Red Hat Certificate System 10

計画、インストール、およびデプロイメントのガイド

Red Hat Certificate System 10.4 向けに更新

Last Updated: 2023-09-10

Red Hat Certificate System 10 計画、インストール、およびデプロイメントのガイド

Red Hat Certificate System 10.4 向けに更新

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

法律上の通知

概要

本ガイドでは、PKI インフラストラクチャーの計画に関する主要な PKI (Public Key Infrastructure) の概念と決定について説明します。

目次

パート I. RED HAT CERTIFICATE SYSTEM のデプロイ方法の計画

このセクションでは、一般的な PKI プリンシパルや Certificate System およびそのサブシステムの特定機能など、Certificate System の概要を示します。デプロイメントのプランニングは、組織のニーズを満たす PKI インフラストラクチャーの設計に不可欠です。

第1章 公開鍵の暗号化の概要

公開鍵の暗号化と関連する標準規格は、署名および暗号化された電子メール、シングルサインオン、Transport Layer Security/Secure Sockets Layer (SSL/TLS) 通信などの、多くの製品のセキュリティー機能を利用します。本章では、公開鍵暗号の基本概念について説明します。

中間コンピューターを介して情報を渡すインターネットトラフィックは、サードパーティーによる傍受が可能になります。

傍受

情報はそのまま維持されますが、プライバシーが危険にさらされます。たとえば、あるユーザーがクレジットカード番号を収集したり、機密の会話を記録したり、分類された情報をインターセプトしたりできます。

改ざん

転送中の情報は変更または置き換えられ、受信者に送信されます。たとえば、誰かが商品の注文を変更したり、人の履歴書を変更したりする可能性があります。

Impersonation

情報は、意図された受信者を装った人に渡されます。偽装には、2 つの形式を使用できます。

- **なりすまし。**他人のふりをすることができます。たとえば、ユーザーがメールアドレス **jdoe@example.net** に、コンピューターが **www.example.net** という名前のサイトになりすますことができます。
- **詐称。**個人または組織は、自分自身を偽って伝えることができます。たとえば、**www.example.net** というサイトはオンラインの家具店になりすまし、実際にクレジットカードの支払いを受け取りながら、商品を配送することはありません。

公開鍵暗号 は、以下を使用してインターネットベースの攻撃に対する保護を提供します。

暗号化と復号

暗号化および復号化により、2 つの通信者が互いに送信される情報を不明確化させることができます。送信側は送信前に情報を暗号化またはスクリブルします。受信者は、情報を受信した後、情報を復号化またはスクランブル解除します。移動時には暗号化された情報は侵入できません。

改ざんの検出

改ざん検出により、情報の受信者は、情報が転送中に変更されていないことを確認できます。データの修正または置き換えの試行は検出されます。

認証

認証により、情報の受信者は、送信者の ID を確認することにより、その発信元を判別できます。

否認防止

否認防止は、情報の送信者が後日、情報が送信されなかったと主張することを防ぎます。

1.1. 暗号化と復号

暗号化 とは、情報を変換するプロセスであり、意図された受信者意外には誰も理解できません。**復号化**は、暗号化された情報をデコードするプロセスです。**暗号** とも呼ばれる暗号化アルゴリズムは、暗号

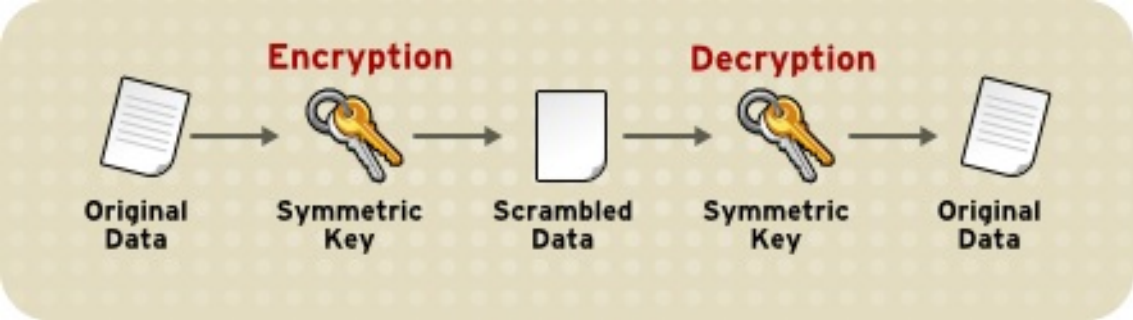
化または複号に使用される関数です。通常は、2 つの関連する機能が使用されます。1 つは暗号化用で、もう 1 つは復号化用です。

最新の暗号化では、暗号化された情報を秘密に保つ機能は、広く知られている暗号化アルゴリズムではなく、暗号化された結果を生成したり、以前に暗号化された情報を復号化するためにアルゴリズムで使用する必要がある **キー** と呼ばれる番号に基づいています。正しいキーを使用した復号はシンプルです。正しいキーがない状態での復号化は、不可能ではないにしても、非常に困難です。

1.1.1. 対称キーの暗号化

対称キーの暗号化では、暗号化キーを複号鍵から計算することもできます。ほとんどの対称アルゴリズムでは、図1.1「対称キーの暗号化」 に示されるように、同じ鍵が暗号化と復号の両方に使用されます。

図1.1 対称キーの暗号化



対称鍵暗号化の実装は非常に効率的であるため、暗号化と復号化の結果としてユーザーに大きな時間遅延が発生することはありません。

対称鍵暗号化は、関係する 2 つの当事者によって対称鍵が秘密にされている場合にのみ有効です。他のユーザーが鍵を検出した場合は、機密性と認証の両方に影響します。承認されていない対称キーを持つユーザーは、その鍵で送信されたメッセージのみに復号できますが、新しいメッセージを暗号化して、鍵を使用して信頼できる送信者の 1 つから送信されたかのように送信することができます。

対称キーの暗号化は、SSL/TLS 通信で重要なロールを果たします。これは、TCP/IP ネットワーク上で認証、改ざん検出、および暗号化に幅広く使用されます。SSL/TLS は公開鍵の暗号化技術も使用します。これについては、次のセクションで説明します。

1.1.2. 公開鍵の暗号化

公開鍵の暗号化 (非対称暗号化とも呼ばれます) には、エンティティーに関連付けられた鍵、公開鍵、および秘密鍵のペアが必要です。各公開鍵が公開され、対応する秘密鍵はシークレットが保持されます。(公開鍵の公開方法に関する詳細は、「証明書および認証」を参照してください。) 公開鍵で暗号化したデータは、対応する秘密鍵でのみ復号できます。図1.2「公開鍵の暗号化」 は、公開鍵の暗号化が機能する方法の簡単なビューを示しています。

図1.2 公開鍵の暗号化

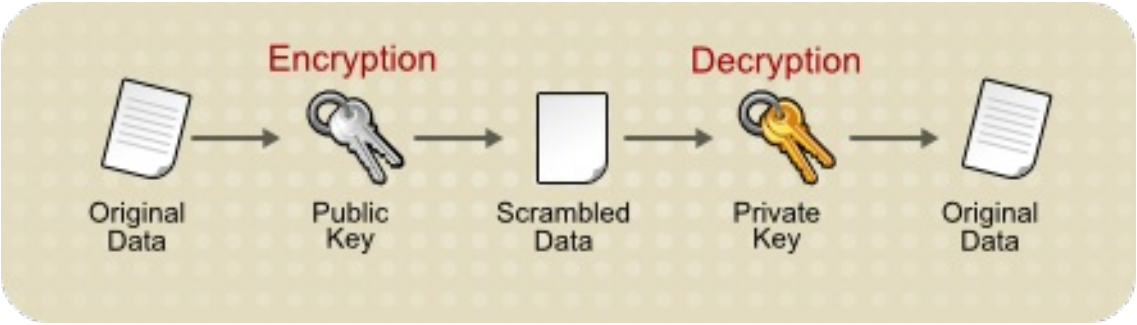


図1.2「公開鍵の暗号化」 に表示されるスキームでは、公開鍵を自由に配布できますが、承認されたユーザーだけがこのキーを使用して暗号化されたデータを読み取ることができます。一般的に、暗号化されたデータを送信するには、そのユーザーの公開鍵でデータは暗号化され、暗号化されたデータを対応する秘密鍵で復号します。

対称鍵暗号化と比較して、公開鍵暗号化はより多くの処理を必要とし、大量のデータの暗号化と復号には適さない場合があります。ただし、公開鍵の暗号化を使用して対称キーを送信できます。これにより、追加データの暗号化に使用できます。これは、SSL/TLS プロトコルによって使用される方法です。

図1.2「公開鍵の暗号化」 にあるスキームの逆もまた機能します。秘密鍵で暗号化されたデータは、対応する公開鍵でのみ復号できます。ただし、機密データを暗号化することは推奨される方法ではありません。これは、定義によって公開されている公開鍵を持つすべてのユーザーがデータを復号する可能性があるためです。それでも、秘密鍵暗号化は、電子商取引やその他の暗号化の商用アプリケーションにとって重要な要件であるデジタル署名を使用してデータに署名するために秘密鍵を使用できることを意味するため、便利です。その後、Mozilla Firefox などのクライアントソフトウェアは公開鍵を使用して、メッセージが適切な秘密鍵で署名され、署名されてから改ざんされていないことを確認できます。「デジタル署名」 は、この確認プロセスがどのように機能するかを説明します。

1.1.3. キー長および暗号化の強化

暗号化アルゴリズムを **壊す** と、暗号化されたデータにアクセスするための鍵をプレーンテキストで検索できます。対称アルゴリズムでは、アルゴリズムがテキストの暗号化に使用する鍵を判断しようとします。公開鍵アルゴリズムの場合、アルゴリズムを破ることは通常、2 人の受信者間で共有秘密情報を取得することを意味します。

対称アルゴリズムを壊す方法の 1 つは、適切なキーを見つけるまで、完全なアルゴリズム内のすべての鍵を単純に試すことです。公開鍵アルゴリズムの場合、鍵ペアの半分は公開されているため、残りの半分 (秘密鍵) は、複雑ではありますが、公開された数学的計算を使用して導出できます。アルゴリズムを壊す鍵を手動で検索することはブルートフォース攻撃と呼ばれます。

アルゴリズムを破ると、個人情報を傍受したり、なりすまして不正に検証したりするリスクが生じます。

アルゴリズムの **キー強度** は、アルゴリズムを壊し、ブルートフォース攻撃と比較する最速な方法を見つけることで決定します。

対称キーでは、暗号化の実行に使用する鍵のサイズや **長さ** において、暗号化強度が説明されています。通常は、より強力な暗号化が提供されます。キーの長さはビット単位で測定されます。

キーを破壊する最もよく知られている攻撃が、すべてのキーの可能性をテストするブルートフォース攻撃よりも速くない場合、暗号化キーは完全な強度であると見なされます。

異なるタイプのアルゴリズム (特に公開鍵アルゴリズム) では、対称鍵暗号と同じレベルの暗号化強度を実現するために、異なる鍵の長さが必要になる場合があります。RSA 暗号は、それが基づいている数学的問題の性質により、特定の長さのキーに対して可能なすべての値のサブセットのみを使用できます。対称キーの暗号化に使用されるその他の暗号は、特定の長さのキーに可能なすべての値を使用できます。より一致するオプションがあると、セキュリティーが強化されます。

RSA 鍵を解読することは比較的簡単であるため、RSA 公開鍵暗号化暗号は、暗号的に強力であると見なされるために、非常に長い鍵 (少なくとも 2048 ビット) を持っている必要があります。一方、対称鍵暗号は、ほとんどのアルゴリズムで 80 ビットという非常に短いキー長を使用すると、同等に強力であると見なされます。同様に、Elliptic Curve Digital Signature Algorithm (ECDSA) 暗号などの楕円曲線暗号 (ECC) に基づく公開鍵暗号では、RSA 暗号化よりもビットも少なくなる必要があります。

1.2. デジタル署名

改ざん検出は、**一方向ハッシュ** (**メッセージダイジェスト**とも呼ばれる) と呼ばれる数学関数に依存します。一方向ハッシュは、以下の特性を持つ多数の固定長です。

- ハッシュの値はハッシュデータに対して一意です。1 文字を削除または変更しても、データの変更は異なります。
- ハッシュされたデータの内容をハッシュから推測することはできません。

「公開鍵の暗号化」で説明されているように、秘密鍵を暗号化に使用して、対応する公開鍵を復号に使用できます。機密情報を暗号化する場合は推奨されませんが、データをデジタル署名する上では重要となります。署名ソフトウェアは、データ自体を暗号化する代わりに、データの一方向ハッシュを作成し、秘密鍵を使用してハッシュを暗号化します。暗号化したハッシュと、ハッシュアルゴリズムなどの他の情報はデジタル署名と呼ばれます。

図1.3「デジタル署名を使用したデータの整合性の検証」 は、デジタル署名を使用して署名されたデータの整合性を検証する方法を説明します。

図1.3 デジタル署名を使用したデータの整合性の検証

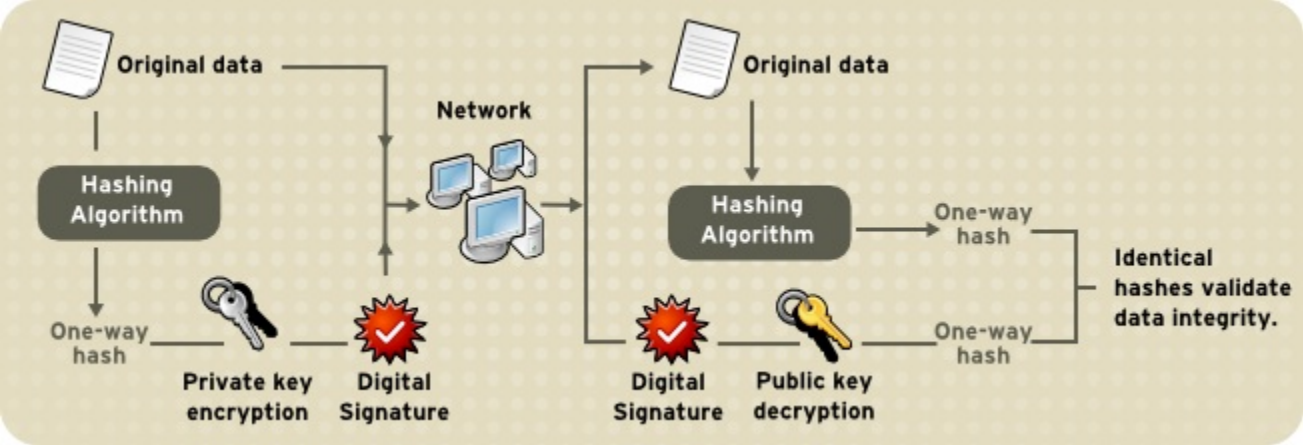


図1.3「デジタル署名を使用したデータの整合性の検証」 は、一部の署名済みデータの受信者に転送される 2 つの項目を示しています。元のデータとデジタル署名は、署名側の秘密鍵で暗号化した元のデータの一方向ハッシュです。データの整合性を検証するために、受信側のソフトウェアは最初に公開鍵を使用してハッシュを復号化します。その後、元のハッシュを生成したものと同じハッシュを使用して、同じデータの新しい一方向ハッシュを生成します。(使用されるハッシュアルゴリズムに関する情報がデジタル署名で送信されます。) 最後に、受信ソフトウェアは、新しいハッシュを元のハッシュと比較します。2 つのハッシュが一致する場合、データは署名されてから変更されていません。一致しない場合は、署名後にデータが改ざんされているか、署名者が提示した公開鍵に対応しない秘密鍵を使用して署名が作成されている可能性があります。

2 つのハッシュが一致する場合は、デジタル署名の復号に使用する公開鍵が、デジタル署名の作成に使用される秘密鍵に対応していることを確認することができます。署名側のアイデンティティーを検証するには、公開鍵が特定のエンティティーに属することを確認する方法も必要になります。認証の詳細は、「証明書および認証」を参照してください。

デジタル署名は手書きの署名に似ています。データが署名されたら、秘密鍵が侵害されていないと、後で実行を拒否するのが困難になります。この品質のデジタル署名は、高度な否認防止を提供します。デジタル署名は、署名者がデータに署名したことを否定することを困難にします。場合によって、デジタル署名は、手動で記述された署名として合理的にバインドされます。

1.3. 証明書および認証

1.3.1. 証明書は誰または何を識別

証明書とは、個人、サーバー、会社、または他のエンティティーを特定し、そのアイデンティティーを公開鍵に関連付けるために使用される電子ドキュメントです。運転免許または乗車のように、証明書は、一般的にユーザーのアイデンティティーを証明する証明を提供します。公開鍵暗号では、証明書を使用してなりすましの問題に対処します。

運転免許申請などの個人 ID を取得するには、そのユーザーが本人であることを検証する、他の形式の識別が必要です。証明書はほぼ同じように機能します。認証局 (CA) は ID を検証し、証明書を発行します。CA は、独立したサードパーティー、Certificate System などの独自の証明書発行サーバーソフトウェアを実行している組織のいずれかです。アイデンティティーの検証に使用される方法は、要求する証明書のタイプの特定の CA のポリシーによって異なります。証明書を実行する前に、CA は標準検証手順を使用してユーザーの ID を確認する必要があります。

CA によって発行された証明書は、特定の公開鍵を、従業員やサーバーの名前など、証明書が識別するエンティティーの名前にバインドします。証明書は、なりすましのための偽の公開鍵の使用を防ぐのに役立ちます。証明書で認定された公開鍵のみが、証明書で識別されたエンティティーが所有する対応する秘密鍵で機能します。

公開鍵の他に、証明書には常に識別するエンティティーの名前、有効期限、証明書を発行した CA 名、シリアル番号が含まれます。証明書には、発行する CA のデジタル署名が常に含まれます。CA のデジタル署名により、証明書は CA を把握して信頼しているが、証明書で識別したエンティティーを認識しないユーザーの有効な認証情報として機能できます。

CA のロールの詳細は、「CA 証明書による信頼の仕組み」を参照してください。

1.3.2. 認証によるアイデンティティーの確認

認証 は、アイデンティティーを確認するプロセスです。ネットワークの対話については、認証には、別の当事者による識別が必要です。ネットワーク上で認証を使用する方法は複数あります。証明書はその方法の 1 つになります。

通常、ネットワークの対話は、Web ブラウザーやサーバーなどのクライアント間で行われます。*クライアント認証* とは、サーバーによりクライアント (ソフトウェアの使用を前提としたユーザー) を特定することを指します。*サーバーの認証* とは、クライアントによりサーバー (ネットワークアドレスでサーバーの実行を想定している組織) を特定することを指します。

クライアントおよびサーバーの認証は、証明書がサポートする認証形式ではありません。たとえば、送信者を特定する証明書とともに、電子メールメッセージ上のデジタル署名は、メッセージの送信者を認証できます。同様に、HTML フォームのデジタル署名は、署名者を識別する証明書と組み合わせて、その証明書によって識別された人がフォームの内容に同意したという証拠を提供できます。認証に加えて、どちらの場合もデジタル署名はある程度の否認防止を保証します。デジタル署名は、署名者が後で電子メールまたはフォームを送信していないと主張することを困難にします。

クライアント認証は、ほとんどのイントラネットまたはエクストラネット内のネットワークセキュリティーの重要な要素です。クライアント認証には、主に 2 つの形式があります。

パスワードベースの認証

ほぼすべてのサーバーソフトウェアは、サーバーへのアクセスを付与する前に認識された名前およびパスワードを要求することで、クライアント認証を許可します。

証明書ベースの認証

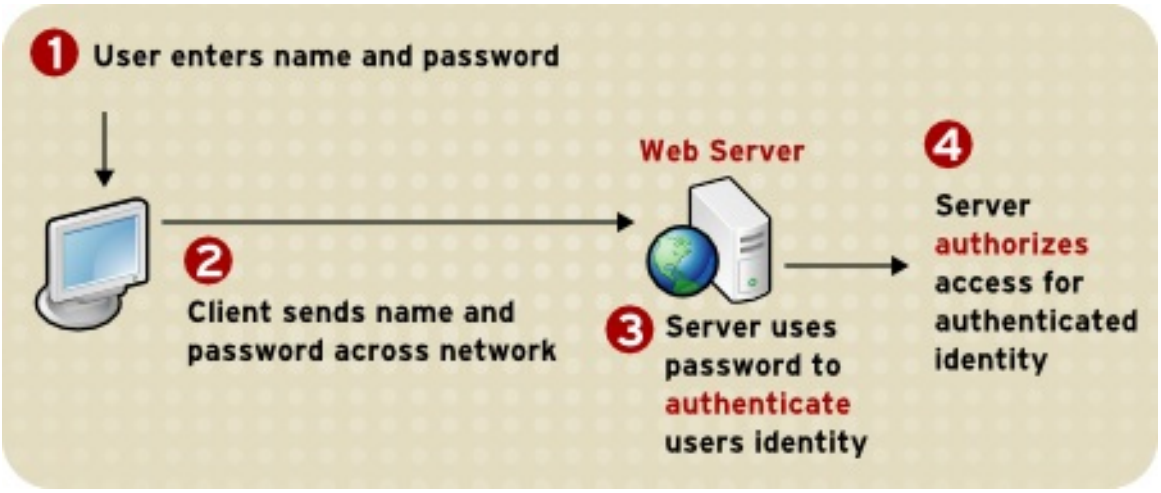
証明書に基づくクライアント認証は、SSL/TLS プロトコルの一部です。クライアントは無作為に生成されたデータの一部に署名し、ネットワーク全体で証明書および署名されたデータの両方を送信します。サーバーは署名を検証し、証明書の有効性を確認します。

1.3.2.1. パスワードベースの認証

図1.4「パスワードを使用したクライアントのサーバーへの認証」 は、ユーザー名とパスワードを使用してユーザーを認証するプロセスを示しています。この例では、以下を前提としています。

- ユーザーは、認証なしで、または SSL/TLS によるサーバー認証のベースとして、すでにサーバーを信頼しています。
- ユーザーがサーバーが制御するリソースを要求しました。
- 要求されたリソースへのアクセスを許可する前に、サーバーの認証が必要です。

図1.4 パスワードを使用したクライアントのサーバーへの認証



この認証プロセスの手順は次のとおりです。

1. サーバーがクライアントから認証を要求すると、クライアントはそのサーバーのユーザー名およびパスワードを要求するダイアログボックスが表示されます。
2. クライアントは、プレーンテキストまたは暗号化された SSL/TLS 接続を使用して、ネットワーク全体で名前とパスワードを送信します。
3. サーバーは、ローカルパスワードデータベースで名前とパスワードを検索し、一致する場合は、ユーザーの ID を認証するエビデンスとして受け入れます。
4. サーバーは、識別されたユーザーが要求されたリソースへのアクセスを許可されているかどうかを判断し、許可されている場合は、クライアントがそのリソースにアクセスできるようにします。

この方法では、ユーザーがアクセスする各サーバーに新しいパスワードを提供する必要があり、管理者は各ユーザーの名前とパスワードを追跡する必要があります。

1.3.2.2. 証明書ベースの認証

証明書ベースの認証の利点の 1 つは、認証の最初の 3 ステップを、ネットワークを越えて送信されない 1 つのパスワードを供給する仕組みに置き換えることができ、管理者がユーザーの認証を一元的に制御できることです。これは **シングルサインオン** と呼ばれます。

図1.5「証明書を使用したクライアントのサーバーへの認証」 は、証明書と SSL/TLS を使用してクライアント認証がどのように機能するかを示しています。ユーザーをサーバーに認証するには、クライアントが無作為に生成されたデータの一部に署名し、ネットワーク全体で証明書と署名済みデータの両方を送信します。サーバーは、証明書および署名されたデータのデータに基づいてユーザーのアイデンティティーを認証します。

図1.4「パスワードを使用したクライアントのサーバーへの認証」、図1.5「証明書を使用したクライアントのサーバーへの認証」と同様に、ユーザーがすでにサーバーを信頼し、リソースをリクエストしたこと、および要求されたリソースへのアクセスを付与する前にクライアント認証が要求されていることを前提としています。

図1.5 証明書を使用したクライアントのサーバーへの認証

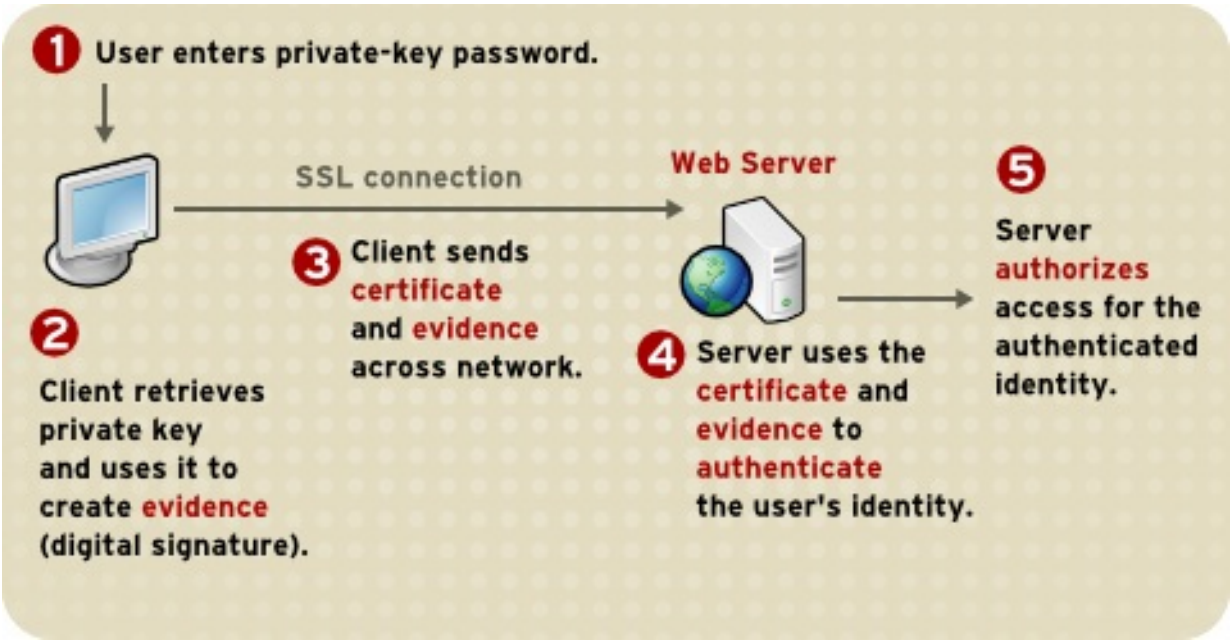
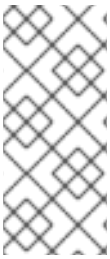


図1.4「パスワードを使用したクライアントのサーバーへの認証」の認証プロセスとは異なり、図1.5「証明書を使用したクライアントのサーバーへの認証」の認証プロセスには SSL/TLS が必要です。また、図1.5「証明書を使用したクライアントのサーバーへの認証」は、クライアントがサーバーに対してクライアントを識別するのに使用できる有効な証明書を持っていることを前提としています。証明書ベースの認証は、秘密鍵を所有し、パスワードを知っているユーザーの両方に基づいているため、パスワードベースの認証よりも優先されます。ただし、これら 2 つの仮定は、権限のない担当者がユーザーのマシンまたはパスワードにアクセスできず、クライアントソフトウェアの秘密鍵データベースのパスワードが設定されており、ソフトウェアが適度な頻度でパスワードを要求するように設定されている場合にのみ当てはまります。



注記

パスワードベースの認証または証明書ベースの認証は、個々のマシンまたはパスワードへの物理的なアクセスに関連するセキュリティーの問題に対応します。公開鍵暗号は、一部のデータの署名に使用される秘密鍵が証明書の公開鍵に対応していることを確認することしかできません。マシンの物理的なセキュリティーを保護し、秘密鍵のパスワードを秘密にしておくことは、ユーザーの責任です。

図1.5「証明書を使用したクライアントのサーバーへの認証」に記載されている認証手順は以下のとおりです。

1. クライアントソフトウェアは、そのクライアントに対して発行された証明書に発行される公開鍵に対応する秘密鍵のデータベースを維持します。クライアントは、証明書ベースのクライアント認証を必要とする SSL/TLS 対応サーバーに初めてアクセスしようとしたときなど、特定のセッション中にクライアントが初めてデータベースにアクセスする必要があるときに、このデータベースのパスワードを要求します。

   このパスワードを一度入力すると、他の SSL/TLS 対応サーバーにアクセスする場合でも、残りのセッションに対して再度入力する必要はありません。

2. クライアントは秘密鍵データベースのロックを解除し、ユーザーの証明書の秘密鍵を取得し、その秘密鍵を使用してクライアントとサーバーの両方からランダムなデータに署名します。このデータとデジタル署名は、秘密鍵の有効性について証明されています。デジタル署名はその秘密鍵でのみ作成でき、SSL/TLS セッションに固有の署名済みデータに対して、対応する公開鍵で検証できます。

3. クライアントは、ユーザーの証明書とランダムに生成されたデータの両方をネットワーク経由で送信します。

4. サーバーは、証明書と署名済みデータを使用してユーザーのアイデンティティーを認証します。

5. サーバーは、クライアントが提示する証明書が LDAP ディレクトリー内のユーザーのエントリーに保存されていることを確認するなど、他の認証タスクを実行できます。その後、サーバーは、指定のユーザーが要求されたリソースにアクセスできるかどうかを確認します。この評価プロセスでは、LDAP ディレクトリーまたは企業のデータベースで追加情報を使用すると、さまざまな標準的な承認メカニズムを使用できます。評価の結果が正である場合、サーバーは、要求されたリソースにクライアントがアクセスすることを許可します。

証明書は、クライアントとサーバーと間の相互作用の認証部分を置き換えます。ユーザーがネットワーク全体でパスワードを送信しなければならない代わりに、シングルサインオンでは、ネットワーク経由で送信せずに、ユーザーが秘密鍵のデータベースパスワードを一度入力する必要があります。セッションの残りの部分では、クライアントはユーザーの証明書を提示して、遭遇したそれぞれの新しいサーバーでユーザーを認証します。認証されたユーザー ID に基づく既存の承認メカニズムには影響はありません。

1.3.3. 証明書に使用

証明書の目的は、信頼を確立することです。その使用法は、それが保証するために使用される信頼の種類によって異なります。提示した者の ID を確認するために、いくつかの種類の証明書が使用されたり、オブジェクトまたはアイテムが改ざんされていないことを確認したりするために使用されます。

1.3.3.1. SSL/TLS

SSL/TLS (Transport Layer Security/Secure Sockets Layer) プロトコルは、サーバーの認証、クライアント認証、サーバーとクライアント間の暗号化された通信を管理します。SSL/TLS はインターネット上で広く使用され、特にクレジットカード番号などの機密情報に関連する対話に使用されます。

SSL/TLS には、SSL/TLS サーバー証明書が必要です。最初の SSL/TLS ハンドシェイクの一環として、サーバーは証明書をクライアントに提示してサーバーのアイデンティティーを認証します。認証は公開鍵の暗号化とデジタル署名を使用して、サーバーが、そのサーバーであると主張するサーバーであることを確認します。サーバーが認証されると、クライアントとサーバーは、対称鍵の暗号化を使用して、残りのセッションに対して交換されたすべての情報を暗号化し、改ざんを検出します。

サーバーは、クライアント認証とサーバーの認証を必要とするように設定できます。この場合、サーバーの認証が正常に完了した後に、クライアントの証明書をサーバーに提示して、暗号化された SSL/TLS セッションが確立される前にクライアントのアイデンティティーを認証する必要があります。

SSL/TLS によるクライアント認証の概要と、パスワードベースの認証との相違点は、「認証によるアイデンティティーの確認」を参照してください。

1.3.3.2. 署名済みおよび暗号化電子メール

メールプログラムは、S/MIME (Secure Multipurpose Internet Mail Extension) と呼ばれる広く使用されているプロトコルを使用して、署名済みおよび暗号化された電子メールをサポートします。S/MIME を

使用して電子メールメッセージに署名または暗号化するには、メッセージの送信者に S/MIME 証明書が必要です。

デジタル署名が含まれる電子メールメッセージは、メッセージヘッダーに名前が表示されるユーザーによって送信されたロールを果たすようにするため、送信者を認証します。電子メールソフトウェアがデジタル署名を検証できない場合には、ユーザーが警告されます。

デジタル署名は、それに付随するメッセージに固有のものです。受信したメッセージが送信したメッセージと何らかの形で異なる場合は、1 文字を追加または削除しても、デジタル署名を検証できません。そのため、署名された電子メールは、電子メールが改ざんされていないという保証も提供します。この種の保証は否認防止と呼ばれ、送信者がメッセージの送信を拒否することを困難にします。これはビジネス通信で重要になります。デジタル署名の動作方法は、「デジタル署名」を参照してください。

S/MIME を使用すると、一部のビジネスユーザーが重要な電子メールメッセージを暗号化することもできます。ただし、電子メールの暗号化を使用する場合は、注意して計画する必要があります。暗号化された電子メールメッセージの受信側が秘密鍵を失い、キーのバックアップコピーにアクセスできない場合は、暗号化されたメッセージが復号できなくなります。

