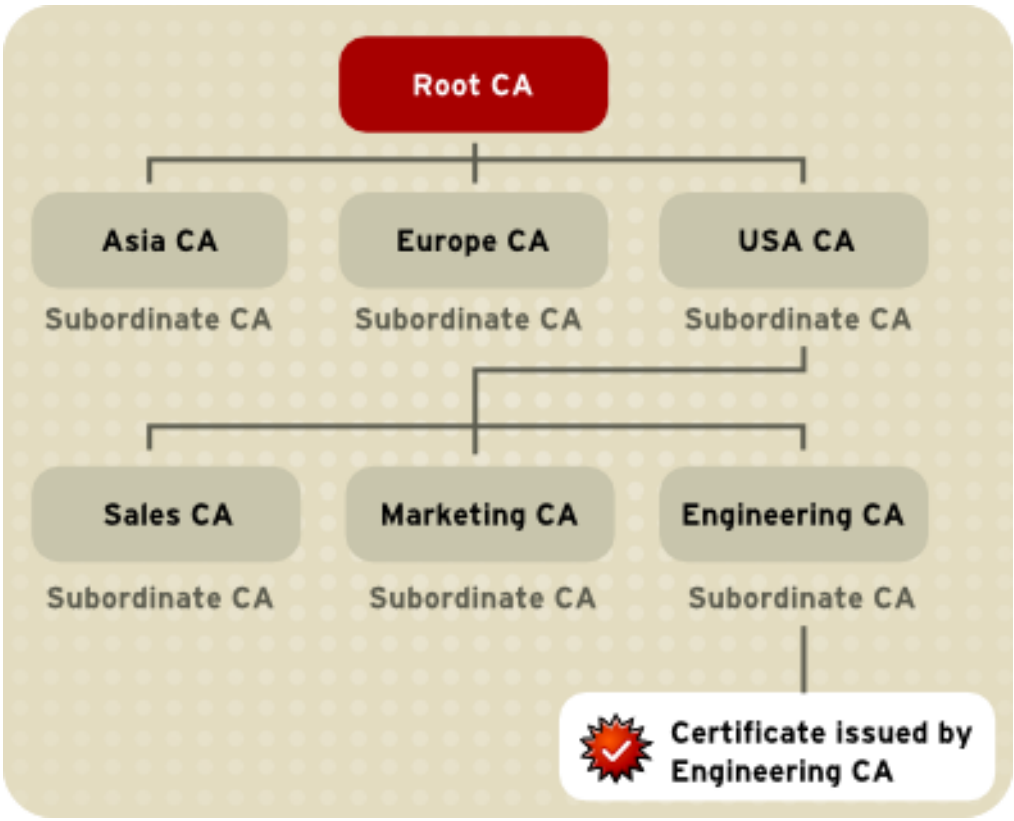
以下章节解释了证书层次结构和证书链如何确定软件可以信任哪些证书。

1.3.6.1. CA 层次结构

在大型机构中，发布证书的责任可以委派给多个不同的 CA。例如，为单个 CA 维护所需的证书数量可能太大；不同的机构单元可能具有不同的策略要求；或者，CA 可能需要物理位于与发出证书的人员相同的地理位置。

这些证书职责可以划分到从属的 CA 中。X.509 标准包括一个设置 CA 层次结构的模型，如 图 1.6 “证书颁发机构层次结构示例” 所示。

图 1.6. 证书颁发机构层次结构示例



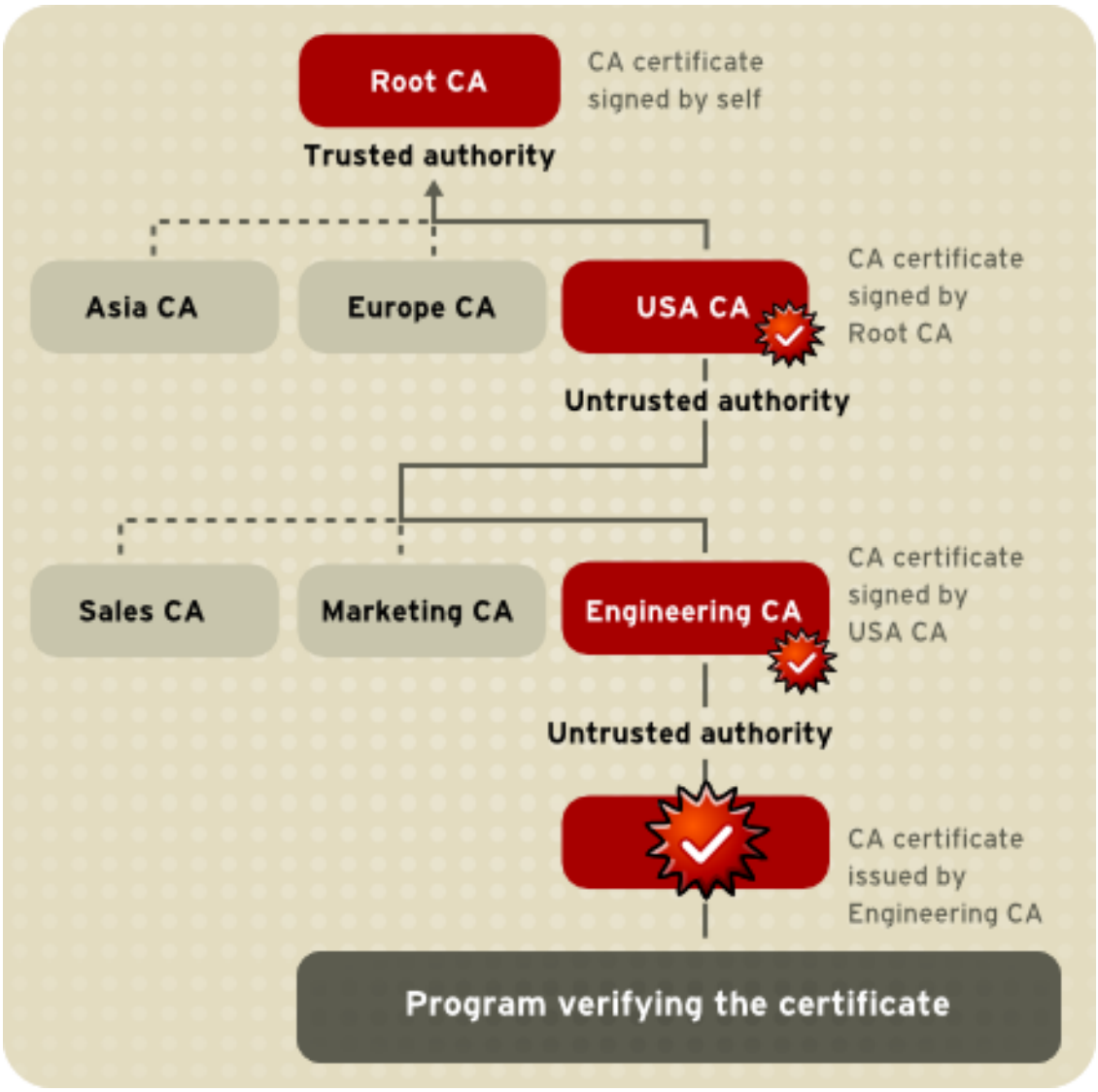
root CA 位于层次结构的顶部。root CA 的证书是一个 *自签名证书* ；即，证书由证书标识的同一实体进行数字签名。直接分配给 root CA 的 CA 具有由 root CA 签名的 CA 证书。层次结构中从属 CA 下的 CA 具有其由更高级别的从属 CA 签名的 CA 证书。

机构在如何设置 CA 层次结构方面具有很大的灵活性；图 1.6 “证书颁发机构层次结构示例” 只显示一个例子。

1.3.6.2. 证书链

CA 层次结构反映在证书链中。*证书链* 是由连续 CA 发布的证书系列。图 1.7 “证书链示例” 显示源自证书的证书链，该链通过两个从属 CA 证书向 root CA 的 CA 证书标识实体，具体取决于 图 1.6 “证书颁发机构层次结构示例” 中显示的 CA 层次结构。

图 1.7. 证书链示例



证书链跟踪层次结构中从分支到层次结构根目录的证书路径。在证书链中，会出现以下情况：

- 每个证书后跟其签发者的证书。
- 每个证书都包含该证书签发者的名称(DN)，它与链中下一个证书的主题名称相同。

  在 图 1.7 “证书链示例” 中，**工程 CA** 证书包含签发该证书的 CA (**美国CA**)的 DN。**美国 CA** 的 DN 也是链中下一个证书的 主题名称。

- 每个证书都使用其签发者的私钥签名。签名可以使用签发者的证书中的公钥进行验证，这是链中的下一个证书。

  在 图 1.7 “证书链示例” 中，**美国 CA** 的证书中的公钥可用于验证 **工程 CA 的证书中的美国 CA** 的数字签名。

1.3.6.3. 验证证书链

证书链验证可确保给定证书链得到良好格式、有效、正确签名和可信。以下过程描述涵盖了组成和验证证书链的最重要步骤，从提供的证书进行验证开始：

1. 针对 verifier 的系统时钟提供的当前时间检查证书有效期。

2. 签发者的证书位于。源可以是该客户端或服务器上 verifier 的本地证书数据库，也可以是主题提供的证书链，如 SSL/TLS 连接。

3. 证书签名请求使用签发者证书中的公钥进行验证。

4. 将服务的主机名与 Subject Alternative Name (SAN)扩展进行比较。如果证书没有这样的扩展，则会将主机名与主题的 CN 进行比较。

5. 系统会验证证书的基本约束要求，即证书是 CA 以及允许它签名的子公司数量。

6. 如果签发者的证书被验证器的证书在验证器的证书数据库中信任，则验证会在这里成功停止。否则，会检查签发者的证书，以确保它包含证书类型扩展中的相应从属 CA，并且链验证从这个新证书开始。图 1.8 “验证到根 CA 的证书链” 提供了此过程的示例。

图 1.8. 验证到根 CA 的证书链

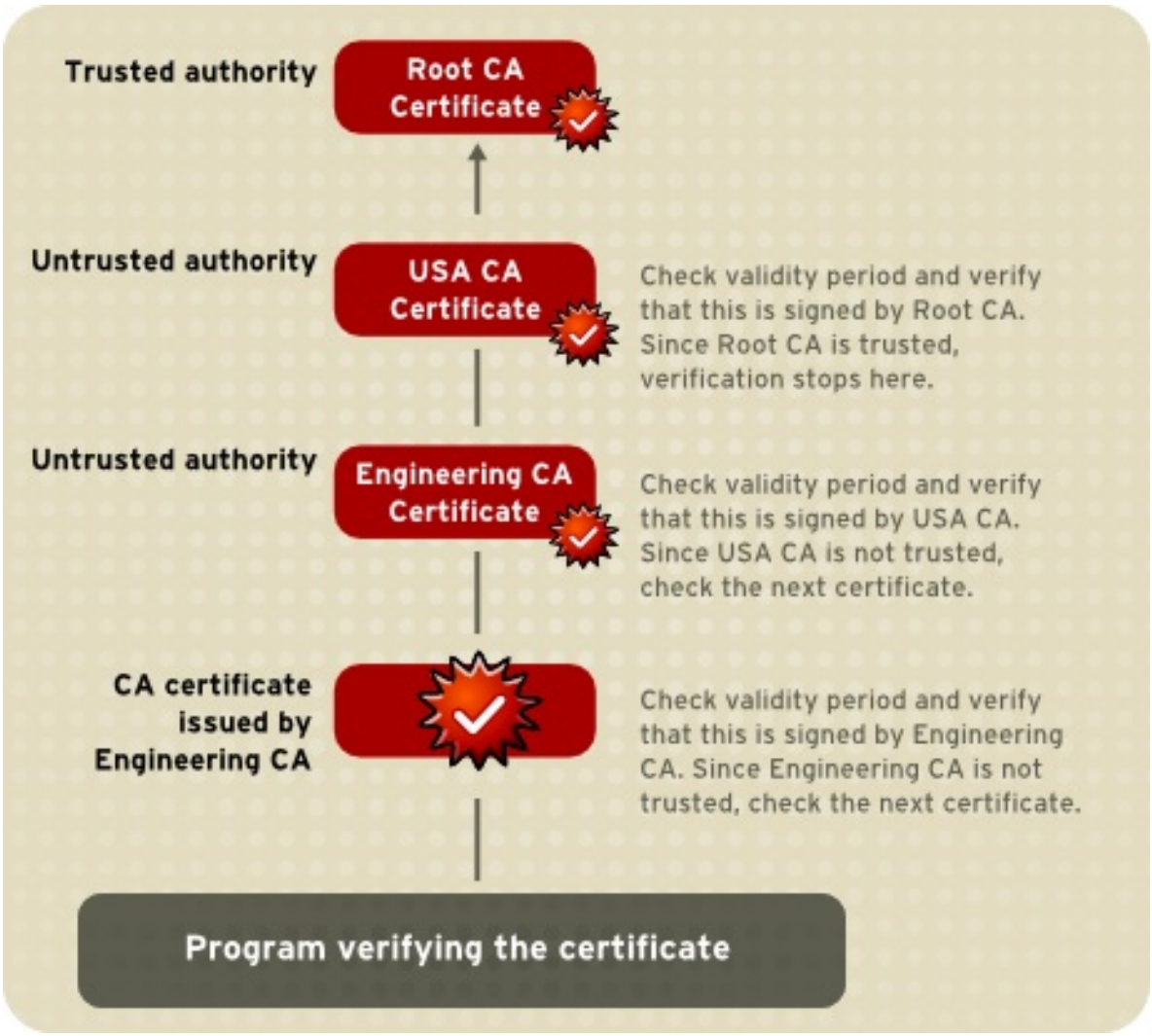
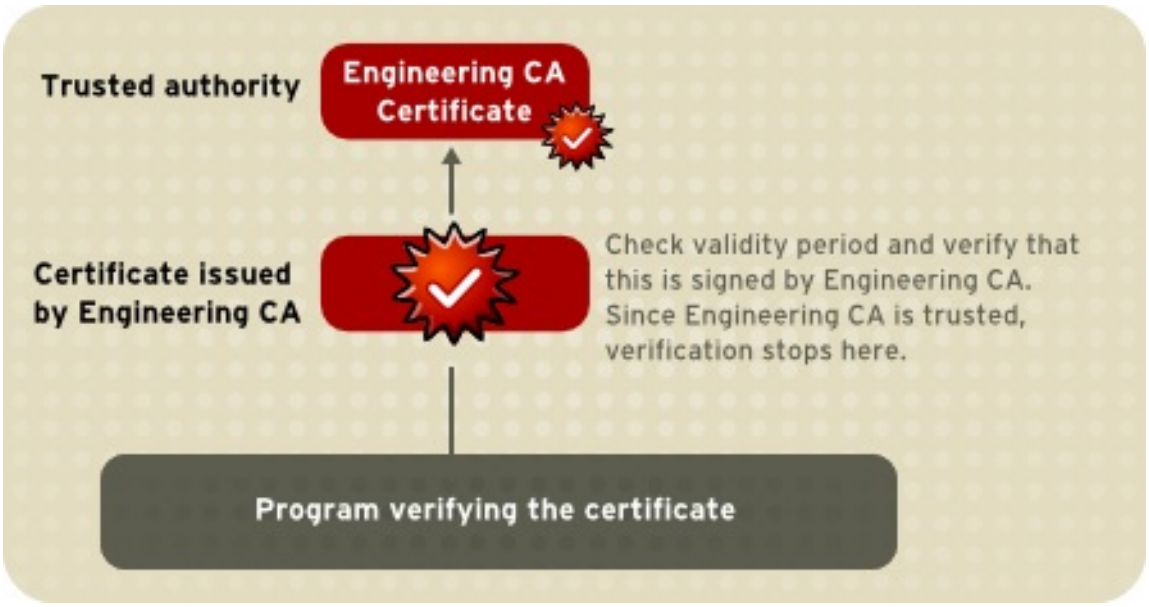


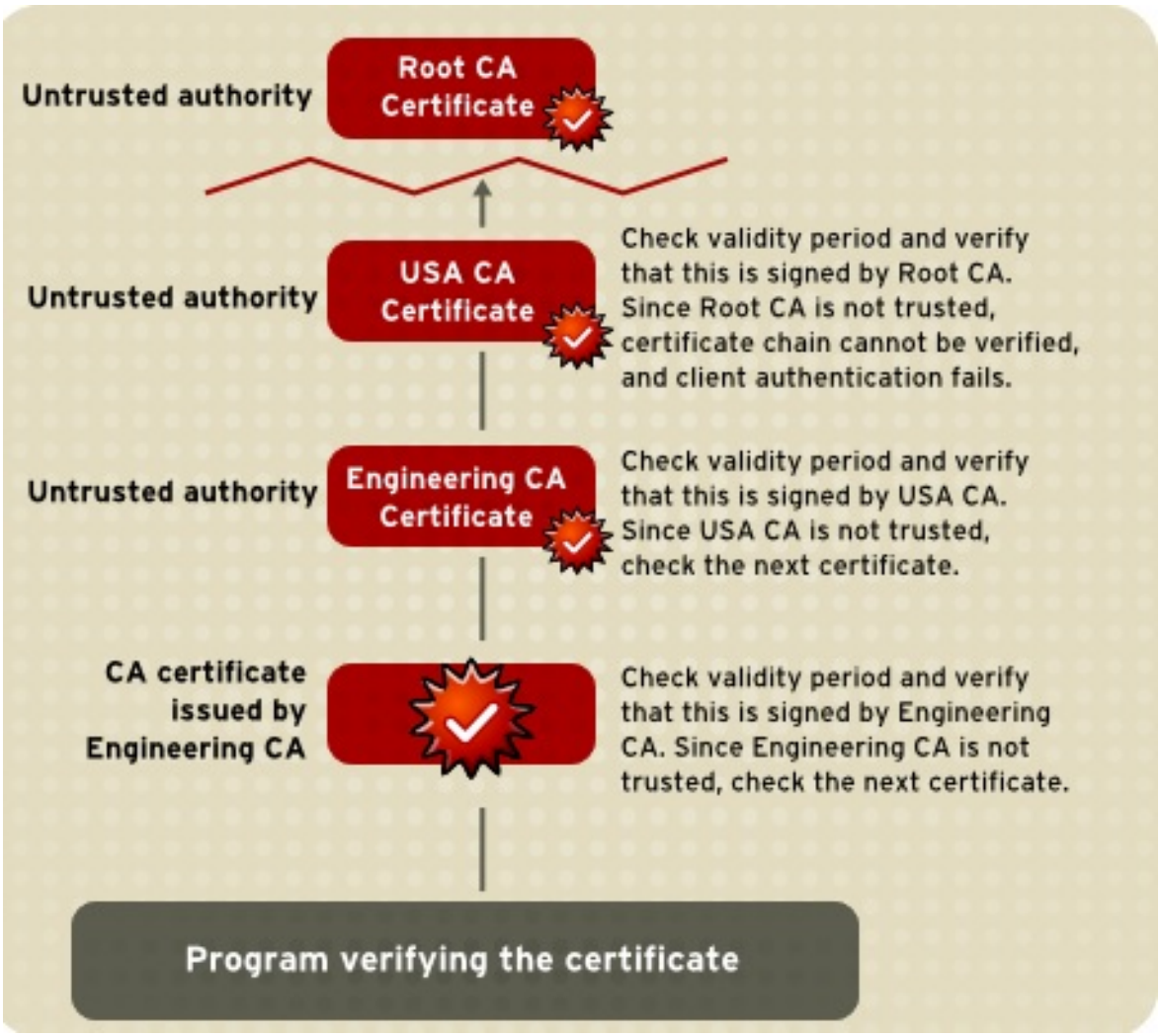
图 1.8 “验证到根 CA 的证书链” 演示了当验证器的本地数据库中仅包含根 CA 时会发生什么。如果一个中间 CA 的证书（如 **Engineering CA** ）位于验证器的本地数据库中，则验证会停止使用该证书，如 图 1.9 “验证证书链到中间 CA” 所示。

图 1.9. 验证证书链到中间 CA



过期的有效期日期、无效签名或在证书链的任意点上发布 CA 的证书会导致身份验证失败。图 1.10 "无法验证的证书链" 显示在验证器的本地数据库中不包含 root CA 证书或任何中间 CA 证书时验证会失败。

图 1.10. 无法验证的证书链



1.3.7. 证书状态

有关证书撤销列表(CRL)的更多信息，请参阅 第 2.4.4.2.1 节 “CRL”

有关在线证书状态协议(OCSP)的更多信息，请参阅 第 2.4.4.2.2 节 “OCSP 服务”

1.4. 证书生命周期

证书在很多应用程序中使用，从加密电子邮件到访问网站。证书生命周期有两个主要阶段：发布的时间（颁发和注册）以及证书不再有效（续订或撤销）的期间。还可以在其周期内管理证书。将证书提供给其他应用的信息是 *发布证书*，然后备份密钥对，以便在丢失证书时恢复证书。

1.4.1. 证书颁发

发布证书的过程取决于发布证书的 CA 以及将使用它的目的。发出非数字标识形式在类似的方式有所不同。获取库卡的要求与获取驱动程序许可证的要求不同。同样，不同的 CA 有不同的步骤来发出不同类型的证书。接收证书的要求可以是电子邮件地址或用户名和密码的简单，以汇总文档、后台检查和个人面试。

根据机构的策略，发布证书的过程对用户完全透明，需要大量用户参与和复杂的流程。通常，发布证书的流程应该非常灵活，因此组织可以根据变化的需求对其进行定制。

1.4.2. 证书过期和续订

与驱动程序的许可证一样，证书指定了有效时间。尝试在有效期之前或之后使用证书进行身份验证会失败。管理证书过期和续订是证书管理策略的重要部分。例如，管理员可能希望在证书即将过期时自动获得通知，以便在不中断系统操作的情况下完成适当的续订过程。续订过程可能涉及重复使用相同的公钥对或发布新密钥对。

此外，可能需要在证书过期前撤销证书，例如当员工离开公司或迁移到公司内的不同单元时，可能需要撤销证书。

证书撤销可以通过几种不同方式处理：

验证证书是否存在于目录中

可以配置服务器，以便身份验证过程检查目录是否存在所呈现的证书。当管理员撤销证书时，证书可以从目录中自动删除，使用该证书的后续身份验证尝试将失败，即使证书在所有其他方面都保持有效。

证书撤销列表(CRL)

撤销的证书列表(CRL)可以定期发布到目录中。CRL 可以作为身份验证过程的一部分进行检查。

实时状态检查

每当提供证书进行身份验证时，也可以直接检查发出 CA。这个过程有时被称为实时状态检查。

在线证书状态协议

可以配置在线证书状态协议(OCSP)服务来确定证书的状态。

有关续订证书的详情，请参考 第 2.4.2 节 “续订证书”。有关撤销证书的更多信息，包括 CRL 和 OCSP，请参阅 第 2.4.4 节 “撤销证书和检查状态”。

1.5. 密钥管理

在发布证书之前，必须生成包含证书的公钥，并且必须生成对应的私钥。有时，为签名操作发布单个个人证书和密钥对以及加密操作的另一个证书和密钥对可能很有用。单独的签名和加密证书只在本地机器中保留私有签名密钥，从而提供最大非填充。这也有助于在用户丢失原始密钥或离开公司时检索私钥在一些中央位置备份私钥。

密钥可以由客户端软件生成，或者由 CA 集中管理并通过 LDAP 目录分发到用户。与任一方法相关的成本。本地密钥生成提供了最大的非排期，但可能涉及用户对发出过程进行更多的参与。灵活的关键管理功能对于大多数组织至关重要。

密钥恢复，或者在明确定义的条件下检索加密密钥备份的能力可能是证书管理的一个重要部分，具体取决于组织如何使用证书。在某些 PKI 设置中，必须先同意一些授权人员，然后才能恢复加密密钥，以确保密钥仅恢复给授权部的合法所有者。当信息加密时，可能需要恢复密钥，且只能由丢失的密钥解密。

第 2 章 RED HAT CERTIFICATE SYSTEM 简介

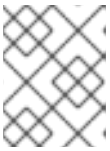
每个常见的 PKI 操作（如发出、续订和撤销证书）、归档和恢复密钥；通过 Red Hat Certificate System 中的交互子系统执行发布 CRL 和验证证书状态。本章介绍了各个子系统的功能以及它们协同工作以建立强大和本地 PKI 的方式。

2.1. 证书系统子系统检查

Red Hat Certificate System 提供五个不同的子系统，每个子系统都侧重于 PKI 部署的不同方面：

- *名为 Certificate Manager 的证书颁发机构。CA 是 PKI 的核心；它发出并撤销所有证书。证书管理器也是证书系统的核心。通过建立可信子系统 的安全域，它会建立和管理其他子系统之间的关系。*

- *密钥恢复机构(KRA)。证书基于特定且唯一的密钥对创建。如果私钥丢失，则该密钥用于访问（如加密电子邮件）的数据也会丢失，因为它无法访问。KRA 存储密钥对，以便可以根据恢复的密钥生成新的相同证书，即使私钥丢失或损坏，也可以访问所有加密的数据。*



注意

在以前的证书系统版本中，KRA 也被称为数据恢复管理器(DRM)。有些代码、配置文件条目、Web 面板和其他资源可能仍然使用术语 DRM 而不是 KRA。

- *在线证书状态协议(OCSP)响应器。OCSP 验证证书是否有效且未过期。此函数也可以由 CA 完成， 它具有内部 OCSP 服务，但使用外部 OCSP 响应程序会降低发布 CA 的负载。*

- *令牌密钥服务 (TKS)。TKS 根据令牌 CCID、私有信息和定义的算法生成密钥。TPS 使用这些派生的密钥来格式化令牌并在令牌中注册证书。*

- *令牌处理系统 (TPS)。TPS 与外部令牌（如智能卡）直接交互，并通过本地客户端、企业安全客户端(ESC) 管理这些令牌上的密钥和证书。当存在令牌操作时，ESC 会联系 TPS，并且 TPS 根据需要与 CA、KRA 或 TKS 交互，然后以企业安全客户端的方式将信息发回到令牌。*

即使安装了所有可能子系统，证书系统的核心仍然是 CA （或 CA），因为它们最终处理所有与证书相关的请求。其他子系统连接到 wheel 中的 CA 或 CA，如 spoke。这些子系统协同工作，共同创建公钥基础架构(PKI)。根据安装哪些子系统，PKI 可以通过以下两种方式之一（或两者）运行：

- *令牌管理系统或 TMS 环境，用于管理智能卡。这需要一个 CA、TKS 和 TPS， 它有一个可选的 KRA 用于服务器端密钥生成。*

- *传统的 非令牌管理系统 或非TMS 环境，用于管理智能卡之外的环境中使用的证书，通常是在软件数据库中。非TMS 至少需要一个 CA，但非TMS 环境也可以使用 OCSP 响应器和 KRA 实例。*

2.2. 证书系统子系统概述

2.2.1. 独立和共享实例

Red Hat Certificate System 支持为所有子系统部署单独的 PKI 实例：

- *单独的 PKI 实例作为基于 Java 的 Apache Tomcat 实例运行。*

- *单独的 PKI 实例包含一个 PKI 子系统(CA、KRA、OCSP、TKS 或 TPS)。*

- *如果在同一物理机或虚拟机(VM)上共存，则单独的 PKI 实例必须使用唯一的端口。*

或者，证书系统支持部署共享 PKI 实例：

- 共享 PKI 实例也作为基于 Java 的 Apache Tomcat 实例运行。
- 包含单个 PKI 子系统的共享 PKI 实例与单独的 PKI 实例相同。
- 共享 PKI 实例可能包含每种 PKI 子系统之一的组合：
  - 仅限 CA
  - 仅限 TKS
  - CA 和 KRA
  - CA 和 OCSP
  - TKS 和 TPS
  - CA、KRA、TKS 和 TPS
  - CA, KRA, OCSP, TKS, 和 TPS
  - etc.
- 共享 PKI 实例允许其实例中包含的所有子系统共享同一端口。
- 如果多个端口位于同一台物理机或虚拟机上，则共享 PKI 实例必须使用唯一端口。

2.2.2. 实例安装前提条件

2.2.2.1. 目录服务器实例可用性

在安装证书系统实例之前，必须使用本地或远程红帽目录服务器 LDAP 实例。有关安装红帽目录服务器的说明，请参阅 红帽目录服务器安装指南。

2.2.2.2. PKI 软件包

Red Hat Certificate System 由以下列出的软件包组成：

- 以下基本软件包组成了证书系统的核心，并包括在基本 Red Hat Enterprise Linux 软件仓库中：
  - pki-core
    - pki-base
    - pki-base-java
    - pki-ca
    - pki-javadoc
    - pki-kra
    - pki-server
    - pki-symkey

        - pki-tools

- 以下列出的软件包在基本 Red Hat Enterprise Linux 订阅频道 中不可用。要安装这些软件包，您必须附加一个 Red Hat Certificate System 订阅池并启用 RHCS 存储库。如需更多信息，请参阅 第 6.8 节 "附加红帽订阅并启用证书系统软件包存储库"。

    - pki-core

        - pki-console
        - pki-ocsp
        - pki-tks
        - pki-tps

    - redhat-pki

        - redhat-pki: 包含 pki-core 模块的所有软件包。如果要单独选择 redhat-pki 软件包，建议禁用 pki-core 模块。
        - redhat-pki-console-theme
        - redhat-pki-server-theme

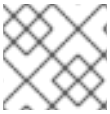
使用 Red Hat Enterprise Linux 8 系统（可选）使用 第 4 章 支持的平台中列出的受支持硬件安全模块配置的系统，并确保在安装 Red Hat Certificate System 前，所有软件包都保持最新状态。

要安装所有证书系统软件包(pki-javadoc除外)，使用 **dnf** 安装 redhat-pki metapackage：

```
# dnf install redhat-pki
```

或者，您也可以根据需要安装一个或多个顶级 PKI 子系统软件包；请参阅上面的列表。如果使用此方法，请确保也要安装 redhat-pki-server-theme 软件包，以及可选的 redhat-pki-console-theme 和 pki-console 来使用 PKI 控制台。

最后，开发人员和管理员可能还希望安装 JSS 和 PKI javadoc (jss-javadoc 和 pki-javadoc)。



注意

jss-javadoc 软件包要求您在 **Subscription Manager** 中启用 Server-Optional 存储库。

2.2.2.3. 实例安装和配置

pkispawn 命令行工具用于安装和配置新的 PKI 实例。它消除了对独立安装和配置步骤的需求，并可以作为批处理进程以交互方式运行，或者全部组合（带有提示密码的批量进程）。实用程序不提供安装或配置基于浏览器的图形界面的方法。

有关使用信息，请使用 **pkispawn --help** 命令。

pkispawn 命令：

1. 从纯文本配置文件(**/etc/pki/default.cfg**)读取其默认 **name=value** 对。
2. 以互动方式或自动覆盖指定的任何对，并以 Python 字典的形式存储最终结果。

3. *执行一系列有序的 scriptlet 来执行子系统和实例安装。*

4. *配置 scriptlet 将 Python 字典打包为 JavaScript Object Notation (JSON)数据对象，然后传递给基于 Java 的配置 servlet。*

5. *配置 servlet 利用此数据来配置新的 PKI 子系统，然后将控制权传递回 **pkispawn** 可执行文件，从而完成 PKI 设置。最终部署文件的副本保存在 **/var/lib/pki/instance_name/ <subsystem> /registry/<subsystem>/deployment.cfg**中*

*详情请查看 **pkispawn** 手册页。*

*默认配置文件 **/etc/pki/default.cfg** 是一个纯文本文件，其中包含在上述过程开始时读取的默认安装和配置值。它由 **name=value** 对组成，分为 **[DEFAULT]**, **[Tomcat]**, **[CA]**, **[KRA]**, **[OCSP]**, **[TKS]**, 和 **[TPS]** 部分。*

*如果您将 **-s** 选项与 **pkispawn** 搭配使用，并且指定子系统名称，则仅读取该子系统的部分。*

*这些部分具有层次结构：在子系统部分中指定的 **name=value** 对将覆盖 **[Tomcat]** 部分中的对，后者又覆盖 **[DEFAULT]** 部分中的对。默认对可以通过交互式输入或指定 PKI 实例配置文件中的对覆盖。*

注意

*每当使用非交互式文件覆盖默认 **name=value** 对时，它们可以存储在任何位置并随时指定。这些文件在 **pkispawn** man page 中被称为 **myconfig.txt**，但它们通常被称为 **.ini** 文件，或者通常被称为 PKI 实例配置文件。*

*如需更多信息，请参阅 **pki_default.cfg** 手册页。*

Configuration Servlet 由存储在 **/usr/share/java/pki/pki-certsrv.jar** 中的 Java 字节码组成，作为 **com/netscape/certsrv/system/ConfigurationRequest.class**。Servlet 使用 **pkispawn** 从配置脚本处理作为 JSON 对象传递的数据，然后使用与 **com/netscape/certsrv/system/ConfigurationResponse.class** 在相同的文件中提供的 Java bytecode 来返回到 **pkispawn**。

*交互式安装示例仅涉及以 **root** 用户身份在命令行上运行 **pkispawn** 命令：*

```
# pkispawn
```

重要

目前，交互式安装仅存在于非常基本的部署中。例如，在使用克隆、Elliptic Curve Curve Cryptography (ECC)、外部 CA、硬件安全模块(HSM)、子 CA 和其他级 CA 等高级功能时，必须在单独的配置文件中提供必要的覆盖参数。

非交互式安装需要 PKI 实例配置覆盖文件，进程可能类似以下示例：

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

2. *使用文本编辑器（如 **vim** ）创建名为 **/root/pki/ca.cfg** 的配置文件，其内容如下：*

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

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

有关各种配置示例，请参阅 **pkispawn** 手册页。

2.2.2.4. 实例删除

要删除现有的 PKI 实例，请使用 **pkidestroy** 命令。它可以以交互方式运行，也可以作为批处理运行。使用 **pkidestroy -h** 在命令行中显示详细的使用信息。

pkidestroy 命令在 PKI 子系统部署配置文件中读取，该文件在创建子系统时存储 (**/var/lib/pki/instance_name/ <subsystem> /registry/<subsystem> /deployment.cfg**)，使用 read-in 文件来删除 PKI 子系统，然后在没有额外的子系统时删除 PKI 实例。如需更多信息，请参阅 **pkidestroy** 手册页。

使用 **pkidestroy** 的交互式删除过程可能类似如下：

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

Begin uninstallation (Yes/No/Quit)? Yes

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

Uninstallation complete.
```

非互动删除过程可能类似以下示例：

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

Uninstallation complete.
```

2.2.3. 执行管理(systemctl)

2.2.3.1. 启动、停止、重启和获取状态

Red Hat Certificate System 子系统实例可以使用 Red Hat Enterprise Linux 8 上的 **systemctl** 执行管理系统工具停止并启动：

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

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

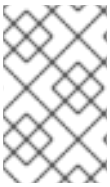
```
# systemctl stop <unit-file>@instance_name.service
```

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

<unit-file> 具有以下值之一：

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

有关 **watchdog** 服务的详情，请参考 Red Hat Certificate System Administration Guide 中的 第 2.3.10 节 "密码和 Watchdog (nuxwdog)" 和使用证书系统 Watchdog Service 部分。



注意

在 RHCS 10 中，这些 **systemctl** 操作支持 **pki-server** 别名： **pki-server <command> subsystem_instance_name** 是 **systemctl <command> pki-tomcatd@<instance>.service** 的别名。

2.2.3.2. 自动启动实例

Red Hat Enterprise Linux 中的 **systemctl** 工具管理服务器上每个进程的自动启动和关闭设置。这意味着，当系统重启时，一些服务可以被自动重启。系统单元文件控制服务启动，以确保服务以正确顺序启动。**systemd** 服务和 **systemctl** 工具信息包括在 为 Red Hat Enterprise Linux 8 配置基本系统设置 指南中。

证书系统实例可由 **systemctl** 管理，因此这个工具可以设置是否自动重启实例。创建证书系统实例后，它会在引导时启用。这可以通过使用 **systemctl** 来更改：

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

重新启用实例：

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

注意

systemctl enable 和 **systemctl disable** 命令不会立即启动或停止证书系统。

2.2.4. 进程管理(pki-server 和 pkidaemon)

2.2.4.1. pki-server 命令行工具

Red Hat Certificate System 的主要进程管理工具是 **pki-server**。使用 **pki-server --help** 命令，并查看 **pki-server** man page 以获得使用信息。

pki-server 命令行界面(CLI)管理本地服务器实例（如服务器配置或系统证书）。按如下方式调用 CLI：

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

CLI 使用服务器实例的配置文件和 NSS 数据库，因此 CLI 不需要任何之前的初始化。由于 CLI 直接访问文件，它只能由 root 用户执行，而且不需要客户端证书。此外，无论服务器的状态如何，CLI 都可以运行；它不需要正在运行的服务器。

CLI 支持以分级结构组织的多个命令。要列出顶级命令，请在没有任何附加命令或参数的情况下执行 CLI：

```
$ pki-server
```

某些命令包含子命令。若要列出它们，可使用命令名称执行 CLI，并且无附加选项。例如：

```
$ pki-server ca
$ pki-server ca-audit
```

要查看命令用法信息，请使用 **--help** 选项：

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

2.2.4.2. 使用 pki-server 启用和禁用已安装的子系统

要启用或禁用已安装的子系统，请使用 **pki-server** 工具。

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

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

使用有效的子系统标识符替换 subsystem_id ： **ca**、**kra**、**tks**、**ocsp** 或 **tps**。

注意

一个实例只能具有每种类型的子系统之一。

例如，要在名为 **pki-tomcat** 的实例中禁用 OCSP 子系统：

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

列出实例的已安装的子系统：

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

显示特定子系统的状态：

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

2.2.4.3. pkidaemon 命令行工具

Red Hat Certificate System 的另一个进程管理工具是 **pkidaemon** 工具：

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

- **pkidaemon status tomcat** - 提供状态信息，如 on/off、端口、系统上所有 PKI 子系统的每个 PKI 子系统的 URL。

- **pkidaemon status tomcat instance_name** - 提供状态信息，如 on/off、端口、特定实例的每个 PKI 子系统的 URL。
- **pkidaemon start tomcat instance_name.service** - 使用 **systemctl** 内部使用。

如需更多信息，请参阅 **pkidaemon** 手册页。

2.2.4.4. 查找子系统 Web 服务 URL

CA, KRA, OCSP, OCSP, TKS, 和 TPS 子系统有用于代理的 Web 服务页面，以及常规用户和管理员（如果适用）。可以通过子系统的安全最终用户端口打开子系统主机的 URL 来访问这些 Web 服务。例如，对于 CA：

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

注意

要获取实例的所有接口、URL 和端口的完整列表，请检查该服务的状态。例如：

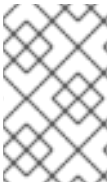
```
pkidaemon status instance_name
```

每个子系统的主要 Web 服务页面都有可用服务页面列表；这些内容在 表 2.1 “默认 Web 服务页面” 中进行了概述。要特别访问任何服务，请访问适当的端口，并将相应的目录附加到 URL。例如，要访问 CA 的末尾实体（常规用户）Web 服务：

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

如果没有配置 DNS，则可以使用 IPv4 或 IPv6 地址来连接到服务页面。例如：

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



注意

任何人都可以访问子系统的最终用户页面。但是，访问代理或管理 Web 服务页面要求在 Web 浏览器中发布代理或管理员证书并安装代理或管理员证书。否则，对 Web 服务的身份验证会失败。

表 2.1. 默认 Web 服务页面

port	用于 SSL/TLS	用于客户端身份验证[a]	Web 服务	Web 服务位置
证书管理器				
8080	否		结束实体	ca/ee/ca
8443	是	否	结束实体	ca/ee/ca
8443	是	是	代理	ca/agent/ca

port	用于 SSL/TLS	用于客户端身份验证[a]	Web 服务	Web 服务位置
8443	是	否	服务	ca/services
8443	是	否	控制台（Console）	pkiconsole https://*host:port*/ca
密钥恢复授权机构				
8080	否		结束实体[b]	kra/ee/kra
8443	是	否	结束实体[b]	kra/ee/kra
8443	是	是	代理	kra/agent/kra
8443	是	否	服务	kra/services
8443	是	否	控制台（Console）	pkiconsole https://*host:port*/kra
在线证书状态管理器				
8080	否		结束实体[c]	ocsp/ee/ocsp
8443	是	否	结束实体[c]	ocsp/ee/ocsp
8443	是	是	代理	ocsp/agent/ocsp
8443	是	否	服务	ocsp/services
8443	是	否	控制台（Console）	pkiconsole https://*host:port*/ocsp
令牌密钥服务				
8080	否		结束实体[b]	tks/ee/tks
8443	是	否	结束实体[b]	tks/ee/tks
8443	是	是	代理	tks/agent/tks
8443	是	否	服务	tks/services

port	用于 SSL/TLS	用于客户端身份验证[a]	Web 服务	Web 服务位置
8443	是	否	控制台（Console）	pkiconsole https://*host:port*/tks
令牌处理系统				
8080	否		未安全的服务	tps/tps
8443	是		安全服务	tps/tps
8080	否		企业安全客户端电话主页	tps/phoneHome
8443	是		企业安全客户端电话主页	tps/phoneHome
8443	是	是	管理、代理和 Operator 服务 [d]	tps/ui

[a] 客户端身份验证值为 **No** 的服务可以重新配置为需要客户端身份验证。没有 **Yes** 或 **No** 值的服务不能被配置为使用客户端身份验证。

[b] 虽然此子系统类型具有端实体端口和接口，但这些最终用户服务无法通过 Web 浏览器访问，因为其他最终用户服务是：

[c] 虽然 OCSP 有端实体端口和接口，但这些最终用户服务无法通过 Web 浏览器访问，因为其他最终用户服务是：最终用户 OCSP 服务可通过发送 OCSP 请求来访问。

[d] 代理、管理员和操作器服务都通过同一 Web 服务页面访问。每个角色只能访问仅对该角色成员可见的特定部分。

*2.2.4.5. 启动证书**系统控制台***

重要

此控制台已弃用。

CA、KRA、OCSP 和 TKS 子系统有一个 Java 接口，它可以被访问来执行管理功能。对于 KRA、OCSP 和 TKS，这包括配置日志记录和管理用户和组等非常基本的任务。对于 CA，这包括创建证书配置集和配置发布等其他配置设置。

*通过使用 **pkiconsole** 工具通过 SSL/TLS 端口连接到子系统实例来打开控制台。这个工具使用以下格式：*

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

*subsystem_type 可以是 **ca**、**kra**、**ocsp** 或 **tks**。例如，这将打开 KRA 控制台：*

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

如果没有配置 DNS，则可以使用 IPv4 或 IPv6 地址来连接到控制台。例如：

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

2.3. 证书系统架构概述

虽然每个服务都提供不同的服务，但所有 RHCS 子系统(CA、KRA、OCSP、TKS、TPS)共享一个通用架构。以下架构图显示了所有这些子系统共享的通用架构。

2.3.1. Java Application Server

Java 应用服务器是运行服务器应用程序的 Java 框架。证书系统设计为在 Java 应用服务器内运行。目前，Tomcat 8 唯一支持的 Java 应用程序服务器。以后可能会添加对其他应用服务器的支持。如需更多信息，请参阅 http://tomcat.apache.org/。

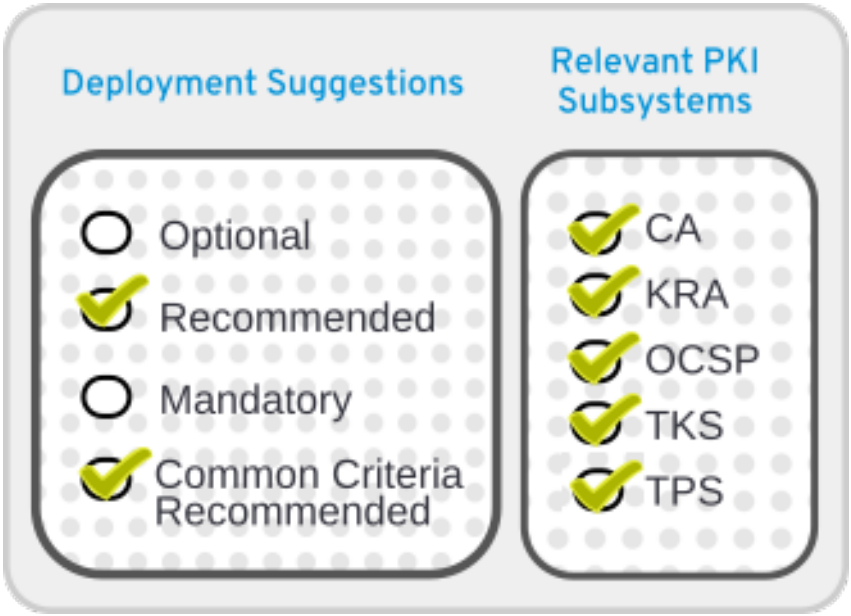
每个证书系统实例都是 Tomcat 服务器实例。Tomcat 配置存储在 **server.xml** 中。以下链接提供有关 Tomcat 配置的更多信息：https://tomcat.apache.org/tomcat-8.0-doc/config/

每个证书系统子系统（如 CA 或 KRA）都部署为 Tomcat 中的 Web 应用程序。Web 应用程序配置存储在 **web.xml** 文件中，该文件在 Java Servlet 3.1 规范中定义。详情请查看 https://www.jcp.org/en/jsr/detail?id=340。

证书系统配置本身存储在 **CS.cfg** 中。

有关这些文件的实际位置，请参阅 第 2.3.15 节 “实例布局”。

2.3.2. Java 安全管理器



Java 服务可以选择使用安全管理器，为应用程序定义不安全和安全操作。安装子系统后，它们会自动启用 Security Manager，这意味着每个 Tomcat 实例都会从运行 Security Manager 开始。

如果实例是通过运行 pkispawn 创建的，并使用覆盖配置文件（在其自己的 Tomcat 部分下指定 **pki_security_manager=false** 选项）创建，则安全管理器会被禁用。

可以按照以下流程从已安装的实例禁用安全管理器：

-

1. ```
   # pki-server stop instance_name
   ```

   或（如果使用 **nuxwdog** watchdog）

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

2. 打开 **/etc/sysconfig/instance_name** 文件，并设置 **SECURITY_MANAGER="false"**

3. ```
   # pki-server start instance_name
   ```

   或（如果使用 **nuxwdog** watchdog）

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

当实例启动或重启时，会在以下文件中由 **pkidaemon** 构建或重新构建 Java 安全策略：

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

然后，它被保存到 **/var/lib/pki/instance_name/conf/catalina.policy** 中。

2.3.3. 接口

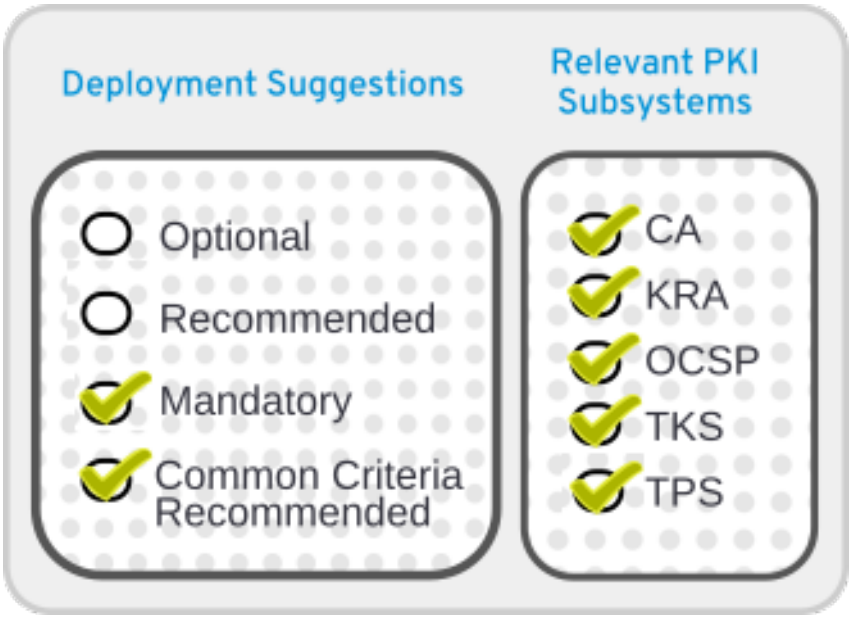
2.3.3.1. Servlet 接口

每个子系统包含允许与子系统的不同部分交互的接口。所有子系统共享一个通用管理界面，并有一个代理接口，允许代理执行分配给它们的任务。CA 子系统有一个端点接口，它允许终端实体注册到 PKI。OCSP Responder 子系统有一个最终用户接口，允许终端化和应用程序检查当前证书撤销状态。最后，TPS 有一个 operator 接口。

虽然应用服务器提供连接入口点，但证书系统通过提供特定于每个接口的 servlet 来完成接口。

每个子系统的 servlet 在对应的 **web.xml** 文件中定义。同一文件还定义每个 servlet 的 URL 和访问 servlet 的安全要求。请参阅 第 2.3.1 节 “Java Application Server” 了解更多信息。

2.3.3.2. 管理界面

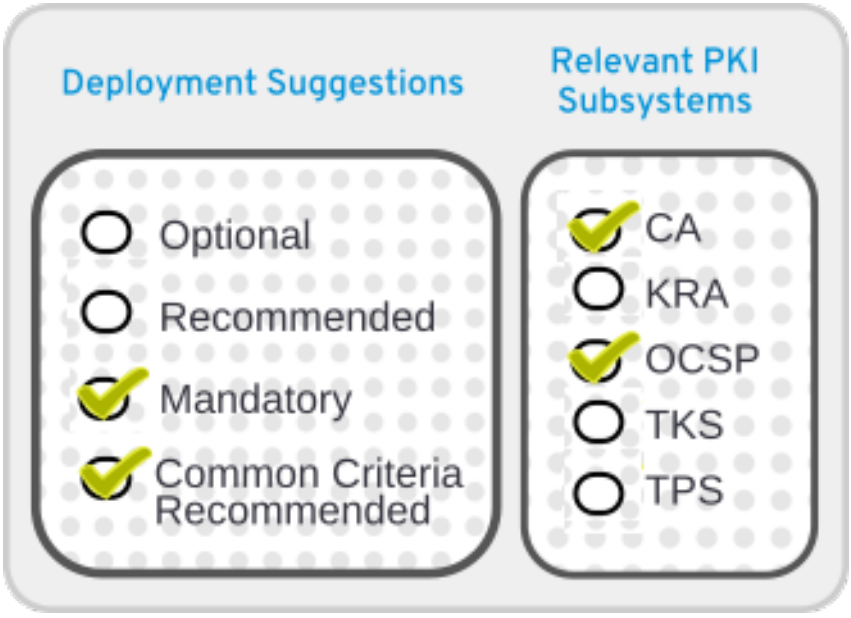


代理接口提供 Java servlet 来处理来自代理入口点的 HTML 表单提交。根据每个表单提交中提供的信息，代理 servlet 允许代理执行代理任务，如编辑和批准证书批准请求、证书续订和证书撤销、批准证书配置文件。KRA 子系统或 TKS 子系统的代理接口，或 OCSP Responder 特定于子系统。

在非TMS 设置中，代理接口也用于 CA-to-KRA 可信连接的 CIMC 边界通信。此连接受 SSL 客户端身份验证保护，并由单独的可信角色区分，称为 受信任的管理器。与 agent 角色一样，受信任的管理器（仅为 CIMC 边界连接创建的伪用户）需要 SSL 客户端验证。但是，与代理角色不同，它们不提供任何代理功能。

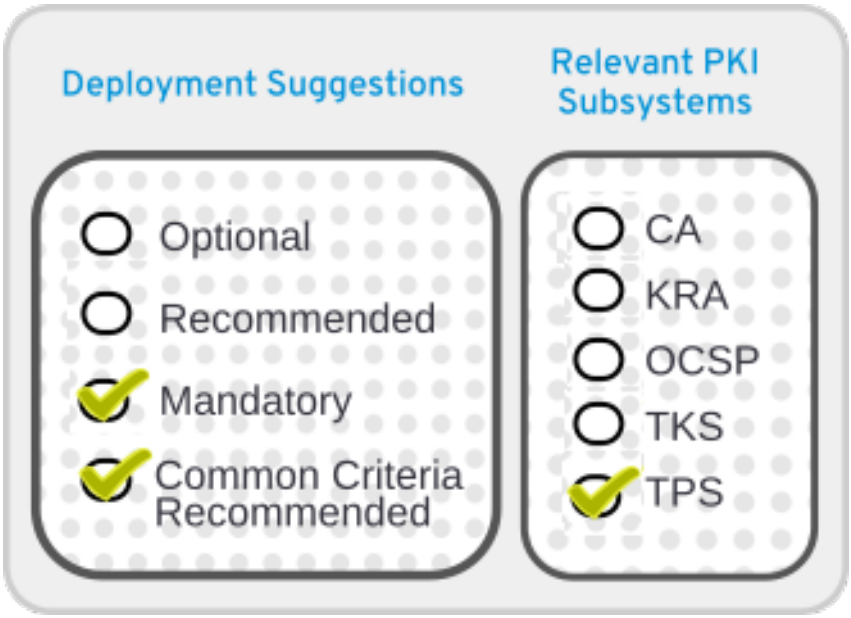
在 TMS 设置中，inter-CIMC 边界通信来自 TPS-to-CA、TPS-to-KRA 和 TPS-to-TKS。

2.3.3.3. 端到端接口



对于 CA 子系统，最终用户接口提供 Java servlet 来处理来自最终用户入口点的 HTML 表单提交。根据表格提交中收到的信息，最终用户 servlet 允许终端实体注册、续订证书、撤销自己的证书，以及获取签发的证书。OCSP Responder 子系统的 End-Entity 接口提供 Java servlet 以接受和处理 OCSP 请求。KRA、TKS 和 TPS 子系统不提供任何最终用户。

2.3.3.4. Operator Interface



操作器接口仅在 TPS 子系统中找到。

2.3.4. REST 接口

Representational state transfer (REST)是一种使用 HTTP 定义和组织 Web 服务的方法，可简化与其他应用程序的互操作性。Red Hat Certificate System 提供了一个 REST 接口来访问服务器上的各种服务。

红帽证书系统中的 REST 服务使用 RESTEasy 框架来实施。RESTEasy 实际上作为 servlet 在 web 应用中运行，因此 RESTEasy 配置也可以在对应的子系统的 ***web.xml*** *中找到。有关 RESTEasy 的更多信息，请访问 http://resteasy.jboss.org/。*

每个 REST 服务都定义为一个单独的 URL。例如：

- *CA 证书服务：* ***http:// <host_name>: <port> /ca/rest/certs/***
- *KRA 密钥服务：* ***http:// <host_name>:<port> /kra/rest/agent/keys/***
- *TKS 用户服务：* ***http:// <host_name>:<port> /tks/rest/admin/users/***
- *TPS 组服务：* ***http:// <host_name>:<port> /tps/rest/admin/groups/***

有些服务可以使用普通 HTTP 连接访问，但有些服务可能需要 HTTPS 连接才能实现安全性。

REST 操作指定为 HTTP 方法（如 GET、PUT、POST、DELETE）。例如，若要获取 CA 用户，客户端将发送 GET /ca/rest/users 请求。

REST 请求和响应消息可以以 XML 或 JSON 格式发送。例如：

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

可以使用 CLI、Web UI 或通用 REST 客户端等工具访问 REST 接口。证书系统还提供 Java、Python 和 JavaScript 库，以编程方式访问服务。

REST 接口支持两种类型的身份验证方法：

- 用户名和密码
- 客户端证书

每个服务所需的身份验证方法在 **/usr/share/pki/ca/conf/auth-method.properties** 中定义。

REST 接口可能需要某些权限才能访问该服务。权限在 LDAP 中的 ACL 资源中定义。REST 接口映射到 **/usr/share/pki/ <subsystem> /conf/acl.properties** 中的 ACL 资源。

有关 REST 接口的更多信息，请参阅 http://www.dogtagpki.org/wiki/REST。

2.3.5. JSS

Java 安全服务 (JSS) 为 NSS 执行的加密操作提供了一个 Java 接口。JSS 及更高级别的证书系统架构使用 Java 原生接口 (JNI) 构建，提供对 Java 虚拟机 (JVM) 内原生系统库的访问。这个设计允许我们使用 FIPS 批准的加密供应商，比如作为系统一部分分发的 NSS。JSS 支持 NSS 支持的大多数安全标准和加密技术。JSS 还为 ASN.1 类型和 BER-DER 编码提供了一个纯 Java 接口。

2.3.6. tomcatjss

Red Hat Certificate System 中的基于 Java 的子系统使用一个名为 **tomcatjss** 的 JAR 文件作为 Tomcat 服务器 HTTP 引擎和 JSS 之间的桥接，Java 接口用于 NSS 执行的安全操作。**tomcatjss** 是一个 Java 安全套接字扩展 (JSSE) 实现，使用 Tomcat 的 Java 安全服务 (JSS) 实现。

tomcatjs 实现了使用 TLS 并创建 TLS 套接字所需的接口。tomcatjss 实施的套接字工厂利用下面列出的各种属性来创建 TLS 服务器侦听套接字并将其返回到 tomcat。tomcatjs 本身使用 java JSS 系统最终与机器上的原生 NSS 加密服务通信。

加载 Tomcat 服务器和证书系统类时，会加载 tomcatjs。载入过程如下所述：

1. 服务器已启动。
2. Tomcat 指向为证书系统安装创建侦听套接字需要的位置。
3. **server.xml** 文件已被处理。此文件中的配置告知系统使用 Tomcatjs 实施的套接字工厂。
4. 对于每个请求的套接字，Tomcajss 会在创建套接字时读取和处理包含的属性。生成的套接字的行为是因为已被这些参数要求。
5. 服务器运行后，我们所需的一组侦听套接字等待进入基于 Tomcat 的证书系统的连接。

请注意，在启动时创建套接字时，Tomcatjs 是证书系统 中的第一个 实体，实际上处理底层 JSS 安全服务。处理第一个侦听套接字后，会创建一个 JSS 实例以供继续使用。

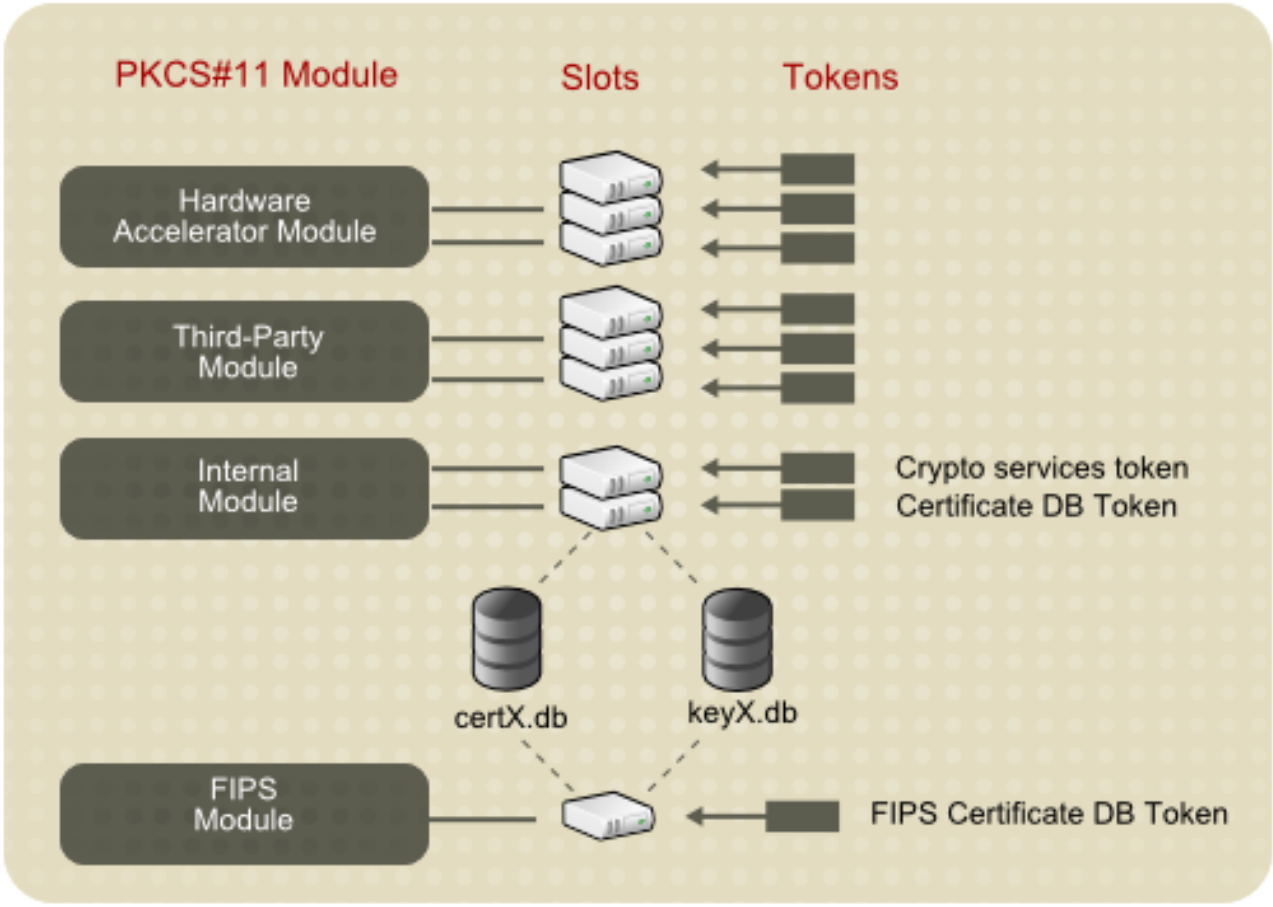
有关 **server.xml** 文件的详情，请参考 第 14.4 节 “Tomcat Engine 和 Web 服务的配置文件”。

2.3.7. PKCS #11

公钥加密标准 (PKCS) 是指定用来与保存加密信息的设备通信的 API，并执行加密操作。由于它支持 PKCS rebased，证书系统与广泛的硬件和软件设备兼容。

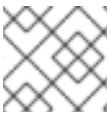
任何证书系统子系统实例必须至少有一个 PKCSBACKEND 模块可用。PKCSBACKEND 模块（也称为加密模块或加密服务提供商）管理加密服务，如加密和解密。PKCS current 模块与可以在硬件或软件中实施的加密设备的驱动程序类似。证书系统包含一个内置的 PKCS facilities 模块，可以支持第三方模块。

PKCSBACKEND 模块始终具有一个或多个插槽，可作为物理读取器中的物理硬件插槽（如智能卡或软件中的概念插槽）实现。PKCSjpeg 模块的每个插槽可以包含一个令牌，它是一个硬件或软件设备，它实际提供加密服务，并选择性地存储证书和密钥。



证书系统定义了两种类型的令牌，即 内部和外部 。内部令牌用于存储证书信任锚。外部令牌用于存储属于证书系统子系统的密钥对和证书。

2.3.7.1. NSS 软令牌（内部令牌）



注意

证书系统使用 NSS 软令牌来存储证书信任锚。

NSS 软令牌也称为内部令牌或软件令牌。软件令牌由两个文件组成，这些文件通常称为证书数据库 (cert9.db)和密钥数据库(key4.db)。文件是在证书系统子系统配置期间创建的。这些安全数据库位于 **/var/lib/pki/instance_name/alias** 目录中。

由 NSS 软令牌提供的两个加密模块包含在证书系统中：

- 默认内部 PKCSROX 模块，它附带两个令牌：
  - 内部加密服务令牌，执行所有加密操作，如加密、解密和哈希。
  - 内部密钥存储令牌("Certificate DB 令牌")，它处理与存储证书和密钥的证书和密钥数据库文件的所有通信。
- FIPS 140 模块。这个模块符合对加密模块实现的 FIPS 140 政府标准。FIPS 140 模块包含一个内置 FIPS 140 证书数据库令牌，它处理加密操作以及与证书和密钥数据库文件的通信。

有关如何将证书导入到 NSS 软令牌的具体信息包括在 第 15 章 管理证书/密钥加密策略 中。

有关网络安全服务(NSS)的更多信息，请参阅相同名称的 Mozilla Developer web 页面。

2.3.7.2. 硬件安全模块(HSM、外部令牌)

注意

证书系统使用 HSM 存储属于证书系统子系统的密钥对和证书。

任何 PKCSGIO 模块都可以与证书系统一起使用。要将外部硬件令牌与子系统搭配使用，请在配置子系统前加载其 PKCSBACKEND 模块，并且新令牌可供子系统使用。

可用的 PKCS Anaconda 模块在子系统的 **pkcs11.txt** 数据库中跟踪。**当系统** 有变化时，modutil 实用程序用于修改此文件，如安装用于签名操作的硬件加速器。有关 **modutil** 的更多信息，请参阅 Mozilla Developer 网页上的网络安全服务(NSS)。

PKCSBACKEND 硬件设备还为存储在硬件令牌中的信息提供密钥备份和恢复功能。有关从令牌检索密钥的详情，请参考 PKCSROX 供应商文档。

有关如何导入证书和管理 HSM 的具体说明位于 第 15 章 管理证书/密钥加密策略 中。

支持的硬件安全模块位于 第 4.4 节 "支持的硬件安全模块" 中。

2.3.8. 证书系统序列号管理

2.3.8.1. 序列号范围

证书请求和序列号由 Java 的大整数表示

默认情况下，由于其效率，CA 子系统按顺序分配证书请求号、证书序列号和副本 ID。

序列号范围对于请求、证书和副本 ID 可指定：

- 当前序列号管理基于分配顺序序列号的范围。
- 当超过定义的阈值时，实例会请求新的范围。
- 实例在分配给实例后存储有关新获取范围的信息。
- 实例继续使用旧的范围，直到所有数字都用尽，然后移动到新范围。
- 克隆的子系统通过复制冲突来同步其范围分配。

对于新克隆：

- 在克隆过程中，当前 master 范围的一部分被传送到一个新的克隆。
- 如果传输的范围低于定义的阈值，新的克隆可能会请求新的范围。

所有范围均可在 CA 实例安装时配置，方法是向 PKI 实例覆盖配置文件中添加 **[CA]** 部分，并根据需要在该部分下添加以下 **name=value** 对。以下示例中显示了 **/etc/pki/default.cfg** 中已存在的默认值：

```
[CA]
pki_serial_number_range_start=1
```

```
pki_serial_number_range_end=10000000
pki_request_number_range_start=1
pki_request_number_range_end=10000000
pki_replica_number_range_start=1
pki_replica_number_range_end=100
```

2.3.8.2. 随机序列号管理

除了顺序序列号管理外，红帽认证系统还提供可选的随机序列号管理。通过在 PKI 实例覆盖文件中添加 **[CA]** 部分并在该部分下添加以下 **name=value** 对，可以在 CA 实例安装时选择随机序列号：

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

如果选择，证书请求号和证书序列号将在指定的范围内随机选择。

2.3.9. 安全域

安全域 是 PKI 服务的注册表。CA 等服务注册这些域中自身的信息，因此 PKI 服务的用户可以通过检查注册表来查找其他服务。RHCS 中的安全域服务管理为证书系统子系统和一组共享信任策略的 PKI 服务注册。

详情请查看 第 5.3 节 "规划安全域"。

2.3.10. 密码和 Watchdog (nuxwdog)

在默认设置中，RHCS 子系统实例需要充当客户端并向某些其他服务进行身份验证，如 LDAP 内部数据库（除非设置了 TLS 客户端身份验证，其中将用来进行身份验证的证书）、NSS 令牌数据库或有时带有密码的 HSM。在安装配置时，管理员会提示您设置此密码。然后，此密码将写入文件 **<instance_dir>/conf/password.conf**。同时，标识字符串存储在主配置文件中 **CS.cfg** 中，作为参数 **cms.passwordlist** 的一部分。

配置文件 **CS.cfg** 受 Red Hat Enterprise Linux 的保护，且只能被 PKI 管理员访问。没有密码存储在 **CS.cfg** 中。

在安装过程中，安装程序会选择并登录到内部软件令牌或硬件加密令牌。到这些令牌的登录密码短语也写入 **password.conf**。

稍后配置也可以将密码放在 **password.conf** 中。LDAP 发布是一个示例，新为发布目录配置的 Directory Manager 密码被输入 **password.conf**。

nuxwdog (watchdog)是一种轻量级辅助守护进程进程，用于启动、停止、监控其状态并重新配置服务器程序。当用户需要提示您输入密码以启动服务器时，它最有用，因为它会在内核密钥环中安全地缓存这些密码，以便在服务器崩溃时自动重启。

注意

nuxwdog 是唯一 允许的 watchdog 服务。

安装完成后，可以完全删除 **password.conf** 文件。重启时，**nuxwdog** watchdog 程序将提示管理员输入所需的密码，使用参数 **cms.passwordlist**（如果使用 **cms.tokenList**），作为要提示的密码列表。然后，在内核密钥环中由 **nuxwdog** 缓存密码，以允许从服务器崩溃自动恢复。当出现不受控制的关闭 (crash)时，会发生此自动恢复（自动子系统重启）。如果管理员控制的关闭，管理员将再次提示输入密码。

使用 watchdog 服务时，启动和停止 RHCS 实例会有所不同。详情请查看 第 14.3.2 节 "使用证书系统 Watchdog 服务"。

有关详情请参考 第 14.3 节 "管理系统密码"。

2.3.11. 内部 LDAP 数据库

Red Hat Certificate System 使用红帽目录服务器(RHDS)作为其内部数据库，用于存储证书、请求、用户、角色、ACL 等信息，以及其他各种内部信息。证书系统可以通过密码与内部 LDAP 数据库通信，或者通过 SSL 身份验证安全地通信。

如果在证书系统实例和目录服务器之间需要基于证书的身份验证，务必要按照说明在这两个实体之间设置信任。安装此类证书系统实例也需要正确 **pkispawn** 选项。

详情请查看 第 6.5 节 "安装 Red Hat Directory Server"。

2.3.12. Security-Enhanced Linux (SELinux)

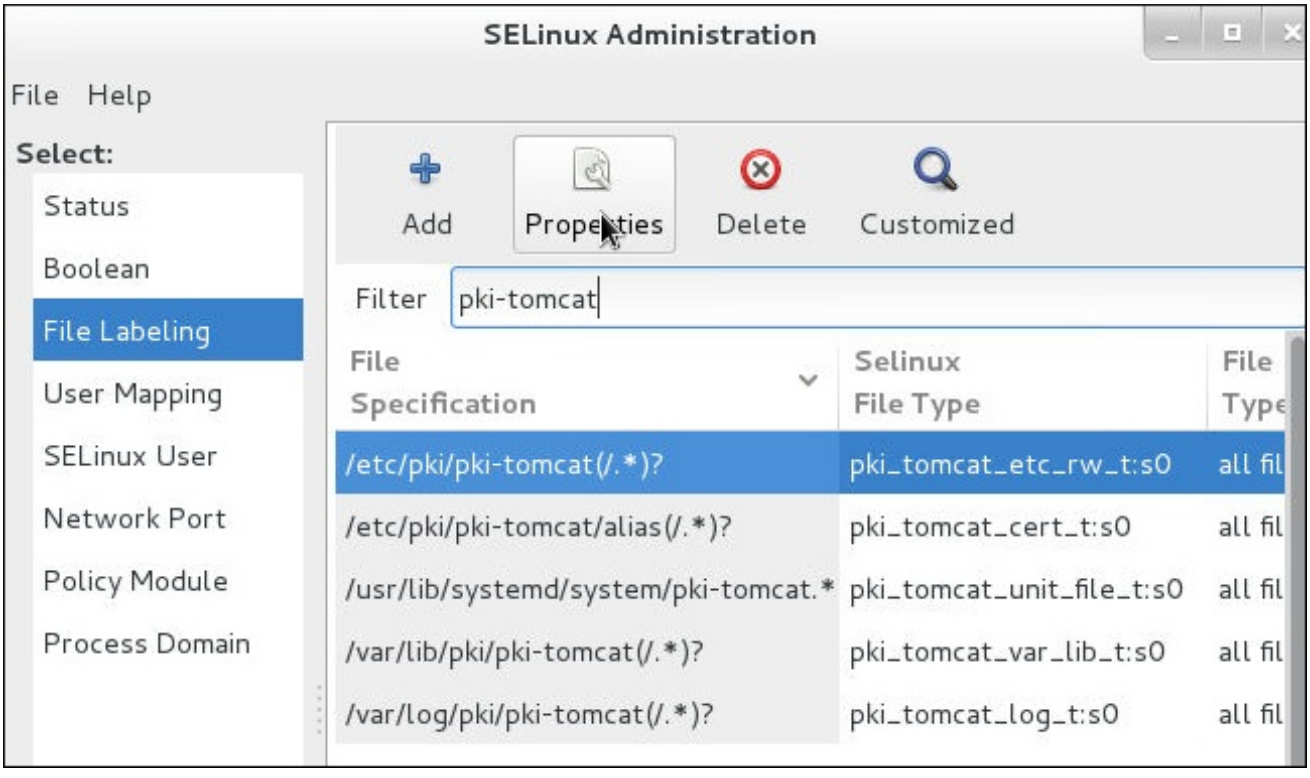
SELinux 是强制访问控制规则的集合，用于限制未经授权的访问和篡改。在 Red Hat Enterprise Linux 8 中使用 SELinux 指南中 更详细地描述了 SELinux。

基本上，SELinux 会识别系统上的 对象，可以是文件、目录、用户、进程、套接字或 Linux 主机上的任何其他资源。这些对象与 Linux API 对象对应。然后，每个对象 都会映射到一个安全上下文，用于定义对象类型以及如何在 Linux 服务器上正常工作。

对象可以被分组到域中，然后为每个域分配正确的规则。每个安全上下文都有规则设置对其可以执行什么操作的限制、它可以访问哪些资源，以及它们具有什么权限。

证书系统的 SELinux 策略被合并到标准系统 SELinux 策略中。这些 SELinux 策略适用于证书系统使用的每个子系统和服务。通过以 enforcing 模式运行 SELinux 的证书系统，增强了由证书系统创建和管理的信息的安全性。

图 2.1. CA SELinux 端口策略



证书系统 SELinux 策略为每个子系统实例定义 SELinux 配置：

- 每个子系统实例的文件和目录使用特定的 SELinux 上下文标记。
- 每个子系统实例的端口使用特定的 SELinux 上下文标记。
- 所有证书系统进程都在特定于子系统的域中进行限制。
- 每个域具有特定的规则，用于定义授权域的操作。
- SELinux 策略中没有指定的访问权限都会被拒绝访问证书系统实例。

对于证书系统，每个子系统被视为 SELinux 对象，每个子系统分配有唯一的规则。定义的 SELinux 策略允许证书系统对象在 enforcing 模式下设置 SELinux 运行。

每次运行 **pkispawn** 时，用来配置证书系统子系统，与该子系统关联的文件和端口都会使用所需的 SELinux 上下文标记。使用 **pkidestroy** 删除特定子系统时，这些上下文将被删除。

SELinux 策略中的中央定义是 **pki_tomcat_t** 域。证书系统实例是 Tomcat 服务器，**pki_tomcat_t** 域扩展了标准 **tomcat_t** Tomcat 域的策略。服务器上的所有证书系统实例共享相同的域。

当每个证书系统进程启动时，它最初在无限制域(**unconfined_t**)中运行，然后过渡到 **pki_tomcat_t** 域。然后，这个进程具有某些访问权限，如对标记为 **pki_tomcat_log_t** 的日志文件进行写入访问，对标记为 **pki_tomcat_etc_rw_t** 的配置文件进行读写访问，或者对 **http_port_t** 端口打开和写入。

SELinux 模式可以从 enforcing 模式改为 permissive，即使不建议这样做。

2.3.13. self-tests

Red Hat Certificate System 提供了一个自测试框架，它允许在启动或按需进行期间检查 PKI 系统完整性。如果出现非关键自我测试失败，则消息将存储在日志文件中，而如果出现关键自我测试失败，则消息将存储在日志文件中，而证书系统子系统将正确关闭。如果管理员希望在启动时看到自测试报告，则管理员预期会在子系统启动期间观察自测试日志。它们也可以在启动时查看日志。

当因为自测试失败而关闭子系统时，也会自动禁用它。这是为了确保子系统不会部分运行，并生成误导的响应。解决此问题后，可以通过在服务器上运行以下命令来重新启用子系统：

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

有关如何配置自助测试的详情，请参考 第 18.3.2 节 “配置自测试”。

2.3.14. 日志

证书系统子系统创建记录与活动相关的日志文件，如管理、使用服务器支持的协议的通信，以及由子系统使用的各种其他进程。在子系统实例运行时，它会保留其管理的所有组件的信息和错误消息的日志。此外，Apache 和 Tomcat Web 服务器还会生成错误和访问日志。

每个子系统实例维护自己的日志文件，用于安装、审计和其他日志记录功能。

日志插件模块是作为 Java™ 类实施的监听程序，并在配置框架中注册。

当实例使用 **pkispawn** 创建时，所有日志文件和轮转的日志文件（除审计日志除外）都位于 **pki_subsystem_log_path** 中指定的任何目录中。 常规审计日志位于日志目录中，使用其他类型的日志进行日志，而签名的审计日志则写入 **/var/log/pki/instance_name/subsystem_name/signedAudit**。可以通过修改配置来更改日志的默认位置。

有关在安装过程中日志配置的详情，请参考 第 18 章 配置日志。

有关安装后日志管理的详情，请参考 Red Hat Certificate System Administration Guide 中的 Configuring Subsystem Logs 部分。

2.3.14.1. 审计日志

审计日志包含为可记录事件设置的可选择事件的记录。您还可以将审计日志配置为签名以进行完整性检查。

注意

审计记录应根据 - 第 18.4 节 "审计保留" 中指定的审计保留规则保存。

2.3.14.2. 调试日志

调试日志（默认启用）会为所有子系统维护，不同的程度和类型的信息。

每个子系统记录详细信息的调试日志，包含子系统执行的每个操作的非常具体的信息，包括运行插件和 servlets 的插件和 servlet，以及服务器请求和响应消息。

记录到调试日志的服务包括授权请求、处理证书请求、证书状态检查以及归档和恢复密钥以及访问 Web 服务。

debug 记录有关子系统进程的详细信息。每个日志条目都采用以下格式：

```
[date:time] [processor]: servlet: message
```

消息 可以是来自子系统的返回消息，也可以包含提交到子系统的值。

例如，TKS 记录此消息以连接到 LDAP 服务器：

```
[10/Jun/2022:05:14:51][main]: Established LDAP connection using basic authentication to host
localhost port 389 as cn=Directory Manager
```

处理器 是主 的，消息 是服务器有关 LDAP 连接的消息，没有 servlet。

另一方面，CA 记录有关证书操作的信息以及子系统连接：

```
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestowner$
value=KRA-server.example.com-8443
```

在这种情况下，处理器是通过 CA 的代理端口的 HTTP 协议，而它指定了处理配置文件的 servlet，并包含提供 profile 参数（请求的子系统所有者）及其值(KRA 启动请求) 的消息。

例 2.1. CA 证书请求日志消息

```
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.profileapprovedby$ value=admin
    [06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet:
key=$request.cert_request$
value=MIIBozCCAZ8wggEFAgQqTfoHMIHHgAECpQ4wDDEKMAgGA1UEAxMBeKaBnzANBgkqhki
G9w0BAQEFAAOB...
    [06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profile$
```

value=true
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.cert_request_type$ value=crmf
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestversion$ value=1.0.0
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_locale$ value=en
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestowner$ value=KRA-server.example.com-8443
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.dbstatus$ value=NOT_UPDATED
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.subject$ value=uid=jsmith, e=jsmith@example.com
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requeststatus$ value=begin
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.user$ value=uid=KRA-server.example.com-8443,ou=People,dc=example,dc=com
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_key$ value=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDreuEsBWq9WuZ2MaBwtNYxvkLP^M
HcN0cusY7gxLzB+XwQ/VsWEoObGldg6WwJPOcBdvLiKKfC605wFdynbEgKs0fChV^M
k9HYDhmJ8hX6+PaquiHJSVNhsv5tOshZkCfMBbyxwrKd8yZ5G5I+2gE9PUznxJaM^M
HTmlOqm4HwFxzy0RRQIDAQAB
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.authmgrinstname$ value=raCertAuth
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.uid$ value=KRA-server.example.com-8443
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.userid$ value=KRA-server.example.com-8443
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestor_name$ value=
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.profileid$ value=caUserCert
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.userdn$ value=uid=KRA-server.example.com-4747,ou=People,dc=example,dc=com
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.requestid$ value=20
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.auth_token.authtime$ value=1212782378071
[06/Jun/2022:14:59:38][http-8443;-Processor24]: ProfileSubmitServlet: key=$request.req_x509info$ value=MIICIKADAgECAgEAMA0GCSqGSIb3DQEBBQUAMEAxHjAcBgNVBAoTFVJlZGJ1ZGNv^M

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

```
HQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMEMCQGA1UdEQQdMBuBGSRyZXF1^
M
    ZXN0LnJlcXVlc3Rvcl9lbWFpbCQ=
```

同样，OCSP 显示 OCSP 请求信息：

```
[07/Jul/2022:06:25:40][http-11180-Processor25]: OCSPServlet: OCSP Request:
    [07/Jul/2022:06:25:40][http-11180-Processor25]: OCSPServlet:
    MEUwQwIBADA+MDwwOjAJBgUrDgMCGgUABBSEWjCarLE6/BiSiENSsV9kHjqB3QQU
```

2.3.14.3. 安装日志

所有子系统都保留安装日志。

每次通过初始安装创建子系统时，或使用 **pkispawn** 创建额外实例，安装文件包含安装的完整调试输出，包括任何错误，如果安装成功，则为实例配置界面的 URL 和 PIN。该文件在实例的 **/var/log/pki/** 目录中创建一个，格式为 **pki-subsystem_name-spawn.timestamp.log**。

安装日志中的每一行都遵循安装过程中的一个步骤。

例 2.2. CA 安装日志

```
==========================================================================
                      INSTALLATION SUMMARY

==========================================================================

    Administrator's username:             caadmin
    Administrator's PKCS #12 file:
        /root/.dogtag/pki-tomcat/ca_admin_cert.p12

    Administrator's certificate nickname:
        caadmin
    Administrator's certificate database:
        /root/.dogtag/pki-tomcat/ca/alias

    To check the status of the subsystem:
        systemctl status pki-tomcatd@pki-tomcat.service

    To restart the subsystem:
        systemctl restart pki-tomcatd@pki-tomcat.service

    The URL for the subsystem is:
        https://localhost.localdomain:8443/ca

    PKI instances will be enabled upon system boot


==========================================================================
```

2.3.14.4. Tomcat 错误和访问日志

CA、KRA、OCSP、CKS 和 TPS 子系统将 Tomcat Web 服务器实例用于其代理和终端实体的接口。

Tomcat web 服务器创建错误和访问日志，该服务器安装有证书系统并提供 HTTP 服务。错误日志包含服务器遇到的 HTTP 错误消息。访问日志通过 HTTP 接口列出访问活动。

Tomcat 创建的日志：

- admin.timestamp
- catalina.timestamp
- catalina.out
- host-manager.timestamp
- localhost.timestamp
- localhost_access_log.timestamp
- manager.timestamp

这些日志在证书系统中不可用或可配置；它们只能在 Apache 或 Tomcat 中进行配置。有关配置这些日志的详情，请查看 Apache 文档。

2.3.14.5. 自助测试日志

当服务器启动或者自助测试被手动运行时，self-tests 记录在 self-tests 期间获得的信息。可以通过打开此日志来查看测试。此日志无法通过控制台配置。这个日志只能通过更改 **CS.cfg** 文件中的设置来配置。本节中有关日志的信息与此日志无关。有关自测试的更多信息，请参阅 第 2.6.5 节 "自助测试"。

2.3.14.6. journalctl Logs

启动证书系统实例时，在设置并启用 logging 子系统之前会有一个短暂的时间。在此期间，日志内容被写入标准输出，由 **systemd** 捕获并通过 **journalctl** 实用程序公开。

要查看这些日志，请运行以下命令：

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

如果使用 **nuxwdog** 服务：

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

通常，在实例启动时监控这些日志会很有用（例如，在启动时出现自测试失败）。要做到这一点，请在启动实例前在单独的控制台中运行这些命令：

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

如果使用 **nuxwdog** 服务：

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

2.3.15. 实例布局

每个证书系统实例都取决于多个文件。其中一些位于特定于实例的文件夹中，另一些则位于共同的文件夹中，与其他服务器实例共享。

例如，服务器配置文件存储在 **/etc/pki/instance_name/server.xml** 中，后者是特定于实例的，但 CA servlet 在 **/usr/share/pki/ca/webapps/ca/WEB-INF/web.xml** 中定义，后者由系统上的所有服务器实例共享。

2.3.15.1. 证书系统的文件和目录位置

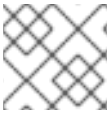
证书系统服务器是由一个或多个证书系统子系统组成的 Tomcat 实例。证书系统子系统是提供特定类型的 PKI 功能的 Web 应用。一般的共享子系统信息包含在不可重新分配的、RPM 定义的共享库、Java 归档文件、二进制文件和模板中。它们存储在固定的位置。

这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。

目录包含子系统的自定义配置文件和模板、配置文件、证书数据库和其他文件。

表 2.2. Tomcat 实例信息

设置	值
主目录	/var/lib/pki/pki-tomcat
配置目录	/etc/pki/pki-tomcat
配置文件	/etc/pki/pki-tomcat/server.xml /etc/pki/pki-tomcat/password.conf
安全数据库	/var/lib/pki/pki-tomcat/alias
子系统证书	SSL 服务器证书 子系统证书 [a]
日志文件	/var/log/pki/pki-tomcat
Web 服务文件	/usr/share/pki/server/webapps/ROOT - Main page /usr/share/pki/server/webapps/pki/admin - Admin templates /usr/share/pki/server/webapps/pki/js - JavaScript 库
[a] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。	



注意

/var/lib/pki/instance_name/conf/ 目录是到 **/etc/pki/instance_name/** 目录的符号链接。

2.3.15.2. CA 子系统信息

*这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。*

表 2.3. CA 子系统信息

设置	值
主目录	/var/lib/pki/pki-tomcat/ca
配置目录	/etc/pki/pki-tomcat/ca
配置文件	/etc/pki/pki-tomcat/ca/CS.cfg
子系统证书	CA 签名证书 OCSP 签名证书（用于 CA 的内部 OCSP 服务） 审计日志签名证书
日志文件	/var/log/pki/pki-tomcat/ca
安装日志	/var/log/pki/pki-ca-spawn.YYYYMMDDhhmmss.log
配置集文件	/var/lib/pki/pki-tomcat/ca/profiles/ca
电子邮件通知模板	/var/lib/pki/pki-tomcat/ca/emails
Web 服务文件	/usr/share/pki/ca/webapps/ca/agent - Agent 服务 /usr/share/pki/ca/webapps/ca/admin - Admin 服务 /usr/share/pki/ca/webapps/ca/ee - 最终用户服务

2.3.15.3. KRA 子系统信息

*这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。*

表 2.4. KRA 子系统信息

设置	值
主目录	/var/lib/pki/pki-tomcat/kra
配置目录	/etc/pki/pki-tomcat/kra
配置文件	/etc/pki/pki-tomcat/kra/CS.cfg

设置	值
子系统证书	传输证书 存储证书 审计日志签名证书
日志文件	/var/log/pki/pki-tomcat/kra
安装日志	/var/log/pki/pki-kra-spawn.YYYYMMDDhhmmss.log
Web 服务文件	/usr/share/pki/kra/webapps/kra/agent - Agent 服务 /usr/share/pki/kra/webapps/kra/admin - Admin services

2.3.15.4. OCSP 子系统信息

*这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。*

表 2.5. OCSP 子系统信息

设置	值
主目录	/var/lib/pki/pki-tomcat/ocsp
配置目录	/etc/pki/pki-tomcat/ocsp
配置文件	/etc/pki/pki-tomcat/ocsp/CS.cfg
子系统证书	OCSP 签名证书 审计日志签名证书
日志文件	/var/log/pki/pki-tomcat/ocsp
安装日志	/var/log/pki/pki-ocsp-spawn.YYYYMMDDhhmmss.log
Web 服务文件	/usr/share/pki/ocsp/webapps/ocsp/agent - Agent 服务 /usr/share/pki/ocsp/webapps/ocsp/admin - Admin services

2.3.15.5. TKS 子系统信息

*这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。*

表 2.6. TKS 子系统信息

设置	值
主目录	/var/lib/pki/pki-tomcat/tks
配置目录	/etc/pki/pki-tomcat/tks
配置文件	/etc/pki/pki-tomcat/tks/CS.cfg
子系统证书	审计日志签名证书
日志文件	/var/log/pki/pki-tomcat/tks
安装日志	/var/log/pki/pki-tomcat/pki-tks-spawn.YYYYMMDDhhmmss.log

2.3.15.6. TPS 子系统信息

*这些目录特定于实例，与实例名称相关联。在这些示例中，实例名称为 **pki-tomcat**; true 值是在使用 **pkispawn** 创建子系统时指定的。*

表 2.7. TPS 子系统信息

设置	值
主目录	/var/lib/pki/pki-tomcat/tps
配置目录	/etc/pki/pki-tomcat/tps
配置文件	/etc/pki/pki-tomcat/tps/CS.cfg
子系统证书	审计日志签名证书
日志文件	/var/log/pki/pki-tomcat/tps
安装日志	/var/log/pki/pki-tps-spawn.YYYYMMDDhhhmmss.log
Web 服务文件	/usr/share/pki/tps/webapps/tps - TPS 服务

2.3.15.7. 共享证书系统子系统文件位置

有一些目录供所有证书系统子系统实例用于常规服务器操作，列在 表 2.8 “子系统文件位置” 中。

表 2.8. 子系统文件位置

目录位置	内容

目录位置	内容
/usr/share/pki	包含用于创建证书系统实例的常用文件和模板。除了所有子系统的共享文件外，子文件夹中还有特定于子系统的文件： • pki/ca (CA) • pki/kra (KRA) • pki/ocsp (OCSP) • pki/tks (TKS) • pki/tps (TPS)
/usr/bin	包含 **pkispawn** 和 **pkidestroy** 实例配置脚本，以及由证书系统子系统共享的 Java、原生和安全性。
/usr/share/java/pki	包含由本地 Tomcat Web 应用程序共享的 Java 归档文件，并由证书系统子系统共享。

2.4. 使用证书系统的 PKI

证书系统由子系统组成，它们各自贡献了公钥基础架构的不同功能。通过为子系统实施不同的功能和功能，可以自定义 PKI 环境，以满足各个需求。

注意

传统 PKI 环境提供基本框架来管理软件数据库中存储的证书。这是一个 非 TMS 环境，因为它不管理智能卡上的证书。TMS 环境管理智能卡上的证书。

非TMS 至少需要一个 CA，但非TMS 环境也可以使用 OCSP 响应器和 KRA 实例。

2.4.1. 发布证书

如前文所述，证书管理器是证书系统的核心。它在每个阶段，通过注册（颁发）、续订和吊销请求来管理证书。

证书系统支持从各种端实体（如 Web 浏览器、服务器和虚拟专用网络(VPN)客户端）注册和处理证书请求。发布的证书符合 X.509 版本 3 标准。

如需更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 About rolling and Renewing Certificates 部分。

2.4.1.1. 注册过程

最终实体通过最终用户界面提交注册请求，从而在 PKI 环境中注册。有很多使用不同注册方法的注册，或者需要不同的身份验证方法。不同的接口也可以接受不同类型的证书签名请求(CSR)。

证书管理器支持不同的方法来提交 CSR，如使用图形界面和命令行工具。

2.4.1.1.1. 使用用户界面注册

对于用户界面的每个注册，创建一个单独的注册页面，该页面特定于注册类型、身份验证类型以及与证书类型关联的证书配置文件。可以为外观和内容自定义与注册相关的表单。或者，可以通过为每个注册类型创建证书配置文件来自定义注册过程。证书配置文件通过配置与证书配置文件关联的输入来动态自定义这些表单。不同的接口也可以接受不同类型的证书签名请求(CSR)。

当端点通过请求证书注册到 PKI 时，可能会发生以下事件，具体取决于 PKI 和安装的子系统的配置：

1. 最终实体以其中一个注册表提供信息并提交请求。

   从最终实体收集的信息可以自定义表单，具体取决于收集以存储在证书中的信息，或针对与表单关联的身份验证方法进行身份验证。该表单会创建一个请求，然后提交到证书管理器。

2. 注册表单会触发公钥和私钥的创建，或为请求创建双密钥对。

3. 最终实体在提交请求前提供身份验证凭据，具体取决于身份验证类型。这可以是 LDAP 身份验证、基于 PIN 的身份验证或基于证书的身份验证。

4. 请求提交至代理批准的注册过程或自动过程。

   - 代理批准的进程（不包括最终用户身份验证）将请求发送到代理服务接口中的请求队列，其中代理必须处理请求。然后，代理可以修改请求的部分，更改请求的状态、拒绝请求或批准请求。

     可以设置自动通知，以便在请求出现在队列中时将电子邮件发送到代理。另外，可以将自动化作业设置为发送队列内容列表，以便在预配置的调度时代理。

   - 涉及最终用户身份验证的自动化过程（涉及最终用户身份验证）会在端点实体成功验证后立即处理证书请求。

5. 表单在提交表单时从 LDAP 目录中收集有关末尾实体的信息。对于基于证书的注册，可以使用表单的默认值来收集用户 LDAP ID 和密码。

6. 与表单关联的证书配置文件决定了要发布的证书的各个方面。根据证书配置文件，评估请求来确定请求是否满足约束集、是否提供了所需信息以及新证书的内容。

7. 表单也可以请求用户导出私钥。如果使用此 CA 设置 KRA 子系统，则会请求最终用户的密钥，并将归档请求发送到 KRA。此过程通常不需要来自端点实体的交互。

8. 证书请求可能被拒绝，因为它不符合证书配置文件或身份验证要求，或者发布证书。

9. 证书被传送到最终实体。

   - 在自动注册中，证书会立即发送给用户。由于注册通常通过 HTML 页面，因此证书将返回为对另一个 HTML 页面的响应。

   - 在代理批准的注册中，证书可以通过序列号或最终用户接口上请求 Id 来检索。

   - 如果设置了通知功能，则可获取证书的链接将发送到最终用户。

10. 当证书发布或被拒绝时，可以向最终实体发送自动通知。

11. 新证书存储在证书管理器的内部数据库中。

12. 如果为证书管理器设置了发布，则证书将发布到文件或 LDAP 目录中。

13. 内部 OCSP 服务在收到证书状态请求时检查内部数据库中的证书状态。

最终用户接口具有搜索表单，用于发布的证书和 CA 证书链。

默认情况下，用户界面支持 PKCS #10 和证书请求消息格式(CRMF)中的 CSR。

2.4.1.1.2. 使用命令行注册

本节论述了使用命令行注册证书时的一般工作流。

2.4.1.1.2.1. 使用 pki 实用程序注册

详情请查看：

- pki-cert(1) man page
- Red Hat Certificate System Administration Guide 中的 命令行界面 部分。

2.4.1.1.2.2. 使用 CMC 注册

要将证书注册到 CMC，请按如下所示操作：

1. 使用工具（如 **PKCS10Client** 或 **CRMFPopClient**）生成 PKCS #10 或 CRMF 证书签名请求 (CSR)。

   注意

   如果在密钥恢复代理(KRA)中启用了密钥归档，请使用带有 KRA 的 Privacy Enhanced Mail (PEM)格式的 KRA 的传输证书的 **CRMFPopClient** 工具，格式为 **kra.transport** 文件。

2. 使用 **CMCRequest** 工具将 CSR 转换为 CMC 请求。 **CMCRequest** 工具使用配置文件作为输入。例如，此文件包含到 CSR 的路径和 CSR 格式。

   详情请查看 CMCRequest(1) man page。

3. 使用 **HttpClient** 实用程序将 CMC 请求发送到 CA。httpclient 使用带有设置的配置文件，如 CMC 请求文件的路径和 servlet。

   如果 HttpClient 命令成功，工具会从 CA 接收一个 PKCS #7 链，其中包含 CMC 状态控制。

   如需有关实用程序提供哪些参数的详细信息，请输入不带任何参数的 HttpClient 命令。

4. **使用 CMCResponse 工具检查由 HttpClient 生成的 PKCS #7 文件的颁发结果。如果请求成功，CMCResponse 以可读格式显示证书链。**

   详情请查看 CMCResponse(1) man page。

5.

将新证书导入到应用程序中。详情请参阅您要导入证书的应用程序的说明。

注意

HttpClient 检索的证书采用 PKCS #7 格式。如果应用程序只支持 Base64 编码的证书，请使用 BtoA 实用程序转换证书。

此外，某些应用程序需要以 Privacy Enhanced Mail (PEM)格式的证书标头和页脚。如果需要它们，请在转换证书后手动将它们添加到 PEM 文件中。

2.4.1.1.2.2.1. 没有 POP 的 CMC 注册

当缺少 proof Of Possession (POP)时，HttpClient 工具会收到 EncryptedPOP CMC 状态，该状态由 CMCResponse 命令显示。在这种情况下，再次输入 CMCRequest 命令，在配置文件中使用不同的参数。

详情请查看 Red Hat Certificate System Administration Guide 中的 CMC Enrollment Process 部分。

2.4.1.1.2.2.2. 签名的 CMC 请求

CMC 请求可以由用户或 CA 代理签名：

- 如果代理为请求签名，请将配置集中的验证方法设置为 CMCAuth。
- 如果用户为请求签名，请将配置集中的验证方法设置为 CMCUserSignedAuth。

详情请查看 Red Hat Certificate System Administration Guide 中的 CMC 身份验证插件 部分。

2.4.1.1.2.2.3. 未签名的 CMC 请求

当在配置集中配置 CMCUserSignedAuth 身份验证插件时，您必须使用未签名的 CMC 请求与 Shared Secret 身份验证机制结合使用。

注意

未签名的 CMC 请求也称为 自签名 CMC 请求。

详情请查看 Red Hat Certificate System Administration Guide 中的 CMC Authentication Plug-ins 部分，和 第 14.8.3 节 “启用 CMC 共享 Secret 功能”。

2.4.1.1.2.2.4. 共享 Secret 工作流

证书系统根据 RFC 5272 为 CMC 请求提供共享 Secret 身份验证机制。为了保护密码短语，在使用 CMCSharedToken 命令时必须提供颁发保护证书。颁发保护证书的工作方式类似于 KRA 传输证书。详情请查看 CMCSharedToken(1) man page 和 第 14.8.3 节 “启用 CMC 共享 Secret 功能”。

由最终用户创建的共享 Secret （首选）

下面描述了当用户生成共享 secret 时的工作流：

1. 最终用户从 CA 管理员获取颁发保护证书。
2. 最终用户使用 CMCSharedToken 工具来生成共享 secret 令牌。

注意

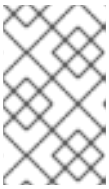
p 选项设置在 CA 和用户之间共享的密码短语，而不是令牌的密码。

3. 最终用户向管理员发送 CMCSharedToken 工具生成的加密共享令牌。
4. 管理员将共享令牌添加到用户的 LDAP 条目的 shrTok 属性中。
5. 最终用户使用密码短语在传递给 CMCRequest 工具的配置文件中设置 witness.sharedSecret 参数。

由 CA 管理员创建的共享 Secret

以下描述了如果 CA 管理员为用户生成共享 secret，则工作流如下：

1. 管理员使用 CMCSharedToken 工具为用户生成共享 secret 令牌。



注意

p 选项设置在 CA 和用户之间共享的密码短语，而不是令牌的密码。

2. 管理员将共享令牌添加到用户的 LDAP 条目的 shrTok 属性中。

3. 管理员与用户共享密语。

4. 最终用户使用密码短语在传递给 CMCRequest 工具的配置文件中设置 witness.sharedSecret 参数。

2.4.1.1.2.2.5. 简单的 CMC 请求

证书系统允许简单的 CMC 请求。但是，这个过程不支持与完整 CMC 请求相同的安全要求，因此只能在安全环境中使用。

使用简单的 CMC 请求时，请在 HttpClient 工具的配置文件中设置以下内容：

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

2.4.1.2. 证书配置文件

证书系统使用证书配置文件来配置证书的内容、发布证书的约束、使用的注册方法以及该注册的输入和输出表单。单个证书配置文件与签发特定类型的证书相关联。

为最常见的证书类型包括一组证书配置文件；可修改配置集设置。证书配置文件由管理员配置，然后发送到代理服务页面进行代理批准。批准证书配置文件后，它就被启用以供使用。如果注册 UI，则在终端实体页面中使用动态生成的 HTML 表单进行证书注册，该表单会调用证书配置文件。如果是基于命令行的注册，则在 上调用证书配置文件来执行相同的处理，如身份验证、授权、输入、输出、默认值和约束。在对请求执行前，服务器会验证证书配置文件中设置的默认值和限制，并使用证书配置文件来确定发布的证书的内容。

证书管理器可使用以下任何特征发布证书，具体取决于配置集和提交的证书请求：

- X.509 版本 3 兼容的证书
- Unicode 支持证书主题名称和签发者名称
- 支持空证书主题名称
- 支持自定义主题名称组件
- 支持自定义扩展

默认情况下，证书注册配置文件以 < profile id &gt ; .cfg格式存储在 <instance 目录 >/ca/profiles/ca 中。可以使用正确的 pkispawn 配置参数进行基于 LDAP 的配置集。

2.4.1.3. 证书注册身份验证

证书系统提供证书注册的验证选项。这包括代理批准的注册（代理处理请求）和自动注册，用于验证端点，然后 CA 会自动发布证书。CMC 注册也被支持，它会自动处理代理批准的请求。

2.4.1.4. 跨证书

通过在这两个 CA 之间发布和存储跨签名证书，可以在两个独立 CA 之间创建可信关系。通过使用跨签名证书对，可在系统内信任在组织的 PKI 外部发布的证书。

2.4.2. 续订证书

当证书达到其过期日期时，可以允许它们释放，或者可以续订它们。

续订 使用该证书的现有密钥对重新生成证书请求，然后将请求重新提交到证书管理器。续订的证书与原始证书相同（因为是使用同一密钥材料的同一配置文件创建），但有一个例外 - 它具有不同的过期日期。

续订可以使用户和服务器之间的证书和关系更加平稳地管理，因为续订的证书功能会精确作为旧证书功能。对于用户证书，续订允许在不丢失任何丢失的情况下访问加密数据。

2.4.3. 发布证书和 CRL

证书可以发布到文件和 LDAP 目录，以及 CRL 到文件、LDAP 目录和 OCSP 响应程序。发布框架提供了一组强大的工具，用于发布到所有这三个位置，并设置规则来定义什么类型的证书或 CRL 在哪里发布。

2.4.4. 撤销证书和检查状态

最终实体可以请求撤销自己的证书。当终端实体发出请求时，必须将证书提供给 CA。如果证书和密钥可用，则请求将处理并发送到证书管理器，证书将被撤销。证书管理器在其数据库中将证书标记为已撤销，并将其添加到任何适用的 CRL 中。

代理可以通过在代理服务界面中搜索证书，然后标记它来撤销证书。证书被撤销后，它将标记为在数据库中和发布目录中撤销（如果为发布证书）。

如果配置了内部 OCSP 服务，该服务可以通过在内部数据库中查找证书来确定证书的状态。

通过启用和配置证书撤销时，可以将自动通知设置为将电子邮件消息发送到最终实体。

2.4.4.1. 吊销证书

用户可以使用以下方法撤销其证书：

- 最终用户页面。详情请查看 Red Hat Certificate System Administration Guide 中的 Certificate Revocation Pages 部分。
- 命令行上的 CMCRequest 工具。详情请参阅 Red Hat Certificate System Administration Guide 中的 执行 CMC Revocation 部分。
- 命令行中的 pki 工具。详情请查看 pki-cert(1) man page。

2.4.4.2. 证书状态

2.4.4.2.1. CRL

证书系统可以从可配置的框架创建证书撤销列表(CRL)，允许用户定义的发布点，以便为每个发出点创建 CRL。也可以为定义的任何发出点创建 delta CRL。可以为每种类型的证书发布 CRL，适用于特定类型的证书，或针对根据配置集或配置文件列表生成的证书。发布 CRL 时可使用的扩展以及可以配置 CRL 的频率和间隔。

证书管理器发出 X.509 标准 CRL。每当撤销证书或指定间隔时，都可自动更新 CRL。

2.4.4.2.2. OCSP 服务

证书系统 CA 支持 PKIX 标准 RFC 2560 中定义的在线证书状态协议(OCSP)。OCSP 协议可让 OCSP 兼容应用程序来确定证书的状态，包括撤销状态，而无需直接检查 CA 发布的 CRL 到验证机构。验证授权（也称为 OCSP 响应器 ）检查应用程序。

1. 将 CA 设置为发布包含授权信息访问扩展的证书，用于标识可查询证书状态的 OCSP 响应程序。
2. CA 定期向 OCSP 响应程序发布 CRL。
3. OCSP 响应器维护它从 CA 接收的 CRL。
4. 兼容 OCSP 的客户端会发送请求，其中包含将证书标识到 OCSP 响应程序以进行验证所需的所有信息。应用程序从被验证的证书中的授权信息访问扩展值确定 OCSP 响应器的位置。
5. OCSP 响应器确定请求是否包含处理它所需的所有信息。如果没有或未为请求的服务启用它，则会发送拒绝通知。如果它有足够的信息，它会处理请求并发回一个报告，说明证书的状态。

2.4.4.2.2.1. OCSP 响应签名

客户端收到的每个响应（包括拒绝通知）都由响应者进行数字签名；客户端应验证签名，以确保响应来自提交请求的响应者。响应者用来签署消息的键取决于 PKI 设置中如何部署 OCSP 响应程序。RFC 2560 建议用于为响应签名的密钥属于以下之一：

- 签发正在检查其状态的证书的 CA。

- 带有客户端信任的公钥的响应程序。这样的响应者称为 可信响应器。

- 一个响应程序，包含由 CA 直接向它发布的证书，用于撤销证书并发布 CRL。此证书由响应者拥有此证书，表示 CA 已授权响应者为 CA 撤销的证书发布 OCSP 响应。此类响应程序称为 CA 设计的响应程序或 CA 授权响应器。

证书管理器的端点页面包括用于为 OCSP 响应程序手动请求证书的表单。默认注册表包括将证书识别为 OCSP 响应器证书的所有属性。提交证书请求时，可以将所需的证书扩展（如 OCSPNoCheck 和 Extended Key Usage）添加到证书中。

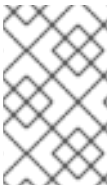
2.4.4.2.2.2. OCSP 响应

客户端接收的 OCSP 响应指示证书的当前状态，由 OCSP 响应器决定。响应可以是以下任意一种：

- 良好或验证的 .指定对状态查询的正响应，这意味着证书尚未被撤销。它不一定意味着签发证书，或者在证书的有效性间隔内。响应扩展可用于传达由响应者有关证书状态的断言的附加信息。

- 已撤销 .指定证书已被撤销，可以是永久或临时撤销。

根据状态，客户端决定是否验证证书。



注意

OCSP 响应器永远不会返回 Unknown 的响应。响应始终为 Good 或 Revoked。

2.4.4.2.2.3. OCSP 服务

设置 OCSP 服务的方法有两种：

- 内置在证书管理器中的 OCSP

- 在线证书状态管理器子系统

除了内置的 OCSP 服务外，证书管理器还可以将 CRL 发布为兼容 OCSP 的验证机构。CA 可以被配置为将 CRL 发布到证书系统在线证书状态管理器。在线证书状态管理器将每个证书管理器的 CRL 存储在其内部数据库中，并使用适当的 CRL 在由 OCSP 兼容客户端查询时验证证书的撤销状态。

每当证书被撤销和按指定间隔撤销时，证书管理器都可以生成并发布 CRL。由于 OCSP 响应器的目的是有助于立即验证证书，因此每次撤销证书时，证书管理器都应该将 CRL 发布到在线证书状态管理器。仅按间隔发布意味着 OCSP 服务正在检查过时的 CRL。

注意

如果 CRL 较大，则证书管理器可能需要很长时间才能发布 CRL。

在线证书状态管理器将每个证书管理器的 CRL 存储在其内部数据库中，并将其用作 CRL 来验证证书。在线证书状态管理器也可以使用发布到 LDAP 目录的 CRL，这意味着证书管理器不必直接将 CRL 更新至在线证书状态管理器。

2.4.5. 归档、恢复和轮转密钥

在 PKI 世界中，私钥存档允许各方在私钥丢失时恢复加密的数据。由于硬件故障，可能会丢失私钥，如硬件故障，忘记密码丢失，智能卡丢失，受损密码持有者、et caetera 等。此类归档和恢复功能由 RHCS 的密钥恢复授权(KRA)子系统提供。

只有专门用于加密数据的密钥才应存档；特定中的签名密钥永远不会存档。拥有两个签名密钥副本会导致无法识别使用密钥的某些人；第二个存档副本可用于模拟原始密钥所有者的数字身份。

2.4.5.1. 归档密钥

KRA 提供两种关键归档机制：

- 客户端密钥生成 ：使用此机制，客户端将以 CRMF 格式生成 CSR，并将请求提交到 CA（具有正确的 KRA 设置），以进行注册和密钥存档。请参阅 Red Hat Certificate System Administration Guide 中的使用 CRMFPopClient 创建 CSR 部分。

- 服务器端密钥生成 ：使用此机制，正确配备证书注册配置文件将触发 KRA 上生成的 PKI 密钥，从而可以选择与新发布的证书一起存档。请参阅 Red Hat Certificate System Administration Guide 中的使用 Server-Side Key Generation 生成 CSR 部分。

如果配置了存档，则 KRA 会自动归档私钥。

KRA 将私钥存储在安全密钥存储库中；每个密钥加密并存储为密钥记录，并被授予唯一的密钥标识符。

密钥的归档副本与 KRA 的存储密钥一起嵌套。它只能通过使用存储证书的对应私钥对来解密或解压缩。一个或多个密钥恢复（或 KRA）代理的证书组合授权 KRA 完成密钥恢复，以检索其私钥并使用它来解密/恢复归档的私钥。请参阅 第 17.3.1 节 “在命令行中配置代理批准密钥恢复”。

KRA 根据密钥号、所有者名称和公钥的哈希索引存储的密钥，从而实现高效率的搜索。密钥恢复代理具有插入、删除和搜索密钥记录的特权。

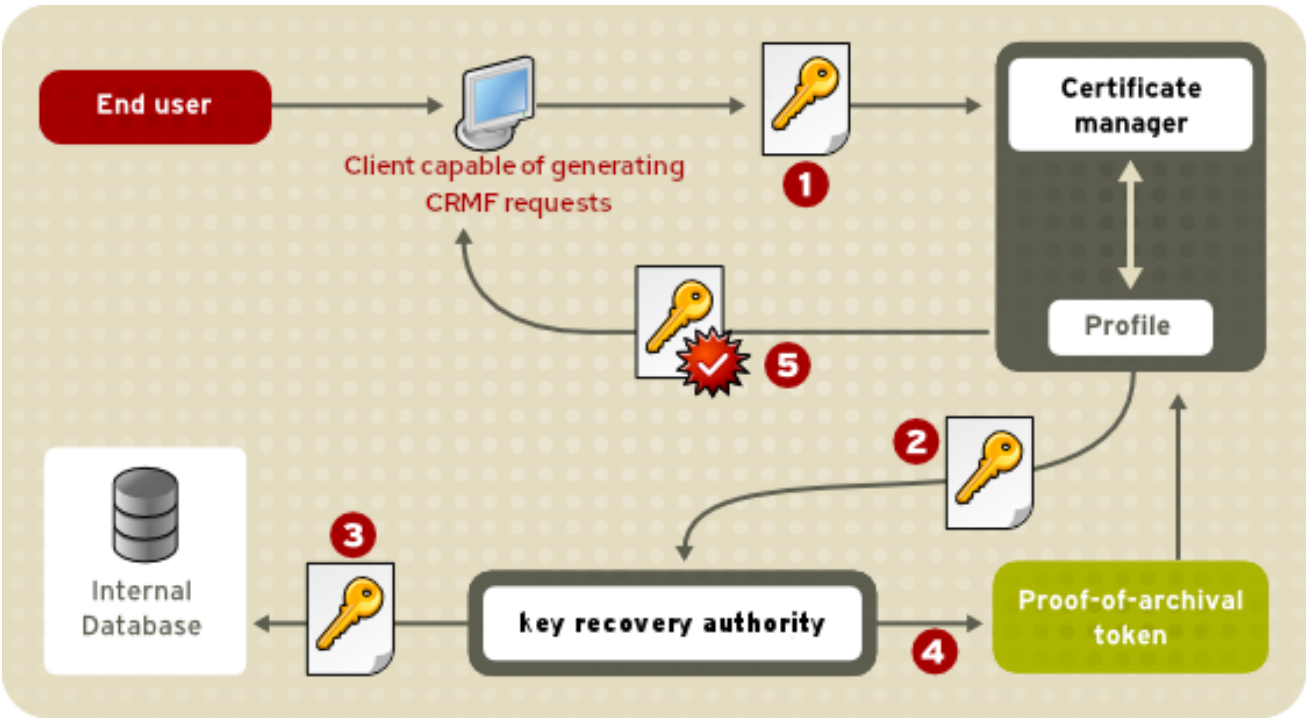
- 当密钥恢复代理根据密钥 ID 搜索时，只返回与该 ID 对应的键。
- 当代理根据用户名搜索时，所有属于该所有者的存储密钥都会被返回。
- 当代理根据证书中的公钥搜索时，仅返回对应的私钥。

当证书管理器收到包含密钥归档选项的证书请求时，它会自动将请求转发到 KRA 以归档加密密钥。私钥由传输密钥加密，KRA 接收加密的副本，并将密钥存储在其密钥存储库中。要归档密钥，KRA 使用两个特殊密钥对：

- 传输密钥对和对应的证书。
- 存储密钥对。

图 2.2 “Client-Side Key Generation 中的 Key Archival Process Works” 演示了当最终用户在客户端密钥生成时请求证书时，密钥归档过程如何进行。

图 2.2. Client-Side Key Generation 中的 Key Archival Process Works



1. 客户端生成 CRMF 请求，并通过 CA 的注册门户提交它。

   a. 客户端的私钥嵌套在 CRMF 请求中，只能由 KRA 解封。

2. 检测它是带有密钥归档选项的 CRMF 请求，CA 会将请求转发到 KRA 的私钥存档。

3. KRA 解密 / 取消打包用户私钥，并在确认私钥与公钥对应后，KRA / 在将其存储在其内部 LDAP 数据库中之前再次加密 /。

4. 成功存储私钥后，KRA 会对 CA 响应，确认密钥已成功存档。

5. CA 会向请求发送 Enrollment Profile Framework，以进行证书信息内容创建和验证。当一切通过后，都会发出证书并将其发回到其响应中的后端实体。

2.4.5.2. 恢复密钥

KRA 支持 代理发起的密钥恢复。代理初始化的恢复是指定的恢复代理使用 KRA 代理服务门户上的密钥恢复表单来处理和批准密钥恢复请求。批准指定数量的代理后，组织可以在密钥所有者不可用时或者密钥丢失时恢复密钥。

通过 KRA 代理服务门户，密钥恢复代理可以集中授权并检索私钥和相关证书到 PKCS grants 软件包，然后可以导入到客户端。

在密钥恢复授权中，其中一个关键恢复代理会通知所有必需的恢复代理有关即将进行密钥恢复的情况。所有恢复代理都访问 KRA 密钥恢复门户。其中一个代理启动密钥恢复过程。KRA 向代理返回通知包括恢复授权引用号，用于标识代理需要的特定密钥恢复请求。每个代理都使用引用号并单独授权密钥恢复。

KRA 支持 异步恢复，这意味着恢复过程的每个步骤（初始请求和后续批准或拒绝）都存储在 KRA 的内部 LDAP 数据库中。即使原始浏览器会话关闭或 KRA 关闭，也可以检索恢复过程的状态数据。代理可以搜索要恢复的密钥，而无需使用参考号。

这个异步恢复选项在 图 2.3 “异步恢复” 中进行了说明。

图 2.3. 异步恢复

KRA 告知代理启动授权状态的密钥恢复过程。

输入所有授权后，KRA 会检查信息。如果显示的信息正确，它会检索请求的密钥，并以 PKCSGRESS 软件包的形式返回对应的证书，到启动密钥恢复过程的代理。

警告

PKCSRG 软件包包含加密的私钥。为最大程度降低密钥威胁的风险，恢复代理必须使用安全方法向密钥接收者提供 PKCSRG 软件包和密码。代理应该使用很好的密码来加密 PKCSRG 软件包并设置适当的交付机制。

密钥恢复代理方案 将 KRA 配置为识别密钥恢复代理所属的组，并指定在恢复归档密钥前授权密钥恢复请求所需的数量。

重要

以上信息指的是使用 Web 浏览器，如 Firefox。但是，在 Red Hat Enterprise Linux 7 平台上发布的 Firefox 版本 31.6 中不再包括 KRA 使用的功能。在这种情况下，需要使用 pki 工具来复制此行为。如需更多信息，请参阅 pki(1) 和 pki-key(1) man page 或运行 CRMFPopClient --help 和 man CMCRequest。

除了存储非对称密钥外，KRA 还可以存储与对称密钥类似的对称密钥或机密，如卷加密 secret，甚至密码和密码短语。pki 工具支持启用存储和检索这些其他类型的 secret 的选项。

2.4.5.3. KRA 传输密钥轮转

KRA 传输轮转允许使用当前和新的传输密钥在 CA 和 KRA 子系统实例之间进行无缝转换。这允许定期轮转 KRA 传输密钥，方法是允许旧的和新的传输密钥在转换期间运行；单个子系统实例在其他克隆继续不停机的情况下进行配置。

在 KRA 传输密钥轮转过程中，会生成一个新的传输密钥对，提交证书请求，并检索新的传输证书。新的传输密钥对和证书必须包含在 KRA 配置中，以支持第二个传输密钥。KRA 支持两种传输密钥后，管理员可以开始将 CA 转换到新的传输密钥。所有 CA 移至新传输密钥后，可以删除对旧传输密钥的 KRA 支持。

配置 KRA 传输密钥轮转：

1. **生成新的 KRA 传输密钥和证书**
2. **将新传输密钥和证书传输到 KRA 克隆**
3. **使用新的 KRA 传输证书更新 CA 配置**
4. **更新 KRA 配置，使其仅使用新的传输密钥和证书**

之后，KRA 传输证书轮转已完成，所有受影响的 CA 和 KRA 都使用新的 KRA 证书。有关如何执行上述步骤的更多信息，请参阅以下步骤。

生成新的 KRA 传输密钥和证书

1. **请求 KRA 传输证书。**
   a. **停止 KRA：**

```
# pki-server stop pki-kra
```

或（如果使用 nuxwdog watchdog）

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

   b. **进入 KRA NSS 数据库目录：**

```
# cd /etc/pki/pki-kra/alias
```

   c. **创建子目录并将所有 NSS 数据库文件保存到其中。例如：**

```
mkdir nss_db_backup
cp *.db nss_db_backup
```

d. 使用 PKCS10Client 工具创建新请求。例如：

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

或者，使用 certutil 工具。例如：

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

e. 在 CA End-Entity 页面的 Manual Data Recovery Manager 传输证书注册 页中提交传输证书请求。

f. 通过检查 End-Entity 检索页面中的请求状态，等待提交请求的代理批准检索证书。

2. 通过 CA Agent Services 接口批准 KRA 传输证书。

3. 检索 KRA 传输证书。

a. 进入 KRA NSS 数据库目录：

```
# cd /etc/pki/pki-kra/alias
```

b. 通过检查 End-Entity 检索页面中的请求状态，等待提交请求的代理批准检索证书。

c. 新的 KRA 传输证书可用后，将其 Base64 编码的值粘贴到文本文件中，例如名为 cert-serial_number.txt 的文件。不要包括标头(-----BEGIN CERTIFICATE-----)或页脚(-----END CERTIFICATE-----)。

4. 导入 KRA 传输证书。

a. 进入 KRA NSS 数据库目录：

```
# cd /etc/pki/pki-kra/alias
```

b. 将传输证书导入到 KRA NSS 数据库中：

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

5. 更新 KRA 传输证书配置。

a. 进入 KRA NSS 数据库目录：

```
# cd /etc/pki/pki-kra/alias
```

b. 验证新的 KRA 传输证书是否已导入：

```
# certutil -d . -L
# certutil -d . -L -n 'transportCert-serial_number cert-pki-kra KRA'
```

c. 打开 /var/lib/pki/pki-kra/kra/conf/CS.cfg 文件并添加以下行：

```
kra.transportUnit.newNickName=transportCert-serial_number cert-pki-kra KRA
```

将新传输密钥和证书传播到 KRA 克隆

1. 启动 KRA：

```
# pki-server start pki-kra
```

或（如果使用 nuxwdog watchdog）

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

2. 提取新的传输密钥和证书以传播到克隆。

   a. 进入 KRA NSS 数据库目录：

   ```
   # cd /etc/pki/pki-kra/alias
   ```

   b. 停止 KRA：

   ```
   # pki-server stop pki-kra
   ```

   或（如果使用 nuxwdog watchdog）

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

   c. 验证新的 KRA 传输证书是否存在：

   ```
   # certutil -d . -L
   # certutil -d . -L -n 'transportCert-serial_number cert-pki-kra KRA'
   ```

   d. 导出 KRA 新传输密钥和证书：

   ```
   # pk12util -o transport.p12 -d . -n 'transportCert-serial_number cert-pki-kra KRA'
   ```

   e. 验证导出的 KRA 传输密钥和证书：

   ```
   # pk12util -l transport.p12
   ```

3. 在每个 KRA 克隆中执行这些步骤：

   a. 将 transport.p12 文件（包括传输密钥和证书）复制到 KRA 克隆位置。

   b. 进入克隆 NSS 数据库目录：

```
# cd /etc/pki/pki-kra/alias
```

c. **停止 KRA 克隆：**

```
# pki-server stop pki-kra
```

或（如果使用 nuxwdog watchdog）

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

d. **检查克隆 NSS 数据库的内容：**

```
# certutil -d . -L
```

e. **导入克隆的新传输密钥和证书：**

```
# pk12util -i transport.p12 -d .
```

f. **在克隆上的 /var/lib/pki/pki-kra/kra/conf/CS.cfg 文件中添加以下行：**

```
kra.transportUnit.newNickName=transportCert-serial_number cert-pki-kra KRA
```

g. **启动 KRA 克隆：**

```
# pki-server start pki-kra
```

或（如果使用 nuxwdog watchdog）

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

使用新的 KRA 传输证书更新 CA 配置

1. **格式化新的 KRA 传输证书以包含在 CA 中。**

a. **获取在前面的过程中获取 KRA 传输证书时创建的 cert-serial_number.txt KRA**

传输证书文件。

b. 将 cert-serial_number.txt 中包含的 Base64 编码的证书转换为单行文件：

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

2. 对 CA 及其所有与上述 KRA 对应的克隆执行以下操作：

a. 停止 CA：

```
# pki-server stop pki-ca
```

或（如果使用 nuxwdog watchdog）

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

b. 在 /var/lib/pki/pki-ca/ca/conf/CS.cfg 文件中，找到以下行中包含的证书：

```
ca.connector.KRA.transportCert=certificate
```

将该证书替换为 cert-one-line-serial_number.txt 中包含的证书。

c. 启动 CA：

```
# pki-server start pki-ca
```

或（如果使用 nuxwdog watchdog）

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

注意

虽然 CA 及其克隆都使用新的 KRA 传输证书更新，但完成转换的 CA 实例使用新的 KRA 传输证书，但尚未更新的 CA 实例继续使用旧的 KRA 传输证书。由于对应的 KRA 及其克隆已更新为同时使用这两个传输证书，因此不会停机。

更新 KRA 配置，使其仅使用新的传输密钥和证书

对于 KRA 及其每个克隆，请执行以下操作：

1. 进入 KRA NSS 数据库目录：

```
# cd /etc/pki/pki-kra/alias
```

2. 停止 KRA：

```
# pki-server stop pki-kra
```

或（如果使用 nuxwdog watchdog）

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

3. 验证新的 KRA 传输证书是否已导入：

```
# certutil -d . -L
# certutil -d . -L -n 'transportCert-serial_number cert-pki-kra KRA'
```

4. 打开 /var/lib/pki/pki-kra/kra/conf/CS.cfg 文件，并查找以下行中包含的 nickName 值：

```
kra.transportUnit.nickName=transportCert cert-pki-kra KRA
```

将 nickName 值替换为以下行中包含的 newNickName 值：

```
kra.transportUnit.newNickName=transportCert-serial_number cert-pki-kra KRA
```

因此，CS.cfg 文件包含这一行：

```
kra.transportUnit.nickName=transportCert-serial_number cert-pki-kra KRA
```

5. 从 /var/lib/pki/pki-kra/kra/conf/CS.cfg 中删除以下行：

```
kra.transportUnit.newNickName=transportCert-serial_number cert-pki-kra KRA
```

6. 启动 KRA：

```
# pki-server start pki-kra
```

或（如果使用 nuxwdog watchdog）

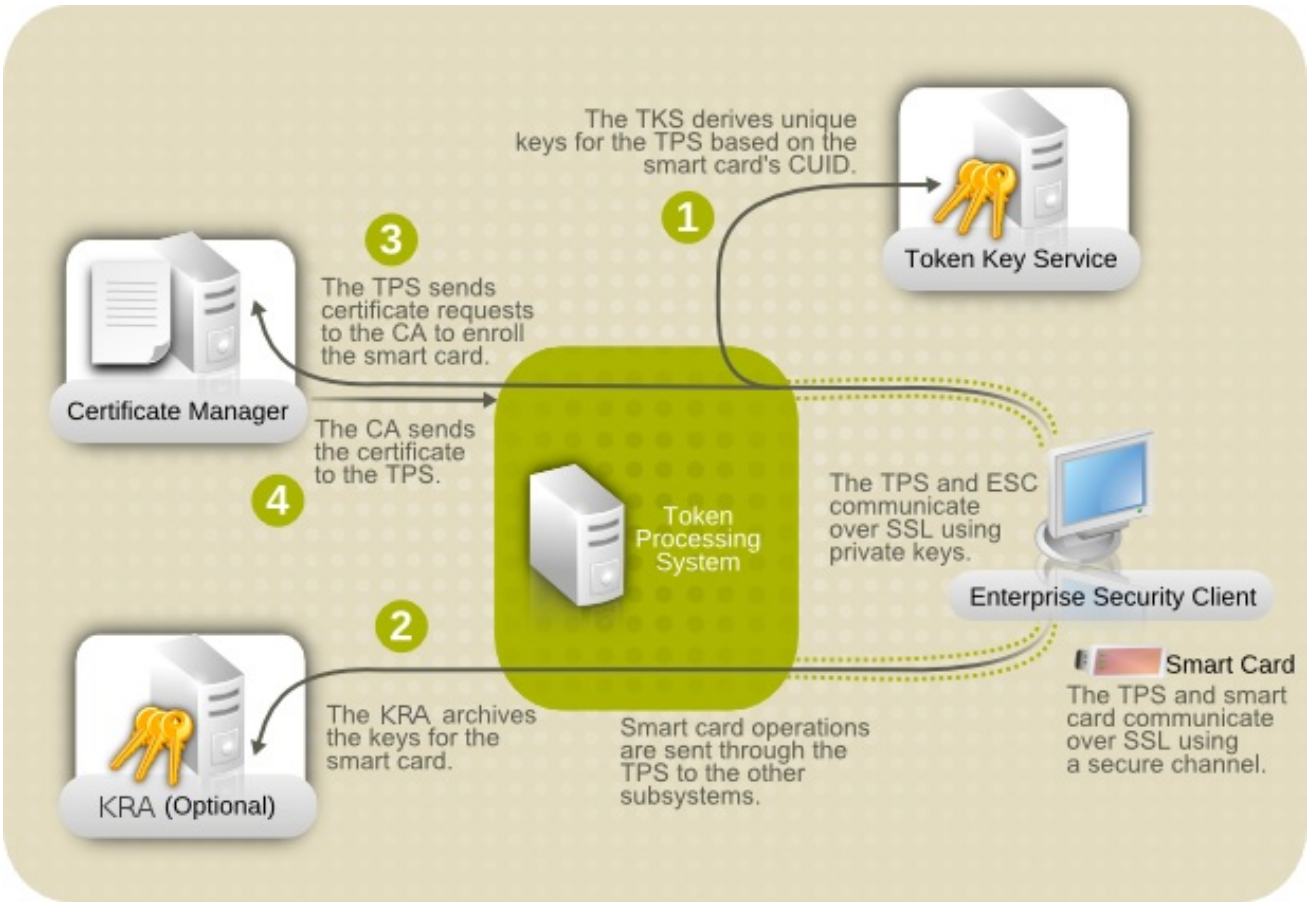
```
# systemctl start pki-tomcatd-nuxwdog@pki-kra.service
```

2.5. 使用证书系统进行智能卡令牌管理

智能卡 是一个包含加密证书和密钥的硬件加密设备。用户可使用它参与安全网站访问和安全邮件等操作。它还可作为身份验证设备来登录到 Red Hat Enterprise Linux 等各种操作系统。在服务的整个生命周期中，这些卡或令牌的 管理都通过令牌管理系统 (TMS)来完成。

TMS 环境需要证书颁发机构(CA)、令牌密钥服务(TKS)和令牌处理系统(TPS)，以及可选的密钥恢复授权(KRA)，用于服务器端密钥生成和密钥存档和恢复。在线证书状态协议(OCSP)也可用于与 CA 一起使用，为在线证书状态请求提供服务。本章概述了 TKS 和 TPS 系统，它提供了 Red Hat Certificate System 的智能卡管理功能，以及用户末尾与 TMS 一起工作的企业安全客户端(ESC)。

图 2.4. TMS 如何管理智能卡



2.5.1. 令牌密钥服务(TKS)

令牌密钥服务(TKS)负责管理一个或多个主密钥。它维护主密钥，是 TMS 中可访问密钥资料的唯一实体。在操作环境中，每个有效的智能卡令牌包含一组对称密钥，这些密钥来自主密钥和对卡的唯一 ID (CUID)。

最初，制造商在每个智能卡上初始化一组默认的（每个制造商主密钥）。部署站点中应通过一个 Key Changeover 操作来更改此默认设置，以在 TKS 上生成新的主密钥。作为主密钥的唯一所有者，当给出智能卡的 CUID 时，TKS 能够派生位于该特定智能卡上的对称密钥集合，然后允许 TKS 建立基于会话的安全频道，以便在 TMS 和每个智能卡之间进行安全通信。

注意

由于 TKS 管理的数据敏感度，应在具有受限访问权限的防火墙后设置 TKS。

2.5.1.1. 主密钥和密钥设置

TKS 支持多个智能卡密钥集。每个智能卡厂商为其智能卡令牌库存创建不同的默认（开发人员）静态密钥集，而 TKS 则配备了静态密钥集（每个制造商）来 kickstart 空白令牌的格式过程。

在空白智能卡令牌的格式过程中，Java 小程序和唯一派生的对称密钥集将注入到令牌中。每个主密钥（在某些情况下称为 keySet）是在 TKS 配置文件(CS.cfg)中有一组条目。每个 TPS 配置集都包含一个配置，用于将其注册到匹配密钥派生过程的适当 TKS keySet，这基本上负责在 TMS 和智能卡令牌之间建立由一组特定于会话的密钥保护的安全频道。

在 TKS 上，主密钥通过命名 keySets 来定义，供 TPS 引用。在 TPS 上，根据注册类型（内部或外部注册）在 TPS 配置集中指定，或者由 keySet Mapping Resolver 决定。

2.5.1.2. Key Ceremony (Shared Key Transport)

Key Ceremony 是一个以安全的方式将高度敏感的密钥从一个位置传输到另一个位置的过程。在一个情况下，在一个高度安全的部署环境中，可以在没有网络到外部的安全库中生成主密钥。或者，组织可能希望在不同的物理机器上具有 TKS 和 TPS 实例。在这两种情况下，假设没有人信任密钥，Red Hat Certificate System TMS 都提供了一个称为 tkstool 的工具来管理安全密钥传输。

2.5.1.3. 密钥更新（密钥更改）

当在工厂上创建 Global Platform 兼容智能卡时，制造商会将一组默认对称密钥刻录到令牌中。TKS 最初配置为使用这些对称密钥(TKS 配置中的每个厂商一个 KeySet 条目)。但是，由于这些对称密钥不是同一股票的智能卡，并且由于这些都是知名的密钥，因此强烈建议将这些对称密钥替换为每个令牌的唯一集合，而不是制造商共享，以限制一组可以操作令牌的实体。

通过令牌密钥 Service 子系统帮助更改密钥。TKS 的一个功能是查看之前讨论的智能卡令牌密钥派生的主密钥。可以在 TKS 控制下有多个主键。

重要

当在令牌上进行这个密钥更改过程时，令牌可能会在以后无法使用，因为它不再启用默认密钥。只要置备令牌有效的 TPS 和 TKS 系统，密钥基本上就好。因此，务必要保留所有主密钥，即使其中任何一个密钥都已过时。

您可以禁用 TKS 中的旧主密钥以更好地控制，但除非禁用的令牌是您的计划的一部分，否则请不要删除它们。支持将令牌密钥还原回原始密钥集，如果令牌要在某种测试方案中再次重复使用，则这是可行的。

2.5.1.4. APDU 和安全频道

Red Hat Certificate System Token Management System (TMS)支持 GlobalPlatform 智能卡规格，其中安全频道实现通过管理主密钥(TKS)的 Token Key System (TKS)完成，以及与 应用程序协议数

据单元 (APDU)通信的智能卡（令牌）。

APDU 有两种类型：

- 命令 A PDU 由 TPS 发送到智能卡
- 响应 A PDU，由智能卡发送给 TPS，作为命令 APDU 的响应

当客户端采取行动并连接到证书系统服务器以进行请求时，可能会触发 APDU 命令的启动。安全频道从 TPS 发送到智能卡令牌 InitializeUpdate APDU 开始，并完全由 ExternalAuthenticate APDU 建立。然后，令牌和 TMS 都建立了一组共享 secret，称为会话密钥，用于加密和验证通信。这个经过验证和加密的通信通道被称为 Secure Channel。

因为 TKS 是唯一可以访问主密钥的实体，它能够生成一组唯一的对称 on-token 智能卡密钥，因此 Secure Channel 提供了 TMS 和每个令牌之间的足够保护通信。任何与频道断开连接都需要为新频道重新建立新会话密钥。

2.5.2. 令牌处理系统(TPS)

令牌处理系统(TPS)是智能卡证书注册的注册机构。它充当用户中心化企业安全客户端(ESC)之间的代表，它与客户端侧智能卡令牌交互，以及证书系统后端子系统，如证书颁发机构(CA)和密钥恢复授权(KRA)。

在 TMS 中，需要 TPS 来管理智能卡，因为它是唯一了解 APDU 命令和响应的 TMS 实体。TPS 将命令发送到智能卡，以帮助它们为特定实体（如用户或设备）生成和存储密钥和证书。智能卡操作经过 TPS，并转发到相应的子系统，如 CA 来生成证书或 KRA 来生成、存档或恢复密钥。

2.5.2.1. CoolKey Applet

Red Hat Certificate System 包括 Coolkey Java 小程序，专门在 TMS 支持的智能卡令牌上运行。Coolkey 小程序连接到处理证书和密钥相关操作的 PKCSFeature 模块。在令牌格式操作过程中，这个小程序使用 Secure Channel 协议注入到智能卡令牌中，并可为每个配置进行更新。

2.5.2.2. 令牌操作

Red Hat Certificate System 中的 TPS 可用于代表智能卡最终用户置备智能卡。令牌处理系统提供对以下主要令牌操作的支持：

- Token Format - 格式操作负责将正确的 Coolkey 小程序安装到令牌中。小程序提供了一个平台，稍后可以放置后续的加密密钥和证书。

- 令牌注册 - 注册操作会导致智能卡使用所需的加密密钥和加密证书填充。本材料允许智能卡的用户参与诸如安全网站访问和安全邮件等操作。支持两种类型的注册，这些注册在全局配置：

  - 内部注册 - 由配置集 映射 解析器决定的 TPS 配置集注册。

  - 外部注册 - 由 TPS 配置集注册，由用户的 LDAP 记录中的条目决定。

- Token PIN Reset - 令牌 PIN 重置操作允许用户指定用来登录到令牌的新 PIN，使其可用于执行加密操作。

以下其他操作可以被视为上面所列主要操作的补充或固有的操作。它们可以根据相关配置或令牌状态触发。

- Key Generation - 每个 PKI 证书由一个公钥/私钥对组成。在 Red Hat Certificate System 中，可以通过两种方式生成密钥，具体取决于 TPS 配置集配置 ：

  - Token Side Key Generation - PKI 密钥对在智能卡令牌中生成。在令牌端生成密钥对不允许 进行密钥归档。

  - Server Side Key Generation - PKI 密钥对在 TMS 服务器端生成。然后，密钥对会使用 Secure Channel 发回到令牌。在服务器端生成密钥对允许密钥归档。

- 证书续订 - 此操作允许之前注册的令牌在重新使用同一密钥时重新发布令牌中的证书。当旧证书到期并且您希望创建新证书但维护原始密钥材料时，这非常有用。

- 证书撤销 - 证书撤销可根据 TPS 配置集配置或基于令牌状态触发。

通常，只有发布的证书的 CA 可以撤销证书，这可能意味着重新创建 CA 无法撤销某些证书。但是，可以在将令牌的撤销请求路由到已停用的 CA 时，仍然会将所有其他请求路由，如注册至新的活动 CA。这种机制被称为撤销 路由。

- 令牌密钥更改 - 由格式操作触发的关键 更改操作，从而能够将令牌的内部密钥从默认开发人员密钥集更改为由令牌处理系统的部署者控制的新密钥。这通常在任何实际的部署场景中完成，因为开发人员密钥集更适合测试情况。

- 小程序更新 - 在 TMS 部署过程中，可以根据需要更新或降级 Coolkey 智能卡小程序。

2.5.2.3. TPS 配置文件

证书系统令牌处理系统子系统有助于管理智能卡令牌。令牌由 TPS 置备，以便它们从空状态转换为格式或注册条件。格式化令牌是一个包含 TPS 支持的 CoolKey 小程序令牌，而 Enrolled 令牌是个人（称为 绑定）的个人，具有必要的证书和加密密钥。这个完全置备的令牌可用于 crytptographic 操作。

TPS 也可以管理配置文件 。令牌配置文件的概念与以下内容相关：

- 格式化或注册令牌的步骤。

- 在操作成功完成后，已完成的令牌中包含的属性。

以下列表包含组成唯一令牌配置集的一些数量：

- TPS 如何连接到用户的身份验证 LDAP 数据库？

- 此令牌操作是否需要用户身份验证？如果是，将使用什么身份验证管理器？

- TPS 如何连接到要获取证书的证书系统 CA？

- 如何在此令牌中生成私钥和公钥？它们是否在令牌端或服务器端生成？

- 生成私钥和公钥时要使用的密钥大小（以位为单位）？

- 哪个证书注册配置文件（由 CA 置备）用于在这个令牌上生成证书？

注意

此设置将决定要写入令牌的证书的最终结构。根据证书中包含的扩展，可以为不同的使用创建不同的证书。例如，一个证书可以特殊化数据加密，另一个证书可用于签名操作。

- 令牌上需要哪个 Coolkey 小程序版本？
- 在此令牌中，将放置多少个证书用于注册操作？

以上以及许多其他令牌类型或配置集可以为每个令牌类型或配置集进行配置。Red Hat Certificate System Administration Guide 中提供了可用配置选项的完整列表。

要考虑的另一个问题是用户置备给定令牌的方式将被映射到单个令牌配置文件。注册有两种类型：

- 内部注册 - 在这种情况下，TPS 配置集(tokenType)由配置集 Mapping Resolver 决定。这个基于过滤器的解析器可以配置为考虑令牌提供的任何数据，并确定目标配置集。
- 外部注册 - 使用外部注册时，配置集（仅在名称中 - 实际的配置文件仍然在 TPS 中定义，其方式与内部注册使用的相同方式在每个用户的 LDAP 记录中指定，该记录在身份验证过程中获得）。这允许 TPS 从存储用户信息的外部注册目录服务器获取密钥注册和恢复信息。这可让您控制覆盖 TPS 内部注册机制固有的注册、撤销和恢复策略。与外部注册相关的用户 LDAP 记录属性名称可以配置。

  当需要"组证书"的概念时，外部注册非常有用。在这种情况下，组中的所有用户都可以在其 LDAP 配置文件中配置特殊的记录来下载共享证书和密钥。

要使用的注册会根据 TPS 实例全局配置。

2.5.2.4. 令牌数据库

令牌处理系统利用 LDAP 令牌数据库存储，用于存储保留活动令牌及其对应证书的列表，并跟踪每个令牌的当前状态。全新的令牌称为 未初始化，而完全注册的令牌 已注册。此数据存储会持续更新，并在处理令牌时被 TPS 查询。

2.5.2.4.1. 令牌状态和转换

令牌处理系统将状态存储在其内部数据库中，以确定当前的令牌状态以及可以在令牌上执行的操作。

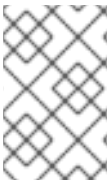
2.5.2.4.1.1. 令牌状态

下表列出了所有可能的令牌状态：

表 2.9. 可能的令牌状态

Name	代码	标签
格式化	0	格式化（未初始化）
DAMAGED	1	物理损坏的
PERM_LOST	2	永久丢失
暂停	3	暂停（临时丢失）
ACTIVE	4	Active
已终止	6	已终止
未格式化的	7	未格式化的

命令行界面使用上面列出的 Name 显示令牌状态。图形界面改为使用 Label。



注意

以上表格不包含代码 5 (之前属于已删除的状态)的状态。

2.5.2.4.1.2. 使用图形或命令行界面的令牌状态转换完成

每个令牌状态的下一个状态有限，它可能会过渡到。例如，令牌可以将状态从 FORMATTED 更改为 ACTIVE 或 DAMAGED，但它不能从 FORMATTED 过渡到 UNFORMATTED。

另外，根据使用命令行或图形界面手动触发转换，或自动使用令牌操作，令牌可以过渡到的状态列表会有所不同。允许的手动转换列表存储在 tokendb.allowedTransitions 属性中，tps.operations.allowedTransitions 属性控制由令牌操作触发的转换。

基于手动和令牌操作的转换的默认配置存储在 /usr/share/pki/tps/conf/CS.cfg 配置文件中。

2.5.2.4.1.2.1. 使用命令行或图形界面的令牌状态转换

在使用 tokendb.allowedTransitions 属性中描述了 TPS 配置文件允许的所有可能转换：

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

属性包含以逗号分隔的转换列表。每个转换都以 < current code> 格式编写： <new code>。这些代码在 表 2.9 “可能的令牌状态” 中进行了描述。默认配置在 /usr/share/pki/tps/conf/CS.cfg 中保留。

下表描述了每个可能的转换信息：

表 2.10. 可能的手动令牌状态转换

转换	当前状态	下一个状态	描述
0:1	格式化	DAMAGED	这个令牌已被物理损坏。
0:2	格式化	PERM_LOST	这个令牌已被永久丢失。
0:3	格式化	暂停	这个令牌已被暂停（临时丢失）。
0:6	格式化	已终止	此令牌已被终止。
3:2	暂停	PERM_LOST	这个暂停的令牌已永久丢失。
3:6	暂停	已终止	这个暂停的令牌已被终止。
4:1	ACTIVE	DAMAGED	这个令牌已被物理损坏。
4:2	ACTIVE	PERM_LOST	这个令牌已被永久丢失。

转换	当前状态	下一个状态	描述
4:3	ACTIVE	暂停	这个令牌已被暂停（临时丢失）。
4:6	ACTIVE	已终止	此令牌已被终止。
6:7	已终止	未格式化的	重复使用此令牌。

以下转换会根据令牌的原始状态自动生成。如果令牌最初是 FORMATTED，然后成为 SUSPENDED，则它只能返回 FORMATTED 状态。如果令牌最初是 ACTIVE，然后变为 SUSPENDED，它只能返回 ACTIVE 状态。

表 2.11. 令牌状态转换自动触发

转换	当前状态	下一个状态	描述
3:0	暂停	格式化	找到这个暂停（临时丢失）令牌。
3:4	暂停	ACTIVE	找到这个暂停（临时丢失）令牌。

2.5.2.4.1.3. 使用令牌操作的令牌状态转换

使用 tokendb.allowedTransitions 属性描述使用令牌操作的所有可能转换：

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

属性包含以逗号分隔的转换列表。每个转换都以 < current code> 格式编写： <new code>。这些代码在 表 2.9 “可能的令牌状态” 中进行了描述。默认配置在 /usr/share/pki/tps/conf/CS.cfg 中保留。

下表描述了每个可能的转换信息：

表 2.12. 使用令牌操作可能的令牌状态转换

转换	当前状态	下一个状态	描述

转换	当前状态	下一个状态	描述
0:0	格式化	格式化	这允许重新格式化令牌或升级令牌中的小程序/密钥。
0:4	格式化	ACTIVE	这允许注册令牌。
4:4	ACTIVE	ACTIVE	这允许重新注册一个活跃的令牌。可能对外部注册很有用。
4:0	ACTIVE	格式化	这允许格式化活跃的令牌。
7:0	未格式化的	格式化	这允许格式化空白或之前使用的令牌。

2.5.2.4.1.4. 令牌状态和转换标签

令牌状态的默认标签和转换存储在 /usr/share/pki/tps/conf/token-states.properties 配置文件中。默认情况下，该文件包含以下内容：

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

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

2.5.2.4.1.5. 自定义允许令牌状态转换

要自定义令牌状态转换列表，请在 /var/lib/pki/instance_name/tps/conf/CS.cfg 中编辑以下属性：

- tokendb.allowedTransitions 用于自定义使用命令行或图形界面执行的允许转换列表
- TPS.operations.allowedTransitions，以使用令牌操作自定义允许的转换列表

如果需要，可以从默认列表中删除转换，但无法添加新转换，除非它们位于默认列表中。默认值存储在 /usr/share/pki/tps/conf/CS.cfg 中。

2.5.2.4.1.6. 自定义令牌状态和转换标签

要自定义令牌状态并转换标签，请将默认的 /usr/share/pki/tps/conf/token-states.properties 复制到您的实例文件夹(/var/lib/pki/instance_name/tps/conf/CS.cfg)，并根据需要更改列出的标签。

更改将立即生效，不需要重新启动服务器。TPS 用户界面可能需要重新加载。

要恢复到默认状态和标签名称，请从 instance 文件夹中删除编辑的 token-states.properties 文件。

2.5.2.4.1.7. 令牌活动日志

某些 TPS 活动记录在日志中。下表中列出了日志文件中可能的事件。

表 2.13. TPS 活动日志事件

活动	描述
add	添加了令牌。
格式	令牌已被格式化。
注册	令牌已注册。
recovery	令牌已被恢复。

活动	描述
续订	令牌已续订。
pin_reset	令牌 PIN 被重置。
token_status_change	使用命令行或图形界面更改令牌状态。
token_modify	修改了令牌。
delete	令牌已删除。
cert_revocation	令牌证书已被撤销。
cert_unrevocation	令牌证书被取消撤销。

2.5.2.4.2. 令牌策略

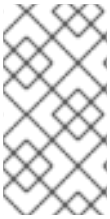
如果进行内部注册，每个令牌都可以由一组令牌策略进行管理。默认策略为：

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

内部注册下的所有 TPS 操作都受到令牌记录中指定的策略。如果没有为令牌指定策略，TPS 将使用默认的策略集合。

2.5.2.5. 映射 Resolver

Mapping Resolver 是 TPS 使用的一种可扩展机制，用于根据可配置的标准确定要分配给特定令牌的配置集。每个映射解析器实例可以在配置中唯一定义，每个操作都可以指向各种定义的映射解析器实例。



注意

映射解析器框架提供了一个用于编写自定义插件的平台。但是，有关如何编写插件的说明超出了本文档的范围。

FilterMappingResolver 是默认由 TPS 提供的唯一映射解析器实现。它允许您为每个 映射 定义一组映射和目标结果。每个映射都包含一组过滤器，其中：

- 如果输入过滤器参数传递映射 中的所有 过滤器，则会分配 目标值。
- 如果输入参数失败，则会跳过该映射，并按顺序尝试下一个映射。
- 如果过滤器没有指定的值，它将始终通过。
- 如果过滤器具有指定的值，则输入参数必须完全匹配。
- 定义映射的顺序非常重要。传递的第一个映射被视为已解析，并返回到调用者。

输入过滤器参数是从带有或没有扩展的智能卡令牌接收的信息。它们会根据上述规则针对 FilterMappingResolver 运行。FilterMappingResolver 支持以下输入过滤器参数：