1.3.3.3. シングルサインオン

ネットワークユーザーは、使用するサービスに複数のパスワードを覚えておく必要が頻繁にあります。たとえば、ユーザーがネットワークにログインするのに別のパスワードを入力してログインし、電子メールを収集して、ディレクトリーサービスを使用し、企業カレンダープログラムを使用して、さまざまなサーバーにアクセスしなければならない場合があります。複数のパスワードは、ユーザーおよびシステム管理者向けに継続されます。ユーザーはさまざまなパスワードを追跡するのが難しく、貧弱なパスワードを選択する傾向があり、わかりやすい場所にそれらを書き留める傾向があります。管理者は、各サーバー上の個別のパスワードデータベースを追跡し、パスワードがネットワークを介して定期的かつ頻繁に送信されるという事実に関連する潜在的なセキュリティー問題に対処する必要があります。

この問題の解決には、ユーザーが一度ログインして単一のパスワードを使用し、ユーザーがネットワーク経由でパスワードを送信できなくなるすべてのネットワークリソースへの認証アクセスが必要になります。この機能はシングルサインオンと呼ばれます。

クライアント SSL/TLS 証明書と S/MIME 証明書の両方が、包括的なシングルサインオンソリューションで大きなロールを果たすことができます。たとえば、Red Hat 製品がサポートするシングルサインオンの 1 つに、SSL/TLS クライアント認証を使用します。ユーザーは、ローカルクライアントの秘密鍵データベースに単一のパスワードを使用して一度ログインでき、ユーザーがネットワーク経由でパスワードを送信できなくなるすべての SSL/TLS 対応サーバーに認証されます。このアプローチでは、各新規サーバーにパスワードを入力する必要がないため、ユーザーのアクセスが簡素化されます。また、管理者はユーザーとパスワードのリストよりもはるかに長いリストではなく、認証局 (CA) のリストを制御することで、ネットワークの管理を簡素化できます。

証明書の使用に加え、ソリューションの完全なシングルサインオンは、パスワードやその他の認証形式に依存する基礎となるオペレーティングシステムなどのエンタープライズシステムとの対話を必要とする必要があります。

1.3.3.4. オブジェクトの署名

多くのソフトウェア技術は、**オブジェクト署名** と呼ばれるツールセットをサポートします。オブジェクト署名は、公開鍵暗号化の標準的な手法を使用して、ユーザーがダウンロードしたコードに関する信頼できる情報を、パッケージソフトウェアに関する信頼できる情報を取得するのとほぼ同じ方法で取得できるようにします。

最も重要な点として、オブジェクト署名は、ユーザーとネットワーク管理者がイントラネットまたはインターネットを介して配布されるソフトウェアに関する決定を実装するのに役立ちます。たとえば、特定のエンティティーによって署名された Java アプレットが特定のユーザーのマシンで特定のコン

ピューター機能を使用できるようにするかどうかなどです。

オブジェクト署名テクノロジーで署名されたオブジェクトは、アプレットや他の Java コード、JavaScript スクリプト、プラグイン、または何らかのファイルです。署名はデジタル署名です。署名オブジェクトとその署名は、通常 JAR ファイルと呼ばれる特別なファイルに格納されます。

オブジェクト署名テクノロジーを使用してファイルに署名するソフトウェア開発者やその他の人は、最初にオブジェクト署名証明書を取得する必要があります。

1.3.4. 証明書の種類

Certificate System は、さまざまな用途、およびさまざまな形式で、さまざまな種類の証明書を生成できます。PKI インスタンスと Certificate System インスタンスの両方を管理するには、必要な証明書を計画し、必要な形式の決定や更新の計画方法など、証明書の管理方法を計画することが重要です。

LDAP ディレクトリー、ファイル署名証明書、およびその他のサブシステム証明書のデュアル用途証明書用の証明書登録フォームがあります。これらのフォームは、**https://server.example.com:8443/ca/ee/ca** の Certificate Manager のエンドエンティティーページから入手できます。

さまざまな Certificate System サブシステムがインストールされると、必要となる基本的な証明書とキーが生成されます。たとえば、 Certificate Manager を設定すると、自己署名ルート CA の CA 署名証明書と、内部 OCSP 署名、監査署名、SSL/TLS サーバー、およびエージェントユーザー証明書が生成されます。KRA 設定時に、Certificate Manager はストレージ、トランスポート、監査署名、およびエージェント証明書を生成します。追加の証明書を個別に作成してインストールできます。

表1.1 共通証明書

証明書の種類	使用	例
クライアント SSL/TLS 証明書	SSL/TLS 経由でサーバーへのクライアント認証に使用されます。通常、クライアントの ID は、従業員などの個人の ID と同じであると見なされます。SSL/TLS クライアント証明書がクライアント認証に使用される方法の説明は、「証明書ベースの認証」を参照してください。クライアント SSL/TLS 証明書は、シングルサインオンの一部として使用することもできます。	銀行は顧客に SSL/TLS クライアント証明書を提供します。これにより、銀行のサーバーはその顧客を識別し、顧客のアカウントへのアクセスを承認できます。 会社は、会社のサーバーがその従業員を識別してその会社のサーバーへのアクセスを承認できるようにする SSL/TLS クライアント証明書を新たに付与します。
サーバー SSL/TLS 証明書	SSL/TLS でクライアントへのサーバーの認証に使用されます。サーバーの認証はクライアント認証なしで使用できます。暗号化された SSL/TLS セッションには、サーバーの認証が必要です。詳細は、「SSL/TLS」を参照してください。	電子商取引を行うインターネットサイトは通常、証明書ベースのサーバー認証をサポートして、暗号化された SSL/TLS セッションを確立し、会社で識別される Web サイトを扱っていることを顧客に保証します。暗号化された SSL/TLS セッションは、クレジットカード番号などのネットワーク上で送信する個人情報を簡単に傍受できないようにします。

証明書の種類	使用	例
S/MIME 証明書	署名および暗号化された電子メールに使用されます。SSL/TLS クライアント証明書と同様に、クライアントのアイデンティティーは従業員などのユーザーのアイデンティティーと同じであると仮定されます。1 つの証明書は S/MIME 証明書および SSL/TLS 証明書の両方として使用できます。「署名済みおよび暗号化電子メール」を参照してください。S/MIME 証明書はシングルサインオンの一部としても使用できます。	S/MIME 証明書と SSL/TLS 証明書を組み合わせることで、従業員の ID 認証のみを許可するため、署名済み電子メールと SSL/TLS クライアント認証は許可されますが、電子メールは暗号化されません。別の企業が S/MIME 証明書を署名して暗号化し、機密や法規制を扱う電子メールに署名して暗号化します。
CA 証明書	CA を識別するために使用されます。クライアントおよびサーバーソフトウェアは CA 証明書を使用して、その他の証明書を信頼できるものを決定します。詳細は、「CA 証明書による信頼の仕組み」を参照してください。	Mozilla Firefox に保存されている CA 証明書は、他の証明書を認証できるものを決定します。管理者は、各ユーザーの Firefox のコピーに保存されている CA 証明書を制御することで、企業のセキュリティーポリシーを実装できます。
オブジェクト署名証明書	Java コード、JavaScript スクリプト、またはその他の署名付きファイルの署名者を識別するのに使用されます。	ソフトウェア会社は、インターネット上で配布されるソフトウェアに頻繁に署名して、そのソフトウェアがその会社の合法的な製品であることをユーザーに保証します。証明書とデジタル署名を使用することで、ユーザーはダウンロードしたソフトウェアが自分のコンピューターにアクセスできる種類を識別して制御することもできます。

1.3.4.1. CA 署名証明書

すべての Certificate Manager には、発行する証明書および証明書失効リスト (CRL) の署名に使用する公開鍵と秘密鍵のペアを持つ CA 署名証明書があります。この証明書は、Certificate Manager のインストール時に作成され、インストールされます。

注記

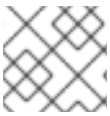
CRL の詳細は、「証明書の取り消しとステータスの確認」を参照してください。

Certificate Manager のステータスがルートまたは下位 CA として評価されるかどうかは、その CA 署名証明書が自己署名の証明書であるか、または別の CA により署名されているかにより決まります。自己署名ルート CA は、発行先名、発行可能な証明書のタイプ、証明書の発行先など、証明書を発行するのに使用するポリシーを設定します。下位 CA には、別の CA (通常は CA 階層の上位レベル (ルート CA である場合とそうでない場合があります)) によって署名された CA 署名証明書があります。Certificate Manager が CA 階層にある下位 CA の場合は、Certificate Manager を使用して証明書を発行する前に、ルート CA の署名証明書を個別のクライアントおよびサーバーにインポートする必要があります。

CA が発行するサーバーまたはユーザー証明書がそのクライアントにインストールされている場合は、その CA 証明書をクライアントにインストールする必要があります。CA 証明書は、サーバー証明書を信頼することを確認します。理想的には、証明書チェーンがインストールされています。

1.3.4.2. その他の署名証明書

OCSP (Online Certificate Status Protocol) レスポンダーサービスや CRL 公開などの他のサービスは、CA 証明書以外の署名証明書を使用できます。たとえば、別の CRL 署名証明書を使用して、CA 署名証明書を使用する代わりに CA によって公開される失効一覧に署名できます。

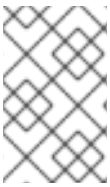


注記

OCSP の詳細は、「証明書の取り消しとステータスの確認」を参照してください。

1.3.4.3. SSL/TLS サーバーおよびクライアント証明書

サーバー証明書は、SSL/TLS などの安全な通信やその他の安全な機能に使用されます。サーバー証明書は、操作中に自身を認証し、データを暗号化するために使用されます。クライアント証明書は、サーバーに対してクライアントを認証します。



注記

サードパーティーが署名証明書を発行した CA はサーバー証明書を発行できない可能性があります。サードパーティーの CA には、部下がサーバー証明書を発行することを禁止するルールが設定されている場合があります。

1.3.4.4. ユーザー証明書

エンドユーザー証明書は、サーバーまたはシステムにユーザーを識別するために使用されるクライアント証明書のサブセットです。ユーザーは、SSL/TLS などのセキュアな通信に使用する証明書や、電子メールの暗号化やシングルサインオンに使用するその他の機能に割り当てることができます。Certificate System エージェントなどの特別なユーザーには、特別なサービスにアクセスするためのクライアント証明書を指定できます。

1.3.4.5. デュアルキーペア

デュアルキーペアは、2 組の秘密鍵と公開鍵で、もう 1 つは署名用のセットで、もう 1 つは暗号化に使用されます。これらのデュアルキーは、デュアル証明書の作成に使用されます。デュアル証明書の登録フォームは、Certificate Manager のエンドエンティティーページにリストされている標準形式です。

デュアルキーペアを生成する場合には、署名と暗号化用に別の証明書を生成する際に、証明書プロファイルが正しく機能するように設定します。

1.3.4.6. クロスペアの証明書

Certificate System は、クロスペアの CA 証明書を発行、インポート、および公開できます。クロスペアの証明書では、1 つの CA が署名し、2 番目の CA に対してクロスペアの証明書を発行します。2 番目の CA は署名し、最初の CA にペアの証明書を発行します。その後、どちらの CA も **crossCertificatePair** エントリーとして両方の証明書を保存または公開します。

ブリッジ証明書は、ルート CA にチェーンされない CA が発行する証明書を許可するために実行できます。クロスペアの CA 証明書で、Certificate System CA と他の CA との間で信頼を確立することで、クロスペアの証明書をダウンロードして、他の CA が発行する証明書を信頼するために使用できます。

1.3.5. 証明書の内容

証明書の内容は、国際標準化団体である国際電気通信連合 (ITU) によって推奨されている X.509v3 証明書仕様に従って編成されています。

通常は、証明書の正確な内容について懸念する必要はありません。ただし、証明書を操作するシステム管理者は、証明書に含まれる情報にある程度精通している必要がある場合があります。

1.3.5.1. 証明書データの形式

証明書要求と証明書を作成し、保存して、いくつかの形式でインストールできます。これらの形式はすべて X.509 標準に準拠します。

1.3.5.1.1. バイナリー

以下のバイナリーフォーマットが認識されます。

- **DER でエンコードされた証明書**。これは、単一のバイナリーの DER でエンコードされた証明書です。
- **PKCS#7 証明書チェーンオブジェクト**。これは、PKCS #7 **SignedData** オブジェクトです。**SignedData** オブジェクトの唯一の重要なフィールドは証明書で、署名およびコンテンツなどは無視されます。PKCS #7 形式を使用すると、複数の証明書を一度にダウンロードできます。
- **Netscape 証明書シーケンス**。これは、PKCS #7 **ContentInfo** 構造で証明書チェーンをダウンロードする簡単な形式で、証明書のシーケンスをラップします。**contentType** フィールドの値は **netscape-cert-sequence** で、content フィールドには以下の構造があります。

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

  この形式により、複数の証明書を同時にダウンロードできます。

1.3.5.1.2. テキスト

バイナリー形式はどれでもテキスト形式でインポートできます。テキストフォームは、次の行で始まります。

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

この行に従う証明書データで、上記のバイナリー形式のいずれかになります。このデータは、RFC 1113 で説明されているように base-64 でエンコードされる必要があります。証明書情報の後に以下の行を指定します。

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

1.3.5.2. 識別名

X.509 v3 証明書は、識別名 (DN) を公開鍵にバインドします。DN は、エンティティーを一意に識別する **uid=doe** などの一連の名前と値のペアです。これは、証明書の *サブジェクト名* とも呼ばれます。

以下は、Example Corp の従業員の DN の例です。

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

この DN では、**uid** はユーザー名、**cn** はユーザーの共通名、**o** は組織または会社名で、**c** は国です。

DN には、さまざまな名前と値のペアを含めることができます。LDAP ディレクトリー

DNS には、さまざまな名前と値のペアを含めることができます。LDAP (Lightweight Directory Access Protocol) をサポートするディレクトリーの証明書サブジェクトとエントリーの両方を識別するために使用されます。

DN を構築するルールは複雑です。DN に関する包括的な情報は、http://www.ietf.org/rfc/rfc4514.txt の『A String Representation of Distinguished Names』 を参照してください。

1.3.5.3. 典型的な証明書

すべての X.509 証明書は、以下の 2 つのセクションで設定されます。

データセクション

本セクションでは、以下を説明します。

- 証明書でサポートされる X.509 標準のバージョン番号。
- 証明書のシリアル番号CA が発行するすべての証明書には、その CA が発行した証明書間で一意のシリアル番号があります。
- 使用されるアルゴリズムや鍵自体の表現など、ユーザーの公開鍵に関する情報。
- 証明書を発行した CA の DN。
- 証明書が有効である期間。たとえば、2004 年 11 月 15 日の午後 1 時から、2022 年 11 月 15 日の午後 1 時までの間。
- 証明書サブジェクトの DN。サブジェクト名とも呼ばれます。たとえば、SSL/TLS クライアント証明書では、これはユーザーの DN です。
- 任意の *証明書エクステンション*。クライアントまたはサーバーによって使用される追加データを提供できます。以下に例を示します。
  - Netscape Certificate Type 拡張は、SSL/TLS クライアント証明書、SSL/TLS サーバー証明書、メール署名用の証明書などの証明書のタイプを示します。
  - SAN (Subject Alternative Name) 拡張が、証明書を 1 つ以上のホスト名にリンクします。

  証明書拡張機能は、別の目的でも使用できます。

署名セクション

本セクションでは、以下を説明します。

- 独自のデジタル署名を作成するために CA の発行に使用される暗号化アルゴリズムまたは暗号。
- 証明書内のすべてのデータを一緒にハッシュし、CA の秘密鍵で暗号化することによって取得された CA のデジタル署名。

読み取り可能な pretty-print (整形表示) 形式で表示される証明書のデータセクションと署名セクションは次のとおりです。

```
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
```

以下は、base-64 でエンコードされた形式と同じ証明書です。

```
-----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 証明書による信頼の仕組み

CA は ID を検証し、証明書を発行します。CA は、独立したサードパーティー、Certificate System などの独自の証明書発行サーバーソフトウェアを実行している組織のいずれかです。

証明書をサポートするクライアントまたはサーバーソフトウェアは、信頼できる CA 証明書のコレクションを維持します。これらの CA 証明書は、ソフトウェアが信頼または検証できる証明書の発行者を決定します。最も単純なケースでは、ソフトウェアは、証明書を持っている CA の 1 つによって発行された証明書のみを検証できます。信頼できる CA 証明書を CA 証明書のチェーンの一部にすることもできます。各証明書は、証明書階層の上位にある CA によって発行されます。

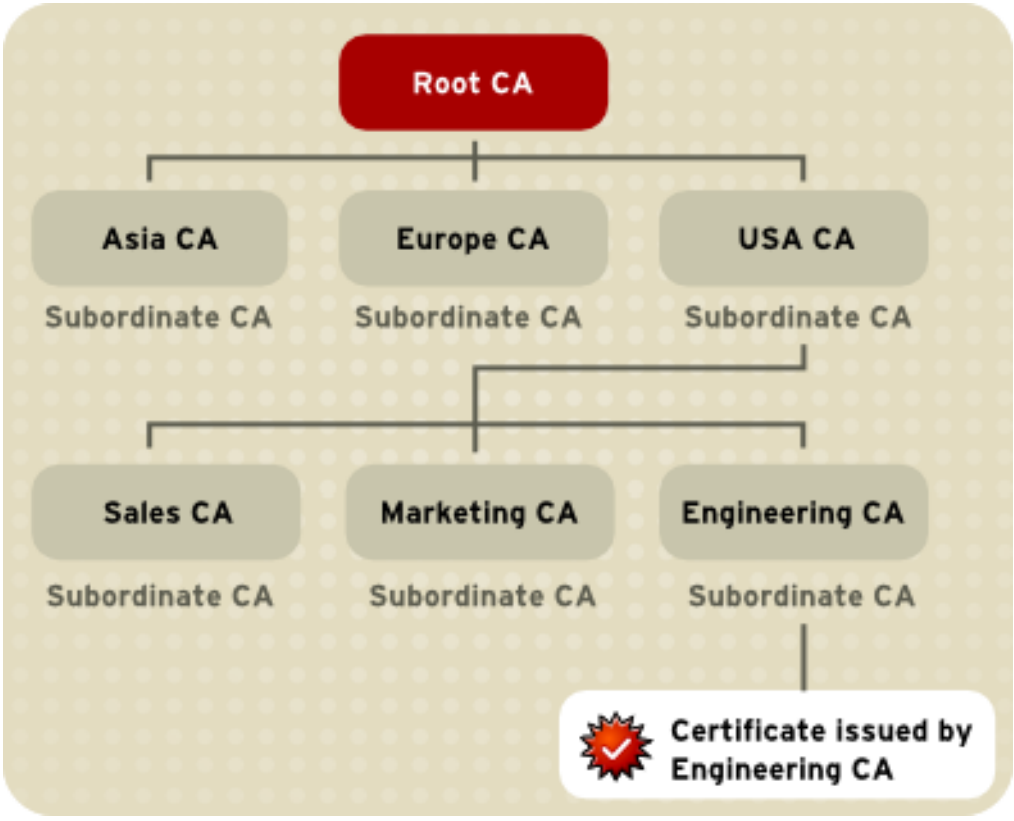
以下のセクションでは、証明書階層と証明書チェーンを使用して、信頼できる証明書ソフトウェアを決定する方法を説明します。

1.3.6.1. CA 階層

大規模な組織では、証明書を発行する責任を複数の CA に委任できます。たとえば、必要な証明書の数が多すぎて、単一の CA が維持できない場合があります。組織単位が異なれば、ポリシー要件も異なる場合があります。または、CA は、証明書を発行している人と同じ地理的領域に物理的に配置する必要がある場合があります。

これらの証明書発行の責任は、下位の CA 間で分割できます。X.509 標準には、CA の階層を設定するモデルが含まれています (例: 図1.6「認証局の階層の例」)。

図1.6 認証局の階層の例



ルート CA は階層の上部にあります。ルート CA の証明書は *自己署名証明書* です。つまり、証明書は、証明書を識別する同じエンティティーによってデジタル署名されます。ルート CA に直接従属する

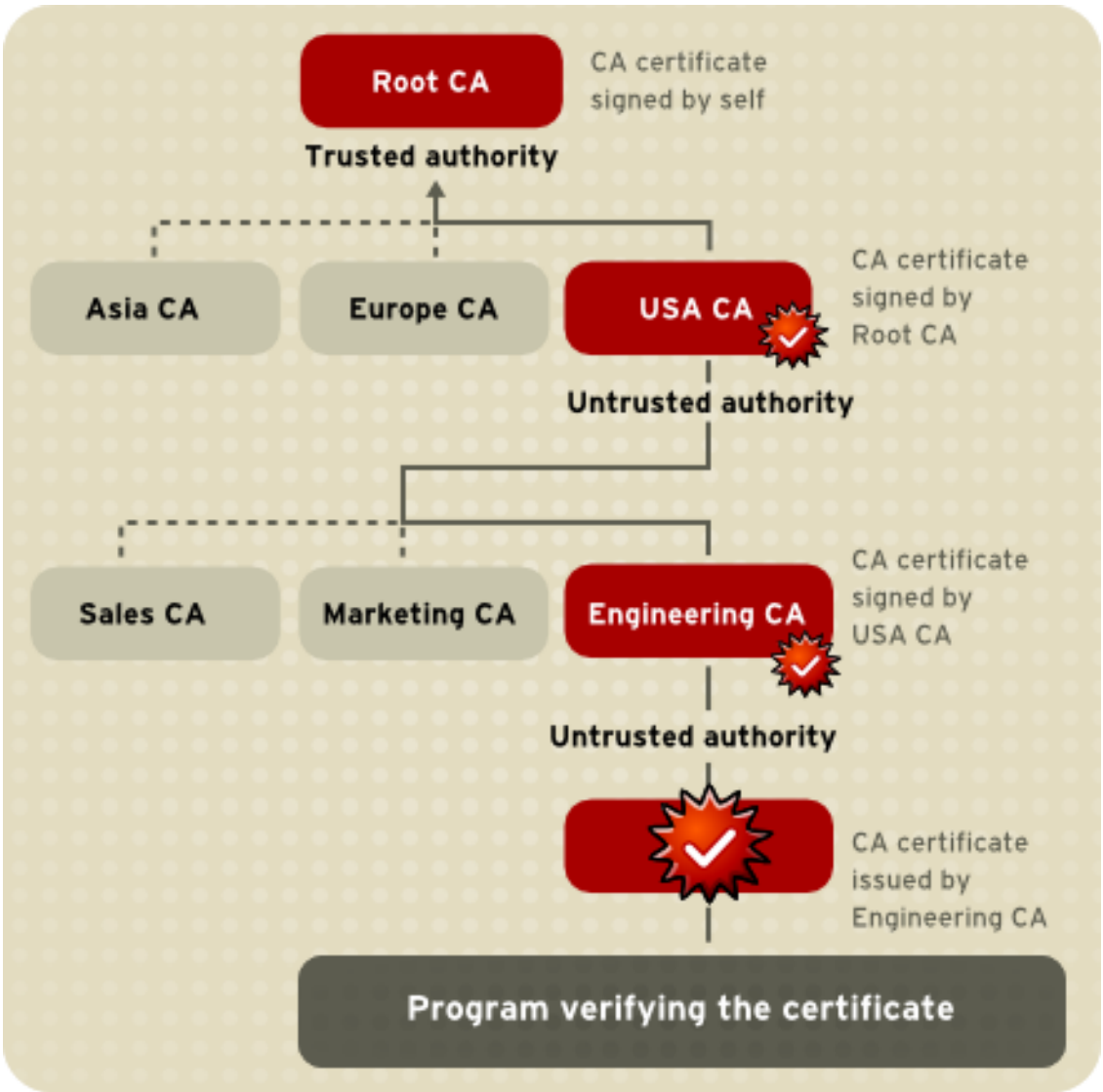
CA には、ルート CA によって署名された CA 証明書があります。階層内の下位 CA の下にある CA には、上位レベルの下位 CA によって署名された CA 証明書があります。

組織は、CA 階層の設定方法に多くの柔軟性を備えています。図1.6「認証局の階層の例」 では、例を 1 つだけ示します。

1.3.6.2. 証明書チェーン

CA 階層は証明書チェーンに反映されます。*証明書チェーン*は、連続する CA により発行された一連の証明書です。図1.7「証明書チェーンの例」 は、図1.6「認証局の階層の例」 に示される CA 階層に基づいて、2 つの下位 CA 証明書を介してルート CA の CA 証明書へのエンティティーを識別する証明書からそれに続く証明書チェーンを示しています。

図1.7 証明書チェーンの例



証明書チェーンは、階層のブランチから階層のルートへの証明書のパスを追跡します。証明書チェーンでは、以下が発生します。

- 各証明書の後に、その発行者の証明書が追加されます。
- 各証明書には、その証明書の発行者の名前 (DN) が含まれており、チェーン内の次の証明書のサブジェクト名と同じです。

図1.7「証明書チェーンの例」 では、**エンジニアリング CA** 証明書には、その証明書を発行した CA (**USA CA**) の DN が含まれます。**USA CA** の DN はチェーン内の次の証明書のサブジェクト名でもあります。

- 各証明書は、発行者の秘密鍵で署名されます。この署名は、発行者の証明書内の公開鍵 (チェーン内の次の証明書) で検証できます。

  図1.7「証明書チェーンの例」 では、**USA CA** の証明書内の公開鍵を使用して、**エンジニアリング CA** に対して、**USA CA** の証明書のデジタル署名を検証できます。

1.3.6.3. 証明書チェーンの確認

証明書チェーンの検証では、特定の証明書チェーンが適切な形式で、有効で、適切に署名され、信頼できるものであることを確認します。以下のプロセスの説明は、認証用に提示される証明書から、証明書チェーンを形成および検証する最も重要な手順を示しています。

1. 証明書の有効期間は、検証者のシステムクロックによって提供される現在時刻に対してチェックされます。

2. 発行者の証明書が位置しています。ソースは、クライアントまたはサーバーのローカル証明書データベース、または SSL/TLS 接続と同様に、サブジェクトが提供した証明書チェーンのいずれかになります。

3. 証明書の署名は、発行者の証明書の公開鍵を使用して検証されます。

4. サービスのホスト名は、SAN (Subject Alternative Name) 拡張と照合されます。証明書にそのような拡張がない場合、ホスト名はサブジェクトの CN に対して比較されます。

5. システムは、証明書の基本制約要件、つまり、証明書が CA であるかどうか、および署名が許可されている子会社の数を確認します。

6. 発行者の証明書が検証者の証明書データベース内の検証者によって信頼されている場合、検証はここで正常に停止します。それ以外の場合は、発行者の証明書をチェックして、証明書タイプ拡張に適切な下位 CA 表示が含まれていることを確認し、チェーン検証をこの新しい証明書からやり直します。図1.8「ルート CA への証明書チェーンの確認」 は、このプロセスの例を示しています。

図1.8 ルート CA への証明書チェーンの確認

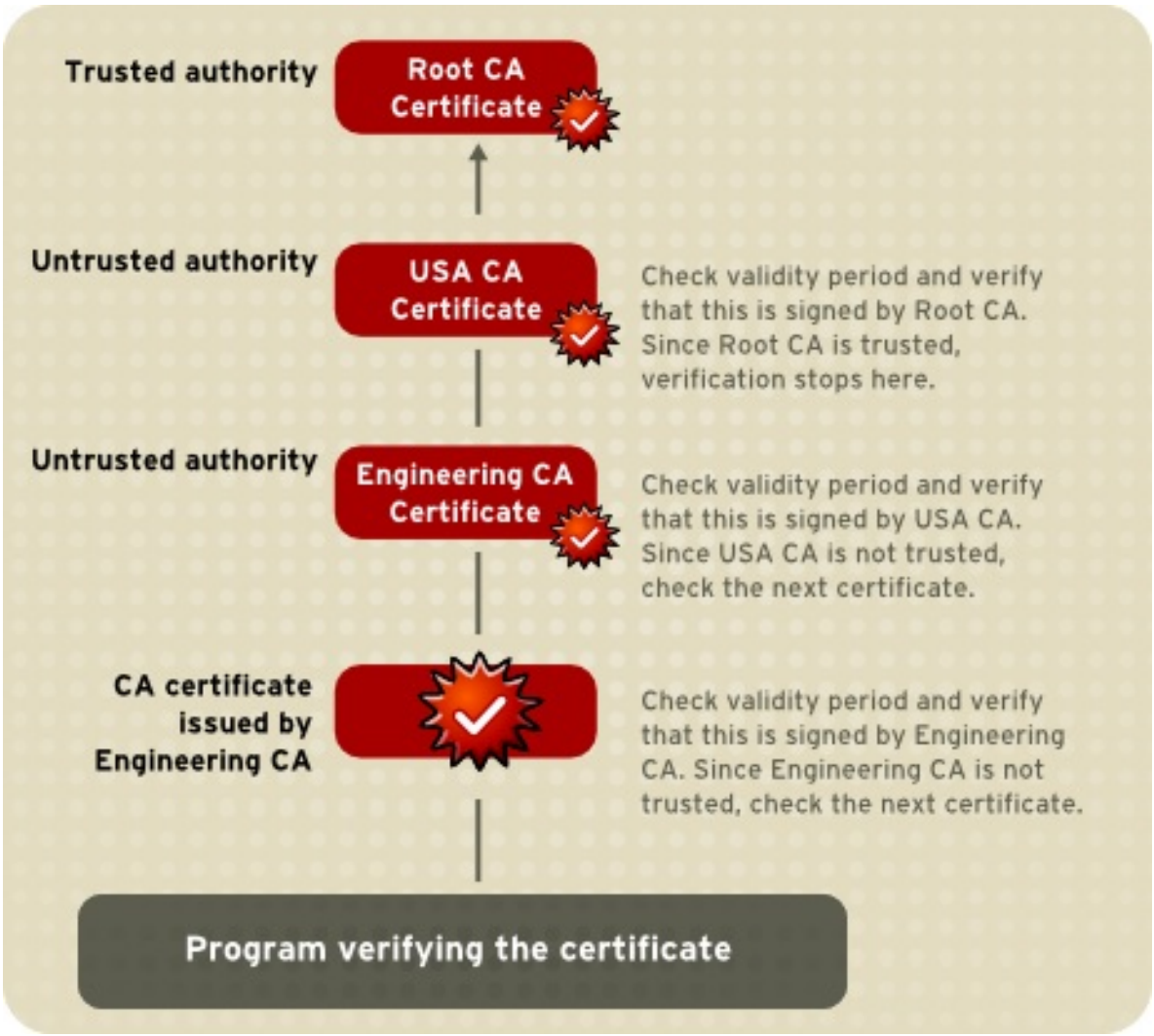
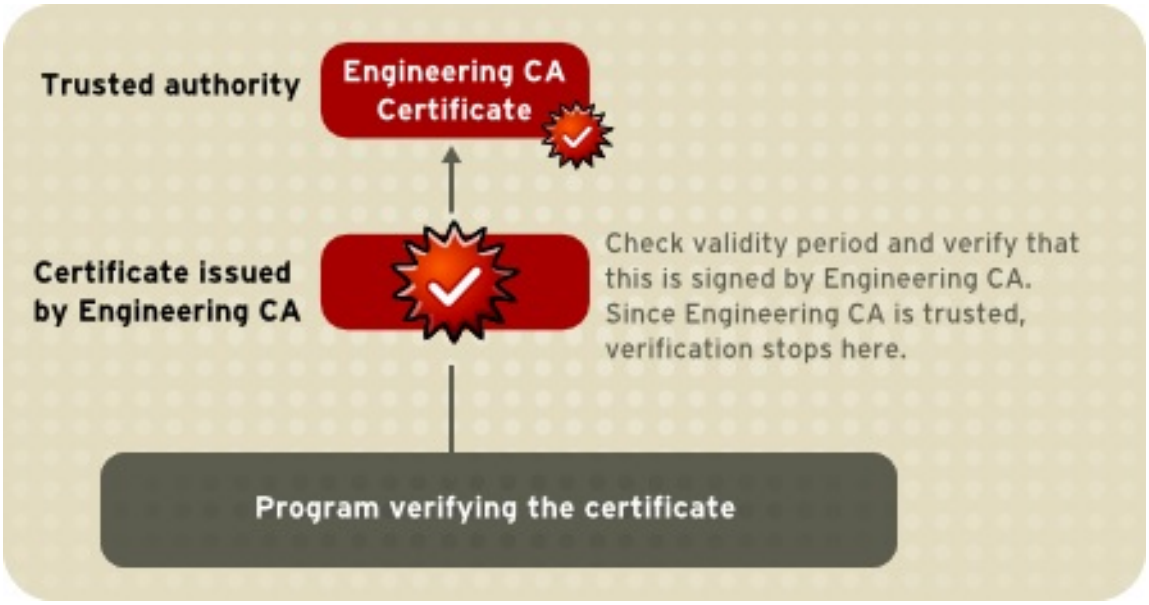


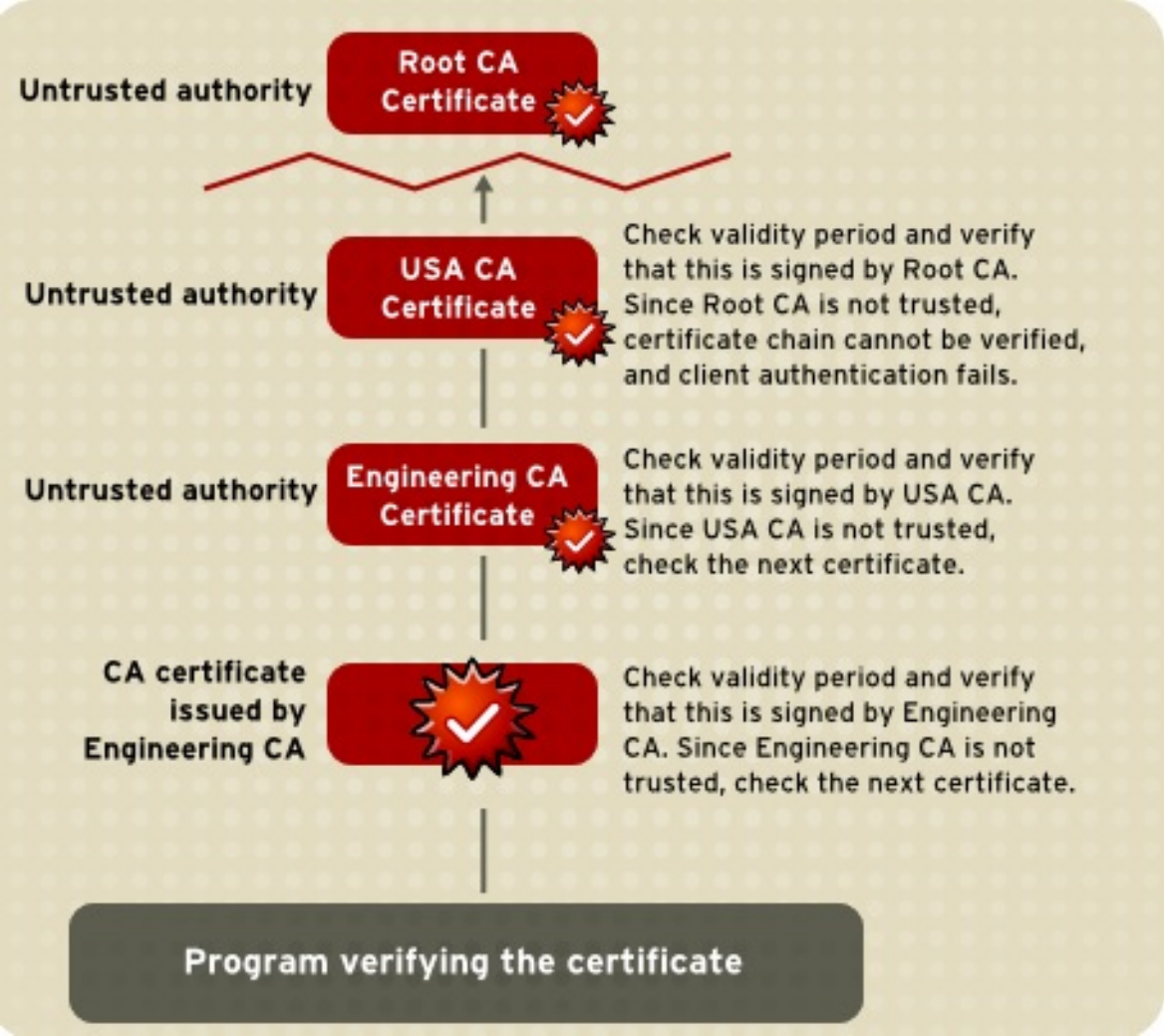
図1.8「ルート CA への証明書チェーンの確認」 は、ルート CA のみが検証機能のローカルデータベースに含まれる場合に何が発生するかを示しています。**エンジニアリング CA** など、中間 CA のいずれかの証明書が検証者のローカルデータベースにある場合は、図1.9「証明書 Chain の中間 CA の確認」 に示されているように、検証はその証明書で停止します。

図1.9 証明書 Chain の中間 CA の確認



有効期限が切れている、署名が無効である、または証明書チェーンのいずれかの時点で発行元 CA の証明書がない場合は、認証が失敗します。図1.10「検証できない証明書チェーン」 は、root CA 証明書も中間 CA 証明書も検証者のローカルデータベースに含まれていない場合に、検証がどのように失敗するかを示しています。

図1.10 検証できない証明書チェーン



1.3.7. 証明書の状態

証明書失効リスト (CRL) の詳細は、 「CRL」を参照してください。

オンライン証明書ステータスプロトコル (OCSP) の詳細は、 「OCSP サービス」を参照してください。

1.4. 証明書のライフサイクル

証明書は、Web サイトへのアクセスに電子メールを暗号化することによって、多くのアプリケーションで使用されます。証明書のライフサイクルには、2 つの主要ステージがあります。発行時 (発行と登録) と、証明書が有効になっていない (更新または失効) 期間です。また、サイクル時に証明書を管理する方法もあります。他のアプリケーションで利用できる証明書に関する情報を **公開** し、キーペアのバックアップを作成して、証明書が失われた場合に証明書を復旧できるようにします。

1.4.1. 証明書の発行

証明書を発行するプロセスは、証明書を発行する CA と、それが使用する目的によって異なります。非デジタル形式の ID の発行も、同様の方法で異なります。ライブラリーカードを取得する要件は、ドラ

イバーのライセンスを取得する方法とは異なります。同様に、異なる CA には、さまざまな種類の証明書を発行するためのさまざまな手順があります。証明書を受け取るための要件は、認証された文書の電子メールアドレスまたはユーザー名とパスワード、身元調査、および個人面接のように単純な場合があります。

組織のポリシーに応じて、証明書を発行するプロセスは、ユーザーに対して完全に透過的であるものから、ユーザーの多大な参加と複雑な手順を要求するものまでさまざまです。一般に、証明書を発行するプロセスは柔軟である必要があります。これにより、組織は変化するニーズに合わせて証明書を調整できます。

1.4.2. 証明書の有効期限および更新

証明書のライセンスと同様、証明書は、有効である期間を指定します。有効期間の前後に認証に証明書を使用しようとすると失敗します。証明書の有効期限および更新の管理は、証明書管理ストラテジーの重要な部分です。たとえば、管理者は、証明書の有効期限が近づいたときに自動的に通知を受け取り、システムの動作を中断することなく適切な更新プロセスを完了できるようにすることができます。更新プロセスでは、同じ公開鍵のペアを再使用するか、新しい鍵を発行できます。

さらに、従業員が会社を辞めたり、会社内の別の部署で新しい仕事に異動したりした場合など、証明書の有効期限が切れる前に証明書を取り消す必要がある場合があります。

証明書失効は複数の方法で処理できます。

証明書がディレクトリーに存在しているかどうかの確認

サーバーは、認証プロセスが提示されている証明書の存在についてディレクトリーをチェックするように設定できます。管理者が証明書を取り消すと、証明書はディレクトリーから自動的に削除され、証明書が他のすべての点で有効なままであっても、その証明書を使用した後続の認証試行は失敗します。

証明書失効リスト (CRL)

失効した証明書 (CRL) は、一定間隔でディレクトリーに公開できます。CRL は認証プロセスの一部として確認できます。

リアルタイムのステータスチェック

認証用に証明書が提示されるたびに、発行 CA を直接確認することもできます。この手順は、リアルタイムステータスチェックと呼ばれることもあります。

オンライン証明書ステータスプロトコル

OCSP (Online Certificate Status Protocol) サービスを設定して、証明書のステータスを確認できます。

証明書の更新に関する詳細は、「証明書の更新」を参照してください。CRL や OCSP を含む証明書の取り消しに関する詳細は、「証明書の取り消しとステータスの確認」を参照してください

1.5. キー管理

証明書を発行する前に、その証明書に含まれる公開鍵と、対応する秘密鍵を生成する必要があります。1 人の人間に、署名操作用の証明書と鍵のペアを発行し、暗号化操作用に別の証明書と鍵のペアを発行するのが便利な場合もあります。個別の署名証明書と暗号化証明書は、秘密署名キーをローカルマシン上にのみ保持し、否認防止を最大限に提供します。これは、ユーザーが元のキーを紛失したり会社を辞めたりした場合に、秘密暗号化キーを取得できる中央の場所にバックアップするのにも役立ちます。

鍵はクライアントソフトウェアにより生成したり、CA が集中して生成されたり、LDAP ディレクトリーを介してユーザーに配布したりできます。どちらの方法にも関わるコストが発生します。ローカルキーの生成により、否認防止が最大になりますが、発行プロセスへのユーザーの参加が増える可能性があります。柔軟なキー管理機能は、ほとんどの組織にとって不可欠です。

鍵リカバリー、または、定義した条件下で暗号鍵のバックアップを取得する機能は、組織が証明書を使用する方法に応じて、証明書管理に不可欠であることがあります。一部の PKI 設定では、暗号化キーを回復する前に、許可された状況で正当な所有者にのみキーが回復されるように、複数の許可された担当者が同意する必要があります。情報を暗号化するときに鍵を復元する必要がある場合があります。また、失われたキーのみが復号化できます。

第2章 RED HAT CERTIFICATE SYSTEM の概要

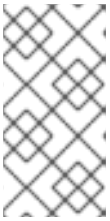
証明書の発行、更新、取り消しなど、すべての一般的な PKI 操作。キーのアーカイブと回復。CRL の公開と証明書ステータスの検証は、Red Hat Certificate System 内の相互運用サブシステムによって実行されます。この章では、個々のサブシステムの機能と、それらが連携して堅牢でローカルな PKI を確立する方法を説明します。

2.1. CERTIFICATE SYSTEM サブシステムのレビュー

Red Hat Certificate System は 5 つの異なるサブシステムを提供します。それぞれは、PKI デプロイメントのさまざまな側面に重点を置いています。

- **Certificate Manager** と呼ばれる **認証局**。CA は PKI の中核で、すべての証明書を発行して取り消します。Certificate Manager は、Certificate System の中核でもあります。信頼できるサブシステムの **セキュリティードメイン** を確立することで、別のサブシステム間の関係を確立し、管理します。

- **キーリカバリー認証局** (KRA)。証明書は、特定のキーペアと一意の鍵のペアに基づいて作成されます。秘密鍵が失われると、その鍵がアクセスに使用されたデータ (暗号化された電子メールなど) にもアクセスできないため、失われます。KRA は鍵のペアを格納するため、復元された鍵に基づいて新しい同一証明書を生成でき、秘密鍵が損失または破損してもすべての暗号化されたデータにアクセスできます。

  注記

  以前のバージョンの Certificate System では、KRA はデータリカバリーマネージャー (DRM) とも呼ばれます。コード、設定ファイルエントリー、Web パネルなどの一部のリソースは、KRA ではなく DRM という用語を使用する場合があります。

- **オンライン証明書ステータスプロトコル** (OCSP) レスポンダーOCSP は、証明書が有効かどうかを検証し、有効期限が切れていないかどうかを確認します。この機能は、内部 OCSP サービスがある CA からも実行できますが、外部の OCSP レスポンダーを使用すると、発行 CA の負荷が低くなります。

- **トークンキーサービス** (TKS)。TKS は、トークン CCID、プライベート情報、および定義済みのアルゴリズムに基づいて鍵を取得します。これらの派生キーは、TPS によりトークンのフォーマットに使用され、トークンに証明書を登録します。

- **トークン処理システム** (TPS)。TPS は、スマートカードなどの外部トークンと直接対話し、ローカルクライアント (Enterprise Security Client (ESC)) を使用してこれらのトークンの鍵と証明書を管理します。トークン操作がある場合には TPS に問い合わせ、TPS は必要に応じて CA、KRA、または TKS と対話し、Enterprise Security Client の方法で情報をトークンに送信します。

すべてのサブシステムがインストールされている場合でも、Certificate System のコアは最終的に証明書関連のすべての要求を処理するため、CA になります。その他のサブシステムは、wheel のスポークのように CA に接続します。これらのサブシステムは連携して、公開鍵インフラストラクチャー (PKI) を作成します。インストールされているサブシステムに応じて、PKI は 2 つの方法の 1 つ (または両方) で機能できます。

- スマートカードを管理する **トークン管理システム** または **TMS** 環境。これには CA、TKS、および TPS が必要です。サーバー側の鍵生成には任意の KRA が必要です。

- 通常、ソフトウェアデータベースでスマートカード以外の環境で使用される証明書を管理する

従来の **非トークン管理システム** または **非 TMS** 環境です。少なくとも、TMS 以外の環境では CA のみが必要ですが、TMS 以外の環境では OCSP レスポンダーと KRA インスタンスも使用できます。

2.2. CERTIFICATE SYSTEM サブシステムの概要

2.2.1. 個別インスタンスと共有インスタンス

Red Hat Certificate System は、すべてのサブシステムに個別の PKI インスタンスのデプロイメントをサポートします。

- 個別の PKI インスタンスは、単一の Java ベースの Apache Tomcat インスタンスとして実行されます。
- 個別の PKI インスタンスには、単一の PKI サブシステム (CA、KRA、OCSP、TKS、または TPS) が含まれます。
- 同じ物理マシンまたは仮想マシン (VM) 上に共存する場合、別の PKI インスタンスは一意のポートを使用する必要があります。

または、Certificate System は、共有 PKI インスタンスのデプロイメントをサポートします。

- 共有 PKI インスタンスは、単一の Java ベースの Apache Tomcat インスタンスとしても実行されます。
- 単一の PKI サブシステムを含む共有 PKI インスタンスは、別の PKI インスタンスと同じです。
- 共有 PKI インスタンスには、各タイプの PKI サブシステムへの任意の組み合わせが含まれる可能性があります。
  - CA のみ
  - TKS のみ
  - CA および KRA
  - CA および OCSP
  - TKS および TPS
  - CA、KRA、TKS、および TPS
  - CA、KRA、OCSP、TKS、および TPS
  - など。
- 共有 PKI インスタンスを使用すると、そのインスタンスに含まれるサブシステムがすべて同じポートを共有できます。
- 共有 PKI インスタンスは、複数の同一物理マシンまたは仮想マシンに共存する場合、一意のポートを使用する必要があります。

2.2.2. インスタンスインストールの要件

2.2.2.1. Directory Server インスタンスの可用性

Certificate System インスタンスをインストールする前に、ローカルまたはリモートの Red Hat Directory Server LDAP インスタンスが利用できる必要があります。Red Hat Directory Server のインストール方法は、『Red Hat Directory Server インストールガイド』 を参照してください。

2.2.2.2. PKI パッケージ

Red Hat Certificate System は、以下のパッケージで設定されています。

- 以下のベースパッケージは Certificate System のコアを形成し、ベースの Red Hat Enterprise Linux リポジトリーで利用できます。
  - pki-core
    - pki-base
    - pki-base-java
    - pki-ca
    - pki-javadoc
    - pki-kra
    - pki-server
    - pki-symkey
    - pki-tools
- 以下に記載するパッケージは、ベースの Red Hat Enterprise Linux サブスクリプションチャンネルでは利用 **できません**。これらのパッケージをインストールするには、Red Hat Certificate System サブスクリプションプールを割り当て、RHCS リポジトリーを有効にする必要があります。詳細は、「Red Hat サブスクリプションの添付および Certificate System パッケージリポジトリーの有効化」 を参照してください。
  - pki-core
    - pki-console
    - pki-ocsp
    - pki-tks
    - pki-tps
  - redhat-pki
    - redhat-pki: pki-core モジュールのパッケージをすべて含めます。redhat-pki パッケージを個別に選択する場合は、pki-core モジュールを無効にすることを推奨します。
    - redhat-pki-console-theme
    - redhat-pki-server-theme

Red Hat Enterprise Linux 8 システムを使用 (必要に応じて *4章 サポート対象のプラットフォーム* に記載されているサポート対象のハードウェアセキュリティーモジュールで設定されているものを使用) して、Red Hat Certificate System をインストールする前に、すべてのパッケージが最新の状態であるこ

とを確認します。

(pki-javadocを除く) すべての Certificate System パッケージをインストールするには、**dnf** を使用して redhat-pki メタパッケージをインストールします。

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

必要に応じて 1 つ以上の最上位の PKI サブシステムパッケージをインストールしてください。実際のパッケージ名については、上記の一覧を参照してください。このアプローチを使用する場合は、必ず redhat-pki-server-theme パッケージもインストールし、必要に応じて redhat-pki-console-theme および pki-console で PKI コンソールを使用するようにしてください。

最後に、開発者および管理者は JSS および PKI javadoc (jss-javadoc および pki-javadoc) をインストールすることもできます。

注記

jss-javadoc パッケージでは、**Subscription Manager** で Server-Optional リポジトリーを有効にする必要があります。

2.2.2.3. インスタンスのインストールと設定

pkispawn コマンドラインツールを使用して、新しい PKI インスタンスをインストールおよび設定します。個別インストールと設定手順がなく、バッチプロセスとして、または両方の組み合わせ (パスワードを要求するバッチプロセス) のいずれかを対話的に実行することができます。このユーティリティーは、ブラウザーベースのグラフィカルインターフェイスをインストールまたは設定する方法を提供していません。

使用方法は、**pkispawn --help** コマンドを使用します。

pkispawn コマンド:

1. プレーンテキストの設定ファイル (**/etc/pki/default.cfg**) からデフォルトの **name=value** ペアで読み取ります。
2. 指定されたペアを対話的にまたは自動的に上書きし、最終結果を Python ディクショナリーとして保存します。
3. 順序付けされた *スクリプトレット* を実行し、サブシステムおよびインスタンスインストールを実行します。
4. 設定スクリプトレットは、Python ディクショナリーを JavaScript Object Notation (JSON) データオブジェクトとしてパッケージ化し、Java ベースの設定サーブレットに渡されます。
5. 設定サーブレットはこのデータを使用して新しい PKI サブシステムを設定し、制御を **pkispawn** 実行可能ファイルに渡して、PKI 設定の最終処理を行います。最終的なデプロイメントファイルのコピーは **/var/lib/pki/*instance_name*/*<subsystem>*/registry/*<subsystem>*/deployment.cfg** に保存されます。

詳細は、**pkispawn** の man ページを参照してください。

デフォルトの設定ファイル **/etc/pki/default.cfg** は、上記のプロセスの開始時に読み込まれるデフォルトのインストールと設定値を含むプレーンテキストのファイルです。これは、**DEFAULT**、**Tomcat**、**CA**、**KRA**、**OCSP**、**TKS**、および **TPS** セクションに分割された **name=value** ペアで設定されます。

pkispawn で **-s** オプションを使用してサブシステム名を指定すると、そのサブシステムのセクションのみが読み取られます。

セクションには階層があります。サブシステムセクションで指定した **name=value** のペアは、**[Tomcat]** セクションのペアを上書きし、**[DEFAULT]** セクションのペアを上書きします。デフォルトのペアは、インタラクティブ入力か、指定された PKI インスタンス設定ファイル内のペアによってさらに上書きできます。

注記

非対話形式ファイルを使用してデフォルトの **name=value** ペアを上書きする場合は常に、任意の場所に保存され、いつでも指定できます。これらのファイルは、**pkispawn** の man ページにある **myconfig.txt** や、**.ini** ファイル、またはより一般的には PKI インスタンス設定オーバーライドファイルとも呼ばれます。

詳細は、**pki_default.cfg** の man ページを参照してください。

設定サーブレット は、**/usr/share/java/pki/pki-certsrv.jar** を **com/netscape/certsrv/system/ConfigurationRequest.class** として保存されている Java バイトコードで設定されます。サーブレットは、**pkispawn** を使用して設定スクリプトレットから JSON オブジェクトとして渡されるデータを処理し、**com/netscape/certsrv/system/ConfigurationResponse.class** と同じファイルで提供される Java バイトコードを使用して **pkispawn** に戻ります。

対話型インストールの例では、**root** 権限で、コマンドラインで **pkispawn** コマンドを実行するだけです。

```
# pkispawn
```

重要

現在、インタラクティブなインストールは、非常に基本的なデプロイメントでのみ存在します。たとえば、クローン作成、Elliptic Curve Cryptography (ECC)、外部 CA、Hardware Security Module (HSM)、下位 CA などの高度な機能を使用する場合、デプロイメントは個別の設定ファイルで必要なオーバーライドパラメーターを提供する必要があります。

非対話的なインストールでは、PKI インスタンス設定の上書きファイルが必要ですが、プロセスは以下の例のようになります。

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

2. **vim** などのテキストエディターを使用して、以下の内容を含む **/root/pki/ca.cfg** という名前の設定ファイルを作成します。

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

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

さまざまな設定例は、**pkispawn** の man ページを参照してください。

2.2.2.4. インスタンスの削除

既存の PKI インスタンスを削除するには、**pkidestroy** コマンドを使用します。対話的に実行することも、バッチプロセスとして実行することもできます。**pkidestroy -h** を使用して、コマンドラインで詳細な使用方法を表示します。

pkidestroy コマンドは、サブシステムの作成時に保存された PKI サブシステムデプロイメント設定ファイル (**/var/lib/pki/*instance_name*/*<subsystem>*/registry/*<subsystem>*/deployment.cfg**) で読み込んで、読み込んだファイルを使用して PKI サブシステムを削除してから、追加のサブシステムがない場合は PKI インスタンスを削除します。詳細は、**pkidestroy** の man ページを参照してください。

pkidestroy を使用したインタラクティブな削除手順は、以下のようになります。

```
# 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.
```

非対話的な削除手順は、以下の例のようになります。

```
# 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. 実行管理 (systemctl)

2.2.3.1. 起動、停止、再起動、およびステータスの取得

Red Hat Certificate System サブシステムインスタンスは、Red Hat Enterprise Linux 8 で **systemctl** 実行管理システムツールを使用して停止および起動できます。

```
# 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> には、以下のいずれかの値を使用できます。

-

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

watchdog サービスの詳細は、「パスワードおよびウォッチドッグ (nuxwdog)」 および『Red Hat Certificate System 管理ガイド』の『Certificate System Watchdog Service の使用 』セクションを参照してください。

注記

RHCS 10 では、これらの **systemctl** アクションは **pki-server** エイリアスをサポートします。**pki-server <command> subsystem_instance_name** は **systemctl <command> pki-tomcatd@<instance>.service** のエイリアスです。

2.2.3.2. インスタンスの自動起動

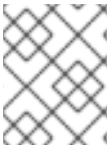
Red Hat Enterprise Linux の **systemctl** ユーティリティーは、サーバー上の各プロセスの自動起動およびシャットダウン設定を管理します。これは、システムが再起動すると、一部のサービスを自動的に再起動できることを意味します。システムユニットファイルは、サービスの起動を制御し、サービスが正しい順序で起動されるようにします。**systemd** サービスと **systemctl** ユーティリティーは、Red Hat Enterprise Linux 8 の 『基本的なシステム設定』 ガイドで説明されています。

Certificate System インスタンスは **systemctl** で管理できます。したがって、このユーティリティーは、インスタンスを自動的に再起動するかどうかを設定できます。Certificate System インスタンスが作成されると、システムの起動時に有効になります。これは **systemctl** を使用することで変更できます。

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

インスタンスを再度有効にするには、以下を実行します。

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



注記

systemctl enable コマンドおよび **systemctl disable** コマンドは、証明書システムをすぐに起動したり、停止したりしません。

2.2.4. プロセス管理 (pki-server および pkidaemon)

2.2.4.1. pki-server コマンドラインツール

Red Hat Certificate System の他のプロセス管理ツールは、**pki-server** です。使用方法は、**pki-server --help** コマンドを使用して、**pki-server** の man ページを参照してください。

pki-server コマンドラインインターフェイス (CLI) は、ローカルサーバーインスタンス (サーバー設定やシステム証明書など) を管理します。以下のように CLI を起動します。

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

CLI はサーバーインスタンスの設定ファイルおよび NSS データベースを使用するため、CLI では事前に初期化は必要ありません。CLI はファイルに直接アクセスできるため、これは root ユーザーがのみ実行でき、クライアント証明書は必要ありません。また、CLI はサーバーのステータスに関係なく実行できます。実行中のサーバーは必要ありません。

CLI は、階層構造で多数のコマンドをサポートしています。トップレベルのコマンドを一覧表示するには、追加のコマンドまたはパラメーターを指定せずに CLI を実行します。

```
$ pki-server
```

コマンドにはサブコマンドがあります。それらを一覧表示するには、コマンド名を指定し、追加オプションを指定せずに CLI を実行します。以下に例を示します。

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

コマンドの使用情報を表示するには **--help** オプションを使用します。

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

2.2.4.2. pki-server を使用したインストール済みサブシステムの有効化および無効化

インストール済みのサブシステムを有効または無効にするには、**pki-server** ユーティリティーを使用します。

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

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

subsystem_id を、有効なサブシステム識別子 (**ca**、**kra**、**tks**、**ocsp** または **tps**) に置き換えます。

注記

1 つのインスタンスにはサブシステムのタイプのみを設定できます。

たとえば、**pki-tomcat** という名前のインスタンスで OCSP サブシステムを無効にするには、次のコマンドを実行します。

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

インスタンスのインストールされたサブシステムを一覧表示するには、以下を実行します。

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

特定のサブシステムの状態を表示するには、次のコマンドを実行します。

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

2.2.4.3. pkidaemon コマンドラインツール

Red Hat Certificate System 用の別のプロセス管理ツールは、**pkidaemon** ツールです。

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

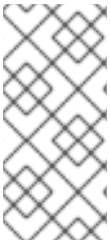
- **pkidaemon status tomcat**: システム上のすべての PKI インスタンスの PKI サブシステムのオン/オフ、ポート、URL などのステータス情報を提供します。
- **pkidaemon status tomcat*instance_name***: 特定のインスタンスの各 PKI サブシステムのオン/オフ、ポート、URL などのステータス情報を提供します。
- **pkidaemon start tomcat *instance_name*.service** - **systemctl** を内部で使用します。

詳細は、**pkidaemon** の man ページを参照してください。

2.2.4.4. サブシステムの Web サービス URL の検索

CA、KRA、OCSP、TKS、および TPS サブシステムには、エージェント用の Web サービスページと、通常のユーザーおよび管理者が含まれます。これらの Web サービスには、サブシステムのセキュアなユーザーのポート上でサブシステムホストに URL を開くとアクセスできます。CA の場合の例を以下に示します。

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



注記

インスタンスのインターフェイス、URL、およびポートの全一覧を取得するには、サービスのステータスを確認します。以下に例を示します。

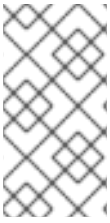
```
pkidaemon status instance_name
```

各サブシステムの主な Web サービスページには、利用可能なサービスページの一覧があります。これらは 表2.1「デフォルトの Web サービスページ」 で要約されています。特定のサービスにアクセスするには、適切なポートにアクセスし、適切なディレクトリーを URL に追加します。たとえば、CA のエンドエンティティー (通常のユーザー) の Web サービスにアクセスするには、以下を実行します。

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

DNS が設定されていない場合は、IPv4 アドレスまたは IPv6 アドレスを使用して、サービスページに接続できます。以下に例を示します。

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



注記

すべてのユーザーがサブシステムのエンドユーザーページにアクセスできます。ただし、エージェントまたは管理 Web サービスページにアクセスするには、エージェントまたは管理者の証明書を Web ブラウザーにインストールする必要があります。それ以外の場合は、Web サービスへの認証に失敗します。

表2.1 デフォルトの Web サービスページ

ポート	SSL/TLS に使用	クライアント認証に使用[a]	Web サービス	Web サービスの場所
Certificate Manager				
8080	いいえ		エンドエンティティー	ca/ee/ca
8443	はい	いいえ	エンドエンティティー	ca/ee/ca
8443	はい	はい	エージェント	ca/agent/ca
8443	はい	いいえ	サービス	ca/services
8443	はい	いいえ	コンソール	pkiconsole https://*host:port*/ca
キーリカバリー認証局				
8080	いいえ		エンドエンティティー[b]	kra/ee/kra
8443	はい	いいえ	エンドエンティティー[b]	kra/ee/kra
8443	はい	はい	エージェント	kra/agent/kra
8443	はい	いいえ	サービス	kra/services
8443	はい	いいえ	コンソール	pkiconsole https://*host:port*/kra
オンライン証明書ステータスマネージャー				
8080	いいえ		エンドエンティティー[c]	ocsp/ee/ocsp
8443	はい	いいえ	エンドエンティティー[c]	ocsp/ee/ocsp
8443	はい	はい	エージェント	ocsp/agent/ocsp

ポート	SSL/TLS に使用	クライアント認証に使用[a]	Web サービス	Web サービスの場所
8443	はい	いいえ	サービス	ocsp/services
8443	はい	いいえ	コンソール	pkiconsole https://*host:port*/ocsp
トークンキーサービス				
8080	いいえ		エンドエンティティー[b]	tks/ee/tks
8443	はい	いいえ	エンドエンティティー[b]	tks/ee/tks
8443	はい	はい	エージェント	tks/agent/tks
8443	はい	いいえ	サービス	tks/services
8443	はい	いいえ	コンソール	pkiconsole https://*host:port*/tks
トークン処理システム				
8080	いいえ		安全でないサービス	tps/tps
8443	はい		安全なサービス	tps/tps
8080	いいえ		Enterprise Security Client Phone Home	tps/phoneHome
8443	はい		Enterprise Security Client Phone Home	tps/phoneHome
8443	はい	はい	管理、エージェント、および Operator サービス[d]	tps/ui

ポート	SSL/TLS に使用	クライアント認証に使用[a]	Web サービス	Web サービスの場所

[a] クライアント認証値が **No** のサービスは、クライアント認証を要求するように再設定できます。**Yes** または **No** のいずれかの値を持たないサービスは、クライアント認証を使用するように設定することはできません。

[b] このサブシステムタイプにはエンドエンティティーポートとインターフェイスがありますが、他のエンドエンティティーサービスとは異なり、これらのエンドエンティティーサービスには Web ブラウザーからはアクセスできません。

[c] OCSP にはエンドエンティティーポートおよびインターフェイスがありますが、他のエンドクリティーサービスも Web ブラウザーを介してアクセスすることはできません。エンドユーザー OCSP サービスには、OCSP 要求を送信するクライアントがアクセスします。

[d] エージェント、管理者、および Operator サービスはすべて、同じ Web サービスページを介してアクセスされます。各ロールは、そのロールのメンバーにのみ表示される特定のセクションにのみアクセスできます。

2.2.4.5. Certificate System コンソールの起動

重要

このコンソールは非推奨になりました。

CA、KRA、OCSP、および TKS サブシステムには、管理機能を実行するための Java インターフェイスがあります。KRA、OCSP、および TKS では、ユーザーおよびグループのログ設定や管理など、非常に基本的なタスクが含まれます。CA の場合は、証明書プロファイルの作成や公開の設定など、その他の設定が含まれます。

pkiconsole ユーティリティーを使用して SSL/TLS ポート経由でサブシステムインスタンスに接続すると、コンソールが開きます。このユーティリティーは、以下の形式を使用します。

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

subsystem_type は **ca**、**kra**、**ocsp**、または **tks** にすることができます。たとえば、これにより KRA コンソールが開きます。

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

DNS が設定されていない場合は、IPv4 アドレスまたは IPv6 アドレスを使用してコンソールに接続できます。以下に例を示します。

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

2.3. CERTIFICATE SYSTEM のアーキテクチャーの概要

それぞれのサービスが異なるサービスを提供しますが、すべての RHCS サブシステム (CA、KRA、OCSP、TKS、TPS) が共通のアーキテクチャーを共有します。以下のアーキテクチャー図は、これらのすべてのサブシステムによって共有される共通のアーキテクチャーを示しています。

2.3.1. Java Application Server

Java アプリケーションサーバーは、サーバーアプリケーションを実行する Java フレームワークです。

Certificate System は、Java アプリケーションサーバー内で実行されるように作られています。現在、サポートされている唯一の Java アプリケーションサーバーは Tomcat 8 です。他のアプリケーションサーバーのサポートは今後追加される可能性があります。詳細は、http://tomcat.apache.org/ を参照してください。

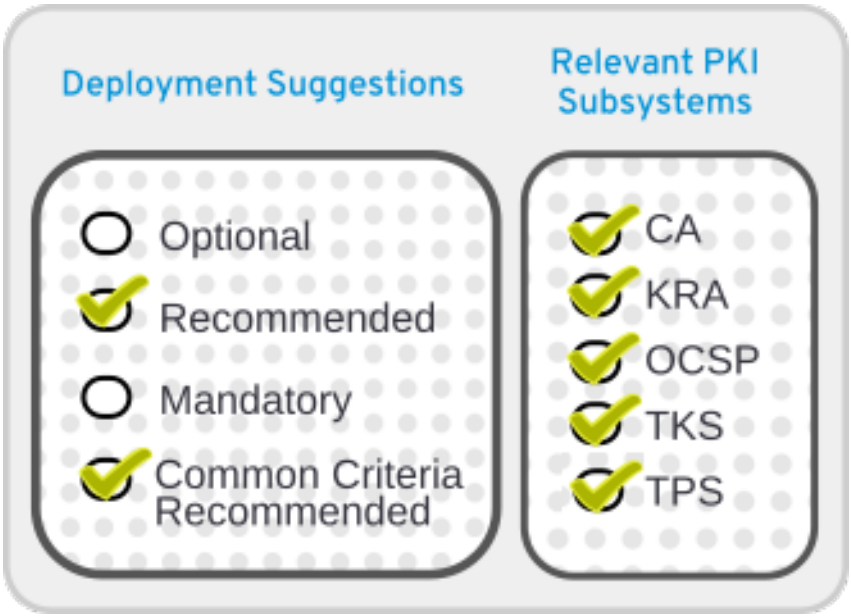
Certificate System の各インスタンスは Tomcat サーバーインスタンスです。Tomcat 設定は **server.xml** に保存されます。https://tomcat.apache.org/tomcat-8.0-doc/config/ では、Tomcat 設定の詳細情報が提供されています。

各 Certificate System サブシステム (CA や KRA など) は、Tomcat に Web アプリケーションとしてデプロイされます。Web アプリケーション設定は **web.xml** ファイルに保存され、Java Servlet 3.1 仕様で定義されます。詳細は https://www.jcp.org/en/jsr/detail?id=340 を参照してください。

Certificate System の設定自体は **CS.cfg** に保存されます。

これらのファイルの場所は、「インスタンスのレイアウト」 を参照してください。

2.3.2. Java Security Manager



Java サービスには、アプリケーションを実行するための安全でない操作と安全な操作を定義する Security Manager オプションがあります。サブシステムがインストールされると、Security Manager が自動的に有効になります。つまり、各 Tomcat インスタンスは Security Manager が実行されている状態で起動します。

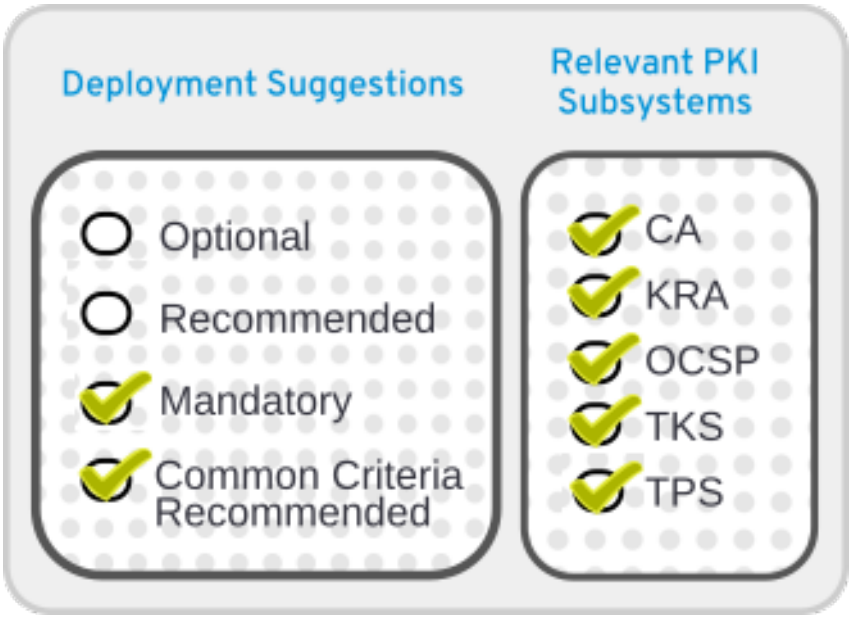
pkispawn を実行し、独自の Tomcat セクションで **pki_security_manager=false** オプションを指定するオーバーライド設定ファイルを使用すると、Security Manager が無効になります。

以下の手順に従って、インストールされたインスタンスから Security Manager を無効にすることができます。

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

   または (**nuxwdog** ウォッチドッグを使用している場合)

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

2. **/etc/sysconfig/*instance_name*** ファイルを開き、**SECURITY_MANAGER="false"** を設定します。

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

   または (**nuxwdog** ウォッチドッグを使用している場合)

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

インスタンスが起動または再起動すると、Java セキュリティーポリシーは、以下のファイルから **pkidaemon** により構築または再構築されます。