- appletMajorVersion - 令牌上 Coolkey 小程序的主要版本。
- appletMinorVersion - 令牌上 Coolkey applet 的次要版本。
- keySet or tokenType
  - keySet - 可以设置为客户端请求中的扩展。如果指定了扩展，必须与过滤器中的值匹配。keySet 映射解析器用于在使用外部注册时确定 keySet 值。当支持多个密钥集（例如，不同的智能卡令牌供应商）时，外部注册环境中需要 Key Set Mapping Resolver。在 TKS 中标识 master 密钥需要 keySet 值，这对于建立安全频道至关重要。当用户的 LDAP 记录填充了集合 tokenType (TPS 配置集)时，它不知道哪个卡最终会进行注册，因此 keySet 无法预先确定。keySetMappingResolver 有助于通过允许 keySet 在身份验证前解决 keySet 来解决此问题。
  - tokenType - okenType 可以设置为客户端请求中的扩展。如果指定扩展，则必须匹配过滤器中的值。当前为内部注册环境确定了 tokenType （也称为 TPS Profile）。
- tokenATR - 令牌回答(ATR)。
-

tokenCUID - "start" 和 "end" 定义令牌的卡唯一 ID (CUID)的范围必须回退到传递此过滤器。

2.5.2.6. TPS 角色

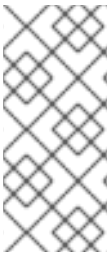
TPS 默认支持以下角色：

- *TPS Administrator - 允许此角色：*
  - *管理 TPS 令牌*
  - *查看 TPS 证书和活动*
  - *管理 TPS 用户和组*
  - *更改常规 TPS 配置*
  - *管理 TPS 验证器和连接器*
  - *配置 TPS 配置集和配置集映射*
  - *配置 TPS 审计日志记录*
- *TPS Agent - 允许此角色：*
  - *配置 TPS 令牌*
  - *查看 TPS 证书和活动*
  - *更改 TPS 配置集的状态*

- TPS Operator - 允许此角色：
  - 查看 TPS 令牌、证书和活动

2.5.3. TKS/TPS 共享 Secret

在 TMS 安装过程中，在令牌密钥服务和令牌处理系统之间建立了一个共享对称密钥。这个键的目的是嵌套和解封会话密钥，这些密钥对于安全频道至关重要。



注意

共享密钥目前仅保存在软件加密数据库中。计划在以后的 Red Hat Certificate System 发行版本中，支持将该密钥保留在硬件安全模块(HSM)设备上。实现此功能后，您将指示您使用 tkstool 将密钥传输到 HSM 来运行关键 Ceremony。

2.5.4. 企业安全客户端(ESC)

企业安全客户端 是一个 HTTP 客户端应用程序，类似于 Web 浏览器，它与 TPS 通信并处理来自客户端侧的智能卡令牌。在 ESC 和 TPS 之间建立 HTTPS 连接，但在每个 TLS 会话中的令牌和 TMS 之间也会建立一个底层的安全频道。

2.6. RED HAT CERTIFICATE SYSTEM SERVICES

证书系统具有很多不同的功能，供管理员使用，这样可以更轻松地维护单个子系统和 PKI 作为一个整体。

2.6.1. 通知

发生特定事件时，如签发或撤销证书时，可以将通知直接发送到指定的电子邮件地址。通知框架附带可以启用和配置的默认模块。

2.6.2. Jobs

自动化作业以定义的间隔运行。

2.6.3. 日志记录

证书系统和每个子系统会生成大量系统和错误日志，这些日志记录系统事件以便监控和调试系统。所有日志记录都存储在本地文件系统中，以便快速检索。日志可以被配置，因此可以为特定类型的事件和所需的日志记录级别创建日志。

证书系统允许在日志归档或分发日志以进行审核之前进行数字签名。此功能允许在签名后检查日志文件进行篡改。

2.6.4. Auditing

证书系统为所有事件维护审计日志，如请求、发布和撤销证书并发布 CRL。这些日志然后被签名。这允许检测到授权访问或活动。然后，外部审核员可以在需要时审核系统。分配的 auditor 用户帐户是唯一能够查看已签名的审计日志的帐户。此用户的证书用于签名和加密日志。审计日志记录配置为指定日志记录的事件。

2.6.5. 自助测试

证书系统为系统自我测试提供了框架，它们在启动时自动运行并可按需运行。证书系统中已包含一组可配置的自测试。

2.6.6. 用户、授权和访问控制

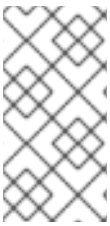
证书系统用户可以分配到组（也称为角色），然后具有他们所属的组的权限。用户仅具有创建用户的子系统实例的特权，以及该用户所属的组的特权。

身份验证 意味着，证书系统子系统用来验证客户端的身份，无论它们是否对证书配置文件或其中一个服务接口进行身份验证。客户端可以通过多种方式执行身份验证，包括简单用户名/密码、SSL/TLS 客户端身份验证、LDAP 身份验证、NIS 身份验证或 CMC。可以为子系统的任何访问权限执行身份验证；例如，配置文件定义请求者如何向 CA 进行身份验证。

在确定并验证客户端后，子系统会执行 授权检查 来确定允许对特定用户的子系统的访问级别。

授权与组或角色关联，权限不直接与单个用户关联。证书系统提供了一个授权框架，用于创建组并为这些组分配访问控制。可以修改预先存在的组的默认访问控制，也可以将访问控制分配给单个用户和 IP 地址。为系统的主部分创建授权访问点，并且可以为每个点设置访问控制规则。

2.6.6.1. 默认管理角色

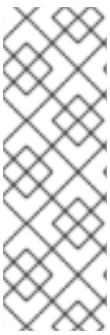


注意

Red Hat Certificate System 在提供给用户的权限上下文中互换使用词语 角色和组。

证书系统默认配置为系统具有不同的访问级别的三种用户类型：

- 管理员可以为 子系统执行任何管理或配置任务。
- 代理，执行 PKI 管理任务，如批准证书请求、管理令牌注册或恢复密钥。
- 审核员，可以查看和配置审计日志。



注意

默认情况下，为了 bootstrap，在运行 pkispawn 工具时，在 Red Hat Certificate System 实例创建过程中创建管理用户处理管理员和代理特权。此 bootstrap 管理员默认使用 caadmin 用户名，但可以被传递给 pkispawn 命令的配置文件中的 pki_admin_uid 参数覆盖。bootstrap 管理员的目的是创建第一个管理员和代理用户。此操作需要管理员特权来优化用户和组，以及代理特权来发布证书。

2.6.6.2. 内置子系统信任角色

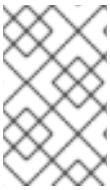
此外，创建安全域时，托管域的 CA 子系统会自动被授予 Security Domain Administrator 的特殊角色，这使得子系统能够管理安全域及其中的子系统实例。可以为不同的子系统实例创建其他安全域管理员角色。这些特殊角色不应具有实际的用户作为成员。

2.7. 克隆

2.7.1. 关于 Cloning

通过提供一个或多个子系统克隆，规划 高可用性 可减少计划外中断和其他问题。当主机停机时，克隆的子系统可以处理请求和执行服务，同时从主（原始）子系统无缝处理并保持不间断的服务。

使用克隆的子系统还允许系统离线进行修复、故障排除或其他管理任务，而不中断整体 PKI 系统的服务。



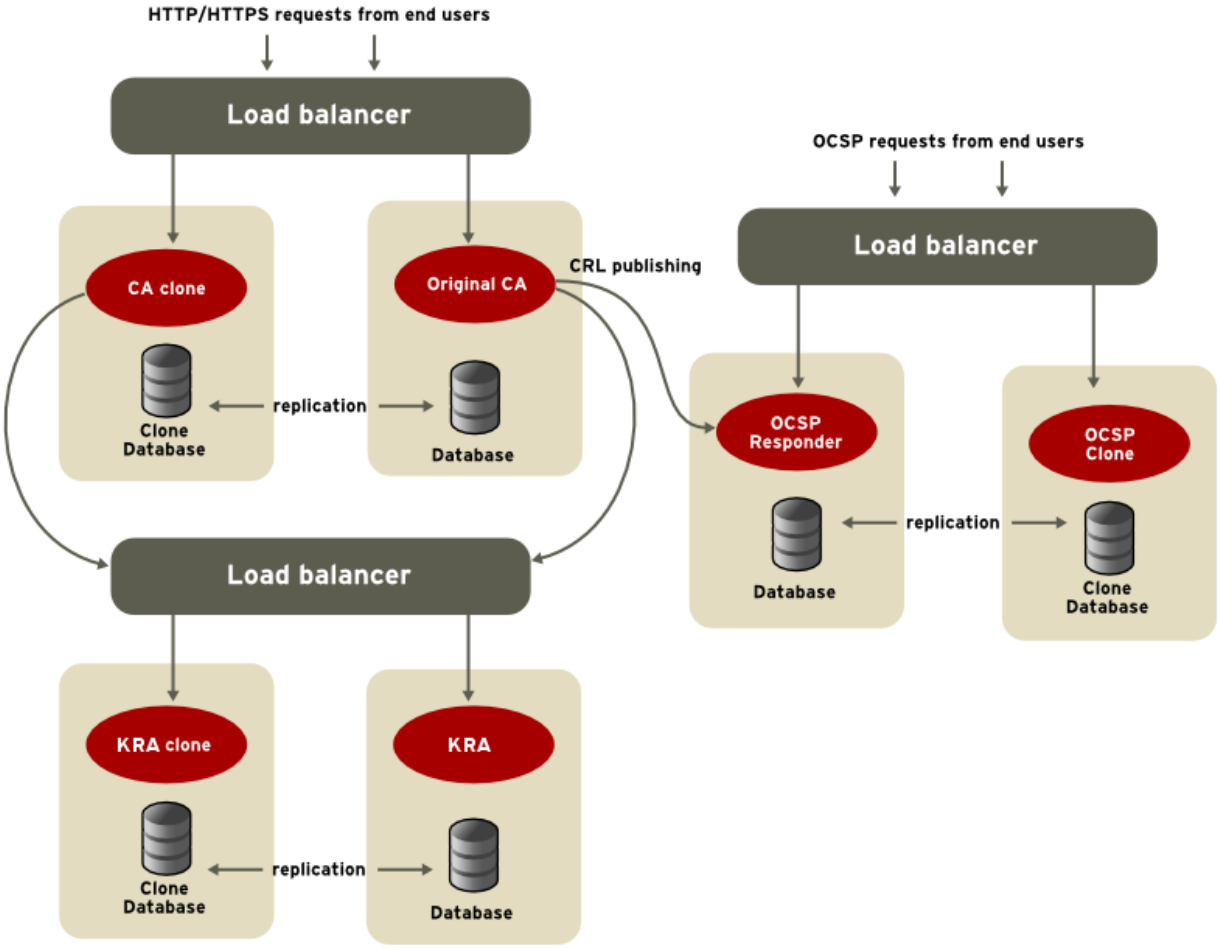
注意

除可克隆 TPS 外的所有子系统。

克隆是一种为 PKI 提供可扩展性的方法，方法是将相同的任务（如处理证书请求）分配给不同计算机上的单独的实例。master 及其克隆的内部数据库会相互复制，因此一个子系统上的证书请求或存档密钥的信息可在所有其他子系统上可用。

通常，master 和克隆的实例安装在不同的计算机上，这些计算机则放置在 负载均衡器后面。负载平衡器接受对证书系统子系统发出的 HTTP 和 HTTPS 请求，并在主控和克隆的实例之间正确定向这些请求。如果一台机器失败，负载均衡器会透明地将所有请求重定向到仍在运行的机器，直到其他机器重新上线为止。

图 2.5. 克隆示例

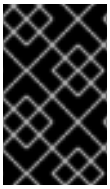


证书系统子系统前面的负载均衡器是在高可用性系统中提供实际故障转移支持。负载均衡器也可以作为证书系统子系统的一部分提供：

- *DNS 循环(round-robin)，一种用于管理在多个不同服务器间分布负载的网络拥塞功能。*
- *粘性 SSL/TLS，这使得返回到系统的用户能够路由之前使用的相同主机。*

通常，当克隆系统时，配置 servlet 会在内部 LDAP 数据库之间设置复制协议。但是，有些用户可能希望建立和管理自己的复制协议。pkispawn 可执行文件已被修改，通过在 PKI 实例覆盖配置文件中添加 [Tomcat] 部分并在该部分中添加以下 name=value 对：

```
[Tomcat]
pki_clone_setup_replication=False
pki_clone_reindex_data=False
```



重要

在指定这类安装时，必须先复制数据，然后才能 调用 pkispawn。

2.7.2. 准备克隆

您需要将证书导出到 p12 文件，然后在安装克隆子系统时使用该文件。此命令不安装或设置克隆子系统，而是为安装准备它。以下是如何使用 pki-server SUBSYSTEM-clone-prepare 命令导出子系统证书的示例。

例 2.3. 导出子系统证书

```
[root@pki1 ~]$ pki-server ca-clone-prepare --i topology-02-CA --pkcs12-file
/tmp/caclone.p12 --pkcs12-password SECret.123
----------------------------------------------------
Added certificate "subsystemCert cert-topology-02-CA"
----------------------------------------------------
----------------------------------------------------
Added certificate "caSigningCert cert-topology-02-CA CA"
----------------------------------------------------
----------------------------------------------------
Added certificate "ocspSigningCert cert-topology-02-CA CA"
----------------------------------------------------
----------------------------------------------------
Added certificate "auditSigningCert cert-topology-02-CA CA"
----------------------------------------------------
```

您可以验证导出的 p12 文件是否包含证书。在以下示例中，pki pkcs12-cert-find 的输出返回四个证书：

```
[root@pki1 ~]$ pki pkcs12-cert-find --pkcs12-file /tmp/caclone.p12 --pkcs12-password
SECret.123
---------------
4 entries found
---------------
Certificate ID: 4649cef11b90c78d126874b91654de98ded54073
Serial Number: 0x4
Nickname: subsystemCert cert-topology-02-CA
Subject DN: CN=Subsystem Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
Issuer DN: CN=CA Signing Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
Trust Flags: u,u,u
Has Key: true

Certificate ID: a304cf107abd79fbda06d887cd279fb02cefe438
Serial Number: 0x1
Nickname: caSigningCert cert-topology-02-CA CA
Subject DN: CN=CA Signing Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
```

```
Issuer DN: CN=CA Signing Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
Trust Flags: CTu,Cu,Cu
Has Key: true

Certificate ID: 4a09e057c1edfee40f412551db1959831abe117d
Serial Number: 0x2
Nickname: ocspSigningCert cert-topology-02-CA CA
Subject DN: CN=CA OCSP Signing Certificate,OU=topology-02-CA,O=topology-
02_Foobarmaster.org
Issuer DN: CN=CA Signing Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
Trust Flags: u,u,u
Has Key: true

Certificate ID: 3f42f88026267f90f59631d38805cc60ee4c711a
Serial Number: 0x5
Nickname: auditSigningCert cert-topology-02-CA CA
Subject DN: CN=CA Audit Signing Certificate,OU=topology-02-CA,O=topology-
02_Foobarmaster.org
Issuer DN: CN=CA Signing Certificate,OU=topology-02-CA,O=topology-02_Foobarmaster.org
Trust Flags: u,u,Pu
Has Key: true
```

使用导出的 p12 文件，您现在可以设置克隆子系统。

2.7.3. 为 CA 克隆

克隆的实例具有与 master 相同的私钥，因此其证书是相同的。对于 CA，这意味着 CA 签名证书与原始 master CA 及其克隆的 CA 相同。从客户端的角度来看，它们看起来像一个 CA。

每个克隆的 CA 和 master 都可能会发布证书并处理撤销请求。

管理复制 CA 的主要问题是如何为它们发布的证书分配序列号。不同的复制 CA 可以有不同的流量级别，使用不同速率的序列号，并且没有两个复制 CA 签发具有相同序列号的证书。这些序列号范围通过使用共享来动态分配和管理，复制条目定义每个 CA 的范围和下一个可用范围，以便在一个 CA 范围运行较低时重新分配。

克隆的 CA 的序列号范围是 fluid。所有复制 CA 共享一个通用配置条目，用于定义下一个可用范围。当一个 CA 在可用数字上开始运行较低时，它会检查此配置条目并声明下一个范围。该条目会自动更新，以便下一个 CA 获取新范围。

范围在 begin*Number 和 end*Number 属性中定义，为请求和证书序列号定义单独的范围。例如：

```
dbs.beginRequestNumber=1
dbs.beginSerialNumber=1
```

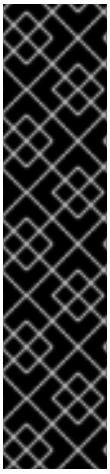
```
dbs.enableSerialManagement=true
dbs.endRequestNumber=9980000
dbs.endSerialNumber=ffe0000
dbs.replicaCloneTransferNumber=5
```

只有一个复制 CA 才能生成、缓存和发布 CRL；这是 CRL CA。提交到其他复制 CA 的 CRL 请求会立即重定向到 CRL CA。虽然其他复制 CA 可以撤销、显示、导入和下载之前由 CRL CA 生成的 CRL，但如果生成了新的 CRL，则可能会出现同步问题。有关如何将 CRL 生成移到不同的复制 CA 的说明，请参阅 第 10.7.1 节 “转换 CA 克隆和 Master”。

主 CA 还通过监控对内部数据库的复制更改来管理克隆的 CA 之间的关系和信息共享。

注意

如果克隆了安全域 master 的 CA，则该克隆的 CA 也是安全域 master。在这种情况下，原始 CA 及其克隆都共享相同的安全域配置。



重要

如 第 2.7.9 节 “自定义配置和克隆” 中所述，LDAP 数据库中的数据会在主控机和克隆之间复制，但不会复制不同实例 的配置文件。这意味着，如果在克隆创建后发生更改，则影响 CS.cfg 文件的任何更改（如添加 KRA 连接或创建自定义配置集）都不会复制到克隆的配置中。

在创建克隆 前，对 CA 配置的任何更改都应进行（因此自定义更改包含在克隆过程中），或者必须在创建后手动复制任何配置更改到克隆实例。

2.7.4. 克隆 KRA

使用 KRA，一个 KRA 中存档的所有密钥都会复制到其他 KRA 的内部数据库。这允许在任何克隆 KRA 上启动密钥恢复，无论最初存档密钥的 KRA。

处理密钥恢复后，恢复的记录将存储在所有克隆的 KRA 的内部数据库中。

通过同步密钥恢复，虽然可以在任何克隆上启动恢复过程，但它必须在启动它的同一单一 KRA 上完成。这是因为在从 KRA 代理获取适当数量的批准后，才会在复制数据库中记录恢复操作。在此之前，启动恢复的 KRA 是唯一知道恢复操作的 KRA。

重要

存在一个在 Red Hat Certificate System 9 中弃用的同步密钥恢复机制，它不在本文档中讨论。红帽建议使用异步密钥恢复。

2.7.5. 克隆其他子系统

复制的 TKS 之间没有实际操作差异；在单个服务器上创建或维护的信息将与其他服务器一起复制。

对于 OCSP，只有一个复制的 OCSP 接收 CRL 更新，然后发布的 CRL 复制到克隆。

2.7.6. 克隆和密钥存储

克隆子系统创建两个执行相同功能的服务器进程：创建子系统的另一个新实例，并配置为使用相同密钥和证书来执行其操作。根据为主克隆存储密钥的位置，克隆访问密钥的方法非常不同。

如果密钥和证书存储在内部软件令牌中，那么首次配置时必须从主子系统导出它们。在配置 master 实例时，可以通过在 pkispawn 配置文件中指定 pki_backup_keys 和 pki_backup_password 参数，将系统密钥和证书备份到 PKCS detailed 文件。详情请查看 pki_default.cfg(5) man page 中的 BACKUP PARAMETERS 部分。

如果在初始配置过程中没有备份密钥，您可以使用 PKCS12Export 工具将其提取到 PKCSall 文件中，如 第 10.1 节 “从软件数据库备份子系统密钥” 所述。

然后，将 PKCSbusybox 文件复制到克隆子系统，并使用 pki_clone_pkcs12_ password 和 pki_clone_pkcs12 _path 参数在 pkispawn 配置文件中定义其位置和密码，请参阅 pkispawn(8) man page 中的 安装克隆 部分。特别是，请确保 pkiuser 用户可以访问 PKCScriu 文件，并且它具有正确的 SELinux 标签。

如果密钥和证书存储在硬件令牌中，则必须使用硬件令牌特定实用程序复制密钥和证书，或者直接在令牌中引用：

- 复制所有必需的密钥和证书，但 SSL/TLS 服务器密钥和证书除外。保留这些证书的 nickname。此外，将 master 实例中的所有必要可信根证书复制到克隆实例，如链或交叉对证书。

- 如果令牌基于网络，那么令牌只需要使用密钥和证书；不需要复制密钥和证书。

- 在使用基于网络的硬件令牌时，请确保在硬件令牌上启用高可用性功能，以避免出现单点故障。

2.7.7. LDAP 和端口注意事项

如 第 2.7.1 节 “关于 Cloning” 所述，克隆行为的一部分是在复制子系统之间复制信息，以便它们处理同一组数据和记录。这意味着复制子系统的 LDAP 服务器需要能够进行通信。

如果 Directory 服务器实例位于不同的主机上，请确保有适当的防火墙访问权限来允许目录服务器实例相互连接。

注意

克隆的子系统及其 master 必须使用单独的 LDAP 服务器，但它们在通用后缀之间复制数据。

子系统可以通过 LDAPS 端口或通过 LDAP 端口的标准连接，使用 SSL/TLS 连接到其内部数据库。克隆子系统时，克隆实例使用与 master 连接数据库相同的连接方法(SSL/TLS 或标准)。通过克隆，还需要额外的数据库连接：主目录服务器数据库到克隆目录服务器数据库。对于那个连接，有三个 连接选项：

- 如果 master 使用 SSL/TLS 连接到其数据库，则克隆使用 SSL/TLS，并且 master/clone 目录服务器数据库使用 SSL/TLS 连接进行复制。

- 如果 master 使用标准连接到其数据库，则克隆必须使用标准连接，而目录服务器数据库 可以使用 未加密的连接进行复制。

- 如果 master 使用标准连接到其数据库，则克隆必须使用标准连接，但 可以选择将 Start TLS 用于 master/clone Directory Server 数据库进行复制。启动 TLS，通过标准端口打开安全连接。

注意

要使用 Start TLS，目录服务器仍必须配置为接受 SSL/TLS 连接。这意味着，在配置克隆前，必须在 Directory Server 上安装服务器证书和 CA 证书，且必须启用 SSL/TLS。

master 使用的任何连接方法（安全或标准）都必须由克隆使用，且必须在配置克隆前为目录服务器数据库正确配置。

重要

即使克隆通过安全连接连接到 master，标准 LDAP 端口（默认为 389）必须在 LDAP 服务器上打开并启用，同时配置克隆。

对于安全环境，在配置了克隆后，可以在主目录服务器实例上禁用标准 LDAP 端口。

2.7.8. 副本 ID 号

克隆基于为 master 实例在 Directory 服务器之间设置复制协议，以及用于克隆的实例的目录服务器。

涉及复制的服务器位于同一 复制拓扑 中。每次克隆子系统实例时，都会将其添加到整个拓扑中。目录服务器根据 副本 ID 号，在拓扑中的不同服务器间识别不同的服务器。这个副本 ID 在拓扑中的所有服务器中必须是唯一的。

与用于请求和证书的序列号范围一样（在 第 2.7.3 节 “为 CA 克隆”中覆盖），每个子系统都会被分配一个允许的副本 ID 范围。克隆子系统时，它会将一个副本 ID 从其范围分配给新的克隆实例。

```
dbs.beginReplicaNumber=1
dbs.endReplicaNumber=95
```

如果实例开始耗尽其当前范围，则可以使用新数字刷新副本 ID 范围。

2.7.9. 自定义配置和克隆

创建克隆后，克隆之间或主控机和克隆之间不会复制 配置更改。实例配置位于复制的数据库外的 CS.cfg 文件中。

例如，有两个 CA，一个 master 和一个克隆。安装了一个新的 KRA，它将在其配置上与主 CA 一起关联。CA-KRA 连接器信息存储在 master CA 的 CS.cfg 文件中，但不会将这个连接器信息添加到克隆 CA 配置中。如果包含密钥归档的证书请求提交到 master CA，则使用 CA-KRA 连接器信息将密钥归档转发到 KRA。如果请求提交到克隆 CA，则不会识别 KRA，并且禁止密钥存档请求。

对主服务器配置或克隆服务器所做的更改不会复制到其他克隆的实例。任何关键设置都必须手动添加到克隆。

注意

在配置任何克隆前，您可以为 master 服务器设置所有必需的自定义配置。例如，安装所有 KRA，以便所有连接器信息都位于主 CA 配置文件中，创建任何自定义配置集，或者为 master OCSP 响应程序配置所有发布点。请注意，如果 LDAP 配置文件存储在 Directory 服务器中，则它们会被复制并跨服务器保持同步。

master 实例中的任何自定义设置都会在克隆实例时包含在克隆的实例中（但不在之后）。

第 3 章 支持的标准和协议

Red Hat Certificate System 基于许多公共和标准协议和 RFC，以确保实现最佳性能和互操作性。本章介绍了证书系统 10 使用或支持的主要标准和协议，以帮助管理员有效地规划其客户端服务。

3.1. TLS、ECC 和 RSA

传输层安全性(TLS)协议是客户端和服务器之间验证和加密的通信的通用标准。客户端和服务器身份验证都通过 TLS 进行。

TLS 使用公钥和对称密钥加密的组合。对称密钥加密比公钥加密快得多，但公钥加密提供了更好的身份验证技术。TLS 会话始终从名为 handshake 的消息交换、服务器和客户端之间的初始通信开始。握手允许服务器使用公钥技术向客户端进行身份验证（可选）允许客户端向服务器进行身份验证，然后允许客户端和服务器在创建用于快速加密、解密和完整性的会话期间创建用于快速加密、解密和完整性验证的对称密钥。

TLS 支持各种不同的加密算法 或密码，用于验证服务器和客户端、传输证书和建立会话密钥等操作。客户端和服务器可能支持不同的密码套件或一组密码。在其他功能中，握手还决定服务器和客户端如何协商使用哪些密码套件相互验证、传输证书并建立会话密钥。

密钥交换算法（如 RSA 和 Elliptic Curve Diffie-Hellman (ECDH)）管理服务器和客户端在 TLS 会话中使用的对称密钥的方式。TLS 支持 ECC (Elliptic Curve Curve Cryptography)密码套件和 RSA。证书系统原生支持 RSA 和 ECC 公钥加密系统。

在最新的 practise 中，密钥交换算法被 key-agreement 协议 取代，其中两个或更多方各自在为安全通信建立通用密钥时影响结果。密钥协议优先于密钥交换，因为它允许实施 Perfect Forward Secrecy (PFS)。当使用 PFS 时，会为密钥协议使用非确定算法为每个会话生成随机公钥（也称为临时密码参数或临时密钥）。因此，没有单个 secret 值可能会导致多个消息受到破坏，从而保护过去和将来的通信。

注意

随着计算功能增加，需要较长的 RSA 密钥来提供安全性。推荐的 RSA 密钥长度为 2048 位。虽然许多服务器继续使用 1024 位密钥，但服务器应至少迁移到 2048 位。对于 64 位机器，请考虑使用更强大的密钥。所有 CA 应该至少使用 2048 位密钥，以及更强大的密钥（如 3072 或 4096 位）。

3.1.1. 支持的加密套件

密码和哈希算法与各种漏洞和安全强度一致。作为常规规则，Red Hat Certificate System 遵循 NIST

指南，并支持与服务器密钥相关的 TLS 1.1 和 TLS 1.2 密码套件。

3.1.1.1. 推荐的 TLS 密码套件

传输层安全性(TLS)协议是客户端和服务器之间验证和加密的通信的通用标准。Red Hat Certificate System 支持 TLS 1.1 和 1.2。

当服务器充当服务器或客户端时，Red Hat Certificate System 支持以下密码套件：

ECC

- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

RSA

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA

- *TLS_DHE_RSA_WITH_AES_256_CBC_SHA*
- *TLS_DHE_RSA_WITH_AES_128_CBC_SHA256*
- *TLS_DHE_RSA_WITH_AES_256_CBC_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384*
- *TLS_DHE_RSA_WITH_AES_128_GCM_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*
- *TLS_DHE_RSA_WITH_AES_256_GCM_SHA384*
- *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384*

3.2. 允许的密钥算法及其大小

红帽证书系统支持以下密钥算法和大小（如果它们由底层 PKCS facilities 模块提供）。

- *允许的 RSA 密钥大小：*
  - *2048 位或更高*
- *允许 EC curves 或等同于 FIPS PUB 186-4 标准中定义的：*

- *nistp256*
- *nistp384*
- *nistp521*

3.3. 允许的哈希功能

允许以下密钥哈希消息身份验证(HMAC)：

- *SHA-256*
- *SHA-384*
- *SHA-512*

允许以下加密哈希功能：

- *SHA-256*
- *SHA-384*
- *SHA-512*

3.4. IPV4 和 IPV6 地址

证书系统支持 IPv4 地址和 IPv6 地址。在非常广泛的情况下，证书系统子系统或操作引用主机名或 IP 地址；支持 IPv4 和 IPv6 样式的地址可确保与网络协议兼容。支持 IPv6 连接的操作包括：

- 子系统之间的通信，包括 TPS、TKS 和 CA 之间的通信，并加入安全域
- TPS 和企业安全客户端之间的令牌操作
- 子系统日志记录
- 访问控制指令
- 使用证书系统工具（包括 pki 工具、Subject Alt Name Extension 工具、HttpClient 和 Bulk Issuance 工具）执行的操作
- 客户端通信，包括 pkiconsole 工具和启用 IPv6 的浏览器用于 Web 服务
- 证书请求名称和证书主题名称，包括用户、服务器和路由器证书
- 发布
- 连接到内部数据库和身份验证目录的 LDAP 数据库

每当引用主机名或 URL 时，都可以使用 IP 地址：

- IPv4 地址的格式必须是 n.n.n.n 或 n. n.n,m.m.m.m。例如，128.21.39.40 或 128.21.39.40,255.255.255.00。
- IPv6 地址使用 128 位命名空间，其 IPv6 地址用冒号和以句点分隔的子网掩码。例如：0:0:0:0:0:0:13.1.68.3,FF01::43, 或 0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0.

如果正确配置了 DNS，则可以使用 IPv4 或 IPv6 地址连接到 Web 服务页面和子系统 Java 控制台。最常见的方法是使用完全限定域名：

```
https://ipv6host.example.com:8443/ca/services
pkiconsole https://ipv6host.example.com:8443/ca
```

要使用 IPv6 数字地址，请将 URL 中的完全限定域名替换为 IPv6 地址，并用括号括起来([])。例如：

```
https://[00:00:00:00:123:456:789:00:]:8443/ca/services
pkiconsole https://[00:00:00:00:123:456:789:00:]:8443/ca
```

3.5. 支持的 PKIX 格式和协议

证书系统支持 IETF 在公钥基础架构(X.509)中定义的许多协议和格式。除了此处列出的 PKIX 标准外，在 IETF Datatracker 网站上还提供其他 PKIX 列出的标准。

表 3.1. 证书系统 10 中支持的 PKIX 标准

格式或协议	RFC 或 Draft	描述
X.509 版本 1 和版本 3		International Telecommunications Union (ITU)推荐的数字证书格式。
证书请求消息格式(CRMF)	RFC 4211	将证书请求发送到 CA 的消息格式。
证书管理消息格式(CMMF)		消息格式，将来自端点实体的证书请求和撤销请求发送到 CA，并将信息返回到结束实体。CMMF 已被另一个标准 CMC 使用。
通过 CS (CMC)的证书管理消息	RFC 5274	基于 CS 和 PKCS #10 的公共密钥认证产品的通用接口，包括使用 Diffie-Hellman public-keys 的 RSA 签名证书的证书注册协议。CMC 包含 CRMF 和 CMMF。
加密消息语法(CMS)	RFC 2630	用于数字签名和加密的 PKCS #7 语法的超集。
PKIX 证书和 CRL 配置文件	RFC 5280	由 IETF 为互联网的公共密钥基础架构开发的标准。它为证书指定配置集和 CRL。
在线证书状态协议(OCSP)	RFC 6960	此协议可用于确定数字证书的当前状态，而无需 CRL。

第 4 章 支持的平台

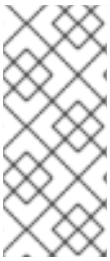
这部分论述了 Red Hat Certificate System 10 支持的不同服务器平台、硬件、令牌和软件。

4.1. 常规要求

详情请查看 Red Hat Certificate System 10 发行注记中的相应部分。

4.2. 服务器支持

在 Red Hat Enterprise Linux 8.6 及更高版本中，支持运行证书颁发机构(CA)、密钥恢复授权机构(KRA)、在线证书状态协议(OCSP)、令牌密钥服务(TKS)和令牌处理系统(TPS)子系统。支持的目录服务器版本为 11.1 及更新的版本。



注意

在认证的 hypervisor 上的 Red Hat Enterprise Linux 8.6 虚拟客户机中运行证书系统 10.4。详情请查看 哪个 hypervisor 经过认证以运行 Red Hat Enterprise Linux？ 解决方案文档。

4.3. 支持的 WEB 浏览器

证书系统 10.0 支持以下浏览器：

表 4.1. 平台支持的 Web 浏览器

平台	代理服务	最终用户页面
Red Hat Enterprise Linux	Firefox 60 及更新的版本 [a]	Firefox 60 及更新的版本 [a]

[a] 此 Firefox 版本不再支持用于从浏览器生成和归档密钥的加密 Web 对象。因此，预期在这个区域中具有有限的功能。

注意

基于 HTML 的实例配置的唯一完全支持的浏览器是 Mozilla Firefox。

4.4. 支持的硬件安全模块

下表列出了红帽认证系统支持的硬件安全模块(HSM)：

HSM	固件	设备软件	客户端软件
nCipher nShield Connect XC (High)	nShield_HSM_Firmware-12.72.1	12.71.0	SecWorld_Lin64-12.71.0
Thales TCT Luna Network HSM Luna-T7	lunafw_update-7.11.1-4	7.11.0-25	610-500244-001_LunaClient-7.11.1-5

第 5 章 规划证书系统

每个 Red Hat Certificate System 子系统可以安装在同一服务器计算机上，安装在独立的服务器上，或者将多个实例安装到一个机构中。在安装任何子系统之前，务必要规划部署：您需要的 PKI 服务？网络要求是什么？哪些人需要访问证书系统、角色是什么及其物理位置？您需要发布哪些类型的证书，以及为它们设置哪些限制或规则？

本章介绍了规划证书系统部署的一些基本问题。其中很多决策是相互相关的；一个选择会影响另一个选择，例如决定是否使用智能卡来确定是否安装 TPS 和 TKS 子系统。

5.1. 决定所需的子系统

证书系统子系统涵盖了管理证书的不同方面。规划要安装的子系统是定义部署需要执行的 PKI 操作的一种方式。

证书（如软件或设备）具有定义阶段的生命周期。最基本的是，有三个步骤：

- 它被请求并发布。
- 它有效。
- 它过期。

但是，这个简化的场景不包括与证书相关的很多常见问题：

- 如果员工在证书过期前离开公司怎么办？
- 当 CA 签名证书过期时，使用该证书发布并签名的证书也会过期。那么将续订 CA 签名证书，允许其发布的证书保持有效，或者将重新发布？
- 如果员工丢失了智能卡或将其留在家，则怎么办。使用原始证书密钥发布替换证书吗？其他证书是否被暂停或撤销？允许临时证书？

- 当证书过期时，将发布新证书，或将续订原始证书？

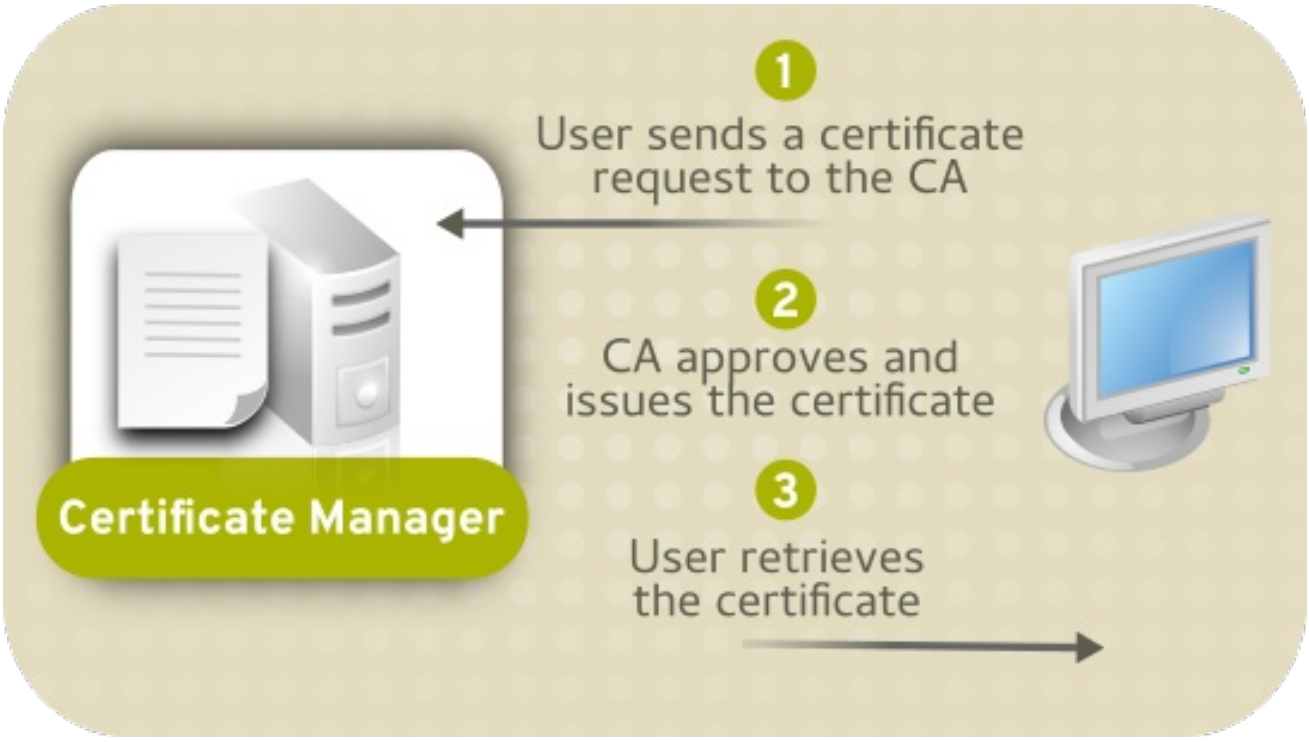
这介绍了三个管理证书的注意事项：撤销、续订和替换。

其他注意事项是在证书颁发机构上加载的。有很多颁发或续订请求？是否有很多来自客户端的流量来验证证书是否有效？请求证书的人员应该如何验证其身份，这个过程是否减慢了问题过程？

5.1.1. 使用单一证书管理器

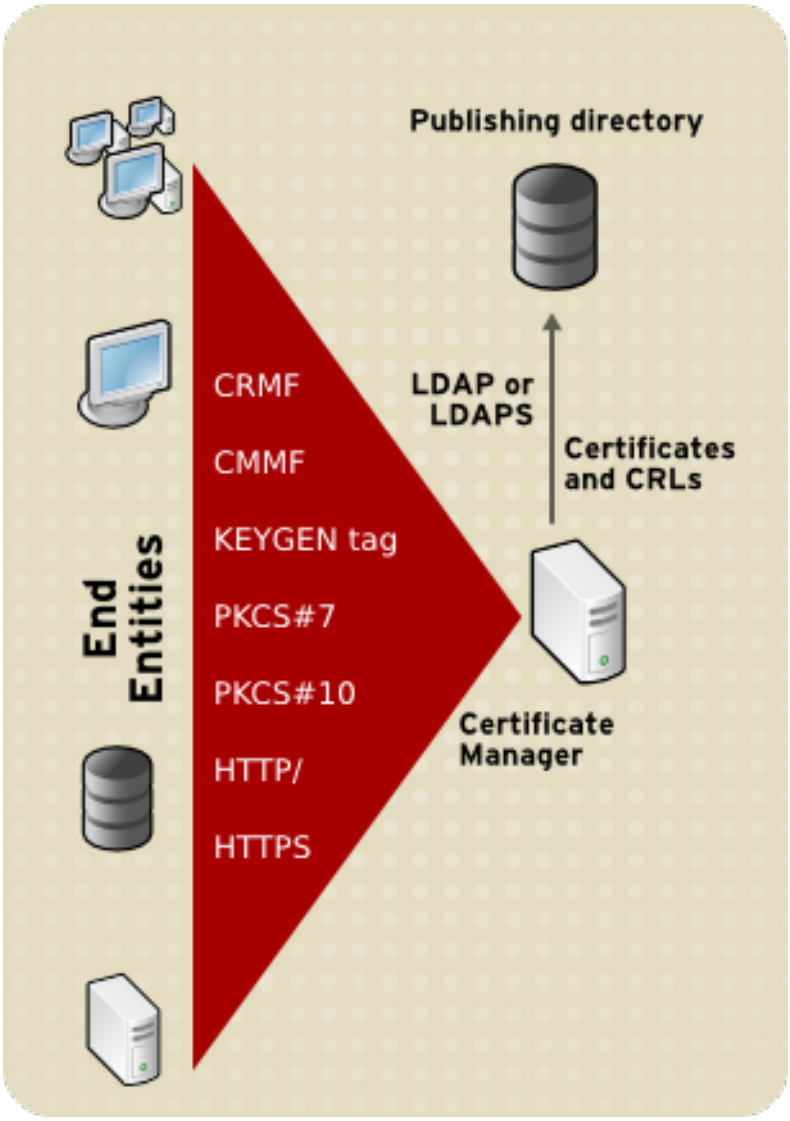
证书系统 PKI 的核心是证书管理器（证书颁发机构）。CA 接收证书请求并发布所有证书。

图 5.1. 仅限 CA 的证书系统



请求和发布证书的所有基本处理都可以由证书管理器处理，它是唯一的子系统。根据机构的需求，可以有单个证书管理器或多个证书管理器安排许多不同的关系。

除了发布证书外，证书管理器也可以撤销证书。管理员的一个问题是，如果证书丢失、被破坏，或者他们要发出的人员或设备离开公司时，如何处理证书。撤销证书在过期日期之前无效，并且已撤销的证书列表由发出 CA 编译并发布，以便客户端能够验证证书状态。



5.1.2. 为 Lost 密钥计划：密钥归档和恢复

虽然 CA 无法执行一个操作，但是密钥归档和恢复。一个非常真实的场景是，用户将丢失其私钥 - 例如，可以从浏览器数据库中删除密钥，或者智能卡可能会保留在家。许多常见业务操作使用加密数据，如加密电子邮件，并丢失解锁数据的密钥意味着数据本身会丢失。根据公司的策略，可能需要恢复密钥以重新生成或重新导入替换证书，并且两个操作都需要私钥。

这需要一个 KRA，该子系统专门存档和检索密钥。

图 5.2. CA 和 KRA

Key Recovery Authority 存储加密密钥（密钥归档），并可以检索这些密钥，以便 CA 可以重新发布证书（密钥恢复）。KRA 可以为证书系统生成的任何证书存储密钥，无论是针对常规证书还是注册智能卡。

第 2.4.5 节 “归档、恢复和轮转密钥” 中详细介绍了密钥归档和恢复过程。

注意

KRA 仅用于归档和恢复私钥。因此，最终用户必须使用支持双密钥生成的浏览器来存储其公钥对。

5.1.3. 平衡证书请求处理

子系统如何协同工作的另一个方面是负载平衡。如果站点具有高的流量，那么可以简单地安装很多 CA，作为彼此克隆，或者以扁平层次结构（每个 CA 独立）或树层次结构中（其中某些 CA 从属到其他 CA）；这在 第 5.2 节 “定义证书颁发机构层次结构” 中进行了进一步介绍。

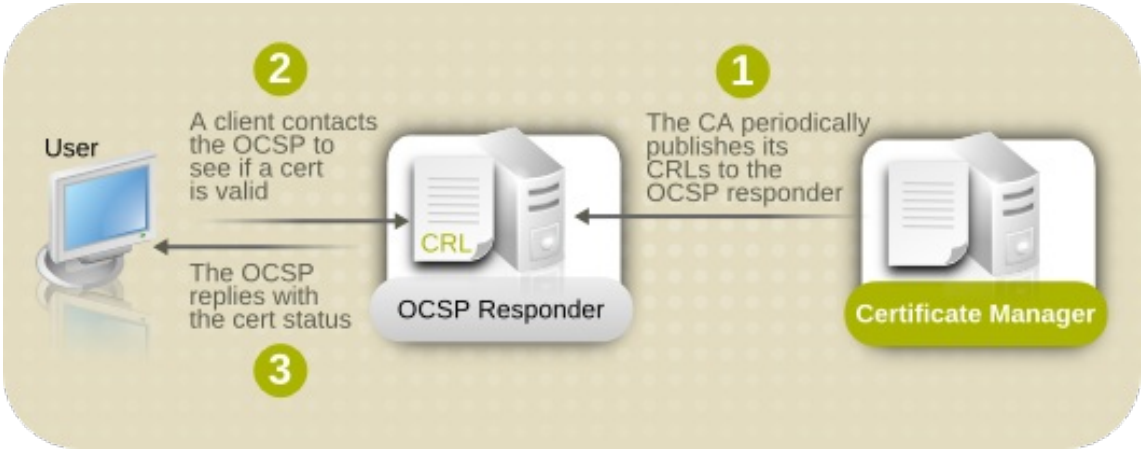
5.1.4. 平衡客户端 OCSP 请求

如果证书在有效期内，但需要无效，则可以撤销证书。证书管理器可以发布已撤销证书的列表，以便在客户端需要验证证书是否仍然有效时，它可以检查列表。这些请求 是在线证书状态协议请求，这意味着

它们具有特定请求和响应格式。证书管理器有一个内置 OCSP 响应程序，以便它可以自行验证 OCSP 请求。

但是，与证书请求流量一样，站点可能会具有大量客户端请求来验证证书状态。示例 Corp. 具有大型 Web 存储，每个客户的浏览器都试图验证其 SSL/TLS 证书的有效性。同样，CA 可以处理发布证书数量，但高请求流量会影响到其性能。在这种情况下，Example Corp. 使用外部 OCSP Manager 子系统来验证证书状态，证书管理器只需要每次都发布更新的 CRL。

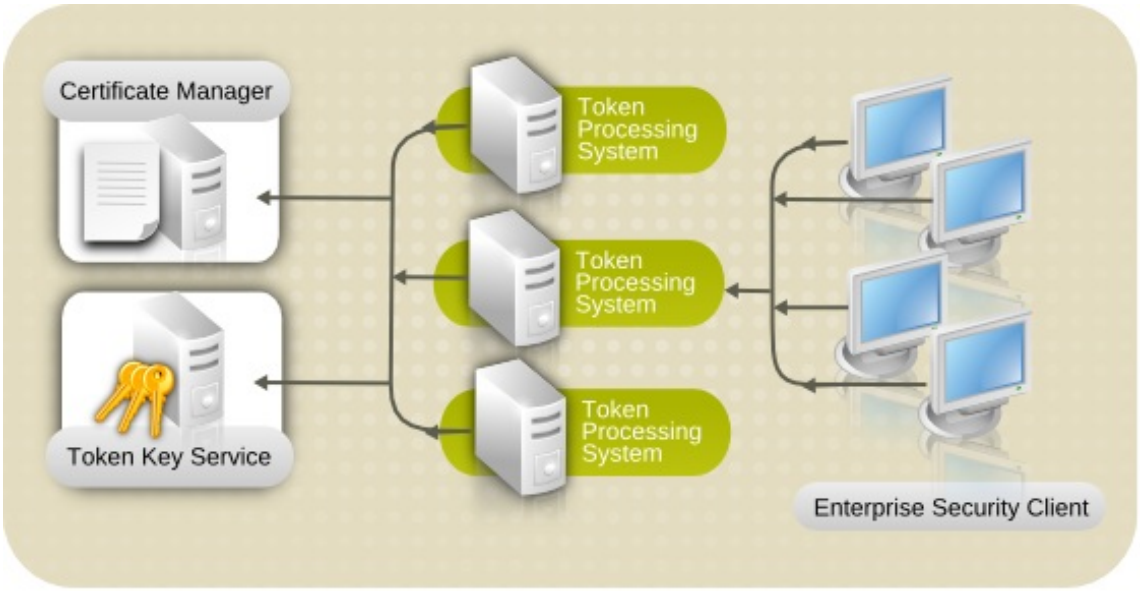
图 5.3. CA 和 OCSP



5.1.5. 使用智能卡

智能卡通常需要个人注册和批准流程，因为它们使用物理介质存储密钥和证书数据。这意味着多个代理需要访问不同的办公室或地理位置的 TPS 子系统。

令牌管理系统非常可扩展。可将多个 TPS 配置为处理单个 CA、TKS 或 KRA 实例，而多个企业安全客户端可以与单个 TPS 通信。随着其他客户端被安装，他们可以重新指向 TPS 实例，而无需重新配置 TPS；同样，由于添加了 TPS，他们可以指向同一 CA、TKS 和 KRA 实例，而无需重新配置这些子系统。



安装后，可以编辑 TPS 配置以使用额外的 CA、KRA 和 TKS 实例来实现故障转移支持，因此如果主子系统不可用，则 TPS 可以在不中断其令牌服务的情况下切换到下一个可用的系统。

5.2. 定义证书颁发机构层次结构

CA 是 PKI 中心，因此 CA 系统与彼此的关系(CA 层次结构)和其他子系统（安全域）对于规划证书系统 PKI 至关重要。

当 PKI 中有多个 CA 时，CA 以层次结构或链中的结构。链中另一个 CA 上面的 CA 称为 根 CA ；链中另一个 CA 的 CA 称为 从属 CA。CA 也可以从属到证书系统部署之外的 root；例如，作为证书系统部署中的 root CA 的 CA 可以从属到第三方 CA。

证书管理器（或 CA）与另一个 CA 从属，因为其 CA 签名证书（允许它的证书由另一个 CA 发布）。发布从属 CA 签名证书的 CA 通过 CA 签名证书的内容控制 CA。CA 可以通过它可以发布的证书类型限制下级 CA 的限制、允许包含在证书中的扩展、从属 CA 的级别数量，以及它可能发布的证书的有效性周期，以及子 CA 签名证书的有效性周期。

注意

虽然从属 CA 可以创建违反这些限制的证书，但验证违反了这些限制的证书的客户端将不接受该证书。

自签名 root CA 为自己的 CA 签名证书签名，并根据它发布的子 CA 签名证书设置限制。

证书管理器可以配置为 root CA 或从属 CA。最简单的方法是使第一个 CA 安装自签名 root，因此不需要应用到第三方并等待证书发布。但是，在部署完整 PKI 之前，请考虑是否具有 root CA、要拥有多少个，以及根和从属 CA 的位置。

5.2.1. 协调到公共 CA

将证书系统 CA 链接到第三方公共 CA 引入了对公共 CA 放置在从属 CA 可能发布的证书类型上的限制，以及证书链的性质。例如，链到第三方 CA 的 CA 可能会限制为仅发出安全多用途互联网邮件扩展(S/MIME)和 SSL/TLS 客户端身份验证证书，但不提供 SSL/TLS 服务器证书。使用公共 CA 还有其他可能的限制。对于某些 PKI 部署，这可能可接受。

与公共 CA 链的一个好处是第三方负责将 root CA 证书提交到 Web 浏览器或其他客户端软件。对于带有无法由管理员控制的不同公司访问的证书，这是一项主要优势。在 CA 层次结构中创建 root CA 意味着本地机构必须将 root 证书获取到所有浏览器中，以使用证书系统发布的证书。在内部网内有一些工具，但使用 extranet 很难完成此操作。

5.2.2. 从属证书系统 CA

证书系统 CA 可以充当 root CA，这意味着服务器签署自己的 CA 签名证书和其他 CA 签名证书，从而创建特定于组织的 CA 层次结构。服务器也可以配置为 从属 CA，这意味着服务器的 CA 签名密钥由现有 CA 层次结构中的另一个 CA 签名密钥签名。

将证书系统 CA 设置为 root CA 意味着证书系统管理员通过设置控制 CA 签名证书内容的策略来控制所有从属 CA。从属 CA 通过评估其自身的身份验证和证书配置文件配置来发布证书，而不考虑 root CA 的配置。

5.2.3. 链接的 CA

证书系统证书管理器可以作为 链接 CA 运作，将多个第三方或公共 CA 串联进行验证；这提供了跨公司信任，因此应用程序可以在公司证书层次结构外验证证书链。证书管理器通过从第三方 CA 请求证书来串联到第三方 CA。

与此相关，证书管理器也可以发出 跨对 或跨签名证书。跨对证书通过在两个 CA 之间发布和存储跨签名证书，在两个独立 CA 之间创建一个可信关系。通过使用跨签名证书对，可在系统内信任在组织的 PKI 外部发布的证书。

它们也称为 网桥证书，与联邦信息处理标准认证机构(FBCA)定义相关。

5.2.4. CA Cloning

除了创建根和从属 CA 的层次结构外，可以创建证书管理器的多个克隆，并将每个克隆配置为在一系列序列号内发布证书。

克隆的 证书管理器使用与主证书管理器(master 证书管理器)相同的 CA 签名密钥和证书。

注意

如果克隆子系统的机会，那么在配置过程中导出其密钥对并将其密钥对保存到安全位置。在配置克隆实例时，原始证书管理器的密钥对必须可用，以便克隆可以从原始证书管理器的密钥生成其证书。

另外，也可以稍后从安全数据库导出密钥，使用 pk12util 或 PKCS12Export 命令。

因为克隆 CA 和原始 CA 使用相同的 CA 签名密钥和证书来签署证书，因此所有证书中的 签发者名称都是相同的。克隆 CA 和原始证书管理器发布证书，就像它们是单个 CA 一样。这些服务器可以放置在不同的主机上，以实现高可用性故障转移支持。

克隆的优点在于，它在多个进程甚至多台物理机器之间分发证书管理器负载。对于具有高注册需求的 CA，从克隆中获取的发行版允许在给定时间间隔内签名和发布更多证书。

克隆的证书管理器具有与常规证书管理器相同的功能，如代理和端点网关功能。

克隆发布的证书的序列号是动态分布。每个克隆和 master 的数据库都会被复制，因此所有证书请求和签发的证书都会被复制。这样可确保没有序列号冲突，而序列号范围不必手动分配给克隆的证书管理器。

5.3. 规划安全域

安全域 是 PKI 服务的注册表。PKI 服务（如 CA）注册这些域中自身的信息，以便 PKI 服务的用户可以通过检查注册表来查找其他服务。证书系统中的安全域服务管理 PKI 服务的注册，以进行证书系统子系统和一组共享信任策略。

该注册表提供该域内子系统提供的所有 PKI 服务的完整视图。每个证书系统子系统必须是主机或安全域的成员。

CA 子系统是可以托管安全域的唯一子系统。安全域共享用于特权用户和组信息的 CA 内部数据库，以确定哪些用户可以更新安全域、注册新的 PKI 服务和发布证书。

在 CA 配置期间创建安全域，它会在安全域 CA 的 LDAP 目录中自动创建条目。每个条目都包含关于域的所有重要信息。域中的每个子系统（包括注册安全域）都会记录在安全域容器条目下。

CA 唯一标识安全域的 URL。安全域也提供了一个友好的名称，如 Corp I intranet PKI。所有其他子系统 - KRA、TPS、TKS、OCSP 和其他 CA - 必须在配置子系统时提供安全域 URL 来成为安全域的成员。

安全域中的每个子系统共享同一信任策略和可信根，这些策略可以从不同的服务器和浏览器检索。安全域中提供的信息在新子系统配置过程中使用，从而简化和自动化配置过程。例如，当 TPS 需要连接到 CA 时，它可以查阅安全域来获取可用 CA 的列表。

每个 CA 都有自己的 LDAP 条目。安全域是 CA 条目下的机构组：

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

然后，在安全域组织组下都有每个子系统类型的列表，它有一个特殊的对象类(pkiSecurityGroup)来识别组类型：

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

然后，每个子系统实例都作为该组的成员存储，使用特殊的 pkiSubsystem 对象类来识别条目类型：

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

如果子系统需要联系另一个子系统来执行某个操作，它将联系托管安全域的 CA （通过调用通过 CA 管理端口连接的 servlet）。然后，安全域 CA 从其 LDAP 数据库检索子系统的信息，并将该信息返回到请求子系统。

子系统使用子系统证书向安全域进行身份验证。

在规划安全域时请考虑以下几点：

- 托管安全域的 CA 可以由外部授权签名。
- 在一个机构中可以设置多个安全域。但是，每个子系统只能属于一个安全域。
- 可以克隆域中的子系统。克隆子系统实例分发系统负载并提供故障转移点。
- 安全域简化了 CA 和 KRA 之间的配置；KRA 可以将其 KRA 连接器信息推送(KRA)信息，并将证书传送到 CA，而不必手动复制证书到 CA。
- 证书系统安全域允许设置离线 CA。在这种情况下，离线 root 拥有自己的安全域。所有在线从属 CA 属于不同的安全域。
- 安全域简化了 CA 和 OCSP 之间的配置。OCSP 可将其信息推送到 CA，以设置 OCSP 发布，并从 CA 检索 CA 证书链并将其存储在内部数据库中。

5.4. 确定子系统证书的要求

CA 配置决定了它发布的证书的许多特征，无论正在发布的实际证书类型。对 CA 自己的有效期、区分名称并允许加密算法的约束会影响签发的证书中的相同特征。另外，证书管理器具有预定义的配置集，它为不同的证书类型设置规则，并可添加或修改其他配置集。这些配置集配置也会影响发布的证书。

5.4.1. 确定要安装的证书

首次安装和配置证书系统子系统时，会自动创建访问和管理它所需的证书。这包括代理的证书、服务器证书和特定于子系统的证书。这些初始证书显示在 表 5.1 “初始子系统证书” 中。

表 5.1. 初始子系统证书

子系统	证书
证书管理器	• CA 签名证书 • OCSP 签名证书 • SSL/TLS 服务器证书 • 子系统证书 • 用户的(agent/administrator)证书 • 审计日志签名证书
OCSP	• OCSP 签名证书 • SSL/TLS 服务器证书 • 子系统证书 • 用户的(agent/administrator)证书 • 审计日志签名证书
KRA	• 传输证书 • 存储证书 • SSL/TLS 服务器证书 • 子系统证书 • 用户的(agent/administrator)证书 • 审计日志签名证书
TKS	• SSL/TLS 服务器证书 • 用户的(agent/administrator)证书 • 审计日志签名证书
TPS	• SSL/TLS 服务器证书 • 用户的(agent/administrator)证书 • 审计日志签名证书

关于替换现有子系统证书有一些注意事项。

- 为 root CA 创建新自签名 CA 证书时生成新的密钥对，使之前 CA 证书下发布的所有证书都无效。

  这意味着，使用其旧密钥的 CA 发布或签名的证书都无法正常工作；从属证书管理器、KRA、OCSP、TKS 和 TPS 将不再正常工作，代理无法再访问代理接口。

  如果从属 CA 的 CA 证书被新密钥对替换，会出现这种情况。该 CA 发布的所有证书都无效，且无法正常工作。

  考虑续订现有的 CA 签名证书，而不是从新密钥对创建新证书。

- 如果 CA 配置为发布到 OCSP，并且具有新的 CA 签名证书或新的 CRL 签名证书，则必须将 CA 再次标识到 OCSP。

- 如果为 KRA 创建新的传输证书，则必须在 CA 配置文件 CS.cfg 中更新 KRA 信息。现有传输证书必须替换为 ca.connector.KRA.transportCert 参数中的新证书。

- 如果克隆了 CA，则在为主证书管理器创建新的 SSL/TLS 服务器证书时，克隆 CA 的证书数据库都需要使用新的 SSL/TLS 服务器证书进行更新。

- 如果证书管理器被配置为发布证书，并将 CRL 发布到 LDAP 目录，并使用 SSL/TLS 服务器证书进行 SSL/TLS 客户端身份验证，那么必须使用适当的扩展来请求新的 SSL/TLS 服务器证书。安装证书后，必须将发布目录配置为使用新的服务器证书。

- 可以为子系统实例发布任意数量的 SSL/TLS 服务器证书，但它实际上只需要一个 SSL/TLS 证书。此证书可以根据需要续订或替换。

5.4.2. 规划 CA 可辨识名称

CA 的核心元素是签名单元和证书管理器身份。签名单元签署终端实体请求的证书。证书管理器必须具有自己的可分辨名称(DN)，该名称列在每个证书中。

与其他证书一样，CA 证书将 DN 绑定到公钥。DN 是一系列单独标识实体的 name-value 对。例如，以下 DN 识别了名为 Example Corporation 的公司工程部门的证书管理器：

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

许多名称值对组合可用于证书管理器的 DN。DN 必须是唯一的且易识别，因为任何端点都可以检查它。

5.4.3. 设置 CA Signing Certificate Validity Period

每个证书（包括证书管理器签名证书）都必须具有有效期。证书系统不限制可以指定的有效性周期。根据证书续订的要求、将 CA 放置到证书层次结构中的要求，以及 PKI 中包含的任何公共 CA 的要求，以尽可能设置有效期周期。

证书管理器无法发布有效期超过其 CA 签名证书的有效性周期的证书。如果请求在超过 CA 证书的有效期内发出请求，则请求的有效性日期将被忽略，并使用 CA 签名证书有效期周期。

5.4.4. 选择签名密钥类型和长度

子系统使用签名密钥来验证和"密封"内容。CA 使用 CA 签名证书为证书签名或 CRL，Cryry 使用签名证书来验证其对证书状态请求的响应；所有子系统都使用日志文件签名证书为其审计日志签名。

签名密钥必须经过加密，才能为其签名操作提供保护和安全性。以下签名算法被视为安全：

- SHA256withRSA
- SHA512withRSA
- SHA256withEC
- SHA512withEC

注意

证书系统包括原生 ECC 支持。也可以加载并使用启用了 ECC 的第三方 PKCS facilities 模块。这在 第 9 章 使用 ECC 系统证书安装实例 中进行了介绍。

除了密钥类型 外，每个密钥都有一个特定的 位长度。较长的密钥比较短的密钥更强。但是，较长的密钥需要更多时间来签名操作。

配置向导中的默认 RSA 密钥长度为 2048 位；对于提供对高度敏感数据或服务的访问的证书，请考虑将长度增加到 4096 位。ECC 密钥比 RSA 密钥更强，因此推荐 ECC 密钥的长度为 256 位，这等同于 2048 位 RSA 密钥。

5.4.5. 使用证书扩展

X.509 v3 证书包含一个扩展字段，允许将任意数量的其他字段添加到证书。证书扩展提供了一种为证书添加其他主题名称和使用限制等信息的方法。较旧的 Netscape 服务器（如 Red Hat Directory Server 和 Red Hat Certificate System）需要定义 Netscape 的扩展，因为它们在 PKIX 第 1 部分标准之前被开发。

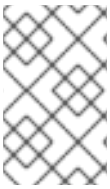
X.509 v1 证书规格最初设计为将公钥绑定到 X.500 目录中的名称。当证书开始在互联网上使用时，无法总是执行 extranets 和 directory 查找，因此问题区域会被原始规格涵盖的问题。

- 信任.X.500 规范通过严格的目录层次结构建立信任。相比之下，互联网和额外网络部署通常涉及分布式信任模型，它们不符合分层 X.500 方法。
- 证书使用。有些机构限制了证书的使用方式。例如，一些证书可能仅限于客户端身份验证。
- 多个证书.证书用户没有相同主题名称但不同的密钥材料的多个证书并不常见。在这种情况下，需要确定哪些密钥和证书用于什么目的。
- 备用名称.出于某些目的，具有替代的主题名称也绑定到证书中的公钥非常有用。
- 其他属性。有些机构将其他信息存储在证书中，比如无法在目录中查找信息。
- 与 CA 的关系.当证书链涉及中间 CA 时，可以包含有关嵌入在其证书中的 CA 之间的关系信

息。

- CRL 检查。由于无法始终能够针对目录或原始证书颁发机构检查证书的撤销状态，因此证书对检查 CRL 的信息非常有用。

X.509 v3 规范通过更改证书格式来在证书中包含其他信息，方法是定义证书扩展的通用格式，并指定证书中包含的扩展。为 X.509 v3 证书定义的扩展可启用与用户或公钥关联的额外属性，并管理认证层次结构。Internet X.509 公钥基础架构证书和 CRL 配置文件 建议一组用于互联网证书和标准位置的扩展，用于证书或 CA 信息。这些扩展称为 标准扩展。



注意

有关标准扩展的更多信息，请参阅 RFC 2459、RFC 3280 和 RFC 3279。

证书的 X.509 v3 标准允许组织定义自定义扩展并在证书中包含它们。这些扩展称为 私有、专有 或自定义 扩展，它们向组织或业务提供唯一信息。应用程序可能无法验证包含私有关键扩展的证书，因此不建议在广泛使用它们。

X.500 和 X.509 规范由国际电信联合(ITU)控制，这是一个国际组织，主要服务于大型电信公司、政府组织以及与国际电信网络相关的其他实体。Internet Engineering Task Force (IETF)，其控制互联网的许多标准目前正在开发公钥基础架构 X.509 (PKIX)标准。这些建议的标准进一步重新定义 X.509 v3 方法，以便在互联网上使用。证书和 CRL 的建议已达到了标准状态，并位于称为 PKIX 第 1 部分的文档中。

其他两个标准是 Abstract Syntax Notation One (ASN.1)和 Distinguished Encoding rules (DER)，一般使用证书系统和证书。这些是在 CCITT Recommendations X.208 和 X.209 中指定的。有关 ASN.1 和 DER 的快速摘要，请参阅 Layman 指南到 ASN.1、BER 和 DER 的子集，该指南可通过 RSA 实验室网站 http://www.rsa.com 获得。

5.4.5.1. 证书扩展结构

在 RFC 3280 中，X.509 证书扩展定义如下：

```
Extension  ::=  SEQUENCE  {

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

这意味着证书扩展由以下内容组成：

- 扩展的对象标识符(OID)。此标识符唯一标识扩展。它还决定了 value 字段中的值的 ASN.1 类型以及如何解释值。当扩展出现在证书中时，OID 显示为扩展 ID 字段(extnID)，对应的 ASN.1 编码的结构显示为 octet 字符串值(extnValue)。

- 名为 critical 的标志或布尔值字段。

  分配给此字段的值可以是 true 或 false，表示扩展是否为对证书的关键还是非关键。

  - 如果扩展是关键的，并且证书发送到不根据扩展 ID 理解扩展的应用程序，则应用必须拒绝证书。

  - 如果扩展不是关键的，并且证书发送到不根据扩展 ID 理解扩展的应用程序，则应用可以忽略扩展并接受证书。

- 包含扩展值的 DER 编码的 octet 字符串。

通常，接收证书的应用会检查扩展 ID，以确定它是否可以识别 ID。如果它可以，它会使用扩展 ID 来确定所使用的值类型。

X.509 v3 标准中定义的一些标准扩展包括：

- 授权密钥标识符扩展，用于标识 CA 的公钥，这是用于签署证书的密钥。

- 主题 Key Identifier 扩展，用于标识主题的公钥，公钥被认证。

注意

并非所有应用程序都支持版本 3 扩展的证书。支持这些扩展的应用程序可能无法解释这些特定扩展的一些或全部。

5.4.6. 使用和自定义证书配置文件

证书有不同的类型和不同的应用程序。它们可用于为公司网络建立单点登录环境，设置 VPN、加密电子邮件或向网站进行身份验证。所有这些证书的要求都不同，因为对于不同类别用户，同一类型的证书也有不同的要求。这些证书特征在 证书配置文件 中设置。证书管理器定义了一组证书配置文件，在用户或机器请求证书时用作注册表单。

证书配置文件

证书配置文件定义与发布特定类型的证书有关的所有内容，包括身份验证方法、证书内容(defaults)、内容值的限制，以及证书配置文件的输入和输出内容。注册请求将提交至证书配置文件，然后受到该证书配置文件中设置的默认值和约束。无论请求是通过与证书配置文件关联的输入表单还是通过其他方法提交，这些约束就位。从证书配置文件请求发布的证书包含默认值所需的内容，以及默认参数所需的信息。约束为证书中允许的内容提供规则。

例如，用户证书的证书配置文件定义了该证书的所有方面，包括证书的有效性周期。默认有效期可设置为 2 年，约束可以在通过此证书配置文件请求的证书的有效性周期不能超过两年的配置文件。当用户使用与这个证书配置文件关联的输入表单请求证书时，签发的证书包含在默认值中指定的信息，并在两年内有效。如果用户为具有 4 年有效周期的证书提交预格式化的请求，则请求将被拒绝，因为限制允许此类型的证书有效期最多两年。

为发布的最常见证书预定义了一组证书。这些证书配置文件定义默认值和约束，关联身份验证方法，并为证书配置文件定义所需的输入和输出。

修改证书配置文件参数

可以修改默认证书配置文件的参数；这包括身份验证方法、默认值、每个配置文件中使用的约束、分配给配置集中的任何参数的值、输入和输出的值。也可以为其他类型的证书创建新证书配置文件，或者为证书类型创建多个证书配置文件。特定类型的证书可以有多个证书配置文件，以使用不同的身份验证方法发布相同类型的证书，或为默认值和约束的不同定义发布。例如，注册 SSL/TLS 服务器证书有两个证书配置文件，其中一个证书配置文件发布有效期为 6 个月的证书，另一个证书配置文件发布有效期为 2 年的证书。

输入会设置注册表单中的文本字段以及从端点实体收集的所需信息类型；这包括设置要粘贴证书请求的文本区域，这允许在输入表单外创建请求，其中包含所需的任何请求信息。输入值设置为证书中的值。默认输入在证书系统中不可配置。

输出指定如何显示对成功注册的响应页面。它通常以用户可读格式显示证书。默认输出显示结果证书的可打印版本；其他输出设置注册末尾生成的信息类型，如 PKCS #7。

策略集合是一组限制，以及附加到通过配置集处理的每个证书的默认扩展。扩展定义证书内容，如有效期和主题名称要求。配置集处理一个证书请求，但单个请求可以包含多个证书的信息。PKCS#10 请求

包含一个公钥。一个 CRMF 请求可以包含多个公钥，这意味着多个证书请求。配置集可以包含多个策略集合，每个策略都指定如何在 CRMF 请求中处理一个证书请求。

证书配置文件管理

管理员通过将现有身份验证插件或方法与证书配置文件相关联来设置证书配置文件；启用和配置默认值和约束；以及定义输入和输出。管理员可以使用现有证书配置文件、修改现有证书配置文件、创建新的证书配置文件，以及删除此 PKI 中不使用的任何证书配置文件。

设置证书配置文件后，它会出现在代理服务页的 Manage Certificate Profiles 页面中，代理可以批准，从而启用证书配置文件。启用证书配置文件后，它将显示在终端实体页面的 Certificate Profile 选项卡中，其中最终实体可以使用证书配置文件注册证书。

最终接口中的证书配置文件注册页面包含到代理启用的每个证书配置文件的链接。当端点选择其中一个链接时，将显示一个注册页面，其中包含特定于该证书配置文件的注册表。注册页面从为配置集定义的输入动态生成。如果配置了身份验证插件，则可以添加额外的字段来验证用户。

当最终实体提交与代理批准（手动）注册关联的证书配置文件请求时，没有配置身份验证插件的注册，证书请求会在代理服务界面中排队。代理可以更改注册、请求、验证、取消、拒绝、更新或批准它的一些方面。代理可以在不提交请求的情况下更新请求，或者验证请求是否遵循配置集的默认值和约束。此验证过程仅用于验证，不会造成提交请求。代理由约束集绑定；它们无法更改请求，从而违反约束。已签名的批准会被立即处理，证书会被发布。

当证书配置文件与身份验证方法关联时，请求会立即批准，并在用户成功验证时自动生成证书，提供所需的所有信息，请求不会违反为证书配置文件设置的任何约束。有允许用户提供的设置（如主题名称或有效期）的配置集策略。证书配置文件框架也可以在发布的证书中的原始证书请求中保留用户定义的内容。

发布的证书包含此证书配置文件的默认值中定义的内容，如证书的扩展和有效期。证书的内容受到为每个默认设置的约束。可以为一个配置集设置多个策略(defaults 和 constraints)，通过使用策略设置 ID 中的相同值来区分每个集合。这对处理将加密密钥和签名密钥提交到同一配置集的双密钥注册特别有用。服务器会为其接收的每个请求评估每个集合。发布单个证书时，将评估一个集合，并忽略任何其他集合。发出双密钥对后，将使用第一个证书请求评估第一个集合，并且使用第二个证书请求评估第二个集合。不需要多个集合来发布单个证书或多个集合来发布双密钥对。

自定义证书配置文件指南

根据机构的实际需求定制配置文件，并预计由机构使用的证书类型：

- 决定 PKI 中需要哪些证书配置文件。发行的每种证书类型应至少有一个配置文件。每种证书可以有多个证书配置文件，来为特定类型的证书类型设置不同的身份验证方法或不同的默认值和

约束。管理界面中提供的任何证书配置文件均可由代理批准，然后供终端实体用于注册。

- 删除任何不使用的证书配置文件。
- 为公司证书的特定特征修改现有证书配置文件。
  - 更改证书配置文件中设置的默认值、默认值中设置的参数的值，或控制证书内容的限制。
  - 通过更改参数的值来更改设置的限制。
  - 更改身份验证方法。
  - 通过在证书配置文件中添加或删除输入来更改输入，该输入控制输入页上的字段。
  - 添加或删除输出。

5.4.6.1. 在 SSL 服务器证书中添加 SAN 扩展

证书系统可在安装非 root CA 或其他证书系统实例时，为 SSL 服务器证书添加主题备用名称(SAN)扩展。要做到这一点，请按照 /usr/share/pki/ca/profiles/ca/caInternalAuthServerCert.cfg 文件中的说明进行操作，并将以下参数添加到提供给 pkispawn 工具的配置文件中：

pki_san_inject

将此参数设置为 True。

pki_san_for_server_cert

提供用逗号(,)分隔所需的 SAN 扩展列表。

例如：