```
/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
```

次に、**/var/lib/pki/*instance_name*/conf/catalina.policy** に保存されます。

2.3.3. インターフェイス

2.3.3.1. サーブレットインターフェイス

各サブシステムには、サブシステムのさまざまな部分との対話を可能にするインターフェイスが含まれています。すべてのサブシステムが共通の管理インターフェイスを共有し、エージェントに割り当てられたタスクを実行するエージェントインターフェイスがあります。CA サブシステムには、エンドエンティティーを PKI に登録することができるエンドエンティティーのインターフェイスがあります。OCSP Responder サブシステムには、エンドエンティティーとアプリケーションが現在の証明書失効ステータスをチェックできるエンドエンティティーのインターフェイスがあります。最後に、TPS には Operator インターフェイスがあります。

アプリケーションサーバーは接続エントリーポイントを提供しますが、Certificate System は各インターフェイスに固有のサーブレットを提供してインターフェイスを完了します。

各サブシステムのサーブレットは、対応する **web.xml** ファイルで定義されます。同じファイルは各サーブレットの URL と、サーブレットにアクセスするセキュリティー要件も定義します。詳細は、「Java Application Server」を参照してください。

2.3.3.2. 管理インターフェイス
```

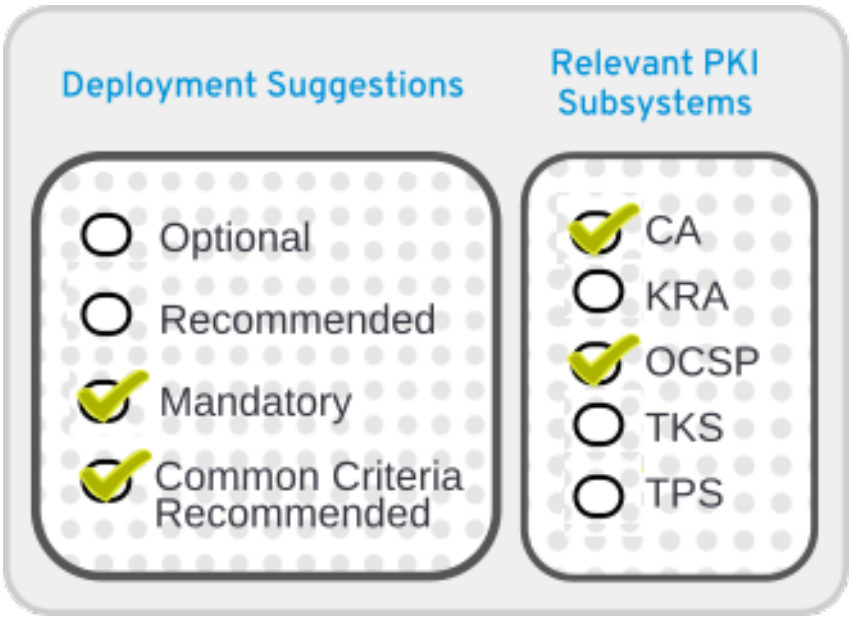


エージェントインターフェイスは、エージェントのエントリーポイントからの HTML フォームの送信を処理する Java サーブレットを提供します。エージェントサーブレットは、各フォーム送信で提供された情報に基づいて、証明書の承認、証明書の更新、証明書の失効の要求の編集と承認、証明書プロファイルの承認などのエージェントタスクを実行できるようにします。KRA サブシステムまたは TKS サブシステムのエージェントインターフェイス、または OCSP Responder はサブシステムに固有のものです。

非 TMS セットアップでは、エージェントインターフェイスは、CA から KRA への信頼できる接続の CIMC 間境界通信にも使用されます。この接続は SSL クライアント認証によって保護され、*Trusted Manager* と呼ばれる信頼されている個別のロールによって区別されます。エージェントロールと同様、Trusted Manager (CIMC 間境界接続専用に作成された疑似ユーザー) は、SSL クライアント認証を受ける必要があります。ただし、エージェントのロールとは異なり、エージェント機能は提供されません。

TMS 設定では、CIMC 間境界通信は、TPS から CA、TPS から KRA、および TPS から TKS に送信されます。

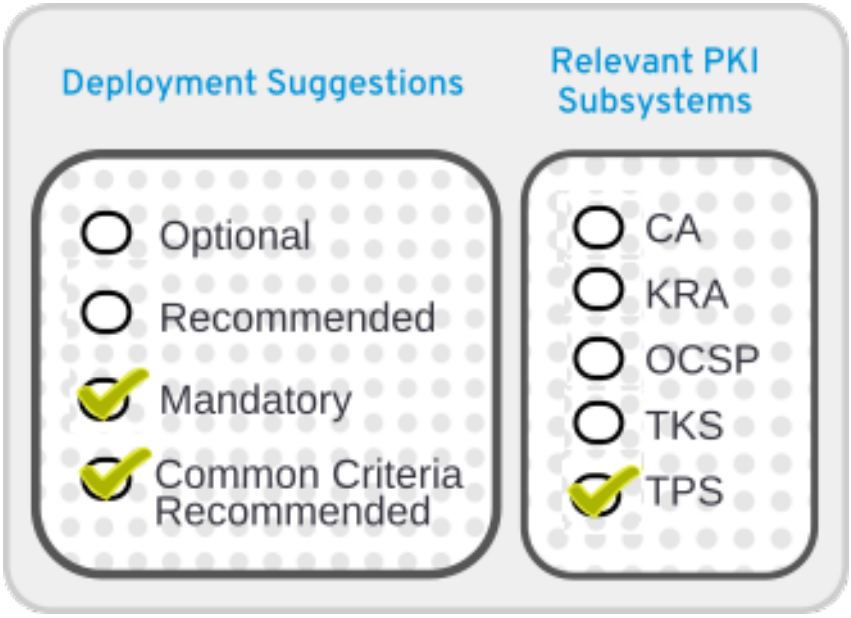
2.3.3.3. エンドエンティティーインターフェイス



CA サブシステムでは、エンドエンティティーのインターフェイスは、エンドエンティティーエントリーポイントからの HTML フォームの送信を処理する Java サーブレットを提供します。フォーム送信

から受け取った情報に基づいて、エンドエンティティーサーブレットにより、エンドエンティティーが証明書を登録、更新、独自の証明書を取り消します。OCSP Responder サブシステムの End-Entity インターフェイスは、OCSP リクエストを許可および処理するための Java サーブレットを提供します。KRA、TKS、および TPS サブシステムは、End-Entity サービスを提供しません。

2.3.3.4. Operator インターフェイス



オペレーターインターフェイスは、TPS サブシステムにのみあります。

2.3.4. REST インターフェイス

Hitsal state transfer (REST) は、HTTP を使用して Web サービスを定義し、整理する手段で、他のアプリケーションとの相互運用性を単純化します。Red Hat Certificate System は、サーバーのさまざまなサービスにアクセスするための REST インターフェイスを提供します。

Red Hat Certificate System の REST サービスは、RESTEasy フレームワークを使用して実装されます。RESTEasy は実際には Web アプリケーションのサーブレットとして実行されるため、RESTEasy の設定は、対応するサブシステムの **web.xml** にあります。RESTEasy の詳細は、http://resteasy.jboss.org/ を参照してください。

各 REST サービスは、個別の URL として定義されます。以下に例を示します。

- CA 証明書サービス: **http://*<host_name>*:*<port>*/ca/rest/certs/**
- KRA キーサービス: **http://*<host_name>*:*<port>*/kra/rest/agent/keys/**
- TKS ユーザーサービス: **http://*<host_name>*:*<port>*/tks/rest/admin/users/**
- TPS グループサービス: **http://*<host_name>*:*<port>*/tps/rest/admin/groups/**

一部のサービスは、プレーンの HTTP 接続を使用してアクセスできますが、セキュリティーに HTTPS 接続が必要になる場合があります。

REST 操作は HTTP メソッドとして指定されます (たとえば、GET、PUT、POST、DELETE)。たとえば、クライアントが GET /ca/rest/users リクエストを送信する CA ユーザーを取得する場合は、次のコマンドを実行します。

REST リクエストおよび応答メッセージは、XML または JSON 形式で送信できます。以下に例を示します。

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

CLI、Web UI、または汎用 REST クライアントなどのツールを使用して、REST インターフェイスにアクセスできます。Certificate System は、プログラムでサービスにアクセスするための Java、Python、および JavaScript ライブラリーも提供します。

REST インターフェイスは、2 種類の認証方法をサポートします。

- ユーザー名およびパスワード
- クライアント証明書

各サービスに必要な認証方法は、**/usr/share/pki/ca/conf/auth-method.properties** で定義されます。

REST インターフェイスでは、サービスへのアクセスに特定のパーミッションが必要になる場合があります。パーミッションは LDAP の ACL リソースで定義されます。REST インターフェイスは、**/usr/share/pki/*<subsystem>*/conf/acl.properties** の ACL リソースにマッピングされます。

REST インターフェイスの詳細は、http://www.dogtagpki.org/wiki/REST を参照してください。

2.3.5. JSS

Java Security Services (JSS) は、NSS が実行する暗号化操作の Java インターフェイスを提供します。Certificate System アーキテクチャーの JSS 以降は、Java Virtual Machine (JVM) 内からネイティブシステムライブラリーへのアクセスを提供する Java Native Interface (JNI) で構築されます。この設計により、システムの一部として配布される NSS などの FIPS で承認された暗号化プロバイダーを使用できます。JSS は、NSS でサポートされるセキュリティー標準および暗号化技術の多くに対応しています。JSS は、ASN.1 タイプと BER-DER エンコードの純粋な Java インターフェイスも提供します。

2.3.6. Tomcatjss

Red Hat Certificate System の Java ベースのサブシステムは、Tomcat Server HTTP エンジンと JSS 間のブリッジとして **tomcatjss** と呼ばれる単一の JAR ファイルを使用します。これは、NSS が実行するセキュリティー操作の Java インターフェイスです。**Tomcatjss** は、Tomcat の Java Security Services (JSS) を使用した Java Secure Socket Extension (JSSE) 実装です。

Tomcatjss は、TLS を使用して TLS ソケットを作成するために必要なインターフェイスを実装します。tomcatjss が実装するソケットファクトリーは、以下に挙げるさまざまなプロパティーを使用して、ソケットをリッスンして tomcat に戻ります。tomcatjss 自体は、Java JSS システムを利用して、最終的にマシン上のネイティブ NSS 暗号化サービスと通信します。

Tomcat サーバーおよび Certificate System クラスが読み込まれると、tomcatjss が読み込まれます。ロードプロセスの説明を以下に示します。

1. サーバーが起動している。

2. Tomcat は、Certificate System インストールにリスニングソケットを作成する必要がある場所に移動します。

3. **server.xml** ファイルが処理されます。このファイルの設定は、Tomcatjs が実装するソケットファクトリーを使用するようシステムに指示します。

4. Tomcajss は要求される各ソケットについて、ソケットの作成時に含まれる属性を読み取り、処理します。結果として生成されるソケットは、それらのパラメーターによって要求されているために動作します。

5. サーバーが稼働したら、Tomcat ベースの Certificate System への着信接続を待機する必要なリッスンソケットのセットがあります。

ソケットが起動時に作成されると、Tomcat は、基礎となる JSS セキュリティーサービスと実際に処理される Certificate System の **最初** のエンティティーになります。最初のリスニングソケットが処理されると、後で使用できるように JSS のインスタンスが作成されます。

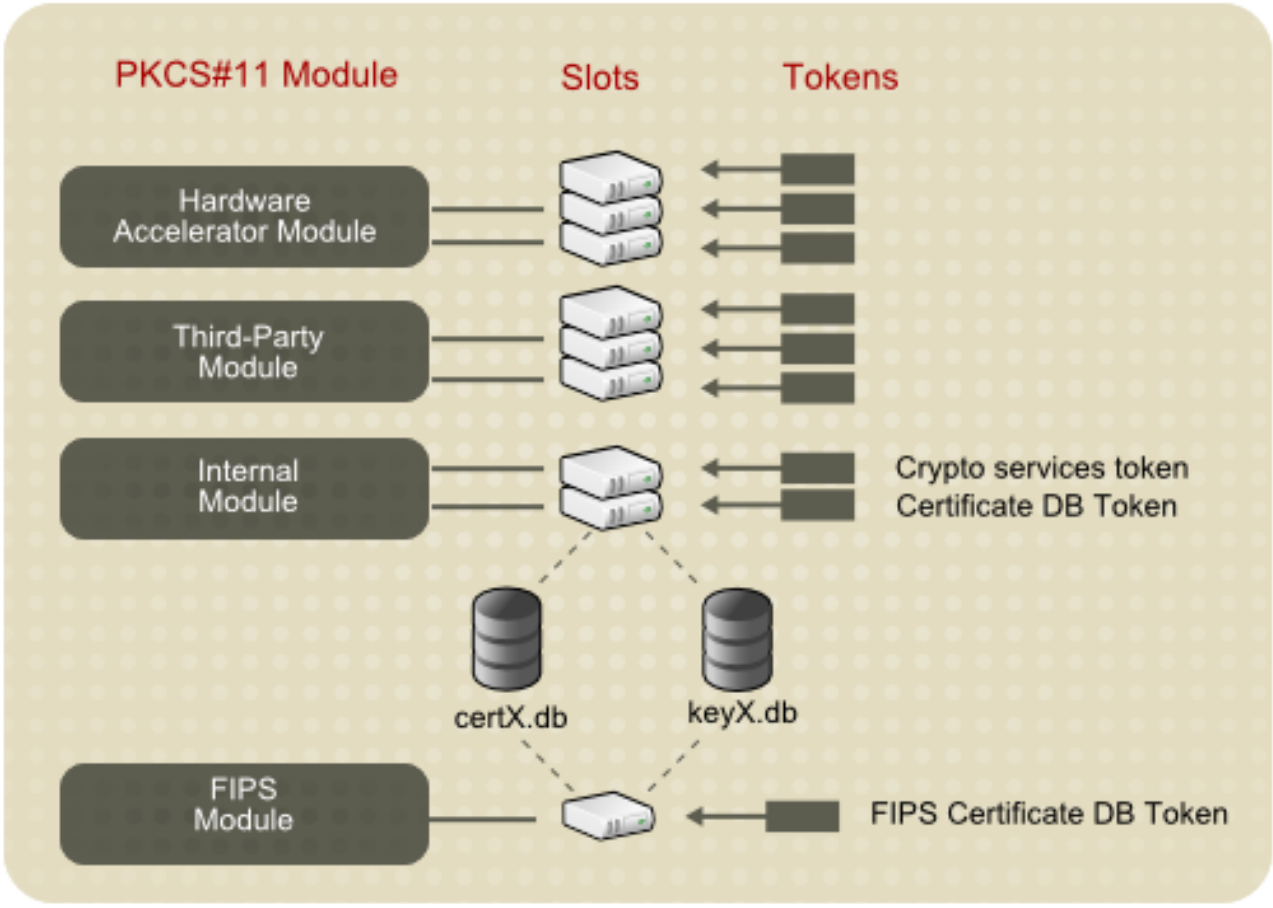
server.xml ファイルの詳細は、「Tomcat Engine および Web サービスの設定ファイル」 を参照してください。

2.3.7. PKCS #11

Public-Key Cryptography Standard (PKCS) #11 は、暗号化情報を保持するデバイスと通信し、暗号化操作を実行するのに使用する API を指定します。Certificate System は、PKCS #11 に対応しているため、Certificate System は、さまざまなハードウェアおよびソフトウェアデバイスと互換性があります。

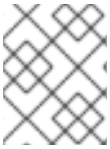
Certificate System サブシステムインスタンスで、少なくとも 1 つの PKCS #11 モジュールが利用可能である必要があります。PKCS #11 モジュール (暗号化モジュールまたは暗号化サービスプロバイダーとも呼ばれる) は、暗号化や復号などの暗号化サービスを管理します。PKCS #11 モジュールは、ハードウェアまたはソフトウェアのいずれかで実装できる暗号化デバイスのドライバーに類似しています。Certificate System には、ビルトイン PKCS #11 モジュールが含まれており、サードパーティーモジュールに対応します。

PKCS #11 モジュールには、常に 1 つ以上のスロットがあり、スマートカードなどの物理リーダーの物理ハードウェアスロットとして、またはソフトウェアの概念スロットとして実装できます。PKCS #11 モジュールの各スロットには、トークンを追加できます。このトークンは、暗号化サービスを実際に提供し、必要に応じて証明書と鍵を保存するハードウェアまたはソフトウェアのデバイスです。



Certificate System は、**内部** と **外部** の 2 種類のトークンを定義します。内部トークンは、証明書トラストアンカーを保存するために使用されます。外部トークンは、Certificate System サブシステムに属するキーペアと証明書を保存するために使用されます。

2.3.7.1. NSS Soft Token (内部トークン)



注記

Certificate System は、証明書のトラストアンカーを保存する NSS ソフトトークンを使用します。

NSS Soft トークンは、内部トークンまたはソフトウェアトークンとも呼ばれます。ソフトウェアトークンは 2 つのファイルで設定されており、通常は証明書データベース (cert9.db) とキーデータベース (key4.db) と呼ばれる 2 つのファイルで設定されます。このファイルは、Certificate System サブシステムの設定時に作成されます。これらのセキュリティーデータベースは、**/var/lib/pki/*instance_name*/alias** ディレクトリーにあります。

NSS ソフトトークンが提供する暗号化モジュールは、Certificate System に含まれます。

- デフォルトの内部 PKCS #11 モジュールには、2 つのトークンが含まれています。
  - 暗号化、復号化、ハッシュなどのすべての暗号化操作を実行する内部暗号化サービストークン。
  - 内部キーストレージトークン (証明書 DB トークン)。このトークンは、証明書および鍵を保存する証明書および鍵データベースファイルをすべて処理します。
- FIPS 140 モジュールこのモジュールは、暗号化モジュール実装用の FIPS 140 政府標準に準拠

します。FIPS 140 モジュールには、暗号化操作と証明書およびキーデータベースファイルとの通信の両方を処理する単一の組み込み FIPS 140 証明書データベーストークンが含まれています。

NSS ソフトトークンに証明書をインポートする方法の具体的な手順は、15章 *証明書/キー暗号トークンの管理*を参照してください。

Network Security Services (NSS) の詳細は、同じ名前の Mozilla Developer の Web ページを参照してください。

2.3.7.2. ハードウェアセキュリティーモジュール (HSM、外部トークン)

注記

Certificate System は、HSM を使用して、Certificate System サブシステムに属する鍵のペアと証明書を格納します。

PKCS #11 モジュールは、Certificate System で使用できます。サブシステムで外部ハードウェアトークンを使用するには、サブシステムの設定前に PKCS #11 モジュールを読み込み、新しいトークンをサブシステムで利用できるようになります。

利用可能な PKCS #11 モジュールは、サブシステムの **pkcs11.txt** データベースで追跡されています。**modutil** ユーティリティーは、署名操作に使用するハードウェアアクセラレーターのインストールなど、システムへの変更時にこのファイルを修正するために使用されます。**modutil** の詳細は、Mozilla Developer の Web ページで Network Security Services (NSS) を参照してください。

PKCS #11 ハードウェアデバイスは、ハードウェアトークンに保存されている情報に、主要なバックアップと復旧機能を提供します。トークンから鍵を取得する方法は、PKCS #11 ベンダーのドキュメントを参照してください。

証明書のインポートおよび HSM の管理方法の方法は、15章 *証明書/キー暗号トークンの管理*を参照してください。

サポート対象のハードウェアセキュリティーモジュールは、「サポート対象のハードウェアセキュリティーモジュール」に記載されています。

2.3.8. Certificate System のシリアル番号管理

2.3.8.1. シリアル番号の範囲

証明書要求とシリアル番号は Java の大きな整数 で表されます。

デフォルトでは、効率性のために、証明書要求番号、証明書シリアル番号、およびレプリカ ID が CA サブシステムに順番に割り当てられます。

シリアル番号の範囲は、要求、証明書、およびレプリカ ID によって異なります。

- 現在のシリアル番号管理は、連続したシリアル番号範囲の割り当てに基づいています。
- インスタンスは、定義されたしきい値を下回る場合に新しい範囲を要求します。
- インスタンスは、インスタンスに割り当てられたら、新たに取得した範囲に関する情報を保存します。

- インスタンスは、すべての数字が使い切られるまで古い範囲を引き続き使用し、新しい範囲に移動します。
- クローン作成されたサブシステムは、レプリケーションの競合を使用して範囲割り当てを同期します。

新しいクローンの場合

- マスターの現在の範囲には、クローン作成のプロセスの新しいクローンに転送されます。
- 転送範囲が定義されたしきい値を下回る場合に、新規クローンが新しい範囲を要求する可能性があります。

すべての範囲は、CA インスタンスのインストール中に設定可能です。これにより、PKI インスタンスの上書き設定ファイルに **[CA]** セクションを追加し、必要に応じて以下の **name=value** ペアをそのセクションに追記します。**/etc/pki/default.cfg** にすでに存在するデフォルト値を以下の例に示します。

```
[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. ランダムなシリアル番号管理

順次シリアル番号管理に加えて、Red Hat Certificate System は任意のランダムシリアル番号管理を提供します。乱数は、PKI インスタンスの上書きファイルに **[CA]** セクションを追加し、そのセクションに以下の **name=value** ペアを追加することによって、CA インスタンスのインストール時に選択可能です。

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

このチェックボックスを選択した場合、証明書要求番号と証明書のシリアル番号は、指定した範囲内で無作為に選択されます。

2.3.9. セキュリティードメイン

セキュリティードメイン は PKI サービスのレジストリーです。CA などのサービスは、これらのドメインに独自の情報を登録するため、PKI サービスのユーザーはレジストリーを確認して他のサービスを検索できます。RHCS のセキュリティードメインサービスは、Certificate System サブシステムの PKI サービスの登録と、共有信頼ポリシーのセットの両方を管理します。

詳細は、「セキュリティードメインの計画」を参照してください。

2.3.10. パスワードおよびウォッチドッグ (nuxwdog)

デフォルトの設定では、RHCS サブシステムインスタンスはクライアントとして機能し、LDAP 内部データベース (TLS クライアント認証が設定されていない限り、代わりに証明書が認証に使用されます、NSS トークンデータベース、またはパスワードを持つ HSM などの他のサービスに認証する必要があります。管理者は、インストール設定時にこのパスワードを設定するように求められます。このパスワードは、***<instance_dir>*/conf/password.conf** ファイルに書き込まれます。同時に、識別文字列は、パラメーター **cms.passwordlist** の一部としてメインの設定ファイル **CS.cfg** に保存されます。

設定ファイル **CS.cfg** は Red Hat Enterprise Linux により保護され、PKI 管理者のみがアクセスできます。パスワードは **CS.cfg** に保存されません。

インストール時に、インストーラーは内部ソフトウェアトークンまたはハードウェア暗号化トークンのいずれかを選択してログインします。これらのトークンへのログインパスフレーズは、**password.conf** にも記述されます。

後で設定は、パスワードを **password.conf** に配置することもできます。LDAP 公開は、公開ディレクトリーに対して新たに設定された Directory Manager パスワードが **password.conf** に送信される一例です。

Nuxwdog (watchdog) は、サーバープログラムを起動、停止、監視、および再設定するために使用される軽量な補助デーモンプロセスです。サーバーを起動するためにユーザーにパスワードの入力を求める必要がある場合に最も役立ちます。これは、これらのパスワードをカーネルキーリングに安全にキャッシュし、サーバーがクラッシュした場合に自動的に再起動できるようにするためです。

注記

Nuxwdog は、唯一許可されるウォッチドッグサービスです。

インストールが完了したら、**password.conf** ファイルを完全に削除できます。再起動時には、**nuxwdog** ウォッチドッグプログラムは、プロンプトが表示されたら、パラメーター **cms.passwordlist** (および HSM が使用される場合は ***cms.tokenList***) を使用して、必要なパスワードを要求します。その後、パスワードは、サーバークラッシュからの自動リカバリーを可能にするために、カーネルキーリングの **nuxwdog** によりキャッシュされます。制御されていないシャットダウン (クラッシュ) が原因で、この自動リカバリー (自動サブシステム再起動) が発生します。管理者が制御したシャットダウンの場合は、管理者がパスワードを再度要求します。

ウォッチドッグサービスを使用する場合は、RHCS インスタンスの起動と停止が異なります。詳細は、「Certificate System の Watchdog サービスの使用」 を参照してください。

詳細は、「システムパスワードの管理」 を参照してください。

2.3.11. 内部 LDAP データベース

Red Hat Certificate System は、証明書、要求、ユーザー、ロール、ACL、およびその他のさまざまな内部情報などの情報を格納するための内部データベースとして Red Hat Directory Server (RHDS) を採用しています。Certificate System は、パスワードを使用して内部 LDAP データベースと通信するか、SSL 認証により安全な方法で通信します。

Certificate System インスタンスと Directory Server 間で証明書ベースの認証が必要な場合は、これら 2 つのエンティティー間で信頼を設定する手順を実行することが重要です。このような Certificate System インスタンスのインストールにも、**pkispawn** の適切なオプションが必要です。

詳細は、「Red Hat Directory Server のインストール」 を参照してください。

2.3.12. SELinux (Security Enhanced Linux)

SELinux は、承認されていないアクセスと改ざんを制限するためにシステム全体で実施される、必須のアクセス制御ルールのコレクションです。SELinux の詳細は、『Red Hat Enterprise Linux 8 SELinux ユーザーおよび管理者のガイド』 を参照してください。

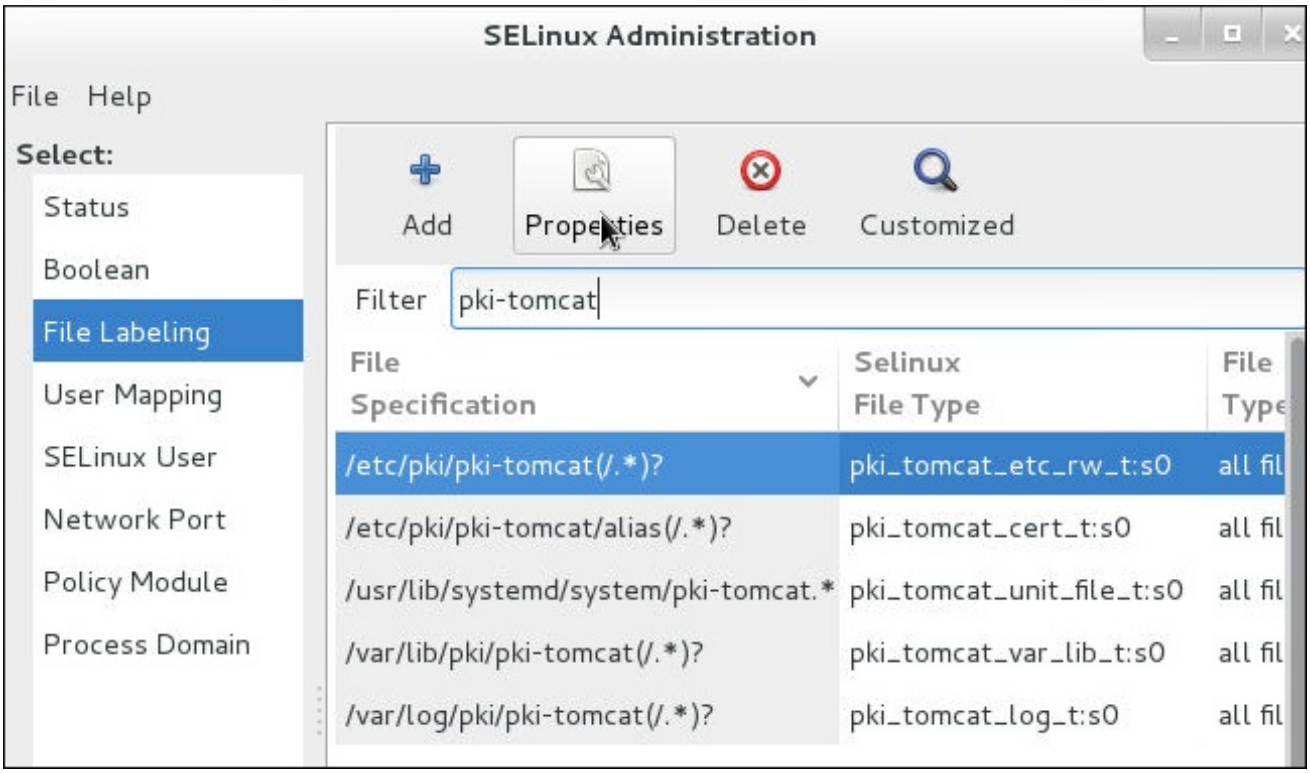
基本的に、SELinux はシステム上の **オブジェクト** を識別します。これは、ファイル、ディレクトリー、ユーザー、プロセス、ソケット、または Linux ホスト上その他のリソースになります。これらのオブジェクトは Linux API オブジェクトに対応します。各オブジェクトは **セキュリティーコンテキスト**

にマッピングされ、オブジェクトのタイプと、Linux サーバーでの機能が可能になります。

オブジェクトはドメインにグループ化でき、各ドメインには適切なルールが割り当てられます。各セキュリティーコンテキストには、実行できる操作、アクセスできるリソース、および所有するアクセス許可に制限を設定するルールがあります。

Certificate System の SELinux ポリシーは、標準のシステムの SELinux ポリシーに組み込まれています。これらの SELinux ポリシーは、Certificate System が使用するすべてのサブシステムとサービスに適用されます。SELinux で Certificate System を実行すると、Certificate System で作成および維持される情報のセキュリティーが強化されます。

図2.1 CA SELinux ポートポリシー



Certificate System SELinux ポリシーは、すべてのサブシステムインスタンスの SELinux 設定を定義します。

- 各サブシステムインスタンスのファイルおよびディレクトリーには、特定の SELinux コンテキストのラベルが付けられます。
- 各サブシステムインスタンスのポートは、特定の SELinux コンテキストでラベル付けされます。
- すべての Certificate System プロセスは、サブシステム固有のドメイン内で制限されます。
- 各ドメインには、ドメインに承認されたアクションを定義する特定のルールがあります。
- SELinux ポリシーに指定されていないアクセスは、Certificate System インスタンスに対して拒否されます。

Certificate System の場合、各サブシステムは SELinux オブジェクトとして扱われ、各サブシステムには一意のルールが割り当てられます。定義済みの SELinux ポリシーを使用すると、Certificate System オブジェクトが Enforcing モードで SELinux を設定して実行できます。

pkispawn が実行されるたびに、Certificate System サブシステム、そのサブシステムに関連付けられたファイル、およびポートには、必要な SELinux コンテキストのラベルが付けられます。このコンテキストは、特定のサブシステムが **pkidestroy** を使用して削除されると削除されます。

SELinux ポリシーの中央定義は **pki_tomcat_t** ドメインです。Certificate System インスタンスは Tomcat サーバーであり、**pki_tomcat_t** ドメインは標準の **tomcat_t** Tomcat ドメインのポリシーを拡張します。サーバー上のすべての Certificate System インスタンスが同じドメインを共有します。

各 Certificate System プロセスが開始すると、最初に制限のないドメイン (**unconfined_t**) で実行され、次に **pki_tomcat_t** ドメインに移行します。その後、このプロセスには、**pki_tomcat_log_t** というラベルが付けられたログファイルへの書き込みアクセス、**pki_tomcat_etc_rw_t** というラベルが付いた設定ファイルへの読み書きアクセス、**http_port_t** ポートのオープンおよび書き込み機能など、特定のアクセスパーミッションが設定されます。

SELinux モードは、Enforcing から Permissive に変更できますが、これは推奨されません。

2.3.13. セルフテスト

Red Hat Certificate System は、起動時またはオンデマンド、あるいはその両方で PKI システムの整合性をチェックできるセルフテストフレームワークを提供します。クリティカルでない自己テストに失敗すると、メッセージはログファイルに保存されますが、重大なセルフテストに失敗すると、メッセージはログファイルに保存されますが、Certificate System サブシステムは適切にシャットダウンします。管理者は、起動時にセルフテストレポートを確認する場合、サブシステムの起動時にセルフテストログを監視する必要があります。起動後にログを表示することもできます。

セルフテストの失敗によりサブシステムがシャットダウンすると、自動的に無効になります。これは、サブシステムが部分的に実行されなくなり、誤解を招く応答を生成するために行われます。問題が解決されると、サーバーで以下のコマンドを実行してサブシステムを再度有効にできます。

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

セルフテストの設定方法の詳細は、「セルフテストの設定」を参照してください。

2.3.14. ログ

Certificate System サブシステムは、管理、サーバーがサポートするプロトコルを使用した通信、およびサブシステムで使用されるその他のさまざまなプロセスなどのアクティビティーに関連するイベントを記録するログファイルを作成します。サブシステムインスタンスが実行中に、それが管理するすべてのコンポーネントの情報およびエラーメッセージのログを保持します。また、Apache および Tomcat の Web サーバーはエラーを生成し、ログにアクセスします。

各サブシステムインスタンスは、インストール、監査、およびその他のログに記録された機能に対する独自のログファイルを維持します。

ログプラグインモジュールは、Java™ クラスとして実装され、設定フレームワークに登録されるリスナーです。

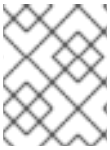
監査ログを除くすべてのログファイルとローテーションされたログファイルは、**pkispawn** によるインスタンスの作成時に、**pki_subsystem_log_path** に指定されているすべてのディレクトリーに配置されます。 通常の監査ログは他のログタイプとともにログディレクトリーに置かれ、署名された監査ログは **/var/log/pki/*instance_name*/*subsystem_name*/signedAudit** に書き込まれます。ログのデフォルトの場所を変更するには、設定を変更してください。

インストール中のログ設定の詳細および追加情報は、18章 *ログの設定*を参照してください。

インストール後のログ管理の詳細は、『Red Hat Certificate System 管理ガイド』の『サブシステムログの設定』セクションを参照してください。

2.3.14.1. 監査ログ

監査ログには、記録可能なイベントとして設定された選択可能なイベントのレコードが含まれます。監査ログを整合性チェック目的で署名するように設定することもできます。



注記

監査レコードは、「監査の保持」に指定された監査保持ルールに従って維持する必要があります。

2.3.14.2. デバッグログ

デフォルトで有効になっているデバッグログは、さまざまな程度と種類の情報とともに、すべてのサブシステムに対して維持されます。

各サブシステムのデバッグログには、実行されるプラグインやサーブレット、接続情報、サーバー要求および応答メッセージなど、サブシステムによって実行されるす全操作に関する具体的な情報が含まれる詳細情報が記録されます。

デバッグログに記録されるサービスには、承認要求、証明書要求の処理、証明書ステータスの確認、キーのアーカイブおよび復元、Web サービスへのアクセスが含まれます。

デバッグログは、サブシステムのプロセスの詳細情報を記録します。各ログエントリーの形式は以下のとおりです。

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

メッセージ はサブシステムから返されたメッセージになるか、サブシステムに送信された値を含めることができます。

たとえば、TKS は、LDAP サーバーに接続するためにこのメッセージを記録します。

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

processor は **main** で、**message** は LDAP 接続に関するサーバーからメッセージであり、サーブレットはありません。

一方、CA は、証明書操作およびサブシステム接続に関する情報を記録します。

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

この場合、**プロセッサー** は CA のエージェントポート上の HTTP プロトコルですが、プロファイルを処理するサーブレットを指定し、profile パラメーター (リクエストのサブシステム所有者) とその値 (KRA が要求を開始した) を示す **メッセージ** が含まれます。

> **例2.1 CA 証明書要求ログメッセージ**
>
> > [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=
```

同様に、OCSP には OCSP 要求情報が表示されます。

```
[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. インストールログ

すべてのサブシステムはインストールログを保持します。

サブシステムが初期インストールで作成される場合や **pkispawn** によって追加のインスタンスを作成するたびに、インストールからの完全なデバッグ出力を含むインストールファイル (エラーを含む) と、インストールに成功すると、インスタンスの設定インターフェイスへの URL および PIN が作成されます。このファイルは、**/var/log/pki/** ディレクトリーに、名前が **pki-*subsystem_name*-spawn.*timestamp*.log** の形式で指定します。

インストールログの各行は、インストールプロセスのステップに従います。

例2.2 CA インストールログ

```
==========================================================================
                      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 のエラーとアクセスログ

CA、KRA、OCSP、TKS、および TPS サブシステムは、それらのエージェントおよびエンドエンティティーのインターフェイスに Tomcat Web サーバーインスタンスを使用します。

エラーログとアクセスログは、Certificate System とともにインストールされ、HTTP サービスを提供する Tomcat Web サーバーによって作成されます。エラーログには、サーバーが検出した HTTP エラーメッセージが含まれます。アクセスログは、HTTP インターフェイスを介したアクセスアクティビティーを一覧表示します。

Tomcat によって作成されたログ:

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

これらのログは、Certificate System 内では利用できません。それらは Apache または Tomcat 内でのみ設定可能です。これらのログの設定に関する詳細は、Apache ドキュメントを参照してください。

2.3.14.5. セルフテストログ

セルフテストのログは、サーバーの起動時またはセルフテストを手動で実行するときに取得した情報を記録します。このログを開くとテストを表示できます。このログはコンソールを介して設定できません。このログは、**CS.cfg** ファイルの設定を変更してのみ設定できます。このセクションのログに関する情報は、このログには関係しません。セルフテストについての詳細は、「セルフテスト」を参照してください。

2.3.14.6. journalctl ログ

Certificate System インスタンスを起動すると、ログサブシステムがセットアップされて有効になるまでに少し時間がかかります。この間に、ログの内容は標準出力に書き込まれます。これは **systemd** でキャプチャーされ、**journalctl** ユーティリティーを介して公開されます。

これらのログを表示するには、以下のコマンドを実行します。

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

nuxwdog サービスを使用している場合は、以下を実行します。

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

多くの場合、インスタンスの起動時にこれらのログを監視すると便利です (たとえば、起動時にセルフテストが失敗した場合)。そのためには、インスタンスを起動する前に、別のコンソールでこれらのコマンドを実行します。

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

nuxwdog サービスを使用している場合は、以下を実行します。

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

2.3.15. インスタンスのレイアウト

各 Certificate System インスタンスは、多数のファイルによって異なります。その一部はインスタンス固有のフォルダーにありますが、他のサーバーインスタンスと共有される共通フォルダーに置かれています。

たとえば、サーバー設定ファイルはインスタンス固有の **/etc/pki/*instance_name*/server.xml** に保存されますが、CA サーブレットは **/usr/share/pki/ca/webapps/ca/WEB-INF/web.xml** で定義されています。これはシステム上のすべてのサーバーインスタンスで共有されます。

2.3.15.1. Certificate System のファイルおよびディレクトリーの場所

Certificate System サーバーは、1 つ以上の Certificate System サブシステムで設定される Tomcat インスタンスです。Certificate System サブシステムは、特定の PKI 機能を提供する Web アプリケーションです。一般的な共有サブシステム情報は、再配置不可能な RPM 定義の共有ライブラリー、Java アーカイブファイル、バイナリー、およびテンプレートに含まれています。これらは固定された場所に保存されます。

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

ディレクトリーには、カスタマイズされた設定ファイルとテンプレート、プロファイル、証明書データベース、およびサブシステムの他のファイルが含まれます。

表2.2 Tomcat インスタンス情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat
設定ディレクトリー	/etc/pki/pki-tomcat
設定ファイル	/etc/pki/pki-tomcat/server.xml /etc/pki/pki-tomcat/password.conf
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias
サブシステム証明書	SSL サーバー証明書 サブシステム証明書 [a]

設定	値
ログファイル	/var/log/pki/pki-tomcat
Web サービスファイル	/usr/share/pki/server/webapps/ROOT - メインページ /usr/share/pki/server/webapps/pki/admin - 管理テンプレート /usr/share/pki/server/webapps/pki/js - JavaScript ライブラリー
[a] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。	

注記

/var/lib/pki/*instance_name*/conf/ ディレクトリーは、**/etc/pki/*instance_name*/** ディレクトリーへのシンボリックリンクです。

2.3.15.2. CA サブシステム情報

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

表2.3 CA サブシステム情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/ca
設定ディレクトリー	/etc/pki/pki-tomcat/ca
設定ファイル	/etc/pki/pki-tomcat/ca/CS.cfg
サブシステム証明書	CA 署名証明書 OCSP 署名証明書 (CA の内部 OCSP サービス用) 監査ログ署名証明書
ログファイル	/var/log/pki/pki-tomcat/ca
ログのインストール	/var/log/pki/pki-ca-spawn.YYYYMMDDhhmmss.log
プロファイルファイル	/var/lib/pki/pki-tomcat/ca/profiles/ca
メール通知テンプレート	/var/lib/pki/pki-tomcat/ca/emails

設定	値
Web サービスファイル	/usr/share/pki/ca/webapps/ca/agent: エージェントサービス /usr/share/pki/ca/webapps/ca/admin: 管理サービス /usr/share/pki/ca/webapps/ca/ee: エンドユーザーサービス

2.3.15.3. KRA サブシステム情報

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

表2.4 KRA サブシステム情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/kra
設定ディレクトリー	/etc/pki/pki-tomcat/kra
設定ファイル	/etc/pki/pki-tomcat/kra/CS.cfg
サブシステム証明書	トランスポート証明書 ストレージ証明書 監査ログ署名証明書
ログファイル	/var/log/pki/pki-tomcat/kra
ログのインストール	/var/log/pki/pki-kra-spawn.YYYYMMDDhhmmss.log
Web サービスファイル	/usr/share/pki/kra/webapps/kra/agent: エージェントサービス /usr/share/pki/kra/webapps/kra/admin: 管理サービス

2.3.15.4. OCSP サブシステム情報

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

表2.5 OCSP サブシステム情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/ocsp
設定ディレクトリー	/etc/pki/pki-tomcat/ocsp

設定	値
設定ファイル	/etc/pki/pki-tomcat/ocsp/CS.cfg
サブシステム証明書	OCSP 署名証明書 監査ログ署名証明書
ログファイル	/var/log/pki/pki-tomcat/ocsp
ログのインストール	/var/log/pki/pki-ocsp-spawn.YYYYMMDDhhmmss.log
Web サービスファイル	/usr/share/pki/ocsp/webapps/ocsp/agent: エージェントサービス /usr/share/pki/ocsp/webapps/ocsp/admin: 管理サービス

2.3.15.5. TKS サブシステム情報

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

表2.6 TKS サブシステム情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/tks
設定ディレクトリー	/etc/pki/pki-tomcat/tks
設定ファイル	/etc/pki/pki-tomcat/tks/CS.cfg
サブシステム証明書	監査ログ署名証明書
ログファイル	/var/log/pki/pki-tomcat/tks
ログのインストール	/var/log/pki/pki-tomcat/pki-tks-spawn.YYYYMMDDhhmmss.log

2.3.15.6. TPS サブシステム情報

ディレクトリーはインスタンスに固有のものでり、インスタンス名に関連付けられています。この例では、インスタンス名は **pki-tomcat** です。true の値は、 **pkispawn** を使用したサブシステムの作成時に指定されます。

表2.7 TPS サブシステム情報

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/tps

設定	値
設定ディレクトリー	/etc/pki/pki-tomcat/tps
設定ファイル	/etc/pki/pki-tomcat/tps/CS.cfg
サブシステム証明書	監査ログ署名証明書
ログファイル	/var/log/pki/pki-tomcat/tps
ログのインストール	/var/log/pki/pki-tps-spawn.YYYYMMDDhhhmmss.log
Web サービスファイル	/usr/share/pki/tps/webapps/tps - TPS サービス

2.3.15.7. 共有 Certificate System サブシステムファイルの場所

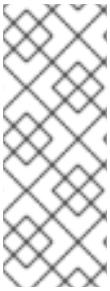
サーバー全般操作のために、すべての Certificate System サブシステムインスタンスで使用されるディレクトリーがあります (表2.8「サブシステムファイルの場所」 に記載されています)。

表2.8 サブシステムファイルの場所

ディレクトリーの場所	コンテンツ
/usr/share/pki	Certificate System インスタンスの作成に使用する一般的なファイルとテンプレートが含まれます。すべてのサブシステムの共有ファイルとともに、サブディレクトリーにサブシステム固有のファイルがあります。 • pki/ca (CA) • pki/kra (KRA) • pki/ocsp (OCSP) • pki/tks (TKS) • pki/tps (TPS)
/usr/bin	Certificate System サブシステムが共有する **pkispawn** および **pkidestroy** インスタンス設定スクリプトおよびツール (Java、ネイティブ、およびセキュリティー) が含まれます。
/usr/share/java/pki	ローカルの Tomcat Web アプリケーションが共有し、Certificate System サブシステムで共有される Java アーカイブファイルが含まれます。

2.4. PKI (CERTIFICATE SYSTEM あり)

Certificate System はサブシステムで設定されており、各サブシステムが公開鍵インフラストラクチャーのさまざまな機能を提供します。PKI 環境は、サブシステムにさまざまな機能を実装することで、個々のニーズに合わせてカスタマイズできます。



注記

従来の PKI 環境は、ソフトウェアデータベースに保存されている証明書を管理する基本的なフレームワークを提供します。これは、スマートカードで証明書を管理しないため、**TMS 以外** の環境です。**TMS** 環境では、スマートカードで証明書を管理します。

少なくとも、TMS 以外の環境では CA のみが必要ですが、TMS 以外の環境では OCSP レスポンダーと KRA インスタンスも使用できます。

2.4.1. 証明書の発行

通常、Certificate Manager は Certificate System の中核となります。リクエストから登録 (発行)、更新、失効まで、あらゆる段階で証明書を管理します。

Certificate System は、証明書の登録と発行、および Web ブラウザー、サーバー、仮想プライベートネットワーク (VPN) クライアントなどのさまざまなエンドエンティティーからの証明書要求の処理をサポートします。発行した証明書は X.509 バージョン 3 標準仕様に準拠します。

詳細は、『Red Hat Certificate System 管理ガイド』の『証明書の登録および更新について』セクションを参照してください。

2.4.1.1. 登録プロセス

エンドエンティティーは、エンドエンティティーインターフェイスを介して登録要求を送信することにより、PKI 環境に登録します。さまざまな登録方法を使用したり、さまざまな認証方法を必要とする多くの種類の登録があります。異なるインターフェイスが、異なるタイプの証明書署名要求 (CSR) を受け入れることもできます。

Certificate Manager は、グラフィカルインターフェイスやコマンドラインツールの使用など、CSR を送信するさまざまな方法をサポートします。

2.4.1.1.1. ユーザーインターフェイスを使用した登録

ユーザーインターフェイスを介した登録ごとに、登録の種類、認証の種類、および証明書の種類に関連付けられた証明書プロファイルに固有の個別の登録ページが作成されます。登録に関連するフォームは、外観とコンテンツの両方でカスタマイズできます。または、登録タイプごとに証明書プロファイルを作成することにより、登録プロセスをカスタマイズできます。証明書プロファイルは、証明書プロファイルに関連付けられた入力を設定することによってカスタマイズされたフォームを動的に生成します。異なるインターフェイスが、異なるタイプの証明書署名要求 (CSR) を受け入れることもできます。

エンドエンティティーが証明書を要求して PKI に登録すると、PKI の設定とインストールされているサブシステムに応じて、次のイベントが発生する可能性があります。

1. エンドエンティティーは、登録フォームの 1 つに情報を提供し、リクエストを送信します。

   エンドエンティティーから収集された情報は、収集された情報に応じてフォームでカスタマイズ可能であり、証明書に保存したり、フォームに関連付けられた認証方法に対して認証したりできます。このフォームは、Certificate Manager に送信される要求を作成します。

2. 登録フォームは、リクエストの公開鍵と秘密鍵、またはデュアルキーペアの作成をトリガーします。

3. エンドエンティティーは、認証タイプに応じて、リクエストの送信前に認証情報を提供します。LDAP 認証、PIN ベースの認証、または証明書ベースの認証を指定できます。

4. リクエストは、エージェントの承認登録プロセスまたは自動プロセスのいずれかに送信されます。

    - エンドエンティティー認証を含まないエージェント承認プロセスは、エージェントが要求を処理する必要があるエージェントサービスインターフェイスの要求キューに要求を送信します。その後、エージェントはリクエストの一部を変更したり、リクエストのステータスを変更したり、リクエストを拒否したり、リクエストを承認することができます。

      自動通知を設定して、リクエストがキューに表示されるたびにエージェントにメールが送信されるようにすることができます。また、自動化されたジョブを設定して、事前に設定されたスケジュールでキューの内容のリストをエージェントに送信することもできます。

    - エンドエンティティー認証を含む自動化されたプロセスは、エンドエンティティーが正常に認証されるとすぐに証明書要求を処理します。

5. フォームの送信時に LDAP ディレクトリーからエンドエンティティーに関する情報を収集します。証明書プロファイルベースの登録では、フォームのデフォルト値を使用して、ユーザーの LDAP ID およびパスワードを収集します。

6. フォームに関連付けられた証明書プロファイルは、発行する証明書の要素を決定します。証明書プロファイルに応じて、リクエストは、要求が制約セットを満たすか、必要な情報が提供されているか、および新しい証明書の内容を決定するために評価されます。

7. フォームは、ユーザーが秘密鍵をエクスポートするようにリクエストすることもできます。KRA サブシステムがこの CA で設定されている場合は、エンドエンティティーのキーが要求され、アーカイブされた要求が KRA に送信されます。このプロセスは通常、エンドエンティティーからの対話を必要としません。

8. 証明書要求は、証明書プロファイルまたは認証要件を満たしていないために拒否されるか、証明書が発行されます。

9. 証明書はエンドエンティティーに配信されます。

    - 自動登録では、証明書は即座にユーザーに配信されます。通常、登録は HTML ページを介して行われるため、証明書は別の HTML ページの応答として返されます。

    - エージェントが承認した登録では、証明書はシリアル番号またはエンドエンティティーインターフェイスのリクエスト ID で取得できます。

    - 通知機能が設定されている場合は、証明書の取得先のリンクがエンドユーザーに送信されます。

10. 証明書が発行または拒否されると、自動通知をエンドエンティティーに送信できます。

11. 新しい証明書は Certificate Manager の内部データベースに保存されます。

12. Certificate Manager に公開が設定されている場合、証明書はファイルまたは LDAP ディレクトリーに公開されます。

13. 内部 OCSP サービスは、証明書ステータス要求を受け取る際に内部データベースの証明書のステータスを確認します。

エンドエンティティーインターフェイスには、発行された証明書および CA 証明書チェーンを検索するフォームがあります。

デフォルトでは、ユーザーインターフェイスは PKCS #10 および Certificate Request Message Format (CRMF) の CSR に対応しています。

2.4.1.1.2. コマンドラインを使用した登録

本セクションでは、コマンドラインを使用して証明書を登録する際の一般的なワークフローを説明します。

2.4.1.1.2.1. pki ユーティリティーを使用した登録

詳細は、以下を参照してください。

- pki-cert(1) の man ページ
- 『Red Hat Certificate System 管理ガイド』の『コマンドラインインターフェイス』セクションを参照してください。

2.4.1.1.2.2. CMC を使用した登録

CMC に証明書を登録するには、次の手順に従います。

1. **PKCS10Client**、**CRMFPopClient** などのユーティリティーを使用して、PKCS #10 または CRMF 証明書署名要求 (CSR) を生成します。

   注記

   Key Recovery Agent (KRA) で鍵のアーカイブが有効になっている場合は、**kra.transport** ファイルに設定されている Privacy Enhanced Mail (PEM) 形式の KRA のトランスポート証明書とともに、**CRMFPopClient** ユーティリティーを使用します。

2. **CMCRequest** ユーティリティーを使用して、CSR を CMC 要求に変換します。**CMCRequest** ユーティリティーは、設定ファイルを入力として使用します。このファイルには、CSR へのパスや CSR の形式などが含まれます。

   詳細および例は、CMCRequest(1) の man ページを参照してください。

3. **HttpClient** ユーティリティーを使用して、CMC 要求を CA に送信します。**HttpClient** は、CMC 要求ファイルやサーブレットへのパスなどの設定を含む設定ファイルを使用します。

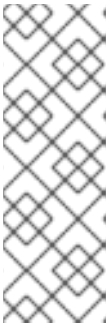
   HttpClient コマンドが成功すると、ユーティリティーは CA から CMC ステータス制御を備えた PKCS #7 チェーンを受け取ります。

   ユーティリティーが提供するパラメーターの詳細は、パラメーターを指定せずに **HttpClient** コマンドを入力します。

4. **CMCResponse** ユーティリティーを使用して、**HttpClient** で生成された PKCS #7 ファイルの発行結果を確認します。リクエストが正常に行われると、**CMCResponse** は証明書チェーンを読み取り可能な形式で表示します。

   詳細は、CMCResponse(1) の man ページを参照してください。

5. 新しい証明書をアプリケーションにインポートします。詳細は、証明書をインポートするアプリケーションの手順に従います。



注記

HttpClient が取得した証明書は、PKCS #7 形式になります。アプリケーションが Base64 でエンコードされた証明書のみに対応している場合は、**BtoA** ユーティリティーを使用して証明書を変換します。

また、一部のアプリケーションでは、PEM (Privacy Enhanced Mail) 形式の証明書にヘッダーとフッターが必要です。必要な場合は、証明書を変換した後に PEM ファイルに手動で追加します。

2.4.1.1.2.2.1. POP なしの CMC 登録

POP (Proof Of Possession) がない場合、**HttpClient** ユーティリティーは、**EncryptedPOP** CMC ステータスを受け取ります。これは、**CMCResponse** コマンドにより表示されます。この場合は、設定ファイルに異なるパラメーターを指定して **CMCRequest** コマンドを再び入力します。

詳細は、『Red Hat Certificate System 管理ガイド』 の 『CMC 登録プロセス』 セクションを参照してください。

2.4.1.1.2.2.2. 署名付き CMC 要求

CMC 要求は、ユーザーまたは CA エージェントのいずれかによって署名できます。

- エージェントが要求に署名する場合は、プロファイルの認証方法を **CMCAuth** に設定します。
- ユーザーがリクエストに署名する場合は、プロファイルの認証方法を **CMCUserSignedAuth** に設定します。

詳細は、『Red Hat Certificate System 管理ガイド』 の 『CMC 認証プラグイン』 セクションを参照してください。

2.4.1.1.2.2.3. 署名なしの CMC 要求

CMCUserSignedAuth 認証プラグインはプロファイルで設定されているため、共有秘密認証メカニズムと組み合わせて署名されていない CMC 要求を使用する必要があります。

注記

署名なしで要求は **自己署名 CMC 要求** とも呼ばれます。

詳細は、『Red Hat Certificate System 管理ガイド』 の 『Firewall Authentication Plug-ins』 セクションおよび 「CMC 共有シークレット機能の有効化」 を参照してください。

2.4.1.1.2.2.4. 共有シークレットのワークフロー

Certificate System は、RFC 5272 に従って、CMC リクエストの共有シークレット認証メカニズムを提供します。パスフレーズを保護するには、**CMCSharedToken** コマンドの使用時に、発行保護証明書を提供する必要があります。保証保護証明書は、KRA トランスポート証明書と同じように機能します。詳細は、CMCSharedToken(1) の man ページおよび 「CMC 共有シークレット機能の有効化」 を参照してください。

エンドエンティティーユーザーによって作成された共有シークレット (推奨)
以下に、ユーザーが共有の秘密を生成する場合にワークフローを説明します。

1. エンドエンティティーユーザーは、CA 管理者から発行保護証明書を取得します。

2. エンドエンティティーユーザーは **CMCSharedToken** ユーティリティーを使用して共有シークレットトークンを生成します。

注記

-p オプションは、トークンのパスワードではなく、CA とユーザー間で共有されるパスフレーズを設定します。

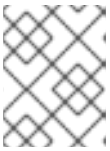
3. エンドエンティティーユーザーは、**CMCSharedToken** ユーティリティーによって生成された暗号化された共有トークンを管理者に送信します。

4. 管理者は、ユーザーの LDAP エントリーの **shrTok** 属性に共有トークンを追加します。

5. エンドエンティティーユーザーはパスフレーズを使用して、**CMCRequest** ユーティリティーに渡される設定ファイルの ***witness.sharedSecret*** パラメーターを設定します。

CA 管理者によって作成された共有シークレット
以下では、CA 管理者がユーザーの共有のシークレットを生成する場合にワークフローを説明します。

1. 管理者は **CMCSharedToken** ユーティリティーを使用して、ユーザーの共有シークレットトークンを生成します。



注記

-p オプションは、トークンのパスワードではなく、CA とユーザー間で共有されるパスフレーズを設定します。

2. 管理者は、ユーザーの LDAP エントリーの **shrTok** 属性に共有トークンを追加します。

3. 管理者は、パスフレーズをユーザーと共有します。

4. エンドエンティティーユーザーはパスフレーズを使用して、**CMCRequest** ユーティリティーに渡される設定ファイルの ***witness.sharedSecret*** パラメーターを設定します。

2.4.1.1.2.2.5. 単純な CMC 要求

Certificate System は、シンプルな CMC 要求を許可します。ただし、このプロセスは完全な CMC 要求と同じレベルのセキュリティー要件をサポートしていないため、安全な環境でのみ使用する必要があります。

簡単な CMC 要求を使用する場合は、**HttpClient** ユーティリティーの設定ファイルで以下を設定します。

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

2.4.1.2. 証明書プロファイル

Certificate System は証明書プロファイルを使用して、証明書の内容、証明書を発行する制約、使用する登録方法、およびその登録方法の入力フォームおよび出力フォームを設定します。1 つの証明書プロファイルが、特定の証明書の発行に関連付けられます。

最も一般的な証明書プロファイルのセットは、最も一般的な証明書タイプに含まれます。プロファイル設定は変更できます。証明書プロファイルは管理者が設定し、エージェントの承認のためにエージェントサービスページに送信されます。証明書プロファイルが承認されると、使用が有効になっています。

UI の登録の場合、証明書プロファイルに対して動的に生成される HTML フォームは、証明書プロファイルで呼び出す証明書登録のエンドエンティティーページで使用されます。コマンドラインベースの登録の場合は、認証、認可、入力、出力、デフォルト、制約などの同じ処理を実行するために証明書プロファイルが呼び出されます。サーバーは、要求に対応する前に、証明書プロファイルに設定されているデフォルトと制約が満たされていることを確認し、証明書プロファイルを使用して発行された証明書の内容を判別します。

Certificate Manager は、プロファイルや送信の証明書要求の設定に応じて、以下のいずれかの特徴で証明書を発行できます。

- X.509 バージョン 3 に準拠する証明書
- 証明書サブジェクト名と発行者名に対する Unicode サポート
- 空の証明書のサブジェクト名のサポート
- カスタマイズされたサブジェクト名コンポーネントのサポート
- カスタマイズされた拡張機能のサポート

デフォルトでは、証明書登録プロファイルは ***<instance directory>/ca/profiles/ca*** に保存され、その名前は ***<profile id>*.cfg** の形式になります。適切な **pkispawn** 設定パラメーターを使用すると、LDAP ベースのプロファイルが可能です。

2.4.1.3. 証明書登録の認証

Certificate System は、証明書登録の認証オプションを提供します。これには、エージェントが要求を処理するエージェント承認済み登録と、エンドエンティティーを認証する認証方法をが使用され、CA が自動的に証明書を発行する自動登録があります。エージェントによって承認されたリクエストを自動的に処理する CMC 登録もサポートされています。

2.4.1.4. クロスペアの証明書

これら 2 つの CA 間でクロス署名証明書を発行および格納することにより、2 つの異なる CA 間で信頼される関係を作成できます。クロス署名証明書ペアを使用すると、組織の PKI 以外で発行された証明書は、システム内で信頼できます。

2.4.2. 証明書の更新

証明書が有効期限に達すると、失効を許可するか、更新することができます。

更新 は、その証明書の既存の鍵ペアを使用して証明書要求を再生成し、要求を Certificate Manager に再送信します。更新された証明書は、1 つの例外を除いて、元の証明書と同じです (同じプロファイルから同じキーマテリアルを使用して作成されたため)。有効期限は異なり、遅くなります。

更新された証明書は古い証明書とまったく同じように機能するため、更新により、証明書の管理とユーザーとサーバー間の関係がはるかにスムーズになります。ユーザー証明書の更新により、暗号化されたデータには失われなくてもアクセスすることができます。

2.4.3. 証明書および CRL の公開

証明書はファイルおよび LDAP ディレクトリー、CRL をファイル、LDAP ディレクトリー、および OCSP レスポンダーに公開できます。公開フレームワークは、3 つのすべての場所に公開するための堅牢なツールセットを提供し、証明書や CRL を公開する場所をより詳細に定義するルールを設定します。

2.4.4. 証明書の取り消しとステータスの確認

エンドエンティティーは、独自の証明書が取り消されることを要求できます。エンドエンティティーが要求を行うとき、証明書を CA に提示する必要があります。証明書とキーが使用可能な場合、要求は処理されて Certificate Manager に送信され、証明書は取り消されます。Certificate Manager は、証明書をデータベースで取り消されたとマークし、該当する CRL に追加します。

エージェントは、エージェントサービスインターフェイスで証明書を検索し、取り消しにすることにより、Certificate Manager が発行する証明書をすべて取り消すことができます。証明書が取り消されると、証明書が公開用に設定されている場合は、データベースと公開ディレクトリーで取り消されたとマークされます。

内部の OCSP サービスが設定されている場合、サービスは内部データベースで証明書のステータスを判断します。

自動通知は、証明書失効通知メッセージを有効にして設定すると、証明書が取り消された時にエンドエンティティーに電子メールメッセージを送信するように、自動通知を設定することができます。

2.4.4.1. 証明書の取り消し

ユーザーは、以下を使用して証明書を取り消すことができます。

- エンドエンティティーページ。詳細は、『Red Hat Certificate System 管理ガイド』 の 『証明書取り消しページ』 セクションを参照してください。
- コマンドラインでの **CMCRequest** ユーティリティー。詳細は、『Red Hat Certificate System 管理ガイド』 の 『CMC 取り消しの実行』 セクションを参照してください。
- コマンドラインの **pki** ユーティリティー詳細は、pki-cert(1) の man ページを参照してください。