```
pki_san_inject=True
pki_san_for_server_cert=intca01.example.com,intca02.example.com,intca.example.com
```

5.4.7. 规划身份验证方法

作为 第 5.4.6 节 “使用和自定义证书配置文件” 中的代表，证书过程 的身份验证 意味着请求证书的用户或实体证明自己是谁。证书系统可以通过三种方式验证实体：

- 在 代理批准的 注册中，最终用户请求将发送到代理以进行批准。代理批准证书请求。
- 在 自动注册 中，最终用户请求使用插件进行身份验证，然后处理证书请求；代理不会涉及注册过程。
- 在 CMC 注册 中，第三方应用程序可以创建由代理签名的请求，然后自动处理。

最初为代理批准注册和 CMC 身份验证配置证书管理器。通过配置其中一个身份验证插件模块来启用自动注册。可以在子系统的单一实例中配置多个身份验证方法。HTML 注册页面包含指定使用方法的隐藏值。使用证书配置文件时，每个启用的配置集都会动态生成最终用户注册页面。与此证书配置文件关联的身份验证方法在动态生成的注册页面中指定。

身份验证过程很简单。

1. 最终实体提交注册请求。用于提交请求的表单标识了身份验证和注册方法。所有 HTML 表单都由配置集动态生成，它会自动将适当的身份验证方法与表单关联。
2. 如果身份验证方法是代理批准的注册，则请求将发送到 CA 代理的请求队列。如果设置了队列中请求的自动通知，则会发送电子邮件到收到新请求的相应代理。代理可以根据该表单和配置集限制修改请求。批准后，请求必须传递为证书管理器设置的证书配置文件，然后发布证书。发布证书时，它存储在内部数据库中，并可通过终端实体通过序列号或请求 ID 从终端实体检索。
3. 如果身份验证方法是自动的，则最终实体将提交请求以及验证用户身份所需的信息，如 LDAP 用户名和密码。当用户成功通过身份验证时，会在不发送到代理队列的情况下处理请求。如果请求通过证书管理器的证书配置文件配置，则会发布证书并存储在内部数据库中。它通过 HTML 表单立即发送到终端实体。

对证书请求的验证方式的要求可以直接影响所需的子系统和配置文件设置。例如，如果代理批准的注册要求代理符合个人请求者，并通过支持的文档验证其身份，则身份验证过程可能会非常耗时，并由代理

和请求者的物理可用性的限制。

5.4.8. 发布证书和 CRL

CA 可以发布证书和 CRL。证书可以发布到普通文件或 LDAP 目录；也可以使用 CRL 发布到文件或 LDAP 目录，也可以发布给 OCSP 响应程序来处理证书验证。

配置发布非常简单，可轻松调整。但是，为了实现连续性和可访问性，最好计划证书和 CRL 需要发布的位置，以及哪些客户端需要访问它们。

发布到 LDAP 目录需要目录中的特殊配置才能发布：

- 如果证书发布到目录中，超过签发证书的每个用户或服务器必须在 LDAP 目录中具有对应的条目。
- 如果 CRL 发布到目录中，超过它们必须发布到签发它们的 CA 的条目。
- 对于 SSL/TLS，必须在 SSL/TLS 中配置目录服务，并选择性地进行配置以允许证书管理器使用基于证书的身份验证。
- 目录管理员应配置适当的访问控制规则，以控制 DN （条目名称）和密码对 LDAP 目录的身份验证。

5.4.9. 续订或恢复 CA 签署证书

当 CA 签名证书过期时，所有使用 CA 对应的签名密钥签名的证书都无效。结束实体使用 CA 证书中的信息来验证证书的真实性。如果 CA 证书本身已过期，应用程序无法将证书链接到可信 CA。

解析 CA 证书过期的方法有两种：

- 续订 CA 证书涉及使用与旧 CA 证书相同的主题名称和公钥和私钥材料发布一个新的 CA 证书，但具有延长的有效性周期。只要新 CA 证书在旧 CA 证书过期前向所有用户分发，续订证书允许旧 CA 证书下发布的证书在有效期内继续工作。
-

Reissuing a CA 证书涉及使用新名称、公钥和私钥材料发布新的 CA 证书以及有效期。这可避免与更新 CA 证书有关的一些问题，但它需要更多工作才能使管理员和用户实施。旧 CA 发布的所有证书（包括尚未过期的证书）都必须由新 CA 更新。

续订或重新颁发 CA 证书时可能会有问题和优势。在安装任何证书管理器前，开始规划 CA 证书续订或恢复，并考虑计划的流程在 PKI 部署的其他方面具有扩展、策略和其他方面。

注意

正确使用扩展（如 authorityKeyIdentifier 扩展）可能会影响从旧 CA 证书转换到新证书的转换。

5.5. 规划网络和物理安全性

在部署任何证书系统子系统时，必须考虑子系统实例的物理和虚拟安全性，因为子系统生成的数据的敏感度。

5.5.1. 考虑防火墙

关于在证书系统子系统中使用防火墙有两个注意事项：

- 保护敏感子系统不受未授权访问的影响
- 允许适当的访问防火墙外的其他子系统和客户端

CA、KRA 和 TKS 始终放在防火墙中，因为它们包含关键信息，如果它们被破坏，可能会导致安全后果。

TPS 和 OCSP 可以在防火墙外放置。同样，证书系统使用的其他服务和客户端也可以位于防火墙之外的其他计算机上。在这种情况下，本地网络必须配置为允许防火墙后面的子系统及其外服务间的访问。

LDAP 数据库也可以位于不同的服务器上，甚至位于不同的网络上，这与使用它的子系统不同。在这种情况下，所有 LDAP 端口(LDAP 的389 和 636 用于 LDAPS)需要在防火墙中打开，以允许到目录服务的流量。如果没有访问 LDAP 数据库，所有子系统操作都可能会失败。

作为配置防火墙的一部分，如果启用了 iptables，则必须有配置策略以允许通过适当的证书系统端口进行通信。使用和配置 firewalld 指南 中描述了配置 iptables。

5.5.2. 考虑物理安全性和位置

由于它们保存的敏感数据，请考虑在具有足够物理访问限制的安全工具中保持 CA、KRA 和 TKS。正如需要对系统的网络访问一样，还需要限制物理访问。

除了查找安全位置外，请考虑子系统代理和管理员的性能。例如，密钥恢复可能需要多个代理进行批准；如果这些代理分散在大型地理区域上，则时间差异可能会对检索密钥的功能造成负面影响。根据代理和将管理子系统的管理员的位置，规划子系统的物理位置。

5.5.3. 规划端口

每个证书系统服务器使用最多四个端口：

- 对于不需要身份验证的最终用户的非安全 HTTP 端口
- 用于最终用户服务、代理服务、管理控制台和需要身份验证的 admin 服务的安全 HTTP 端口
- Tomcat 服务器管理端口
- Tomcat AJP Connector 端口

Red Hat Certificate System Administration Guide 中的 Red Hat Certificate System User Interfaces 章节中描述的所有服务页面和接口都连接到使用实例的 URL 和相应的端口号。例如：

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

要访问管理控制台，URL 指定 admin 端口：

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

所有代理和 admin 功能都需要 SSL/TLS 客户端身份验证。对于来自端实体的请求，证书系统侦听

SSL/TLS （加密）端口和非 SSL/TLS 端口。

端口在 server.xml 文件中定义。如果没有使用端口，可以在该文件中禁用它。例如：

```
<Service name="Catalina">
   <!--Connector port="8080" ... /-->  unused standard port
   <Connector port="8443" ... />
```

每当安装的一个新实例时，请确保新端口在主机系统上是唯一的。

要验证端口是否可用，请检查操作系统的适当文件。网络可访问的服务的端口号通常在名为 services 的文件中维护。在 Red Hat Enterprise Linux 上，通过运行命令 semanage port -l 列出当前具有 SELinux 上下文的所有端口，从而确认 SELinux 未分配端口。

当创建新子系统实例时，1 到 65535 之间的任意数字都可以指定为安全端口号。

5.6. 用于清理证书系统子系统密钥和证书的令牌

令牌是 执行加密功能的硬件或软件设备，并存储公钥证书、加密密钥和其他数据。 证书系统定义了两种类型的令牌，即 内部和外部 ，用于存储属于证书系统子系统的密钥对和证书。

内部（软件）令牌是一组文件，通常称为 证书数据库 (cert9.db)和 密钥数据库 (key4.db)，用于生成和存储其密钥对和证书。首次使用内部令牌时，证书系统会在主机机器的文件系统中自动生成这些文件。如果为密钥对生成选择了内部令牌，则这些文件是在证书系统子系统配置期间创建的。

这些安全数据库位于 /var/lib/pki/instance_name/alias 目录中。

外部令牌引用外部硬件设备，如智能卡或硬件安全模块(HSM)，证书系统用来生成和存储其密钥对和证书。证书系统支持任何与 PKCSENCODED 兼容的硬件令牌。

PKCSBACKEND 是一组标准的 API 和共享库，可将应用程序与加密设备的详情隔离开来。这可让应用程序为 PKCSENCODED 兼容加密设备提供统一接口。

证书系统中实施的 PKCSBACKEND 模块支持由许多不同制造商提供的加密设备。此模块允许证书系统插入由外部加密设备提供的共享库，并使用它们为证书系统管理器生成和存储密钥和证书。

考虑使用外部令牌生成和存储证书系统使用的密钥对和证书。这些设备是保护私钥的另一个安全措施，因为硬件令牌有时被视为比软件令牌更安全。

在使用外部令牌前，请规划如何将外部令牌用于子系统：

- 子系统的所有系统密钥都必须在同一令牌上生成。
- 该子系统必须安装在空的 HSM 插槽中。如果 HSM 插槽之前已经用于存储其他密钥，则使用 HSM 厂商的工具删除插槽的内容。证书系统必须能够在带有默认 nickname 的插槽上创建证书和密钥。如果没有正确清理，这些对象的名称可能会与之前的实例冲突。

证书系统还可以将 硬件加密加速器 与外部令牌一起使用。许多加速器提供以下安全功能：

- 快速 SSL/TLS 连接。速度对于同时注册或服务请求的数量非常重要。
- 私钥的硬件保护。这些设备的行为与智能卡不同，不允许从硬件令牌中复制或移除私钥。这一点非常重要，这是对在线证书管理器主动攻击的关键问题。

证书系统默认支持 nCipher nShield Connect XC 硬件安全模块(HSM)。如果在默认安装路径的预配置阶段，使用 modutil 的证书系统支持的 HSM 会自动添加到 pkcs11.txt 数据库中。

在配置过程中，Security Modules 面板显示支持的模块，以及 NSS 内部软件 PKCS facilities 模块。所有检测到的支持的模块都显示 Found 状态，并单独标记为 Logged in 或 Not logged。如果找到了令牌但没有登录，则可以使用 Operations 下的 Login 登录。如果管理员可以成功登录令牌，密码将存储在配置文件中。在下一次启动或重启证书系统实例时，密码存储中的密码用于尝试每个对应令牌的登录。

管理员可以选择任何作为默认令牌登录的令牌，该令牌用于生成系统密钥。

5.7. 规划 PKI 的检查列表

问：

将创建多少个安全域，以及将哪些子系统实例放在每个域中？

答：

如果子系统具有可信关系，则只能与另一个子系统通信。由于安全域中的子系统相互自动信任关系，因此子系统加入域非常重要。安全域可以有不同的证书发布策略、不同的子系统类型，或者不同的目录服务器数据库。映射每个子系统所属的位置（包括物理机上和相互关系），并相应地将其分配到安全域。

问：

应为每个子系统分配哪些端口？是否需要具有单个 SSL/TLS 端口，或者最好让端口分离以提高安全性？

答：

最安全的解决方案是为每个 SSL/TLS 接口使用单独的端口。但是，实现多个端口的可行性可能取决于您的网络设置、防火墙规则和其他网络状况。

问：

哪些子系统应放在防火墙后面？哪些客户端或其他子系统需要访问这些受防火墙保护的子系统，以及如何授予访问权限？LDAP 数据库是否允许防火墙访问？

答：

安装子系统的位置取决于网络设计。OCSP 子系统特别设计为在防火墙外操作方便用户，而 CA、KRA 和 TPS 在防火墙后面都应安全以提高安全性。

在决定子系统的位置时，务必要规划服务器需要访问 的其他子系统 或服务（包括内部数据库、CA 或外部用户），并查看防火墙、子网和 VPN 配置。

问：

需要物理保护哪些子系统？将如何授予访问权限，以及谁将被授予访问权限？

答：

CA、TKS 和 KRA 所有存储非常敏感的密钥和证书信息。对于某些部署，可能需要限制这些子系统运行的计算机的物理访问。在这种情况下，子系统及其主机机器都必须包含在更大的基础架构清单和安全设计中。

问：

所有代理和管理员的物理位置是什么？子系统的物理位置是什么？管理员或代理如何及时访问子系统服务？每个地理位置或时区需要有子系统吗？

答：

证书系统用户的物理位置可能会影响请求批准和令牌注册等内容，以及因为网络滞后造成的系统性能。在决定要安装的子系统的位置和数量时，应考虑处理证书操作的响应时间的重要性。

问：

您需要安装多少个子系统？

答：

主要考虑是每个子系统的预期负载，然后是第二级的地理位置或部门部门。具有公平负载（如 KRA 或 TKS）的子系统可能只需要整个 PKI 的一个实例。具有高负载(CA)的系统，或者可能受益于本地代理的本地实例（如 TPS）可能需要多个实例。

可以克隆子系统，这意味着它们本质上是集群，作为一个单元运行，这适用于负载平衡和高可用性。克隆特别适用于 CA 和 KRA，其中可能需要通过大量用户访问相同的密钥和证书数据，或者具有可靠的故障转移。

在规划多个子系统实例时，请记住子系统在已建立的安全域中如何适合。安全域在子系统之间创建可信关系，允许它们一起查找可用子系统以响应即时需求。单个 PKI 中可以使用多个安全域，以及多种类型的子系统实例，或者一个安全域可用于所有子系统。

问：

是否需要克隆任何子系统，如果如此，那么可以安全地存储其关键资料的方法？

答：

克隆的子系统可以一起工作，基本上作为一个实例。这对高需求系统、故障转移或负载平衡可能很好，但维护可能变得困难。例如，克隆的 CA 具有它们发布的证书的序列号范围，克隆可能会达到其范围的末尾。

问：

子系统证书和密钥是否存储在证书系统中的内部软件令牌或外部硬件令牌上？

答：

证书系统支持两个硬件安全模块(HSM)：nCipher nShield Connect XC 和 Thales Luna HSM。在安装子系统前，使用硬件令牌可能需要额外的设置和配置，但它也会添加另一个安全层。

问：

CA 签名证书的要求是什么？证书系统是否需要控制属性，如有效期？CA 证书如何分发？

答：

这里的实际问题是，CA 将是一个 root CA，它在发布证书时设置自己的规则，或者将一个从属 CA （您的 PKI 中的另一个 CA 还是外部 CA）设置对其可能发布的证书类型的限制。

证书管理器可以配置为 root CA 或从属 CA。root CA 和从属 CA 之间的区别在于谁为 CA 签名证书签名。root CA 为自己的证书签名。从属 CA 有另一个 CA （内部或外部）为其证书签名。

自签名 root CA 问题并签署自己的 CA 签名证书。这允许 CA 设置自己的配置规则，如有效周期和允许的从属 CA 的数量。

从属 CA 具有其由公共 CA 或其他证书系统 root CA 发布的证书。此 CA 会 从属 到其他 CA 的规则有关其证书设置以及如何使用证书，如它可以发布的证书类型、允许包含在证书中的扩展以及从属 CA 的从属 CA 可以创建。

一个选项是让证书管理器从属到公共 CA。这可能非常严格，因为它引入了公共 CA 放置到从属 CA 可能会发布的证书类型的限制，以及证书链的性质。另一方面，与公共 CA 串联的一个好处是，第三方负责将 root CA 证书提交到 Web 浏览器或其他客户端软件，这对具有管理员无法由浏览器控制的不同公司访问的证书具有主要优势。

其他选项使 CA 从属到证书系统 CA。将证书系统 CA 设置为 root CA 意味着证书系统管理员通过设置控制 CA 签名证书内容的策略来控制所有从属 CA。

最简单的方法是使第一个 CA 安装自签名 root，因此不需要应用到第三方并等待证书发布。请确定您确定有多少个 root CA 以及根和从属 CA 的位置。

问：

将发布哪些证书？它们需要具备哪些特征，以及哪些配置文件设置可用于这些特征？需要对证书放置哪些限制？

答：

与 第 5.4.6 节 “使用和自定义证书配置文件” 中的联系一样，可以自定义配置 CA 发布的每个证书类型的配置集（配置由 CA 发布的证书类型的格式）。这意味着主题 DN、有效周期、SSL/TLS 客户端的类型和其他元素可由管理员为特定目的定义。为安全起见，配置文件应仅提供 PKI 所需的功能。应该禁用任何不必要的配置集。

问：

批准证书请求的要求是什么？请求者如何对自己进行身份验证，以及批准请求需要哪种流程？

答：

请求批准过程直接影响证书的安全性。自动化流程速度更快，但安全性较低，因为请求者的身份仅经过监管地验证。同样，代理批准的注册也更为安全（因为在最安全的场景中，他们可能需要个人验证和支持识别功能），但它们也可能是最耗时的。

首先确定需要保护身份验证过程的安全，然后确定验证身份所需的身份验证(approval)机制。

问：

许多外部客户端是否需要验证证书状态？证书管理器中的内部 OCSP 能否处理负载？

答：

发布 CRL 并验证证书非常重要。确定哪个类型的客户端将检查证书状态（主要是内部客户端？外部客户端会验证用户证书或服务器证书？），并尝试确定将运行多少 OCSP 检查。CA 有一个内部 OCSP，可用于内部检查或不经常检查，但大量 OCSP 检查可能会减慢 CA 性能的速度。对于大量 OCSP 检查，特别是对于大型 CRL，请考虑使用单独的 OCSP Manager 来关闭 CA。

问：

PKI 是否允许替换密钥？它是否需要密钥归档和恢复？

答：

如果丢失的证书或令牌被简单地撤销和替换，则不需要有一个机制来恢复它们。但是，当使用证书签名或加密数据（如电子邮件、文件或硬盘）时，密钥必须始终可用，以便可以恢复数据。在这种情况下，使用 KRA 归档密钥，以便始终访问数据。

问：

组织是否使用智能卡？如果是这样，如果智能卡被误解，则允许临时智能卡，需要密钥存档和恢复？

答：

如果没有使用智能卡，则无需配置 TPS 或 TKS，因为这些子系统仅用于令牌管理。但是，如果使用智能卡，则需要 TPS 和 TKS。如果令牌可以被丢失和替换，那么还需要有一个 KRA 来归档密钥，以便重新生成令牌的证书。

问：

发布证书和 CRL 在哪里？在接收结束时需要进行什么配置才能发布工作？需要发布哪些证书或 CRL？

答：

确定什么客户端需要访问证书或 CRL 信息以及允许该访问的方式。在这里，您的 cna 定义发布策略。

5.8. 可选的第三方服务

5.8.1. Load Balancers

5.8.2. 备份硬件和软件

部分 II. 安装 RED HAT CERTIFICATE SYSTEM

这部分论述了安装 Red Hat Certificate System 的要求和步骤。

第 6 章 安装的先决条件和准备

Red Hat Certificate System 安装过程需要一些准备环境。本章论述了安装证书系统子系统时的要求、依赖项和其他先决条件。

6.1. 安装 RED HAT ENTERPRISE LINUX

Red Hat Certificate System 10.4 需要 Red Hat Enterprise Linux 8.6 版本。有关安装 Red Hat Enterprise Linux 的详情，请参阅执行标准 RHEL 安装。

要在 Red Hat Enterprise Linux 上启用联邦信息处理标准(FIPS)，请参阅红帽安全强化指南。

重要

在安装证书系统前，必须在 RHEL 主机上启用 FIPS 模式。确保启用了 FIPS 模式：

```
# sysctl crypto.fips_enabled
```

如果返回的值是 1，则启用 FIPS 模式。

6.2. 使用 SELINUX 保护系统

Security-Enhanced Linux (SELinux)是 Linux 内核中强制访问控制机制的实现，在检查标准自主访问控制后检查允许的操作。SELinux 可以根据定义的策略对 Linux 系统中的文件和进程以及其操作实施规则。

在大多数情况下，不需要在 enforcing 模式下运行带有 SELinux 的证书系统。如果证书系统文档中的流程需要手动设置与 SELinux 相关的设置，比如在使用硬件安全模块(HSM)时，这在相应的部分中提到。

有关 SELinux 的详情，请查看 使用 SELinux 指南。

6.2.1. 验证 SELinux 是否在强制模式中运行

默认情况下，安装 Red Hat Enterprise Linux 后，SELinux 会被启用并在 enforcing 模式下运行，且不需要进一步的操作。

要显示当前的 SELinux 模式，请输入：

```
# getenforce
```

6.3. 防火墙配置

下表列出了证书系统子系统使用的默认端口：

表 6.1. 证书系统默认端口

服务	端口	协议
HTTP	8080	TCP
HTTPS	8443	TCP
Tomcat 管理	8005	TCP

当使用 pkispawn 工具设置证书系统时，您可以自定义端口号。如果您使用与以上列出的默认值不同的端口，请在防火墙中相应地打开它们，如 第 6.3.1 节 “在防火墙中打开所需端口” 所述。有关端口的详情，请参考 第 5.5.3 节 “规划端口”。

有关访问目录服务器所需的端口，请参阅 目录服务器安装指南中的相应部分。

6.3.1. 在防火墙中打开所需端口

要启用客户端和证书系统之间的通信，请在防火墙中打开所需的端口：

1. 确保 firewalld 服务正在运行。

```
# systemctl status firewalld
```

2. 启动 firewalld 并将其配置为在系统引导时自动启动：

```
# systemctl start firewalld
# systemctl enable firewalld
```

3. 使用 firewall-cmd 工具打开所需的端口。例如，要在默认防火墙区中打开证书系统默认端口：

```
# firewall-cmd --permanent --add-port={8080/tcp,8443/tcp,8009/tcp,8005/tcp}
```

有关使用 firewall-cmd 在系统中打开端口的详情，请查看 安全网络 或 firewall-cmd(1) man page。

4. 重新载入防火墙配置以确保更改会立即进行：

```
# firewall-cmd --reload
```

6.4. 硬件安全模块

要使用硬件安全模块(HSM)，需要一个联邦信息处理标准(FIPS) 140-2 验证的 HSM。如需了解在 FIPS 模式下安装、配置以及如何设置 HSM，请参阅您的 HSM 文档。

6.4.1. 为 HSM 设置 SELinux

某些 HSM 要求您在安装证书系统前手动更新 SELinux 设置。

下面的部分描述了支持的 HSMs 所需的操作：

nCipher nShield

安装 HSM 并在开始安装证书系统前：

1. 重置 /opt/nfast/ 目录中文件的上下文：

```
# restorecon -R /opt/nfast/
```

2. 重新启动 nfast 软件。

-

```
# /opt/nfast/sbin/init.d-ncipher restart
```

Thales Luna HSM

在开始安装证书系统前，不需要与 SELinux 相关的操作。

有关支持的 HSM 的详情，请参考 第 4.4 节 “支持的硬件安全模块”。

6.4.2. 在 HSM 中启用 FIPS 模式

要在 HSM 上启用 FIPS 模式，请参阅 HSM 供应商文档。

重要

nCipher HSM

在 nCipher HSM 中，只有生成 Security World 时才能启用 FIPS 模式，之后无法更改它。虽然有各种生成 Security World 的方法，但首选的方法是使用 new-world 命令。有关如何生成 FIPS 兼容安全世界的指导，请按照 nCipher HSM 供应商的文档进行操作。

LunaSA HSM

同样，必须在 Luna HSM 上启用 FIPS 模式，因为更改此策略会将 HSM 归类为安全措施。详情请查看 Luna HSM 供应商的文档。

6.4.3. 验证 HSM 上是否启用了 FIPS 模式

本节描述了如何为某些 HSM 验证是否启用了 FIPS 模式。有关其他 HSM，请查看硬件制造商的文档。

6.4.3.1. 验证 nCipher HSM 中是否启用了 FIPS 模式

注意

有关完整的流程，请参阅 HSM 供应商的文档。

要验证在 nCipher HSM 上是否启用了 FIPS 模式，请输入：

```
# /opt/nfast/bin/nfkminfo
```

在使用旧版本的软件时，如果 StrictFIPS140 列在 state 标记中，则启用 FIPS 模式。在较新的版本中，最好检查新模式行并查找 fips1402level3。在所有情况下，nfkminfo 输出中也应该有一个 hkfips 密钥。

6.4.3.2. 验证 Luna SA HSM 上是否启用了 FIPS 模式

注意

有关完整的流程，请参阅 HSM 供应商的文档。

验证 Luna SA HSM 上是否启用了 FIPS 模式：

1. 打开 lunash 管理控制台

2. 使用 hsm show 命令，并验证输出中是否包含文本 The HSM in FIPS 140-2 approved operation mode。

```
lunash:> hsm show
...
    FIPS 140-2 Operation:
    ====================
    The HSM is in FIPS 140-2 approved operation mode.
...
```

6.4.4. 准备使用 HSM 安装证书系统

在 第 7.3 节 “了解 pkispawn 实用程序” 中，在使用 HSM 安装证书系统时，您会被指示在配置文件中使用您传递给 pkispawn 工具的以下参数：

```
...
[DEFAULT]
############################
# Provide HSM parameters #
############################
pki_hsm_enable=True
pki_hsm_libfile=hsm_libfile
pki_hsm_modulename=hsm_modulename
pki_token_name=hsm_token_name
pki_token_password=pki_token_password
```

```
########################################
# Provide PKI-specific HSM token names #
########################################
pki_audit_signing_token=hsm_token_name
pki_ssl_server_token=hsm_token_name
pki_subsystem_token=hsm_token_name
...
```

- **pki_hsm_libfile 和 pki_token_name 参数的值取决于您的特定 HSM 安装。这些值允许 pkispawn 工具设置 HSM 并启用证书系统来连接它。**

- **pki_token_password 的值取决于您的特定 HSM 令牌的密码。密码提供 pkispawn 工具读写权限，以便在 HSM 上创建新密钥。**

- **pki_hsm_modulename 的值是后续 pkispawn 操作中使用的名称来识别 HSM。字符串是一个标识符，您可以原样设置。它允许 pkispawn 和 Certificate System 在后续操作中按名称引用 HSM 和配置信息。**

以下部分为各个 HSMs 提供设置。如果您的 HSM 没有列出，请咨询您的 HSM 制造商的文档。

6.4.4.1. nCipher HSM 参数

对于 **nCipher HSM，请设置以下参数：**

```
pki_hsm_libfile=/opt/nfast/toolkits/pkcs11/libcknfast.so
pki_hsm_modulename=nfast
```

请注意，您可以将 pki_hsm_modulename 的值设置为任何值。以上是推荐的值。

例 6.1. 识别令牌名称

要识别令牌名称，以 root 用户身份运行以下命令：

```
[root@example911 ~]# /opt/nfast/bin/nfkminfo
World
 generation  2

...~snip~...

Cardset
```

```
name          "NHSM-CONN-XC"
k-out-of-n    1/4
flags         NotPersistent PINRecoveryRequired(enabled) !RemoteEnabled
timeout       none

...~snip~...
```

Cardset 部分中的 name 字段的值列出了令牌名称。

设置令牌名称，如下所示：

```
pki_token_name=NHSM-CONN-XC
```

6.4.4.2. SafeNet / Luna SA HSM 参数

对于 **SafeNet / Luna SA HSM，如 SafeNet Luna Network HSM，请指定以下参数：**

```
pki_hsm_libfile=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
pki_hsm_modulename=lunasa
```

请注意，您可以将 pki_hsm_modulename 的值设置为任何值。以上是推荐的值。

例 6.2. 识别令牌名称

要识别令牌名称，以 **root** 用户身份运行以下命令：

```
# /usr/safenet/lunaclient/bin/vtl verify

The following Luna SA Slots/Partitions were found:

Slot    Serial #          Label
====    ================    =====
  0     1209461834772     lunasaQE
```

label 列中的值列出了令牌名称。

设置令牌名称，如下所示：

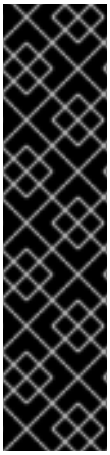
```
pki_token_name=lunasaQE
```

6.4.5. 在硬件安全模块中备份密钥

无法将 HSM 中存储的密钥和证书导出到 .p12 文件。如果要备份这样的实例，请联络您的 HSM 厂商以获得支持。

6.5. 安装 RED HAT DIRECTORY SERVER

证书系统使用红帽目录服务器来存储系统证书和用户数据。您可以在网络中的相同或任何其他主机上安装目录服务器和证书系统。



重要

在安装 Directory 服务器前，必须在 RHEL 主机上启用 FIPS 模式。确保启用了 FIPS 模式：

```
# sysctl crypto.fips_enabled
```

如果返回的值是 1，则启用 FIPS 模式。

6.5.1. 为证书系统准备目录服务器实例

执行以下步骤安装 Red Hat Directory Server：

1. 确保您已附加了一个向主机提供目录服务器的订阅。

2. 启用 Directory Server 存储库：

```
# subscription-manager repos --enable=dirsrv-11-for-rhel-8-x86_64-rpms
```

3. 安装 Directory 服务器和 openldap-clients 软件包：

```
# dnf module install redhat-ds
```

```
# dnf install openldap-clients
```

4. 设置 Directory 服务器实例。

a. 生成 DS 配置文件，例如 /tmp/ds-setup.inf ：

```
$ dscreate create-template /tmp/ds-setup.inf
```

b. 自定义 DS 配置文件，如下所示：

```
$ sed -i \
-e "s/;instance_name = .*/instance_name = localhost/g" \
-e "s/;root_password = .*/root_password = Secret.123/g" \
-e "s/;suffix = .*/suffix = dc=example,dc=com/g" \
-e "s/;create_suffix_entry = .*/create_suffix_entry = True/g" \
-e "s/;self_sign_cert = .*/self_sign_cert = False/g" \
/tmp/ds-setup.inf
```

c. 使用带有 设置 配置文件的 dscreate 命令创建实例：

```
# dscreate from-file /tmp/ds-setup.inf
```

具体步骤请查看 红帽目录服务器安装指南。

6.5.2. 准备配置证书系统

在 第 7.3 节 “了解 pkispawn 实用程序” 中，如果您选择在证书系统和目录服务器之间设置 TLS，请在安装证书系统时使用您传递给 pkispawn 工具的配置文件中的以下参数：

注意

我们需要首先创建基本的 TLS 服务器身份验证连接。最后，在在安装后，我们将返回并建立连接，要求向目录服务器呈现客户端身份验证证书。此时，一旦设置了客户端身份验证，pki_ds_password 不再相关。

```
pki_ds_database=back_end_database_name
pki_ds_hostname=host_name
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
```

```
pki_ds_password=password
pki_ds_ldaps_port=port
pki_ds_bind_dn=cn=Directory Manager
```

pki_ds_database 参数的值是 pkispawn 实用程序使用的名称，用于在 Directory Server 实例上创建对应的子系统数据库。

pki_ds_hostname 参数的值取决于目录服务器实例的安装位置。这取决于 第 6.5.1 节 “为证书系统准备目录服务器实例” 中使用的值。

当您设置 pki_ds_secure_connection=True 时，必须设置以下参数：

- pki_ds_secure_connection_ca_pem_file ：设置完全限定路径，包括包含目录服务器 CA 证书的导出副本的文件名称。在 pkispawn 能够使用该文件之前，该文件必须已存在。
- pki_ds_ldaps_port ：设置安全 LDAPS 端口目录服务器的值。默认值为 636。

6.6. 在目录服务器(CA)中替换临时自签名证书

如果内部 LDAP 服务器最初是使用临时自签名服务器证书创建的，请在安装完成后，请参阅 Post-Installation 部分中的 第 7.10.2 节 “替换临时证书”，将临时证书替换为 CA 发布的新证书。

6.7. 为内部 LDAP 服务器启用 TLS 客户端身份验证

Red Hat Certificate System 可以通过 TLS 双向身份验证与其内部 LDAP 服务器通信。完成安装后，请参阅安装后部分中的 第 7.10.3 节 “启用 TLS 客户端身份验证” 以了解有关如何启用它的详情。

6.8. 附加红帽订阅并启用证书系统软件包存储库

在安装和更新证书系统前，您必须启用对应的存储库：

1. 将红帽订阅附加到系统：

   如果您的系统已经注册或者有附加了证书服务器的订阅，请跳过这一步。

a.

将该系统注册到红帽订阅管理服务。

您可以使用 **--auto-attach** 选项为操作系统自动应用可用的订阅。

```
# subscription-manager register --auto-attach
Username: admin@example.com
Password:
The system has been registered with id: 566629db-a4ec-43e1-aa02-9cbaa6177c3f

Installed Product Current Status:
Product Name:          Red Hat Enterprise Linux Server
Status:                Subscribed
```

b.

列出可用的订阅，并记录提供红帽证书系统的池 ID。例如：

```
# subscription-manager list --available --all
...
Subscription Name:   Red Hat Enterprise Linux Developer Suite
Provides:            ...
                     Red Hat Certificate System
                     ...
Pool ID:             7aba89677a6a38fc0bba7dac673f7993
Available:           1
...
```

如果您有很多订阅，命令的输出可能会非常长。您可以选择将输出重定向到文件中：

```
# subscription-manager list --available --all > /root/subscriptions.txt
```

c.

使用上一步中的池 ID 将证书系统订阅附加到系统：

```
# subscription-manager attach --pool=7aba89677a6a38fc0bba7dac673f7993
Successfully attached a subscription for: Red Hat Enterprise Linux Developer Suite
```

2.

启用证书系统存储库：

```
# subscription-manager repos --enable certsys-10.x-for-rhel-8-x86_64-rpms
```

其中 **x** 表示最新的证书系统版本。例如，要为 **RHCS 10.4** 启用证书系统存储库，请使用以下命令：

```
# subscription-manager repos --enable certsys-10.4-for-rhel-8-x86_64-rpms
Repository 'certsys-10.4-for-rhel-8-x86_64-rpms' is enabled for this system.
```

3. 启用证书系统模块流：

```
# dnf module enable redhat-pki
```

第 7 章 安装和配置证书系统 章节描述了安装所需的软件包。

注意

为满足合规性，仅启用红帽批准的存储库。只有红帽批准的存储库才能通过 **subscription-manager** 工具启用。

6.9. 证书系统用户和组

安装证书系统时，则会自动创建 **pkiuser** 帐户和对应的 **pkiuser** 组。证书系统使用此帐户和组来启动服务。

作为安装后流程的一部分，稍后您将被指示为每个应该能够启动和停止或直接配置操作系统的系统管理员创建操作系统用户。详情请查看 第 7.10 节 “安装后的任务”。

第 7 章 安装和配置证书系统

Red Hat Certificate System 提供不同的子系统，可以单独安装。例如，您可以在单一服务器上安装多个子系统实例，或者您可以在不同的主机上独立运行它们。这可让您根据环境调整安装，以提供更高的可用性、可扩展性和故障切换支持。本章论述了软件包安装以及如何设置各个子系统。

证书系统包括以下子系统：

- 证书颁发机构(CA)
- 密钥恢复授权中心 (KRA)
- 在线证书状态协议(OCSP)响应
- 令牌密钥服务(TKS)
- 令牌处理系统(TPS)

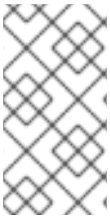
每个子系统都单独安装和配置为独立 Tomcat Web 服务器实例。但是，Red Hat Certificate System 还支持运行一个共享 Tomcat web 服务器实例，它最多可包含每个子系统之一。

7.1. 子系统配置顺序

由于不同子系统之间的关系，设置单个子系统的顺序非常重要：

1. 在安装任何其他公钥基础架构(PKI)子系统之前，至少需要一台作为安全域运行的 CA。
2. 配置 CA 后安装 OCSP。
3. KRA 和 TKS 子系统可以在配置 CA 和 OCSP 后以任何顺序安装。

4. TPS 子系统依赖于 CA 和 TKS，以及可选的 KRA 和 OCSP 子系统。



注意

在某些情况下，管理员希望安装不需要作为安全域运行的 CA 的独立 KRA 或 OCSP。详情请查看 第 7.9 节 “设置独立 KRA 或 OCSP”。

7.2. 证书系统软件包

安装证书系统软件包时，您可以单独为每个子系统安装它们，也可以一次性安装它们。

重要

要安装和更新证书服务器软件包，您必须启用对应的存储库。详情请查看 第 6.8 节 “附加红帽订阅并启用证书系统软件包存储库”。

以下子系统软件包和组件在 Red Hat Certificate System 中提供：

- pki-ca: 提供证书颁发机构(CA)子系统。
- pki-kra ：提供密钥恢复授权机构(KRA)子系统。
- pki-ocsp ：提供在线证书状态协议(OCSP)响应器。
- pki-tks ：提供令牌密钥服务(TKS)。
- pki-tps ：提供令牌处理服务(TPS)。
- pki-console 和 redhat-pki-console-theme ：提供基于 Java 的红帽 PKI 控制台。必须安装这两个软件包。
- pki-server 和 redhat-pki-server-theme ：提供基于 Web 的证书系统接口。必须安装这两个

软件包。

如果您安装以下软件包之一，则这个软件包会被安装为依赖项： pki-ca,pki-kra,pki-ocsp,pki-tks,pki-tps

7.2.1. 安装证书系统软件包

- 使用 redhat-pki 模块，您可以在 RHEL 8 系统上一次性安装所有证书系统子系统软件包和组件。redhat-pki 模块会安装 Red Hat Certificate System 的五个子系统：除了 pki-core 模块 (CA、KRA)之外，它还是 Red Hat Identity Management (IdM)的一部分，包括特定于 RHCS 的子系统(OCSP、TKS 和 TPS)，以及处理所需依赖项的 pki-deps 模块。

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

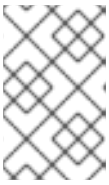
- 或者，您可以单独安装软件包。例如，要安装 CA 子系统和可选 Web 界面：

```
# yum install pki-ca redhat-pki-server-theme
```

对于其他子系统，将 pki-ca 软件包名称替换为您要安装的子系统之一。

- 如果您需要可选的 PKI 控制台：

```
# yum install pki-console redhat-pki-console-theme
```



注意

pkiconsole 工具将被弃用。

7.2.2. 更新证书系统软件包

要更新证书系统和操作系统软件包，请使用以下流程：

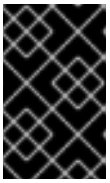
1. 按照 第 7.2.3 节 “确定证书系统产品版本” 中的说明检查产品版本。

2. 执行 ; yum update

以上命令更新整个系统，包括 RHCS 软件包。

注意

我们建议调度维护窗口，您可以在其中使 PKI 基础架构离线来安装更新。



重要

更新证书系统需要重启 PKI 基础架构。

3. 然后，使用以下 第 7.2.3 节 “确定证书系统产品版本” 再次检查版本。

版本号应该确认已成功安装了更新。

要选择性地在不安装的情况下下载更新，请使用以上流程中的 --downloadonly 选项：

```
yum update --downloadonly
```

下载的软件包存储在 /var/cache/yum/ 目录中。如果软件包是最新版本，yum update 将会使用这些软件包。

7.2.3. 确定证书系统产品版本

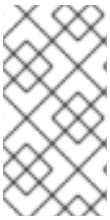
Red Hat Certificate System 产品版本存储在 /usr/share/pki/CS_SERVER_VERSION 文件中。显示版本：

```
# cat /usr/share/pki/CS_SERVER_VERSION
Red Hat Certificate System 10.0 (Batch Update 1)
```

要查找正在运行的服务器的产品版本，请从浏览器访问以下 URL：

- http://host_name:port_number/ca/admin/ca/getStatus

- *http://host_name:port_number/kra/admin/kra/getStatus*
- *http://host_name:port_number/ocsp/admin/ocsp/getStatus*
- *http://host_name:port_number/tks/admin/tks/getStatus*
- *http://host_name:port_number/tps/admin/tps/getStatus*



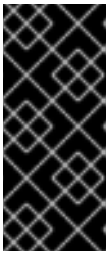
注意

请注意，每个组件都是一个单独的软件包，因此可以有单独的版本号。以上将显示每个当前运行的组件的版本号。

7.3. 了解 PKISPAWN 实用程序

在 Red Hat Certificate System 中，您可以使用 pkispawn 实用程序设置单个公钥基础架构(PKI)子系统。在设置过程中，pkispawn:

1. 从 /etc/pki/default.cfg 文件中读取默认值。详情请查看 pki_default.cfg(5) man page。



重要

不要编辑 /etc/pki/default.cfg 文件。反之，创建一个配置文件，并覆盖默认值，并将其传递给 pkispawn 工具。有关使用配置文件的详情，请参考 第 7.7 节 “两步安装”。

2. 根据设置模式，使用提供的密码和其他特定于部署的信息：
   - 交互模式 ：在设置过程中要求用户单独设置。将此模式用于简单部署。
   - 批处理模式：值从用户提供的配置文件中读取。配置文件中未设置的参数使用默认值。

3. 执行请求的 PKI 子系统的安装。

4. 将设置传递给根据设置执行配置的 Java servlet。

使用 pkispawn 工具安装：

- root CA。详情请查看 第 7.4 节 “设置根证书颁发机构”。
- 从属 CA 或任何其他子系统。详情请查看 第 7.6 节 “设置额外的子系统”。

注意

请参阅 第 7.4 节 “设置根证书颁发机构” 了解如何使用 pkispawn 工具设置根 CA。有关从属 CA 或非 CA 子系统的设置，请参阅 第 7.8 节 “使用外部 CA 设置子系统”。

有关 pkispawn 和示例的详情，请参考 pkispawn(8) man page。

7.4. 设置根证书颁发机构

证书颁发机构(CA)子系统是所有其他证书系统子系统的先决条件。因此，在配置其他子系统前设置 CA。

要在证书系统中设置 root CA，有以下选项：

- 基于配置文件的安装：

  使用此方法进行高级别自定义。这个安装方法使用配置文件来覆盖默认安装参数。

  您可以在一个步骤中使用配置文件或两个步骤安装证书系统。有关详情和示例，请参阅：

- 单步安装的 pkispawn(8) man page。
- 第 7.7 节 “两步安装” 对于两步安装。

- 交互式安装：

```
# pkispawn -s CA
```

如果您只想设置所需的配置选项，请使用交互式安装程序。

7.5. 安装后

按照以下步骤操作：

- 第 7.7.5.2 节 “启用签名的审计日志记录”
- 第 14.4.1.1 节 “TLS Cipher 配置”
- 第 14.4.1.3 节 “为子系统启用证书撤销检查”

完成以上步骤后，请按照 第 7.10 节 “安装后的任务” 进行额外的安装后操作。

7.6. 设置额外的子系统

安装 root 证书颁发机构(CA)后，如 第 7.4 节 “设置根证书颁发机构” 所述，您可以安装额外的证书系统子系统。

先决条件

所有其他子系统都需要 root 证书颁发机构(CA)。如果您还没有安装 root 证书系统 CA，请参阅 第 7.4 节 “设置根证书颁发机构”。

安装子系统

要设置额外的子系统，有以下选项：

- 基于配置文件的安装：

  使用此方法进行高级别自定义。这个安装方法使用配置文件来覆盖默认安装参数。

  您可以在一个步骤中使用配置文件或两个步骤安装证书系统。有关详情和示例，请参阅：

  - 单步安装的 pkispawn(8) man page。
  - 第 7.7 节 “两步安装” 对于两步安装。

- 交互式安装：

  如果您只想设置所需的配置选项，请使用交互式安装程序。

  例如：

  ```
  # pkispawn -s subsystem
  ```

  使用以下 子系统 之一替换 subsystem： KRA、OCSP、TKS 或 TPS。

  交互式安装程序不支持安装从属 CA。要安装从属 CA，请使用两个步骤安装。请参阅 第 7.7 节 “两步安装”。

7.7. 两步安装

要在安装过程中自定义某些配置参数，需要用两个步骤执行安装过程，并在它们之间进行配置。为此，pkispawn 工具可让您在两个步骤中运行子系统的安装。

7.7.1. 何时使用双步安装

在以下情况下使用两步安装：

- 提高安全性。
- 自定义子系统证书。
- 在安装要连接到现有证书系统的新证书系统实例时，自定义 /etc/pki/instance_name/server.xml 文件中的 sslRangeCiphers 参数中的 cipher 列表。
- 在 FIPS 模式中安装 CA 克隆、KRA、OCSP、TKS 和 TPS。
- 在 FIPS 模式下使用硬件安全模块(HSM)安装证书系统。

7.7.2. 这两个步骤安装的主要部分

两步安装由以下两个主要部分组成：

1. 安装

   在这一步中，pkispawn 将配置文件从 /usr/share/pki/ 目录复制到特定于实例的 /etc/pki/instance_name/ 目录。另外，pkispawn 根据部署配置文件中定义的值设置。

   这个安装部分包含以下子步骤：

   1. 第 7.7.3 节 “为安装的第一个步骤创建配置文件”
   2. 第 7.7.4 节 “启动安装步骤”

2. Configuration

在这一步中，pkispawn 根据特定于实例的 /etc/pki/instance_name/ 目录中的配置文件继续安装。

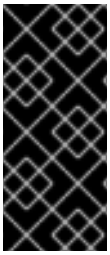
这个安装部分包含以下子步骤：

1. 第 7.7.5 节 “自定义安装步骤之间的配置”

2. 第 7.7.6 节 “启动配置步骤”

7.7.3. 为安装的第一个步骤创建配置文件

为配置设置创建一个文本文件，如 /root/config.txt，并使用下面描述的设置填充它。



重要

本节论述了在与证书系统相同的主机上运行的 Directory 服务器的最低配置。根据您的环境，可能需要其他参数。有关其他示例，请参阅 pkispawn(8) man page 中的 EXAMPLES 部分。

有关本部分涵盖参数的描述，请查看 pki_default.cfg(5) man page。

独立于子系统的设置

独立于您安装的子系统，配置文件中需要以下设置：

1. 设置证书系统 admin 用户的密码、PKCS TOTP 文件和目录服务器：

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

2. 要使用到在同一主机上运行的目录服务器的 LDAPS 连接，请在配置文件中的 [DEFAULT] 部分添加以下参数：

```
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
```

–

注意

为了安全起见，红帽建议使用到目录服务器的加密连接。

如果您在目录服务器中使用自签名证书，请使用以下命令从目录服务器的网络安全服务(NSS)数据库中导出它：

```
# certutil -L -d /etc/dirsrv/slapd-instance_name/ \
    -n "server-cert" -a -o /root/ds.crt
```

重要

默认情况下，证书系统在安装后删除 ~/.dogtag/instance_name/子系统/alias 客户端数据库。为了安全起见，红帽建议在配置文件中启用 pki_client_database_purge 参数。如果手动将此参数设置为 True，则证书系统在安装后不会删除客户端数据库。

创建初始配置文件后，向其中添加特定于子系统的设置。请参阅：

- “CA 设置”一节
- “其他子系统的设置”一节

CA 设置

除了 “独立于子系统的设置”一节 外，还需要以下设置来安装 CA：

1. 要提高安全性，请通过在配置文件中添加 [CA] 部分来启用随机序列号：

```
[CA]
pki_random_serial_numbers_enable=true
```

2. 另外，还可在 [CA] 部分中设置以下参数，以指定 admin 用户的数据，这将在安装过程中自动创建：

```
pki_admin_nickname=caadmin
```

```
pki_admin_name=CA administrator account
pki_admin_password=password
pki_admin_uid=caadmin
pki_admin_email=caadmin@example.com
```

证书系统为这个帐户分配管理员特权。安装后使用此帐户来管理证书系统并创建更多用户帐户。

3.

要启用证书系统生成唯一的 nickname，请在 [DEFAULT] 部分中设置以下参数：

```
pki_instance_name=instance_name
pki_security_domain_name=example.com Security Domain
pki_host=server.example.com
```

重要

如果使用网络共享硬件安全模块(HSM)安装证书系统，则必须使用唯一的证书 nickname。

4.

（可选）在生成证书时使用 Elliptic Curve Curve Cryptography (ECC)而不是 RSA：

a.

在 [DEFAULT] 部分添加以下参数：

```
pki_admin_key_algorithm=SHA256withEC
pki_admin_key_size=nistp256
pki_admin_key_type=ecc
pki_sslserver_key_algorithm=SHA256withEC
pki_sslserver_key_size=nistp256
pki_sslserver_key_type=ecc
pki_subsystem_key_algorithm=SHA256withEC
pki_subsystem_key_size=nistp256
pki_subsystem_key_type=ecc
```

b.

在 [CA] 部分添加以下参数：

```
pki_ca_signing_key_algorithm=SHA256withEC
pki_ca_signing_key_size=nistp256
pki_ca_signing_key_type=ecc
pki_ca_signing_signing_algorithm=SHA256withEC
pki_ocsp_signing_key_algorithm=SHA256withEC
```

```
pki_ocsp_signing_key_size=nistp256
pki_ocsp_signing_key_type=ecc
pki_ocsp_signing_signing_algorithm=SHA256withEC
```

c. 在 [CA] 部分添加以下参数以使用 ECC 配置集覆盖 RSA 配置集：

```
pki_source_admincert_profile=/usr/share/pki/ca/conf/eccAdminCert.profile
pki_source_servercert_profile=/usr/share/pki/ca/conf/eccServerCert.profile
pki_source_subsystemcert_profile=/usr/share/pki/ca/conf/eccSubsystemCert.profil
e
```

其他子系统的设置

除了 “独立于子系统的设置”一节 外，还需要以下设置来安装从属 CA、KRA、OCSP、TKS 或 TPS：

1. 在您的配置文件的 [DEFAULT] 部分添加以下条目：

```
pki_client_database_password=password
```

2. 如果要安装 TPS：

a. 使用以下部分添加以下部分：

```
[TPS]
pki_authdb_basedn=basedn_of_the_TPS_authentication_database
```

b. 另外，要配置 TPS 使用在共享 CA 实例中已安装的 KRA 的服务器端密钥生成，请将以下条目添加到 [TPS] 部分：

```
pki_enable_server_side_keygen=True
```

7.7.4. 启动安装步骤

按照 第 7.7.3 节 “为安装的第一个步骤创建配置文件” 所述准备配置文件后，开始安装的第一个步骤：

```
# pkispawn -f /root/config.txt -s subsystem --skip-configuration
```

使用以下 子系统 之一替换 subsystem： CA、KRA、OCSP、TKS 或 TPS。

7.7.5. 自定义安装步骤之间的配置

在 第 7.7.4 节 “启动安装步骤” 中描述的安装步骤成功完成后，您可以在实际配置开始前手动更新特定于实例的配置文件。本节提供了您可以在安装的第一步和第二步间自定义的内容。

7.7.5.1. 配置证书配置文件

在很多情况下，站点希望自定义某些证书注册配置文件（例如更改证书的默认有效期时间），或添加/删除某些配置文件。有关选项的完整列表，请参阅 第 16 章 证书配置文件配置。

7.7.5.2. 启用签名的审计日志记录

签名的审计日志记录功能可检测未经授权的日志操作。详情请查看 第 18.3.1 节 “启用并配置签名的审计日志”。

7.7.5.3. 更新 Ciphers 列表

在某些情况下，管理员希望更新密码列表。例如：

- 保护证书系统实例
- 要安装证书系统实例，并将其添加到只支持特定密码的现有站点

有关更新密码列表的详情，请查看 Red Hat Certificate System Administration Guide 中的对应部分。

RSA 加密的默认 FIPS 模式启用 Ciphers

默认情况下，证书系统包含以下为 RSA 加密启用了 FIPS 模式的密码：

- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

- *TLS_DHE_RSA_WITH_AES_256_CBC_SHA*
- *TLS_DHE_RSA_WITH_AES_256_CBC_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_128_CBC_SHA*
- *TLS_RSA_WITH_AES_128_CBC_SHA256*
- *TLS_RSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_256_CBC_SHA256*

ECC 加密的默认 FIPS 模式启用 Ciphers

默认情况下，证书系统包含以下为 Elliptic Curve Curve Cryptography (ECC)加密启用了 FIPS 模式的密码：

- *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256*
- *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256*

- *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_256_CBC_SHA*
- *TLS_RSA_WITH_AES_256_CBC_SHA256*

在启用了 FIPS 模式的系统中运行 HSM 时所需的 RSA 密码

如果您在为 RSA 启用 FIPS 模式的系统中安装带有 LunaSA 或硬件安全模块(HSM)的证书系统，请禁用以下密码，因为它们在 HSMs 上不被支持：

- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
- *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA*

7.7.5.4. 配置 PKI 控制台超时

有关配置 PKI 控制台超时的详情，请参考 第 14.4.2 节 “会话超时” 中的相应部分。

7.7.5.5. 将 KRA 设置为加密模式

如果您使用硬件安全模块(HSM)，在某些情况下需要把密钥恢复授权(KRA)设置为加密模式。详情请查看 “将 KRA 设置为加密模式”一节。

7.7.5.6. 启用 OCSP

有关启用在线证书状态协议(OCSP)的详情，请参考 第 14.4.1.3 节 “为子系统启用证书撤销检查”。

7.7.5.7. 为请求和序列号配置范围

有关为请求和序列号配置范围的详情，请参考 第 14.2.3.14 节 “为请求和序列号配置范围”。

7.7.6. 启动配置步骤

根据 第 7.7.5 节 “自定义安装步骤之间的配置” 自定义配置文件后，开始安装的第二个步骤：

```
# pkispawn -f /root/config.txt -s subsystem --skip-installation
```

使用以下 子系统 之一替换 **subsystem**： **CA**、**KRA**、**OCSP**、**TKS** 或 **TPS**。

如果配置步骤成功，**pkispawn** 工具会显示安装概述。例如：

```
==========================================================================
                      INSTALLATION SUMMARY
==========================================================================

  Administrator's username:             caadmin
  Administrator's PKCS #12 file:
      /root/.dogtag/instance_name/ca_admin_cert.p12

  To check the status of the subsystem:
      systemctl status pki-tomcatd@instance_name.service

  To restart the subsystem:
      systemctl restart pki-tomcatd@instance_name.service

  The URL for the subsystem is:
      https://server.example.com:8443/ca/

  PKI instances will be enabled upon system boot

==========================================================================
```

7.7.7. 安装后

完成以上步骤后，请按照 第 7.10 节 “安装后的任务” 进行额外的安装后操作。

7.8. 使用外部 CA 设置子系统

7.8.1. 内部和外部 CA 间的差异

在 **Red Hat Certificate System** 中，当 **pkispawn** 实用程序将子系统证书签名请求(CSR)发送到之前安装的证书系统时，生成的发布的证书会被 **pkispawn** 接收和使用，**CA CSR** 发送到内部 **CA**。

相反，外部 CA 可以是以下之一：

- 为证书系统从属 CA 发布签名证书的非红帽证书系统 CA。
- 以前安装的 Red Hat Certificate System CA 不允许直接提交 CSR。例如，如果您的环境需要来自从属 CA、KRA、OCSP、TKS 或 TPS 的 CSR，则情况是，采用 PKCS #10 以外的其他格式。

7.8.2. 使用外部 CA 安装子系统

这部分论述了如何设置从属 CA 或其他由外部 CA 签名的子系统。

为外部 CA 安装准备配置文件

根据您希望将子系统集成到证书系统还是独立性准备配置文件：

- 如果您安装了一个子系统，它集成到现有的证书系统安装中，但使用外部 CA 签名的证书：
  1. 为子系统创建配置文件。请参阅 第 7.7.3 节 “为安装的第一个步骤创建配置文件”。
  2. 在相应的 [CA]、[KRA] 或 [OCSP] 部分的配置文件中添加以下设置：
     - 对于 CA 安装：

```
[CA]
pki_external=True
pki_external_step_two=False

pki_ca_signing_csr_path=path/to/ca_signing.csr
pki_ca_signing_cert_path=path/to/ca_signing.crt
```

     - 对于 KRA 安装：

```
[KRA]
pki_external=True
```

```
pki_external_step_two=False

pki_storage_csr_path=/home/user_name/kra_storage.csr
pki_transport_csr_path=/home/user_name/kra_transport.csr
pki_subsystem_csr_path=/home/user_name/subsystem.csr
pki_sslserver_csr_path=/home/user_name/sslserver.csr
pki_audit_signing_csr_path=/home/user_name/kra_audit_signing.csr
pki_admin_csr_path=/home/user_name/kra_admin.csr
```

- 对于 OCSP 安装：

```
[OCSP]
pki_external=True
pki_external_step_two=False

pki_ocsp_signing_csr_path=/home/user_name/ocsp_signing.csr
pki_subsystem_csr_path=/home/user_name/subsystem.csr
pki_sslserver_csr_path=/home/user_name/sslserver.csr
pki_audit_signing_csr_path=/home/user_name/ocsp_audit_signing.csr
pki_admin_csr_path=/home/user_name/ocsp_admin.csr
```

- 如果您安装了一个没有集成到现有证书系统安装中的独立 KRA 或 OCSP，请执行 第 7.9 节 “设置独立 KRA 或 OCSP” 中介绍的步骤。

使用外部 CA 启动子系统安装

使用配置文件开始安装：

1. 使用 pkispawn 工具开始安装：

```
# pkispawn -f /root/config.txt -s subsystem
```

使用您要安装的 子系统 替换 subsystem： CA、KRA 或 OCSP。

在这一步中，设置会将 CSR 存储在配置中指定的文件中。

2. 将 CSR 提交给外部 CA。在 CA 发布相应证书后继续。

在某些环境中，如果外部 CA 也是证书系统实例，在提交到 CA 前，需要把 PKCS#10 格式的 CSR 转换为 CMC 格式。有关 发布证书的详细信息，请参阅 Red Hat Certificate System

Administration Guide 中的使用 CMC 的颁发证书部分。

3. *（可选）自定义安装。详情请查看 第 7.7.5 节 “自定义安装步骤之间的配置”。*

4. *在外部 CA 发布证书后，编辑部署配置文件：*

   a. *将 pki_external_step_two 设置为 True ：*

   ```
   pki_external_step_two=True
   ```

   b. *根据您要安装的子系统，添加以下参数：*

   - *对于 CA，设置证书文件的路径。例如：*

     ```
     pki_ca_signing_cert_path=/home/user_name/ca_signing.crt
     ```

     如果指定文件不包含包含证书链的证书，还要指定证书链文件的路径及其 nickname。例如：

     ```
     pki_cert_chain_path=/home/user_name/cert_chain.p7b
     pki_cert_chain_nickname=CA Signing Certificate
     ```

   - *对于 KRA，设置证书文件的路径。例如：*

     ```
     pki_storage_cert_path=/home/user_name/kra_storage.crt
     pki_transport_cert_path=/home/user_name/kra_transport.crt
     pki_subsystem_cert_path=/home/user_name/subsystem.crt
     pki_sslserver_cert_path=/home/user_namesslserver.crt
     pki_audit_signing_cert_path=/home/user_name/kra_audit_signing.crt
     pki_admin_cert_path=/home/user_name/kra_admin.crt
     ```

     如果指定文件不包含包含证书链的证书，还要指定签名证书文件的路径，以及证书链文件及其 nickname。例如：

     ```
     pki_ca_signing_nickname=CA Signing Certificate
     pki_ca_signing_cert_path=/home/user_name/ca_signing.crt
     pki_cert_chain_nickname=External Certificate Chain
     ```

```
pki_cert_chain_path=/home/user_name/cert_chain.p7b
```

- 对于 OCSP，设置证书文件的路径。例如：

```
pki_ocsp_signing_cert_path=/home/user_name/ocsp_signing.crt
pki_subsystem_cert_path=/home/user_name/subsystem.crt
pki_sslserver_cert_path=/home/user_name/sslserver.crt
pki_audit_signing_cert_path=/home/user_name/ocsp_audit_signing.crt
pki_admin_cert_path=/home/user_name/ocsp_admin.crt
```

如果指定文件不包含包含证书链的证书，还要指定签名证书文件的路径，以及证书链文件及其 nickname。例如：

```
pki_ca_signing_nickname=CA Signing Certificate
pki_ca_signing_cert_path=/home/user_name/ca_signing.crt
pki_cert_chain_nickname=External Certificate Chain
pki_cert_chain_path=/home/user_name/cert_chain.p7b
```

5. （可选）自定义配置文件。例如，请参阅 第 7.7.5 节 “自定义安装步骤之间的配置”。

6. 启动配置步骤：

```
# pkispawn -f /root/config.txt -s subsystem
```

使用您要安装的 子系统 替换 subsystem： CA、KRA 或 OCSP。

7.8.3. 安装后

完成以上步骤后，请按照 第 7.10 节 “安装后的任务” 进行额外的安装后操作。

7.9. 设置独立 KRA 或 OCSP

这部分论述了如何安装独立 KRA 和 OCSP。独立安装提供了使用非证书系统 CA 发布证书的灵活性，因为在安装过程中生成的 CSR 不会自动提交到 CA 并导入到子系统中。另外，在独立模式下安装的 KRA 或 OCSP 不是 CA 的安全域的一部分，不会配置密钥归档的 CA 中的连接器。

安装独立 KRA 或 OCSP：

1. 创建包含以下内容的配置文件，如 /root/config.txt ：

```
[DEFAULT]
pki_admin_password=password
pki_client_database_password=password
pki_client_pkcs12_password=password
pki_ds_password=password
pki_token_password=password
pki_client_database_purge=False
pki_security_domain_name=EXAMPLE

pki_standalone=True
pki_external_step_two=False
```

2. 对于独立 KRA，请在配置文件中添加以下部分：

```
[KRA]
pki_admin_email=kraadmin@example.com
pki_ds_base_dn=dc=kra,dc=example,dc=com
pki_ds_database=kra

pki_admin_nickname=kraadmin
pki_audit_signing_nickname=kra_audit_signing
pki_sslserver_nickname=sslserver
pki_storage_nickname=kra_storage
pki_subsystem_nickname=subsystem
pki_transport_nickname=kra_transport

pki_standalone=True
```

3. 对于独立的 OCSP，请在配置文件中添加以下部分：

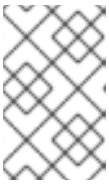
```
[OCSP]
pki_admin_email=ocspadmin@example.com
pki_ds_base_dn=dc=ocsp,dc=example,dc=com
pki_ds_database=ocsp

pki_admin_nickname=ocspadmin
pki_audit_signing_nickname=ocsp_audit_signing
pki_ocsp_signing_nickname=ocsp_signing
pki_sslserver_nickname=sslserver
pki_subsystem_nickname=subsystem

pki_standalone=True
```

4. 要使用到在同一主机上运行的目录服务器的 LDAPS 连接，请在配置文件中的 DEFAULT 部分添加以下参数：

```
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
```



注意

为了安全起见，红帽建议使用到目录服务器的加密连接。

如果您在目录服务器中使用自签名证书，请使用以下命令从目录服务器的网络安全服务(NSS)数据库中导出它：

```
# certutil -L -d /etc/dirsrv/slapd-instance_name/ \
    -n "server-cert" -a -o /root/ds.crt
```

5. 继续执行 “使用外部 CA 启动子系统安装”一节 中描述的步骤。

7.10. 安装后的任务

使用 pkispawn 工具安装完成后，可以采取进一步的步骤来自定义配置，具体取决于站点的首选项。它们在 第 III 部分 “配置证书系统” 中进行了描述。

本节提供了 第 III 部分 “配置证书系统” 的操作列表，用于提高部署的安全性。

7.10.1. 为 RHCS 设置日期/时间

运行 RHCS 务必要正确设置时间；请参阅 Red Hat Certificate System Administration Guide 中的 设置时间和日期 部分。

7.10.2. 替换临时证书

注意

本节需要安装并运行 root CA。安装完成后，这些步骤将执行。

以下流程描述了将临时自签名 Directory 服务器证书替换为新安装的 CA 发布的永久目录服务器证书，或者将旧的 Directory 服务器证书替换为一个新的目录服务器证书。

1. 获取新的目录服务器服务器证书。提交 CA 签名的新证书的请求。要使用 CMC 方法获取新的目录服务器证书，请参阅 Red Hat Certificate System Administration Guide 中的 Obtaining System and Server Certificates 部分。

   在上一节中，按照指导创建 TLS 服务器证书。创建证书后，必须将其重新导入到目录服务器的证书数据库中。

2. 在访问 NSS 数据库前停止 Directory 服务器实例：

   ```
   # systemctl stop dirsrv@instance_name
   ```

3. 删除旧的 Directory 服务器证书：

   ```
   # certutil -F -d /etc/dirsrv/slapd-instance_name -f
   /etc/dirsrv/slapd-instance_name/password.txt -n "DS Certificate"
   ```

4. 导入之前下载的 CA 证书：

   ```
   # PKICertImport -d /etc/dirsrv/slapd-instance_name -f
   /etc/dirsrv/slapd-instance_name/password.txt -n "CA Certificate" -t "CT,C,C" -a -i ca.crt
   -u L
   ```

5. 导入之前下载的新目录服务器证书：

   ```
   # PKICertImport -d /etc/dirsrv/slapd-instance_name -f
   /etc/dirsrv/slapd-instance_name/password.txt -n "DS Certificate" -t ",," -a -i ds.crt -u V
   ```

   另请参阅 第 15.4 节 “将证书导入到 HSM 中”。

6. 启动 Directory 服务器实例：

   ```
   # systemctl start dirsrv@instance_name
   ```

7. 从 PKI CA 中删除旧目录服务器证书：

a. 停止证书系统实例：

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

b. 删除证书：

```
# certutil -D -d /var/lib/pki/instance_name/alias/ -n "DS Certificate"
```

c. 启动证书系统实例：

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

8. 验证新的 Directory 服务器证书是否由 NSS 数据库中安装的 CA 签名：

a. 显示目录服务器证书

```
$ certutil -L -d /etc/dirsrv/slapd-instance_name -n "DS Certificate"

Issuer: "CN=CA Signing Certificate,O=EXAMPLE"
Subject: "CN=server.example.com"
```

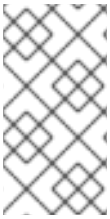
b. 验证 PKI NSS 数据库中不再存在旧的 Directory 服务器证书：

```
$ certutil -L -d /var/lib/pki/instance_name/alias
```

c. 验证证书系统可以使用新证书连接到目录服务器：

```
$ pki cert-find
```

7.10.3. 启用 TLS 客户端身份验证



注意

本节需要安装并运行 root CA。如果您使用临时 LDAP 服务器证书，请首先将其替换为以下 第 7.10.2 节 “替换临时证书”。安装完成后，这些步骤将执行。

如果您选择启用 TLS 客户端身份验证，则当基本 TLS 服务器身份验证在安装 CS 子系统时，我们现在可以重复运行，并尝试从给定子系统启用客户端身份验证到 LDAP 服务器。

设置客户端身份验证有两个部分。第一部分将 LDAP 目录配置为需要 TLS 相互身份验证。此流程在红帽目录服务器管理指南中的使用基于证书的客户端身份验证 中详细介绍。

注意以下几点：

- pkispawn 已在其内部目录服务器上自动创建 pkidbuser，其中 CS 实例的"子系统证书"（如 subsystemCert cert-pki-ca）存储在用户条目中。因此，不需要为 TLS 客户端身份验证创建另一个 LDAP 用户或其他证书。

- 为 /etc/dirsrv/slapd-instance_name/certmap.conf 创建内容时，请使用以下格式：

```
certmap rhcs <certificate issuer DN>
 rhcs:CmapLdapAttr seeAlso
 rhcs:verifyCert on
```

例如：

```
certmap rhcs CN=CA Signing Certificate,OU=pki-tomcat-ca,O=pki-tomcat-ca-SD
 rhcs:CmapLdapAttr seeAlso
 rhcs:verifyCert on
```

- 配置后，重启 Directory 服务器。

第二个部分是向 Red Hat Certificate System 实例添加配置，以便它知道使用 TLS mutual 身份验证与其内部 LDAP 服务器通信的端口和证书。这包括编辑位于 < instance 目录> / <subsystem type> /conf/ CS.cfg 的 RHCS 实例的 CS.cfg 文件。例如 /var/lib/pki/instance_name/ca/conf/CS.cfg

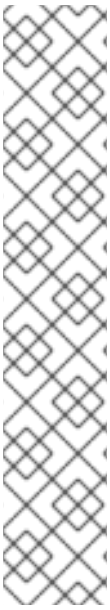
在 CS.cfg 中，请将 RHCS 实例的子系统证书别名添加到 internaldb.ldapauth.clientCertNickname，并删除两个未使用的条目：

```
internaldb.ldapauth.bindDN
 internaldb.ldapauth.bindPWPrompt
```

如下所示：

```
internaldb._000=##
 internaldb._001=## Internal Database
 internaldb._002=##
 internaldb.basedn=o=pki-tomcat-ca-SD
 internaldb.database=pki-tomcat-ca
 internaldb.maxConns=15
 internaldb.minConns=3
 internaldb.ldapauth.authtype=SslClientAuth
 internaldb.ldapauth.clientCertNickname=HSM-A:subsystemCert pki-tomcat-ca
 internaldb.ldapconn.host=example.com
 internaldb.ldapconn.port=11636
 internaldb.ldapconn.secureConn=true
```

在安装后步骤结束时重启 CS 实例。



注意

internaldb.basedn 和 internaldb.database 参数必须配置为与特定的 LDAP 实例匹配。

为合规，必须设置 internaldb.ldapauth.authtype=SslClientAuth 和 internaldb.ldapconn.secureConn=true，并且 internaldb.ldapauth.clientCertNickname 的值必须与 TLS 客户端证书的 nickname 匹配，以便在 NSS DB 中针对 LDAP 进行身份验证。

所有其他值都可以根据需要更改，以反映您的环境或可用性需求。

7.10.4. 配置会话超时

系统中存在各种超时配置，可能会影响在终止前允许 TLS 会话保持闲置的时间。详情请查看 第 14.4.2 节 “会话超时”。

7.10.5. CRL 或 Certificate Publishing

CRL 发布对于提供 OCSP 服务至关重要。证书发布是可选的，但通常被站点需要。详情请参阅 Red Hat Certificate System Administration Guide 中的发布证书 和 CRL 部分。

7.10.6. 配置证书注册配置文件(CA)

RHCS 具有丰富的配置文件框架，允许自定义证书注册配置文件。站点非常常见，以启用/禁用系统附带的默认配置集，或者修改现有配置集或创建自己的配置集。详情请查看 第 16 章 证书配置文件配置。

7.10.7. 启用访问横幅

要启用用户界面横幅，请参阅 第 14.7.1 节 “启用访问横幅”。

7.10.8. 启用 Watchdog 服务

watchdog (nuxwdog)服务提供安全的系统密码管理。详情请查看 第 14.3.2.1 节 “启用 Watchdog 服务”。

7.10.9. CMC Enrollment 和 Revocation (CA)的配置

证书注册和撤销可以通过 CMC 来完成。

- 有关启用 CMC 共享文件系统服务功能的详情，请参考 第 14.8.3 节 “启用 CMC 共享 Secret 功能”。
- 有关启用 PopLinkWittness 功能的详情，请参考 第 14.8.2 节 “启用 PopLinkWittnessV2 功能”。
- 有关为 Web 用户界面启用 CMCRevoke 的详情，请参考 第 14.8.4 节 “为 Web 用户界面启用 CMCRevoke”。

7.10.10. Java 控制台的 TLS 客户端身份验证

要要求证书系统管理员在登录到 Java 控制台时提供用户 TLS 客户端证书，请参阅 第 14.2.3.15 节 “为 pkiconsole 设置要求以使用 TLS 客户端证书验证”。

注意

pkiconsole 已被弃用。

7.10.11. 创建角色用户

创建实际角色用户，以便您可以删除 bootstrap 用户。

要创建用户并将其分配到不同的特权角色来管理证书系统，请参阅 第 19 章 创建角色用户。

7.10.12. 删除 Bootstrap 用户

创建真实角色用户后，不再需要在安装过程中自动创建的 bootstrap 用户。要删除此帐户，请参阅 第 20 章 删除 Bootstrap 用户，确定您创建了一个分配给个人个人的新管理员帐户。

7.10.13. 禁用多角色支持

要在 bootstrap 用户被删除后禁用多角色支持，请参阅 第 20.1 节 “禁用多角色支持”。

7.10.14. KRA 配置

7.10.14.1. 为密钥恢复授权(KRA)添加多个代理批准要求

要设置多个 KRA 代理的要求以批准密钥恢复，请参阅 Red Hat Certificate System Administration Guide 中的 Command Line 部分中的配置 Agent-Approved Key Recovery。

7.10.14.2. 配置 KRA 加密设置

要配置密钥加密/包装算法，请参阅 第 17.2 节 “加密 Of KRA 操作”。

7.10.15. 设置用户以使用用户界面

在用户可以使用批准的用户界面之前，需要执行初始化。需要管理用户（管理角色或其他用户）来设置其客户端访问用户界面的客户端。请参阅 Red Hat Certificate System Administration Guide 中的 Client NSS Database Initialization 部分。

第 8 章 为子系统安全数据库使用硬件安全模块

子系统实例生成其密钥信息并将其存储在密钥存储中，称为 安全模块 或令牌。可以为要生成的密钥配置子系统实例，并使用内部 NSS 令牌或单独的加密设备（硬件令牌）存储。

8.1. 使用 HSM 安装证书系统

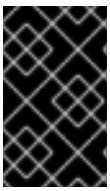
在使用 HSM 安装证书系统时，请使用您传递给 pkispawn 工具的配置文件中的以下参数：

```
[DEFAULT]
############################
# Provide HSM parameters #
############################
pki_hsm_enable=True
pki_hsm_libfile=hsm_libfile
pki_hsm_modulename=hsm_modulename
pki_token_name=hsm_token_name
pki_token_password=pki_token_password

########################################
# Provide PKI-specific HSM token names #
########################################
pki_audit_signing_token=hsm_token_name
pki_ssl_server_token=hsm_token_name
pki_subsystem_token=hsm_token_name
```

8.2. 使用带有子系统的硬件安全模块

证书系统默认支持 nCipher nShield Connect XC 硬件安全模块(HSM)和 Thales Luna HSM。如果在安装的预配置阶段，使用 modutil 命令在指定的安装路径中使用 modutil 命令自动添加到 pkcs11.txt 数据库中的证书系统支持的 HSM。



重要

某些部署需要设置其 HSM 以使用 FIPS 模式。

8.2.1. 在 HSM 中启用 FIPS 模式

要在 HSM 上启用 FIPS 模式，请参阅 HSM 供应商文档。

重要

nCipher HSM

在 nCipher HSM 中，只有生成 Security World 时才能启用 FIPS 模式，之后无法更改它。虽然有各种生成 Security World 的方法，但首选的方法是使用 new-world 命令。有关如何生成 FIPS 兼容安全世界的指导，请按照 nCipher HSM 供应商的文档进行操作。

LunaSA HSM

同样，必须在 Luna HSM 上启用 FIPS 模式，因为更改此策略会将 HSM 归类为安全措施。详情请查看 Luna HSM 供应商的文档。

8.2.2. 验证 HSM 上是否启用了 FIPS 模式

本节描述了如何为某些 HSM 验证是否启用了 FIPS 模式。有关其他 HSM，请查看硬件制造商的文档。

8.2.2.1. 验证 nCipher HSM 中是否启用了 FIPS 模式

注意

有关完整的流程，请参阅 HSM 供应商的文档。

要验证在 nCipher HSM 上是否启用了 FIPS 模式，请输入：