2.4.4.2. 証明書の状態

2.4.4.2.1. CRL

Certificate System は、設定可能なフレームワークから証明書失効リスト (CRL) を作成できます。これにより、ユーザー定義の発行ポイントが可能になり、発行する各ポイントに CRL を作成できます。デルタ CRL は、定義した発行ポイントにも作成できます。CRL は、証明書の各タイプ、証明書のタイプの特定のサブセット、またはプロファイルまたはプロファイルのリストに従って生成された証明書に対して発行できます。CRL を公開する際の頻度と間隔がすべて設定できます。

Certificate Manager は X.509 標準 CRL を発行します。CRL は、証明書が取り消されたり、指定した間隔で毎回更新される可能性があります。

2.4.4.2.2. OCSP サービス

Certificate System CA は、PKIX 標準 RFC 2560 で定義されている Online Certificate Status Protocol (OCSP) をサポートします。OCSP プロトコルを使用すると、OCSP 準拠のアプリケーションは、CA から検証機関に公開された CRL を直接確認しなくても、失効ステータスを含む証明書の状態を判別できます。*OCSP レスポンダー* とも呼ばれる検証権限は、アプリケーションをチェックします。

1. CA は、証明書のステータスに対してクエリーできる OCSP レスポンダーを特定する Authority Information Access 拡張を含む証明書を発行するよう設定されます。
2. CA は CRL を OCSP レスポンダーに定期的に公開します。

3. OCSP レスポンダーは、CA から受け取る CRL を維持します。

4. OCSP 準拠のクライアントは、検証用の OCSP レスポンダーに証明書を特定するのに必要なすべての情報が含まれるリクエストを送信します。アプリケーションは、検証する証明書の Authority Information Access 拡張の値から OCSP レスポンダーの場所を決定します。

5. OCSP レスポンダーは、リクエストに処理に必要なすべての情報が含まれるかどうかを判断します。有効になっていない場合、または要求されたサービスに対して有効になっていない場合は、拒否通知が送信されます。十分な情報がある場合は、要求を処理し、証明書のステータスを示すレポートを送信します。

2.4.4.2.2.1. OCSP レスポンスの署名

拒否通知を含む、クライアントが受信するすべての応答は、応答者によってデジタル署名されます。クライアントは、署名を検証して、要求を送信したレスポンダーからの応答であることを確認する必要があります。レスポンダーがメッセージに署名するために使用するキーは、OCSP レスポンダーが PKI セットアップでどのように展開されているかによって異なります。RFC 2560 は、応答に署名するために使用される鍵が以下のいずれかに属することを推奨します。

- ステータスがチェックされている証明書を発行した CA。
- クライアントが信頼する公開鍵のあるレスポンダー。このようなレスポンダーは *信頼できるレスポンダー* と呼ばれます。
- 証明書を取り消して CRL を公開する CA から直接発行された、特別にマークされた証明書を保持するレスポンダー。レスポンダーによるこの証明書の所持は、CA が、CA によって取り消された証明書に対して OCSP 応答を発行することをレスポンダーに許可したことを示します。このようなレスポンダーは、*CA 設計されたレスポンダー* または *CA 承認レスポンダー* と呼ばれます。

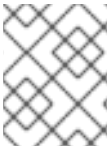
Certificate Manager のエンドエンティティーには、OCSP レスポンダーの証明書を手動で要求するためのフォームが含まれています。デフォルトの登録フォームには、OCSP レスポンダー証明書として証明書を識別するすべての属性が含まれます。OCSPNoCheck や Extended Key Usage などの必要な証明書拡張機能は、証明書が送信されたときに証明書に追加できます。

2.4.4.2.2.2. OCSP レスポンス

クライアントが受け取った OCSP 応答は、OCSP レスポンダーが決定した証明書の現在の状態を示します。応答は以下のいずれかになります。

- **Good or Verified**。ステータス照会に対する肯定応答を指定します。これは、証明書が取り消されていないことを意味します。必ずしも証明書が発行されたことや、証明書の有効期間内であることを意味するわけではありません。応答拡張は、証明書のステータスに関するレスポンダーによるアサーションの追加情報を示すために使用できます。
- **Revoked**。永続的または一時的に、証明書が取り消されたことを指定します。

クライアントは、ステータスに基づいて、証明書を検証するかどうかを決定します。



注記

OCSP レスポンダーは **Unknown** の応答を返しません。応答は常に **Good** または **Revoked** になります。

2.4.4.2.2.3. OCSP サービス

OCSP サービスの設定方法は 2 つあります。

- Certificate Manager に構築された OCSP
- Online Certificate Status Manager サブシステム

Certificate Manager は、組み込みの OCSP サービスの他に、CRL を OCSP 準拠の検証認証局に公開できます。CA は、CRL を Certificate System Online Certificate Status Manager に公開できるように設定できます。Online Certificate Status Manager は、各 Certificate Manager の CRL を内部データベースに保存し、適切な CRL を使用して、OCSP 準拠のクライアントから照会されたときに証明書の失効ステータスを確認します。

Certificate Manager は、証明書が取り消され、指定した間隔で CRL を生成および公開します。OCSP レスポンダーの目的は、証明書の即時検証を容易にすることを目的としているため、Certificate Manager は証明書を取り消すたびに CRL を Online Certificate Status Manager に公開する必要があります。間隔を置いてのみ公開するということは、OCSP サービスが古い CRL をチェックしていることを意味します。

注記

CRL が大きい場合、Certificate Manager は CRL を公開するのにかなり時間がかかる場合があります。

Online Certificate Status Manager は、各 Certificate Manager の CRL を内部データベースに保存し、証明書の検証に CRL として使用します。Online Certificate Status Manager は LDAP ディレクトリーに公開される CRL も使用できます。そのため、Certificate Manager は CRL を Online Certificate Status Manager に直接更新する必要はありません。

2.4.5. キーのアーカイブ、リカバリー、およびローテーション

PKI の間、秘密鍵のアーカイブを使用すると、秘密鍵が失われた場合に暗号化されたデータのリカバリーが可能になります。秘密鍵は、ハードウェア障害、パスワード忘れ、スマートカードの紛失、パスワード所有者の資格喪失など、さまざまな理由で失われる可能性があります。このようなアーカイブおよびリカバリー機能は、RHCS の Key Recovery Authority (KRA) サブシステムによって提供されます。

データの暗号化にのみ使用されるキーのみをアーカイブする必要があります。特に署名キーは決してアーカイブされるべきではありません。署名キーのコピーが 2 つあると、誰がキーを使用したかを確実に特定することができなくなります。2 番目にアーカイブされたコピーを使用して、元のキー所有者のデジタル ID を偽装することができます。

2.4.5.1. キーのアーカイブ

KRA で提供されるキーアーカイブメカニズムには、以下の 2 つのタイプがあります。

- **クライアント側のキー生成**: このメカニズムを使用すると、クライアントは CRMF 形式で CSR を生成し、登録とキーのアーカイブのために (適切な KRA セットアップを使用して) CA に要求を送信します。『Red Hat Certificate System 管理ガイド』の『CRMFPopClient を使用した CSR の作成』セクションを参照してください。
- **サーバー側の鍵生成**: このメカニズムでは、適切に装備された証明書登録プロファイルにより、PKI キーが KRA で生成され、任意で新しく発行された証明書とともにアーカイブされます。『Red Hat Certificate System 管理ガイド』の『サーバー側の鍵生成を使用した CSR の生成』セクションを参照してください。

アーカイブが設定されている場合、KRA は秘密鍵を自動的にアーカイブします。

KRA は秘密鍵を安全な鍵リポジトリーに格納します。各キーは暗号化され、キーレコードとして保存され、一意の鍵 ID が付与されます。

鍵のアーカイブコピーは、KRA のストレージキーでラップされます。ストレージ証明書の対応する秘密鍵ペアを使用することによってのみ、復号またはラップ解除が可能になります。1 つ以上のキーリカバリー (または KRA) エージェントの証明書の組み合わせは、KRA で鍵のリカバリーを完了して秘密鍵を取得し、その証明書を使用してアーカイブされた秘密鍵を復号または再作成できるようにします。「コマンドラインでのエージェント承認キーリカバリーの設定」 を参照してください。

KRA インデックスは、キー番号、所有者名、および公開鍵のハッシュ別に保存されるため、非常に効率的な検索を可能にします。キーリカバリーエージェントには、キーレコードの挿入、削除、および検索を行うための特権があります。

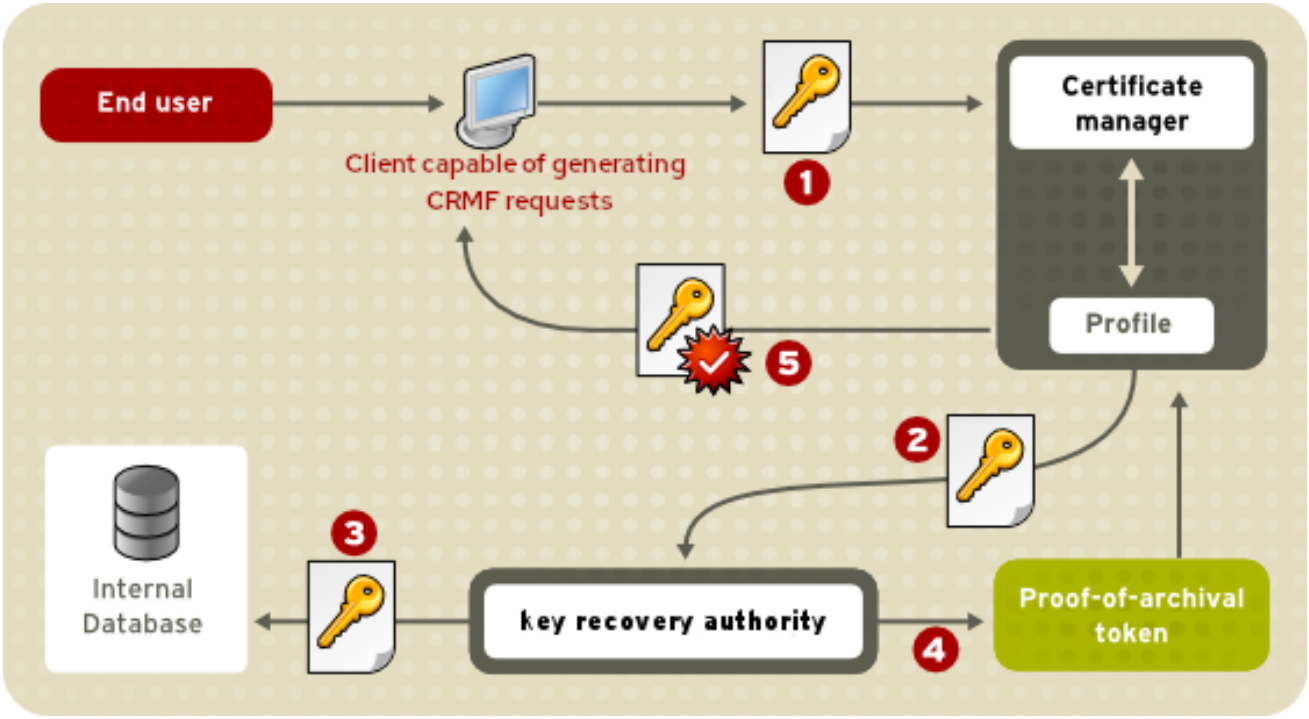
- キーリカバリーエージェントがキー ID で検索されると、その ID に対応するキーのみが返されます。
- ユーザー名でエージェントを検索すると、その所有者に属する保存されたすべてのキーが返されます。
- 証明書の公開鍵でエージェントを検索すると、対応する秘密鍵のみが返されます。

Certificate Manager がキーのアーカイブオプションを含む証明書要求を受信すると、自動的に要求を KRA に転送して暗号鍵をアーカイブします。秘密鍵はトランスポートキーで暗号化され、KRA は暗号化されたコピーを受け取り、キーをキーリポジトリーに保存します。キーをアーカイブするには、KRA は以下の 2 つの特別なキーペアを使用します。

- トランスポートキーペアと対応する証明書。
- ストレージキーペア

図2.2「クライアント側のキー生成における鍵アーカイブプロセスの仕組み」 は、サーバー側の鍵の生成時に、最終的にエンティティーが証明書を要求する際に鍵のアーカイブプロセスがどのように発生するかを示しています。

図2.2 クライアント側のキー生成における鍵アーカイブプロセスの仕組み



1. クライアントは CRMF リクエストを生成し、CA の登録ポータルを介して送信します。

    a. クライアントの秘密鍵は CRMF リクエスト内にラップされ、KRA によってだけラップできます。

2. CA は、キーアーカイブオプションを使用した CRMF リクエストであることを検出し、秘密キーアーカイブのためにリクエストを KRA に転送します。

3. KRA は、ユーザーの秘密鍵を復号/ラップ解除し、秘密鍵が公開鍵に対応していることを確認した後、内部 LDAP データベースに保存する前に再度暗号化/ラップします。

4. 秘密鍵が正常に保存されると、KRA は、鍵が正常にアーカイブされたことを確認する CA に応答します。

5. CA は、証明書情報コンテンツの作成と検証のために、登録プロファイルフレームワークにリクエストを送信します。すべてが渡されると、証明書を発行し、応答でエンドエンティティーに送り返します。

2.4.5.2. キーのリカバリー

KRA は、**agent-initiated key recovery** をサポートします。エージェントが開始するリカバリーとは、指定されたリカバリーエージェントが KRA エージェントサービスポータルのキーリカバリーフォームを使用して、キーリカバリー要求を処理および承認する場合です。特定の数のエージェントを承認すると、システムは、キーの所有者が利用できない、またはキーが失われた場合にキーを回復できます。

キーリカバリーエージェントは、KRA エージェントサービスポータルを介して、秘密暗号化キーと関連する証明書をまとめて承認し、PKCS #12 パッケージに取得して、クライアントにインポートできます。

キーリカバリー許可では、キーリカバリーエージェントの 1 つが、必要なすべてのリカバリーエージェントに、差し迫ったキーリカバリーについて通知します。すべてのリカバリーエージェントは、KRA キーリカバリーポータルにアクセスします。エージェントの 1 つが、キーリカバリープロセスを開始します。KRA はエージェントへの通知を返します。これには、エージェントの認証に必要な特定のキーリカバリー要求を指定するリカバリー承認参照番号が含まれます。各エージェントは参照番号を使用し、キーの復元を個別に承認します。

KRA は **非同期リカバリー** をサポートします。つまり、リカバリープロセスの各ステップ (最初の要求と後続の承認または承認または拒否) は、キーエントリーの KRA の内部 LDAP データベースに保存されます。元のブラウザーセッションが閉じられたり、KRA がシャットダウンしている場合でも、リカバリープロセスのステータスデータを取得することができます。エージェントは、参照番号を使用せずにキーをリカバリーすることができます。

この非同期リカバリーオプションは、図2.3「非同期リカバリー」 で説明されています。

図2.3 非同期リカバリー

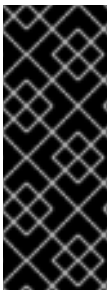
KRA は、認可のステータスキーリカバリープロセスを開始したエージェントに通知します。

すべての承認を入力すると、KRA は情報をチェックします。提示された情報が正しい場合は、要求されたキーを取得し、PKCS #12 パッケージの形式で、キーリカバリープロセスを開始したエージェントに、対応する証明書を返します。

警告

PKCS #12 パッケージには、暗号化された秘密鍵が含まれます。キーの不正使用のリスクを最小限に抑えるには、リカバリーエージェントはセキュアな方法を使用して PKCS #12 パッケージおよびパスワードを鍵受信者に提供する必要があります。このエージェントは、PKCS #12 パッケージを暗号化するために適切なパスワードを使用し、適切な配信メカニズムを設定する必要があります。

キーリカバリーエージェントスキーム は、キーリカバリーエージェントが属するグループを認識するように KRA を設定し、アーカイブされた鍵を復元する前にキーリカバリー要求を承認するために必要なこれらのリカバリーエージェントの数を指定します。



重要

上記の情報は、Firefox などの Web ブラウザーを使用することを指します。ただし、KRA の使用に重要な機能は、Red Hat Enterprise Linux 7 プラットフォームでリリースされた Firefox バージョン 31.6 に含まれなくなりました。このような場合は、**pki** ユーティリティーを使用してこの動作を複製する必要があります。詳細は、pki(1) および pki-key(1) の man ページ、または CRMFPopClient --help および man CMCRequest を実行します。

KRA は非対称鍵を保存する他に、ボリューム暗号化シークレット、パスワード、パスフレーズなどの対称鍵やシークレットも格納することができます。**pki** ユーティリティーは、これらのタイプのシークレットの保存および取得を可能にするオプションをサポートします。

2.4.5.3. KRA トランスポートのキーローテーション

KRA トランスポートローテーションにより、現在のトランスポートキーと新しいトランスポートキーを使用して、CA サブシステムインスタンスと KRA サブシステムインスタンスとの間のシームレスな移行が可能になります。これにより、移行時に古いトランスポートキーと新しいトランスポートキーの両方を操作できるようにすることで、KRA トランスポートキーを定期的にローテーションしてセキュリティーを強化できます。個々のサブシステムインスタンスは順番に設定され、他のクローンはダウンタイムなしでサービスを継続します。

KRA トランスポートキーローテーションプロセスでは、新しいトランスポートキーペアが生成され、証明書要求が送信され、新しいトランスポート証明書が取得されます。2 番目のトランスポートキーのサポートを提供するために、新しいトランスポートキーペアと証明書を KRA 設定に含める必要があります。KRA が 2 つのトランスポートキーをサポートすると、管理者は CA を新しいトランスポートキーに移行できます。古いトランスポートキーの KRA サポートは、すべての CA が新しいトランスポートキーに移動した後に削除できます。

KRA トランスポートキーローテーションを設定するには、以下を実行します。

1. 新しい KRA トランスポートの鍵および証明書の生成
2. 新しいトランスポートキーおよび証明書の KRA クローンへの転送
3. 新しい KRA トランスポート証明書を使用した CA 設定の更新
4. 新しいトランスポートキーおよび証明書のみを使用するように KRA 設定を更新

これにより、KRA トランスポート証明書のローテーションが完了し、影響を受ける CA および KRA がすべて新しい KRA 証明書のみを使用します。上記の手順の実行方法は、以下の手順を参照してください。

新しい KRA トランスポートキーおよび証明書の生成

1. KRA トランスポート証明書を要求します。
   a. KRA を停止します。

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

   または (**nuxwdog watchdog** を使用している場合)