```
# /opt/nfast/bin/nfkminfo
```

在使用旧版本的软件时，如果 StrictFIPS140 列在 state 标记中，则启用 FIPS 模式。在较新的版本中，最好检查新模式行并查找 fips1402level3。在所有情况下，nfkminfo 输出中也应该有一个 hkfips 密钥。

8.2.2.2. 验证 Luna SA HSM 上是否启用了 FIPS 模式

注意

有关完整的流程，请参阅 HSM 供应商的文档。

验证 Luna SA HSM 上是否启用了 FIPS 模式：

1. *打开 lunash 管理控制台*

2. *使用 hsm show 命令，并验证输出中是否包含文本 The HSM in FIPS 140-2 approved operation mode。*

```
lunash:> hsm show
...
   FIPS 140-2 Operation:
   ====================
   The HSM is in FIPS 140-2 approved operation mode.
...
```

8.2.3. 为子系统添加或管理 HSM 条目

在安装过程中，当将 HSM 选为默认令牌，其中传递给 pkispawn 命令的配置文件中设置适当的 HSM 特定参数，则会将以下参数添加到 HSM 密码的 /var/lib/pki/instance_name/conf/password.conf 文件中：

```
hardware-HSM_token_name=HSM_token_password
```

8.2.4. 为 HSM 设置 SELinux

如果要使用硬件安全模块(HSM)和 SELinux 安装证书系统以 enforcing 模式运行，某些 HSM 需要在您可以安装证书系统前手动更新 SELinux 设置。

下面的部分描述了支持的 HSMs 所需的操作：

nCipher nShield Connect XC

安装 HSM 并在开始安装证书系统前：

1. *重置 /opt/nfast/ 目录中文件的上下文：*

```
# restorecon -R /opt/nfast/
```

2. 重新启动 nfast 软件。

```
# /opt/nfast/sbin/init.d-ncipher restart
```

Thales Luna HSM

在开始安装证书系统前，不需要与 SELinux 相关的操作。

8.2.5. 使用 nCipher nShield Connect XC HSM 安装子系统

要安装使用 nCipher nShield Connect XC HSM 的子系统实例，请按照以下步骤执行：

1. 准备一个覆盖文件，对应于您的特定部署。以下 default_hms.txt 文件是一个此类覆盖文件的示例：

注意

此文件仅充当 nCipher HSM 覆盖配置文件的示例，可以覆盖大量其他值，包括默认哈希算法。另外，根据 pkispawn 命令行中指定的子系统调用，将只使用 [CA]、[KRA]、[TKS] 或 [TPS] 部分之一。

例 8.1. 使用 nCipher HSM 的覆盖文件示例

```
##############################################################################
#######
##############################################################################
#######
##############################################################################
#######
##                                                          ##
## EXAMPLE: Configuration File used to override '/etc/pki/default.cfg'      ##
##          when using an nCipher Hardware Security Module (HSM):           ##
##                                                          ##
##                                                          ##
##     # modutil -dbdir . -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. nfast                                                     ##
##          library name: /opt/nfast/toolkits/pkcs11/libcknfast.so  ##
##           slots: 2 slots attached                                ##
##          status: loaded                                          ##
##                                                                  ##
##           slot: <serial_number> Rt1                              ##
##          token: accelerator                                      ##
##                                                                  ##
##           slot: <serial_number> Rt1 slot 0                       ##
##          token: <HSM_token_name>                                 ##
##    ------------------------------------------------------        ##
##                                                                  ##
##                                                                  ##
##    Based on the example above, substitute all password values,   ##
##    as well as the following values:                              ##
##                                                                  ##
##       <hsm_libfile>=/opt/nfast/toolkits/pkcs11/libcknfast.so     ##
##       <hsm_modulename>=nfast                                     ##
##       <hsm_token_name>=NHSM-CONN-XC                              ##
##                                                                  ##
######################################################################
#######
######################################################################
#######
######################################################################
#######

[DEFAULT]
############################
# Provide HSM parameters #
############################
pki_hsm_enable=True
pki_hsm_libfile=<hsm_libfile>
pki_hsm_modulename=<hsm_modulename>
pki_token_name=<hsm_token_name>
pki_token_password=<pki_token_password>

########################################
# Provide PKI-specific HSM token names #
########################################
pki_audit_signing_token=<hsm_token_name>
pki_ssl_server_token=<hsm_token_name>
pki_subsystem_token=<hsm_token_name>

###################################
# Provide PKI-specific passwords #
###################################
pki_admin_password=<pki_admin_password>
pki_client_pkcs12_password=<pki_client_pkcs12_password>
pki_ds_password=<pki_ds_password>
```

```
#####################################
# Provide non-CA-specific passwords #
#####################################
pki_client_database_password=<pki_client_database_password>

#####################################################################
# ONLY required if specifying a non-default PKI instance name #
#####################################################################
#pki_instance_name=<pki_instance_name>

#####################################################################
# ONLY required if specifying non-default PKI instance ports #
#####################################################################
#pki_http_port=<pki_http_port>
#pki_https_port=<pki_https_port>

#############################################################################
# ONLY required if specifying non-default 389 Directory Server ports #
#############################################################################
#pki_ds_ldap_port=<pki_ds_ldap_port>
#pki_ds_ldaps_port=<pki_ds_ldaps_port>

#############################################################################
# ONLY required if PKI is using a Security Domain on a remote system #
#############################################################################
#pki_ca_hostname=<pki_ca_hostname>
#pki_issuing_ca_hostname=<pki_issuing_ca_hostname>
#pki_issuing_ca_https_port=<pki_issuing_ca_https_port>
#pki_security_domain_hostname=<pki_security_domain_hostname>
#pki_security_domain_https_port=<pki_security_domain_https_port>

#############################################################
# ONLY required for PKI using an existing Security Domain #
#############################################################
# NOTE:  pki_security_domain_password == pki_admin_password
#        of CA Security Domain Instance
#pki_security_domain_password=<pki_admin_password>


[Tomcat]
#####################################################################
# ONLY required if specifying non-default PKI instance ports #
#####################################################################
#pki_ajp_port=<pki_ajp_port>
#pki_tomcat_server_port=<pki_tomcat_server_port>


[CA]
#########################################
# Provide CA-specific HSM token names #
#########################################
pki_ca_signing_token=<hsm_token_name>
pki_ocsp_signing_token=<hsm_token_name>

#############################################################################
```

```
###
# ONLY required if 389 Directory Server for CA resides on a remote system #
##############################################################################
###
#pki_ds_hostname=<389 hostname>


[KRA]
########################################
# Provide KRA-specific HSM token names #
########################################
pki_storage_token=<hsm_token_name>
pki_transport_token=<hsm_token_name>

##############################################################################
####
# ONLY required if 389 Directory Server for KRA resides on a remote system #
##############################################################################
####
#pki_ds_hostname=<389 hostname>


[OCSP]
########################################
# Provide OCSP-specific HSM token names #
########################################
pki_ocsp_signing_token=<hsm_token_name>

##############################################################################
#####
# ONLY required if 389 Directory Server for OCSP resides on a remote system #
##############################################################################
#####
#pki_ds_hostname=<389 hostname>


[TKS]
########################################
# Provide TKS-specific HSM token names #
########################################

##############################################################################
####
# ONLY required if 389 Directory Server for TKS resides on a remote system #
##############################################################################
####
#pki_ds_hostname=<389 hostname>


[TPS]
####################################
# Provide TPS-specific parameters #
####################################
pki_authdb_basedn=<dnsdomainname where hostname.b.c.d is dc=b,dc=c,dc=d>

########################################
```

```
# Provide TPS-specific HSM token names #
########################################

#########################################################################
####
# ONLY required if 389 Directory Server for TPS resides on a remote system #
#########################################################################
####
#pki_ds_hostname=<389 hostname>

############################################################
# ONLY required if TPS requires a CA on a remote machine #
############################################################
#pki_ca_uri=https://<pki_ca_hostname>:<pki_ca_https_port>

########################################
# ONLY required if TPS requires a KRA #
########################################
#pki_enable_server_side_keygen=True

#################################################################
# ONLY required if TPS requires a KRA on a remote machine #
#################################################################
#pki_kra_uri=https://<pki_kra_hostname>:<pki_kra_https_port>

#################################################################
# ONLY required if TPS requires a TKS on a remote machine #
#################################################################
#pki_tks_uri=https://<pki_tks_hostname>:<pki_tks_https_port>
```

2. **使用配置文件，如 第 7.7 节 “两步安装” 所述。**

- `# pkispawn -s CA -f ./default_hsm.txt -vvv`
- `# pkispawn -s KRA -f ./default_hsm.txt -vvv`
- `# pkispawn -s OCSP -f ./default_hsm.txt -vvv`
- `# pkispawn -s TKS -f ./default_hsm.txt -vvv`
- `# pkispawn -s TPS -f ./default_hsm.txt -vvv`

3. *验证 **HSM 是否包含以下证书：***

```
$ certutil -L -d /etc/pki/pki-tomcat/alias -h token -f token.pwd

Certificate Nickname                                         Trust Attributes
```

```
                                                    SSL,S/MIME,JAR/XPI

token:ca_signing                                    CTu,Cu,Cu
token:ca_ocsp_signing                               u,u,u
token:subsystem                                     u,u,u
token:ca_audit_signing                              u,u,Pu
token:sslserver/pki.example.com                     u,u,u
```

8.2.6. 使用 Thales Luna HSM 安装子系统

要安装使用 Thales Luna HSM 的子系统实例，请按照 第 8.2.5 节 “使用 nCipher nShield Connect XC HSM 安装子系统” 中详述的步骤操作。覆盖文件应与 例 8.1 “使用 nCipher HSM 的覆盖文件示例” 中提供的示例类似，这与特定部署相关的值不同。以下示例提供了一个 LunaSA 标头示例，它被替换为 nCipher 覆盖文件的标头，并用于 [DEFAULT], [Tomcat], [CA], [KRA], [OCSP], [TKS], [TKS], 和 [TPS] 部分。

例 8.2. LunaSA Override File 标头的示例

```
#############################################################################################

#############################################################################################

#############################################################################################

##                                                         ##
## EXAMPLE: Configuration File used to override '/etc/pki/default.cfg'      ##
##        when using a LunaSA Hardware Security Module (HSM):              ##
##                                                         ##
##                                                         ##
##    # modutil -dbdir . -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/safenet/lunaclient/lib/libCryptoki2_64.so   ##
##           slots: 4 slots attached                                   ##
##          status: loaded                                           ##
##                                                         ##
##           slot: LunaNet Slot                                        ##
##          token: dev-intca                                          ##
##                                                         ##
##           slot: Luna UHD Slot                                       ##
```

```
##          token:                                          ##
##                                                      ##
##          slot: Luna UHD Slot                                 ##
##          token:                                          ##
##                                                      ##
##          slot: Luna UHD Slot                                 ##
##          token:                                          ##
##    ------------------------------------------------------        ##
##                                                      ##
##                                                      ##
##    Based on the example above, substitute all password values,         ##
##    as well as the following values:                                ##
##                                                      ##
##       <hsm_libfile>=/usr/safenet/lunaclient/lib/libCryptoki2_64.so     ##
##       <hsm_modulename>=lunasa                                  ##
##       <hsm_token_name>=dev-intca                               ##
##                                                      ##
#########################################################################################

#########################################################################################

#########################################################################################
```

8.3. 在硬件安全模块中备份密钥

无法将 HSM 中存储的密钥和证书导出到 .p12 文件。如果要备份这样的实例，请联络您的 HSM 厂商以获得支持。

8.4. 使用 HSM 安装克隆子系统

通常，克隆子系统会使用包含主子系统的系统密钥的 PKCS TOTP 文件创建。此文件是在安装 master 子系统的过程中生成的，方法是：在您用于安装的配置文件中将 pki_backup_keys 设置为 True，以及为 pki_backup_password 选项定义的值，或使用 PKCS12Export 工具导出密钥。

在使用 HSM 时，您无法使用 PKCS12Export 生成 PKCSzFCP 文件，因为无法从 HSM 检索密钥。相反，克隆子系统应指向主子系统使用的 HSM，以便它可以访问相同的密钥。另外，如果使用单独的 HSM，请使用 HSM 供应商提供的工具将密钥克隆到此 HSM。在运行 pkispawn 工具前，您必须提供对 master 子系统的密钥的访问。

当使用 HSM 安装时，证书系统没有使用 PKCS TOTP，因此您不需要在配置文件中为 PKCSRG 文件及其密码设置参数。以下示例提供了从相应 PKI 配置文件的 [CA] 部分中生成 CA 克隆和列出选项所需的配置参数：

生成非HSM CA 克隆：

```
[CA]
pki_clone=True
pki_clone_pkcs12_password=Secret123
pki_clone_pkcs12_path=<path_to_pkcs12_file>
pki_clone_replicate_schema=True
pki_clone_uri=https://<master_ca_host_name>:<master_ca_https_port>
```

使用 HSM 生成 CA 克隆：

```
[CA]
pki_clone=True
pki_clone_replicate_schema=True
pki_clone_uri=https://<master_ca_host_name>:<master_ca_https_port>
```

注意

要让 master 子系统与其克隆共享相同的证书和密钥，HSM 必须位于共享模式的网络中，并可以被所有子系统访问。

8.5. 查看令牌

要查看当前为证书系统实例安装的令牌列表，请使用 modutil 实用程序。

1. *更改到实例 别名 目录。例如：*

```
# cd /var/lib/pki/pki-tomcat/alias
```

2. *使用 modutil 工具显示已安装 PKCS facilities 模块以及相应令牌的信息。*

```
# modutil -dbdir . -nocertdb -list
```

8.6. 检测令牌

要查看证书系统是否可以检测到令牌，请使用 TokenInfo 实用程序，指向子系统实例 的别名 目录。这是在安装证书系统软件包后可用的证书系统工具。

例如：

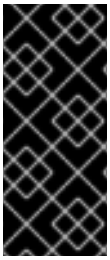
```
# TokenInfo /var/lib/pki/pki-tomcat/alias
```

这个工具会返回证书系统可以检测到的所有令牌，而不仅仅是在证书系统中安装的令牌。

第 9 章 使用 ECC 系统证书安装实例

在某些情况下，Elliptic curve 加密 (ECC)可能会首选使用 RSA 风格的加密，因为它允许使用非常短的密钥长度，并使其更快地生成证书。启用 ECC 的 CA 可以使用其 ECC 签名证书发布 RSA 和 ECC 证书。

证书系统包括对 ECC 功能的原生支持；支持默认从 NSS 3.16 开始启用。也可以加载和使用第三方 PKCS facilities 模块，如硬件安全模块(HSM)。要使用 ECC 模块，必须先加载它，然后才能配置子系统实例。



重要

第三方 ECC 模块必须为其配置 SELinux 策略，或者 SELinux 需要从 enforcing 模式改为 permissive 模式，以允许该模块正常工作。否则，任何需要 ECC 模块的子系统操作都将失败。

9.1. 加载第三方 ECC 模块

加载第三方 ECC 模块遵循与加载 Certificate System 支持的 HSM 相同的原则，如 第 8 章 为子系统安全数据库使用硬件安全模块 所述。如需更多信息，请参阅本章。

9.2. 使用带有 HSM 的 ECC

证书系统支持的 HSM 支持自己的原生 ECC 模块。使用 ECC 系统证书创建实例：

1. 设置 HSM 每个制造商的说明。如果多个主机共享 HSM，请确保它们都可以根据需要访问同一分区，以及站点策略是否允许它。

2. 在 pkispawn 工具配置文件中定义所需的参数，并运行 pkispawn。例如，要将证书系统配置为创建 ECC CA，假设配置文件为 ecc.inf ：

    1. 编辑 ecc.inf 以指定适当的设置。有关配置文件的示例，请查看 pkispawn(8) man page。

    2. 针对 ecc.inf 运行 pkispawn ：

```
$ script -c 'pkispawn -s CA -f /root/pki/ecc.inf -vvv'
```

第 10 章 克隆子系统

首次配置新子系统实例时，红帽认证系统允许克隆或重复子系统，以进行证书系统的高可用性。克隆的实例在不同的机器上运行，以避免通过复制同步单点故障及其数据库。

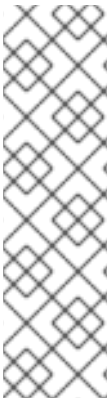
主 CA 及其克隆的功能相同，它们只在序列号分配和 CRL 生成中有所不同。因此，本章将 master 或其任何克隆指代为 复制 CA。

10.1. 从软件数据库备份子系统密钥

理想情况下，在实例首次创建时，master 实例的密钥会被备份。如果没有备份密钥，或者备份文件丢失，那么可以使用 PKCS12Export 工具从子系统实例的内部软件数据库中提取密钥。例如：

```
PKCS12Export -debug -d /var/lib/pki/instance_name/alias -w p12pwd.txt -p internal.txt -o master.p12
```

然后，将 PKCS libpmem 文件复制到克隆实例配置中使用的克隆机器中。如需了解更多详细信息，请参阅 第 2.7.6 节 “克隆和密钥存储”。



注意

无法从 HSM 导出密钥。但是，在典型的部署中，HSM 支持网络访问权限，只要克隆实例使用与 master 相同的 HSM。如果两个实例都使用相同的密钥存储，则克隆会自然使用密钥。

如果需要从 HSM 备份密钥，请联系 HSM 制造商以获得帮助。

10.2. 克隆 CA

1. 配置主 CA 并备份密钥。

2. 在 master CA 的 CS.cfg 文件中，通过添加 ca.listenToCloneModifications 参数来启用 master CA 来监控复制数据库更改：

```
ca.listenToCloneModifications=true
```

3. 创建 clone 子系统实例。

有关克隆 CA 子系统时 pkispawn 所需的配置文件示例，请参阅 在 pkispawn(8) man page 的同一主机上安装 CA 克隆 和安装 CA 克隆。

4. 重启克隆使用的 Directory 服务器实例。

```
# systemctl restart pki-tomcatd@kra-clone-ds-instance.service
```

注意

重启 Directory 服务器会重新载入更新的模式，这是正确的性能所必需的。

5. 重启克隆实例。

```
# pki-server restart instance_name
```

配置克隆后，测试以确保 master-clone 关系可以正常工作：

1. 从克隆的 CA 请求证书。

2. 批准请求。

3. 将证书下载到浏览器。

4. 吊销证书。

5. 检查 master CA 的 CRL 是否有撤销的证书。在 master Certificate Manager 的代理服务页面中，单击 Update Certificate Revocation List。在列表中找到 CRL。

CRL 应该显示克隆的证书管理器撤销的证书。如果没有列出该证书，请检查日志来解决这个问题。

10.3. 在关闭后更新 CA-KRA CONNECTOR 信息

如 第 2.7.9 节 “自定义配置和克隆” 所述，如果在克隆创建后进行了配置信息，则不会 在 克隆实例中更新配置信息。同样，对克隆所做的更改不会复制到 master 实例。

如果在克隆 CA 创建后安装或克隆新的 KRA，则克隆 CA 在其配置中没有新的 KRA 连接器信息。这意味着克隆 CA 无法将任何归档请求发送到 KRA。

每当创建或克隆新的 KRA 时，将其连接器信息复制到部署中的所有 克隆的 CA 中。为此，请使用 pki ca-kraconnector-add 命令。

如果需要手动进行此操作，请按照以下步骤执行：

1. 在 master 克隆机器上，打开 master CA 的 CS.cfg 文件，并为新的 KRA 连接器复制所有 ca.connector.KRA prerequisites 行。

```
[root@master ~]# vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

2. 停止克隆 CA 实例。例如：

```
[root@clone-ca ~]# pki-server stop instance_name
```

3. 打开克隆 CA 的 CS.cfg 文件。

```
[root@clone-ca ~]# vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

4. 复制新 KRA 实例或克隆的连接器信息。

```
ca.connector.KRA.enable=true ca.connector.KRA.host=server-kra.example.com
ca.connector.KRA.local=false ca.connector.KRA.nickName=subsystemCert cert-pki-ca
ca.connector.KRA.port=10444 ca.connector.KRA.timeout=30
ca.connector.KRA.transportCert=MIIDbD...ZR0Y2zA==
ca.connector.KRA.uri=/kra/agent/kra/connector
```

5. 启动克隆 CA。

```
[root@clone-ca ~]# pki-server start instance_name
```

10.4. 克隆 OCSP 子系统

1. 配置主 OCSP 并备份密钥。

2. 在 master OCSP 的 CS.cfg 文件中，将 OCSP.Responder.store.defStore.refreshInSec 参数设置为 21600 以外的任何非零数；21600 是克隆的设置。

```
# vim /etc/instance_name/CS.cfg

OCSP.Responder.store.defStore.refreshInSec=15000
```

3. 使用 pkispawn 实用程序创建克隆子系统实例。

有关克隆 OCSP 子系统时 pkispawn 所需的配置文件示例，请查看 pkispawn(8) man page。

4. 重启克隆使用的 Directory 服务器实例。

```
# systemctl dirsrv@instance_name.service
```

注意

重启 Directory 服务器会重新载入更新的模式，这是正确的性能所必需的。

5. 重启克隆实例。

```
# pki-server restart instance_name
```

配置克隆后，测试以确保 master-clone 关系可以正常工作：

1. 在 master CA 中设置 OCSP 发布，以便 CRL 发布到 master OCSP。

2. 成功发布 CRL 后，检查代理页面中的主和克隆的 OCSP 列表证书颁发机构 链接。列表应该相同。

3. 使用 OCSPClient 工具向 master 和克隆的在线证书状态管理器提交 OCSP 请求。该工具应该从两个管理器接收相同的 OCSP 响应。

10.5. 克隆 KRA 子系统

1. 配置主子系统，再备份密钥。

2. 使用 pkispawn 工具创建克隆子系统实例：

```
$ pkispawn -s <subsystem> -f myconfig.txt
```

在克隆 KRA 子系统时，pkispawn 所需的配置文件示例：

```
[DEFAULT]
pki_admin_password=<Secret.123>
pki_client_database_password=<Secret.123>
pki_client_pkcs12_password=<Secret.123>
pki_ds_password=<Secret.123>
pki_security_domain_password=<Secret.123>
pki_security_domain_hostname=<master_ca_hostname>
pki_security_domain_https_port=<master_ca_https_port>
pki_security_domain_user=caadmin

[KRA]
pki_clone=True
pki_clone_pkcs12_password=<Secret.123>
pki_clone_pkcs12_path=<path_to_pkcs12_file>
pki_clone_replicate_schema=True
pki_clone_uri=https://<master_subsystem_host:master_subsystem_https_port>
pki_issuing_ca=https://<ca_hostname:ca_https_port>
```

3. 重启克隆使用的 Directory 服务器实例。

```
# systemctl dirsrv@instance_name.service
```

注意

重启 Directory 服务器会重新载入更新的模式，这是正确的性能所必需的。

4. 重启克隆实例。

```
# pki-server restart instance_name
```

对于 KRA 克隆，测试以确保 master-clone 关系正常运行：

1. 进入 KRA 代理的页面。
2. 单击 **List Requests**。
3. 选择 **Show all requests for the request type and status**。
4. 单击 **Submit**。
5. 比较克隆的 KRA 和主 KRA 的结果。结果相同。

10.6. 克隆 TKS 子系统

1. 配置主子系统，再备份密钥。
2. 使用 **pkispawn** 实用程序创建克隆子系统实例。

```
$ pkispawn -s <subsystem> -f myconfig.txt
```

克隆 TKS 子系统时 **pkispawn** 所需的配置文件示例：

```
[DEFAULT]
pki_admin_password=<Secret.123>
```

```
pki_client_database_password=<Secret.123>
pki_client_pkcs12_password=<Secret.123>
pki_ds_password=<Secret.123>
pki_security_domain_password=<Secret.123>
pki_security_domain_hostname=<master_ca_hostname>
pki_security_domain_https_port=<master_ca_https_port>
pki_security_domain_user=caadmin

[TKS]
pki_clone=True
pki_clone_pkcs12_password=<Secret.123>
pki_clone_pkcs12_path=<path_to_pkcs12_file>
pki_clone_replicate_schema=True
pki_clone_uri=https://<master_subsystem_host:master_subsystem_https_port>
pki_issuing_ca=https://<ca_hostname:ca_https_port>
```

3. 重启克隆实例。

```
# pki-server restart instance_name
```

对于 TKS，注册智能卡，然后运行 ldapsearch 以确保两个数据库中包含相同的密钥信息。

10.7. 转换 MASTER 和 CLONES

同一拓扑中只能有一个活跃的 CA 生成 CRL。同样，在同一拓扑中只能有一个 OCSP 接收 CRL。因此，可以有任意数量的克隆，但只能为 CA 和 OCSP 配置了一个 master。

对于 KRA 和 TKS，master 和克隆之间没有配置区别，但 CA 和 OCSP 有一些配置区别。这意味着，当 master 离线时 - 由于失败或维护，或更改 PKI 中的子系统的功能 - 然后必须重新配置现有的 master 来克隆，并将提升为主克隆之一。

10.7.1. 转换 CA 克隆和 Master

1. 如果 master CA 仍在运行，则停止它。

2. 打开现有的 master CA 配置目录：

```
# cd /var/lib/pki/instance_name/ca/conf
```

3. 编辑 master 的 CS.cfg 文件，并更改 CRL 和维护线程设置，使其设置为克隆：

- ***禁用对数据库维护线程的控制：***

```
ca.certStatusUpdateInterval=0
```

- ***禁用监控数据库复制更改：***

```
ca.listenToCloneModifications=false
```

- ***禁用 CRL 缓存维护：***

```
ca.crl.IssuingPointId.enableCRLCache=false
```

- ***禁用 CRL 生成：***

```
ca.crl.IssuingPointId.enableCRLUpdates=false
```

- ***将 CA 设置为将 CRL 请求重定向到新 master：***

```
master.ca.agent.host=new_master_hostname
master.ca.agent.port=new_master_port
```

4. ***停止克隆的 CA 服务器。***

```
# pki-server stop instance_name
```

5. ***打开克隆的 CA 的配置目录。***

```
# cd /etc/instance_name
```

6. ***编辑 CS.cfg 文件，将克隆配置为新 master。***

   a. ***删除以 ca.crl. 前缀开头的每行。***

   b.

将前 master CA CS.cfg 文件中的 ca.crl. 前缀开头的每行复制到克隆的 CA 的 CS.cfg 文件中。

c. 启用控制数据库维护线程；主 CA 的默认值为 600。

```
ca.certStatusUpdateInterval=600
```

d. 启用监控数据库复制：

```
ca.listenToCloneModifications=true
```

e. 启用 CRL 缓存维护：

```
ca.crl.IssuingPointId.enableCRLCache=true
```

f. 启用 CRL 生成：

```
ca.crl.IssuingPointId.enableCRLUpdates=true
```

g. 禁用 CRL 生成请求的重定向设置：

```
master.ca.agent.host=hostname
master.ca.agent.port=port number
```

7. 启动新的 master CA 服务器。

```
# pki-server start instance_name
```

10.7.2. 转换 OCSP 克隆

1. 如果仍在运行，则停止 OCSP master。

2. 打开现有的 master OCSP 配置目录。

```
# cd /etc/instance_name
```

3. 编辑 CS.cfg，并将 OCSP.Responder.store.defStore.refreshInSec 参数重置为 21600 ：

```
OCSP.Responder.store.defStore.refreshInSec=21600
```

4. 停止在线克隆的 OCSP 服务器。

```
# pki-server stop instance_name
```

5. 打开克隆的 OCSP 响应器配置目录。

```
# cd /etc/instance_name
```

6. 打开 CS.cfg 文件，并删除 OCSP.Responder.store.defStore.refreshInSec 参数，或者将其值改为任何非零数字：

```
OCSP.Responder.store.defStore.refreshInSec=15000
```

7. 启动新的 master OCSP 响应程序服务器。

```
# pki-server start instance_name
```

10.8. 克隆具有 BEEN RE-KEYED 的 CA

当证书过期时，必须替换它。这可以通过续订证书来完成，该证书可重新使用原始密钥对来生成新证书，或者通过生成新的密钥对和证书来完成。第二种方法称为 re-keying。

当 CA 重新密钥时，新密钥对存储在其证书数据库中，它们是正常操作的密钥引用。但是，为了克隆子系统，克隆过程会检查存储在 CS.cfg 配置文件中的 CA 私钥 ID - 这些密钥 ID 在证书数据库密钥更改时不会更新。

如果 CA 已重新密钥，然后管理员尝试克隆它，克隆的 CA 无法为重新密钥的证书生成任何证书，并在错误日志中显示出错信息：

```
CertUtil::createSelfSignedCert() - CA private key is null!
```

克隆已重新密钥的 CA：

1. *在 CS.cfg 文件中找到所有私钥 ID。*

```
# grep privkey.id /var/lib/pki/instance_name/ca/conf/CS.cfg
cloning.signing.privkey.id     =-4d798441aa7230910d4e1c39fa132ea228d5d1bc
cloning.ocsp_signing.privkey.id =-3e23e743e0ddd88f2a7c6f69fa9f9bcebef1a60
cloning.subsystem.privkey.id     =-c3c1b3b4e8f5dd6d2bdefd07581c0b15529536
cloning.sslserver.privkey.id    =3023d30245804a4fab42be209ebb0dc683423a8f
cloning.audit_signing.privkey.id=2fe35d9d46b373efabe9ef01b8436667a70df096
```

2. *打印存储在 NSS 数据库中的所有当前私钥 ID，并将其与存储在 CS.cfg 文件中的私钥 ID 进行比较：*

```
# 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      d684da39bf4f2789a3fc9d42204596f4578ad2d9   subsystemCert cert-ca4-
test-master
< 3> rsa      a8edd7c2b5c94f13144cacd99624578ae30b7e43   sslserverCert cert-ca4-
test1
< 4> rsa      2fe35d9d46b373efabe9ef01b8436667a70df096   auditSigningCert cert-ca4-
test1
```

在本例中，只有审计签名密钥是相同的，其他密钥已更改。

3. *取步骤 2 返回的键，并将它们从未签名的值（即 certutil 返回的内容）转换为签名 Java BigIntegers （这是密钥如何存储在证书系统数据库中）。*

这可以通过计算器或使用 例 10.1 “certutil 到 BigInteger Conversion Program” 中的脚本来完成。

4. *将新键值复制到 CS.cfg 文件中。*

```
# vim /var/lib/pki/instance_name/ca/conf/CS.cfg
cloning.signing.privkey.id     =-584f6bb4847c688d65b373650c563d4690b6390d
```

```
cloning.ocsp_signing.privkey.id =6006094af3e5d02aaa91426594ca66cb53e73ac0
cloning.subsystem.privkey.id   =-297b25c640b0d8765c0362bddfba690ba8752d27
cloning.sslserver.privkey.id   =-5712283d4a36b0ecebb3532669dba8751cf481bd
cloning.audit_signing.privkey.id=2fe35d9d46b373efabe9ef01b8436667a70df096
```

5. 克隆 CA，如 第 10.2 节 “克隆 CA” 所述。

例 10.1. certutil 到 BigInteger Conversion Program

此 Java 程序可将密钥输出从 certutil 转换为所需的 BigInteger 格式。

将此保存为 .java 文件，如 Test.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));
 }
}
```

然后编译该文件：

```
# javac Test.java
```

第 11 章 设置 PKI ACME RESPONDER

本章论述了在已具有 CA 子系统的 PKI 服务器上的 ACME 响应器上安装和初始配置。

注意

以下假设您安装了带有默认实例名称的 CA （例如 pki-tomcat）。

有关如何管理 PKI ACME Responder 的详情，请参考 Red Hat Certificate System Administration Guide 中的 Managing PKI ACME Responder 章节。

11.1. 安装 PKI ACME RESPONDER

要在 PKI 服务器上安装 PKI ACME Responder,

1. 首先下载并安装 pki-acme RPM 软件包：

```
$ dnf install pki-acme
```

2. 使用以下命令，在 PKI 服务器实例中创建 ACME 响应器：

```
$ pki-server acme-create
```

这会在 /etc/pki/pki-tomcat/acme 目录中创建初始配置文件。

如需更多信息，请参阅 pki-server-acme manpage。

11.2. 配置 ACME 数据库

本节论述了如何为 ACME 响应器配置数据库。数据库配置位于 /etc/pki/pki-tomcat/acme/database.conf。

- 您可以使用 pki-server acme-database-mod 命令通过命令行配置数据库。在没有参数的情

况下调用此命令会启动一个互动模式，例如：

```
$ pki-server acme-database-mod

The current value is displayed in the square brackets.
To keep the current value, simply press Enter.
To change the current value, enter the new value.
To remove the current value, enter a blank space.

Enter the type of the database. Available types: ds, in-memory, ldap, openldap, postgresql.
Database Type: ds

Enter the location of the LDAP server (e.g. ldap://localhost.localdomain:389).
Server URL [ldap://localhost.localdomain:389]:

Enter the authentication type. Available types: BasicAuth, SslClientAuth.
Authentication Type [BasicAuth]:

Enter the bind DN.
Bind DN [cn=Directory Manager]:

Enter the bind password.
Bind Password [********]:

Enter the base DN for the ACME subtree.
Base DN [dc=acme,dc=pki,dc=example,dc=com]:
```

- *使用 **--type** 参数调用命令会根据指定类型创建新配置。*
- *使用其他参数调用 命令会更新指定的参数。*

*某些 **ACME** 配置属性存储在数据库中，可让您一致地配置集群中的所有 **ACME** 响应器。默认情况下，**ACAC** 响应器在检索或更新 **ACME** 配置属性时直接访问数据库，这可能会增加数据库的负载。有些数据库可能会提供 **ACME** 配置监控器来减少这个负载。*

11.2.1. 配置 DS 数据库

*您可以将 **ACME** 响应器配置为使用 **DS** 数据库。**DS** 数据库配置示例位于 **/usr/share/pki/acme/database/ds/database.conf**。*

*配置 **DS** 数据库：*

1. *首先，使用以下命令导入 **/usr/share/pki/acme/database/ds/schema.ldif** 文件来添加*

ACME DS 模式：

```
$ ldapmodify -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/database/ds/schema.ldif
```

2. **接下来，准备 LDIF 文件以创建 ACME 子树。示例 LDIF 文件位于 /usr/share/pki/acme/database/ds/create.ldif。本例使用 dc=acme,dc=pki,dc=example,dc=com 作为基本 DN。**

3. **使用 ldapadd 命令导入 LDIF 文件：**

```
$ ldapadd -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/database/ds/create.ldif
```

4. **将示例数据库配置文件从 /usr/share/pki/acme/database/ds/database.conf 复制到 /etc/pki/pki-tomcat/acme 目录中，或者执行以下命令来自定义一些参数：**

```
$ pki-server acme-database-mod --type ds \
        -DbindPassword=Secret.123
```

5. **根据需要自定义配置：**

   - **在独立 ACME 部署中，database.conf 应该类似如下：**

```
class=org.example.acme.database.DSDatabase
url=ldap://<hostname>:389
authType=BasicAuth
bindDN=cn=Directory Manager
bindPassword=Secret.123
baseDN=dc=acme,dc=pki,dc=example,dc=com
```

   - **在共享的 CA 和 ACME 部署中，database.conf 应该类似如下：**

```
class=org.example.acme.database.DSDatabase
        configFile=conf/ca/CS.cfg
        baseDN=dc=acme,dc=pki,dc=example,dc=com
```

DS 数据库使用搜索持久性提供 ACME 配置监控器。您可以通过启用以下参数来启用它：monitor.enabled=true

11.3. 配置 ACME 颁发者

本节描述了如何为 PKI ACME Responder 配置签发者。ACME Issuer 的配置位于 /etc/pki/pki-tomcat/acme/issuer.conf。

您可以使用 pki-server acme-issuer-mod 命令通过命令行配置签发者。

- 在没有参数的情况下调用此命令会启动一个互动模式，例如：

```
$ pki-server acme-issuer-mod
The current value is displayed in the square brackets.
To keep the current value, simply press Enter.
To change the current value, enter the new value.
To remove the current value, enter a blank space.

Enter the type of the certificate issuer. Available types: nss, pki.
Issuer Type: pki

Enter the location of the PKI server (e.g. https://localhost.localdomain:8443).
Server URL [https://localhost.localdomain:8443]:

Enter the certificate nickname for client authentication.
This might be the CA agent certificate.
Enter blank to use basic authentication.
Client Certificate:

Enter the username of the CA agent for basic authentication.
Enter blank if a CA agent certificate is used for client authentication.
Agent Username [caadmin]:

Enter the CA agent password for basic authentication.
Enter blank if the password is already stored in a separate property file
or if a CA agent certificate is used for client authentication.
Agent Password [********]:

Enter the certificate profile for issuing ACME certificates (e.g. acmeServerCert).
Certificate Profile [acmeServerCert]:
```

- 使用 --type 参数调用命令会根据指定类型创建新配置。

- 使用其他参数调用 命令会更新指定的参数。

11.3.1. 配置 PKI Issuer

您可以配置 PKI ACME Responder，以使用 PKI Issuer 发布证书。示例配置位于 /usr/share/pki/acme/issuer/pki/issuer.conf。

- 要配置 PKI 签发者，请将这个示例 issuer.conf 复制到 /etc/pki/pki-tomcat/acme 目录中，或者执行以下命令来自定义一些参数：

```
$ pki-server acme-issuer-mod --type pki \
        -Dusername=caadmin \
        -Dpassword=Secret.123
```

根据需要自定义配置。issuer.conf 文件应该类似如下：

```
class=org.example.acme.issuer.PKIIssuer
url=https://localhost.localdomain:8443
profile=acmeServerCert
username=caadmin
password=Secret.123
```

- url 参数指定 PKI 签发者位置。
- profile 参数指定要使用的证书配置文件。
- 要使用客户端证书身份验证，请在 nickname 参数中指定客户端证书 别名。
- 要使用基本身份验证，请在 username 参数中指定用户名，并在 password 参数中指定密码。

11.4. 配置 ACME REALM

本节描述了如何为 PKI ACME 响应器 配置域。realm 配置位于 /etc/pki/pki-tomcat/acme/realm.conf。

您可以使用 pki-server acme-realm-mod 命令通过命令行配置 ACME Realm 域。

- 在没有参数的情况下调用此命令会启动一个互动模式，例如：

```
$ pki-server acme-realm-mod
The current value is displayed in the square brackets.
To keep the current value, simply press Enter.
To change the current value, enter the new value.
To remove the current value, enter a blank space.

Enter the type of the realm. Available types: ds.
Database Type: ds

Enter the location of the LDAP server (e.g. ldap://localhost.localdomain:389).
Server URL [ldap://localhost.localdomain:389]:

Enter the authentication type. Available types: BasicAuth, SslClientAuth.
Authentication Type [BasicAuth]:

Enter the bind DN.
Bind DN [cn=Directory Manager]:

Enter the bind password.
Bind Password [********]:

Enter the base DN for the ACME users subtree.
Users DN [ou=people,dc=acme,dc=pki,dc=example,dc=com]:

Enter the base DN for the ACME groups subtree.
Groups DN [ou=groups,dc=acme,dc=pki,dc=example,dc=com]:
```

- 使用 **--type** 参数调用命令会根据指定类型创建新配置。

- 使用其他参数调用 命令会更新指定的参数。

11.4.1. 配置 DS Realm

您可以将 **PKI ACME Responder** 配置为使用 **DS** 域。**DS Realm** 的示例配置位于 **/usr/share/pki/acme/realm/ds/realm.conf**。

配置 **DS** 域：

1. 为 **DS** 中的 **ACME** 用户和组准备子树。示例 **LDIF** 文件位于 **/usr/share/pki/acme/realm/ds/create.ldif]**。本例使用 **dc=acme,dc=pki,dc=example,dc=com** 作为基本 **DN**。

2. 使用 ldapadd 命令导入 LDIF 文件：

```
$ ldapadd -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/realm/ds/create.ldif
```

3. 将示例配置文件从 /usr/share/pki/acme/realm/ds/realm.conf 复制到 /etc/pki/pki-tomcat/acme 目录中，或者运行以下命令来自定义一些参数：

```
$ pki-server acme-realm-mod --type ds \
        -DbindPassword=Secret.123
```

4. 根据需要自定义配置：

- 在独立 ACME 部署中，realm.conf 文件应该类似如下：

```
class=org.example.acme.realm.DSRealm
url=ldap://<hostname>:389
authType=BasicAuth
bindDN=cn=Directory Manager
bindPassword=Secret.123
usersDN=ou=people,dc=acme,dc=pki,dc=example,dc=com
groupsDN=ou=groups,dc=acme,dc=pki,dc=example,dc=com
```

- 在共享的 CA 和 ACME 部署中，realm.conf 文件应该类似如下：

```
class=org.example.acme.realm.DSRealm
configFile=conf/ca/CS.cfg
usersDN=ou=people,dc=ca,dc=pki,dc=example,dc=com
groupsDN=ou=groups,dc=ca,dc=pki,dc=example,dc=com
```

11.5. 部署 ACME RESPONDER

1. 配置 ACME 响应器后，使用以下命令部署它：

```
$ pki-server acme-deploy
```

这会在 /etc/pki/pki-tomcat/Catalina/localhost/acme.xml 中创建一个部署描述符。

PKI 服务器在几秒钟后自动启动 ACME 响应器，您不需要重新启动服务器。

2. *要验证 ACME 响应器是否正在运行，请使用以下命令：*

```
$ curl -s -k https://$HOSTNAME:8443/acme/directory | python -m json.tool
{
"meta": {
"caaIdentities": [
"example.com"
],
"externalAccountRequired": false,
"termsOfService": "https://example.com/acme/tos.pdf",
"website": "https://www.example.com"
},
"newAccount": "https://<hostname>:8443/acme/new-account",
"newNonce": "https://<hostname>:8443/acme/new-nonce",
"newOrder": "https://<hostname>:8443/acme/new-order",
"revokeCert": "https://<hostname>:8443/acme/revoke-cert"
}
```

如需更多信息，请参阅 pki-server-acme manpage。

第 12 章 其他安装选项

使用 pkispawn 创建的所有 Red Hat Certificate System 实例都会对要安装的实例进行某些假设，如用于 CA 签名证书的默认签名算法，以及是否允许主机的 IPv6 地址。

本章论述了影响新实例安装和配置的其他配置选项，因此创建实例前会发生其中许多流程。

12.1. 轻量级子 CA

使用默认设置，您可以创建轻量级子 CA。它们允许您配置服务，如虚拟专用网络(VPN)网关，以仅接受由一个子 CA 发布的证书。同时，您可以将其他服务配置为仅接受由不同子 CA 或 root CA 发布的证书。

如果您撤销了子 CA 的中间证书，则此子 CA 发布的所有证书都会自动无效。

如果您在证书系统中设置 CA 子系统，则会自动使用 root CA。您创建的所有子 CA 都从属到这个 root CA。

12.1.1. 设置轻量级子 CA

根据您的环境，子 CA 的安装在内部 CA 和外部 CA 之间有所不同。如需更多信息，请参阅 第 7.8 节 “使用外部 CA 设置子系统”。

12.1.2. 禁用创建轻量级子 CA

在某些情况下，管理员希望禁用轻量级子 CA。要防止添加、修改或删除子 CA，请在证书系统使用的 Directory Server 实例中输入以下命令：

```
# ldapmodify -D "cn=Directory Manager" -W -x -h server.example.com

dn: cn=aclResources,o=instance_name
changetype: modify
delete: resourceACLS
resourceACLS: certServer.ca.authorities:create,modify:allow (create,modify)
  group="Administrators":Administrators may create and modify lightweight authorities
delete: resourceACLS
resourceACLS: certServer.ca.authorities:delete:allow (delete)
  group="Administrators":Administrators may delete lightweight authorities
```

这个命令会删除默认的访问控制列表(ACL)条目，该条目授予管理子 CA 的权限。

注意

如果修改了与轻量级子 CA 创建相关的 ACL，请删除相关值。

12.1.3. 重新启用轻量级子 CA 的创建

如果您之前禁用了创建轻量级子 CA，您可以通过在证书系统使用的 Directory Server 实例中输入以下命令来重新启用该功能：

```
# ldapmodify -D "cn=Directory Manager" -W -x -h server.example.com

dn: cn=aclResources,o=instance_name
changetype: modify
add: resourceACLS
resourceACLS: certServer.ca.authorities:create,modify:allow (create,modify)
  group="Administrators":Administrators may create and modify lightweight authorities
resourceACLS: certServer.ca.authorities:delete:allow (delete)
  group="Administrators":Administrators may delete lightweight authorities
```

此命令添加访问控制列表(ACL)条目，该条目授予管理子 CA 的权限。

12.2. 为子系统启用 IPV6

证书系统会自动配置和管理子系统之间的连接。每个子系统都必须与 CA 交互，作为安全域的成员并执行其 PKI 操作。

对于这些连接，证书系统子系统可以通过其主机的完全限定域名或 IP 地址来识别。默认情况下，证书系统自动解析 IPv4 地址和主机名，但证书系统也可以将 IPv6 用于其连接。IPv6 支持所有服务器连接：到其他子系统，到管理控制台(pkiconsole)，或者通过命令行脚本，如 tpsclient ：

```
op=var_set name=ca_host value=IPv6 address
```

1. 安装 Red Hat Certificate System 软件包。

2. 在 /etc/hosts 文件中设置 IPv4 和 IPv6 地址。例如：

```
vim /etc/hosts

192.0.0.0    server.example.com IPv4 address
3ffe:1234:2222:2000:202:55ff:fe67:f527        server6.example.com IPv6 address
```

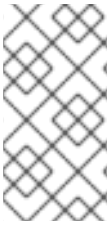
3. 然后，导出环境变量以使用服务器的 IPv6 地址。例如：

```
export PKI_HOSTNAME=server6.example.com
```

4. 运行 pkispawn 以创建新实例。CS.cfg 文件中的服务器主机名的值将设置为 IPv6 地址。

12.3. 启用基于 LDAP 的注册配置文件

要使用基于 LDAP 的配置集进行安装，请在 pkispawn 配置文件的 [CA] 部分中设置 pki_profile_in_ldap=True 选项。



注意

在这种情况下，配置文件仍会出现在 /var/lib/pki/instance_name/ca/profiles/ca/ 中，但将被忽略。

要在现有实例上启用基于 LDAP 的配置集，请在实例的 CS.cfg 中更改以下内容：

```
subsystem.1.class=com.netscape.cmscore.profile.LDAPProfileSubsystem
```

然后，使用 pki 命令行工具或自定义脚本手动将配置集导入到数据库中。

12.4. 自定义 TLS CIPHERS

在安装过程中可以强制使用 TLS 密码。请参阅 Red Hat Certificate System Administration Guide 中的 Configuring Ciphers 部分。

第 13 章 安装和克隆故障排除

本章论述了在安装证书系统时遇到的一些比较常见的安装和升级问题。

13.1. 安装

问：

我无法看到任何证书系统软件包或更新。

答：

验证您的系统是否已正确注册到红帽订阅管理服务，分配了有效的订阅，并且启用了证书系统存储库。详情请查看 第 6.8 节 “附加红帽订阅并启用证书系统软件包存储库”。

问：

初始化脚本会返回 OK 状态，但我的 CA 实例没有响应。这是因为什么？

答：

这不应发生。通常（但不总是）表示 CA 的监听程序问题，但可能有很多不同的原因。要查看发生的错误，请运行以下命令检查日志 日志 ：

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

或者，检查 /var/log/pki/instance_name/subsystem_type/debug 的调试日志文件。

一种情况是 CA 有 PID，表示进程正在运行，但没有为服务器打开任何监听程序。这将返回 catalina.out 文件中的 Java 调用类错误：

```
Oct 29, 2010 4:15:44 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-9080
java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:64)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:615)
        at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:243)
        at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:408)
Caused by: java.lang.UnsatisfiedLinkError: jss4
```

这可能意味着您有错误的 JSS 或 NSS 版本。进程在路径中需要 libnss3.so。使用这个命令进行检查：

```
ldd /usr/lib64/libjss4.so
```

如果没有找到 libnss3.so，请在 /etc/sysconfig/instance_name配置文件中设置 正确的 classpath。然后，使用 systemctl restart pki-tomcatd@instance_name.service 命令重启 CA。

问：

我想自定义 CA 签名证书的主题名称，但不要使用 pkispawn 交互式安装模式来执行此操作。

答：

要做到这一点，需要一个代表到 /etc/pki/default.cfg 文件的 delta 链接的配置文件。请参阅 pkispawn(8) 和 pki_default.cfg(5) man page。

问：

我希望为我的根证书颁发机构设置不同的证书有效期和扩展 - 但我没有看到使用 pkispawn 设置它的方法。

答：

您目前无法使用 pkispawn 执行此操作。但是，可以使用以下方法编辑 pkispawn 使用的证书配置文件来生成 root CA 证书。

重要

您必须在运行 pkispawn 前 执行此操作来创建新 CA 实例。

1. 备份 pkispawn 使用的原始 CA 证书配置文件。

```
cp -p /usr/share/pki/ca/conf/caCert.profile /usr/share/pki/ca/conf/caCert.profile.orig
```

2. 打开配置向导使用的 CA 证书配置文件。

```
vim /usr/share/pki/ca/conf/caCert.profile
```

3. 将 Validity Default 中的有效期重置为您想要的任何有效期。例如，将周期改为两年：

```
2.default.class=com.netscape.cms.profile.def.ValidityDefault
2.default.name=Validity Default
2.default.params.range=7200
```

4. 通过在配置集中创建新的默认条目并将其添加到列表中来添加任何扩展。例如，要添加 Basic Constraint Extension，请添加默认值（在本例中是 default #9））：

```
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
```

然后，将默认数量添加到默认值列表中，以使用新默认值：

```
list=2,4,5,6,7,8,9
```

5. 设置新配置集后，请运行 pkispawn 来创建新的 CA 实例，再通过配置向导。

问：

在配置子系统实例后，我试图连接到 Web 服务页面时，会看到一个 HTTP 500 错误代码。

答：

这是一个意外的通用错误，可能会造成很多不同原因。检查日志、系统 和 调试 实例的日志文件，以查看发生的错误。这列出了几个常见错误，但有很多其他可能。

错误 11：LDAP 数据库没有运行。

如果 Red Hat Directory Server 实例用于内部数据库没有运行，则无法连接到实例。这在实例未就绪 的日志文件中 明显明显：

```
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)
```

Tomcat 日志将特别识别 LDAP 连接的问题：

```
5558.main - [29/Oct/2010:11:13:40 PDT] [8] [3] In Ldap (bound) connection pool
to host ca1 port 389, Cannot connect to LDAP server. Error:
netscape.ldap.LDAPException: failed to connect to server
ldap://ca1.example.com:389 (91)
```

与实例的 调试日志 一样：

```
[29/Oct/2010:11:39:10][main]: CMS:Caught EBaseException
Internal Database Error encountered: Could not connect to LDAP server host
ca1 port 389 Error netscape.ldap.LDAPException: failed to connect to
server ldap://ca1:389 (91)
        at com.netscape.cmscore.dbs.DBSubsystem.init(DBSubsystem.java:262)
```

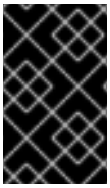
*错误 **2: VPN 阻止访问。***

*另一个可能是通过 **VPN** 连接到子系统。**VPN** 必须有一个 **配置**选项，**如** 仅将此连接用于启用其网络上的资源。**如果没有**启用该选项，则实**例的 Tomcat 服**务的日志文件会显示一系列连接错误，这会导**致 HTTP 500** 错误：*

```
May 26, 2010 7:09:48 PM org.apache.catalina.core.StandardWrapperValve invoke
SEVERE: Servlet.service() for servlet services threw exception
java.io.IOException: CS server is not ready to serve.
        at com.netscape.cms.servlet.base.CMSServlet.service(CMSServlet.java:441)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:803)
        at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:269)

        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
        at
org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:210)
        at
org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:172)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:127)
        at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:117)
        at org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:542)
        at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:108)
        at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:151)
        at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:870)
        at
org.apache.coyote.http11.Http11BaseProtocol$Http11ConnectionHandler.processConnection(Htt
p11BaseProtocol.java:665)
        at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:528)
        at
org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.jav
a:81)
        at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:685)
 at java.lang.Thread.run(Thread.java:636)
```

13.2. Java 控制台



重要

pkiconsole 已被弃用。

问：

我无法打开 pkiconsole，在 stdout 中看到 Java 异常。

答：

这可能意味着您安装了错误的 JRE 或错误的 JRE 设置为默认值。运行 alternatives --config

问：

我试图运行 pkiconsole，在 stdout 中遇到套接字异常。这是因为什么？

答：

这意味着存在端口问题。管理端口的 SSL/TLS 设置不正确（假设 server.xml中存在错误的配置），或者提供了错误的端口来访问管理界面。

端口错误类似如下：

```
NSS Cipher Supported '0xff04'
java.io.IOException: SocketException cannot read on socket
        at org.mozilla.jss.ssl.SSLSocket.read(SSLSocket.java:1006)
        at org.mozilla.jss.ssl.SSLInputStream.read(SSLInputStream.java:70)
        at
com.netscape.admin.certsrv.misc.HttpInputStream.fill(HttpInputStream.java:303)
        at
com.netscape.admin.certsrv.misc.HttpInputStream.readLine(HttpInputStream.java:224)
        at
com.netscape.admin.certsrv.connection.JSSConnection.readHeader(JSSConnection.java:439)
        at
com.netscape.admin.certsrv.connection.JSSConnection.initReadResponse(JSSConnection.java:4
30)
        at
com.netscape.admin.certsrv.connection.JSSConnection.sendRequest(JSSConnection.java:344)

        at
com.netscape.admin.certsrv.connection.AdminConnection.processRequest(AdminConnection.java
:714)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:62
3)
        at
com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:59
0)
        at
com.netscape.admin.certsrv.connection.AdminConnection.authType(AdminConnection.java:323)

        at
com.netscape.admin.certsrv.CMSServerInfo.getAuthType(CMSServerInfo.java:113)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:499)
        at com.netscape.admin.certsrv.CMSAdmin.run(CMSAdmin.java:548)
        at com.netscape.admin.certsrv.Console.main(Console.java:1655)
```

问：

我试图启动控制台，系统会提示我输入我的用户名和密码。输入这些凭证后，控制台将无法显示。

答：

确保输入的用户名和密码有效。如果是，启用 debug 输出并检查它。

要启用 debug 输出，打开 /usr/bin/pkiconsole 文件，并添加以下行：

```
==========================================
${JAVA} ${JAVA_OPTIONS} -cp ${CP} -Djava.util.prefs.systemRoot=/tmp/.java -
Djava.util.prefs.userRoot=/tmp/java com.netscape.admin.certsrv.Console -s instanceID -D 9:all
-a $1
----------
note: "-D 9:all" is for verbose output on the console.
==========================================
```

部分 III. 配置证书系统

第 14 章 证书系统配置文件

每个子系统的主配置文件都是其 CS.cfg 文件。本章论述了编辑 CS.cfg 文件的基本信息。本章还介绍了子系统使用的一些其他有用配置文件，如密码和 Web 服务文件。

14.1. 证书系统子系统的文件和目录位置

证书系统服务器由 Apache Tomcat 实例组成，其中包含 一个或多个子系统。每个子系统包含一个 Web 应用，它处理特定类型的 PKI 功能请求。

可用的子系统有：CA、KRA、OCSP、TKS 和 TPS。每个实例只能包含每种 PKI 子系统之一。

子系统可以使用 pkispawn 命令在特定的实例中安装。

14.1.1. 实例特定信息

有关默认实例(pki-tomcat)的实例信息，请参阅 ???

表 14.1. 证书服务器端口分配（默认）

端口类型	端口号	备注
安全端口	8443	用于通过 HTTPS 通过最终用户、代理和管理员访问 PKI 服务的主要端口。
不安全的端口	8080	用于通过 HTTP 对一些端点功能进行非安全访问服务器。用于提供已签名的 CRL，因此不需要加密。
AJP 端口	8009	用于通过 AJP 连接从前端 Apache 代理服务器访问服务器。重定向到 HTTPS 端口。
Tomcat 端口	8005	由 Web 服务器使用。

14.1.2. CA 子系统信息

本节包含有关 CA 子系统的详细信息，这是可作为 Web 应用在证书服务器实例中安装的可能子系统之一。

表 14.2. 默认实例的 CA 子系统信息(pki-tomcat)

设置	值
主目录	/var/lib/pki/pki-tomcat/ca/
配置目录	/var/lib/pki/pki-tomcat/ca/conf/[a]
配置文件	/var/lib/pki/pki-tomcat/ca/conf/CS.cfg
子系统证书	CA 签名证书
	OCSP 签名证书（用于 CA 的内部 OCSP 服务）
	TLS 服务器证书
	审计日志签名证书
	子系统证书[b]
安全数据库	/var/lib/pki/pki-tomcat/alias/[c]
日志文件	/var/log/pki/pki-tomcat/ca/logs/[d]
安装日志	/var/log/pki/pki-ca-spawn.*date*.log
卸载日志	/var/log/pki/pki-ca-destroy.*date*.log
审计日志	/var/log/pki/pki-tomcat/ca/signedAudit/
配置集文件	/var/lib/pki/pki-tomcat/ca/profiles/ca/
电子邮件通知模板	/var/lib/pki/pki-tomcat/ca/emails/
Web 服务文件	*代理服务：* /var/lib/pki/pki-tomcat/ca/webapps/ca/agent/
	管理服务： /var/lib/pki/pki-tomcat/ca/webapps/ca/admin/
	最终用户服务： /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/

[a] 别名化为 /etc/pki/pki-tomcat/ca/

[b] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。

[c] 请注意，所有子系统证书都存储在实例安全数据库中

[d] 别名化为 /var/lib/pki/pki-tomcat/ca

14.1.3. KRA 子系统信息

本节包含有关 KRA 子系统的详细信息，这是可作为 Web 应用在证书服务器实例中安装的可能子系统之一。

表 14.3. 默认实例的 KRA 子系统信息(pki-tomcat)

设置	值
主目录	/var/lib/pki/pki-tomcat/kra/
配置目录	/var/lib/pki/pki-tomcat/kra/conf/[a]
配置文件	/var/lib/pki/pki-tomcat/kra/conf/CS.cfg
子系统证书	传输证书
	存储证书
	TLS 服务器证书
	审计日志签名证书
	子系统证书[b]
安全数据库	/var/lib/pki/pki-tomcat/alias/[c]
日志文件	/var/lib/pki/pki-tomcat/kra/logs/
安装日志	/var/log/pki/pki-kra-spawn-*date*.log
卸载日志	/var/log/pki/pki-kra-destroy-*date*.log
审计日志	/var/log/pki/pki-tomcat/kra/signedAudit/
Web 服务文件	*代理服务：* /var/lib/pki/pki-tomcat/kra/webapps/kra/agent/
	管理服务： /var/lib/pki/pki-tomcat/kra/webapps/kra/admin/

[a] 链接到 /etc/pki/pki-tomcat/kra/

[b] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。

[c] 请注意，所有子系统证书都存储在实例安全数据库中

14.1.4. OCSP 子系统信息

本节包含有关 OCSP 子系统的详细信息，这是可作为证书服务器实例中的 Web 应用安装的可能子系统之一。

表 14.4. 默认实例的 OCSP 子系统信息(pki-tomcat)

设置	值
主目录	/var/lib/pki/pki-tomcat/ocsp/
配置目录	/var/lib/pki/pki-tomcat/ocsp/conf/[a]
配置文件	/var/lib/pki/pki-tomcat/ocsp/conf/CS.cfg
子系统证书	传输证书
	存储证书
	TLS 服务器证书
	审计日志签名证书
	子系统证书[b]
安全数据库	/var/lib/pki/pki-tomcat/alias/[c]
日志文件	/var/lib/pki/pki-tomcat/ocsp/logs/
安装日志	/var/log/pki/pki-ocsp-spawn-*date*.log
卸载日志	/var/log/pki/pki-ocsp-destroy-*date*.log
审计日志	/var/log/pki/pki-tomcat/ocsp/signedAudit/
Web 服务文件	*代理服务：* /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/agent/
	管理服务： /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/admin/

[a] 链接到 /etc/pki/pki-tomcat/ocsp/

[b] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。

[c] 请注意，所有子系统证书都存储在实例安全数据库中

14.1.5. TKS 子系统信息

本节包含有关 TKS 子系统的详细信息，这是可作为证书服务器实例中的 Web 应用程序安装的可能子

系统之一。

表 14.5. 每次通过初始安装创建子系统时，或使用(pki-tomcat)创建其他实例。

设置	值
主目录	/var/lib/pki/pki-tomcat/tks/
配置目录	/var/lib/pki/pki-tomcat/tks/conf/[a]
配置文件	/var/lib/pki/pki-tomcat/tks/conf/CS.cfg
子系统证书	传输证书
	存储证书
	TLS 服务器证书
	审计日志签名证书
	子系统证书[b]
安全数据库	/var/lib/pki/pki-tomcat/alias/[c]
日志文件	/var/lib/pki/pki-tomcat/tks/logs/
安装日志	/var/log/pki/pki-tks-spawn-*date*.log
卸载日志	/var/log/pki/pki-tks-destroy-*date*.log
审计日志	/var/log/pki/pki-tomcat/tks/signedAudit/
Web 服务文件	*代理服务：* /var/lib/pki/pki-tomcat/tks/webapps/tks/agent/
	管理服务： /var/lib/pki/pki-tomcat/tks/webapps/tks/admin/

[a] 链接到 /etc/pki/pki-tomcat/tks/

[b] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。

[c] 请注意，所有子系统证书都存储在实例安全数据库中

14.1.6. TPS 子系统信息

本节包含有关 TPS 子系统的详细信息，这是可作为 Web 应用在证书服务器实例中安装的可能子系统之一。

表 14.6. 默认实例的 TPS 子系统信息(pki-tomcat)

设置	值
主目录	/var/lib/pki/pki-tomcat/tps
配置目录	/var/lib/pki/pki-tomcat/tps/conf/[a]
配置文件	/var/lib/pki/pki-tomcat/tps/conf/CS.cfg
子系统证书	传输证书
	存储证书
	TLS 服务器证书
	审计日志签名证书
	子系统证书[b]
安全数据库	/var/lib/pki/pki-tomcat/alias/[c]
日志文件	/var/lib/pki/pki-tomcat/tps/logs/
安装日志	/var/log/pki/pki-tps-spawn-*date*.log
卸载日志	/var/log/pki/pki-tps-destroy-*date*.log
审计日志	/var/log/pki/pki-tomcat/tps/signedAudit/
Web 服务文件	*代理服务：* /var/lib/pki/pki-tomcat/tps/webapps/tps/agent/
	管理服务： /var/lib/pki/pki-tomcat/tps/webapps/tps/admin/

[a] 链接到 /etc/pki/pki-tomcat/tps/

[b] 子系统证书始终由安全域发布，以便需要客户端身份验证的域级别操作基于此子系统证书。

[c] 请注意，所有子系统证书都存储在实例安全数据库中

14.1.7. 共享证书系统子系统文件位置

有一些目录供所有证书系统子系统实例用于常规服务器操作，列在 ??? 中。

表 14.7. 子系统文件位置

目录位置	内容
/var/lib/*instance_name*	包含主实例目录，这是特定于用户的目录位置，以及自定义配置文件、配置文件、证书数据库、Web 文件，以及子系统实例的其他文件的位置。
/usr/share/java/pki	包含由证书系统子系统共享的 Java 存档文件。除了所有子系统的共享文件外，子文件夹中还有特定于子系统的文件： pki/ca/ (CA) pki/kra/ (KRA) pki/ocsp/ (OCSP) pki/tks/ (TKS) *不是由 TPS 子系统使用。*
/usr/share/pki	包含用于创建证书系统实例的常用文件和模板。除了所有子系统的共享文件外，子文件夹中还有特定于子系统的文件： pki/ca/ (CA) pki/kra/ (KRA) pki/ocsp/ (OCSP) pki/tks/ (TKS) pki/tps (TPS)
/usr/bin	包含 **pkispawn** 和 **pkidestroy** 实例配置脚本，以及由证书系统子系统共享的 Java、原生和安全性。
/var/lib/tomcat5/common/lib	包含指向本地 Tomcat Web 应用程序共享的 Java 归档文件的链接，并由证书系统子系统共享。*不是由 TPS 子系统使用。*
/var/lib/tomcat5/server/lib	包含本地 Tomcat Web 服务器使用的 Java 归档文件的链接，并由证书系统子系统共享。*不是由 TPS 子系统使用。*
/usr/shared/pki	包含 Tomcat 服务器以及证书系统实例使用的应用程序所使用的 Java 归档文件。*不是由 TPS 子系统使用。*

目录位置	内容
/usr/lib/httpd/modules /usr/lib64/httpd/modules	包含 TPS 子系统使用的 Apache 模块。*没有被 CA、KRA、OCSP 或 TKS 子系统使用。*
/usr/lib/mozldap /usr/lib64/mozldap	TPS 子系统使用的 Mozilla LDAP SDK 工具。*没有被 CA、KRA、OCSP 或 TKS 子系统使用。*

14.2. CS.CFG FILES

证书系统子系统的运行时属性由一组配置参数进行管理。这些参数存储在服务器在启动期间读取的文件 CS.cfg。

在首次安装子系统时，创建 CS.cfg 是一个 ASCII 文件，并使用适当的配置参数填充。修改实例功能的方式是通过子系统控制台进行更改的，这是推荐的方法。管理控制台中所做的更改反映在 配置文件中。

也可以直接编辑 CS.cfg 配置文件，在某些情况下，这是管理子系统的最简单方法。

14.2.1. 查找 CS.cfg 文件

证书系统子系统的每个实例都有自己的配置文件 CS.cfg。每个子系统实例的文件内容根据子系统配置的方式、其他设置和配置（如添加新配置文件或启用自测试）以及子系统类型的不同而有所不同。

CS.cfg 文件位于实例的配置目录中。

```
/var/lib/pki/instance_name/subsystem_type/conf
```

例如：

```
/var/lib/pki/instance_name/ca/conf
```

14.2.2. 编辑配置文件

WARNING

不要在不熟悉配置参数的情况下直接编辑配置文件，或者确保更改对服务器可以接受。如果配置文件被错误修改，证书系统将无法启动。不正确的配置也可以导致数据丢失。

修改 CS.cfg 文件：

1. 停止子系统实例。

```
# pki-server stop instance_name
```

或（如果使用 nuxwdog watchdog）

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

当实例启动时，配置文件存储在缓存中。通过控制台对实例所做的任何更改都会在文件的缓存版本中改变。当服务器停止或重启时，缓存中存储的配置文件将写入磁盘。在编辑配置文件或更改更改之前，服务器停止服务器将会被缓存的版本覆盖。

2. 打开 /var/lib/pki/instance_name/subsystem_type/conf 目录。

3. 在文本编辑器中打开 CS.cfg 文件。

4. 编辑文件中的参数，并保存更改。

5. 启动子系统实例。

```
# pki-server start instance_name
```

或（如果使用 nuxwdog watchdog）

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

14.2.3. CS.cfg 配置文件概述

每个子系统实例都有自己的主配置文件 CS.cfg，其中包含实例的所有设置，如插件和用于配置的 Java 类。参数和特定设置根据子系统类型的不同，但通常情况下，CS.cfg 文件定义了子系统实例的这些部分：

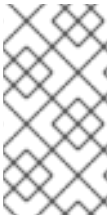
- 基本子系统实例信息，如其名称、端口分配、实例目录和主机名
- 日志记录
- 插件和方法，对实例的用户目录（授权）进行身份验证
- 实例所属的安全域
- 子系统证书
- 子系统实例使用的其他子系统
- 子系统使用的数据库类型和实例
- PKI 相关任务的设置，如 TKS 中的密钥配置文件、CA 中的证书配置文件以及 KRA 中密钥恢复所需的代理

许多配置参数（来自 PKI 任务除外）在 CA、OCSP、KRA 和 TKS 之间非常相似，因为它们都使用基于 Java 的控制台，因此可以在控制台中管理的配置设置具有类似的参数。

CS.cfg 文件是一个基本的 parameter=value 格式。

```
#comment
parameter=value
```

在 CS.cfg 文件中，许多参数块都有描述性注释，使用 pound (#)字符进行注释。服务器会忽略注释、空行、未知参数或拼写错误的参数。



注意

TPS 中的一个错误可防止它忽略在 CS.cfg 文件中注释掉的行。无需在 TPS CS.cfg 文件中注释掉行，只需删除这些行即可。

配置实例相同区域的参数往往分组到同一块中。

某些功能区域通过插件（如自我测试、作业和授权）来实现，以访问子系统。对于这些参数，插件实例具有唯一标识符（因为对于子系统调用的相同插件的多个实例）、实施插件名称和 Java 类。

例 14.1. 子系统授权设置

```
authz.impl._000=##
authz.impl._001=## authorization manager implementations
authz.impl._002=##
authz.impl.BasicAclAuthz.class=com.netscape.cms.authorization.BasicAclAuthz
authz.instance.BasicAclAuthz.pluginName=BasicAclAuthz
```

注意

配置参数的值必须正确格式化，因此它们必须遵循两个规则：

- 需要本地化的值必须在 UTF8 字符中。
- CS.cfg 文件支持参数值中的正斜杠(/)。如果在值中需要反斜杠(\)，则必须用反斜杠转义，即必须使用一行中的两个反斜杠。

以下部分是 CS.cfg 文件设置和参数的快照。这些不是 CS.cfg 文件参数的详细参考或示例。此外，每个子系统配置文件中可用的参数也非常不同，尽管有相似性。

14.2.3.1. 基本子系统设置

基本设置特定于实例本身，而不与子系统的功能或行为直接相关。这包括实例名称、根目录、进程的用户 ID 和端口号的设置。首次安装或配置实例时分配的许多设置都以 pkispawn 开头。

例 14.2. CA 的基本实例参数： pkispawn 文件 ca.cfg

```
[DEFAULT]
pki_admin_password=Secret.123
pki_client_pkcs12_password=Secret.123
pki_ds_password=Secret.123
# Optionally keep client databases
pki_client_database_purge=False
# Separated CA instance name and ports
pki_instance_name=pki-ca
 pki_http_port=18080
pki_https_port=18443
# This Separated CA instance will be its own security domain
pki_security_domain_https_port=18443

[Tomcat]
# Separated CA Tomcat ports
pki_ajp_port=18009
pki_tomcat_server_port=18005
```

重要

虽然类似端口设置的信息 包含在 CS.cfg 文件中，但它没有在 CS.cfg 中设置。服务器配置在 server.xml 文件中设置。

CS.cfg 和 server.xml 中的端口 必须与 正常工作的 RHCS 实例匹配。

14.2.3.2. 日志记录设置

根据子系统类型，可以配置几种不同类型的日志。每种日志在 CS.cfg 文件中都有自己的配置条目。

有关日志记录设置的更多信息，请参阅 第 18.1 节 “证书系统日志设置”。

14.2.3.3. 认证和授权设置

CS.cfg 文件设置如何识别用户访问子系统实例（身份验证）以及每个经过身份验证的用户批准（授权）的操作。

CS 子系统使用身份验证插件来定义登录子系统的方法。

以下示例显示了一个名为 SharedToken 的身份验证实例，它实例化名为 SharedSecret 的 JAVA 插件。

```
auths.impl.SharedToken.class=com.netscape.cms.authentication.SharedSecret
auths.instance.SharedToken.pluginName=SharedToken
auths.instance.SharedToken.dnpattern=
auths.instance.SharedToken.ldap.basedn=ou=People,dc=example,dc=org
auths.instance.SharedToken.ldap.ldapauth.authtype=BasicAuth
auths.instance.SharedToken.ldap.ldapauth.bindDN=cn=Directory Manager
auths.instance.SharedToken.ldap.ldapauth.bindPWPrompt=Rule SharedToken
auths.instance.SharedToken.ldap.ldapauth.clientCertNickname=
auths.instance.SharedToken.ldap.ldapconn.host=server.example.com
auths.instance.SharedToken.ldap.ldapconn.port=636
auths.instance.SharedToken.ldap.ldapconn.secureConn=true
auths.instance.SharedToken.ldap.ldapconn.version=3
auths.instance.SharedToken.ldap.maxConns=
auths.instance.SharedToken.ldap.minConns=
auths.instance.SharedToken.ldapByteAttributes=
auths.instance.SharedToken.ldapStringAttributes=
auths.instance.SharedToken.shrTokAttr=shrTok
```

对于某些授权设置，可以选择使用 LDAP 数据库存储用户条目的授权方法，在这种情况下，数据库设置与插件一起配置，如下所示。

```
authz.impl.DirAclAuthz.class=com.netscape.cms.authorization.DirAclAuthz
authz.instance.DirAclAuthz.ldap=internaldb
authz.instance.DirAclAuthz.pluginName=DirAclAuthz
authz.instance.DirAclAuthz.ldap._000=##
authz.instance.DirAclAuthz.ldap._001=## Internal Database
authz.instance.DirAclAuthz.ldap._002=##
authz.instance.DirAclAuthz.ldap.basedn=dc=server.example.com-pki-ca
authz.instance.DirAclAuthz.ldap.database=server.example.com-pki-ca
authz.instance.DirAclAuthz.ldap.maxConns=15
authz.instance.DirAclAuthz.ldap.minConns=3
authz.instance.DirAclAuthz.ldap.ldapauth.authtype=SslClientAuth
authz.instance.DirAclAuthz.ldap.ldapauth.bindDN=cn=Directory Manager
authz.instance.DirAclAuthz.ldap.ldapauth.bindPWPrompt=Internal LDAP Database
authz.instance.DirAclAuthz.ldap.ldapauth.clientCertNickname=
authz.instance.DirAclAuthz.ldap.ldapconn.host=localhost
authz.instance.DirAclAuthz.ldap.ldapconn.port=11636
authz.instance.DirAclAuthz.ldap.ldapconn.secureConn=true
authz.instance.DirAclAuthz.ldap.multipleSuffix.enable=false
```

有关安全配置 LDAP 和参数说明的更多信息，请参阅 第 7.10.3 节 “启用 TLS 客户端身份验证”。参数路径与显示的内容不同，但两个位置都允许使用相同的名称和值。

CA 还必须有批准用户请求的机制。与配置授权一样，这可以通过识别适当的身份验证插件并为它配置实例来实现：

```
auths.impl.AgentCertAuth.class=com.netscape.cms.authentication.AgentCertAuthentication
auths.instance.AgentCertAuth.agentGroup=Certificate Manager Agents
auths.instance.AgentCertAuth.pluginName=AgentCertAuth
```