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

b. KRA NSS データベースディレクトリーに移動します。

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

c. サブディレクトリーを作成し、すべての NSS データベースファイルを保存します。以下に例を示します。

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

d. **PKCS10Client** ユーティリティーを使用して新しいリクエストを作成します。以下に例を示します。

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

または、**certutil** ユーティリティーを使用します。以下に例を示します。

```
# 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. CA エンドエンティティーページの **手動データリカバリーマネージャートランスポート証明書の登録** ページで、トランスポート証明書要求を送信します。

f. End-Entity 取得ページで要求ステータスをチェックして、送信した要求のエージェント承認が証明書を取得するのを待ちます。

2. CA Agent Services インターフェイスを使用して KRA トランスポート証明書を承認します。

3. KRA トランスポート証明書を取得します。

a. KRA NSS データベースディレクトリーに移動します。

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

b. End-Entity 取得ページで要求ステータスをチェックして、送信した要求のエージェント承認が証明書を取得するのを待ちます。

c. 新しい KRA トランスポート証明書が利用可能になったら、その Base64 でエンコードされた値をテキストファイル (例: ***cert-serial_number*.txt** ファイル) に貼り付けます。ヘッダー (**-----BEGIN CERTIFICATE-----**) またはフッター (**-----END CERTIFICATE-----**) を追加しないでください。

4. KRA トランスポート証明書を要求します。

a. KRA NSS データベースディレクトリーに移動します。

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

b. トランスポート証明書を KRA NSS データベースにインポートします。

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

5. KRA トランスポート証明書設定を更新します。

   a. KRA NSS データベースディレクトリーに移動します。

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

   b. 新しい KRA トランスポート証明書がインポートされていることを確認します。

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

   c. **/var/lib/pki/pki-kra/kra/conf/CS.cfg** ファイルを開き、以下の行を追加します。

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

新しいトランスポートキーおよび証明書の KRA クローンへの伝搬

1. KRA を起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

2. クローンに伝播するために、新しいトランスポートキーと証明書を抽出します。

   a. KRA NSS データベースディレクトリーに移動します。

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

   b. KRA を停止します。

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

   または (**nuxwdog watchdog** を使用している場合)

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

   c. 新しい KRA トランスポート証明書が存在することを確認します。

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

   d. KRA の新規トランスポートキーおよび証明書をエクスポートします。

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

e. エクスポートした KRA トランスポート鍵および証明書を確認します。

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

3. 各 KRA クローンで以下の手順を実行します。

a. トランスポートキーおよび証明書を含む **transport.p12** ファイルを KRA クローンの場所にコピーします。

b. クローンの NSS データベースディレクトリーに移動します。

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

c. KRA のクローンを停止します。

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

または (**nuxwdog watchdog** を使用している場合)

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

d. クローン NSS データベースの内容を確認します。

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

e. クローンの新規トランスポートキーと証明書をインポートします。

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

f. クローンの **/var/lib/pki/pki-kra/kra/conf/CS.cfg** ファイルに以下の行を追加します。

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

g. KRA のクローンを起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

新しい KRA トランスポート証明書を使用した CA 設定の更新

1. CA に組み込む新しい KRA トランスポート証明書をフォーマットします。

a. 直前の手順で KRA トランスポート証明書を取得する際に作成した ***cert-serial_number*.txt** KRA トランスポート証明書ファイルを取得します。

b. ***cert-serial_number*.txt** に含まれる Base64 でエンコードされた証明書を 1 行のファイルに変換します。

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

2. CA と、上記の KRA に対応するすべてのクローンに対して以下を行います。

   a. CA を停止します。

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

   または (**nuxwdog watchdog** を使用している場合)

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

   b. **/var/lib/pki/pki-ca/ca/conf/CS.cfg** ファイルで、以下の行に含まれる証明書を見つけます。

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

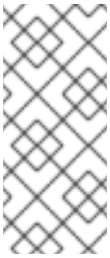
   その証明書を、***cert-one-line-serial_number*.txt** に含まれる証明書に置き換えます。

   c. CA を起動します。

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

   または (**nuxwdog watchdog** を使用している場合)

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



注記

CA とそのすべてのクローンが新しい KRA トランスポート証明書で更新されている間、移行を完了した CA インスタンスは新しい KRA トランスポート証明書を使用し、まだ更新されていない CA インスタンスは引き続き古い KRA トランスポート証明書を使用します。対応する KRA とそのクローンが両方のトランスポート証明書を使用するよう更新されているため、ダウンタイムは発生しません。

新しいトランスポートキーおよび証明書のみを使用するように KRA 設定を更新

KRA とその各クローンについて、以下を実行します。

1. KRA NSS データベースディレクトリーに移動します。

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

2. KRA を停止します。

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

   または (**nuxwdog watchdog** を使用している場合)

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

3. 新しい KRA トランスポート証明書がインポートされていることを確認します。

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

4. **/var/lib/pki/pki-kra/kra/conf/CS.cfg** ファイルを開き、以下の行に含まれる ***nickName*** の値を見つけます。

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

nickName の値を、以下の行に含まれる ***newNickName*** 値に置き換えます。

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

これにより、**CS.cfg** ファイルに以下の行が含まれます。

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

5. **/var/lib/pki/pki-kra/kra/conf/CS.cfg** から以下の行を削除します。

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

6. KRA を起動します。

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

または (**nuxwdog watchdog** を使用している場合)

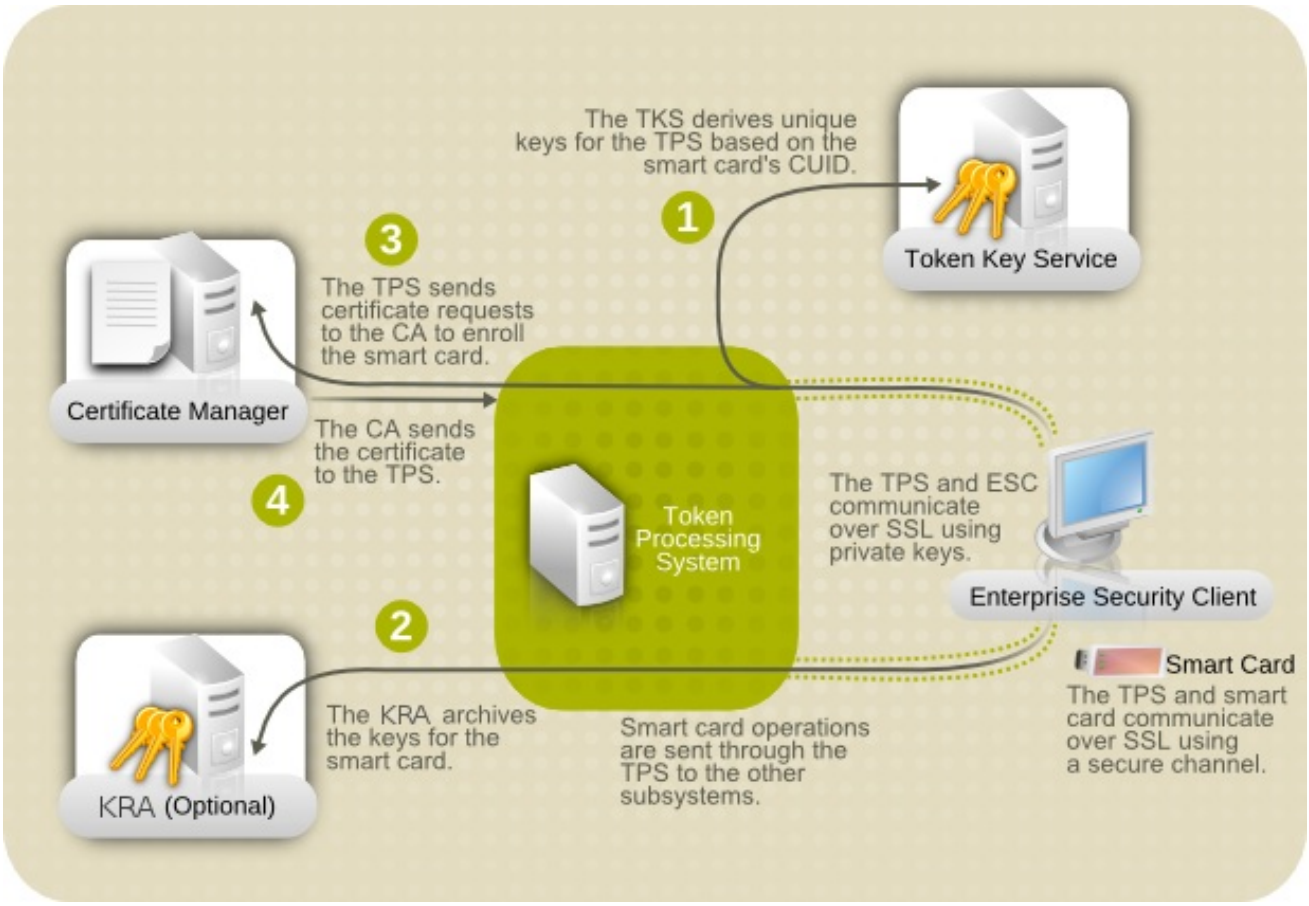
```
# systemctl start pki-tomcatd-nuxwdog@pki-kra.service
```

2.5. CERTIFICATE SYSTEM を使用したスマートカードトークン管理

*スマートカード*は、暗号化証明書および鍵を含むハードウェア暗号化デバイスです。ユーザーは、セキュアな Web アクセスやセキュアなメールなどの操作に参加できます。また、Red Hat Enterprise Linux などのさまざまなオペレーティングシステムにログインする認証デバイスとしても機能します。サービスの有効期間全体でこれらのカードまたはトークンを管理することは、*トークン管理システム* (TMS) によって行われます。

TMS 環境には、認証局 (CA)、トークンキーサービス (TKS)、およびトークン処理システム (TPS) と、サーバー側のキー生成およびキーのアーカイブとリカバリーのための任意のキーリカバリー機関 (KRA) が必要です。OCSP (Online Certificate Status Protocol) を使用して、オンライン証明書ステータス要求に対応する CA と連携することもできます。この章では、Red Hat Certificate System のスマートカード管理機能を提供する TKS システムおよび TPS システムと、ユーザー側から TMS と連携する Enterprise Security Client (ESC) の概要を説明します。

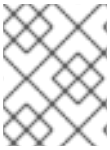
図2.4 TMS スマートカードの管理方法



2.5.1. トークンキーサービス (TKS)

Token Key Service (TKS) は、1つ以上のマスターキーを管理します。これは **マスターキー** を維持し、キー資料にアクセスできる TMS 内の唯一のエンティティーです。運用環境では、有効な各スマートカードトークンには、マスターキーと、カード (CUID) に固有の ID の両方から派生した対称鍵のセットが含まれます。

最初に、対称キーのデフォルト (製造元のマスターキーごとにのみ一意) のセットが、製造元によって各スマートカードで初期化されます。このデフォルトのセットは、**Key Changeover** 操作を実行して TKS で新しいマスターキーを生成することで、デプロイメントサイトで変更する必要があります。マスターキーの唯一の所有者として、スマートカードの CUID が付与されると、TKS はその特定のスマートカードにある対称キーのセットを導出できます。これにより、TKS は、TMS と個々のスマートカード間の安全な通信用にセッションベースの **セキュアチャネル** を確立できます。



注記

TKS が管理するデータの機密性により、TKS はアクセスが制限されたファイアウォールの背後に設定する必要があります。

2.5.1.1. マスターキーおよびキーセット

TKS は、複数のスマートカードの鍵セットをサポートします。各スマートカードベンダーは、スマートカードトークンストックに対して異なるデフォルト (開発者) の静的キーセットを作成し、TKS には、空白のトークンのフォーマットプロセスを開始するための静的キーセット (メーカーごと) が装備されています。

空白のスマートカードトークンのフォーマットプロセス中に、Java アプレットと一意に派生した対称キーセットがトークンに挿入されます。TKS がサポートする各マスターキー (場合によっては **keySet**

と呼ばれます) には、TKS 設定ファイル (**CS.cfg**) にエントリーセットがあります。各 TPS プロファイルには、TMS とスマートカードトークン間のセッション固有のキーのセットによってセキュリティーが保護された Secure Channel の確立を本質的に担当する、一致するキー導出プロセスの適切な TKS キーセットに登録を指示する設定が含まれています。

TKS では、マスターキーは TPS の参照用に名前付きの keySet によって定義されます。TPS では、登録タイプ (内部または外部の登録) に応じて、keySet は TPS プロファイルで指定されます。または、keySet Mapping Resolver で決定されます。

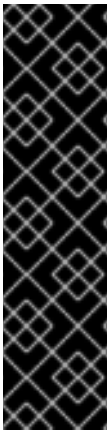
2.5.1.2. キー階層 (共有キートランスポート)

キーセレモニー は、機密性の高いキーをある場所から別の場所に安全な方法で転送するためのプロセスです。1 つのシナリオでは、非常に安全なデプロイメント環境では、外部へのネットワークのないセキュアな vault でマスターキーを生成できます。組織が、異なる物理マシンに TKS インスタンスと TPS インスタンスを持つ場合があります。いずれの場合も、キーを信頼できる人が 1 人もいないという前提の下で、Red Hat Certificate System TMS は、安全な鍵の送信を管理する **tkstool** と呼ばれるユーティリティーを提供します。

2.5.1.3. キー更新 (キーの切り替え)

Global Platform 準拠のスマートカードをファクトリーに作成すると、製造元はデフォルトの対称鍵をトークンに作成します。TKS は、最初にこの対称鍵を使用するように設定されています (TKS 設定のベンダーごとに KeySet エントリーが 1 つ)。しかし、これらの対称鍵は同一ストックのスマートカードに固有のものではなく、周知の鍵であるため、トークンを操作できるエンティティーセットを制限するために、製造元で共有されていない、トークンごとに固有のセットに置き換えることが強く推奨されます。

キーの変更は、Token Key Service サブシステムのサポートにより行われます。TKS の関数の 1 つは、以前に説明したスマートカードトークンキーが派生しているマスターキーを確認することです。TKS の制御下に複数のマスターキーが存在する可能性があります。



重要

このキー切り替えプロセスがトークンに対して実行されると、デフォルトのキーセットが有効でなくなったため、トークンは将来使用できなくなる可能性があります。トークンをプロビジョニングした TPS および TKS システムが有効である限り、鍵は基本的に有効です。このため、マスターキーのいずれかが古くなっている場合でも、すべてのマスターキーを保持することが不可欠です。

TKS の古いマスターキーを無効にして、適切に制御できますが、無効にしたトークンがプランの一部である限り削除しないでください。トークンキーを元のキーセットに戻すためのサポートがあります。これは、ある種のテストシナリオでトークンを再利用する場合に実行可能です。

2.5.1.4. APDU およびセキュアチャンネル

Red Hat Certificate System Token Management System (TMS) は *GlobalPlatform* スマートカード仕様をサポートしており、マスターキーを管理する Token Key System (TKS) と、*Application Protocol Data Units* (APDU) を使用してスマートカード (トークン) と通信する Token Processing System (TPS) で Secure Channel の実装が行われます。

APDU には、以下の 2 つのタイプがあります。

- **コマンド APDU** (TPS からスマートカードに送信)

- **レスポンス APDU** (スマートカードから TPS にレスポンスとして送信)

APDU コマンドの開始は、クライアントがアクションを実行し、要求のために Certificate System サーバーに接続したときにトリガーされる場合があります。安全なチャネルは、TPS からスマートカードトークンに送信される **InitializeUpdate** APDU で始まり、**ExternalAuthenticate** APDU で完全に確立されます。次に、トークンと TMS の両方が、セッションキーと呼ばれる共有シークレットのセットを確立します。これは、通信の暗号化と認証に使用されます。この認証および暗号化された通信チャネルは Secure Channel と呼ばれます。

TKS は、一意の対称オントークンスマートカードキーのセットを取得できるマスターキーにアクセスできる唯一のエンティティーであるため、Secure Channel は、TMS と個々のトークンとの間の適切に保護された通信を提供します。チャンネルの接続解除には、新しいチャンネルに対する新しいセッションキーの再確立が必要になります。

2.5.2. トークン処理システム (TPS)

Token Processing System (TPS) は、スマートカード証明書登録の登録認証局です。これは、クライアント側のスマートカードトークンと対話するユーザー中心のエンタープライズセキュリティークライアント (ESC) と、認証局 (CA) やキーリカバリー機関 (KRA) などの Certificate System バックエンドサブシステムとの間のパイプ役として機能します。

TMS では、スマートカードを管理するために TPS が必要です。これは、TPS が APDU コマンドと応答を理解する唯一の TMS エンティティーであるためです。TPS は、スマートカードにコマンドを送信して、ユーザーやデバイスなどの特定のエンティティーのキーと証明書を生成および保存できるようにします。スマートカード操作は TPS を経由して、証明書を生成するために CA や、鍵を生成、アーカイブ、または復元する KRA などのアクションのために適切なサブシステムに転送されます。

2.5.2.1. Coolkey アプレット

Red Hat Certificate System には、TMS 対応スマートカードトークンで実行するために作成された **Coolkey** Java アプレットが含まれています。**Coolkey** アプレットは、証明書およびキー関連の操作を処理する PKCS#11 モジュールに接続します。トークンフォーマットの操作時に、このアプレットは Secure Channel プロトコルを使用してスマートカードトークンに挿入され、設定ごとに更新できます。

2.5.2.2. トークン操作

Red Hat Certificate System の TPS は、スマートカードのエンドユーザーの代わりにスマートカードをプロビジョニングすることができます。Token Processing System は、以下の主要なトークン操作をサポートします。

- **トークンフォーマット**: フォーマット操作は、適切な Coolkey アプレットをトークンにインストールします。アプレットは、後続の暗号鍵と証明書を後で配置できるプラットフォームを提供します。
- **トークンの登録**: 登録操作により、必要な暗号鍵と暗号証明書でスマートカードが生成されます。この資料により、スマートカードのユーザーは、セキュアな Web サイトアクセスやセキュアなメールなどの操作に参加できます。登録には 2 つのタイプがサポートされています。これは、グローバルに設定されます。
  - **内部登録**: プロファイル **マッピングリゾルバー** で決定される TPS プロファイルで登録します。
  - **外部登録**: ユーザーの LDAP レコードのエントリーで、TPS プロファイルで登録を行います。

- **トークンの PIN リセット**: トークンの PIN リセット操作により、トークンのユーザーはトークンへのログインに使用される新しい PIN を指定でき、暗号化操作を実行できます。

以下の他の操作は、上記の主な操作に対して補助または固有の操作として考慮できます。これらは、関連する設定ごとに、またはトークンの状態によってトリガーできます。

- **キー生成**: 各 PKI 証明書は、公開鍵と秘密鍵のペアで設定されます。Red Hat Certificate System では、**TPS プロファイル** の設定に応じて、鍵の生成は 2 つの方法で実行できます。

  - **トークン側のキー生成**: PKI キーペアがスマートカードトークンで生成されます。トークン側でキーペアを生成すると、キーのアーカイブが許可 **されません**。

  - **サーバー側のキー生成**: PKI キーペアは TMS サーバー側で生成されます。その後、キーペアはセキュアチャネルを使用してトークンに送信されます。サーバー側で鍵のペアを生成すると、キーのアーカイブが可能になります。

- **証明書の更新**: この操作により、以前登録したトークンが同じ鍵を再度使用している間にトークンに現在発行された証明書を使用できるようになります。これは、古い証明書の有効期限が切れて、新しい証明書を作成したいが、元のキーマテリアルを維持したい場合に役立ちます。

- **証明書失効リスト**: TPS プロファイル設定またはトークンの状態をもとに、証明書失効リストをトリガーできます。

  通常、証明書を発行した CA のみがそれを取り消すことができます。つまり、CA を廃止すると、特定の証明書を取り消すことができなくなります。ただし、トークンの失効要求をリタイアした CA にルーティングしながら、登録などの他のすべての要求を新しいアクティブな CA にルーティングすることは可能です。このメカニズムは **取り消しルーティン** と呼ばれます。

- **トークンキーの切り替え**: フォーマット操作によってトリガーされるキーの変更操作により、トークンの内部キーを、デフォルトの開発者キーセットから、トークン処理システムの開発者により制御される新しいキーセットに変更できます。開発者キーセットはテスト状況に適したものであるため、通常、実際のデプロイメントシナリオでこれを行います。

- **アプレットの更新**: TMS デプロイメントでは、必要に応じて Coolkey スマートカードアプレットを更新またはダウングレードできます。

2.5.2.3. TPS プロファイル

Certificate System Token Processing System サブシステムは、スマートカードトークンの管理を容易にします。トークンは TPS によってプロビジョニングされ、空白の状態からフォーマット済みまたは登録済みの状態になります。フォーマットされたトークンには、TPS でサポートされる **CoolKey** アプレットが含まれますが、登録済みトークンは、必要な証明書と暗号化キーを使用して個人にパーソナライズされます (*バインディング*と呼ばれるプロセス)。この完全にプロビジョニングされたトークンは、暗号化操作に使用する準備ができています。

TPS は *プロファイル* も管理できます。トークンプロファイルの概念は以下に関連します。

- トークンのフォーマットまたは登録を行う手順。

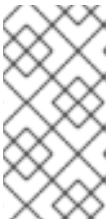
- 操作が正常に完了した後、終了したトークンに含まれる属性。

以下の一覧で、一意のトークンプロファイルを設定する数量について説明します。

- TPS がユーザーの認証 LDAP データベースに接続する方法。

- このトークン操作にはユーザー認証が必要ですか。その場合に使用する認証マネージャー。

- TPS は、証明書を取得する Certificate System CA にどのように接続しますか。
- このトークンで生成された秘密鍵と公開鍵はどのように生成されているか。トークン側またはサーバー側で生成されているか。
- 秘密鍵と公開鍵を生成する際に使用する鍵のサイズ (ビット単位)。
- このトークンで証明書を生成するのに使用される証明書登録プロファイル (CA によるプロビジョニング)



注記

この設定により、トークンに書き込まれる証明書の最終構造が決定されます。証明書に含まれる拡張機能に基づいて、さまざまな用途に応じてさまざまな証明書を作成できます。たとえば、1 つの証明書はデータ暗号化に特化でき、別の証明書は署名操作に使用できます。

- トークンで必要となる Coolkey アプレットのバージョン
- 登録操作のためにこのトークンに配置される証明書の数

上記および他の多くのトークンタイプまたはプロファイルごとに設定できます。利用可能な設定オプションの一覧は、『Red Hat Certificate System 管理ガイド』を参照してください。

考慮すべきもう 1 つの質問は、ユーザーによってプロビジョニングされている特定のトークンがどのように個々のトークンプロファイルにマップされるかです。登録には、以下の 2 つのタイプがあります。

- **Internal Registration**: この場合、TPS プロファイル (**tokenType**) は、プロファイル *Mapping Resolver* で決定されます。このフィルターベースのリゾルバーは、トークンが提供するデータをすべて考慮し、ターゲットプロファイルを決定するように設定できます。
- **外部登録**: 外部登録使用する場合、プロファイル (名前のみ。実際のプロファイルは、内部登録で使用されるものと同じ方法で TPS で定義されます) は、認証中に取得される各ユーザーの LDAP レコードで指定されます。これにより、TPS はユーザー情報が保存される外部登録 Directory Server からキーの登録およびリカバリー情報を取得できます。これにより、TPS 内部登録メカニズムに固有の登録、失効、および復旧ポリシーを上書きする制御が可能になります。外部登録に関連するユーザー LDAP レコード属性名は設定可能です。

  グループ証明書という概念が必要な場合には、外部登録が役立ちます。この場合、グループ内のすべてのユーザーには、共有証明書および鍵をダウンロードするために LDAP プロファイルに特別なレコードを設定できます。

使用する登録は、TPS インスタンスごとにグローバルに設定されます。

2.5.2.4. トークンデータベース

Token Processing System は、LDAP トークンデータベースストアを使用します。これは、アクティブなトークンとその証明書の一覧を維持し、各トークンの現在の状態を追跡します。ブランドの新しいトークンは *Uninitialized* とみなされますが、完全登録されたトークンは *Enrolled* と見なされます。このデータストアは常に更新され、トークンの処理時に TPS によって確認されます。

2.5.2.4.1. トークンの状態および移行

Token Processing System は、現在のトークンステータスとトークンで実行できるアクションを定義するために内部データベースのステータスを保存します。

2.5.2.4.1.1. トークンの状態

以下の表では、可能なトークン状態をすべて紹介します。

表2.9 考えられるトークンの状態

名前	コード	ラベル
FORMATTED	0	フォーマット済み (初期化されていない)
DAMAGED	1	物理的に破損
PERM_LOST	2	永続的に失われる
SUSPENDED	3	一時停止 (一時的に失われる)
ACTIVE	4	アクティブ
TERMINATED	6	終了
UNFORMATTED	7	未フォーマット

コマンドラインインターフェイスは、上記の名前を使用してトークンの状態を表示します。グラフィカルインターフェイスは、代わりに Label を使用します。

注記

上記の表には、コード **5** 付きの状態は含まれません。以前は削除された状態に属していました。

2.5.2.4.1.2. グラフィカルインターフェイスまたはコマンドラインインターフェイスを使用したトークン状態の移行完了

各トークンの状態には、遷移できる次の状態の数が制限されています。たとえば、トークンは **FORMATTED** から **ACTIVE** または **DAMAGED** に状態を変更できますが、**FORMATTED** から **UNFORMATTED** に移行することはできません。

さらに、トークンが遷移できる状態のリストは、遷移がコマンドラインまたはグラフィカルインターフェイスを使用して手動でトリガーされるか、トークン操作を使用して自動的にトリガーされるかによって異なります。許可されている手動遷移のリストは、**tokendb.allowedTransitions** プロパティーに格納され、**tps.operations.allowedTransitions** プロパティーコントロールは、ークン動作によってトリガ遷移を可能にしました。

手動およびトークン操作ベースの両方の移行のデフォルト設定は、**/usr/share/pki/tps/conf/CS.cfg** 設定ファイルに保存されます。

2.5.2.4.1.2.1. コマンドラインインターフェイスまたはグラフィカルインターフェイスを使用したトークン状態の移行

コマンドラインまたはグラフィカルインターフェイスで許可される移行はすべて、**tokendb.allowedTransitions** プロパティーを使用して TPS 設定ファイルに記載されています。

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

プロパティーには、トランジションのコンマ区切りリストが含まれます。それぞれの移行は、***<current code>:<new code>*** 形式で記述されます。このコードは 表2.9「考えられるトークンの状態」 で説明されています。デフォルト設定は **/usr/share/pki/tps/conf/CS.cfg** に保存されます。

以下の表では、可能な各移行の詳細を説明します。

表2.10 可能な手動トークン状態遷移

遷移	現在の状態	次の状態	説明
0:1	FORMATTED	DAMAGED	このトークンは物理的に破損しています。
0:2	FORMATTED	PERM_LOST	このトークンは永続的に失われています。
0:3	FORMATTED	SUSPENDED	このトークンは一時停止されています (一時的で失われています)。
0:6	FORMATTED	TERMINATED	このトークンは終了しました。
3:2	SUSPENDED	PERM_LOST	この一時停止されたトークンは永続的に失われています。
3:6	SUSPENDED	TERMINATED	この一時停止されたトークンは終了しています。
4:1	ACTIVE	DAMAGED	このトークンは物理的に破損しています。
4:2	ACTIVE	PERM_LOST	このトークンは永続的に失われています。
4:3	ACTIVE	SUSPENDED	このトークンは一時停止されています (一時的で失われています)。
4:6	ACTIVE	TERMINATED	このトークンは終了しました。
6:7	TERMINATED	UNFORMATTED	このトークンを再利用します。

以下の移行は、トークンの元の状態に応じて自動的に生成されます。トークンが元々 **FORMATTED** で、**SUSPENDED** となると、**FORMATTED** 状態にのみ戻ることができます。トークンが元々 **ACTIVE** で、**SUSPENDED** になると、**ACTIVE** 状態のみに戻ることができます。

表2.11 自動的にトリガーされるトークンの状態遷移

遷移	現在の状態	次の状態	説明
3:0	SUSPENDED	FORMATTED	この一時停止 (一時的な損失) トークンがあります。
3:4	SUSPENDED	ACTIVE	この一時停止 (一時的な損失) トークンがあります。

2.5.2.4.1.3. トークン操作を使用したトークン状態遷移

トークン操作を使用して実行できる移行はすべて、**tokendb.allowedTransitions** プロパティーを使用して TPS 設定ファイルで説明されています。

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

プロパティーには、トランジションのコンマ区切りリストが含まれます。それぞれの移行は、***<current code>:<new code>*** 形式で記述されます。このコードは 表2.9「考えられるトークンの状態」 で説明されています。デフォルト設定は **/usr/share/pki/tps/conf/CS.cfg** に保存されます。

以下の表では、可能な各移行の詳細を説明します。

表2.12 トークン操作を使用した可能なトークン状態遷移

遷移	現在の状態	次の状態	説明
0:0	FORMATTED	FORMATTED	これにより、トークンの再フォーマットやトークンのアプレット/キーのアップグレードが可能になります。
0:4	FORMATTED	ACTIVE	これにより、トークンを登録できます。
4:4	ACTIVE	ACTIVE	これにより、アクティブなトークンを再登録できます。外部の登録に便利です。
4:0	ACTIVE	FORMATTED	これにより、アクティブなトークンのフォーマットが可能になります。
7:0	UNFORMATTED	FORMATTED	これにより、空白または以前に使用されたトークンのフォーマットが可能になります。

2.5.2.4.1.4. トークンの状態および遷移ラベル

トークン状態および遷移のデフォルトラベルは、**/usr/share/pki/tps/conf/token-states.properties** 設定ファイルに保存されます。デフォルトでは、ファイルの内容は以下のようになります。

```
# 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. 許可されるトークンの状態遷移のカスタマイズ

トークン状態遷移の一覧をカスタマイズするには、**/var/lib/pki/*instance_name*/tps/conf/CS.cfg** で以下のプロパティーを編集します。

- コマンドラインまたはグラフィカルインターフェイスを使用して実行できる移行の一覧をカスタマイズするには、**tokendb.allowedTransitions**
- トークン操作を使用して許可される移行のリストをカスタマイズするには、**tps.operations.allowedTransitions**

必要に応じて、移行はデフォルトのリストから削除できますが、デフォルトの一覧にない限り、新しい移行を追加することはできません。デフォルトは **/usr/share/pki/tps/conf/CS.cfg** に保存されます。

2.5.2.4.1.6. トークンの状態と遷移ラベルのカスタマイズ

トークンの状態と遷移ラベルをカスタマイズするには、デフォルトの **/usr/share/pki/tps/conf/token-states.properties** をインスタンスフォルダー (**/var/lib/pki/*instance_name*/tps/conf/CS.cfg**) にコピーし、必要に応じてラベルを変更します。

変更は即座に有効になり、サーバーを再起動する必要はありません。TPS ユーザーインターフェイスは読み込みが必要になる場合があります。

デフォルトの状態およびラベル名に戻すには、編集した **token-states.properties** ファイルをインスタンスフォルダーから削除します。

2.5.2.4.1.7. トークンアクティビティーログ

特定の TPS アクティビティーがログに記録されます。ログファイル内の考えられるイベントは、以下の表に一覧表示されています。

表2.13 TPS アクティビティーログイベント

アクティビティー	説明
add	トークンが追加されました。
format	トークンがフォーマットされました。
enrollment	トークンが登録されました。
recovery	トークンが復元されました。
更新	トークンが更新されています。
pin_reset	トークンの PIN がリセットされました。
token_status_change	トークンステータスは、コマンドラインまたはグラフィカルインターフェイスを使用して変更されました。
token_modify	トークンが変更されました。
delete	トークンが削除されました。
cert_revocation	トークン証明書が取り消されました。
cert_unrevocation	トークン証明書は失効しませんでした。

2.5.2.4.2. トークンポリシー

内部登録の場合、各トークンをトークンポリシーのセットで制御できます。デフォルトのポリシーは以下のとおりです。

```
RE_ENROLL=YES;RENEW=NO;FORCE_FORMAT=NO;PIN_RESET=NO;RESET_PIN_RESET_TO_NO=NO;RENEW_KEEP_OLD_ENC_CERTS=YES
```

内部登録中の TPS 操作は、トークンの記録に指定したポリシーの対象です。トークンにポリシーが指定されていない場合、TPS はデフォルトのポリシーセットを使用します。

2.5.2.5. マッピングリゾルバー

Mapping Resolver は、設定可能な基準に基づいて特定のトークンに割り当てるトークンプロファイルを決定するために TPS が使用する拡張可能なメカニズムです。各マッピングリゾルバーインスタンスは設定で一意に定義でき、各操作は定義されたマッピングリゾルバーインスタンスを参照できます。

注記

マッピングリゾルバーフレームワークは、カスタムプラグインを作成するプラットフォームを提供します。ただし、プラグインの作成方法に関する説明は、本書の範囲外です。

FilterMappingResolver は、デフォルトで TPS で提供される唯一のマッピングリゾルバー実装です。これにより、各マッピングに *マッピング*のセットおよびターゲットの結果を定義できます。各マッピングにはフィルターのセットが含まれ、ここでは以下のようになります。

- 入力フィルターパラメーターがマッピング内の **すべて** のフィルターを渡すと、**target** の値が割り当てられます。
- 入力パラメーターでフィルターが失敗すると、そのマッピングはスキップされ、次の順番に試行されます。
- フィルターに指定された値がない場合は、常に合格します。
- フィルターに指定された値がある場合、入力パラメーターは完全に一致する必要があります。
- マッピングが定義される順序は重要です。合格した最初のマッピングは解決されたと見なされ、呼び出し元に返されます。

入力フィルターパラメーターは、拡張の有無にかかわらずスマートカードトークンから受け取った情報です。上記のルールに従って、**FilterMappingResolver** に対して実行されます。**FilterMappingResolver** では、以下の入力フィルターパラメーターがサポートされます。

- **appletMajorVersion** - トークン上の Coolkey アプレットのメジャーバージョン。
- **appletMinorVersion** - トークン上の Coolkey アプレットのマイナーバージョン。
- **keySet** または **tokenType**
  - **keySet** - クライアントリクエストでエクステンションとして設定できます。拡張子が指定されている場合は、フィルターの値と一致する必要があります。keySet マッピングリゾルバーは、外部登録を使用する際に keySet 値を決定することを目的としています。複数の鍵セットがサポートされる場合は、外部登録環境で Key Set Mapping Resolver が必要になります (たとえば、スマートカードトークンベンダーごとに必要です)。keySet 値は、Secure Channel を確立する上で重要な TKS のマスターキーを特定するために必要です。ユーザーの LDAP レコードにセット tokenType (TPS プロファイル) が入力されている場合は、どのカードが最終的に登録を行うかがわからないため、keySet を事前に決定することはできません。**keySetMappingResolver** は、認証前に keySet が解決できるようにすることで、問題の解決に役立ちます。
  - **tokenType** - okenType は、クライアントリクエストの拡張機能として設定できます。拡張子が指定されている場合は、フィルターの値と一致する必要があります。tokenType (TPS プロファイルとも呼ばれます) は、この時点で内部登録環境に対して決定されます。
- **tokenATR** - トークンの Answer to Reset (ATR) です。
- **tokenCUID** - start および end は、このフィルターに渡すためにトークンの Card Unique ID (CUID) の範囲を定義します。

2.5.2.6. TPS ロール

TPS は、デフォルトで以下のロールをサポートします。

- **TPS 管理者**: このロールは以下を許可します。
  - TPS トークンの管理
  - TPS 証明書およびアクティビティーの表示
  - TPS ユーザーおよびグループの管理
  - 一般的な TPS 設定の変更
  - TPS オーセンティケーターおよびコネクターの管理
  - TPS プロファイルおよびプロファイルマッピングの設定
  - TPS 監査ロギングの設定
- **TPS エージェント**: このロールは以下を許可します。
  - TPS トークンの設定
  - TPS 証明書およびアクティビティーの表示
  - TPS プロファイルのステータスの変更
- **TPS オペレーター**: このロールは以下を許可します。
  - TPS トークン、証明書、およびアクティビティーの表示

2.5.3. TKS/TPS 共有シークレット

TMS のインストール時に、トークンキーサービスとトークン処理システム間に共有された対称鍵が確立されます。この鍵の目的は、Secure Channel に不可欠であるセッションキーをラップしてアンラップすることです。

注記

現在、共有秘密鍵は、ソフトウェア暗号化データベースにのみ保持されます。Red Hat Certificate System の将来のリリースでは、ハードウェアセキュリティーモジュール (HSM) デバイスでのキーの保持をサポートする計画があります。この機能が実装されたら、キーを HSM に転送するように **tkstool** を使用して Key Ceremony を実行するように指示します。

2.5.4. Enterprise Security Client (ESC)

Enterprise Security Client は、TPS と通信し、クライアント側からスマートカードトークンを処理する Web ブラウザーと同様に HTTP クライアントアプリケーションです。ESC と TPS の間で HTTPS 接続が確立されている間、各 TLS セッション内のトークンと TMS の間でも基盤となる Secure Channel が確立されます。

2.6. RED HAT CERTIFICATE SYSTEM サービス

Certificate System には、管理者が使用できるさまざまな機能があり、個々のサブシステムと PKI 全体の保守が容易になります。

2.6.1. 通知

証明書の発行や取り消し時など、特定のイベントが発生すると、通知は指定のメールアドレスに直接送信できます。通知フレームワークには、有効化および設定できるデフォルトのモジュールが同梱されています。

2.6.2. ジョブ

自動ジョブは、定義した間隔で実行されます。

2.6.3. ログ

Certificate System と各サブシステムは、システムの監視およびデバッグを可能にするためにシステムイベントを記録する豊富なシステムおよびエラーログを生成します。すべてのログレコードは、迅速に取得するためにローカルファイルシステムに保存されます。ログは設定可能なため、ログは特定タイプのイベントおよび必要なログレベルで作成できます。

Certificate System を使用すると、ログをアーカイブしたり、監査用に配布したりする前に、ログにデジタル署名することができます。この機能により、署名後に改ざんの可能性をログファイルがチェックできるようになります。

2.6.4. 監査

Certificate System は、証明書の要求、発行、取り消し、CRL の公開など、すべてのイベントの監査ログを維持します。これらのログは署名されます。これにより、承認されたアクセスやアクティビティーの検出が可能になります。その後、外部監査人は必要に応じてシステムを監査できます。割り当てられた監査ユーザーアカウントは、署名された監査ログを表示できる唯一のアカウントです。このユーザーの証明書は、ログを署名および暗号化するために使用されます。監査ロギングは、ログに記録されるイベントを指定するよう設定されます。

2.6.5. セルフテスト

Certificate System は、システムの起動時に自動的に実行されるシステムのセルフテストフレームワークを提供し、オンデマンドで実行できます。設定可能なセルフテストのセットはすでに Certificate System に含まれています。

2.6.6. ユーザー、認可、およびアクセス制御

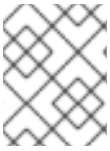
Certificate System ユーザーはグループ (ロールとも呼ばれる) に割り当てることができます。グループには、どのユーザーが所属するグループにも特権があります。ユーザーには、ユーザーが作成したサブシステムのインスタンスと、ユーザーがメンバーとなるグループの権限のみが付与されます。

認証 は、Certificate System サブシステムが、証明書プロファイルまたはサービスインターフェイスのいずれかに対して認証されているか、クライアントのアイデンティティーを検証するのに使用されます。クライアントが認証を実行するには、単純なユーザー名/パスワード、SSL/TLS クライアント認証、LDAP 認証、NIS 認証、CMC など、さまざまな方法があります。サブシステムへの任意のアクセスに対して認証を実行できます。たとえば、証明書の登録の場合、プロファイルはリクエスターが CA に対して認証する方法を定義します。

クライアントが識別および認証されると、サブシステムは **許可** チェックを実行して、特定のユーザーがサブシステムへのアクセスをどのレベルで許可されているかを判別します。

承認は、個々のユーザーに直接ではなく、グループ、ロール、パーミッションに関連付けられます。Certificate System は、グループを作成し、アクセス制御をそれらのグループに割り当てる認可フレームワークを提供します。既存のグループのデフォルトのアクセス制御は変更可能で、アクセス制御は個々のユーザーおよび IP アドレスに割り当てることができます。システムの主要な部分に対して承認のアクセスポイントが作成され、その位置ごとにアクセス制御ルールを設定できます。

2.6.6.1. デフォルトの管理ロール



注記

Red Hat Certificate System は、ユーザーに指定したパーミッションのコンテキストにおいて、**ロール** と **グループ** を区別せずに使用します。

Certificate System はデフォルトで、システムに異なるアクセスレベルを持つ 3 つのユーザータイプで設定されています。

- **管理者** は、サブシステムに対して管理または設定タスクを実行できます。
- 証明書要求の承認、トークン登録の管理、または鍵の復元など、PKI 管理タスクを実行する **エージェント**。
- 監査ログを表示および設定できる **auditors**。

注記

デフォルトでは、ブートストラップの目的で、**pkispawn** ユーティリティーの実行時、Red Hat Certificate System インスタンスの作成中に管理者特権とエージェント特権の両方を処理する管理ユーザーが作成されます。このブートストラップ管理者は、デフォルトで **caadmin** ユーザー名を使用しますが、**pkispawn** コマンドに渡す設定ファイルの ***pki_admin_uid*** パラメーターで上書きできます。ブートストラップ管理者が、最初の管理者およびエージェントユーザーを作成します。この操作には、ユーザーおよびグループの管理者権限と、証明書を発行するエージェントの権限が必要です。

2.6.6.2. 組み込みサブシステムの信頼ロール

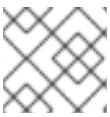
さらに、セキュリティードメインが作成されると、ドメインをホストする CA サブシステムには、**Security Domain Administrator** の特別なロールが自動的に付与されます。これにより、サブシステムはセキュリティードメインとその中のサブシステムインスタンスを管理できます。他のセキュリティードメイン管理者ロールは、異なるサブシステムインスタンスに対して作成できます。これらの特別なロールは、実際のユーザーをメンバーとして持てません。

2.7. クローン

2.7.1. クローン作成について

高可用性 のプランニングは、1 つ以上のサブシステムのクローンを利用できることで、予定外の停止やその他の問題を削減します。ホストマシンがダウンすると、複製されたサブシステムは要求を処理してサービスを実行し、マスター (元の) サブシステムからシームレスに引き継ぎ、サービスを中断することなく維持できます。

クローン作成されたサブシステムを使用すると、PKI システムのサービスを中断せずに、修復、トラブルシューティング、またはその他の管理タスクのためにシステムをオフラインにできます。



注記

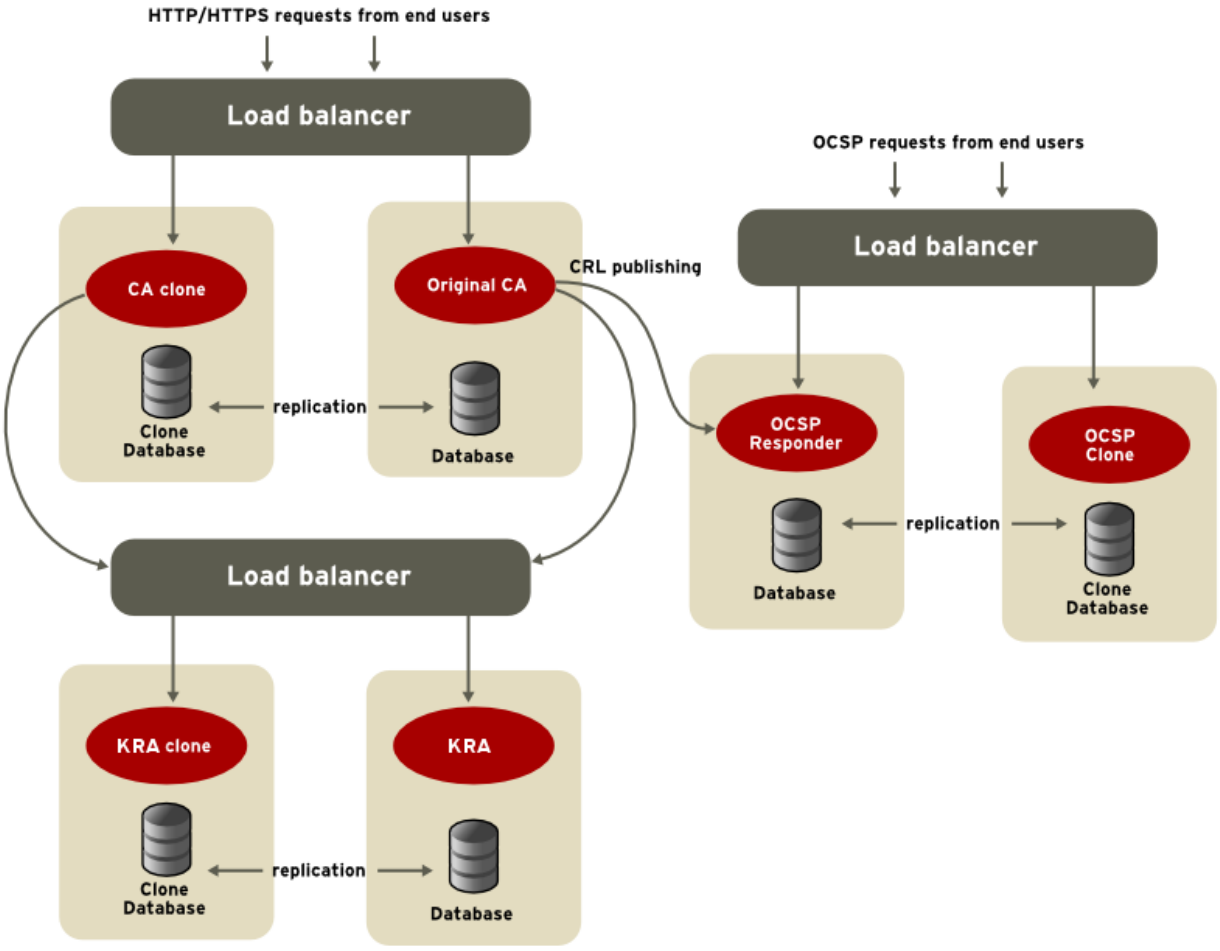
TPS 以外のすべてのサブシステムをクローンできます。

クローン作成は、証明書要求を処理するなど、異なるマシン上のインスタンスを分離することで、PKI にスケーラビリティーを提供する 1 つの方法です。マスターとそのクローンの内部データベースは相互

に複製されるため、1つのサブシステムの証明書要求またはアーカイブされたキーに関する情報は、他のすべてのサブシステムで利用できます。

通常、マスターおよびクローンインスタンスは異なるマシンにインストールされ、それらのマシンは*ロードバランサー*の内側に配置されます。ロードバランサーは、Certificate System サブシステムに送信された HTTP および HTTPS 要求を受け入れ、それらのリクエストをマスターインスタンスとクローンインスタンスとの間で適切にダイレクトします。一方のマシンに障害が発生した場合、ロードバランサーは、もう一方のマシンがオンラインに戻るまで、まだ実行中のマシンにすべての要求を透過的にリダイレクトします。

図2.5 クローン作成の例

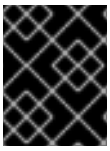


Certificate System サブシステムの前にあるロードバランサーは、高可用性システムで実際のフェイルオーバーサポートを提供するものです。ロードバランサーは、Certificate System サブシステムの一部として以下の利点があります。