14.2.3.4. 子系统证书设置

一些子系统在配置文件中，每个子系统证书都有条目。

```
ca.sslserver.cert=MIIDmDCCAoCgAwIBAgIBAzANBgkqhkiG9w0BAQUFADBAMR4wHAYDVQQ
KExVSZWR...
ca.sslserver.certreq=MIICizCCAXMCAQAwRjEeMBwGA1UEChMVUmVkYnVkY29tcHV0ZXIgRG
9tYWluMSQwIgYDV...
ca.sslserver.nickname=Server-Cert cert-pki-ca
ca.sslserver.tokenname=Internal Key Storage Token
```

14.2.3.5. 所需子系统的设置

至少，每个子系统依赖于一个 CA，这意味着必须在子系统的设置中配置 CA （以及任何其他必要的子系统）。与另一个子系统的任何连接都以 conn 开头，然后是子系统类型和编号。

```
conn.ca1.clientNickname=subsystemCert cert-pki-tps
conn.ca1.hostadminport=server.example.com:8443
conn.ca1.hostagentport=server.example.com:8443
conn.ca1.hostport=server.example.com:9443
conn.ca1.keepAlive=true
conn.ca1.retryConnect=3
conn.ca1.servlet.enrollment=/ca/ee/ca/profileSubmitSSLClient
conn.ca1.servlet.renewal=/ca/ee/ca/profileSubmitSSLClient
conn.ca1.servlet.revoke=/ca/subsystem/ca/doRevoke
conn.ca1.servlet.unrevoke=/ca/subsystem/ca/doUnrevoke
conn.ca1.timeout=100
```

14.2.3.6. 数据库设置

所有子系统都使用 LDAP 目录来存储其信息。此内部数据库在 internaldb 参数中配置，但使用许多其他配置设置在 tokendb 参数中配置 TPS 除外。

```
internaldb._000=##
internaldb._000=##
internaldb._001=## Internal Database
internaldb._002=##
internaldb.basedn=o=pki-tomcat-ca-SD
```

```
internaldb.database=pki-tomcat-ca
internaldb.maxConns=15
internaldb.minConns=3
internaldb.ldapauth.authtype=SslClientAuth
internaldb.ldapauth.clientCertNickname=HSM-A:subsystemCert pki-tomcat-ca
internaldb.ldapconn.host=example.com
internaldb.ldapconn.port=11636
internaldb.ldapconn.secureConn=true
internaldb.multipleSuffix.enable=false
```

有关安全配置 LDAP 和参数说明的详情，请参考 第 7.10.3 节 “启用 TLS 客户端身份验证”。除了作为 第 7.10.3 节 “启用 TLS 客户端身份验证” 的一部分外，不需要额外的配置。

14.2.3.7. 启用和配置发布队列

注册流程的一部分包括将发布的证书发布到任何目录或文件。这基本上会关闭初始证书请求。但是，向外部网络发布证书可能会显著减慢颁发进程 - 这会使请求保持打开。

为避免这种情况，管理员可以启用 发布队列。发布队列将发布操作（可能涉及外部 LDAP 目录）与使用单独的请求队列的请求和注册操作分隔开。请求队列会立即更新，以显示注册过程已完成，而发布队列则以网络流量的速度发送信息。

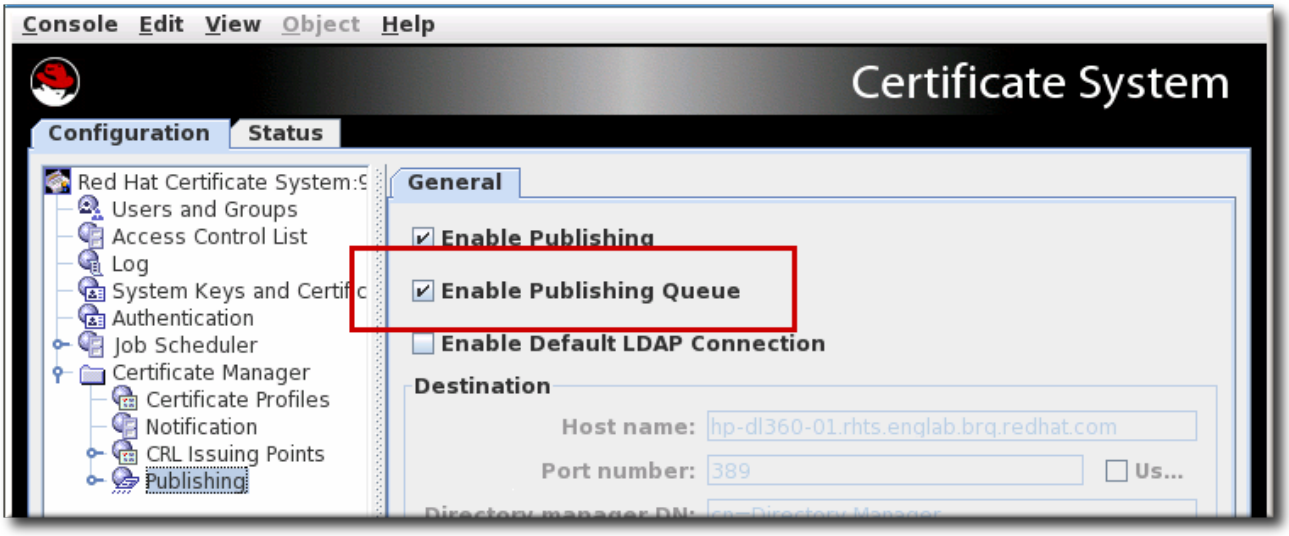
发布队列设置定义的、有限数量的线程，用于发布生成的证书，而不是为每个批准的证书打开新线程。

默认情况下，发布队列被禁用。它可以在 CA 控制台中启用，以及启用发布。

注意

在默认情况下，如果 控制台中 启用了 LDAP 发布，则自动启用发布队列。否则，可以手动启用队列。

图 14.1. 启用发布队列



14.2.3.7.1. 通过编辑 CS.cfg 文件启用和配置发布队列

通过编辑 CS.cfg 文件启用发布队列，管理员可以设置其他选项以进行发布，如用于发布操作的线程数量和队列页大小。

1. 停止 CA 服务器，以便您可以编辑配置文件。

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

2. 打开 CA 的 CS.cfg 文件。

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

3. 将 ca.publish.queue.enable 设置为 true。如果参数不存在，则使用 参数添加一行。

```
ca.publish.queue.enable=true
```

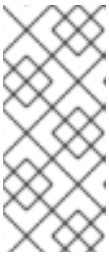
4. 设置其他相关的发布队列参数：

- ca.publish.queue.maxNumberOfThreads 设置可为发布操作打开的最大线程数。默认值为 3。
- ca.publish.queue.priorityLevel 设置发布操作的优先级。优先级值范围从 - 2 （最低优先级）到 2 （最高优先级）。零(0)是普通优先级，也是默认值。

- ca.publish.queue.pageSize 设置可在发布队列页面中存储的最大请求数。默认值为 40。

- ca.publish.queue.saveStatus 设置间隔，以保存其每个指定数量的发布操作的状态。这允许在 CA 重启或崩溃时恢复发布队列。默认值为 200，但任何非零数将在 CA 重启时恢复队列。将此参数设置为 0 可禁用队列恢复。

```
ca.publish.queue.maxNumberOfThreads=1
ca.publish.queue.priorityLevel=0
ca.publish.queue.pageSize=100
ca.publish.queue.saveStatus=200
```



TIP

将 ca.publish.queue.enable 设置为 false，将 ca.publish.queue.maxNumberOfThreads 设置为 0 可禁用发布队列，并将单独的线程用于发布的证书。

5. 重启 CA 服务器。

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

14.2.3.8. PKI 任务的设置

CS.cfg 文件用于为每个子系统配置 PKI 任务。每个子系统的参数都有所不同，没有任何重叠。

例如，KRA 具有恢复密钥所需的代理数量的设置。

```
kra.noOfRequiredRecoveryAgents=1
```

查看每个子系统的 CS.cfg 文件以熟悉其 PKI 任务设置；文件中的注释是学习不同参数的指导。

- CA 配置文件列出所有证书配置集和策略设置，以及生成 CRL 的规则。

- TPS 配置不同的令牌操作。
- TKS 列出从不同密钥类型派生密钥的配置集。
- OCSP 为不同的密钥集设置密钥信息。

14.2.3.9. 更改 CA-Isued 证书中的 DN 属性

在证书系统发布的证书中，DN 识别拥有证书的实体。在所有情况下，如果证书系统与目录服务器连接，则证书中的 DN 格式应与目录中的 DN 格式匹配。名称不完全匹配；证书映射允许证书中的主题 DN 与目录中的对象 DN 不同。

在证书系统中，DN 基于 X.509 标准中定义的组件或属性。表 14.8 “值类型允许的 Characters” 列出默认支持的属性。属性集合可扩展。

表 14.8. 值类型允许的 Characters

属性	值类型	对象标识符
cn	DirectoryString	2.5.4.3
ou	DirectoryString	2.5.4.11
o	DirectoryString	2.5.4.10
c	可打印字符串，双字符	2.5.4.6
l	DirectoryString	2.5.4.7
st	DirectoryString	2.5.4.8
street	DirectoryString	2.5.4.9
title	DirectoryString	2.5.4.12
uid	DirectoryString	0.9.2342.19200300.100.1.1
mail	IA5String	1.2.840.113549.1.9.1

属性	值类型	对象标识符
dc	IA5String	0.9.2342.19200300.100.1.2.25
serialNumber	PrintableString	2.5.4.5
unstructuredname	IA5String	1.2.840.113549.1.9.2
unstructuredaddress	PrintableString	1.2.840.113549.1.9.8

默认情况下，证书系统支持 表 14.8 “值类型允许的 Characters” 中标识的属性。此支持的属性列表可通过创建或添加新属性来扩展。添加额外的 X.500Name 属性或组件的语法如下：

```
X500Name.NEW_ATTRNAME.oid=n.n.n.n
X500Name.NEW_ATTRNAME.class=string_to_DER_value_converter_class
```

值转换器类将字符串转换为 ASN.1 值；此类必须实施 netscape.security.x509.AVAValueConverter 接口。string-to-value 转换类可以是以下之一：

- Netscape.security.x509.PrintableConverter 将 字符串转换为可打印字符串值。字符串必须只有可打印的字符。
- Netscape.security.x509.IA5StringConverter 将字符串转换为 IA5String 值。字符串必须具有 IA5String 字符。
- Netscape.security.x509.DirStrConverter 将字符串转换为 DirectoryString。根据 RFC 2253，字符串预期为 DirectoryString 格式。
- Netscape.security.x509.GenericValueConverter 按以下顺序将字符串字符转换为最大字符集：
  - PrintableString
  - IA5String

- BMPString
- 通用字符串

属性条目类似如下：

```
X500Name.MY_ATTR.oid=1.2.3.4.5.6
X500Name.MY_ATTR.class=netscape.security.x509.DirStrConverter
```

14.2.3.9.1. 添加新或自定义属性

要在证书系统模式中添加新的或专有属性，请执行以下操作：

1. 停止证书管理器。

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

2. 打开 /var/lib/pki/cs_instance/conf/ 目录。

3. 打开配置文件 CS.cfg。

4. 将新属性添加到配置文件中。

   例如，要添加三个专有属性，即 MYATTR1，它是 DirectoryString,MYATTR2 是一个 IA5String，而 MYATTR3 是 可打印字符串，请在配置文件末尾添加以下行：

```
X500Name.attr.MYATTR1.oid=1.2.3.4.5.6
X500Name.attr.MYATTR1.class=netscape.security.x509.DirStrConverter
X500Name.attr.MYATTR2.oid=11.22.33.44.55.66
X500Name.attr.MYATTR2.class=netscape.security.x509.IA5StringConverter
X500Name.attr.MYATTR3.oid=111.222.333.444.555.666
X500Name.attr.MYATTR3.class=netscape.security.x509.PrintableConverter
```

5. 保存更改并关闭该文件。

6. 重启证书管理器。

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

7. 重新加载注册页面并验证更改；新属性应当以表单中显示。

8. 要验证新属性是否生效，请使用手动注册表请求证书。

输入新属性的值，以便它可以验证它们出现在证书主题名称中。例如，为新属性输入以下值，并在主题名称中查找它们：

```
MYATTR1: a_value
MYATTR2: a.Value
MYATTR3: aValue
cn: John Doe
o: Example Corporation
```

9. 打开代理服务页面，并批准请求。

10. 发布证书后，检查主题名称。证书应该显示主题名称中的新属性值。

14.2.3.9.2. 更改 DER-Encoding 顺序

可以更改 DirectoryString 的 DER-encoding 顺序，以便配置字符串，因为不同的客户端支持不同的编码。

更改 DirectoryString 的 DER-encoding 顺序的语法如下：

```
X500Name.directoryStringEncodingOrder=encoding_list_separated_by_commas
```

可能的编码值如下：

- PrintableString

- *IA5String*
- *UniversalString*
- *BMPString*
- *UTF8String*

例如，DER-encoding 排序可以列出，如下所示：

```
X500Name.directoryStringEncodingOrder=PrintableString,BMPString
```

要更改 DirectoryString 编码，请执行以下操作：

1. *停止证书管理器。*

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

2. *打开 /var/lib/pki/cs_instance/conf/ 目录。*

3. *打开 CS.cfg 配置文件。*

4. *将编码顺序添加到配置文件。*

   例如，要指定两个编码值：PrintableString 和 UniversalString，并且编码顺序是 PrintableString，然后是 UniversalString，请在配置文件末尾添加以下行：

   ```
   X500Name.directoryStringEncodingOrder=PrintableString,UniversalString
   ```

5. *保存更改并关闭该文件。*

6.
启动 证书管理器。

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

7.
要验证编码订单是否生效，请使用手动注册表注册证书。对 cn 使用 John_Doe。

8.
打开代理服务页面，并批准请求。

9.
发布证书后，使用 dumpasn1 工具检查证书的编码。

主题名称的 cn 组件应编码为 UniversalString。

10.
使用 John Smith 为 cn 创建并提交新请求。

主题名称的 cn 组件应编码为 可打印字符串。

14.2.3.10. 将 CA 设置为使用不同的证书来签名 CRL

证书管理器使用与其 OCSP 签名证书对应的密钥对来签名证书和证书撤销列表(CRL)。要使用不同的密钥对为 Certificate Manager 生成的 CRL 签名，可以创建 CRL 签名证书。证书管理器的 CRL 签名证书必须自行签名或发布。

要启用证书管理器使用不同的密钥对为 CRL 签名，请执行以下操作：

1.
为证书管理器请求 CRL 签名证书。

或者，使用能够生成密钥对的工具，如 certutil 工具生成密钥对，为密钥对请求证书，并在证书管理器的证书数据库中安装证书。有关 certutil 工具的详情请参考 http://www.mozilla.org/projects/security/pki/nss/tools/。

2.
创建证书请求后，通过证书管理器终端页面提交它，选择正确的配置文件，如"Manual OCSP Manager Signing Certificate Enrollment"配置文件。该页面具有以下格式的 URL：

```
https://hostname:port/ca/ee/ca
```

3. 提交请求后，登录代理服务页面。

4. 检查请求所需的扩展。CRL 签名证书必须包含设置 crlSigning 位的 Key Usage 扩展。

5. 批准请求。

6. 生成 CRL 签名证书后，通过控制台中的 System Keys 和 Certificates 在证书管理器的数据库中安装证书。

7. 停止证书管理器。

```
# pki-server stop instance_name
```

8. 更新证书管理器的配置，以识别新密钥对和证书。

   a. 进入 Certificate Manager 实例配置目录。

   ```
   # cd /var/lib/pki/instance-name/ca/conf/
   ```

   b. 打开 CS.cfg 文件并添加以下行：

   ```
   ca.crl_signing.cacertnickname=nickname cert-instance_ID
   ca.crl_signing.defaultSigningAlgorithm=signing_algorithm
   ca.crl_signing.tokenname=token_name
   ```

   nickname 是分配给 CRL 签名证书的名称。

instance_ id 是证书管理器实例的名称。

如果安装的 CA 是基于 RSA 的 CA，则 signing_algorithm 可以是 SHA256withRSA,SHA384withRSA, 或 SHA512withRSA。如果安装的 CA 是基于 EC 的 CA，则 signing_algorithm 可以是 SHA256withEC，SHA384withEC,SHA512withEC.

token_name 是用于生成密钥对和证书的令牌名称。如果使用内部/软件令牌，请使用 Internal Key Storage Token 作为值。

例如，条目可能类似如下：

```
ca.crl_signing.cacertnickname=crlSigningCert cert-pki-ca
ca.crl_signing.defaultSigningAlgorithm=SHAMD512withRSA
ca.crl_signing.tokenname=Internal Key Storage Token
```

c. 保存更改并关闭该文件。

9. 重启证书管理器。

```
# pki-server restart instance_name
```

现在，证书管理器可以使用 CRL 签名证书为它生成的 CRL 签名证书进行签名。

14.2.3.11. 从 CS.cfg 中的缓存配置 CRL Generation

CRL 缓存是一种简单的机制，它允许从内存中维护的撤销信息集合中获取证书撤销信息。为了获得最佳性能，建议启用此功能，它已代表默认行为。以下配置信息（默认）显示为信息目的，或者需要更改。

1. 停止 CA 服务器。

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

2. 打开 CA 配置目录。

```
# cd /var/lib/instance_name/conf/
```

3. 编辑 CS.cfg 文件，将 enableCRLCache 和 enableCacheRecovery 参数设置为 true ：

```
ca.crl.MasterCRL.enableCRLCache=true
ca.crl.MasterCRL.enableCacheRecovery=true
```

4. 启动 CA 服务器。

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

14.2.3.12. 在 CS.cfg 中为 CRL 配置 Update Intervals

下面介绍如何灵活地配置 CRL 系统，以反映所需的行为。目标是根据两种类型的一些调度配置 CRL 更新。一个类型允许列表显式时间，另一个类型由更新间隔长度组成。还有一个混合场景，两者都被启用以考虑偏移。下面的 Note 条目实际上代表开箱即用的情况。

默认场景如下：

```
ca.crl.MasterCRL.updateSchema=3
ca.crl.MasterCRL.enableDailyUpdates=true
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
ca.crl.MasterCRL.dailyUpdates=1:00
ca.crl.MasterCRL.nextUpdateGracePeriod=0
```

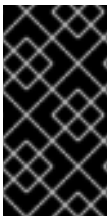
只有在需要更详细且需要特定的更新计划时，才会从此分离。部分的其余部分将讨论如何完成该操作。

在 CS.cfg 文件中为 full 和 delta CRL 配置设置涉及编辑参数。

表 14.9. CRL Extended Interval 参数

参数	描述	接受的值
updateSchema	设置每个完整 CRL 生成的 delta CRL 的比率	整数值
enableDailyUpdates	根据设置时间启用和禁用设置 CRL 更新	true 或 false

参数	描述	接受的值
enableUpdateInterval	根据设置间隔启用和禁用设置 CRL 更新	true 或 false
dailyUpdates	设置 CRL 应该更新的时间	以逗号分隔的时间列表
autoUpdateInterval	设置更新 CRL 的时间间隔（以分钟为单位）	整数值
autoUpdateInterval.effectiveAtStart	允许系统尝试立即使用自动更新的新值，而不是等待当前调度的 nextUpdate 时间	true 或 false
nextUpdateGracePeriod	将时间（分钟）添加到 CRL 的有效性周期，以确保 CRL 在发布或复制期间保持有效	整数值
refreshInSec	在克隆 OCSP 中设置线程的定期性（以秒为单位）以检查 LDAP 是否更新 CRL	整数值



重要

autoUpdateInterval.effectiveAtStart 参数需要一个系统重启才能应用新值。此参数的默认值为 false，只能由确保其正在做什么的用户更改。

过程 14.1. 如何在 CS.cfg 中配置 CRL 更新间隔

1. 停止 CA 服务器。

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

2. 进入 CA 配置目录。

```
# cd /var/lib/instance_name/conf/
```

3. 编辑 CS.cfg 文件，并添加以下行来设置更新间隔：

```
ca.crl.MasterCRL.updateSchema=3
```

默认间隔为 1，这意味着每次生成 CRL 时都会生成完整的 CRL。updateSchema 间隔可以设置为任何整数。

4. 通过指定一个循环间隔或设置更新发生时间来设置更新频率：

- 通过启用 enableDailyUpdates 参数来指定设置时间，并将所需时间添加到 dailyUpdates 参数中：

```
ca.crl.MasterCRL.enableDailyUpdates=true
ca.crl.MasterCRL.enableUpdateInterval=false
ca.crl.MasterCRL.dailyUpdates=0:50,04:55,06:55
```

此字段设置在 CRL 应该更新时的每日时间。要多次指定，请输入以逗号分隔的时间列表，如 01:50,04:55,06:55。要输入多个天数的调度，请输入以逗号分隔的列表，以在同一天内设置时间，然后使用分号分隔列表来标识不同天数的时间。例如，将 01:50,04:55,06:55;02:00,05:00,17:00 设置为在周期的 1:50am、4:55am 和 6:55am 以及第 2 天(5am 和 5pm)中配置撤销。

通过启用 enableUpdateInterval 参数来指定间隔，并将所需的间隔（以分钟为单位）添加到 autoUpdateInterval 参数：

```
ca.crl.MasterCRL.enableDailyUpdates=false
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
```

5. 根据您的环境设置以下参数：

- 如果您运行没有 OCSP 子系统的 CA，请设置：

```
ca.crl.MasterCRL.nextUpdateGracePeriod=0
```

- 如果您使用 OCSP 子系统运行 CA，请设置：

```
ca.crl.MasterCRL.nextUpdateGracePeriod=time_in_minutes
```

ca.crl.MasterCRL.nextUpdateGracePeriod 参数定义时间（以分钟为单位），值必须足够大，以便 CA 将新的 CRL 传播到 OCSP。您必须将参数设置为非零值。

如果您还在您的环境中有 OCSP 克隆，还要设置：

-

```
ocsp.store.defStore.refreshInSec=time_in_seconds
```

ocsp.store.defStore.refreshInSec 参数设置克隆 OCSP 实例通过来自 master OCSP 实例的 LDAP 复制更新更新的频率（以秒为单位）。

有关参数的详情，请查看 表 14.9 “CRL Extended Interval 参数”。

6. 重启 CA 服务器。

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

注意

当按间隔更新 CRL 时，可能会发生调度偏移。通常，偏移会因为手动更新和 CA 重启而发生。

要防止调度偏移，将 enableDailyUpdates 和 enableUpdateInterval 参数设置为 true，并将所需的值设置为 autoUpdateInterval 和 dailyUpdates ：

```
ca.crl.MasterCRL.enableDailyUpdates=true
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
ca.crl.MasterCRL.dailyUpdates=1:00
```

当按间隔更新 CRL 时，只会接受一个 dailyUpdates 值。

间隔更新将每 24 小时重新同步 dailyUpdates 值，防止调度偏移。

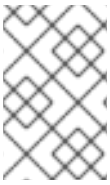
14.2.3.13. 更改子系统的访问控制设置

默认情况下，通过首先评估拒绝规则，然后通过评估 allow 规则来应用访问控制规则。要更改顺序，请在 CS.cfg 中更改 authz.evaluateOrder 参数。

```
authz.evaluateOrder=deny,allow
```

此外，还可以从本地 web.xml 文件(basic ACL)或更复杂的 ACL 评估访问控制规则，方法是检查 LDAP 数据库来访问。authz.sourceType 参数标识要使用的授权类型。

```
authz.sourceType=web.xml
```



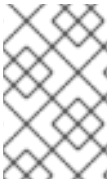
注意

在编辑 CS.cfg 文件以加载更新的设置后，始终重启子系统。

14.2.3.14. 为请求和序列号配置范围

如果没有使用随机序列号，如果克隆的系统，管理员可以指定范围证书系统用于 /etc/pki/instance_name/子系统/CS.cfg 文件中的请求和序列号：

```
dbs.beginRequestNumber=1001001007001
dbs.endRequestNumber=11001001007000
dbs.requestIncrement=10000000000000
dbs.requestLowWaterMark=2000000000000
dbs.requestCloneTransferNumber=10000
dbs.requestDN=ou=ca, ou=requests
dbs.requestRangeDN=ou=requests, ou=ranges
dbs.beginSerialNumber=1001001007001
dbs.endSerialNumber=11001001007000
dbs.serialIncrement=10000000000000
dbs.serialLowWaterMark=2000000000000
dbs.serialCloneTransferNumber=10000
dbs.serialDN=ou=certificateRepository, ou=ca
dbs.serialRangeDN=ou=certificateRepository, ou=ranges
dbs.beginReplicaNumber=1
dbs.endReplicaNumber=100
dbs.replicaIncrement=100
dbs.replicaLowWaterMark=20
dbs.replicaCloneTransferNumber=5
dbs.replicaDN=ou=replica
dbs.replicaRangeDN=ou=replica, ou=ranges
dbs.ldap=internaldb
dbs.newSchemaEntryAdded=true
```



注意

证书系统支持范围的 BigInteger 值。

14.2.3.15. 为 pkiconsole 设置要求以使用 TLS 客户端证书验证

注意

pkiconsole 已被弃用。

编辑每个子系统的 CS.cfg 文件，搜索 authType 参数并将其设置为如下：

```
authType=sslclientauth
```

14.3. 管理系统密码

如 第 2.3.10 节 “密码和 Watchdog (nuxwdog)” 所述，证书系统使用绑定到服务器的密码，或者在服务器启动时解锁令牌。

password.conf 文件以纯文本形式存储系统密码。但是，一些管理员更喜欢完全删除密码文件，以允许 nuxwdog 最初提示每个密码的手动条目，并在未计划关闭时存储 auto-restart。

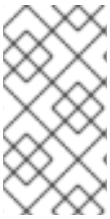
当证书系统实例启动时，子系统会自动检查 password.conf 文件。如果文件存在，则使用这些密码连接到其他服务，如内部 LDAP 数据库。如果该文件不存在，则 watchdog 守护进程会提示输入 PKI 服务器启动所需的所有密码。

注意

如果存在 password.conf 文件，子系统会假定存在所有必需的密码，并以明文格式正确格式化。如果任何密码缺失或格式错误，则系统无法正确启动。

所需的密码列在 CS.cfg 文件中的 cms.passwordlist 参数中：

```
cms.passwordlist=internaldb,replicationdb,CA LDAP Publishing
cms.password.ignore.publishing.failure=true
```

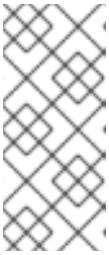


注意

cms.password.ignore.publishing.failure 参数允许 CA 子系统成功启动，即使它有与其中一个 LDAP 发布目录的连接失败。

对于 CA、KRA、OCSP 和 TKS 子系统，默认的预期密码是：

- NSS 数据库的内部
- 内部 LDAP 数据库的 internaldb
- replicationdb 用于复制密码
- 任何用于访问用于发布的外部 LDAP 数据库的密码（仅限 CA）



注意

如果在 password.conf 文件被删除后配置了发布程序，则不会将任何内容写入 password.conf 文件。除非配置了 nuxwdog，否则服务器将在下次实例重启时访问新发布密码的提示。

- 任何外部硬件令牌密码

对于 TPS，这提示输入三个密码：

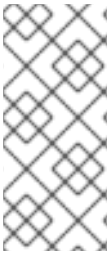
- NSS 数据库的内部
- 内部 LDAP 数据库的 tokendbpass
- 任何外部硬件令牌密码

本节论述了为证书系统提供用来检索这些密码的两个机制：

- password.conf 文件（默认）

- nuxwdog (watchdog)

14.3.1. 配置 password.conf 文件



注意

本节仅供参考。正确的和安全操作必须涉及使用 nuxwdog watchdog。请参阅 第 14.3.2 节 “使用证书系统 Watchdog 服务” 启用 nuxwdog，因为它是完全合规所必需的。

默认情况下，密码存储在子系统 conf/ 目录中的纯文本文件 password.conf 中。因此，只需通过文本编辑器修改它们。

此文件中存储的密码列表包括：

- 证书系统实例用于访问和更新内部数据库的绑定密码。
- HSM 的密码
- 证书系统实例用来访问身份验证目录的绑定密码，以防 CMC 共享文件系统服务。
- 子系统用来访问和更新 LDAP 发布目录的绑定密码；这只有在为发布证书进行了配置并且 CRL 到 LDAP 兼容目录时才需要。
- 子系统使用的绑定密码来访问其复制数据库。
- 对于 TPS 实例，用于访问和更新令牌数据库的绑定密码。

password.conf 文件还包含打开子系统私钥所需的令牌密码。

用于子系统的名称和位置密码文件在 CS.cfg 文件中配置：

-

passwordFile=/var/lib/pki/*instance_name***/conf/password.conf**

内部密码存储和复制数据库具有随机生成的 PIN，这些 PIN 在安装和配置子系统时设置；在配置实例时管理员定义了内部 LDAP 数据库密码。

password.conf 文件中的密码条目采用以下格式：

name=*password*

例如：

internal=413691159497

如果需要 HSM 令牌，请使用以下格式：

hardware-*name*=*password*

例如：

hardware-NHSM-CONN-XC=MyHSM$S8cret

password.conf 文件的内容示例：

```
internal=376577078151
internaldb=secret12
replicationdb=1535106826
hardware-NHSM-CONN-XC=MyHSM$S8cret
```

14.3.2. 使用证书系统 Watchdog 服务

在证书系统中，watchdog 服务用于启动需要密码访问安全数据库的服务，以便启动。如果需要不要在系统中存储未加密的密码，则 watchdog 服务：

- 在服务器启动期间提示输入相关密码并缓存它们。
- 当服务器因为崩溃而自动重启时，请使用缓存的密码。

14.3.2.1. 启用 Watchdog 服务

启用 watchdog 服务：

1. 如果要在此主机上使用 Shared Secret 功能，请启用 Shared Secret 功能，如 第 14.8.3 节 “启用 CMC 共享 Secret 功能” 所述。

2. 从 /var/lib/pki/instance_name/conf/ 目录中备份 server.xml 和 password.conf 文件。例如：

```
# cp -p /var/lib/pki/instance_name/conf/server.xml /root/
# cp -p /var/lib/pki/instance_name/conf/password.conf /root/
```

3. 停止并禁用证书系统实例的服务：

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

4. 如果您使用硬件安全模块(HSM)，请启用 watchdog 服务来提示输入硬件令牌的密码：

   a. 显示硬件令牌的名称：

   ```
   # egrep "^hardware-" /var/lib/pki/instance_name/conf/password.conf
   hardware-HSM_token_name=password
   ```

   上例中突出显示的字符串是硬件令牌名称。

   b. 将 cms.tokenList 参数添加到 /var/lib/pki/instance_name/conf/ca/CS.cfg 文件中，并将其设置为硬件令牌的名称。例如：

   ```
   cms.tokenList=HMS_token_name
   ```

5. 为实例启用 watchdog 配置：

```
# pki-server instance-nuxwdog-enable instance_name
```

或者，为所有实例启用 watchdog：

```
# pki-server nuxwdog-enable
```

详情请查看 pki-server-nuxwdog(8) man page。

6. 默认情况下，nuxwdog 将服务器作为 /etc/sysconfig/pki-tomcat 文件中的 TOMCAT_USER 变量中配置的用户启动。另外，要修改用户和组：

   a. 将实例的 watchdog systemd 单元文件复制到 /etc/systemd/system/ 目录中：

   ```
   # cp -p /usr/lib/systemd/system/instance_name-nuxwdog@.service
   /etc/systemd/system/
   ```

   注意

   /etc/systemd/system/ 目录中的单元文件具有更高的优先级，且不会在更新过程中替换。

   b. 在 /etc/pki/instance_name/nuxwdog.conf 文件中的 [Service] 部分添加以下条目：

   ```
   User new_user_name
   ```

   c. 重新载入 systemd 配置：

   ```
   # systemctl daemon-reload
   ```

7. 启用使用 watchdog 的证书系统服务：

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

8. （可选）：请参阅 第 14.3.2.3 节 “验证证书系统 Watchdog 服务是否已启用”。

9. 要启动证书系统实例，请运行以下命令并输入提示的密码：

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

14.3.2.2. 使用启用 Watchdog 启动和停止证书系统

有关如何管理证书系统实例的详情，请参考 第 2.2.3 节 “执行管理(systemctl)”。

14.3.2.3. 验证证书系统 Watchdog 服务是否已启用

验证 watchdog 服务是否已启用：

1. 验证 pki-tomcatd-nuxwdog 服务是否已启用：

```
# systemctl is-enabled pki-tomcatd-nuxwdog@instance_name.service
enabled
```

2. 验证 pki-tomcatd 服务是否已禁用：

```
# systemctl is-disabled pki-tomcatd@instance_name.service
disabled
```

3. 在 /etc/pki/instance_name/server.xml 文件中：

   a. 验证 passwordFile 参数是否引用 CS.cfg 文件。例如：

```
passwordFile="/var/lib/pki/instance_name/ca/CS.cfg"
```

   b. 验证 passwordClass 参数是否已设置为 com.netscape.cms.tomcat.NuxwdogPasswordStore ：

```
passwordClass="com.netscape.cms.tomcat.NuxwdogPasswordStore"
```

14.3.2.4. 禁用 Watchdog 服务

禁用 watchdog 服务：

1. 停止并禁用使用 watchdog 的证书系统实例的服务：

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

2. 在不监视实例的情况下启用常规服务：

```
# pki-server instance-nuxwdog-disable instance_name
```

3. 禁用实例的 watchdog 配置：

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

详情请查看 pki-server-nuxwdog(8) man page。

4. 将 password.conf 文件恢复到其原始位置。例如：

```
# cp /root/password.conf.bak /var/lib/pki/instance_name/conf/password.conf
```

5. 启动证书系统实例：

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

14.4. TOMCAT ENGINE 和 WEB 服务的配置文件

子系统的所有用户和管理（管理员、代理和审核员）服务都可通过 Web 协议访问。

本节讨论适用于所有 Red Hat Certificate System 子系统的两大配置文件(CA、KRA、OCSP、TKS 和 TPS)：

- /var/lib/pki/instance_name/conf/server.xml 提供 Tomcat 引擎的配置。

- */usr/share/pki/subsystem_type/webapps/WEB-INF/web.xml 提供此实例提供的 Web 服务的配置。*

14.4.1. tomcatjss

注意

后续小节包括有关对参数值所需更改的重要配置信息。确保它们遵循严格的合规性。

在示例 pki-tomcat/conf 目录中找到 的 server.xml 文件中的以下配置可用于解释 Tomcatjs 如何适合于整个证书系统生态系统。secret 端口的 Connector 条目的一部分如下所示。

```
<Connector name="Secure"

# Info about the socket itself
port="8443"
protocol="org.apache.coyote.http11.Http11Protocol"
SSLEnabled="true"
sslProtocol="SSL"
scheme="https"
secure="true"
connectionTimeout="80000"
maxHttpHeaderSize="8192"
acceptCount="100" maxThreads="150" minSpareThreads="25"
enableLookups="false" disableUploadTimeout="true"
# Points to our tomcat jss implementation
sslImplementationName="org.apache.tomcat.util.net.jss.JSSImplementation"

# OCSP responder configuration can be enabled here
enableOCSP="true"
ocspCacheSize="1000"
ocspMinCacheEntryDuration="60"
ocspMaxCacheEntryDuration="120"
ocspTimeout="10"

# A collection of cipher related settings that make sure connections are secure.
strictCiphers="true"
# The "clientAuth" parameter configures the client authentication scheme
# for this server socket. If you set "clientAuth" to "want", the client
# authentication certificate is optional. Alternatively, set the
# parameter to "required" to configure that the certificate is is mandatory.
clientAuth="want"
sslVersionRangeStream="tls1_1:tls1_2"
sslVersionRangeDatagram="tls1_1:tls1_2"
sslRangeCiphers="+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_RSA_WITH_AES
_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA,+TLS_DHE_RSA_WITH_AES_256_
CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_DHE_RSA_WITH_AES_256_CB
C_SHA256,+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_ECDHE_RSA_WITH_AES_2
```

```
56_CBC_SHA384,+TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,+TLS_ECDHE_RSA_WITH_AE
S_128_GCM_SHA256,+TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,+TLS_ECDHE_RSA_WITH
_AES_256_GCM_SHA384
serverCertNickFile="/var/lib/pki/pki-tomcat/conf/serverCertNick.conf"
passwordFile="/var/lib/pki/pki-tomcat/conf/password.conf"
passwordClass="org.apache.tomcat.util.net.jss.PlainPasswordFile"
certdbDir="/var/lib/pki/pki-tomcat/alias"

/>
```

在 Tomcat 引擎的 server.xml 配置文件中，有这个 Connector 配置元素包含指向 tomcatjss 实现的指针，它可以插入到这个 Connector 对象的 sslImplementation 属性中。

以下子小节中解释了每个关键参数元素。

14.4.1.1. TLS Cipher 配置

当 Red Hat Certificate 系统充当客户端和服务器时，server.xml 文件中配置的 TLS 密码会提供系统范围的默认值。这包括充当服务器（例如，从 Tomcat 提供 HTTPS 连接时）和作为客户端（例如，与 LDAP 服务器通信或与其他证书系统实例通信时）。

服务器 TLS 密码的配置位于特定于 Red Hat Certificate System 实例的 /var/lib/pki/instance_name/conf/server.xml 文件中。以下参数控制提供的密码：

- strictCiphers，当设为 true 时，确保只启用 sslRangeCiphers 中的 + 符号的密码。

  ```
  strictCiphers="true"
  ```

  不要更改默认值(true)。

- sslVersionRangeStream 和 sslVersionRangeDatagram 设置服务器支持的 TLS 版本。以下是参数的默认值：

  ```
  sslVersionRangeStream="tls1_1:tls1_2"
  sslVersionRangeDatagram="tls1_1:tls1_2"
  ```

  不要更改参数的默认值。

-

sslRangeCiphers 设置启用和禁用哪些密码。启用了 + 符号的密码，并禁用了 - 符号的密码。

设置 RSA 密码，如下所示：

```
sslRangeCiphers="+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA,+TLS_DHE_RSA_WITH_AES_256_CBC_SHA,+TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,+TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,+TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,+TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,+TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,+TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,+TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

设置 EC 密码，如下所示：

```
sslRangeCiphers="+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,+TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,+TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,+TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,+TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,+TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

有关允许的密码列表，请参阅 第 3.1 节 “TLS、ECC 和 RSA”。

- *如果您在为 RSA 启用 FIPS 模式的系统中安装 LunaSA 或 nCipher Hardware Security Module (HSM)的证书系统，请禁用以下密码，因为 FIPS 模式的 HSMs 不支持它们：*
  - *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA*
  - *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA*
  - *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256*
  - *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*

14.4.1.1.1. 客户端 TLS 密码配置

当作为客户端到另一个 CS 系统时，Red Hat Certificate System 还允许系统上的密码配置。

列表中的密码用逗号分开。

在 CA 上（用于 CA 与 KRA 通信）：

```
ca.connector.KRA.clientCiphers=your selected cipher list
```

例如：

```
ca.connector.KRA.clientCiphers=TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_
CBC_SHA
```

在 TPS 上（用于与 CA 通信、KRA 和 TKS）：

```
tps.connector.ca id.clientCiphers=your selected cipher list
tps.connector.kra id.clientCiphers=your selected cipher list
tps.connector.tks id.clientCiphers=your selected cipher list
```

例如：

```
tps.connector.ca1.clientCiphers=TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_
CBC_SHA
```

14.4.1.2. 在 CA 上启用自动撤销检查

CA 可以配置为检查服务器在 SSL/TLS 客户端身份验证过程中收到的任何证书的撤销状态（包括代理、管理员和注册）。这意味着，当 CA 收到任何客户端身份验证请求时，它会自动检查 OCSP。（有关设置 OCSP 响应器的信息，请参阅 Red Hat Certificate System Administration Guide 中的 Using the Online Certificate Status Protocol (OCSP) Responder。）

作为撤销检查的一部分，CA 能够缓存客户端身份验证，以便它保留验证的证书列表。这允许 CA 在检查其内部数据库或 OCSP 前检查其缓存的结果，从而提高了整体操作性能。在 revocationChecking.enabled 参数中启用自动撤销检查。

撤销状态结果只在指定的时间内有效(revocationChecking.validityInterval)。如果 CA 无法重新验证缓存中的证书状态，则有一个宽限期(revocationChecking.unknownStateInterval)，其中之前缓存的

状态仍被视为有效，即使它超出有效期间隔。

注意

缓存的证书保存在缓冲区中(revocationChecking.bufferSize)。如果缓冲区设置缺失或设置为零，则不会保留缓冲区，这意味着不会缓存撤销检查的结果。在这种情况下，所有撤销检查都会直接针对 CA 的内部数据库执行。

注意

子系统 CS.cfg 配置文件包含一个参数 jss.ocspcheck.enable，它设置证书管理器是否应该使用 OCSP 验证它收到的证书的撤销状态。将此参数的值改为 true 意味着证书管理器读取证书中的授权信息访问扩展，并从扩展中指定的 OCSP 响应程序验证证书的撤销状态。

1. 停止子系统实例。

```
# pki-server stop instance_name
```

2. 打开 CS.cfg 文件。

```
# vim /var/lib/pki/instance-name/ca/conf/CS.cfg
```

3. 编辑与撤销相关的参数。

```
auths.revocationChecking.bufferSize=50
auths.revocationChecking.ca=ca
auths.revocationChecking.enabled=true
auths.revocationChecking.unknownStateInterval=0
auths.revocationChecking.validityInterval=120
```

- revocationChecking.ca.设置提供 OCSP respsonse、CA 或 OCSP 响应程序的服务。
- revocationChecking.enabled.设置撤销检查。true 启用检查； false 禁用检查。默认情况下启用该功能。
-

revocationChecking.bufferSize.设置服务器应在其缓存中维护的最后一次检查证书总数。例如，如果缓冲区大小为 2，服务器会保留在其缓存中检查的最后两个证书。默认情况下，服务器缓存最后 50 个证书。

- *revocationChecking.unknownStateInterval.设置服务器检查撤销状态的频率。默认间隔为 0 秒。unknownStateInterval 是一个宽限期，如果 CA 没有方法（没有访问信息）验证证书状态，则缓存结果将被假定为 true*

- *revocationChecking.validityInterval.设置缓存的证书被视为有效的时长。选择间隔时请小心。例如，如果有效期周期为 60 秒，服务器会每分钟丢弃其缓存中的证书，并尝试从其源中检索证书。证书管理器使用其内部数据库来检索和验证证书的撤销状态。默认有效期为 120 秒(2 分钟)。*

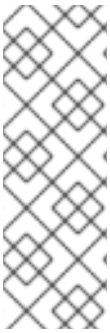
4. *启动证书系统实例。*

```
# pki-server start instance_name
```

14.4.1.3. 为子系统启用证书撤销检查

默认情况下，证书系统子系统没有启用 OCSP 检查来验证子系统证书。这意味着，有人可以登录具有撤销证书的管理或代理接口。

通过编辑 server.xml 文件，为所有子系统启用 OCSP 检查。代理接口和 admin 接口是单独配置的，因此应编辑配置中的两个部分。



注意

如果子系统已配置为使用 SSL/TLS 与其内部数据库的连接，则 LDAP 内部数据库的 SSL/TLS 服务器证书必须由 OCSP 响应器识别。如果 OCSP 响应器无法识别 LDAP 服务器证书，则子系统将无法正确启动。由于子系统-LDAP SSL/TLS 服务器连接被配置为子系统设置的一部分，因此 Red Hat Certificate System 10 规划、安装和部署指南 中涵盖此配置。

1. *获取用于检查证书状态的 OCSP 或 CA 的 OCSP 签名证书名称。例如：*

```
# certutil -L -d /etc/pki/instance-name/alias
Certificate Nickname                                         Trust Attributes
```

```
        SSL,S/MIME,JAR/XPI
Certificate Authority - Example Domain                    CT,c,
ocspSigningCert cert-pki-ocsp CTu,Cu,Cu
subsystemCert cert-pki-ocsp                               u,u,u
Server-Cert cert-pki-ocsp                                 u,u,u
auditSigningCert cert-pki-ocsp                            u,u,Pu
```

2. 为子系统打开 **server.xml** 文件。例如：

```
# vim /etc/pki/instance-name/server.xml
```

3. 如果实例的安全数据库中没有 OCSP 签名证书，请导入它：

```
# certutil -d /etc/pki/instance-name/alias -A -n "ocspSigningCert cert-pki-ca" -t "C,," -a -i
ocspCert.b64
```

4. 启用 OCSP 检查有三个关键参数：

   - 启用**OCSP**，它必须设置为 **true** 才能启用 OCSP 检查。

     这是一个全局设置；如果为一个接口设置，则它适用于实例的每个接口。但是，它必须在 **server.xml** 文件中列出的第一个接口上设置，通常是代理接口。其它接口上的任何设置都会被忽略。

   - **ocspResponderURL**，它为发送 OCSP 请求提供 OCSP 响应器的 URL。

     对于 **OCSP Manager**，这可能是其他 OCSP 或 CA 中的另一个 OCSP 服务。对于其他子系统，这总是指向 OCSP 或 CA 中的外部 OCSP 服务。

   - **ocspResponderCertNickname** 给出用于签署响应的签名证书；对于 CA OCSP 服务，这是 CA 的 OCSP 签名证书，对于 OCSP 响应者，它是一个 OCSP 签名证书。

   其他参数可用于定义 OCSP 通信。所有 OCSP 检查参数都列在 表 14.10 “server.xml 的 OCSP 参数” 中。

   文件中有两个不同的部分用于代理和管理界面。需要向两个部分添加 OCSP 参数，以启用

和配置 OCSP 检查。例如：

例 14.3. 代理接口的 OCSP 设置

```
<Connector name="Agent" port="8443" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="true" sslProtocol="SSL"
    sslOptions="ssl2=true,ssl3=true,tls=true"

 ssl3Ciphers="-SSL3_FORTEZZA_DMS_WITH_NULL_SHA, ..."

 tls3Ciphers="-SSL3_FORTEZZA_DMS_WITH_NULL_SHA, ..."
    SSLImplementation="org.apache.tomcat.util.net.jss.JSSImplementation"
    enableOCSP="true"
    ocspResponderURL="http://server.example.com:8443/ca/ocsp"
    ocspResponderCertNickname="ocspSigningCert cert-pki-ca 102409a"
    ocspCacheSize="1000"
    ocspMinCacheEntryDuration="60"
    ocspMaxCacheEntryDuration="120"
    ocspTimeout="10"
    debug="true"
    serverCertNickFile="/etc/pki/instance-name/serverCertNick.conf"
    passwordFile="/etc/pki/instance-name/password.conf"
    passwordClass="org.apache.tomcat.util.net.jss.PlainPasswordFile"
    certdbDir="/etc/pki/instance-name/alias"/>
```

5. ***如果给定的 OCSP 服务不是 CA，则必须将 OCSP 服务的签名证书导入到子系统的 NSS 数据库中。这可以在控制台或使用 certutil 完成； Red Hat Certificate System Administration Guide 中的 Installing Certificates in the Certificate System Database in the Certificate System Database 中涵盖了这两个选项。***

6. ***重新启动子系统。***

```
# pki-server restart instance_name
```

表 14.10. server.xml 的 OCSP 参数

参数	描述
enableOCSP	为子系统启用（或禁用） OCSP 检查。
ocspResponderURL	设置发送 OCSP 请求的 URL。 对于 OCSP Manager，这可能是其他 OCSP 或 CA 中的另一个 OCSP 服务。对于 TKS 或 KRA，这总是指向 OCSP 或 CA 中的外部 OCSP 服务。

参数	描述
ocspResponderCertNickname	为响应器设置签名证书的 nickname，可以是 OCSP 签名证书或 CA 的 OCSP 签名证书。证书必须导入到子系统的 NSS 数据库中，并设置了适当的信任设置。
ocspCacheSize	设置缓存条目的最大数量。
ocspMinCacheEntryDuration	在进行另一个提取前设置最小秒数。例如，如果将其设置为 120，则无法再次检查证书的有效性，直到最后一次有效期检查前至少 2 分钟为止。
ocspMaxCacheEntryDuration	设置在下一次获取尝试前要等待的最大秒数。这可防止在有效检查之间有太大的窗口。
ocspTimeout	为 OCSP 请求设置超时时间（以秒为单位）。

14.4.1.4. 在注册配置文件中添加 AIA 扩展

要在使用外部 OCSP 时在配置集中设置 AIA URL，请在证书配置文件中添加正确的 URL。例如：

```
policyset.cmcUserCertSet.5.default.params.authInfoAccessADLocation_0=http://example.com:8080/ocsp/ee/ocsp
```

14.4.2. 会话超时

当用户通过客户端应用连接到 PKI 服务器时，服务器将创建一个会话来跟踪用户。只要用户保持活动状态，用户可以对同一会话执行多个操作，而无需重新验证。

会话超时决定了服务器在因为不活跃而终止会话前等待多久。会话终止后，用户需要重新验证来继续访问服务器，服务器将创建新的会话。

有两种超时类型：

- *TLS 会话超时*
- *HTTP 会话超时*

由于客户端的工作方式的区别，客户端会因这些超时的不同而受到不同。

注意

某些客户端有自己的超时配置。例如，Firefox 有一个 keep-alive 超时设置。详情请查看 http://kb.mozillazine.org/Network.http.keep-alive.timeout。如果值与 TLS Session Timeout 或 HTTP Session Timeout 的服务器设置不同，则可以观察不同的行为。

14.4.2.1. TLS 会话超时

TLS 会话是通过 TLS 握手协议建立的 TLS 连接的安全通信频道。

PKI 服务器为 TLS 会话活动生成审计事件。在创建连接时，服务器会生成一个 ACCESS_SESSION_ESTABLISH 审计事件。如果连接创建失败，服务器将使用 Outcome=Failure 生成 ACCESS_SESSION_ESTABLISH 审计事件。连接关闭时，服务器将生成一个 ACCESS_SESSION_TERMINATED 审计事件。

TLS 会话超时（即 TLS 连接超时）在 /etc/pki /& lt;instance>/server.xml 文件中的 Secure <Connector> 元素中配置：

```
...
<Server>
 <Service>
  <Connector name="Secure"
   ...
   keepAliveTimeout="300000"
   ...
   />
 </Service>
</Server>
...
```

默认情况下，超时值设置为 300000 毫秒（即 5 分钟）。要更改这个值，请编辑 /etc/pki/<instance>/server.xml 文件，然后重新启动服务器。

注意

请注意，这个值将影响到服务器的所有 TLS 连接。较大的值可以提高客户端的效率，因为它们可以重复使用尚未过期的现有连接。但是，它也可能会增加服务器必须同时支持的连接数量，因为它需要更长的时间才能过期连接。

14.4.2.2. HTTP 会话超时

HTTP 会话是利用 HTTP cookie 在多个 HTTP 请求之间跟踪用户的机制。PKI 服务器不会为 HTTP 会话生成审计事件。

注意

为了实现审核一致性，请将本节中的 &lt ;session-timeout > 值设置为与 第 14.4.2.1 节 “TLS 会话超时” 中的 keepAliveTimeout 值匹配。例如，如果将 keepAliveTimeout 设为 300000 (5 分钟)，则将 < session-timeout> 设置为 30。

HTTP 会话超时可以在 /etc/pki/& lt;instance>/web.xml 文件中的 <session-timeout > 元素中配置：

```
...
<web-app>
 <session-config>
  <session-timeout>30</session-timeout>
 </session-config>
</web-app>
...
```

默认情况下，超时值设置为 30 分钟。要更改值，请编辑 /etc/pki/<instance>/web.xml 文件，然后重新启动服务器。

注意

请注意，这个值会影响服务器上所有 Web 应用程序中的所有会话。较大的值可以提高用户的体验，因为不需要重新验证或再次查看访问横幅。但是，它也可能会增加安全风险，因为带外的 HTTP 会话需要更长的时间才能过期。

14.4.2.3. PKI Web UI 的会话超时

PKI Web UI 是在浏览器中运行的基于 Web 的交互式客户端。目前，它只支持客户端证书身份验证。

打开 Web UI 时，浏览器可能会创建多个 TLS 连接到服务器。这些连接与单个 HTTP 会话关联。

要为 Web UI 配置超时，请参阅 第 14.4.2.2 节 “HTTP 会话超时”。TLS 会话超时通常无关，因为浏览器会缓存客户端证书，以便它自动重新创建 TLS 会话。

当 HTTP 会话过期时，Web UI 不提供任何立即指示。但是，Web UI 将在用户执行操作前显示访问横幅（如果已启用）。

14.4.2.4. PKI 控制台的会话超时

PKI 控制台是一个交互式的单机图形 UI 客户端。它支持用户名/密码和客户端证书身份验证。

当控制台启动时，它将创建一个到服务器的 TLS 连接。控制台将在打开图形界面前显示访问横幅（如果已启用）。与 Web UI 不同，控制台不会维护与服务器的 HTTP 会话。

要为控制台配置超时，请参阅 第 14.4.2.1 节 “TLS 会话超时”。HTTP 会话超时无关，因为控制台不使用 HTTP 会话。

当 TLS 会话过期时，TLS 连接将关闭，控制台将立即退出系统。如果用户希望继续，用户需要重启控制台。

14.4.2.5. PKI CLI 的会话超时

PKI CLI 是执行一系列操作的命令行客户端。它支持用户名/密码和客户端证书身份验证。

当 CLI 启动时，它将创建一个到服务器和 HTTP 会话的 TLS 连接。CLI 将在执行操作前显示访问横幅（如果已启用）。

两个超时通常都与 PKI CLI 无关，因为操作按顺序执行，且 CLI 在完成后会立即退出。但是，如果 CLI 等待用户输入，速度缓慢或者变得无响应，则 TLS 会话或 HTTP 会话可能会过期，其余操作会失败。如果预期有这种延迟，请参阅 第 14.4.2.1 节 “TLS 会话超时” 和 第 14.4.2.2 节 “HTTP 会话超时” 以适应预期的延迟。

14.5. WEB.XML

14.5.1. 从 web.xml 中删除非使用接口（仅限 CA）

一些传统接口（用于批量颁发或策略框架）仍然包含在 CA 的 web.xml 文件中。但是，由于这些功能已弃用且不再使用，因此可以从 CA 配置中删除它们以提高安全性。

1. **停止 CA。**

```
# pki-server stop instance_name
```

或（如果使用 nuxwdog watchdog）

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

2. **打开 CA 的 Web 文件目录。例如：**

```
# cd /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF
```

3. **备份当前的 web.xml 文件。**

```
# cp web.xml web.xml.servlets
```

4. **编辑 web.xml 文件，并删除以下每个已弃用的 servlet 的 <servlet> 条目：**

- **caadminEnroll**
- **cabulkissuance**
- **cacertbasedenrollment**
- **caenrollment**
- **caProxyBulkIssuance**

例如，删除 caadminEnroll servlet 条目：

```
<servlet>
    <servlet-name>  caadminEnroll  </servlet-name>
```

```
    <servlet-class> com.netscape.cms.servlet.cert.EnrollServlet
</servlet-class>
        <init-param><param-name>  GetClientCert  </param-name>
            <param-value> false      </param-value> </init-param>
        <init-param><param-name>  successTemplate  </param-name>
            <param-value> /admin/ca/EnrollSuccess.template
</param-value> </init-param>
        <init-param><param-name>  AuthzMgr    </param-name>
            <param-value> BasicAclAuthz </param-value>
</init-param>
        <init-param><param-name>  authority   </param-name>
            <param-value> ca         </param-value> </init-param>
        <init-param><param-name>  interface   </param-name>
            <param-value> admin          </param-value>
</init-param>
        <init-param><param-name>  ID          </param-name>
            <param-value> caadminEnroll </param-value>
</init-param>
        <init-param><param-name>  resourceID  </param-name>
            <param-value> certServer.admin.request.enrollment
</param-value> </init-param>
        <init-param><param-name>  AuthMgr     </param-name>
            <param-value> passwdUserDBAuthMgr </param-value>
</init-param>
  </servlet>
```

5. 删除 servlet 条目后，删除对应的 < servlet-mapping> 条目。

```
<servlet-mapping>
    <servlet-name>  caadminEnroll  </servlet-name>
    <url-pattern>   /admin/ca/adminEnroll  </url-pattern>
</servlet-mapping>
```

6. 为最终用户请求接口删除三个 <filter-mapping > 条目。

```
  <filter-mapping>
    <filter-name>  EERequestFilter          </filter-name>
    <url-pattern>  /certbasedenrollment       </url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>  EERequestFilter          </filter-name>
    <url-pattern>  /enrollment              </url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>  EERequestFilter          </filter-name>
    <url-pattern>  /profileSubmit           </url-pattern>
  </filter-mapping>
```

–

7. 再次启动 CA。

```
# pki-server start instance_name
```

或（如果使用 nuxwdog watchdog）

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

14.6. 自定义 WEB 服务

所有子系统(TKS 除外)都有一些基于 Web 的服务页面，用于代理，另一些用于其他角色，如管理员或结束实体。这些基于 Web 的服务页面使用基本的 HTML 和 JavaScript，它们可以自定义为使用不同的颜色、徽标和其他设计元素适合现有站点或内部网。

14.6.1. 自定义子系统 Web 应用程序

每个 PKI 子系统都有对应的 Web 应用，其中包含：

- 包含文本、JavaScript 代码、页面布局、CSS 格式等的 HTML 页面
- web.xml 文件，用于定义 servlet、路径、安全限制和其他
- 到 PKI 库的链接。

子系统 Web 应用程序使用位于 /var/lib/pki/pki-tomcat/conf/Catalina/localhost/ direcotry 中的上下文文件进行部署，例如 ca.xml 文件中：

```
<Context docBase="/usr/share/pki/ca/webapps/ca" crossContext="true" allowLinking="true">
   ...
</Context>
```

docBase 指向默认 Web 应用目录 /usr/share/pki/ 的位置。

要自定义 Web 应用程序，将 Web 应用程序目录复制到实例的 webapps 目录中：

```
$ cp -r /usr/share/pki/ca/webapps/ca /var/lib/pki/pki-tomcat/webapps
```

然后，将 docBase 更改为指向 webapps 目录的自定义 Web 应用程序目录：

```
<Context docBase="ca" crossContext="true" allowLinking="true">
    ...
</Context>
```

这些更改将立即生效，而无需重新启动服务器。

要删除自定义 Web 应用程序，只需恢复 docBase 并删除自定义 Web 应用程序目录：

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/ca
```

14.6.2. 自定义 Web UI 主题

同一实例的子系统 Web 应用程序共享相同的主题，其中包含：

- CSS 文件，该文件决定了全局外观
- 镜像文件，包括徽标、图标和其他
- 品牌属性，它决定了页面标题、徽标链接、标题颜色和其他。

Web UI 主题使用 /var/lib/pki/pki-tomcat/conf/Catalina/localhost/ 目录中的 pki.xml 上下文文件进行部署：

```
<Context docBase="/usr/share/pki/common-ui" crossContext="true" allowLinking="true">
    ...
</Context>
```

docBase 指向默认主题目录 /usr/share/pki/ 的位置。

要自定义主题，请将默认主题目录复制到实例的 webapps 目录中的 pki 目录中：

```
$ cp -r /usr/share/pki/common-ui /var/lib/pki/pki-tomcat/webapps/pki
```

然后，将 docBase 更改为指向与 webapps 目录相关的自定义主题目录：

```
<Context docBase="pki" crossContext="true" allowLinking="true">
    ...
</Context>
```

这些更改将立即生效，而无需重新启动服务器。

要删除自定义主题，只需恢复 docBase 并删除自定义主题目录：

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/pki
```

14.6.3. 自定义 TPS 令牌状态标签

默认令牌状态标签存储在 /usr/share/pki/tps/conf/token-states.properties 文件中，并在第 2.5.2.4.1.4 节 “令牌状态和转换标签” 中描述。

要自定义标签，将文件复制到实例目录中：

```
$ cp /usr/share/pki/tps/conf/token-states.properties /var/lib/pki/pki-tomcat/tps/conf
```

这些更改将立即生效，而无需重新启动服务器。

要删除自定义标签，只需删除自定义文件：

```
$ rm /var/lib/pki/pki-tomcat/tps/conf/token-states.properties
```

14.7. 使用访问横幅

在证书系统中，管理员可以使用自定义文本配置横幅。在以下情况下会显示横幅：

Application	显示横幅时
PKI 控制台	• 在显示控制台之前。 • 会话已过期后。[a]
Web 界面	• 连接到 Web 界面时。 • 会话过期后。[a]
pki 命令行工具	• 在进行实际操作之前。
[a] 有关更改会话超时的详情，请参考 第 14.4.2 节 “会话超时”。	

您可以使用横幅向用户显示重要信息，然后才能使用证书系统。用户必须同意显示的文本才能继续。

> ***例 14.4. 显示 Access Banner 时***
>
> ***以下示例显示，如果您使用 pki 工具，则显示了访问横幅：***
>
> ```
> $ pki cert-show 0x1
> WARNING! Access to this service is restricted to those individuals with specific permissions. If you
> are not an authorized user, disconnect now. Any attempts to gain unauthorized access will be
> prosecuted to the fullest extent of the law. Do you want to proceed (y/N)? y
> -----------------
> Certificate "0x1"
> -----------------
>   Serial Number: 0x1
>   Issuer: CN=CA Signing Certificate,OU=instance_name,O=EXAMPLE
>   Subject: CN=CA Signing Certificate,OU=instance_name,O=EXAMPLE
>   Status: VALID
>   Not Before: Mon Feb 20 18:21:03 CET 2017
>   Not After: Fri Feb 20 18:21:03 CET 2037
> ```

14.7.1. 启用访问横幅

要启用访问横幅，请创建 /etc/pki/instance_name/banner.txt 文件，并输入要显示的文本。

重要

/etc/pki/instance_name/banner.txt 文件中的文本必须使用 UTF-8 格式。要验证，请参阅 第 14.7.4 节 “验证 Banner”。

14.7.2. 禁用访问横幅

要禁用访问横幅，可删除或重命名 /etc/pki/instance_name/banner.txt 文件。例如：

```
# mv /etc/pki/instance_name/banner.txt /etc/pki/instance_name/banner.txt.UNUSED
```

14.7.3. 显示横幅

显示当前配置的横幅：

```
# pki-server banner-show -i instance_name
```

14.7.4. 验证 Banner

验证横幅不包含无效字符：

```
# pki-server banner-validate -i instance_name
---------------
Banner is valid
---------------
```

14.8. CMC 的配置

这部分论述了如何通过 CMS (CMC)为证书管理配置证书系统。

14.8.1. 了解 CMC 的工作方式

在配置 CMC 前，请阅读以下文档以了解更多有关主题的信息：

- 第 2.4.1.1.2.2 节 “使用 CMC 注册”

- Red Hat Certificate System Administration Guide中的使用 CMC 发布证书。

- 在 Red Hat Certificate System Administration Guide 中，生成颁发证书（证书配置文件）的规则。

14.8.2. 启用 PopLinkWittnessV2 功能

对于证书颁发机构(CA)上的高级别安全性，请在 /var/lib/pki/instance_name/ca/conf/CS.cfg 文件中启用以下选项：

```
cmc.popLinkWitnessRequired=true
```

14.8.3. 启用 CMC 共享 Secret 功能

在证书颁发机构(CA)中启用共享令牌功能：

1. 如果主机上启用了 watchdog 服务，请临时禁用该服务。请参阅 第 14.3.2.4 节 “禁用 Watchdog 服务”。

2. 将 shrTok 属性添加到目录服务器的 schema 中：

```
# ldapmodify -D "cn=Directory Manager" -H ldaps://server.example.com:636 -W -x

dn: cn=schema
changetype: modify
add: attributetypes
attributetypes: ( 2.16.840.1.117370.3.1.123 NAME 'shrTok' DESC 'User
 Defined ObjectClass for SharedToken' SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
 SINGLE-VALUE X-ORIGIN 'custom for sharedToken')
```

3. 如果系统密钥存储在硬件安全模块(HSM)上，请在 /var/lib/pki/instance_name/ca/conf/CS.cfg 文件中设置 cmc.token 参数。例如：

```
cmc.token=NHSM-CONN-XC
```

4. 使用以下方法之一启用共享令牌身份验证插件：

- 

  注意

  pkiconsole 已被弃用。

  使用 pkiconsole 工具启用插件：

  1. **使用 pkiconsole 工具登录到系统。例如：**

     ```
     # pkiconsole https:host.example.com:8443/ca
     ```

  2. **在 Configuration 选项卡中，选择 Authentication。**

  3. **点 Add 并选择 SharedToken。**

  4. **点 Next。**

  5. **输入以下信息：**

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

  6. **点确定。**

- **要手动启用插件，请在 /var/lib/pki/instance_name/ca/conf/CS.cfg 文件中添加以下设置：**

  ```
  auths.impl.SharedToken.class=com.netscape.cms.authentication.SharedSecret
  auths.instance.SharedToken.dnpattern=
  auths.instance.SharedToken.ldap.basedn=ou=People,dc=example,dc=org
  auths.instance.SharedToken.ldap.ldapauth.authtype=BasicAuth
  ```

```
auths.instance.SharedToken.ldap.ldapauth.bindDN=cn=Directory Manager
auths.instance.SharedToken.ldap.ldapauth.bindPWPrompt=Rule SharedToken
auths.instance.SharedToken.ldap.ldapauth.clientCertNickname=
auths.instance.SharedToken.ldap.ldapconn.host=server.example.com
auths.instance.SharedToken.ldap.ldapconn.port=636
auths.instance.SharedToken.ldap.ldapconn.secureConn=true
auths.instance.SharedToken.ldap.ldapconn.version=3
auths.instance.SharedToken.ldap.maxConns=
auths.instance.SharedToken.ldap.minConns=
auths.instance.SharedToken.ldapByteAttributes=
auths.instance.SharedToken.ldapStringAttributes=
auths.instance.SharedToken.pluginName=SharedToken
auths.instance.SharedToken.shrTokAttr=shrTok
```

5. **在 /var/lib/pki/instance_name/ca/conf/CS.cfg 文件中设置 ca.cert.issuance_protection.nickname 参数中的 RSA issuance 保护证书的 nickname。例如：**

```
ca.cert.issuance_protection.nickname=issuance_protection_certificate
```

此步骤是：

- **如果您在 ca.cert.subsystem.nickname 参数中使用 RSA 证书，可选。**
- **如果您在 ca.cert.subsystem.nickname 参数中使用 ECC 证书，则需要此项。**

重要

如果没有设置 ca.cert.issuance_protection.nickname 参数，证书系统将自动使用 ca.cert.subsystem.nickname 中指定的子系统的证书。但是，颁发保护证书必须是 RSA 证书。

6. **重启证书系统：**

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

当 CA 启动时，证书系统会提示输入 Shared Token 插件使用的 LDAP 密码。

7. **如果在此流程开始时临时禁用了 watchdog 服务，请重新启用该服务。请参阅 第 14.3.2.1 节 “启用 Watchdog 服务”。**

14.8.4. 为 Web 用户界面启用 CMCRevoke

如 Red Hat Certificate System Administration Guide 中的 执行 CMC Revocation 部分所述，可以通过两种方式提交 CMC 吊销请求。

在使用 CMCRevoke 实用程序创建通过 Web UI 提交的撤销请求时，请将以下设置添加到 /var/lib/pki/instance_name/ca/conf/CS.cfg 文件中：

```
cmc.bypassClientAuth=true
```

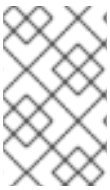
14.9. 使用 CA EE 门户配置 SERVER-SIDE KEY GENERATION FOR CERTIFICATE ENROLLMENT

这部分论述了如何使用 CA EE 门户为证书注册配置 Server-Side Key Generation。

14.9.1. 安装配置

注意

除了设置 Server-Side Keygen 的 CA 外，还需要 KRA 实例。



注意

如果 CA 和 KRA 正在共享 Tomcat 实例，则不需要执行以下步骤来导入传输证书。

安装 CA 和 KRA 实例后，如果独立 Tomcat Web 服务器实例，您需要将 KRA 传输证书添加到 CA 的 nssdb 中。

1. 首先，停止 CA：

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

例如：

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

2. **在 CA 的 CS.cfg 中添加以下行：**

```
ca.connector.KRA.transportCertNickname=KRA transport cert
```

例如：

```
ca.connector.KRA.transportCertNickname=transportCert cert-topology-02-KRA KRA
```

其中 topology-02-KRA 是 KRA 的实例的名称。

3. **查找并导出 KRA 传输证书到文件中：**

```
# grep "kra.transport.cert=" /var/lib/pki/kra_instance_name/kra/conf/CS.cfg | sed
's/kra.transport.cert=//' > kra transport cert file
```

例如：

```
# grep "kra.transport.cert=" /var/lib/pki/pki-kra/kra/conf/CS.cfg | sed 's/kra.transport.cert=//' >
/tmp/kraTransport.cert
```

4. **使用 CA 的 CS.cfg 文件中指定的 nickname 将 KRA 传输证书导入到 CA 的 nssdb 中：**

   1. **列出传输证书别名：**

```
grep "ca.connector.KRA.transportCertNickname"
/var/lib/pki/ca_instance_name/ca/conf/CS.cfg
```

例如：

```
# grep "ca.connector.KRA.transportCertNickname" /var/lib/pki/pki-ca/ca/conf/CS.cfg
      ca.connector.KRA.transportCertNickname=KRA transport cert
```

   - **使用上一步中列出的 nickname 导入证书：**

```
certutil -d /var/lib/pki/ca_instance_name/alias -A -t “,,” -n transportNickName -i kra transport cert file
```

例如：

```
# certutil -d /var/lib/pki/pki-ca/alias -A -t “,,” -n "KRA transport cert" -i /tmp/kraTransport.cert
```

5. 启动 **CA：**

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

例如：

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

14.9.2. 配置集配置

默认提供了两个默认配置文件 **caServerKeygen_UserCert** 和 **caServerKeygen_DirUserCert**，以允许在服务器端生成密钥的证书注册。但是，带有正确输入、输出和策略集的任何配置集都可以转换为服务器端 **keygen** 配置集。

Server-Side Keygen 配置文件必须包含以下组件：

输入

```
input.i1.class_id=serverKeygenInputImpl
```

输出

```
output.o1.class_id=pkcs12OutputImpl
```

policyset

密钥类型和密钥大小参数可以被配置为 **exemplified:**

```
policyset.userCertSet.3.constraint.class_id=keyConstraintImpl
policyset.userCertSet.3.constraint.name=Key Constraint
policyset.userCertSet.3.constraint.params.keyType=-
policyset.userCertSet.3.constraint.params.keyParameters=1024,2048,3072,4096,nistp256,nistp384,nistp521
policyset.userCertSet.3.default.class_id=serverKeygenUserKeyDefaultImpl
```

```
policyset.userCertSet.3.default.name=Server-Side Keygen Default
policyset.userCertSet.3.default.params.keyType=RSA
policyset.userCertSet.3.default.params.keySize=2048
policyset.userCertSet.3.default.params.enableArchival=true
```

身份验证

两个默认的服务器端 keygen 注册配置集在身份验证机制中有所不同，其中

- caServerKeygen_UserCert.cfg

  包含 "auth.class_id=" 的空值，即通过此配置集注册请求需要 CA 代理批准。

- caServerKeygen_DirUserCert.cfg

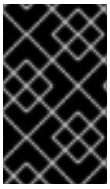
  包含 "auth.instance_id=UserDirEnrollment"，这意味着用户需要传递 LDAP uid/password 身份验证；此类身份验证机制被视为自动证书颁发，因为它不需要从 CA 代理按请求的批准。

  可以通过将 auth.instance_id 指令设置为任何兼容身份验证插件类来配置自动批准，作为上述 caServerKeygen_DirUserCert.cfg 配置集中的 examplified。以下是 CS.cfg 文件中的这样的配置示例：

```
auths.instance.UserDirEnrollment.dnpattern=
auths.instance.UserDirEnrollment.ldap.basedn=ou=People,dc=example,dc=com
auths.instance.UserDirEnrollment.ldap.ldapconn.host=host.example.com
auths.instance.UserDirEnrollment.ldap.ldapconn.port=389
auths.instance.UserDirEnrollment.ldap.ldapconn.secureConn=false
auths.instance.UserDirEnrollment.ldap.maxConns=
auths.instance.UserDirEnrollment.ldap.minConns=
auths.instance.UserDirEnrollment.ldapByteAttributes=
auths.instance.UserDirEnrollment.ldapStringAttributes=mail
auths.instance.UserDirEnrollment.pluginName=UidPwdDirAuth
```

14.10. 配置证书转换

证书系统提供证书转换(CT) V1 支持的基本版本(rfc 6962)。它有从任何可信日志中发布带有嵌入式签名证书时间戳(SCT)的证书的功能，每个部署站点选择具有其根 CA 证书。您还可以将系统配置为支持多个 CT 日志。这个功能至少需要一个可信 CT 日志。



重要

部署站点负责建立与可信 CT 日志服务器的信任关系。

要配置证书转换，请编辑位于 /var/lib/pki/instance name/ca/conf/ CS.cfg 目录中的 CA 的 CS.cfg 文件。

有关如何测试证书转换设置的更多信息，请参阅 Red Hat Certificate System 规划、安装和部署指南 中的测试 证书转换部分。

14.10.1. ca.certTransparency.mode

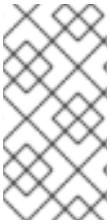
ca.certTransparency.mode 指定三个证书转换模式之一：

- disabled ：签发的证书不会执行 SCT 扩展
- 启用: 签发的证书执行 SCT 扩展
- perProfile: 通过包含以下策略集的配置集注册的证书将执行 SCT 扩展：SignedCertificateTimestampListExtDefaultImpl

默认值 被禁用。

14.10.2. ca.certTransparency.log.num

ca.certTransparency.log.num 指定配置中定义的 CT 日志总数。



注意

并非所有定义的 CT 日志条目都被视为活跃；请参阅 第 14.10.3 节 “ca.certTransparency.log.<id>.*” 中的 ca.certTransparency.log.<id>.enable。

14.10.3. ca.certTransparency.log.<id>.*

ca.certTransparency.log.<id >adtrust 指定与日志 <id> 相关的信息，其中 <id> 是分配给 CT 日志服务器的唯一 id，以将其与其他 CT 日志区分开。

参数名称遵循每个 ca.certTransparency.log.<id>，并属于 <id> ：

- *ca.certTransparency.log.<id>.enable 指定 <id> CT 日志是否已启用(true)还是禁用(false)。*
- *ca.certTransparency.log.<id>.pubKey 包含 CT 日志公钥的 base64 编码。*
- *ca.certTransparency.log.<id>.url 包含 CT 日志 url 的 base64 编码。*
- *ca.certTransparency.log.<id>.version 指定 CT 支持（以及 CT 日志服务器）的 CT 版本号，目前只支持版本 1。*

第 15 章 管理证书/密钥加密策略

本章介绍了如何在加密令牌上管理证书/密钥令牌数据库的说明，特别是如何为各种场景导入和验证证书。

关于加密令牌

有关 NSS 软令牌的详情，请参考 第 2.3.7.1 节 “NSS 软令牌（内部令牌）”。

有关 HSM 的详情，请参考 第 2.3.7.2 节 “硬件安全模块(HSM、外部令牌)”。

15.1. 关于 CERTUTIL 和 PKICERTIMPORT

certutil 命令由网络安全服务(NSS)提供。certutil 用于验证和导入证书。以下是使用 certutil 的基本信息，但 PKICertImport 是我们的 wrapper 脚本，用于安全验证和导入证书。使用 certutil 来这样做，需要多个命令调用，并正确用法超出了本文档的范围。

15.1.1. certutil 基本用法

```
certutil [command] [options]
```

每个 certutil 调用都使用命令标志，通常由大写字母表示，以及控制命令的作用的一系列选项。如果某个选项使用值，则该值将在 "<" 和 ">" 符号之间命名。

15.1.2. PKICertImport 基础知识使用

```
PKICertImport [options]
```

每个 PKICertImport 调用都使用一系列选项来验证和导入指定证书。与 certutil 的广泛用例不同，PKICertImport 仅专注于安全导入和验证证书。有关可用选项的更多信息，请参阅 第 15.1.4 节 “Common certutil 和 PKICertImport 选项”。

注意

PKICertImport 在执行过程中多次提示输入 NSS DB 和/或 HSM 密码。这应该会因为 PKICertImport 必须多次与 NSS DB 交互。要避免重复输入 NSS DB 密码，请通过 -f <filename> 指定密码文件。完成后，请确保删除密码文件。

15.1.3. certutil 常用命令

以下命令特定于 certutil，并概述了几个常用命令。PKICertImport 不兼容，也不需要这些命令标志。

certutil -A

a 表示 "添加"证书。它要求证书导入(-i)、证书的别名(-n)和证书的一系列信任标志(-t)。

certutil -V

-V 命令表示"验证"证书。它需要证书别名来验证(-n)以及要执行的验证类型(-u)。

certutil -D

-D 命令表示"删除"证书。它需要一个证书别名(-n)才能删除。

请注意，它只删除证书的 PUBLIC KEY 部分，如果存在，则 WILL 不会删除任何私钥。

certutil -M

-M 命令表示"修改"证书。它要求修改证书别名(-n)，以及一系列信任标志(-t)来为其提供证书。

certutil -L

-L 命令表示证书或所有证书。如果给出 nickname 选项(-n)，它将列出该证书的详细信息，否则如果省略，它将列出所有证书的常规信息。

certutil -L 的结果会按 nickname 及其信任信息显示每个证书。例如：

表 15.1. 证书别名和信任信息

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

注意

certutil -L 显示的信任属性与使用 -t 选项指定的内容对应。

certutil -L 不修改数据库，因此可以根据需要安全地执行。

15.1.4. Common certutil 和 PKICertImport 选项

遵循以下步骤时，请确保值与您的特定部署场景相关且正确。其中一些选项也可用于 PKICertImport。

-n <nickname>

-n <nickname > 选项指定证书的 nickname。这可以是任何文本，仅用作证书的引用。它必须是唯一的。

根据您的配置更新这个值。

-d <directory>

-d <directory > 选项指定正在使用的 NSS DB 目录的路径。通常，我们假设您已在这个目录中，并使用 "." 引用当前目录。

根据您的配置更新这个值。

-t <trust>

-t <trust > 选项指定证书的信任级别。

信任有三种主要类别：

- TLS 的信任
- 对电子邮件的信任

- 对象签名的信任

每个信任位置可以有一个或多个信任字母，用于指定所需的信任级别。我们使用以下信任字母是 c、C 和 T。

- c 指出此证书应该是证书颁发机构(CA)。

- c 表示这是用于签名服务器证书的可信证书颁发机构(C 表示小写 c，因此您不需要同时指定这两个证书)。

- T 表示此证书是签名客户端证书的可信颁发机构(T 表示小写 c，因此您不需要同时指定 T 和 c)。

要为每个位置指定信任标志，请使用逗号分隔字母。例如，选项 -t CT,C,c 表示为客户端和服务器 TLS 证书签名证书、签名服务器电子邮件证书(S/MIME)并且是用于对象签名的有效 CA （尽管不受信任的）。

- 这样可确保如果此证书为另一个证书签名，则该证书用于对象签名，它将被视为无效。

使用 -t、 、 可以指定信任（或缺少信任）。

要查看数据库中所有证书的信任级别，请运行：

- certutil -L -d

- 将列出每个证书的 nickname，并在行末尾指定信任标志。

请参阅 -h 选项中的 HSMs 备注。

请注意，在 certutil man page 中指定更多信任级别。要参考本文档，请在正确安装了 certutil 的系统中运行 man certutil 命令。

-h <HSM>

-h <HSM > 选项指定要执行操作的 HSM 名称。

-h 选项与 **-t** 选项不兼容，因为 HSM 无法存储信任。只有 NSS DB 可以存储信任，因此使用 certutil -A 命令或 certutil -M 命令以及 -h <HSM> 将失败。反之，在没有 -h 选项的情况下，在单独的 certutil -M 命令中指定所需的信任级别。

根据您的配置更新这个值。

-e

-e 选项指定检查签名的有效性，与 certutil -V 命令一起使用时。PKICertImport 始终执行证书签名请求验证，且不知道 -e 选项。

-a

a 选项指定问题中的键采用 PEM (ASCII)格式。

-i <certificate>

-i <certificate > 选项指定证书的路径。这只在 certutil -A 命令中使用，以指定要导入的证书的路径。

根据您的配置更新这个值。

-u <usage>

-u < usage> 选项指定使用证书与 certutil -V 命令一起使用时进行验证。

以下部分介绍了几个使用字母。

- **-U C** 代表验证客户端 TLS 证书。请注意，这主要接受任何证书，但会检查过期日期和签名。
- **-U V** 代表验证服务器 TLS 证书。请注意，这将拒绝 CA 证书，并将检查过期日期和签名。
- **-U L** 代表验证 CA TLS 证书。请注意，这将验证信任标志（查看是否存在 c ），并将检查密钥的使用以确保密钥是 CA 密钥。这也检查过期和签名。

- -U O 代表验证 OCSP 状态响应器证书。请注意，这会检查到期和签名。

- -u J 代表验证对象签名证书。请注意，这会检查到期和签名。

如果指定了错误使用选项，或者证书上的信任标记错误（如缺少 CA TLS 证书的 c 标志），certutil -V 将给出不正确的结果。

注意

在 certutil manpage 中指定更多用法选项。要参考本文档，请在正确安装了 certutil 的系统中运行 man certutil 命令。

15.2. 导入根证书

首先，将目录改为 NSS DB:

- cd /path/to/nssdb

在执行这些步骤时，请确保您的 web 服务离线（停止、禁用等），并确保其他进程（如浏览器）无法并发访问 NSS DB。这样做可能会损坏 NSS DB，或者不当使用这些证书。

当需要导入新的 root 证书时，请确保以安全的方式获取此证书，因为它能够为多个证书签名。我们假设您已在名为 ca_root.crt 的文件中。请根据您的场景替换该文件的正确名称和路径。

有关以下使用 certutil 和 PKICertImport 选项的更多信息，请参阅 第 15.1 节 “关于 certutil 和 PKICertImport”。

导入 root 证书：

- 执行 PKICertImport -d . -n "CA Root" -t "CT,C,C" -a -i ca_root.crt -u L 命令.

此命令验证并导入 root 证书到 NSS 数据库。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出

视觉错误消息。证书通常无法验证，因为它已过期或因为它不是 CA 证书。因此，请确保您的证书文件正确且最新。联系签发者，并确保您的系统中存在所有中间证书和 root 证书。

15.3. 导入中间证书链

在开始前，请将目录改为 NSS DB:

- cd /path/to/nssdb

在执行这些步骤时，请确保您的 Web 服务离线（停止、禁用等），并确保其他进程（如浏览器）无法并发访问 NSS DB。这样做可能会损坏 NSS DB，或者不当使用这些证书。

如果您还没有导入并信任 root 证书，请参阅 第 15.2 节 “导入根证书”。

当给定 root 和结束服务器或客户端证书之间的一系列中间证书时，我们需要导入并验证签名证书链，以便从最接近的到根 CA 证书。我们假定 Intermediate CA 位于名为 ca_sub_<num>.crt 的文件中（如 ca_sub_1.crt、ca_sub_2.crt 等等）。根据您的部署替换您的证书的名称和路径。

注意

在不太可能的情况下，您会被赋予一个名为 fullchain.crt、fullchain.pem 或类似的一个文件，并通过复制每个块（包括 -----BEGIN CERTIFICATE----- 和 -----END CERTIFICATE----- 标记）将其分成上述格式。第一个命名为 ca_sub_<num>.crt，最后一个将是名为 service.crt 的服务器证书。服务器证书将在后续小节中讨论。

首先，我们将导入并验证任何中间 CA，从而从 root CA 证书中最接近。如果没有，您可以跳到下一部分。

有关以下使用 certutil 和 PKICertImport 选项的更多信息，请参阅 第 15.1 节 “关于 certutil 和 PKICertImport”。

对于链中的每个中间证书：

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

此命令将验证并导入中间 CA 证书到 NSS 数据库。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

15.4. 将证书导入到 HSM 中

此流程描述了如何在获取新发布的证书（比如当系统证书被续订）后将证书导入到 HSM 中，其密钥是在与创建 CSR 过程相同的 HSM 令牌中。

在开始前，请将目录改为 NSS DB:

- cd /path/to/nssdb，如 cd /var/lib/pki/pki-ca/alias

在执行这些步骤时，请确保您的 web 服务离线（停止、禁用等），并确保其他进程（如浏览器）无法并发访问 NSS DB。这样做可能会损坏 NSS DB，或者不当使用这些证书。

如果您还没有导入并信任 root 证书，请参阅 第 15.2 节 “导入根证书”。如果您还没有导入并验证中间证书，请参阅 第 15.3 节 “导入中间证书链”。

请注意，您遵循的哪个指令集合将取决于问题中证书的使用。

- 对于所有 PKI 子程序的 TLS 服务器证书，请按照 服务器证书步骤。
- 对于任何子系统的审计签名证书，请按照以下步骤操作来验证 对象签名证书。
- 对于 CA 子系统的签名证书，请按照上述步骤导入和验证 中间证书链，但仅对 caSigningCert 执行此操作。
- 对于 CA 子系统的 OCSP 签名证书，请按照以下步骤验证 OCSP 证书。
- 对于 PKI 子系统的所有其他系统证书，请按照 客户端证书步骤。

有关以下使用 certutil 和 PKICertImport 选项的更多信息，请参阅 第 15.1 节 “关于 certutil 和 PKICertImport”。

要在 HSM 中导入服务器证书：

- 执行 PKICertImport -d . -h HSM -n "host.name.example.com" -t "," -a -i service.crt -u V

此命令验证服务器证书并将其导入到 HSM。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。证书通常因为父证书或缺失的 CA 信任链（如缺少中间证书或缺少 CA Root）导致验证失败。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

在 HSM 中导入客户端证书：

- Execute PKICertImport -d . -h HSM -n "client name" -t ",," -a -i client.crt -u C

此命令验证客户端证书并将其导入到 HSM。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

要在 HSM 中导入对象签名证书：

- 执行 PKICertImport -d . -h HSM -n "certificate name" -t ",P" -a -i objectsigning.crt -u J

此命令验证对象签名证书并将其导入到 HSM。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

要在 HSM 中导入 OCSP 响应签名证书：

- 执行 PKICertImport -d . -h HSM -n "certificate name" -t "," -a -i ocsp.crt -u O

此命令将 OCSP 响应器证书验证并导入到 HSM。如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

15.5. 将证书导入到 NSS 数据库中

在执行这些步骤时，请确保您的 web 服务离线（停止、禁用等），并确保其他进程（如浏览器）无法并发访问 NSS 数据库。这样做可能会破坏 NSS 数据库，或者不当使用这些证书。

请注意，您遵循的哪个指令集合将取决于问题中证书的使用。

- 对于任何子系统的 auditSigningCert，请按照以下步骤操作来验证 对象签名证书。
- 对于 CA 子系统的 caSigningCert，请按照上述步骤导入和验证 中间证书链，但仅在 caSigningCert 中这样做。
- 对于 CA 子系统的 ocspSigningCert，请按照以下步骤操作来验证 OCSP 证书。
- 对于用户的客户端或 S/MIME 证书，请按照客户端证书 步骤。

有关以下使用 certutil 和 PKICertImport 选项的更多信息，请参阅 第 15.1 节 “关于 certutil 和 PKICertImport”。

将客户端证书导入到 NSS 数据库

将客户端证书导入到 NSS 数据库中：

1. 更改到 NSS 数据库目录。例如：

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

2. 导入并信任 root 证书（如果尚未导入且可信）。详情请查看 第 15.2 节 “导入根证书”。
3. 导入并验证中间证书（如果尚未导入和验证）。详情请查看 第 15.3 节 “导入中间证书链”。
4. 验证并导入客户端证书：

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

如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

导入对象签名证书

导入对象签名证书：

1. 更改到 NSS 数据库目录。例如：

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

2. 导入并信任 root 证书（如果尚未导入且可信）。详情请查看 第 15.2 节 “导入根证书”。

3. 导入并验证中间证书（如果尚未导入和验证）。详情请查看 第 15.3 节 “导入中间证书链”。

4. 验证并导入对象签名证书：

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

如果没有打印错误消息，且返回码为 0 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 echo $?。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 root 证书。

导入 OCSP 响应器

要导入 OCSP 响应程序：

1. 更改到 NSS 数据库目录。例如：

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

2. 导入并信任 root 证书（如果尚未导入且可信）。详情请查看 第 15.2 节 “导入根证书”。

3. 导入并验证中间证书（如果尚未导入和验证）。详情请查看 第 15.3 节 “导入中间证书链”。

4.

验证并导入 **OCSP** 响应程序证书：

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

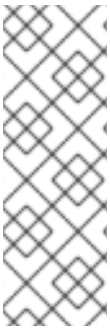
如果没有打印错误消息，且返回码为 **0** 时，验证会成功。要检查返回代码，请在执行上述命令后立即执行 **echo $?**。在大多数情况下，会输出视觉错误消息。如果验证没有成功，请联系签发者并确保系统中存在所有中间证书和 **root** 证书。

第 16 章 证书配置文件配置

16.1. 在文件系统中直接创建和编辑证书配置文件

作为 CA 的安装过程的一部分，可以通过修改配置集的配置文件直接修改证书注册配置文件。安装时默认配置集存在默认文件；当需要新配置集时，将创建新的配置集配置文件。配置文件存储在 CA 配置集目录中，instance_directory/ca/profiles/ca/，如 /var/lib/pki/pki-ca/ca/profiles/ca/。该文件命名为 profile_name.cfg。可以在这些配置集配置文件中设置或修改配置集规则的所有参数。配置集规则可以是输入、输出、身份验证、授权、默认值和约束。

CA 证书的注册配置文件位于 /var/lib/pki/instance_name/ca/conf 目录中，名称为 lf profile。



注意

出于审核原因，仅在部署前 CA 安装过程中使用此方法。

编辑配置文件后重新启动服务器，以使更改生效。

- 第 16.1.1.1 节 “配置集配置参数”
- 第 16.1.1.2 节 “在文件系统中直接修改证书扩展”
- 第 16.1.1.3 节 “在文件系统中直接添加配置文件输入”

16.1.1. 配置非 CA 系统证书配置文件

16.1.1.1. 配置集配置参数

配置集规则的所有参数 - 默认值、输入、输出和约束 - 在单个策略集中配置。为配置集设置的策略具有名称 policyset.policyName.policyNumber。例如：

```
policyset.cmcUserCertSet.6.constraint.class_id=noConstraintImpl
policyset.cmcUserCertSet.6.constraint.name=No Constraint
policyset.cmcUserCertSet.6.default.class_id=userExtensionDefaultImpl
policyset.cmcUserCertSet.6.default.name=User Supplied Key Default
policyset.cmcUserCertSet.6.default.params.userExtOID=2.5.29.15
```

常见配置集配置参数在 表 16.1 “配置文件参数” 中进行了描述。

表 16.1. 配置文件参数

参数	描述
desc	提供证书配置文件的自由文本描述，该配置文件显示在终端实体页面中。例如，*desc=This certificate profile 用于使用代理身份验证注册服务器证书。*
enable	设置配置集是否已启用，因此可通过终端实体页面访问。例如：*enable=true*。
auth.instance_id	设置身份验证管理器插件，用来验证通过配置集提交的证书请求。要进行自动注册，如果身份验证成功，CA 会立即发布证书。如果身份验证失败或者没有指定身份验证插件，则会将请求排队，来由代理手动批准。例如，*auth.instance_id=CMCAuth*。身份验证方法必须是从 **CS.cfg** 中注册的验证实例之一。
authz.acl	指定授权约束。大多数情况下，我们用来设置组评估 ACL。例如，此 caCMCUserCert 参数要求 CMC 请求的签名者属于证书管理器代理组： *authz.acl=group="Certificate Manager Agents"* 在基于目录的用户证书续订中，此选项用于确保原始请求者和当前验证的用户是同一个。 在评估授权前，实体必须验证（绑定或登录到系统）。指定的授权方法必须是从 **CS.cfg** 中注册的授权实例之一。
name	指定配置集的名称。例如，*name=Agent-Authenticated Server Certificate Enrollment*。此名称显示在最终用户注册或续订页面中。
input.list	按名称列出配置集允许的输入。例如，*input.list=i1,i2*。
input.*input_id*.class_id	按输入 ID （在 *input.list*中列出的输入名称）提供输入的 java 类名称。例如：*input.i1.class_id=cmcCertReqInputImpl*。
output.list	按名称列出配置集的可能输出格式。例如 *output.list=o1*。
output.*output_id*.class_id	给出在 *output.list* 中名为 的输出格式的 java 类名称。例如： *output.o1.class_id=certOutputImpl*。

参数	描述
policyset.list	列出配置的配置集规则。对于双证书，一组规则适用于签名密钥，另一组规则适用于加密密钥。单个证书只使用一组配置集规则。例如，*policyset.list=serverCertSet*。
policyset.*policyset_id*.list	根据策略 ID 号为配置集配置的策略集中列出策略集中的策略，按照评估它们的顺序列出。例如：*policyset.serverCertSet.list=1,2,3,4,5,6,7,8*。
policyset.*policyset_id.policy_number.*constraint.class_id	为配置集规则中配置的默认约束插件集提供 java 类名称。例如，*policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl*。
policyset.*policyset_id.policy_number.*constraint.name	提供用户定义的约束名称。例如，*policyset.serverCertSet.1.constraint.name=Subject Name Constraint*。
policyset.*policyset_id.policy_number.*constraint.params.*attribute*	为约束的允许的属性指定值。可能的属性因约束类型而异。例如，*policyset.serverCertSet.1.constraint.params.pattern=CN=adtrust*。
policyset.*policyset_id.policy_number.*default.class_id	给出配置文件规则中默认集的 java 类名称。For example, *policyset.serverCertSet.1.default.class_id=userSubjectNameDefaultImpl*
policyset.*policyset_id.policy_number.*default.name	给出用户定义的默认值的名称。例如，*policyset.serverCertSet.1.default.name=Subject Name Default*
policyset.*policyset_id.policy_number.*default.params.*attribute*	为默认值的允许的属性指定值。可能的属性因默认类型而异。例如，*policyset.serverCertSet.1.default.params.name=CN=(Name)$request.requestor_name$*。

16.1.1.2. 在文件系统中直接修改证书扩展

更改限制会更改可以提供的信息类型的限制。更改 defaults 和 constraints 也可以添加、删除或修改证书请求接受或所需的扩展。

例如，默认的 caFullCMCUserCert 配置集被设置为从请求中的信息创建 Key Usage 扩展。

> *policyset.cmcUserCertSet.6.constraint.class_id=keyUsageExtConstraintImpl*

```
policyset.cmcUserCertSet.6.constraint.name=Key Usage Extension Constraint
policyset.cmcUserCertSet.6.constraint.params.keyUsageCritical=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageCrlSign=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDataEncipherment=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageDigitalSignature=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageEncipherOnly=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyAgreement=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyCertSign=false
policyset.cmcUserCertSet.6.constraint.params.keyUsageKeyEncipherment=true
policyset.cmcUserCertSet.6.constraint.params.keyUsageNonRepudiation=true
policyset.cmcUserCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.cmcUserCertSet.6.default.name=Key Usage Default
policyset.cmcUserCertSet.6.default.params.keyUsageCritical=true
policyset.cmcUserCertSet.6.default.params.keyUsageCrlSign=false
policyset.cmcUserCertSet.6.default.params.keyUsageDataEncipherment=false
policyset.cmcUserCertSet.6.default.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.6.default.params.keyUsageDigitalSignature=true
policyset.cmcUserCertSet.6.default.params.keyUsageEncipherOnly=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyAgreement=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.cmcUserCertSet.6.default.params.keyUsageKeyEncipherment=true
policyset.cmcUserCertSet.6.default.params.keyUsageNonRepudiation=true
```

默认被更新为允许用户提供的密钥扩展：

```
policyset.cmcUserCertSet.6.default.class_id=userExtensionDefaultImpl
policyset.cmcUserCertSet.6.default.name=User Supplied Key Default
policyset.cmcUserCertSet.6.default.params.userExtOID=2.5.29.15
```

这会将服务器设置为接受证书请求中的扩展 OID 2.5.29.15。

其他限制和默认值可类似更改。确保包含适当默认值所需的限制，在需要不同约束时更改了默认值，并且默认只使用允许的约束。如需更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 Defaults Reference 部分和约束 参考部分。

16.1.1.2.1. 密钥使用和扩展的密钥用法

Red Hat Certificate System 为管理员提供了一个灵活的基础架构，可创建自定义注册配置文件来满足其环境的要求。但是，配置集不允许发布违反 RFC 5280 中定义的要求的证书。当创建密钥用法(KU)和扩展的密钥使用情况(EKU)扩展时，务必要确保根据 4.2.1.12 保持两个扩展之间的一致性。RFC 5280 的扩展密钥使用.

有关 KU 扩展的详情，请参考：

下表提供了将一致的密钥使用位映射到扩展的密钥用法扩展指南：

用途/扩展的密钥用法	密钥用法
TLS 服务器身份验证命令 **id-kp-serverAuth**	**数字签名, keyEncipherment**, 或 **KeyAgreement**
TLS 客户端(Mutual)身份验证 **id-kp-clientAuth**	**数字签名、密钥加密和/** 或 **KeyAgreement**
代码签名 **id-kp-codeSigning**	**digitalSignature**
电子邮件保护 **id-kp-emailProtection**	**数字签名、非替换** 和/或(**keyEncipherment** 或 **keyAgreement**)
OCSP 响应签名 **id-kp-OCSPSigning**	**KeyAgreement** and/or **nonRepudiation**

下面显示了两个不一致的 EKU/KU 示例：

- *用于 OCSP 响应签名的注册配置文件包含扩展密钥使用 id-kp-OCSPSigning，但 密钥 Encipherment 键使用位：*

```
policyset.ocspCertSet.6.default.class_id=keyUsageExtDefaultImpl
policyset.ocspCertSet..6.default.name=Key Usage Default
policyset.ocspCertSet..6.default.params.keyUsageCritical=true
policyset.ocspCertSet..6.default.params.keyUsageCrlSign=false
policyset.ocspCertSet..6.default.params.keyUsageDataEncipherment=false
policyset.ocspCertSet..6.default.params.keyUsageDecipherOnly=false
policyset.ocspCertSet..6.default.params.keyUsageDigitalSignature=true
policyset.ocspCertSet..6.default.params.keyUsageEncipherOnly=false
policyset.ocspCertSet..6.default.params.keyUsageKeyAgreement=false
policyset.ocspCertSet..6.default.params.keyUsageKeyCertSign=false
policyset.ocspCertSet..6.default.params.keyUsageKeyEncipherment=true
policyset.ocspCertSet..6.default.params.keyUsageNonRepudiation=true
policyset.ocspCertSet.7.constraint.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.9
policyset.ocspCertSet.7.default.class_id=extendedKeyUsageExtDefaultImpl
policyset.ocspCertSet.7.default.name=Extended Key Usage Default
policyset.ocspCertSet.7.default.params.exKeyUsageCritical=false
policyset.ocspCertSet.7.default.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.9
```

- *用于 TLS 服务器身份验证目的的注册配置文件包含扩展密钥 使用 id-kp-serverAuth，但*

CRL 签名密钥用法位：

```
policyset.serverCertSet.6.default.name=Key Usage Default
policyset.serverCertSet.6.default.params.keyUsageCritical=true
policyset.serverCertSet.6.default.params.keyUsageDigitalSignature=true
policyset.serverCertSet.6.default.params.keyUsageNonRepudiation=false
policyset.serverCertSet.6.default.params.keyUsageDataEncipherment=true
policyset.serverCertSet.6.default.params.keyUsageKeyEncipherment=false
policyset.serverCertSet.6.default.params.keyUsageKeyAgreement=true
policyset.serverCertSet.6.default.params.keyUsageKeyCertSign=false
policyset.serverCertSet.6.default.params.keyUsageCrlSign=true
policyset.serverCertSet.6.default.params.keyUsageEncipherOnly=false
policyset.serverCertSet.6.default.params.keyUsageDecipherOnly=false
policyset.cmcUserCertSet.7.default.class_id=extendedKeyUsageExtDefaultImpl
policyset.cmcUserCertSet.7.default.name=Extended Key Usage Extension Default
policyset.cmcUserCertSet.7.default.params.exKeyUsageCritical=false
policyset.serverCertSet.7.default.params.exKeyUsageOIDs=1.3.6.1.5.5.7.3.1
```

- *Red Hat Certificate System Administration Guide 中的 Extended Key Usage Extension Constraint 部分。*

- *Red Hat Certificate System Administration Guide 中的 keyUsage 部分。*

有关 EKU 扩展的详情，请参考：

- *Red Hat Certificate System Administration Guide 中的 Extended Key Usage Extension Constraint 部分。*

- *Red Hat Certificate System Administration Guide 中的 extKeyUsage 部分。*

16.1.1.2.2. 配置跨Pair 配置集

跨对证书是不同的 CA 签名证书，它们建立一个信任关系，其中来自这两个不同 PKI 的实体将相互信任。两个合作伙伴 CA 将其他 CA 签名证书存储在其数据库中，因此其他 PKI 中发布的所有证书都不被信任并识别。

证书系统支持的两个扩展可用于建立这样的信任关系（跨认证）：

- *证书策略扩展(CertificatePoliciesExtension)指定证书在之下的术语，对于每个 PKI 通常是唯一的。*

- Policy Mapping Extension (PolicyMappingExtension)通过映射两个环境的证书配置文件来封装两个 PKI 之间的信任。

发布跨对证书需要扩展证书，如 Red Hat Certificate System Administration Guide 中的 certificatePoliciesExt annex 所述。

为确保发布的证书包含 CertificatePoliciesExtension，注册配置集需要包含适当的策略规则，例如：

```
policyset.userCertSet.p7.constraint.class_id=noConstraintImpl
policyset.userCertSet.p7.constraint.name=No Constraint
policyset.userCertSet.p7.default.class_id=certificatePoliciesExtDefaultImpl
policyset.userCertSet.p7.default.name=Certificate Policies Extension Default
policyset.userCertSet.p7.default.params.Critical=false
policyset.userCertSet.p7.default.params.PoliciesExt.num=1
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.enable=true
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.policyId=1.1.1.1
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.enable=false

policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.value=
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.enable=fals

policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.explicitTex
alue=
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeRefe
ence.noticeNumbers=
policyset.userCertSet.p7.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeRefe
ence.organization=
```

本例中使用注册配置文件发布的证书将包含以下信息：

```
Identifier: Certificate Policies: - 2.5.29.32
Critical: no
Certificate Policies:
Policy Identifier: 1.1.1.1
```

有关使用跨对证书的更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 Using Cross-Pair Certificates 部分。

有关发布跨对证书的更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 发布跨 证书部分。

16.1.1.3. 在文件系统中直接添加配置文件输入

CA 配置集/ca 目录中的证书配置文件包含该特定证书配置文件表单的输入信息。输入是终端页面注册表单中的字段。有一个参数 input.list，它列出了该配置集中包含的输入。其他参数定义输入；它们通过格式输入来标识。ID.例如，这会在配置集中添加通用输入：

```
input.list=i1,i2,i3,i4
...
input.i4.class_id=genericInputImpl
input.i4.params.gi_display_name0=Name0
input.i4.params.gi_display_name1=Name1
input.i4.params.gi_display_name2=Name2
input.i4.params.gi_display_name3=Name3
input.i4.params.gi_param_enable0=true
input.i4.params.gi_param_enable1=true
input.i4.params.gi_param_enable2=true
input.i4.params.gi_param_enable3=true
input.i4.params.gi_param_name0=gname0
input.i4.params.gi_param_name1=gname1
input.i4.params.gi_param_name2=gname2
input.i4.params.gi_param_name3=gname3
input.i4.params.gi_num=4
```

有关可用的输入或表单字段的更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 Input Reference 部分。

16.1.2. 更改证书的默认有效期时间

在证书颁发机构(CA)的每个配置文件中，您可以设置使用配置文件发布的证书有效的时长。出于安全原因，您可以更改这个值。

例如，要将生成的证书颁发机构(CA)签名证书的有效性设置为 825 天（大约 27 个月），请在编辑器中打开 /var/lib/pki/instance_name/ca/profiles/ca/caCACert.cfg 文件并设置：

```
policyset.caCertSet.2.default.params.range=825
```

16.1.3. 配置 CA 系统证书配置文件

与非 CA 子系统不同，CA 自己的系统证书的注册配置文件保存在 /var/lib/pki/[instance name]/ca/conf 文件中。这些配置集为：

- caAuditSigningCert.profile

- *eccAdminCert.profile*
- *rsaAdminCert.profile*
- *caCert.profile*
- *eccServerCert.profile*
- *saServerCert.profile*
- *caOCSPCert.profile*
- *eccSubsystemCert.profile*
- *rsaSubsystemCert.profile*

如果要更改以上配置集的默认值，请在执行 第 7.7.6 节 “启动配置步骤” 过程前更改配置集。以下是演示的示例：

- *如何将有效性更改为 CA 签名证书。*
- *如何添加扩展（如证书策略扩展）。*

1. *备份 pkispawn 使用的原始 CA 证书配置文件。*

```
# cp -p /usr/share/pki/ca/conf/caCert.profile /usr/share/pki/ca/conf/caCert.profile.orig
```

2. *打开配置向导使用的 CA 证书配置文件。*

```
# vim /usr/share/pki/ca/conf/caCert.profile
```

3. 将 Validity Default 中的有效期重置为您想要的任何有效期。例如，将周期改为两年：

```
2.default.class=com.netscape.cms.profile.def.ValidityDefault
2.default.name=Validity Default
2.default.params.range=7200
```

4. 通过在配置集中创建新的默认条目并将其添加到列表中来添加任何扩展。例如，要添加证书策略扩展，请添加默认值（本例中为 default #9）：

```
9.default.class_id=certificatePoliciesExtDefaultImpl
9.default.name=Certificate Policies Extension Default
9.default.params.Critical=false
9.default.params.PoliciesExt.certPolicy0.enable=false
9.default.params.PoliciesExt.certPolicy0.policyId=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.enable=true
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.CPSURI.value=CertificatePolicies.exa
mple.com
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.enable=false
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.explicitText.value=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeReference.noticeNu
mbers=
9.default.params.PoliciesExt.certPolicy0.PolicyQualifiers0.usernotice.noticeReference.organizat
on=
```

然后，将默认数量添加到默认值列表中，以使用新默认值：

```
list=2,4,5,6,7,8,9
```

16.1.4. 管理智能卡 CA 配置文件

注意

TMS 上的功能没有在评估中测试。本节仅供参考。

TPS 不会生成或批准证书请求；它会将通过企业安全客户端批准的任何请求发送到配置的 CA 以发布证书。这意味着 CA 实际上包含用于令牌和智能卡的配置集。根据卡类型，可以自动分配要使用的配置集。

配置文件位于 /var/lib/pki/instance_name/profiles/ca/ 目录中，以及其他 CA 配置集。默认配置集列在 表 16.2 “默认令牌证书配置文件” 中。

表 16.2. 默认令牌证书配置文件

配置集名称	配置文件	描述
常规注册配置文件		
令牌设备密钥注册	caTokenDeviceKeyEnrollment.cfg	用于注册用于设备或服务器的令牌。
令牌用户加密证书注册	caTokenUserEncryptionKeyEnrollment.cfg	为用户在令牌中注册加密证书。
令牌用户签名证书注册	caTokenUserSigningKeyEnrollment.cfg	为用户在令牌中注册签名证书。
令牌用户 MS 登录证书注册	caTokenMSLoginEnrollment.cfg	用于注册用户证书以单点登录到 Windows 域或 PC。
临时令牌配置集		
临时设备证书注册	caTempTokenDeviceKeyEnrollment.cfg	在临时令牌中注册设备的证书。
临时令牌用户加密证书注册	caTempTokenUserEncryptionKeyEnrollment.cfg	为用户在临时令牌中注册加密证书。
临时令牌用户签名证书注册	caTempTokenUserSigningKeyEnrollment.cfg	为用户在临时令牌中注册签名证书。
续订配置文件[a]		
令牌用户加密证书注册（续订）	caTokenUserEncryptionKeyRenewal.cfg	如果允许续订，在令牌上为用户续订加密证书。
令牌用户签名证书注册（续订）	caTokenUserSigningKeyRenewal.cfg	如果允许续订，在令牌上为用户续订签名证书。

[a] 续订配置文件只能与发布原始证书的配置集一起使用。有两个有用的设置：

- 原始注册配置文件名称没有改变非常重要。
- 在原始注册配置文件中应设置 Renew Grace Period Constraint。这定义了在允许用户续订证书时证书过期日期前和之后的时间长度。默认配置集中只有一些示例，它们主要不启用。

16.1.4.1. 为 TPS 编辑注册配置文件

管理员可以自定义默认的智能卡注册配置文件，与 TPS 一起使用。例如，可以编辑配置文件，以在 Subject Alternative Name 扩展中包含用户的电子邮件地址。用户的电子邮件地址从身份验证目录检

索。要为 LDAP 访问配置 CA，请使用适当的目录信息更改配置集文件中的以下参数：

```
policyset.set1.p1.default.params.dnpattern=UID=$request.uid$, O=Token Key User
policyset.set1.p1.default.params.ldap.enable=true
policyset.set1.p1.default.params.ldap.basedn=ou=people,dc=host,dc=example,dc=com
policyset.set1.p1.default.params.ldapStringAttributes=uid,mail
policyset.set1.p1.default.params.ldap.ldapconn.host=localhost.example.com
policyset.set1.p1.default.params.ldap.ldapconn.port=389
```

这些 CA 配置集默认为禁用 LDAP 查找。ldapStringAttributes 参数告知 CA 从 company 目录检索哪些 LDAP 属性。例如，如果目录包含 uid 作为 LDAP 属性名称，这将在证书的主题名称中使用，则 uid 必须列在 ldapStringAttributes 参数中，并且 request.uid 在 dnpattern 中列为其中一个组件。

编辑证书配置文件包括在 Red Hat Certificate System Administration Guide 中的 Setting up Certificate Profiles 部分。

dnpattern 参数的格式包括在 Subject Name Constraint 部分和 Red Hat Certificate System Administration Guide 中的 Subject Name Default 部分。

16.1.4.2. 创建自定义 TPS 配置集

证书配置文件在 CA 中正常创建，但还必须在 TPS 中配置，以便其可用于令牌注册。

TIP

使用新版本的 Red Hat Certificate System 添加新配置集。如果实例迁移到证书系统 10.0，则需要将新配置集添加到迁移的实例中，就像它们是自定义配置集一样。

1. 为发布 CA 创建新令牌配置文件。设置配置文件的内容包括在 Red Hat Certificate System Administration Guide 中的 Setting up Certificate Profiles 部分。

2. 将配置集复制到 CA 的配置集目录 /var/lib/instance_name/ca/profiles/ca/ 中。

3. 编辑 CA 的 CS.cfg 文件，并将新配置集引用和配置集名称添加到 CA 的配置集列表中。例如：

```
# vim etc/pki/instance_name/ca/CS.cfg
```

```
profile.list=caUserCert,...,caManualRenewal,tpsExampleEnrollProfile
...
profile.caTokenMSLoginEnrollment.class_id=caUserCertEnrollImpl

profile.caTokenMSLoginEnrollment.config=/var/lib/pki/instance_name/profiles/ca/tpsExampleEnrollProfile.cfg
```

4. 编辑 **TPS CS.cfg** 文件，并添加一行以指向新的 **CA** 注册配置文件。例如：

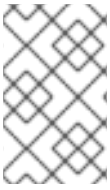
```
# vim /etc/pki/instance_name/tps/CS.cfg

op.enroll.userKey.keyGen.signing.ca.profileId=tpsExampleEnrollProfile
```

5. 编辑智能卡配置集后重启实例：

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

如果 **CA** 和 **TPS** 位于单独的实例中，请重启两个实例。



注意

在用户 **LDAP** 条目中配置外部注册(**externalReg**)设置的注册配置文件。

16.1.4.3. 使用 Windows 智能卡登录配置文件

TPS 使用配置集来生成用于对 **Windows** 域或 **PC** 的单点登录的证书；这是 **Token User MS Login Certificate Enrollment** 配置集(**caTokenMSLoginEnrollment.cfg**)。

但是，在配置 **Windows** 智能卡登录时，管理员必须考虑一些特殊考虑。

- 如果尚未为 **TLS** 配置证书，向域控制器发布证书。
- 为每个用户配置智能卡登录，而不是作为全局策略，以防止锁定域管理员。
- 启用 **CRL** 发布到 **Active Directory** 服务器，因为域控制器在每次登录时检查 **CRL**。

16.1.5. 禁用证书 Enrolment 配置集

本节介绍如何禁用所选配置集。

要禁用证书配置文件，请编辑 /var/lib/pki/instance_name/ca/profiles/ca/ 目录中对应的 lf cfg 文件，并将 visible 和 enable 参数设置为 false。

例如，禁用所有非CMC 配置集：

1. 列出所有非CMC 配置集：

```
# ls -l /var/lib/pki/instance_name/ca/profiles/ca/ | grep -v "CMC"
```

2. 在每个显示的文件中，将以下参数设置为 false ：

```
visible=false
enable=false
```

另外，在所有 CMC 配置集中设置 visible=false，使其在终端实体页面中不可见：

1. 列出所有 CMC 配置集：

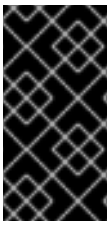
```
# ls -l /var/lib/pki/instance_name/ca/profiles/ca/*CMC*
```

2. 在每个显示的文件中设置：

```
visible=false
```

第 17 章 配置密钥恢复授权

17.1. 手动设置密钥存档



重要

如果 CA 和 KRA 位于同一安全域中，则不需要这个过程。只有安全域 外的 CA 才需要这个过程。

手动配置关键存档需要两件事情：

- 在 CA 和 KRA 之间具有可信关系。
- 为密钥归档启用了注册表单后，这意味着它配置了密钥存档，并以表单存储 KRA 传输证书。

在同一安全域中，配置 KRA 时 自动执行 这两个配置步骤，因为它被配置为与其安全域中的任何 CA 具有可信关系。通过手动配置信任关系和配置文件注册表单，可以创建与安全域之外的证书管理器的信任关系。

1. 如有必要，创建一个可信管理器来建立证书管理器和 KRA 之间的关系。

   要使 CA 能够请求 KRA 的密钥存档，必须配置两个子系统才能识别、信任并相互通信。验证证书管理器是否已设置为具有 KRA 内部数据库中的适当 TLS 客户端身份验证证书的特权用户。默认情况下，证书管理器将其子系统证书用于 TLS 客户端身份验证到 KRA。

2. 复制 KRA 的 base-64 编码传输证书。

   传输证书存储在 KRA 的证书数据库中，可以使用 certutil 工具来检索。如果传输证书由证书管理器签名，则通过 Retrieval 选项卡中的证书管理器终端实体页面提供证书副本。

   或者，使用 pki ultility 下载传输证书：

```
# pki cert-find --name "KRA Transport certificate's subject common name"
# pki cert-show serial_number --output transport.pem
```

3.

将传输证书添加到 CA 的 CS.cfg 文件中。

```
ca.connector.KRA.enable=true
ca.connector.KRA.host=server.example.com
ca.connector.KRA.local=false
ca.connector.KRA.nickName=subsystemCert cert-pki-ca
ca.connector.KRA.port=8443
ca.connector.KRA.timeout=30
ca.connector.KRA.transportCert=MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUF
ADA6MRgwFgYDVQQKEw9Eb21haW4gc28gbmFtZWQxHjAcBgNVBAMTFUNlcnRpZml
jYXRlIEF1dGhvcml0eTAeFw0wNjExMTQxODI2NDdaFw0wODEwMTQxNzQwNThaMD4x
GDAWBgNVBAoTD0RvbWFpbiBzbyBuYW1lZDEiMCAGA1UEAxMZRFJNIFRyYW5zcG9
ydCBDZXJ0aWZpY2F0ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBA
KnMGB3WkznueouwZjrWLFZBLpKt6TimNKV9iz5s0zrGUlpdt81/BTsU5A2sRUwNfoZSM
s/d5KLuXOHPyGtmC6yVvaY719hr9EGYuv0Sw6jb3WnEKHpjbUO/vhFwTufJHWKXFN3
V4pMbHTkqW/x5fu/3QyyUre/5lhG0fcEmfvYxlyvZUJx+aQBW437ATD99Kuh+I+FuYdW+
SqYHznHY8BqOdJwJ1JiJMNceXYAuAdk+9t70RztfAhBmkK0OOP0vH5BZ7RCwE3Y/6y
cUdSyPZGGc76a0HrKOz+lwVFulFStiuZlaG1pv0NNivzcj0hEYq6AfJ3hgxcC1h87LmCxg
RWUCAwEAAaN5MHcwHwYDVR0jBBgwFoAURShCYtSg+Oh4rrgmLFB/Fg7X3qcwRAY
IKwYBBQUHAQEEODA2MDQGCCsGAQUFBzABhihodHRwOi8vY2x5ZGUucmR1LnJlZ
GhhdC5jb206OTE4MC9jYS9vY3NwMA4GA1UdDwEB/wQEAwIE8DANBgkqhkiG9w0BA
QUFAAOCAQEAFYz5ibujdIXgnJCbHSPWdKG0T+FmR67YqiOtoNlGylgJ42fi5lsDPfCblA
e3YFqmF3wU472h8LDLGyBjy9RJxBj+aCizwHkuoH26KmPGntlayqWDH/UGsIL0mvTSO
eLql3KM0luH7bxGXjlION83xWbxumW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWyYdfEO
jpp/UQZOhOM1d8GFiw8N8ClWBGc3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwlF
hC82zqeQ1PbxQDL8QLjzca+IUzq6Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVls
kXJw5ZR0Y2zA==
ca.connector.KRA.uri=/kra/agent/kra/connector
```

4.

然后，编辑注册表，并在 keyTransportCert 方法中添加或替换传输证书值。

```
vim /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/ca/ProfileSelect.template

var keyTransportCert =
MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUFADA6MRgwFgYDVQQKEw9Eb21
haW4gc28gbmFtZWQxHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0wNj
ExMTQxODI2NDdaFw0wODEwMTQxNzQwNThaMD4xGDAWBgNVBAoTD0RvbWFpbiB
zbyBuYW1lZDEiMCAGA1UEAxMZRFJNIFRyYW5zcG9ydCBDZXJ0aWZpY2F0ZTCCASI
wDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKnMGB3WkznueouwZjrWLFZBL
pKt6TimNKV9iz5s0zrGUlpdt81/BTsU5A2sRUwNfoZSMs/d5KLuXOHPyGtmC6yVvaY719
hr9EGYuv0Sw6jb3WnEKHpjbUO/vhFwTufJHWKXFN3V4pMbHTkqW/x5fu/3QyyUre/5lh
G0fcEmfvYxlyvZUJx+aQBW437ATD99Kuh+I+FuYdW+SqYHznHY8BqOdJwJ1JiJMNceX
YAuAdk+9t70RztfAhBmkK0OOP0vH5BZ7RCwE3Y/6ycUdSyPZGGc76a0HrKOz+lwVFul
FStiuZlaG1pv0NNivzcj0hEYq6AfJ3hgxcC1h87LmCxgRWUCAwEAAaN5MHcwHwYDVR
0jBBgwFoAURShCYtSg+Oh4rrgmLFB/Fg7X3qcwRAYIKwYBBQUHAQEEODA2MDQGC
CsGAQUFBzABhihodHRwOi8vY2x5ZGUucmR1LnJlZGhhdC5jb206OTE4MC9jYS9vY3N
wMA4GA1UdDwEB/wQEAwIE8DANBgkqhkiG9w0BAQUFAAOCAQEAFYz5ibujdIXgnJC
bHSPWdKG0T+FmR67YqiOtoNlGylgJ42fi5lsDPfCblAe3YFqmF3wU472h8LDLGyBjy9RJ
xBj+aCizwHkuoH26KmPGntlayqWDH/UGsIL0mvTSOeLql3KM0luH7bxGXjlION83xWbxu
```

> mW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWyYdfEOjpp/UQZOhOM1d8GFiw8N8ClWBG
> c3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwlFhC82zqeQ1PbxQDL8QLjzca+lUzq6
> Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVlskXJw5ZR0Y2zA==;

17.2. 加密 OF KRA 操作

证书系统在密钥恢复授权机构(KRA)中加密以下密钥操作：

- archival:
  - 在证书请求消息格式(CRMF)软件包中归档的密钥加密，以传输到 KRA。
  - 加密 KRA LDAP 数据库中存储的密钥。
- 恢复：
  - 用户提供的会话密钥加密，以传输到密钥。
  - 使用用户提供的会话密钥或创建 PKCSbusybox 软件包来解密 secret 和重新加密。
- generation:
  - 为存储加密生成的密钥。

17.2.1. 客户端如何管理密钥操作加密

证书系统客户端自动使用 KRA 配置中设置的加密算法，且不需要进一步的操作。

17.2.2. 在 KRA 中配置加密算法

注意

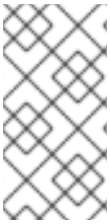
只有 AES CBC （如果 kra.allowEncDecrypt.archive=true 和 kra.allowEncDecrypt.recovery=true）和 AES Key Wrap （如果为 kra.allowEncDecrypt.archive=false 和 kra.allowEncDecrypt.recovery=false）在以下配置中被允许。任何 FIPS140-2 验证的 HSM，它们都支持 KRA 提供的密钥存档和恢复功能。

证书系统在 /var/lib/pki/pki-instance_name/conf/kra/CS.cfg 文件中定义与密钥操作加密相关的配置参数组。我们推荐以下一组参数（请参阅上面的备注用于其他选项）：

```
kra.allowEncDecrypt.archive=false
kra.allowEncDecrypt.recovery=false
kra.storageUnit.wrapping.1.sessionKeyLength=256
kra.storageUnit.wrapping.1.sessionKeyWrapAlgorithm=RSA
kra.storageUnit.wrapping.1.payloadEncryptionPadding=PKCS5Padding
kra.storageUnit.wrapping.1.sessionKeyKeyGenAlgorithm=AES
kra.storageUnit.wrapping.1.payloadEncryptionAlgorithm=AES
kra.storageUnit.wrapping.1.payloadEncryptionMode=CBC
kra.storageUnit.wrapping.1.payloadWrapAlgorithm=AES KeyWrap
kra.storageUnit.wrapping.1.sessionKeyType=AES
kra.storageUnit.wrapping.1.payloadWrapIVLen=16
kra.storageUnit.wrapping.choice=1
```

每个组(kra.storageUnit.wrapping.0 需要 kra.storageUnit.wrapping.1unless)都有单独的设置，数字定义了哪些设置属于同一配置组。当前配置组在 /var/lib/pki/pki-instance_name/conf/kra/CS.cfg 文件中的 kra.storageUnit.wrapping.choice 参数中设置。

在继续操作前，请确保在配置文件中设置了 kra.storageUnit.wrapping.choice=1。



注意

证书系统将解密数据所需的信息添加到 KRA 数据库中的记录中。因此，即使在更改加密算法后，证书系统仍能够使用不同的加密算法解密之前存储在 KRA 中的数据。

17.2.2.1. 参数及其值的说明

每个 secret （一个"payload"）都使用会话密钥加密。控制此加密的参数的前缀为 。要使用的参数集合取决于 kra.allowEncDecrypt.archive 和 kra.allowEncDecrypt.recovery 的值。默认情况下，它们都是 false。请参阅 第 17.2.2.2 节 “在 KRA 中使用 AES 加密时解决 HSM 的限制” 以了解这些两个参数对 HSM 的影响。

当 kra.allowEncDecrypt.archive 和 kra.allowEncDecrypt.recovery 都为 false 时：

- payloadWrapAlgorithm 决定使用的嵌套算法。唯一一个有效的选择是 AES KeyWrap。
- 当 payloadWrapAlgorithm=AES/CBC/PKCS5Padding 时，必须指定 payloadWrapIVLength=16 来告知 PKI 我们需要生成一个 IV （因为 CBC 需要 1）。

当 kra.allowEncDecrypt.archive 和 kra.allowEncDecrypt.recovery 都为 true 时：

- payloadEncryptionAlgorithm 决定使用的加密算法。唯一有效的选择是 AES。
- payloadEncryptionMode 决定块链模式。唯一有效的选择是 CBC。
- payloadEncryptionPadding 决定 padding 方案。唯一有效的选择是 PKCS5Padding。

然后，会话密钥会随 KRA 存储证书(RSA 令牌)嵌套。控制会话密钥及其加密的参数使用 sessionKey 前缀。

- sessionKeyType 是要生成的密钥类型。唯一有效的选择是 AES。
- sessionKeyLength 是生成的会话键的长度。有效选择为 128 和 256，用于分别使用 128 位 AES 或 256 位 AES 加密有效负载。
- sessionKeyWrapAlgorithm 是 KRA 存储证书的密钥类型。本指南中唯一有效的选择是 RSA。

17.2.2.2. 在 KRA 中使用 AES 加密时解决 HSM 的限制

如果您在 KRA 中运行启用了 AES 的证书系统，但硬件安全模块(HSM)不支持 AES 密钥嵌套功能，则密钥归档会失败。要解决这个问题，支持以下解决方案：

- “为密钥封装选择不同的算法”一节
- 第 7.7.5.5 节 “将 KRA 设置为加密模式”

为密钥封装选择不同的算法

有时，KRA 不支持默认的密钥嵌套算法，但它支持其他算法。例如，使用 AES-128-CBC 作为密钥嵌套算法：

1. 在 /var/lib/pki/pki-instance_name/conf/kra/CS.cfg 文件中设置以下参数：

```
kra.storageUnit.wrapping.1.payloadWrapAlgorithm=AES KeyWrap
kra.storageUnit.wrapping.1.payloadWrapIVLen=16
kra.storageUnit.wrapping.1.sessionKeyLength=128
```

2. 重启实例：

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

或（如果使用 nuxwdog watchdog）

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

如果 KRA 在不同的实例中运行，则 CA 会重启这两个实例。

为密钥嵌套选择不同的算法具有好处：如果 HSM 之后添加了对 AES 密钥嵌套的支持，您可以恢复设置，因为密钥记录设置了相关信息。

此配置使用 kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen} 和 kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding} 参数。

将 KRA 设置为加密模式

如果 HSM 不支持任何 KeyWrap 算法，则需要在有些情况下将 KRA 置于加密模式。当将 KRA 设置为加密模式时，所有密钥都将使用加密算法而不是密钥嵌套算法存储。

将 KRA 设置为加密模式：

1. 将 /var/lib/pki/pki-instance_name/conf/kra/CS.cfg 文件中的以下参数设置为 true ：

```
kra.allowEncDecrypt.archive=true
kra.allowEncDecrypt.recovery=true
```

2. 重启服务：

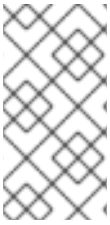
```
# systemctl restart pki-tomcatd@instance_name.service
```

或（如果使用 nuxwdog watchdog）

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

如果 KRA 在与 CA 不同的实例中运行，您需要重启这两个实例。

此配置使用 kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding} 和 kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen} 参数。



注意

如果您稍后根据 “为密钥封装选择不同的算法”一节 切换到密钥换行的不同算法，您必须手动将 KRA 设置为加密模式前创建的记录。

17.3. 设置 AGENT-APPROVED KEY RECOVERY SCHEMES

密钥恢复代理会集中授权并检索 PKCSRG 软件包中的私钥和相关证书。要授权密钥恢复，所需的恢复代理访问 KRA 代理服务页面，并使用 Authorize Recovery 区域来单独输入每个授权。

其中一个代理启动密钥恢复过程。对于同步恢复，每个批准代理都使用参考号（通过初始请求返回）来打开请求，然后单独授权密钥恢复。对于异步恢复，批准代理都会搜索密钥恢复请求，然后授权密钥恢复。无论哪种方式，每当输入所有授权时，KRA 会检查信息。如果显示的信息正确，它会检索请求的密钥，并以 PKCSGRESS 软件包的形式返回对应的证书，到启动密钥恢复过程的代理。

密钥恢复代理方案 将 KRA 配置为识别密钥恢复代理所属的组，并指定在恢复归档密钥前需要授权这些代理的数量。

17.3.1. 在命令行中配置代理批准密钥恢复

要设置代理发起的密钥恢复，请编辑 KRA 配置中的两个参数：

- 设置恢复管理器的数量，需要批准恢复。
- 设置这些用户必须属于的组。

这些参数在 KRA 的 CS.cfg 配置文件中设置。

1. 在编辑配置文件前停止服务器。

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

或（如果使用 nuxwdog watchdog）

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

2. 打开 KRA 的 CS.cfg 文件。

```
# vim /var/lib/pki/pki-tomcat/kra/conf/CS.cfg
```

3. 编辑两个恢复方案参数。

```
kra.noOfRequiredRecoveryAgents=3
kra.recoveryAgentGroup=Key Recovery Authority Agents
```

4. 重新启动服务器。

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

或者

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

注意

有关如何在控制台中配置代理批准的密钥恢复的更多信息，请参阅 Red Hat Certificate System Administration Guide 中的 Console 部分中的配置代理批准的密钥恢复 部分。

17.3.2. 自定义密钥恢复表单

默认密钥代理方案需要来自密钥恢复授权代理组的单个代理，以负责授权密钥恢复。

也可以自定义密钥恢复表单的外观。关键恢复代理需要适当的页面来启动密钥恢复过程。默认情况下，KRA 的代理服务页面包含适当的 HTML 表单，允许密钥恢复代理启动密钥恢复、授权密钥恢复请求并检索加密密钥。这个形式位于 /var/lib/pki/pki-tomcat/kra/webapps/kra/agent/kra/ 目录中，名为 confirmRecover.html。

重要

如果自定义密钥恢复确认表单，请不要删除用于生成响应的任何信息。这对表单的功能至关重要。限制表单中内容的任何更改和外观。

17.3.3. 在新的私有存储密钥中重新打包密钥

在 KRA 中存档时，一些私钥（保留在旧部署中）被嵌套在 SHA-1 中，1024 位存储密钥被嵌套。随着处理器加速改进和算法被损坏，这些算法变得不太安全。作为一种安全措施，可以将私钥重新打包到新的、更强大的存储密钥(SHA-256、2048 位密钥)。

17.3.3.1. 关于 KRATool

重新打包和移动密钥注册和密钥注册和恢复请求是使用 KRATool 实用程序完成的（在之前的 Red Hat Certificate System 版本中称为 DRMTool）。

KRATool 执行两个操作：它可以使用新私钥重新打包密钥，并且它可以重新数字 key 记录的 LDIF 文件条目中的属性，包括注册和恢复请求。必须至少执行一个操作(rewrap 或 renumber)，且它们可以在单个调用中执行。

对于重新包装密钥，工具会访问原始 KRA 文件中导出的 LDIF 文件中的密钥条目，使用原始 KRA 存储密钥来展开密钥，然后在新的 KRA 的更强大的存储密钥中封装密钥。

例 17.1. 重新包装密钥

```
KRATool -kratool_config_file "/usr/share/pki/java-tools/KRATool.cfg" -source_ldif_file
"/tmp/files/originalKRA.ldif" -target_ldif_file "/tmp/files/newKRA.ldif" -log_file
"/tmp/kratool.log" -source_pki_security_database_path "/tmp/files/" -
source_storage_token_name "Internal Key Storage Token" -
source_storage_certificate_nickname "storageCert cert-pki-tomcat KRA" -
target_storage_certificate_file "/tmp/files/omega.cert"
```

当多个 KRA 实例合并到一个实例中时，务必要确保没有密钥或请求记录冲突 CN、DN、序列号或请求 ID 号。可以处理这些值，将更大的值附加到现有值中。

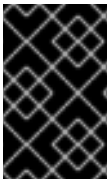
例 17.2. 重新编号密钥

```
KRATool -kratool_config_file "/usr/share/pki/java-tools/KRATool.cfg" -source_ldif_file
"/tmp/files/originalKRA.ldif" -target_ldif_file "/tmp/files/newKRA.ldif" -log_file
"/tmp/kratool.log" -append_id_offset 100000000000 -source_kra_naming_context "pki-
tomcat-KRA" -target_kra_naming_context "pki-tomcat-2-KRA" -
process_requests_and_key_records_only
```

KRATool 选项及其配置文件在 KRATool (1) 手册页中详细讨论。

17.3.3.2. 将来自一个或多个 KRA 的 wrapping 和 Merging 键重新嵌套到单个 KRA 中

此流程重新打包存储在一个或多个证书系统 KRA 中的密钥（例如，sourcekra.example.com上的 pki-tomcat ），并将它们存储在另一个证书系统 KRA 中（例如，targetkra.example.com上的 pki-tomcat- 2）。这不是唯一的用例；工具可以和源和目标在同一实例上运行，以重新包装现有密钥，或者只是将密钥从多个 KRA 实例复制到单个实例中，而无需全部重新包装密钥。



重要

pki-tomcat-2 KRA 中的存储密钥大小和类型必须设置为 2048 位和 RSA。

1. 登录到 targetkra.example.com。

2.

停止 pki-tomcat-2 KRA。

```
[root@targetkra ~]# systemctl stop pki-tomcatd@pki-tomcat-2.service
```

3.

创建一个数据目录，以存储来自 pki-tomcat KRA 实例导入的关键数据，它位于 sourcekra.example.com 上。

```
[root@targetkra ~]# mkdir -p /export/pki
```

4.

将 pki-tomcat-2 KRA 的公共存储证书导出到新数据目录中的平面文件。

```
[root@targetkra ~]# certutil -L -d /var/lib/pki/pki-tomcat-2/alias -n "storageCert cert-pki-tomcat-2 KRA" -a > /export/pki/targetKRA.cert
```

5.

停止 pki-tomcat-2 KRA 的 Directory 服务器实例（如果位于同一机器上）。

```
[root@newkra ~]# systemctl stop dirsrv.target
```

6.

导出 pki-tomcat-2 KRA 的配置信息。

```
root@targetkra ~]# grep nsslapd-localuser /etc/dirsrv/slapd-instanceName/dse.ldif
  nsslapd-localuser: dirsrv

[root@targetkra ~]# chown dirsrv:dirsrv /export/pki

[root@targetkra ~]# /usr/lib64/dirsrv/slapd-instanceName/db2ldif -n pki-tomcat-2-KRA -a /export/pki/pki-tomcat-2.ldif
```

重要

确保 LDIF 文件末尾包含一个空白行。

7. 登录 sourcekra.example.com。

8. 停止 pki-tomcat KRA。

```
[root@sourcekra ~]# systemctl stop pki-tomcatd@pki-tomcat.service
```

9. 创建一个数据目录，以存储将导出到 targetkra.example.com 上的 pki-tomcat-2 KRA 实例的密钥数据。

```
[root@sourcekra ~]# mkdir -p /export/pki
```

10. 停止 pki-tomcat KRA 的 Directory 服务器实例（如果位于同一机器上）。

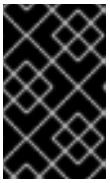
```
[root@sourcekra ~]# systemctl stop dirsrv.target
```

11. 导出 pki-tomcat KRA 的配置信息。

```
[root@sourcekra ~]# grep nsslapd-localuser /etc/dirsrv/slapd-instanceName/dse.ldif
  nsslapd-localuser: dirsrv

[root@sourcekra ~]# chown dirsrv:dirsrv /export/pki

[root@sourcekra ~]# /usr/lib64/dirsrv/slapd-instanceName/db2ldif -n pki-tomcat-KRA -a
/export/pki/pki-tomcat.ldif
```



重要

确保 LDIF 文件末尾包含一个空白行。

12. 将 pki-tomcat KRA NSS 安全数据库复制到这个目录中。

```
[root@sourcekra ~]# cp -p /var/lib/pki/pki-tomcat/alias/cert9.db /export/pki

[root@sourcekra ~]# cp -p /var/lib/pki/pki-tomcat/alias/key4.db /export/pki

[root@sourcekra ~]# cp -p /var/lib/pki/pki-tomcat/alias/pkcs11.txt /export/pki
```

13.

使用 pki-tomcat-2 KRA 机器中的公钥将该文件复制到此计算机。例如：

```
[root@sourcekra ~]# cd /export/pki

[root@sourcekra ~]# sftp root@targetkra.example.com
  sftp> cd /export/pki
  sftp> get targetKRA.cert
  sftp> quit
```

14.

如有必要，编辑默认的 KRATool.cfg 文件以用于该工具。默认 文件也可以在不更改的情况下使用。

15.

运行 KRATool; 所有这些参数都应该在一行中：

```
[root@sourcekra ~]# KRATool -kratool_config_file "/usr/share/pki/java-
tools/KRATool.cfg"
        -source_ldif_file /export/pki/pki-tomcat.ldif \
        -target_ldif_file /export/pki/source2targetKRA.ldif \
        -log_file /export/pki/kratool.log \
        -source_pki_security_database_path /export/pki \
        -source_storage_token_name 'Internal Key Storage Token' \
        -source_storage_certificate_nickname 'storageCert cert-pki-tomcat KRA' \
        -target_storage_certificate_file /export/pki/targetKRA.cert
        -append_id_offset 100000000000 \
        -source_kra_naming_context "pki-tomcat-KRA" \
        -target_kra_naming_context "pki-tomcat-2-KRA" \
        -process_requests_and_key_records_only
```

注意

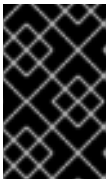
命令可能会提示输入存储在 pki-tomcat KRA NSS 安全数据库中的令牌的密码。

完成后，命令会创建 -target_ldif_file 参数 source2targetKRA.ldif 中指定的文件。

16.

将此 LDIF 文件复制到 pki-tomcat-2 KRA 机器。例如：

```
[root@sourcekra ~]# scp /export/pki/source2targetKRA.ldif
root@targetkra.example.com:/export/pki
```



重要

确保 LDIF 文件末尾包含一个空白行。

17.

如果多个 KRA 实例被合并，则其数据可以合并到单个导入操作中。只需为每个 KRA 执行相同的步骤，这些步骤将被合并。

重要

确保为 -target_ldif_file 参数指定唯一值以创建单独的 LDIF 文件，并指定 unique -append_id_offset 值，以便在 LDIF 文件串联时没有冲突。

18.

在 pki-tomcat-2 KRA 机器上，通过串联 pki-tomcat-2 KRA 配置 LDIF 文件以及其它 KRA 实例的每个导出的 LDIF 文件来与其他密钥数据导入 LDIF 文件。例如：

```
[root@targetkra ~]# cd /export/pki
[root@targetkra ~]# cat pki-tomcat-2.ldif source2targetKRA.ldif > combined.ldif
```

19.

将此组合的 LDIF 文件导入到 pki-tomcat-2 KRA 实例的目录服务器数据库中。

```
[root@targetkra ~]# /usr/lib64/dirsrv/slapd-instanceName/ldif2db -n pki-tomcat-2-KRA -i
/export/pki/combined.ldif
```

20.

启动 pki-tomcat-2 KRA 的 Directory 服务器实例。

```
[root@targetkra ~]# systemctl start dirsrv.target
```

21.

启动 pki-tomcat-2 KRA。

```
[root@targetkra ~]# systemctl start pki-tomcatd@pki-tomcat-2.service
```

17.3.4. 在关闭后更新 CA-KRA Connector 信息

有关在克隆后更新 CA-KRA 信息的更多信息，请参阅 第 17.3.4 节 “在关闭后更新 CA-KRA Connector 信息”。

第 18 章 配置日志

证书系统子系统日志文件记录与该特定子系统实例内操作相关的事件。对于每个子系统，会为安装、访问和 Web 服务器等问题保存不同的日志。

所有子系统都有类似的日志配置、选项和管理路径。

有关安装后日志管理的详情，请参考 Red Hat Certificate System Administration Guide 中的 Configuring Subsystem Logs 部分。

有关日志的概述，请参阅 第 2.3.14 节 “日志”。

18.1. 证书系统日志设置

配置日志的方式可能会影响证书系统性能。例如，日志文件轮转使日志变得太大，这会减慢子系统性能。本节介绍证书系统子系统记录的不同类型的日志，并涵盖了日志文件轮转、缓冲的日志以及可用日志级别等重要概念。

18.1.1. 已登录的服务

证书系统日志消息到日志文件的所有主要组件和协议。表 18.1 “服务日志” 列出默认日志记录的服务。要查看特定服务记录的消息，请相应地自定义日志设置。

表 18.1. 服务日志

服务	描述
ACL	与访问控制列表相关的日志事件。
管理	记录与管理活动相关的事件，如控制台和实例之间的 HTTPS 通信。
All	记录与所有服务相关的事件。
身份验证	使用身份验证模块记录与活动相关的事件。
证书颁发机构	记录与证书管理器相关的事件。

服务	描述
数据库	记录与内部数据库相关的活动事件。
HTTP	记录与服务器的 HTTP 活动相关的事件。请注意，HTTP 事件实际上被记录到属于证书系统所包含的 Apache 服务器的错误日志中，以提供 HTTP 服务。
密钥恢复授权机构	日志记录与 KRA 相关的事件。
LDAP	日志记录与 LDAP 目录相关的活动事件，用于发布证书和 CRL。
OCSP	日志记录与 OCSP 状态相关的事件，如 OCSP 状态 GET 请求。
其他	记录与其他活动相关的事件，如命令行实用程序和其他进程。
请求队列	日志记录与请求队列活动相关的事件。
用户和组	日志记录与实例的用户和组相关的事件。

18.1.2. 日志级别(Message Categories)

证书系统服务记录的不同事件由日志级别决定，这有助于识别和过滤事件。不同的证书系统日志级别列在 表 18.2 “Log Levels 和 Corresponding Log Messages” 中。

日志级别由数字 0 到 10 表示，每个数字代表服务器要执行的日志记录级别。该级别设置如何详细记录。

优先级更高的级别表示较少的详细信息，因为只记录高优先级的事件。

注意

默认日志级别为 1，不应更改这个值。要启用调试日志记录，请参阅 第 18.3.3 节 “Debug Log 的额外配置”。

表 18.2 “Log Levels 和 Corresponding Log Messages” 提供用于更好地了解日志消息的参考。

表 18.2. Log Levels 和 Corresponding Log Messages

日志级别	消息类别	描述
0	调试	这些消息包含调试信息。不建议常规使用这个级别，因为它会生成太多信息。
1	informational （审计日志的默认选择）	这些消息提供有关证书系统状态的常规信息，包括 证书系统初始化完成 等状态信息，以及操作 Request for 操作成功。
2	警告	这些消息只是警告，并不表示服务器正常操作中的任何失败。
3	失败；系统和错误日志的默认选择	这些消息表示阻止服务器正常运行的错误和失败，包括执行证书服务操作失败（用户身份验证失败 或证书 被撤销）和可能导致不良错误的意外情况（服务器无法通过来自客户端的同一频道向客户端发送请求）。
4	错误配置	这些消息表示服务器中的错误配置导致错误。
5	灾难性故障	这些消息表示因为错误，服务无法继续运行。
6	与安全相关的事件	这些消息标识了影响服务器安全性的发生。例如，带有撤销或未列出证书的用户尝试特权访问权限。
7	与 PDU 相关的事件（调试）	这些消息包含 PDU 事件的调试信息。不建议常规使用这个级别，因为它会生成比通常有用的信息。
8	与 PDU 相关的事件	这些消息与在 PDU 上处理的事务和规则相关，如创建 MAC 令牌。
9	与 PDU 相关的事件	此日志级别为 PDU 上处理的事件提供详细的日志消息，如创建 MAC 令牌。

日志级别	消息类别	描述
10	所有日志记录级别	这个日志级别启用了所有日志记录级别。

日志级别可用于根据事件的严重性过滤日志条目。默认情况下，为所有服务设置日志级别 3 (Failure)。

日志级别是连续的；指定值 3 会导致记录级别 4、5 和 6。日志数据可能非常广泛，特别是在较低（详细）日志级别时。确保主机机器有足够的磁盘空间用于所有日志文件。定义日志级别、日志轮转和 server-backup 策略也非常重要，以便备份所有日志文件，并且主机系统不会过载；否则，信息可能会丢失。

18.1.3. buffered 和 Unbuffered Logging

Java 子系统支持对所有类型的日志进行缓冲的日志。可以为 buffered 或未缓冲的日志配置服务器。

如果配置了缓冲的日志，服务器会为对应的日志创建缓冲区，并尽可能在缓冲区中保存消息。服务器仅在出现以下条件之一时清除日志消息到日志文件：

- 缓冲区已满。当缓冲区大小等于或大于 bufferSize 配置参数指定的值时，缓冲已满。此参数的默认值为 512 KB。
- 达到缓冲区的清除间隔。当最后一次缓冲区 flush 等于或大于 flushInterval 配置参数指定的值时，会达到 flush 间隔。此参数的默认值为 5 秒。
- 从控制台读取当前日志时。服务器在查询当前日志时检索最新的日志。

如果为未缓冲的日志配置了服务器，服务器会在生成消息时清除消息。由于服务器每次生成消息时都会执行 I/O 操作（写入日志文件），因此为未缓冲的日志配置服务器会降低性能。

设置日志参数是在 Red Hat Certificate System Administration Guide 中的 Configuring Logs in the Console 部分。

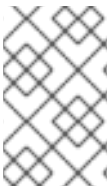
18.1.4. 日志文件轮转

子系统日志有一个可选的日志设置，允许轮转和启动新日志文件，而不是无限期地增加日志文件。当出现以下情况之一时，会轮转日志文件：

- 已达到对应文件的大小限制。对应日志文件的大小等于或大于 maxFileSize 配置参数指定的值。此参数的默认值为 100 KB。
- 达到对应文件的年龄限制。对应的日志文件等于或早于 rolloverInterval 配置参数指定的时间间隔。此参数的默认值为 2592000 秒（每三十天）。

轮转日志文件时，使用附加时间戳的文件名命名旧文件。附加的时间戳是一个整数，指示对应的活跃日志文件轮转的日期和时间。日期和时间格式为 YYYYMMDD （年、月份、天）和 HHMMSS （小时、分钟、秒）。

日志文件，特别是审计日志文件，包含重要信息。这些文件应定期归档到一些备份介质，方法是将整个 日志 目录复制到存档介质中。



注意

证书系统不提供归档日志文件的任何工具或实用程序。

证书系统提供了一个命令行实用程序 signtool，在将日志文件归档为篡改检测之前为日志文件签名。

签名日志文件是签名的审计日志功能的替代选择。签名的审计日志会创建使用子系统签名证书的审计日志。

轮转的日志文件不会被删除。

18.2. 操作系统(RHCS 的外部)日志设置

18.2.1. 启用操作系统级别的审计日志

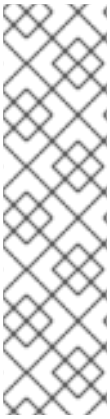
警告

以下部分中的所有操作都必须通过 sudo 以 root 或特权用户身份执行。

auditd 日志记录框架提供了许多额外的审计功能。这些操作系统级别的审计日志补充功能由证书系统直接提供。在执行本节中的任何以下步骤前，请确保安装了 audit 软件包：

```
# sudo yum install audit
```

系统软件包更新（使用 yum 和 rpm 包含证书系统）的审计会自动执行，不需要额外的配置。



注意

添加每个审计规则并重启 auditd 服务后，运行以下命令来验证新规则是否已添加：

```
# auditctl -l
```

新规则的内容应在输出中可见。

有关查看生成的审计日志的说明，请参阅 Red Hat Certificate System Administration Guide中的 Displaying Operating System-level Audit logs 部分。

18.2.1.1. 审计证书系统审计日志删除

要在删除审计日志时接收审计事件，您需要审核目标为证书系统日志的系统调用。

使用以下内容创建文件 /etc/audit/rules.d/rhcs-audit-log-deletion.rules ：

```
-a always,exit -F arch=b32 -S unlink -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S rename -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S rmdir -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S unlinkat -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b32 -S renameat -F dir=/var/log/pki -F key=rhcs_audit_deletion
```

```
-a always,exit -F arch=b64 -S unlink -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S rename -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S rmdir -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S unlinkat -F dir=/var/log/pki -F key=rhcs_audit_deletion
-a always,exit -F arch=b64 -S renameat -F dir=/var/log/pki -F key=rhcs_audit_deletion
```

然后重启 auditd ：

```
# service auditd restart
```

18.2.1.2. 审计未授权证书系统使用 Secret 密钥

要接收对证书系统 Secret 或私钥的所有访问权限的审计事件，您需要审核文件系统对 NSS 数据库的访问。

使用以下内容创建 /etc/audit/rules.d/rhcs-audit-nssdb-access.rules 文件：

```
-w /etc/pki/<instance name>/alias -p warx -k rhcs_audit_nssdb
```

<instance name> 是当前实例的名称。对于 /etc/pki/<instance name>/alias 中的每个文件('<file>')，添加到 /etc/audit/rules.d/rhcs-audit-nssdb-access.rules 中，行：

```
-w /etc/pki/<instance name>/alias/<file> -p warx -k rhcs_audit_nssdb
```

例如，如果实例名称是 pki-ca121318ec 和 cert9.db,key4.db,NHSM-CONN-XCcert9.db,NHSM-CONN-XCkey4.db, 和 pkcs11.txt 是文件，则配置文件将包含：

```
-w /etc/pki/pki-ca121318ec/alias -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/cert9.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/key4.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/NHSM-CONN-XCcert9.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/NHSM-CONN-XCkey4.db -p warx -k rhcs_audit_nssdb
-w /etc/pki/pki-ca121318ec/alias/pkcs11.txt -p warx -k rhcs_audit_nssdb
```

然后重启 auditd ：

```
# service auditd restart
```

18.2.1.3. 审计时间更改事件

要接收审计事件是否有时间变化，您需要审核可修改系统时间的系统调用访问。

使用以下内容创建 /etc/audit/rules.d/rhcs-audit-rhcs_audit_time_change.rules 文件：

```
-a always,exit -F arch=b32 -S adjtimex,settimeofday,stime -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S adjtimex,settimeofday -F key=rhcs_audit_time_change
-a always,exit -F arch=b32 -S clock_settime -F a0=0x0 -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S clock_settime -F a0=0x0 -F key=rhcs_audit_time_change
-a always,exit -F arch=b32 -S clock_adjtime -F key=rhcs_audit_time_change
-a always,exit -F arch=b64 -S clock_adjtime -F key=rhcs_audit_time_change
-w /etc/localtime -p wa -k rhcs_audit_time_change
```

然后重启 auditd ：

```
# service auditd restart
```

有关如何设置时间的说明，请参阅 Red Hat Certificate System Administration Guide 中的在 Red Hat Enterprise Linux 7 中设置时间和日期。

18.2.1.4. 审计对证书系统配置的访问

要接收对证书系统实例配置文件的所有修改的审计事件，请审核这些文件的文件系统访问。

使用以下内容创建 /etc/audit/rules.d/rhcs-audit-config-access.rules 文件：

```
-w /etc/pki/instance_name/server.xml -p wax -k rhcs_audit_config
```

另外，为 /etc/pki/instance_name/ 目录中为每个子系统添加以下内容：

```
-w /etc/pki/instance_name/subsystem/CS.cfg -p wax -k rhcs_audit_config
```

例 18.1. RHCS-audit-config-access.rules 配置文件

例如，如果实例名称是 pki-ca121318ec 且只安装了 CA，则 /etc/audit/rules.d/rhcs-audit-config-access.rules 文件将包含：

```
-w /etc/pki/pki-ca121318ec/server.xml -p wax -k rhcs_audit_config
-w /etc/pki/pki-ca121318ec/ca/CS.cfg -p wax -k rhcs_audit_config
```

请注意，对 PKI NSS 数据库的访问已在 rhcs_audit_nssdb 下进行审核。

18.3. 在 CS.CFG 文件中配置日志

在安装配置中，可以通过直接编辑实例的 CS.cfg 来配置日志记录。

1. 停止子系统实例。

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

2. 打开 /var/lib/pki/instance_name/subsystem_type/conf 目录中的 CS.cfg 文件。

3. 创建新日志。

4. 要配置日志实例，请修改与该日志关联的参数。这些参数以 log.instance 开头。

表 18.3. 日志条目参数

参数	描述
type	日志文件的类型。**系统** 创建错误和系统日志；**事务** 记录审计日志。
enable	设置日志是否活跃。仅启用的日志实际上记录事件。
level	在文本字段中设置日志级别。必须在字段中手动输入该级别；没有选择菜单。日志级别设置是一个数字值，如 第 18.1.2 节 “日志级别(Message Categories)” 中列出的。
fileName	日志文件的完整路径，包括文件名。子系统用户应具有文件的读取/写入权限。
bufferSize	日志的缓冲大小(KB)。当缓冲区达到这个大小后，缓冲区的内容将清除并复制到日志文件中。默认大小为 512 KB。有关缓冲的日志的详情，请参考 第 18.1.3 节 “buffered 和 Unbuffered Logging”。
flushInterval	缓冲区内容被清除并添加到日志文件中的时间（以秒为单位）。默认间隔为 5 秒。

参数	描述
maxFileSize	日志文件的大小(KB)在轮转前可能会成为它。达到这个大小后，文件会被复制到轮转的文件中，日志文件将启动新文件。有关日志文件轮转的详情，请参考 第 18.1.4 节 “日志文件轮转”。默认大小为 2000 KB。
rolloverInterval	服务器轮转活跃日志文件的频率。可用的选项有每小时、每天、每周、每月和每年。默认选择是 monthly。如需更多信息，请参阅 第 18.1.4 节 “日志文件轮转”。
logSigning[a]	启用签名的日志记录。启用此参数后，为 **signedAuditCertNickname** 参数提供一个值。这个选项意味着只能由审核员查看日志。该值可以是 **true** 或 **false**。
signedAuditCertNickname[a]	用于为审计日志签名的证书的别名。此证书的私钥必须可供子系统访问，才能为日志签名。
Events[a]	指定将哪些事件记录到审计日志。日志事件用逗号分开，没有空格。
[a] 这只适用于审计日志。	

5. *保存该文件。*

6. *启动子系统实例。*

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

或（如果使用 nuxwdog watchdog）

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

18.3.1. 启用并配置签名的审计日志

18.3.1.1. 启用签名的审计日志记录

默认情况下，在安装时会启用审计日志记录。但是，需要在安装后手动启用日志签名。

显示当前审计日志记录配置：

```
# pki-server subsystem-audit-config-show
  Enabled: True
  Log File: audit_signing_log_file
  Buffer Size (bytes): 512
  Flush Interval (seconds): 5
  Max File Size (bytes): 2000
  Rollover Interval (seconds): 2592000
  Expiration Time (seconds): 0
  Log Signing: False
  Signing Certificate: audit_signing_certificate
```

启用签名的审计日志记录：

1. *使用 pki-server 实用程序将 --logSigning 选项设置为 true ：*

```
# pki-server subsystem-audit-config-mod --logSigning True
  ...
  Log Signing: True
  ...
```

2. *重启实例：*

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

或（如果使用 nuxwdog watchdog）

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

18.3.1.2. 配置审计事件

18.3.1.2.1. 启用和禁用审计事件

有关启用和禁用审计事件的详情，请参考 Red Hat Certificate System Administration Guide 中的 Console 中的 Configuring a Signed Audit Log 部分。

此外，可以将审计事件过滤器设置为精细的选择。请参阅 第 18.3.1.2.2 节 “过滤审计事件”。

有关证书系统中可审核事件的完整列表，请参阅 Red Hat Certificate System Administration Guide 中的 审计事件 附录。

18.3.1.2.2. 过滤审计事件

在证书系统管理员中，可以设置过滤器来配置根据事件属性将哪些审计事件记录在审计文件中。

过滤器的格式与 LDAP 过滤器的格式相同。但是，证书系统只支持以下过滤器：

表 18.4. 支持的审计事件过滤器

类型	格式	示例
存在	(*attribute*=*)	(ReqID=*)
ç›¸ç‰	(*attribute=value*)	(outcome=Failure)
子字符串	(*attribute=initial*any*...*any*final*)	(SubjectID=*admin*)
AND 操作	(&(*filter_1*)(*filter_2*)...(*filter_n*))	(& (SubjectID=admin) (Outcome=Failure))
OR 操作	(|(*filter_1*)(*filter_2*)...(*filter_n*))	(|(SubjectID=admin)(Outcome=Failure))
NOT 操作	(! (*filter*)	(!(SubjectID=admin))

有关 LDAP 过滤器的详情，请查看 **Red Hat Directory Server Administration Guide** 中的 **Using Compound Search Filters** 部分。

例 18.2. 过滤审计事件

显示配置集证书请求和处理的证书的当前设置：

```
$ pki-server ca-audit-event-show PROFILE_CERT_REQUEST
  Event Name: PROFILE_CERT_REQUEST
  Enabled: True
  Filter: None

$ pki-server ca-audit-event-show CERT_REQUEST_PROCESSED
  Event Name: CERT_REQUEST_PROCESSED
  Enabled: True
  Filter: None
```

要只记录配置文件证书请求和事件失败的事件，用于处理将 **InfoName** 字段设置为 **rejectReason** 或 **cancelReason** 的证书请求：

1. 执行以下命令：

```
$ pki-server ca-audit-event-update PROFILE_CERT_REQUEST --filter "
(Outcome=Failure)"
  ...
  Filter: (Outcome=Failure)

$ pki-server ca-audit-event-update CERT_REQUEST_PROCESSED --filter "(|
(InfoName=rejectReason)(InfoName=cancelReason))"
  ...
  Filter: (|(InfoName=rejectReason)(InfoName=cancelReason))
```

2. 重启证书系统：

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

或（如果使用 nuxwdog watchdog）

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

18.3.2. 配置自测试

在 CS.cfg 文件中注册并配置 self-tests 功能和单独的 self-tests。如果启用了自助测试，则会根据需要列出自助测试或启动，并且 为空或设置为 critical。

Critical 自我测试在自测试的名称后有一个冒号和单词 critical。否则，在这个位置都没有。当需要自我测试期间的关键自测试失败时，服务器将关闭；在启动关键自我测试期间，服务器不会启动。

安装实例时，实施的自测试会自动注册和配置。注册和配置的自我测试是与子系统类型关联的自我测试。

通过更改 CS.cfg 文件中的相应设置来更改自测试的关键性。

18.3.2.1. 启动的默认自测试

以下 self-tests 在启动时默认启用。

对于 CA 子系统，启动时默认启用以下自测试：

- CAPresence - 用于验证 CA 子系统是否存在。
- CAValidity - 用于确定 CA 子系统当前是否有效且未过期。这包括检查 CA 证书的过期时间。
- SystemCertsVerification - 用于确定系统证书当前有效且尚未过期或被撤销。对于 CA 子系统，只有每个证书的有效性测试才会进行，保留证书验证测试，这可能会导致 OCSP 请求。此行为可使用以下配置参数覆盖：

```
selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true
```

默认情况下，如果根本不存在，则此配置参数被视为 false。

对于 KRA 子系统，启用了以下self-tests：

- KRAPresence - 用于验证 KRA 子系统是否存在。
- KRAValidity - 用于确定 KRA 子系统当前是否有效且未过期。这包括检查 KRA 证书的过期时间。
- SystemCertsVerification - 用于确定系统证书当前有效且尚未过期或被撤销。

对于 OCSP 子系统，启用了以下自测试：

- OCSPPresence - 用于验证 OCSP 子系统是否存在。
- OCSPValidity - 用于确定 OCSP 子系统当前是否有效且未过期。这包括检查 OCSP 证书的过期。
- SystemCertsVerification - 用于确定系统证书当前有效且尚未过期或被撤销。对于 OCSP

子系统，只有每个证书的有效性测试才会进行，省略证书验证测试，这可能会导致 OCSP 请求。此行为可使用以下配置参数覆盖：

```
selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true
```

默认情况下，如果根本不存在，则此配置参数被视为 false。

对于 TKS 子系统，启用了以下self-tests：

- TKSKnownSessionKey - 用于验证 TKS 子系统的已知会话密钥。这会验证 TKS 能够代表 TPS 和支持的智能卡正确创建密钥。
- SystemCertsVerification - 用于确定系统证书当前有效且尚未过期或被撤销。

对于 TPS 子系统，启用了以下self-tests：

- TPSPresence - 用于验证 TPS 子系统是否存在。
- TPSValidity - 用于确定 TPS 子系统当前是否有效且未过期。这包括检查 TPS 证书的过期时间。
- SystemCertsVerification - 用于确定系统证书当前有效且尚未过期或被撤销。

18.3.2.2. 修改自测试配置

默认情况下，self-test 配置兼容。但是，某些设置可以更改自测试日志记录的可见性或提高性能。修改 self-tests 的配置设置：

1. 停止子系统实例。
2. 打开位于实例的 conf/ 目录中的 CS.cfg 文件。
3.

要编辑 self-test 日志的设置，请编辑以 selftests.container.logger 开头的条目。除非另有指定，否则这些参数不会影响合规性。这些参数包括以下参数：

- bufferSize - 指定日志的缓冲大小(KB)。默认大小为 512 KB。当缓冲区达到这个大小后，缓冲区的内容将清除并复制到日志文件中。
- enable - 指定要启用的 true。仅启用的日志实际上记录事件。必须启用 这个值才能合规。
- filename - 向写入消息的文件指定完整路径，包括文件名。服务器必须具有文件的读/写权限。默认情况下，self-test 日志文件为 /var/log/pki-name/logs/selftest.log
- flushInterval - 指定间隔（以秒为单位）以将缓冲区刷新到文件。默认间隔为 5 秒。flushInterval 是缓冲区内容被清除并添加到日志文件中的时间长度。
- level - 默认选择为 1；此日志没有为 1 以外的任何级别设置。
- maxFileSize - 以 KB 为单位指定错误日志的文件大小。默认大小为 100 KB。maxFileSize 决定日志文件轮转前的大小。达到此大小后，文件将复制到轮转的文件，并启动新的日志文件。
- rolloverInterval - 指定服务器轮转活跃错误日志文件的频率。选择是每小时、每天、每周、每月和年份。默认选择是 monthly。
- type - 设置为 transaction; 不要更改它。

4. 要编辑运行自测试的顺序，请通过列出任何 self-test 作为用逗号分开的以下参数值来指定顺序。

要标记 self-test critical，请将冒号和单词 critical 添加到列表中自助测试的名称。

要禁用 self-test，请将其移除为 selftests.container.order.onDemand 或 selftests.container.order.startup 参数的值。

5. 保存该文件。

6. 启动子系统。

18.3.3. Debug Log 的额外配置

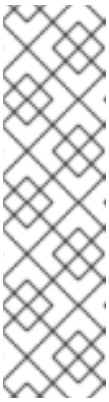
18.3.3.1. 启用和禁用调试日志记录

默认情况下，在证书系统中启用调试日志记录。然而，在某些情况下，管理员希望禁用或重新启用此功能：

1. 编辑 /var/lib/pki/instance_name/subsystem_type/conf/CS.cfg 文件并设置 debug.enabled 参数：

   - 要禁用调试日志记录，请设置：

     ```
     debug.enabled=false
     ```

     注意

     调试日志 不是 审计日志的一部分。当试图在证书系统或失败时调试特定故障时，调试日志非常有用。

     默认情况下启用调试日志。如果不需要，管理员可以安全地禁用调试日志记录来关闭详细程度。

   - 要启用调试日志记录，请设置：

     ```
     debug.enabled=true
     ```

2. 重启证书系统实例：

   ```
   # systemctl restart pki-tomcatd@instance-name.service
   ```

或（如果使用 nuxwdog watchdog）

```
# systemctl restart pki-tomcatd-nuxwdog@instance-name.service
```

18.3.3.2. 设置调试日志文件的轮转

证书系统无法轮转调试日志。默认启用调试日志记录，这些日志会增加，直到文件系统已满为止。使用 logrotate 等外部实用程序来轮转日志。

例 18.3. 使用 logrotate 轮转调试日志

创建包含以下内容的配置文件，如 /etc/logrotate.d/rhcs_debug ：

```
/var/log/pki/instance_name/subsystem/debug {
    copytruncate
    weekly
    rotate 5
    notifempty
    missingok
}
```

要在一个配置文件中轮转多个子系统的调试日志，请在打开大括号前列出日志的路径（以空格分开）。例如：

```
/var/log/pki/instance_name/ca/debug /var/log/pki/instance_name/kra/debug {
    ...
}
```

有关 logrotate 和示例中使用的参数的详情，请查看 logrotate(8) man page。

18.4. 审计保留

需要根据保留类别以某种方式保留审计数据：

- 扩展审计保留：为证书的生命周期保留所需的维护的审计数据（来自其过期或撤销日期）。在证书系统中，它们出现在以下区域：
  - 签名的审计日志 ：在附录 E 中定义的所有事件。Red Hat Certificate System

Administration Guide 的审计事件。

- 在 CA 内部 LDAP 服务器中，CA 接收的证书请求记录和请求被批准的证书记录。

- 普通的审计保留：通常只保留的审计数据来支持正常操作。这包括不在 扩展的审计保留类别下的所有事件。

注意

证书系统不提供任何用于修改或删除审计数据的接口。

18.4.1. 审计数据的位置

本节介绍证书系统存储审计数据的位置，以及在哪里查找扮演重要角色的过期日期，以确定保留类别。

18.4.1.1. 审计日志的位置

证书系统将审计日志存储在 /var/log/pki-name/logs/signedAudit/ 目录中。例如，CA 的审计日志存储在 /var/lib/pki/instance_name/ca/logs/signedAudit/ 目录中。普通用户无法访问此目录中的文件。请参阅

有关需要遵循扩展审计保留周期的审计日志事件列表，请参阅 Red Hat Certificate System Administration Guide 中的 审计事件 附录。

重要

不要删除包含"Extended Audit Events"附录
。

这些审计日志可能会消耗到磁盘分区中所有可用空间的存储空间。

18.4.1.2. 证书请求和证书记录的位置

提交证书签名请求(CSR)时，CA 会将 CSR 存储在 CA 的内部目录服务器提供的请求存储库中。批准

这些请求后，每个证书都会成功发布，将导致同一内部目录服务器在证书存储库中创建 LDAP 记录。

当使用 pkispawn 工具创建 CA 时，在以下参数中指定 CA 的内部目录服务器：

- pki_ds_hostname
- pki_ds_ldap_port
- pki_ds_database
- pki_ds_base_dn

如果证书请求被成功批准，可以通过访问 CA EE 门户或序列号来查看证书的有效性。

显示证书请求记录的有效性：

1. 在 https://host_name 下登录 CA EE 门户：端口/ca/ee/ca/。
2. 单击 Check Request Status。
3. 输入 Request Identifier。
4. 单击 Issued Certificate。
5. 搜索 Validity。

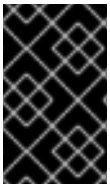
显示证书记录的有效性：

1. 在 https://host_name 下登录 CA EE 门户：端口/ca/ee/ca/。

2. 输入序列号范围。如果您搜索了一个特定记录，请在最低和最高的序列号字段中输入记录的序列号。

3. 点搜索结果。

4. 搜索 Validity。



重要

不要删除尚未过期的证书的证书记录请求。

第 19 章 创建角色用户

如 第 2.6.6.1 节 “默认管理角色” 所述，在安装过程中创建了一个 bootstrap 用户。安装后，创建真实用户并为其分配正确的系统权限。为满足合规性，每个用户都必须仅是一个角色(group)的成员。

本章介绍了如何：

- 在操作系统中创建证书系统管理用户
- 在证书系统中创建 PKI 角色

19.1. 在操作系统上创建 PKI 管理用户

本节适用于管理角色用户。代理和审核员角色用户，请参阅 第 19.2 节 “在证书系统中创建 PKI 角色用户”。

通常，证书系统中的管理员、代理和审核员可以使用客户端应用程序（如命令行实用程序、Java 控制台和浏览器）远程管理证书系统实例。对于大多数 CS 管理任务，证书系统角色用户不需要登录实例运行的主机。例如，允许审核员角色用户远程检索签名的审计日志进行验证，代理角色用户可以使用代理接口批准证书颁发，而管理员角色用户可以使用命令行实用程序来配置配置集。

然而，在某些情况下，证书系统管理员需要登录主机系统直接修改配置文件，或者启动或停止证书系统实例。因此，在操作系统中，管理员角色用户应该是允许更改配置文件的人员，并读取与 Red Hat Certificate System 相关的各种日志。

注意

不要允许证书系统管理员或审核员以外的任何人访问审计日志文件。

1. 在操作系统上创建 pkiadmin 组。

```
# groupadd -r pkiadmin
```

2. 将 pkiuser 添加到 pkiadmin 组中：

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

3. 在操作系统上创建用户。例如，要创建 jsmith 帐户：

```
# useradd -g pkiadmin -d /home/jsmith -s /bin/bash -c "Red Hat Certificate System
Administrator John Smith" -m jsmith
```

详情请查看 useradd(8) man page。

4. 将用户 jsmith 添加到 pkiadmin 组中：

```
# usermod -a -G pkiadmin jsmith
```

详情请查看 usermod(8) man page。

如果您使用 nCipher 硬件安全模块(HSM)，请将管理 HSM 设备的用户添加到 nfast 组中：

```
# usermod -a -G nfast pkiuser
# usermod -a -G nfast jsmith
```

5. 添加正确的 sudo 规则，以允许 pkiadmin 组到证书系统和其他系统服务。

为了简单起见，可以配置证书系统和目录服务器进程，以便 PKI 管理员（而不是只有 root）可以启动和停止服务。

设置子系统时推荐的选项是使用 pkiadmin 系统组。（详细信息为 第 6.9 节 “证书系统用户和组”）。然后，所有将成为证书系统管理员的操作系统用户都会被添加到此组中。如果存在这个 pkiadmin 系统组，则可以授予 sudo 访问权限来执行某些任务。

1. 编辑 /etc/sudoers 文件；在 Red Hat Enterprise Linux 中，可以使用 visudo 命令完成：

```
# visudo
```

2. 根据机器上安装的内容，为目录服务器、管理服务器、PKI 管理工具和每个 PKI 子系统

实例添加一行，为 pkiadmin 组授予 sudo 权限：

```
# For Directory Server services
%pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv.target
%pkiadmin ALL = PASSWD: /usr/bin/systemctl * dirsrv-admin.service

# For PKI instance management
%pkiadmin ALL = PASSWD: /usr/sbin/pkispawn *
%pkiadmin ALL = PASSWD: /usr/sbin/pkidestroy *

# For PKI instance services
%pkiadmin ALL = PASSWD: /usr/bin/systemctl * pki-
tomcatd@instance_name.service
```

重要

确保为计算机上的每个证书系统、目录服务器和管理服务器设置 sudo 权限，并且仅对 计算机上的这些实例设置 sudo 权限。机器上可能有多个相同子系统类型的实例，或者没有子系统类型的实例。它取决于部署。

6. 将以下文件中的组设置为 pkiadmin ：

```
# chgrp pkiadmin /etc/pki/instance_name/server.xml
# chgrp -R pkiadmin /etc/pki/instance_name/alias
# chgrp pkiadmin /etc/pki/instance_name/subsystem/CS.cfg
# chgrp pkiadmin /var/log/pki/instance_name/subsystem/debug
```

在操作系统中创建管理用户后，请遵循 第 19.2 节 “在证书系统中创建 PKI 角色用户”。

19.2. 在证书系统中创建 PKI 角色用户

要创建 PKI 角色用户，请参阅 Red Hat Certificate System Administration Guide中的 管理证书系统用户和组 部分。

第 20 章 删除 BOOTSTRAP 用户

重要

在删除 bootstrap 用户前，创建一个实际的 PKI 管理用户，如 第 19 章 创建角色用户 所述。

要删除 bootstrap 用户，请按照 Red Hat Certificate System Administration Guide中的 删除证书系统用户 一节中所述的步骤进行操作。

20.1. 禁用多角色支持

默认情况下，用户可以同时属于多个子系统组，允许用户充当多个角色。例如，John Smith 可以同时属于代理和管理员组。但是，对于高度安全的环境，应该限制子系统角色，以便用户只能属于一个角色。这可以通过在实例的配置中禁用 multirole 属性来完成。

对于所有子系统：

1. 停止服务器：

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

或（如果使用 nuxwdog watchdog）

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

2. 打开 CS.cfg 文件：

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

3. 将 multiroles.enable 参数值从 true 更改为 false。

4. 在证书系统中添加或编辑受多角色设置影响的默认角色列表。如果禁用了多角色，并且用户

属于 multiroles.false.groupEnforceList 参数中列出的 角色之一，则无法将该用户添加到列表中任何其他角色的任何组中。

```
multiroles.false.groupEnforceList=Administrators,Auditors,Trusted Managers,Certificate Manager Agents,Registration Manager Agents,Key Recovery Authority Agents,Online Certificate Status Manager Agents,Token Key Service Manager Agents,Enterprise CA Administrators,Enterprise KRA Adminstrators,Enterprise OCSP Administrators,Enterprise TKS Administrators,Enterprise TPS Administrators,Security Domain Administrators,Subsystem Group
```

5. ***重启服务器：***

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

或（如果使用 nuxwdog watchdog）

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

部分 IV. 升级到证书系统 10.X

第 21 章 从证书系统 9 升级到证书系统 10

建议用户升级到 Red Hat Certificate System 的最新版本，以通过 Errata-Support 频道获得安全和程序错误修复。目前，最新版本在 RHEL 7.9 和 RHCS 10.4 上是 RHCS 9.7（在 RHEL 8.6 上）。

21.1. 迁移 CA

使用现有证书文件迁移证书系统需要以下步骤：

1. 第 21.1.1 节 “从以前的系统中导出数据”
2. 第 21.1.2 节 “验证 PKCS12 文件”
3. 第 21.1.3 节 “在新主机上设置 CA”
4. 第 21.1.4 节 “将旧数据导入到新 CA”
5. 第 21.1.5 节 “将用户重新分配给默认组”

21.1.1. 从以前的系统中导出数据

在设置新的证书系统实例前，使用以下步骤导出当前证书颁发机构(CA)的数据。在本例中，CA 的实例名称为 < pki-rootCA& gt;，位于 /var/lib/pki/<instance_name >。

1. 导出 CA 签名证书和密钥。

```
# grep internal= /var/lib/pki/<instance_name>/conf/password.conf | awk -F= '{print $2;}' >
internal.txt
# echo <Secret.123> > password.txt
# PKCS12Export -d /var/lib/pki/<instance_name>/alias -p internal.txt -o ca.p12 -w
password.txt
```

2. 导出证书签名请求(CSR)。

```
# echo "-----BEGIN NEW CERTIFICATE REQUEST-----" > ca_signing.csr
# sed -n "/^ca.signing.certreq=/ s/^[^=]*=// p" < /var/lib/pki/<instance_name>/ca/conf/CS.cfg
>> ca_signing.csr
# echo "-----END NEW CERTIFICATE REQUEST-----" >> ca_signing.csr
```

3. 如果旧 CA 是中间 CA （链中有一个 root CA），请从 NSS 数据库中提取 root CA：

```
# certutil -L -d /var/lib/pki/<instance_name>/alias/ -n root_CA_nickname -a >
ca_rootca_signing.crt
```

4. 备份内部 LDAP 数据库。

- 通过检查 CA 的 CS.cfg 中的 internaldb.database 的值来查找 CA 数据库的名称：

```
# grep internaldb.database /etc/pki/<instance_name>/ca/CS.cfg
internaldb.database=<CS_database_name>
```

- 使用在实例创建时由 setup-ds.pl 生成的特定于实例的脚本，将此数据库导出到 .ldif 文件：

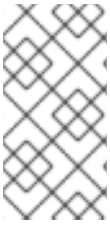
```
# cd /usr/lib64/dirsrv/<instance_name>
# ./db2ldif -n "<CS_database_name>" -a /tmp/old_ca.ldif
```

db2ldif 命令以 DB 用户身份运行，因此它需要具有写入权限的目标文件夹。

5. 将 old_ca.ldif,ca.p 12,ca_signing.csr 文件传送到新的 CA 机器。如果正在迁移的 CA 也是中间 CA，还要传输 ca_rootca_signing.crt。

21.1.2. 验证 PKCS12 文件

在新 CA 上，新数据库实例在标准端口(389)上运行。PKCS12 文件包含旧系统的所有系统证书。验证该文件是否包含 CA 签名证书和密钥。



注意

在 FIPS 模式中，您必须使用 NSS 数据库运行 pki pkcs12 命令。如需更多信息，请参阅 pki-pkcs12 CLI。

1. 查找 CA 签名证书和密钥。

```
$ echo "<Secret.123>" > password.txt
$ pki pkcs12-cert-find --pkcs12-file ca.p12 --pkcs12-password-file password.txt
$ pki pkcs12-key-find --pkcs12-file ca.p12 --pkcs12-password-file password.txt
```

2. 验证是否存在 CA 签名证书的信任标记。如果标志不是 "CTu,Cu" 或缺少它们，请添加它们。

```
$ pki pkcs12-cert-mod "<caSigningCert cert-pki-rootCA>" \ --pkcs12-file ca.p12 --pkcs12-password-file password.txt \ --trust-flags "CTu,Cu,Cu"
```

3. 删除其他证书和密钥。

```
$ pki pkcs12-cert-del "<Server-Cert cert-pki-rootCA>" --pkcs12-file ca.p12 --pkcs12-password-file password.txt
$ pki pkcs12-cert-del "<subsystemCert cert-pki-rootCA>" --pkcs12-file ca.p12 --pkcs12-password-file password.txt
$ pki pkcs12-cert-del "<ocspSigningCert cert-pki-rootCA>" --pkcs12-file ca.p12 --pkcs12-password-file password.txt
$ pki pkcs12-cert-del "<auditSigningCert cert-pki-rootCA>" --pkcs12-file ca.p12 --pkcs12-password-file password.txt
```

4. 如果正在迁移的 CA 也是中间 CA，请从 PKCS criu 文件中删除 root CA 证书。

```
$ pki pkcs12-cert-del "<Top-level Root CA Signing Certificate>" --pkcs12-file ca.p12 --pkcs12-password-file password.txt
```

21.1.3. 在新主机上设置 CA

从现有实例导出数据后，在新主机上创建和设置证书颁发机构(CA)。

1. 使用旧 CA 的参数，为 pkispawn 创建配置文件，如 CA.cfg。此实例在标准端口(389)上运行。

```
[DEFAULT]
pki_instance_name=<new_instance_name>
pki_admin_password=<caadmin_password>
pki_client_pkcs12_password=<pkcs12_file_password>
pki_ds_password=<DS_password>
pki_ds_ldap_port=389
```

```
pki_existing=True

[CA]
pki_ca_signing_csr_path=<path/to/ca_signing.csr>
pki_ca_signing_cert_path=<path/to/ca_signing.crt>
pki_ca_signing_nickname=<caSigningCert cert-nickname>
pki_ds_base_dn=<o=pki-tomcat-CA>
pki_ds_database=<instance_name-CA>
pki_serial_number_range_start=<starting_number_for_cert_serial_numbers>
pki_request_number_range_start=<starting_number_for_request_IDs>
pki_master_crl_enable=False
```

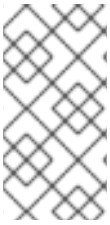
如果旧 CA 是中间 CA，请在配置文件中添加以下两行。它们对应于 root CA 证书的路径和在 NSS 数据库中存储证书时要使用的 nickname。

```
pki_cert_chain_path=<rootca_signing.crt>
pki_cert_chain_nickname=<caSigningCert cert-pki-ca>
```

有关详情和参数描述，请查看 pkispawn(8) man page。

2. **在新主机上运行 pkispawn 以创建新 CA 实例：**

```
# pkispawn -s CA -f ca.cfg -v
```



注意

有关如何为新 CA 设置新的 DS 实例的说明，请参阅安装 Red Hat Directory Server。

21.1.4. 将旧数据导入到新 CA

创建新 CA 实例后，将数据导入到新的 CA 数据库中：

1. **停止 CA 服务。**

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

2. **可选：备份新主机上的 CA 数据库。**

```
# dsctl -v idm-qe-01 db2bak
```

备份存储在 /var/lib/dirsrv/ <instance_name> /bak/ <host_name-time_stamp>/ 目录中。

3. 从新的 RHEL 8 内部数据库中删除签名证书条目。

```
# ldapdelete -x -w <password< -D 'cn=Directory Manager'
"cn=<serial_number<,ou=certificateRepository,ou=ca,o=pki-tomcat-CA"
```

该条目现在从旧数据导入，以便该条目中的 CRL 属性指向正确的条目。

4. 将数据导入到新数据库中：

```
# ldapmodify -x -W <password> -D 'cn=Directory Manager' -a -c -f <path/to/old_ca.ldif>
```

ldapmodify 工具只添加新条目，且不更新安装 CA 时创建的现有条目。

5. 可选：验证 import.log 文件中的输出。您可以搜索失败的操作，如 ldap_add: Invalid syntax (21)。

6. 删除旧的安全域的目录条目。

```
# ldapmodify -W <password> -x -D "cn=Directory Manager"
dn: cn=<server.example.com:9445>,cn=CAList,ou=Security Domain,<o=pki-tomcat-CA>
changetype: delete
```

7. 在 /etc/pki/ <instance_name> /ca/CS.cfg 文件中启用 ca. crl.MasterCRL.enable 参数，使 CA 充当证书撤销列表(CRL) master：

```
ca.crl.MasterCRL.enable=true
```

8. 重启 CA 服务：

```
# systemctl start pki-tomcatd@<instance_name>
```

21.1.5. 将用户重新分配给默认组

使用 ldapmodify 或 pki 工具手动将成员添加到默认组中。

1. 设置客户端：

```
# pki -c <password> client-init
Client initialized
```

```
# pk12util -i ~/.dogtag/<instance_name>/ca_admin_cert.p12 -d ~/.dogtag/nssdb/
Enter Password or Pin for "NSS Certificate DB":
Enter password for PKCS12 file:
pk12util: PKCS12 IMPORT SUCCESSFUL
...
```

2. 将用户帐户添加到 管理员和 安全域 管理员组中 ：

```
# pki -n "<PKI Administrator for example.com>" -c <password> \
user-membership-add <user_name> "Certificate Manager Agents"
```

RHCS 9 和 RHCS 10 中的默认(bootstrap)管理员用户具有相同的默认角色，即相同的组成员资格。但是，如果您有一个非默认的 admin 用户，或者在 RHCS 9 中自定义了 admin 角色，则这些更改可能无法正确迁移。在这种情况下，重新配置 RHCS 10 中的角色：

```
# pki -n "<PKI Administrator for example.com>" -c <password> \
user-membership-add <user> "Administrators"
# pki -n "<PKI Administrator for example.com>" -c <password> \
user-membership-add <user> "Security Domain Administrators"
```

如果要确保证书已被成功迁移，请通过运行 ca-cert-find 命令检查其在新主机上是否存在。

21.2. 迁移 KRA

简单 KRA 迁移需要以下步骤：

1. 第 21.2.1 节 “在新主机上设置 KRA”

2. 第 21.2.2 节 “从以前的系统导出内容”

3. 第 21.2.3 节 “将数据导入到新的 KRA 中”

4. 第 21.2.4 节 “从 KRA Agent Page 验证迁移的密钥是否存在”

位于 alpha.example.com 上的 KRA 包含数据，而位于 omega.example.com 上的 KRA 尚不存在。

Hostname	PKI KRA 版本
alpha.example.com	RHCS 9.7 上的 PKI KRA 10.5
omega.example.com	RHCS 10.4 上的 PKI KRA 10.13

21.2.1. 在新主机上设置 KRA

在 omega.example.com 上，以 root 用户身份：

1. 在 omega.example.com 上安装和配置一个新的 PKI 10.13 KRA。

2. 停止 KRA 实例。

```
# systemctl stop pki-tomcatd@<pki-kra>
```

3. 创建目录以导出所有必需的文件。

```
# mkdir -p /export/pki
```

4. 将 KRA 存储证书导出到文件。稍后您将需要 KRA 存储证书来重新加密 alpha 上 KRA 的解密的旧数据。在以下示例中，该文件名为 omega.crt ：

```
# cd /var/lib/pki/<pki-kra>/alias/
# pki-server cert-export kra_storage -i <pki-kra> --cert-file <omega.crt>
Certificate:
    Data:
```

```
        Version: 3 (0x2)
        Serial Number: 8 (0x8)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O = example.com Security Domain, OU = topology-02-CA, CN = CA Signing
Certificate
        Validity
            Not Before: Dec 19 10:58:02 2019 GMT
            Not After : Dec  8 10:58:02 2021 GMT
        Subject: O = example.com Security Domain, OU = topology-02-KRA, CN = DRM
Storage Certificate
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:99:c1:f6:f4:0d:75:67:ff:58:3a:28:ee:34:f1:
                    40:0a:e1:5b:f3:9d:f4:c2:5a:1e:d0:d5:0c:62:c1:
```

5. **将 omega.crt 文件移到 /export/pki 目录中。**

```
# mv omega.crt /export/pki/
```

6. **停止 Directory 服务器。**

```
# systemctl stop dirsrv@omega
```

注意

如果缺少 db2ldif，请安装 389-ds-base-legacy-tools 软件包。

7. **提取 KRA LDAP 数据库配置。您需要新的 KRA LDAP 数据库配置，才能将旧数据转换为新数据。**

```
# /usr/lib64/dirsrv/slapd-omega/db2ldif -n <pki-kra-KRA> -a /tmp/omega.ldif
```

8. **将 /tmp/omega.ldif 文件移到 /export/pki 目录。**

```
# mv /tmp/omega.ldif /export/pki/
```

21.2.2. 从以前的系统导出内容

在 alpha.example.com 上，以 root 用户身份：

1. **创建通用目录：**

```
# mkdir -p /export/pki
```

2. **停止 Directory 服务器。在本例中，服务器名为 alpha:**

```
# systemctl stop dirsrv@alpha
```

3. **从 KRA LDAP 数据库生成 LDIF：**

```
# /usr/lib64/dirsrv/slapd-alpha/db2ldif -n <pki-kra-KRA> -a /tmp/alpha.ldif
```

4. **将 /tmp/alpha.ldif 文件移到 /export/pki 目录中：**

```
# mv /tmp/alpha.ldif /export/pki/
```

5. **将 KRA NSS 安全数据库复制到数据区域：**

```
# cp -p /var/lib/pki/<pki-kra>/alias/* /export/pki/
```

6. **进入 /export/pki 目录：**

```
# cd /export/pki
```

7. **从 omega.example.com 获取包含新 KRA 存储证书的 flat-file：**

```
sftp root@omega.example.com
sftp> cd /export/pki
sftp> get omega.crt
sftp> quit
```

8. **获取内部密码：**

```
# cat /var/lib/<instance_name>/conf/password.conf
```

9. **在 alpha.example.com 上运行 KRATool ：**

```
# KRATool \
   -kratool_config_file /usr/share/pki/java-tools/KRATool.cfg \
   -source_ldif_file /export/pki/alpha.ldif \
   -target_ldif_file /export/pki/alpha2omega.ldif \
   -log_file /tmp/KRATool_26_05_2023.log \
   -source_pki_security_database_path /export/pki/ \
   -source_storage_token_name "Internal Key Storage Token" \
   -source_storage_certificate_nickname "<storageCert cert-pki-tomcat KRA>" \
   -target_storage_certificate_file /export/pki/omega.crt \
   -source_kra_naming_context alpha.example.com \
   -target_kra_naming_context omega.example.com \
   -unwrap_algorithm AES \
   -process_requests_and_key_records_only

PROCESSING KRATOOL CONFIG FILE: ..... FINISHED.
SUCCESSFULLY processed kratool config file!
Initializing source PKI security databases in '/export/pki/'.
Retrieving token from CryptoManager.
Retrieving source storage token called 'Internal Key Storage Token'.
Retrieving source storage cert with nickname of 'storageCert cert-pki-tomcat KRA'.

BEGIN: Obtaining the private key from the source storage token . . .
Enter password for Internal Key Storage Token
************

FINISHED: Obtaining the private key from the source storage token.
BEGIN: Obtaining the public key from the target storage certificate . . .
FINISHED: Obtaining the public key from the target storage certificate.
PROCESSING: xxxxxxxxxxxxxxxxxxxxxxxxx.....
SUCCESSFULLY converted source LDIF file --> target LDIF file!

FINISHED "KRATool -kratool_config_file /usr/share/pki/java-tools/KRATool.cfg -
source_ldif_file /export/pki/alpha.ldif -target_ldif_file /export/pki/alpha2omega.ldif -log_file
/tmp/DRMTool_20_05_2021.log -source_pki_security_database_path /export/pki/ -
source_storage_token_name 'Internal Key Storage Token' -
source_storage_certificate_nickname 'storageCert cert-pki-tomcat KRA' -
target_storage_certificate_file /export/pki/omega.crt -source_pki_security_database_pwdfile
'/export/pki/password.cfg' -source_kra_naming_context 'alpha.example.com' -
target_kra_naming_context 'omega.example.com' -
process_requests_and_key_records_only"
```

注意

或者，您可以创建一个受未授权访问的纯文本文件，其中包含一个被证书或证书数据库自动访问的密码。使用 -source_pki_security_database_pwdfile < path_to_PKI_password_file> 命令行选项将此文件添加到 KRATool 中。

10. ***将 alpha2omega.ldif 文件复制到 omega.example.com ：***

```
sftp root@omega.example.com
sftp> cd /export/pki
sftp> put alpha2omega.ldif
sftp> quit
```

21.2.3. 将数据导入到新的 KRA 中

在 omega.example.com 上，以 root 用户身份：

1. *进入 /export/pki 目录*

```
# cd /export/pki
```

2. *连接 ldif 文件：*

```
# cat omega.ldif alpha2omega.ldif > omega_alpha.ldif
```

3. ***将 omega_alpha.ldif 文件导入到与 PKI KRA 关联的 LDAP 数据库中：***

```
# /usr/lib64/dirsrv/slapd-omega/ldif2db -n <pki-kra-KRA> -i /export/pki/omega_alpha.ldif
```

4. *启动 Directory 服务器：*

```
# systemctl start dirsrv@omega
```

5. *启动 KRA 实例。*

```
# systemctl start pki-tomcatd@<pki-kra>
```

21.2.4. 从 KRA Agent Page 验证迁移的密钥是否存在

最后一步是执行从 KRA Agent 页面迁移的密钥恢复。

```
[root@pki1 pki]# pki -d /root/nssdb/ -p 21080 -n '<PKI Administrator - example.com Security Domain>'
kra-key-find
Enter password for Internal Key Storage Token

----------------
3 key(s) matched
----------------
  Key ID: 0x1
  Algorithm: 1.2.840.113549.1.1.1
  Size: 1024
  Owner: UID=alpha1

  Key ID: 0x2
  Algorithm: 1.2.840.113549.1.1.1
  Size: 1024
  Owner: UID=alpha2

  Key ID: 0x3
  Algorithm: 1.2.840.113549.1.1.1
  Size: 1024
  Owner: UID=alpha3
----------------------------
Number of entries returned 3
----------------------------
```

21.2.5. 升级后测试

在升级前跟踪用户密钥的关键恢复操作。

过程 21.1. 密钥恢复测试

1. *显示升级前创建的 base64 用户证书。*

```
# <pki -n 'PKI Administrator - example.com>' -c <Secret.123> ca-cert-export <0xd>
  Serial Number: 0xd
  Subject DN: UID=alpha
  Issuer DN: CN=CA Signing Certificate,OU=pki-tomcat,O=example.com Security Domain
  Status: VALID
  Not Valid Before: Wed Jun 07 01:49:07 EDT 2023
  Not Valid After: Mon Dec 04 01:49:07 EST 2023

----BEGIN CERTIFICATE----
MIIDODCCAiCgAwIBAgIBDTANBgkqhkiG9w0BAQsFADBtMTUwMwYDVQQKDCxpZG1xZS5
sYWIuZW5n
LmJvcy5yZWRoYXQuY29tIFNlY3VyaXR5IERvbWFpbjETMBEGA1UECwwKcGtpLXRvbWNhd
DEfMB0G
A1UEAwwWQ0EgU2lnbmluZyBDZXJ0aWZpY2F0ZTAeFw0yMzA2MDcwNTQ5MDdaFw0yMz
```

```
EyMDQwNjQ5

        [...output truncated...]

EJyoMFM+RaAcTh+C3S0JZEoKlAS3UlJOMxk3BFZdWpv7ia+1faV6LFPZSCZ/m8i2c3KZNxF
W2xv1
DTIIVc7a1uEDApVDHf5aFcm0nGpEVeK+yvP4r1eD
----END CERTIFICATE----
```

2. ***使用此证书通过 KRA Agent UI 生成密钥恢复请求。***

3. ***批准恢复请求。***

```
# pki -c <Secret.123> -n '<PKI Administrator - example.com Security Domain>' -p 8443 -P
https kra-key-request-review <0x2> --action approve
-----
Result
------
Request ID: 0x2
Type: recovery
Status: approved
```

4. ***从 KRA UI 下载密钥。***

第 22 章 将证书系统 10 升级到最新的次版本

要将 Red Hat Certificate System 10.0 升级到最新的次版本，如 10.1，您需要更新软件包和配置文件。

1. 升级服务器中的所有软件包：

```
# yum update
```

在更新过程中，证书系统配置文件（如 /etc/pki/ <instance_name> /subsystem/CS.cfg 和 /etc/pki/<instance_name> /server.xml ）会自动修改到新版本。

在证书系统升级过程中生成的日志文件是：

- /var/log/pki/pki-server-upgrade-version.log
- /var/log/pki/pki-upgrade-version.log

请注意，日志文件名称中的版本号代表 pki* 软件包版本，而不是证书系统版本。

注意

如果在其他主机上安装目录服务器，还要更新该主机上的所有软件包。有关更新目录服务器的详情，请参考：

- Red Hat Directory Server 安装 G 中的对应章节。
- Red Hat Directory Server 发行注记，用于目录服务器中的显著变化、错误修复和已知问题。

2. 重启证书系统实例：

```
# systemctl restart pki-tomcatd@<instance_name>.service
```

注意

要从较低主版本（如 9.7）迁移到证书系统 10，请参阅 第 21 章 从证书系统 9 升级到证书系统 10。

部分 V. 卸载证书系统子系统

可以移除各个子系统，或者卸载与整个子系统关联的所有软件包。子系统会单独安装和卸载。例如，可以在离开安装和配置的 CA 子系统时卸载 KRA 子系统。也可以删除单个 CA 子系统，同时保留计算机上的其他 CA 子系统。

第 23 章 删除子系统

删除子系统需要指定子系统类型和运行子系统的服务器名称。此命令会删除与子系统关联的所有文件（不会删除子系统软件包）。

```
pkidestroy -s subsystem_type -i instance_name
```

s 选项指定要删除的子系统（如 CA、KRA、OCSP、TKS 或 TPS）。i 选项指定实例名称，如 pki-tomcat。

例 23.1. 删除 CA 子系统

```
$ pkidestroy -s CA -i pki-tomcat
Loading deployment configuration from /var/lib/pki/pki-tomcat/ca/registry/ca/deployment.cfg.
Uninstalling CA from /var/lib/pki/pki-tomcat.
Removed symlink /etc/systemd/system/multi-user.target.wants/pki-tomcatd.target.

Uninstallation complete.
```

pkidestroy 工具移除子系统以及任何相关的文件，如证书数据库、证书、密钥和相关用户。它不会卸载子系统软件包。如果子系统是服务器实例上的最后一个子系统，则也会删除服务器实例。

第 24 章 删除证书系统子系统软件包

许多与子系统相关的软件包和依赖项与 Red Hat Certificate System 一起安装，它们列在 第 7.2 节 “证书系统软件包” 中。删除子系统仅移除与该特定子系统关联的文件和目录。它不会删除该实例使用的实际安装软件包。完全卸载 Red Hat Certificate System 或其子系统需要使用软件包管理工具（如 yum ）单独删除每个软件包。

卸载单个证书系统子系统软件包：

1. 删除所有关联的子系统。例如：

```
pkidestroy -s CA -i pki-tomcat
```

2. 运行卸载工具。例如：

```
yum remove pki-subsystem_type
```

子系统类型可以是 ca、kra、ocsp、tks 或 tps。

3. 要删除其他软件包和依赖项，请使用 yum 删除软件包。安装的软件包的完整列表位于 第 7.2 节 “证书系统软件包”。

术语表

一个

Administrator

安装和配置一个或多个证书系统管理器并为它们设置特权用户或代理的人员。另请参阅 agent。

agent

属于组的用户，授权为证书系统管理器管理 代理服务。另请参阅 证书管理器代理、密钥恢复授权代理。

APDU

应用程序协议数据单元。用于在智能卡和智能卡读卡器之间的通信的通信的通信单元（大概为字

节）。

代理批准的注册

在签发证书前，注册需要代理批准请求。

代理服务

1.可以通过由证书系统 agent 通过为代理分配所需权限的证书系统子系统提供的 HTML 页面管理的服务。

2.用于管理此类服务的 HTML 页面。

审核员

查看已签名的审计日志的特权用户。

审计日志

记录各种系统事件的日志。此日志可以签名，提供未篡改的证明，并且只能由审核员用户读取。

属性值断言(AVA)

form 属性的断言 = value，其中 属性是 标签，如 o (organization)或 uid (user ID)， value 是 "Red Hat, Inc." 或登录名称。AVAs 被用来组成用来标识证书主题的 可区分名称(DN)，称为证书的 主题名称。

授权

访问由服务器控制的资源的权限。授权通常在服务器评估与资源关联的 ACL 后进行。请参阅 访问控制列表(ACL)。

自动注册

配置证书系统子系统的方法，以允许对最终用户注册进行自动身份验证，而无需人为干预。通过这种身份验证形式，成功完成身份验证模块处理的证书请求将成功批准进行配置文件处理和证书颁发。

访问控制

控制特定用户被允许执行的操作。例如，对服务器的访问控制通常基于由密码或证书建立的身份，以及有关该实体可以做什么的规则。另请参阅 访问控制列表(ACL)。

访问控制列表(ACL)

一组访问控制条目，用于定义在服务器收到访问特定资源时要评估的访问规则的层次结构。请参阅 *访问控制指令(ACI)*。

访问控制指令(ACI)

一个访问规则，用于指定请求访问的主题如何被标识或拒绝特定主题的权限。请参阅 *访问控制列表(ACL)*。

身份验证

自信识别；确保对某些计算机化交易的方不是一个障碍。身份验证通常涉及使用密码、证书、PIN 或其他信息来验证计算机网络上的身份。另请参阅 *基于密码的身份验证*、*基于证书的验证*、*客户端身份验证*、*服务器身份验证*。

身份验证模块

一组规则（作为 Java™ 类实施）来验证端点实体、代理、管理员或需要与证书系统子系统交互的任何其他实体。在典型的最终用户注册中，在用户提供注册表单请求的信息后，注册 servlet 会使用与该表单关联的身份验证模块来验证信息并验证用户的身份。请参阅 *servlet*。

高级加密标准(AES)

高级加密标准(AES)，如其前身数据加密标准(DES)，是一个 FIPS 批准的对称密钥加密标准。AES 由美国政府 2002 年采用。它定义了三个块密码，AES-128、AES-192 和 AES-256。美国国家标准与技术研究院(NIST)在美国.FIPS PUB 197.如需更多信息，请参阅 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf。

B

绑定 DN

用户 ID，采用可分辨名称(DN)的形式，用于向 Red Hat Directory Server 进行身份验证。

C

CA 层次结构

root CA 委派权限以将证书签发到从属 CA 的 CA 层次结构。从属 CA 还可以通过将发出状态委派给其他 CA 来扩展层次结构。另请参阅 *证书颁发机构(CA)*、*subordinate CA*、*root CA*。

CA 服务器密钥

提供 CA 服务的服务器的 SSL 服务器密钥。

CA 签名密钥

与 CA 证书中的公钥对应的私钥。CA 使用其签名密钥为证书签名和 CRL。

CA 证书

标识证书颁发机构的证书。另请参阅 证书颁发机构(CA)、subordinate CA、root CA。

certificate

根据 X.509 标准格式化的数字数据，指定个人、公司或其他实体（证书的 主题名称 ）的名称，证书也包含在证书中，该实体也属于该实体。公钥证书由 证书颁发机构(CA) 发布并数字签名。可以通过 数字签名 技术检查 CA 的 公钥加密 来验证证书的有效性。要在 公钥基础架构(PKI) 中信任，证书必须由在 PKI 中注册的其他实体信任的 CA 发布并签名。

CMC

请参阅 通过加密消息语法(CMC)的证书管理消息。

CMC Enrollment

允许使用代理签名证书将签名注册或签名的撤销请求发送到证书管理器的功能。然后，证书管理器会自动处理这些请求。

CMMF

请参阅 证书管理消息格式(CMMF)。

CRL

请参阅 证书撤销列表(CRL)。

CRMF

请参阅 证书请求消息格式(CRMF)。

CSP

请参阅 加密服务提供商(CSP)。

串联 CA

请参阅 链接的 CA。

信任链

请参阅 证书链。

加密服务提供商(CSP)

一个加密模块，它执行加密服务（如密钥生成、密钥存储和加密）代表使用标准接口（如 PKCSVRF 定义）来请求此类服务的软件。

加密模块

请参阅 PKCS rebased 模块。

加密消息语法(CS)

用于数字签名、摘要、验证或加密任意消息的语法，如 CMMF。

加密算法

用于执行加密操作的一组规则或指示，如 encryption 和 解密。

基于证书的验证

基于证书和公钥加密进行身份验证。另请参阅 基于密码的身份验证。

客户端 SSL 证书

用于通过 SSL 协议将客户端标识到服务器的证书。请参阅 安全套接字层(SSL)。

客户端身份验证

将客户端识别到服务器的过程，例如使用名称和密码，或者使用证书以及一些数字签名数据。请参阅 基于证书的验证、基于密码的身份验证、服务器身份验证。

密码

请参阅 *加密算法*。

证书扩展

X.509 v3 证书包含一个 extensions 字段，允许将任意数量的其他字段添加到证书。证书扩展提供了一种为证书添加其他主题名称和使用限制等信息的方法。PKIX 工作组定义了很多标准扩展。

证书指纹

与证书关联的 单向哈希。数字不是证书本身的一部分，而是通过将哈希功能应用到证书的内容来生成。如果证书的内容发生变化，即使由一个字符表示，相同的功能也会生成不同的数字。因此，可以使用证书指纹来验证证书是否已被篡改。

证书撤销列表(CRL)

由 X.509 标准定义，按序列号吊销的证书列表，由 证书颁发机构(CA) 生成并签名。

证书管理器

充当证书颁发机构的独立证书系统子系统。证书管理器实例问题、续订和撤销证书，该证书可以与 CRL 一起发布到 LDAP 目录。它接受来自端点实体的请求。请参阅 证书颁发机构(CA)。

证书管理器代理

属于组的用户，有权管理证书管理器的代理服务。这些服务包括访问和修改（批准和拒绝）证书请求和发布证书的能力。

证书管理消息格式(CMMF)

消息格式用于传达来自结束实体的证书请求和撤销请求，并将各种信息发送到最终实体。来自互联网工程任务组(IETF) PKIX 工作组的提议标准。CMMF 被另一个提议的标准 通过加密消息语法(CMC)的证书管理消息 使用。有关详细信息，请参阅 https://tools.ietf.org/html/draft-ietf-pkix-cmmf-02。

证书系统

请参阅 *Red Hat Certificate System*、*加密消息语法(CS)*。

证书系统子系统

五个证书系统管理器之一： 证书管理器、在线证书状态管理器、令牌密钥服务或令牌处理系统。密钥恢复授权机构

证书系统控制台

可以为任何单个证书系统实例打开的控制台。证书系统控制台允许证书系统管理员控制相应证书系统实例的配置设置。

证书请求消息格式(CRMF)

用于管理 X.509 证书的消息的格式。这个格式是 CMMF 的子集。另请参阅 证书管理消息格式(CMMF)。有关详细信息，请参阅 http://www.ietf.org/rfc/rfc2511.txt。

证书配置文件

定义特定类型的注册类型的一组配置设置。证书配置文件为特定类型的注册设置策略，以及证书配置文件中的身份验证方法。

证书链

由连续证书颁发机构签名的分层一系列证书。CA 证书标识了一个 证书颁发机构(CA)，用于签署该机构发布的证书。CA 证书可反过由父 CA 的 CA 证书签名，以此类推 root CA。证书系统允许任何端点检索证书链中的所有证书。

证书颁发机构(CA)

在验证证书要识别的人员或实体的身份后，发出 certificate 的可信实体。CA 还会续订并撤销证书并生成 CRL。在证书的 issuer 字段中命名的实体始终是 CA。证书颁发机构可以是独立的第三方，也可以是使用证书颁发服务器软件（如 Red Hat Certificate System）的人员或机构。

跨对证书

由一个 CA 发布的证书到另一个 CA，然后由两个 CA 存储以形成信任圆圈。两个 CA 相互发布证书，然后将两个跨对证书存储为证书对。

跨认证

在不同的认证层次结构或链中通过两个 CA 交换证书。交叉认证扩展了信任链，以便它包含这两个层次结构。另请参阅 证书颁发机构(CA)。

通过加密消息语法(CMC)的证书管理消息

用于将证书请求发送到证书管理器的消息格式。来自互联网工程任务组(IETF) PKIX 工作组的提议标准。有关详细信息，请参阅 https://tools.ietf.org/html/draft-ietf-pkix-cmc-02。

D

delta CRL

一个 CRL，其中包含自上次完整的 CRL 发布以来已撤销的证书列表。

分发点

用于 CRL 以定义一组证书。每个分发点都由发布的一组证书定义。可以为特定的发布点创建 CRL。

双密钥对

两个公钥-私钥对，一起四个键，对应于两个单独的证书。一个对的私钥用于签名操作，其他对的公钥用于加密和解密操作。每个对都对应一个独立的 certificate。另请参阅 加密密钥、公钥加密、签名密钥。

可区分名称(DN)

一系列 AVAs 用于识别证书主题。请参阅 属性值断言(AVA)。

密钥恢复授权代理

属于组的用户，有权管理密钥恢复授权的代理服务，包括管理请求队列并使用基于 HTML 的管理页面授权恢复操作。

密钥恢复授权机构

为结束实体管理 RSA 加密密钥的长期归档和恢复的可选独立证书系统子系统。在发布新证书前，可将证书管理器配置为使用密钥恢复授权来归档最终实体的加密密钥。只有在端点实体加密数据（如敏感电子邮件）时，Key Recovery Authority 才有用，组织可能需要恢复一天。它只能与支持双密钥对的末尾实体一起使用：两个单独的密钥对，一个用于加密，另一个用于数字签名。

密钥恢复授权机构传输证书

认证终端实体使用的公钥，以加密实体的加密密钥以传输到密钥恢复授权。密钥恢复授权机构使用与认证公钥对应的私钥来解密最终实体的密钥，然后才能使用存储密钥对其进行加密。

密钥恢复授权机构存储密钥

密钥恢复机构用来加密最终实体加密密钥的特殊密钥，在使用密钥恢复机构的私钥解密后对其进行加密。存储密钥永远不会保留密钥恢复授权。

密钥恢复授权机构恢复代理

拥有部分存储密钥的 n 人之一 密钥恢复授权机构。

数字 ID

请参阅 certificate。

数字签名

要创建数字签名，签名软件首先从要签名的数据（如新签发的证书）创建一个 单向哈希。然后，使用签名人的私钥加密单向哈希。生成的数字签名对于签名的每个数据都是唯一的。即使在消息中添加了一个逗号会更改该消息的数字签名。使用签名人的公钥成功解密数字签名，并与同一数据的另一个哈希进行比较，提供 篡改检测。为包含公钥的证书验证 证书链，提供签名人验证。另请参阅 nonrepudiation、encryption。

解密

取消加密的数据。请参阅 encryption。

E

eavesdropping

Surreptitious 截获在网络中发送的信息并非预期的实体。

Elliptic Curve Cryptography (ECC)

使用 elliptic curves 为加密密钥基础的数学问题创建 additive logarithms 的加密算法。ECC 密码比 RSA 密码使用效率更高，因为其内部复杂性比 RSA 密码更小。如需更多信息，请参阅 https://tools.ietf.org/html/draft-ietf-tls-ecc-12。

encryption

以忽略其含义的方式对信息进行模糊处理。请参阅 解密。

enrollment

请求和接收 X.509 证书以便在 公钥基础架构(PKI) 中使用的过程。也称为 注册。

extensions 字段

请参阅 证书扩展。

加密密钥

仅用于加密的私钥。加密密钥及其等同的公钥，以及一个 签名密钥 及其等同的公钥组成一个 双密钥对。

结束实体

在 公钥基础架构(PKI) 中，个人、路由器、服务器或其他使用 certificate 识别其自身的实体。

F

Federal Bridge 证书颁发机构(FBCA)

一个配置，其中两个 CA 形成一个信任圆圈，方法是相互发出跨对证书，并将两个跨对证书存储为单个证书对。

fingerprint

请参阅 证书指纹。

FIPS PUBS 140

Federal Information Standards Publications (FIPS PUBS) 140 是实施加密模块的美国政府标准、加密和解密数据的硬件或软件，或者执行其他加密操作，如创建或验证数字签名。许多销售到美国政府的产品必须符合一个或多个 FIPS 标准。请参阅 http://www.nist.gov/itl/fipscurrent.cfm。

firewall

在两个或多个网络间强制实施边界的系统或组合。

H

超文本传输协议(HTTP)和 Hypertext 传输协议安全(HTTPS)

用于与 Web 服务器通信的协议。HTTPS 由通过传输层安全(TLS)加密的连接内通过 HTTP (Hypertext 传输协议)进行通信。HTTPS 的主要目的是验证访问的网站以及交换数据隐私和完整性的保护。

I

IP 欺骗

客户端 IP 地址的伪造。

IPv4 和 IPv6

证书系统支持 IPv4 和 IPv6 地址命名空间，用于与所有子系统和工具的通信和操作，以及客户端、子系统创建和令牌和证书注册。

中间 CA

其证书位于 root CA 和 证书链 中发布的证书的 CA。

模拟

充当通过网络发送的信息的预期接收者。模拟可以采用两种形式： 欺骗 和 misrepresentation。

输入

在证书配置文件功能上下文中，它定义了特定证书配置文件的注册表单。每个输入都会被设置，然后从为此注册配置的所有输入动态创建注册表单。

J

JAR 文件

为压缩的文件集合进行数字信封，根据 Java™ 存档(JAR)格式 进行组织。

Java™ Development Kit (JDK)

Sun Microsystems 提供的软件开发套件，用于使用 Java™ 编程语言开发应用程序和小程序。

Java™ 加密架构(JCA)

由 Sun Microsystems 开发用于加密服务的 API 规格和参考。See http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.Introduction.

Java™ 原生接口(JNI)

在给定平台上提供 Java™ 虚拟机(JVM)不同实施的标准编程接口，允许使用 C 或 C++ 等语言编写的现有代码来绑定到 Java™。See

http://java.sun.com/products/jdk/1.2/docs/guide/jni/index.html.

Java™ 存档(JAR)格式

一组将数字签名、安装程序脚本和其他信息与目录中的文件关联的约定。

Java™ 安全服务(JSS)

用于控制网络安全服务(NSS)执行的安全操作的 Java™ 接口。

K

KEA

请参阅 密钥交换算法(KEA)。

key

加密算法 用来加密或解密数据的大量数字。例如，个人的 公钥 允许其他人加密针对该人员的信息。然后，必须使用对应的 私钥 解密信息。

KEYGEN 标签

生成密钥对以用于证书的 HTML 标签。

密钥交换

一个包括客户端和服务器的流程，以确定他们在 SSL 会话期间要使用的对称密钥。

密钥交换算法(KEA)

美国政府用于密钥交换的算法。

L

轻量级目录访问协议(LDAP)

用于在 TCP/IP 和多个平台上运行的目录服务协议。LDAP 是简化的目录访问协议(DAP)版本，用于访问 X.500 目录。LDAP 受 IETF 更改控制，已演变以满足互联网要求。

链接的 CA

一个内部部署的证书颁发机构(CA)，它的证书由公共的第三方 CA 签名。内部 CA 充当其问题的根 CA，第三方 CA 充当与其他 CA 链接到同一第三方 root CA 发布的证书的根 CA。也称为"链接 CA"，以及由不同公共 CA 使用的其他术语。

M

MD5

由 Ronald Rivest 开发的消息摘要算法。另请参阅 单向哈希。

misrepresentation

将实体表示为不是它的个人或组织。例如，当网站实际上是采用信用卡付款但从未发送任何好状态的站点时，网站可能被认为是光明的存储。Misrepresentation 是 模拟 的一种形式。另请参阅 欺骗。

手动身份验证

配置需要人类批准每个证书请求的证书系统子系统的方法。使用这种身份验证时，servlet 会在成功身份验证模块处理后将证书请求转发到请求队列。然后，在配置集处理和证书颁发前，需要单独批准每个具有适当权限的代理。

消息摘要

请参阅 单向哈希。

N

non-TMS

非令牌管理系统。指的是 不直接 处理智能卡的子系统(CA 和可选，KRA 和 OCSP)的配置。

参见令牌管理系统(TMS).

nonrepudiation

消息的发送者无法拒绝发送邮件。数字签名 提供了一种非缓解形式。

网络安全服务(NSS)

一组库，旨在支持支持安全的通信应用程序的跨平台开发。使用 NSS 库构建的应用程序支持 安全套接字层(SSL) 协议进行身份验证、篡改检测和加密令牌接口，以及用于加密令牌接口的 PKCSROX

协议。NSS 也作为软件开发工具包单独提供。

O

OCSP

在线证书状态协议.

operation

在访问控制指令中允许或拒绝的特定操作，如读取或写入。

output

在证书配置文件功能上下文中，它定义了从成功为特定证书配置文件的证书注册而生成的表单。设置每个输出，然后从为此注册配置的所有输出动态创建表单。

单向哈希

1.从任意长度的数据生成的固定长度很多，并附带哈希算法。数字也称为消息摘要，对散列数据是唯一的。对数据的任何更改（甚至删除或更改单个字符）都会生成不同的值。

2.散列数据的内容无法从哈希中推断出来。

对象签名

文件签名方法，使软件开发人员能够签署 Java 代码、JavaScript 脚本或任何类型的文件，并允许用户识别签名者并对本地系统资源控制访问。

对象签名证书

关联的私钥证书用于为对象签名；与 对象签名 相关。

P

PKCS #10

管理证书请求的公钥加密标准。

PKCS #11

管理加密令牌（如智能卡）的公钥加密标准。

PKCS #12

管理关键可移植性的公钥加密标准。

PKCS #7

管理签名和加密的公钥加密标准。

PKCS rebased 模块

通过 PKCS reflects 接口提供加密服务的加密设备的驱动程序，如加密和解密。在硬件或软件中，可以实施 PKCSBACKEND 模块 （也称为 加密模块或加密服务提供商 ）。PKCSBACKEND 模块始终具有一个或多个插槽，这些插槽可能以某种形式的物理读取器（如用于智能卡）或软件中的概念插槽作为物理硬件插槽实施。PKCSjpeg 模块的每个插槽可以包含一个令牌，这是实际提供加密服务的硬件或软件设备，并选择性地存储证书和密钥。红帽使用证书系统提供了一个内置的 PKCS facilities 模块。

PKIX 证书和 CRL 配置文件

由 IETF 为互联网的公共密钥基础架构开发的标准。它为证书指定配置集和 CRL。

公钥

公钥加密中使用的一对密钥。公钥是免费发布，并作为 certificate 的一部分发布。它通常用于加密发送到公钥所有者的数据，然后使用对应的 私钥 解密数据。

公钥加密

组经过精心设计的技术和标准，允许实体以电子方式验证身份或签名和加密电子数据。涉及两个密钥，一个公钥和一个私钥。公钥 作为证书的一部分发布，该证书将该密钥与特定身份相关联。对应的私钥已保密。使用公钥加密的数据只能使用私钥解密。

公钥基础架构(PKI)

有助于在联网环境中使用公钥加密和 X.509 v3 证书的标准和服务。

基于密码的身份验证

通过名称和密码自信地识别。另请参阅 身份验证、基于证书的验证。

存档证明(POA)

使用私钥恢复授权传输密钥签名的数据，其中包含有关存档最终用户密钥的信息，包括密钥序列号、密钥恢复机构的名称、对应证书的 主题名称 以及归档日期。签名的验证数据是密钥恢复授权在成功密钥归档操作后向证书管理器返回的响应。另请参阅 密钥恢复授权机构传输证书。

私钥

公钥加密中使用的一对密钥。私钥已保存 secret，用于解密使用对应 公钥 加密的数据。

R

RC2, RC4

由 Rivest 为 RSA 数据安全性开发的加密算法。另请参阅 加密算法。

Red Hat Certificate System

一组高度可配置的软件组件和工具，用于创建、部署和管理证书。证书系统由五个主要子系统组成，可在不同物理位置的不同证书系统实例中安装： 证书管理器、在线证书状态管理器、令牌密钥服务和令牌处理系统。密钥恢复授权机构

registration

请参阅 enrollment。

root CA

证书颁发机构(CA)，在证书链的顶部带有一个自签名证书。另请参阅 CA 证书、subordinate CA。

RSA 密钥交换

基于 RSA 算法的 SSL 的密钥交换算法。

RSA 算法

Rivest-Shamir-Adleman 的缩写是加密和身份验证的公共密钥算法。它由 Ronald Rivest、Adi Shamir 和 Leonard Adleman 开发的，在 1978 年推出。

S

sandbox

Java™ 术语，用于精心定义的 Java™ 代码必须在其中操作的限制。

Security-Enhanced Linux (SELinux)

安全增强型 Linux (SELinux)是一组在 Linux 系统内核强制执行强制访问控制的安全协议。SELinux 由美国国家安全局开发，通过宽松或有缺陷的访问控制保持应用程序访问机密或受保护文件。

servlet

用于代表证书系统子系统处理特定类型的与结束实体的 Java™ 代码。例如，证书注册、撤销和密钥恢复请求都由单独的 servlet 来处理。

SHA

安全哈希算法，这是美国政府使用的哈希函数。

slot

PKCS rebased 模块 的一部分，在硬件或软件中实施，其中包含 token。

SSL

请参阅 安全套接字层(SSL)。

subject

由 certificate 标识的实体。特别是，证书的 subject 字段包含一个唯一描述认证实体的 主题名称。

subordinate CA

证书的证书颁发机构由另一个从属 CA 或 root CA 签名。请参阅 CA 证书、root CA。

主题名称

唯一描述了 可区分名称(DN) 的 subject 的 certificate。

单点登录

1.在证书系统中，通过存储内部数据库和令牌的密码来简化到 Red Hat Certificate System 的方法。每次用户登录时，都需要输入此单一密码。

2.用户可以一次性登录单个计算机，并由网络中的各种服务器自动进行身份验证。部分单点登录解决方案可以采用多种形式，包括用于自动跟踪不同服务器使用密码的机制。证书支持 公钥基础架构 (PKI) 中的单点登录。用户可以一次登录到本地客户端的私钥数据库，只要客户端软件正在运行，则依赖 基于证书的验证 访问用户可访问的机构中的每个服务器。

安全域

PKI 子系统的集中存储库或清单。它的主要目的是，通过自动在子系统之间建立可信关系，促进新 PKI 服务的安装和配置。

安全套接字层(SSL)

允许客户端和服务器间的相互身份验证的协议，以及验证和加密的连接建立。SSL 在 TCP/IP 之上运行，低于 HTTP、LDAP、IMAP、NNTP 和其他高级网络协议。

安全频道

TPS 和智能卡之间的安全关联，允许根据 TKS 和智能卡 APDU 生成的共享主密钥进行加密。

对称加密

使用相同的加密密钥来加密和解密给定消息的加密方法。

智能卡

包含微处理器的小型设备并存储加密信息，如密钥和证书，并执行加密操作。智能卡实现一些或者所有 PKCS #11 接口。

服务器 SSL 证书

使用 安全套接字层(SSL) 协议将服务器标识到客户端的证书。

服务器身份验证

向客户端识别服务器的过程。另请参阅 客户端身份验证。

欺骗

准备成为其他人.例如：一个人可以假定使用电子邮件地址 jdoe@example.com，或者计算机可以将自己识别为名为 www.redhat.com 的站点（如果不存在）。欺骗是 模拟 的一种形式。另请参阅 misrepresentation。

签名密钥

仅用于签名的私钥。签名密钥及其等同的公钥，加上 加密密钥 及其等同的公钥构成了 双密钥对。

签名的审计日志

请参阅 审计日志。

签名算法

用于创建数字签名的加密算法。证书系统支持 MD5 和 SHA 签名算法。另请参阅 加密算法、数字签名。

签名证书

是公钥的证书与用于创建数字签名的私钥对应。例如，证书管理器必须具有一个签名证书，该签名证书与其用来为问题的证书签名的私钥对应。

简单的证书注册协议(SCEP)

Cisco 设计的协议，用于指定路由器与 CA 通信以进行路由器注册的方法。证书系统支持 SCEP 的 CA 操作模式，其中请求使用 CA 签名证书加密。

自我测试

在实例启动和按需时测试证书系统实例的功能。

T

token

与 slot 中的 PKCS rebased 模块 关联的硬件或者软件设备。它提供加密服务，并选择性地存储证书和密钥。

trust

确信依赖个人或其他实体。在 公钥基础架构(PKI) 中，信任是指证书用户和签发证书的 证书颁发机

构(CA) 之间的关系。如果 CA 受信任，则可以信任由该 CA 发布的有效证书。

令牌处理系统(TPS)

直接交互企业安全客户端和智能卡的子系统，以管理这些智能卡上的密钥和证书。

令牌密钥服务(TKS)

令牌管理系统中的子系统，它根据智能卡 APDU 和其他共享信息（如令牌 CUID）生成特定、单独的密钥。

令牌管理系统(TMS)

相关的子系统 - CA、TKS、TPS 以及可选的 KRA - 用于管理智能卡（令牌）上的证书。

传输层安全性(TLS)

一组规则，用于管理服务器和客户端之间的服务器身份验证、客户端身份验证和加密通信。

树状层次结构

LDAP 目录的层次结构。

篡改检测

确保以电子形式接收的数据完全与相同数据的原始版本相对应的机制。

U

UTF-8

证书注册页面支持特定字段的所有 UTF-8 字符（通用名称、机构单元、请求者名称和其他备注）。UTF-8 字符串是可搜索的，在 CA、OCSP 和 KRA 最终用户和代理服务页面中正确显示。但是，UU-8 支持没有扩展到国际化的域名，如电子邮件地址中使用的域名。

V

虚拟专用网络(VPN)

连接地理距离企业部门的方式。VPN 允许部门通过加密频道进行通信，允许经过身份验证的机密事务，这些事务通常仅限于专用网络。

X

X.509 版本 1 和版本 3

International Telecommunications Union (ITU)推荐的数字证书格式。

索引

附录 A. 修订历史记录

请注意，修订号与本手册的版本相关，而不是与 Red Hat Certificate System 的版本号相关。

修订 10.3-0 **Tue Feb 22 2022** **Florian Delehaye**

10.3 的各种修复。

修订 10.2-0 **Wed Jul 14 2021** **Florian Delehaye**

10.2 的各种修复。

修订 10.1-2 **Fri Feb 27 2021** **Florian Delehaye**

更正以 FIPS 模式设置 HSM，以及各种次要编辑。

修订 10.1-1 **Mon Jan 25 2021** **Florian Delehaye**

各种修复和改进。

修订 10.1-0 **Wed Dec 3 2020** **Florian Delehaye**

Red Hat Certificate System 10.1 发行版本指南。

修订 10.0-1 **Tues Nov 3 2020** **Florian Delehaye**

添加了 RHCS 10 的升级和迁移步骤。

修订 10.0-0 ***Thur Sep 17 2020*** ***Florian Delehaye***

Red Hat Certificate System 10.0 发行版本。