- DNS ラウンドロビン。複数のサーバーに負荷を分散するネットワーク輻輳を管理する機能です。
- スティッキー SSL/TLS。これにより、ユーザーがシステムに戻して、以前使用された同じホストをルーティングできるようになります。

通常、システムのクローンを作成すると、設定サーブレットは内部 LDAP データベース間でレプリカ合意を設定します。ただし、一部のユーザーは、独自のレプリカ合意を確立して管理する場合があります。PKI インスタンスに **[Tomcat]** セクションを追加し、そのセクションに次の 2 つの **name=value** ペアを追加することにより、pkispawn 実行ファイルが変更され、これを許可するようになりました。

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



重要

このようなインストールを指定する場合は、**pkispawn** を起動する **前に** データを複製する必要があります。

2.7.2. クローンの準備

クローンサブシステムのインストール中に使用できるように、証明書を **p12** ファイルにエクスポートする必要があります。このコマンドは、クローンサブシステムをインストールまたは設定するのではなく、インストールの準備をします。**pki-server *SUBSYSTEM*-clone-prepare** コマンドを使用してサブシステム証明書をエクスポートする方法の例を次に示します。

例2.3 サブシステム証明書のエクスポート

```
[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"
----------------------------------------------------------
```

エクスポートされた **p12** ファイルに証明書が含まれていることを確認できます。次の例では、**pki pkcs12-cert-find** の出力で 4 つの証明書が返されます。

```
[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
```

エクスポートされた **p12** ファイルを使用して、クローンサブシステムを設定できるようになりました。

2.7.3. CA のクローン作成

クローン作成されたインスタンスはマスターと全く同じ秘密鍵を持つため、それらの証明書は同じになります。CA の場合、つまり、CA 署名証明書が元のマスター CA およびそのクローン作成された CA と同一であることを意味します。クライアントの観点からは、これは単一の CA に似ています。

クローンとマスターの両方のすべての CA は、証明書を発行し、失効要求を処理できます。

複製された CA を管理する主な問題は、発行する証明書にシリアル番号を割り当てる方法です。複製された CA が異なれば、異なるレートのシリアル番号を使用してトラフィックのレベルが異なる可能性があり、2 つの複製された CA が同じシリアル番号の証明書を発行しないことが不可欠です。これらのシリアル番号範囲は、各 CA の範囲を定義する共有の複製エントリーと、1 つの CA 範囲が少なくなったときに再割り当てする次の使用可能な範囲を使用して、動的に割り当ておよび管理されます。

クローン CA を使用するシリアル番号の範囲は fluid です。すべての複製された CA は、次の利用可能な範囲を定義する共通の設定エントリーを共有します。1 つの CA が利用可能な数未満の実行を開始すると、この設定エントリーをチェックし、次の範囲を要求します。エントリーは自動的に更新されます。これにより、次の CA が新規範囲を取得します。

範囲は ***begin*Number*** 属性および ***end*Number*** 属性で定義され、個別の範囲が要求および証明書のシリアル番号に対して定義されます。以下に例を示します。

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

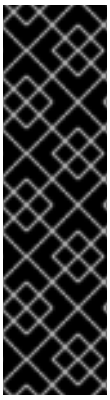
CRL を生成、キャッシュ、公開できるのは 1 つの複製 CA のみになります。これは CRL CA です。複製された他の CA に送信された CRL 要求は、すぐに CRL CA にリダイレクトされます。複製された他の CA は、以前 CRL CA によって生成された CRL を取り消し、表示、インポート、およびダウンロードで

きますが、新しい CRL を生成すると、同期の問題が発生する可能性があります。CRL 生成を別の複製された CA に移動する方法は、「CA クローンおよびマスターの変換」を参照してください。

マスター CA は、内部データベースへのレプリケーションの変更を監視することで、クローンされた CA 間の関係と情報共有も管理します。

注記

セキュリティードメインマスターである CA がクローンされた場合、そのクローン作成された CA はセキュリティードメインマスターになります。この場合、元の CA とクローンの両方が同じセキュリティードメイン設定を共有します。



重要

「カスタム設定およびクローン」で説明されているように、LDAP データベースのデータはマスターとクローン間で複製されますが、異なるインスタンス用の **設定ファイル** は複製されません。つまり、クローンの作成 **後** に変更が発生すると、KRA 接続の追加やカスタムプロファイルの作成など、**CS.cfg** ファイルに影響する変更が、クローンの設定にコピーされません。

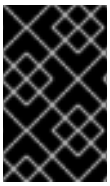
CA 設定への変更は、クローンの作成 **前** にマスターに加える必要があります (これにより、カスタム変更はクローンプロセスに含まれます)。または、作成後に設定の変更をクローンインスタンスに手動でコピーする必要があります。

2.7.4. KRA のクローン作成

KRA では、1 つの KRA にアーカイブされるすべての鍵が他の KRA の内部データベースに複製されます。これにより、キーがアーカイブされた KRA に関係なく、クローンの KRA で鍵のリカバリーを開始できます。

鍵のリカバリーが処理されると、リカバリーの記録はクローンされたすべての KRA の内部データベースに保存されます。

同期キーリカバリーではクローンでリカバリープロセスが開始できますが、開始された単一の KRA で完了する必要があります。これは、適切な承認数が KRA エージェントから取得されてからのみ復元操作がレプリケートされたデータベースに記録されるためです。それまでは、リカバリーが開始する KRA だけが、リカバリー操作について知っています。



重要

Red Hat Certificate System 9 で非推奨になり、このドキュメントでは説明されていない同期キーリカバリーメカニズムが存在します。Red Hat は、代わりに非同期鍵リカバリーの使用を推奨します。

2.7.5. 他のサブシステムのクローン作成

レプリケートされた TKS には実際の運用上の違いはありません。その 1 つで作成または維持される情報は、その他のサーバーに複製されます。

OCSP の場合、1 つの複製された OCSP のみが CRL 更新を受け取り、公開された CRL はクローンに複製されます。

2.7.6. クローンおよびキーストア

サブシステムのクローンを作成すると、同じ機能を実行するサーバープロセス 2 つが作成されます。別のサブシステムの新規インスタンスが作成され、同じ鍵と証明書を使用してその操作を実行するように設定されます。マスタークローン用の鍵を保存する場所に応じて、キーにアクセスするための方法が非常に異なります。

鍵と証明書が内部ソフトウェアトークンに保存されている場合は、初回の設定時にマスターサブシステムからエクスポートする必要があります。マスターインスタンスを設定する際に、**pkispawn** 設定ファイルに ***pki_backup_keys*** パラメーターおよび ***pki_backup_password*** パラメーターを指定して、システムキーと証明書を PKCS #12 ファイルにバックアップできます。詳細は pki_default.cfg(5) の man ページ の **BACKUP PARAMETERS** セクションを参照してください。

初期設定時に鍵がバックアップされていない場合は、「ソフトウェアデータベースからのサブシステムキーのバックアップ」 に記載されているように、**PKCS12Export** ユーティリティーを使用して PKCS #12 ファイルに鍵を抽出できます。

次に、PKCS #12 ファイルを clone サブシステムにコピーし、***pki_clone_pkcs12_password*** パラメーターおよび ***pki_clone_pkcs12_path*** パラメーターを使用して **pkispawn** 設定ファイルにその場所とパスワードを定義します。詳細は、pkispawn(8) man ページの **Installing a Clone** セクションを参照してください。特に、PKCS#12 ファイルが **pkiuser** ユーザーからアクセスできることを確認し、正しい SELinux ラベルがあることを確認します。

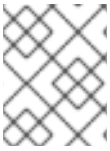
鍵と証明書がハードウェアトークンに保存されている場合は、ハードウェアトークン固有のユーティリティーを使用して鍵と証明書をコピーするか、またはトークンで直接参照する必要があります。

- SSL/TLS サーバーキーと証明書をクローンインスタンスに対して除いた、必要なすべての鍵と証明書を複製します。これらの証明書のニックネームは同じままにします。さらに、必要なすべてのルート証明書をマスターインスタンスからクローンインスタンス (チェーンやクロスペアの証明書など) にコピーします。
- トークンがネットワークベースである場合、鍵と証明書は単にトークンで利用できる必要があり、鍵と証明書はコピーする必要はありません。
- ネットワークベースのハードウェアトークンを使用する場合は、単一障害点を回避するために、ハードウェアトークンで高可用性機能が有効になっていることを確認してください。

2.7.7. LDAP とポートに関する考慮事項

「クローン作成について」で説明したように、クローン作成の動作の一部は、複製されたサブシステム間で情報を複製することです。これにより、サブシステムは同一のデータセットとレコードから機能します。つまり、レプリケートされたサブシステムの LDAP サーバーが通信できる必要があります。

Directory Server インスタンスが異なるホストにある場合は、Directory Server インスタンスが相互に接続できるように、適切なファイアウォールアクセスがあることを確認してください。

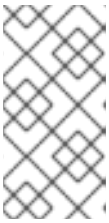


注記

クローン作成されたサブシステムとそのマスターは、共通の接尾辞間でデータを複製しつつ、別の LDAP サーバーを使用する必要があります。

サブシステムは、LDAPS ポートを介した SSL/TLS または LDAP ポートを介した標準接続のいずれかを使用して、内部データベースに接続できます。サブシステムが複製されると、複製インスタンスはそのマスターと同じ接続方法 (SSL/TLS または標準) を使用してデータベースに接続します。ただし、クローン作成では、追加のデータベース接続があります。マスターの Directory Server データベースからクローンの Directory Server データベースへの接続です。この接続には、以下の 3 つの接続オプションがあります。

- マスターが SSL/TLS を使用してデータベースに接続する場合、クローンは SSL/TLS を使用し、マスター/クローンの Directory Server データベースはレプリケーションに SSL/TLS 接続を使用します。
- マスターがデータベースへの標準接続を使用する場合、クローンは標準接続を使用する必要があり、Directory Server データベースは、暗号化されていない接続を複製に使用 **できます**。
- マスターがデータベースへの標準接続を使用する場合、クローンは標準接続を使用する必要があります **が**、複製用のマスター/クローンの Directory Server データベースに Start TLS を使用するオプションがあります。TLS は、標準ポートでセキュアな接続を開きます。



注記

TLS を使用するには、SSL/TLS 接続を受け入れるように Directory Server を設定する必要があります。つまり、クローンを設定する前に、サーバー証明書と CA 証明書を Directory Server にインストールし、SSL/TLS を有効にする必要があります。

マスターが使用する接続メソッド (セキュアまたは標準) はクローンで使用し、クローンを設定する前に Directory Server データベースに対して適切に設定する必要があります。

重要

クローンが安全な接続を介してマスターに接続する場合でも、クローンの設定中は、標準の LDAP ポート (デフォルトでは 389) が開いていて、LDAP サーバーで有効になっている必要があります。

セキュアな環境では、クローンの設定後にマスターの Directory Server インスタンスで標準の LDAP ポートを無効にすることができます。

2.7.8. レプリカ ID 番号

クローン作成は、マスターインスタンスの Directory Server と、クローンインスタンスの Directory Server との間でレプリカ合意を設定することをベースとしています。

レプリケーションとともに関連するサーバーは、同じレプリケーション **トポロジー** にあります。サブシステムインスタンスのクローンが作成されるたびに、トポロジー全体に追加されます。Directory Server は、**レプリカ ID 番号** に基づいてトポロジー内の異なるサーバー間で区別されません。このレプリカ ID は、トポロジー内のすべてのサーバーで一意である必要があります。

要求および証明書に使用されるシリアル番号の範囲 (「CA のクローン作成」) と同様に、すべてのサブシステムには、許可されるレプリカ ID の範囲が割り当てられます。サブシステムのクローン時に、レプリカ ID の 1 つをその範囲から新しいクローンインスタンスに割り当てます。

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

インスタンスが現在の範囲を使い果たし始めると、レプリカ ID の範囲を新しい番号で更新できます。

2.7.9. カスタム設定およびクローン

クローンの作成後、クローン間、またはマスターとクローンの間で **設定** の変更は複製されません。インスタンスの設定は、複製されたデータベースの外部にある **CS.cfg** ファイルにあります。

たとえば、CA にはマスターとクローンの 2 つがあります。設定に関連付けられる新しい KRA が、マスター CA とともにインストールされている。CA-KRA コネクター情報はマスター CA の **CS.cfg** ファイルに保存されますが、このコネクター情報はクローン CA 設定に追加されません。キーアーカイブを含む証明書要求がマスター CA に送信される場合は、CA-KRA コネクター情報を使用してキーのアーカイブが KRA に転送されます。要求がクローン CA に送信される場合、KRA は認識されず、キーのアーカイブ要求は許可されません。

マスターサーバーまたはクローンサーバーの設定に加えた変更は、他のクローンインスタンスに複製されません。クローンを作成するには、重要な設定を手動で追加する必要があります。

注記

マスターサーバーの必要なカスタム設定をすべて設定してから、クローンを設定できます。たとえば、すべてのコネクター情報がマスター CA 設定ファイルにあるようにインストールしたり、カスタムプロファイルを作成したり、マスター OCSP レスポンダーのすべての公開ポイントを設定したりします。LDAP プロファイルが Directory Server に保存されている場合は、複製され、サーバー間で同期されることに注意してください。

マスターインスタンスのカスタム設定は、**クローン時に** クローンのインスタンスに追加されます (ただし、後では機能しません)。

第3章 サポートされる標準およびプロトコル

Red Hat Certificate System は、可能な限りパフォーマンスと相互運用性を確保できるように、多くのパブリックプロトコルおよび標準プロトコル RFC に基づいています。本章では、Certificate System 10 で使用または対応している主な標準およびプロトコルについて、管理者がクライアントサービスを効果的に計画できるように、本章で概説しています。

3.1. TLS、ECC、および RSA

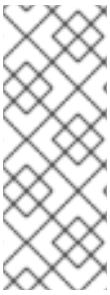
Transport Layer Security (TLS) プロトコルは、クライアントとサーバー間の認証と暗号化された通信の汎用的な標準です。クライアントとサーバーの認証は TLS 上で行われます。

TLS は、公開鍵と対称鍵の暗号化の組み合わせを使用します。対称キーの暗号化は公開鍵の暗号化よりもはるかに高速ですが、公開鍵の暗号化を使用すると認証の手法が向上します。TLS セッションは常に、*ハンドシェイク*と呼ばれるメッセージの交換で開始します。これはサーバーとクライアント間の初期通信です。ハンドシェイクにより、サーバーは公開鍵技術を使用してクライアントに対して自己認証を行い、必要に応じてクライアントがサーバーに対して認証できます。次に、クライアントとサーバーは、以下に示すセッション中に迅速な暗号化、復号、および整合性の検証に使用される対称鍵の作成において連携できるようになります。

TLS は、サーバーやクライアントの認証、証明書の送信、セッション鍵の確立などのさまざまな暗号化アルゴリズム (*暗号*) をサポートします。クライアントとサーバーは、異なる暗号スイートまたは暗号のセットをサポートする場合があります。その他の機能に加えて、ハンドシェイクは、サーバーおよびクライアントが相互に認証に使用される暗号スイートをどのようにネゴシエートし、証明書を送信し、セッションキーを確立する方法を決定します。

RSA や EllipticCurve Diffie-Hellman (ECDH) などの鍵交換アルゴリズムは、サーバーとクライアントが TLS セッション中に使用する対称鍵を決定する方法を管理します。TLS は ECC (Elliptic Curve Cryptography) 暗号化スイートおよび RSA に対応します。Certificate System は、RSA と ECC の両方の公開鍵暗号システムをネイティブにサポートします。

より最近の実施では、鍵交換アルゴリズムは、安全な通信のための共通の鍵を確立するときに、2 人以上の当事者のそれぞれが結果に影響を与える可能性がある *鍵共有プロトコル*に取って代わられています。キー合意は、PFS (Perfect Forward Secrecy) の実装を許可するため、鍵交換が推奨されます。PFS を使用する場合、鍵共有の目的で、非決定論的アルゴリズムによってセッションごとにランダムな公開鍵 (一時暗号パラメーターまたは *一時鍵* とも呼ばれます) が生成されます。その結果、複数のメッセージの不正使用につながる秘密の値がなく、過去や今後の通信量は保護されません。



注記

コンピューティング機能が向上するのに合わせてセキュリティーを提供するには、より長い RSA キーが必要です。推奨される RSA 鍵の長さは 2048 ビットです。多くのサーバーは 1024 ビット鍵を使用しますが、サーバーは少なくとも 2048 ビット鍵に移行する必要があります。64 ビットマシンの場合は、より強力なキーの使用を検討してください。すべての CA は、可能であれば 2048 ビット鍵と、より強力な鍵 (3072、4096 ビットなど) を使用する必要があります。

3.1.1. サポート対象の暗号スイート

暗号化およびハッシュアルゴリズムは、さまざまな脆弱性とセキュリティー強度に関して絶えず変化しています。原則として、Red Hat Certificate System は NIST ガイドライン に従い、サーバーキーに関連する TLS 1.1 および TLS 1.2 暗号スイートをサポートします。

3.1.1.1. 推奨される TLS 暗号スイート

Transport Layer Security (TLS) プロトコルは、クライアントとサーバー間の認証と暗号化された通信の汎用的な標準です。Red Hat Certificate System は TLS 1.1 および 1.2 をサポートします。

サーバーがサーバーまたはクライアントとして機能している場合、Red Hat Certificate System は以下の暗号スイートをサポートします。

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. 許可されるキーアルゴリズムとそのサイズ

Red Hat Certificate System は、基礎となる PKCS #11 モジュールにより提供されている場合は、以下の鍵アルゴリズムとサイズに対応します。

- 許可される RSA 鍵サイズ:
  - 2048 ビット以上
- FIPS PUB 186-4 標準仕様に定義されている EC 曲線または同等です。

- nistp256
- nistp384
- nistp521

3.3. 許可されるハッシュ関数

以下の主要なハッシュメッセージ認証 (HMAC) を使用できます。

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

以下の暗号ハッシュ関数が許可されます。

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

3.4. IPV4 アドレスおよび IPV6 アドレス

Certificate System は、IPv4 アドレスと IPv6 アドレスの両方をサポートします。非常に多様な状況では、Certificate System サブシステムまたは操作がホスト名または IP アドレスを参照します。IPv4 と IPv6 形式のアドレスの両方をサポートすることで、ネットワークプロトコルとの互換性が転送されます。IPv6 接続をサポートする操作には、以下が含まれます。

- TPS、TKS、CA 間を含むサブシステム間の通信、およびセキュリティードメインへの参加
- TPS と Enterprise Security Client 間のトークン操作
- サブシステムロギング
- アクセス制御の手順
- **pki** ユーティリティー、Subject Alt Name Extension ツール、HttpClient、Bulk Issuance Tool を含む Certificate System ツールで実行される操作
- クライアント通信 (**pkiconsole** ユーティリティーおよび Web サービス用の IPv6 対応ブラウザーの両方を含む)
- 証明書要求名と証明書サブジェクト名 (ユーザー、サーバー、ルーターの証明書など)
- 公開
- 内部データベースおよび認証ディレクトリーでの LDAP データベースへの接続

ホスト名または URL が参照されるたびに、IP アドレスを使用することができます。

- IPv4 アドレスは、**n.n.n.n** または **n.n.n.n,m.m.m.m** の形式にする必要があります。たとえば、**128.21.39.40** または **128.21.39.40,255.255.255.00** です。

- IPv6 アドレスは 128 ビット名前空間を使用します。IPv6 アドレスはコロンで区切られ、ネットマスクはピリオドで区切られます。たとえば、**0:0:0:0:0:0:13.1.68.3**、**FF01::43**、または **0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0** です。

DNS が適切に設定されている場合、IPv4 アドレスまたは IPv6 アドレスを使用して Web サービスページおよびサブシステム Java コンソールに接続できます。最も一般的な方法は、完全修飾ドメイン名を使用することです。

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

IPv6 数値アドレスを使用するには、URL の完全修飾ドメイン名を角かっこ (**[]**) で囲まれた IPv6 アドレスに置き換えます。以下に例を示します。

```
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. サポートされる PKIX 形式およびプロトコル

Certificate System は、IETF により Public-Key Infrastructure (X.509) で定義された多くのプロトコルとフォーマットをサポートします。以下に示す PKIX 標準に加えて、その他の PKIX リスト化された標準は IETF Datatracker の Web サイトから入手できます。

表3.1 証明書システム 10 でサポートされている PKIX 標準

形式またはプロトコル	RFC またはドラフト	説明
X.509 バージョン 1 およびバージョン 3		国際電気通信連合 (ITU) が推奨するデジタル証明書形式。
Certificate Request Message Format (CRMF)	RFC 4211	CA に証明書要求を送信するためのメッセージ形式。
証明書管理メッセージ形式 (CMMF)		メッセージ形式。エンドエンティティーから CA に証明書要求および失効リクエストを送信し、エンドエンティティーに情報を返します。CMMF は、別の標準である CMC に含まれています。
CS を介した証明書管理メッセージ (CMC)	RFC 5274	CS および PKCS #10 に基づく公開鍵認定製品への一般的なインターフェイス。これには、Diffie-Hellman 公開鍵で RSA 署名の証明書用の証明書登録プロトコルが含まれます。CMC には CRMF と CMMF が組み込まれています。
暗号化メッセージ構文 (CMS)	RFC 2630	デジタル署名と暗号化に使用される PKCS #7 構文のスーパーセット。

形式またはプロトコル	RFC またはドラフト	説明
PKIX 証明書および CRL プロファイル	RFC 5280	インターネット用の公開鍵インフラストラクチャー向けに IETF により開発された標準規格です。証明書および CRL のプロファイルを指定します。
オンライン証明書ステータスプロトコル (OCSP)	RFC 6960	CRL を使用せずにデジタル署名の現在のステータスを決定するプロトコルは便利です。

第4章 サポート対象のプラットフォーム

本セクションでは、Red Hat Certificate System 10 でサポートされているさまざまなサーバープラットフォーム、ハードウェア、トークン、およびソフトウェアを説明します。

4.1. 一般的な要件

詳細は、Red Hat Certificate System 10 リリースノート で該当するセクションを参照してください。

4.2. サーバーサポート

Certificate System 10.4 の認証局 (CA)、Key Recovery Authority (KRA)、オンライン証明書ステータスプロトコル (OCSP)、トークンキーサービス (TKS)、およびトークン処理システム (TPS) サブシステムの実行は、Red Hat Enterprise Linux 8.6 以降でサポートされています。サポートされる Directory Server バージョンは 11.1 以降です。

注記

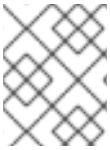
Certificate System 10.4 は、認定済みのハイパーバイザーの Red Hat Enterprise Linux 8.6 仮想ゲストでの実行に対応します。詳細は、ナレッジベースの記事 Red Hat Enterprise Linux の実行が認定されているハイパーバイザー を参照してください。

4.3. サポートされる WEB ブラウザー

Certificate System 10.0 は、以下のブラウザーに対応しています。

表4.1 プラットフォームでサポートされる Web ブラウザー

プラットフォーム	エージェントサービス	エンドユーザーページ
Red Hat Enterprise Linux	Firefox 60 以降 [a]	Firefox 60 以降 [a]
[a] この Firefox バージョンは、ブラウザーからキーの生成およびアーカイブに使用される **暗号化** Web オブジェクトに対応しなくなりました。そのため、この分野では機能が限定されるはずです。		



注記

HTML ベースのインスタンス設定に完全に対応するブラウザーは Mozilla Firefox のみです。

4.4. サポート対象のハードウェアセキュリティーモジュール

以下の表は、Red Hat Certificate System がサポートする Hardware Security Modules (HSM) を示しています。

HSM	ファームウェア	アプライアンスソフトウェア	クライアントソフトウェア
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 CERTIFICATE SYSTEM の計画

各 Red Hat Certificate System サブシステムは同じサーバーマシンにインストールするか、別のサーバーにインストールするか、組織全体で複数のインスタンスをインストールすることができます。サブシステムをインストールする前に、デプロイメントを計画することが重要です。どのような種類の PKI サービスが必要ですか。ネットワーク要件Certificate System にアクセスするために必要な人、そのロール、および物理的な場所は何ですか。発行する証明書の種類と、それらの制約またはルールを設定する必要があるか。

本章では、Certificate System のデプロイメントプランニングに関する基本的な質問を説明します。これらのデシジョンの多くは相互関連です。たとえば、スマートカードを使用して TPS サブシステムおよび TKS サブシステムをインストールするかどうかを決定するかどうかを決定するなど、別の選択肢に影響します。

5.1. 必要なサブシステムの決定

Certificate System サブシステムは、証明書管理のさまざまな側面に対応しています。インストールするサブシステムのプランニングは、デプロイメントが必要な PKI 操作を定義する方法の 1 つです。

ソフトウェアや設備などの証明書は、定義されたステージでライフサイクルを持ちます。最も基本的な手順は以下のとおりです。

- 要求および発行されます。
- これは有効です。
- 期限切れになります。

ただし、このシンプルなシナリオでは、証明書に関する多くの共通問題について説明しません。

- 証明書の有効期限が切れる前に従業員が退職するかどうか
- CA 署名証明書の有効期限が切れると、その証明書を使用して発行および署名されたすべての証明書も有効期限が切れます。これにより、CA 署名証明書が更新され、発行された証明書が有効に保つか、または再発行されますか？
- 従業員がスマートカードを失ったか、またはスマートカードに残るかどうか。元の証明書キーを使用して代替証明書が発行されますか。他の証明書は一時停止または取り消されるか。一時的な証明書は許可されますか。
- 証明書の有効期限が切れると、新しい証明書を発行するか、元の証明書が更新されますか？

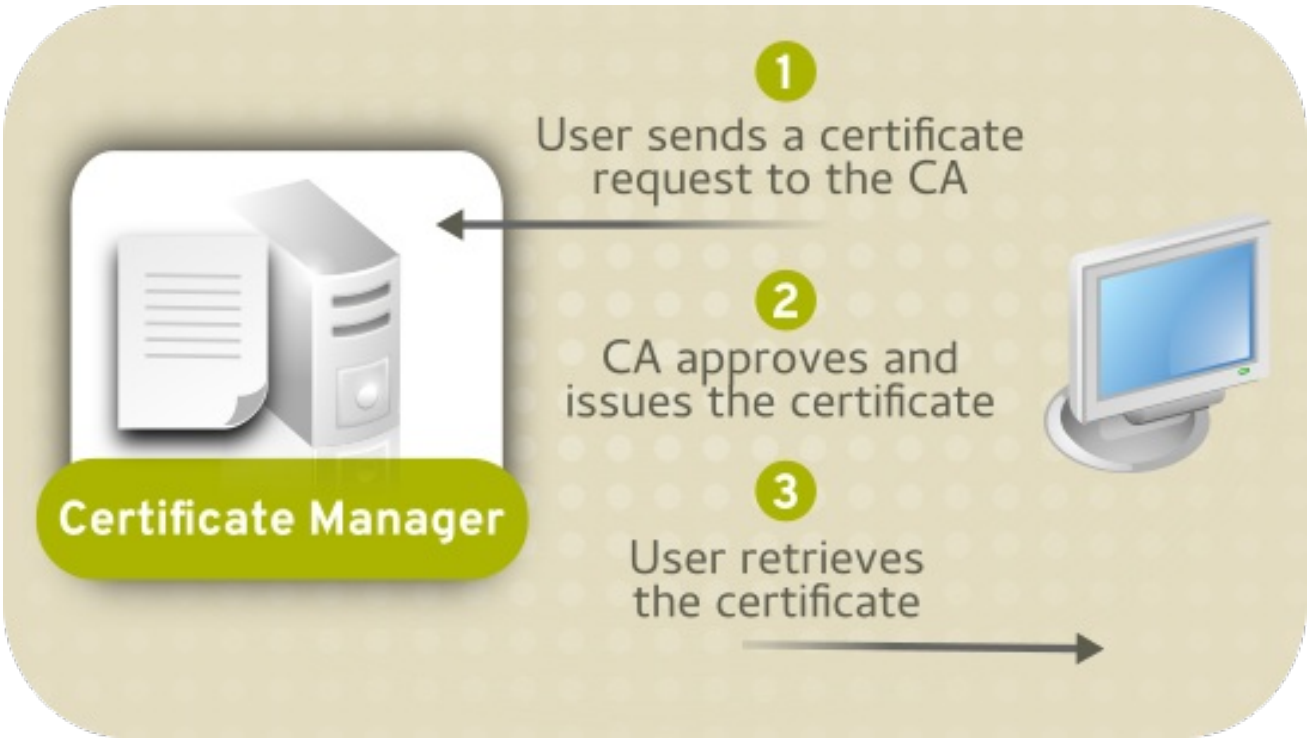
これにより、証明書の管理に関する他の 3 つの考慮事項 (失効、更新、および代替) を紹介します。

他の考慮事項は、認証局への負荷です。発行や更新のリクエストはたくさんありますか。証明書が有効かどうかの検証を試みるクライアントから大量のトラフィックがありますか。ID を認証するための証明書を要求する人はどのようになっていますか。また、そのプロセスによって発行プロセスが遅くなりますか。

5.1.1. 単一証明書マネージャーの使用

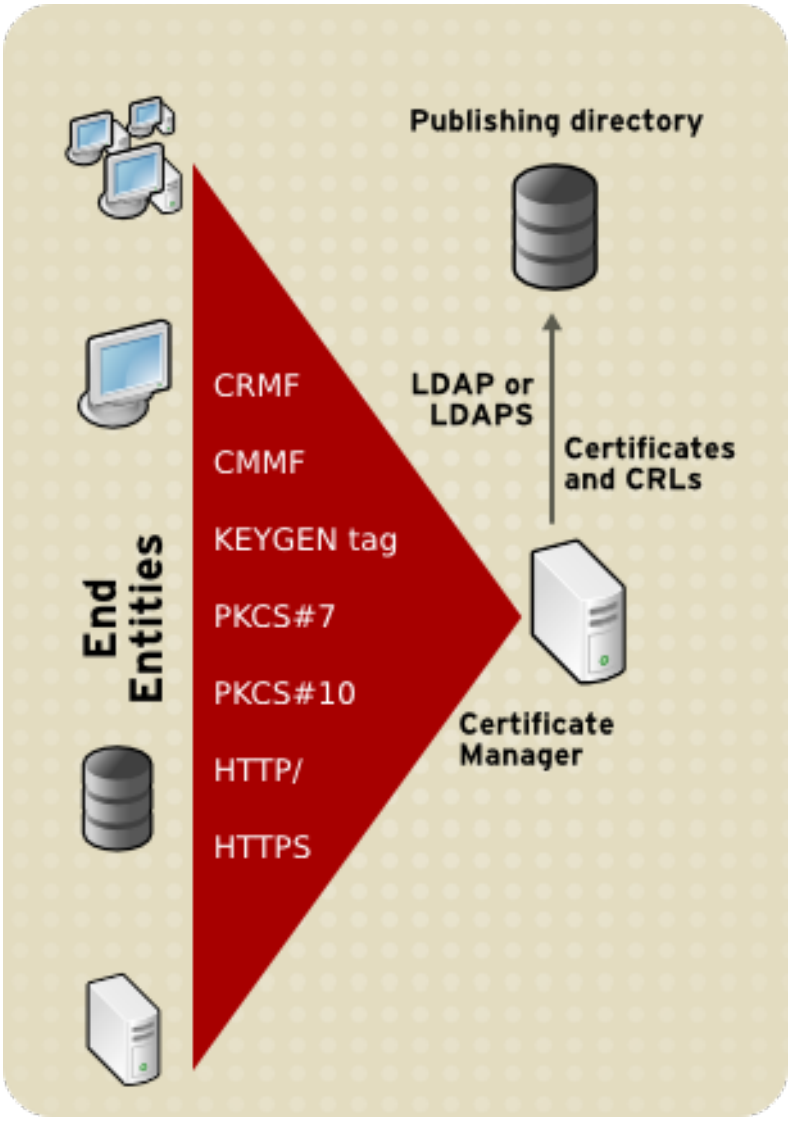
Certificate System PKI の中核は、Certificate Manager (認証局) です。CA は証明書要求を受け取り、すべての証明書を発行します。

図5.1 CA のみの Certificate System



要求および発行のための基本処理はすべて、Certificate Manager が処理でき、これは唯一の必須サブシステムです。組織の要求に応じて、単一または多数の Certificate Manager をさまざまな関係で配置することができます。

証明書の発行に加えて、Certificate Manager は証明書を取り消すこともできます。管理者にとっての 1 つの質問は、証明書が紛失、侵害された場合、または証明書が発行された人や機器が会社をいなくなった場合に、証明書をどのように処理するかです。証明書要求を失効すると、失効日前に証明書を無効化し、失効した証明書の一覧がコンパイルされ、発行 CA によって公開され、クライアントが証明書のステータスを検証できるようになります。



5.1.2. 紛失したキーの計画: キーのアーカイブと回復

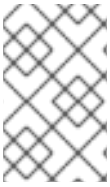
CA が実行できない 1 つの操作はキーのアーカイブとリカバリーです。非常に現実的なシナリオは、ユーザーが秘密鍵を紛失することです。たとえば、鍵がブラウザーデータベースから削除されたり、スマートカードが家に残されたりする可能性があります。多くの一般的なビジネスオペレーションでは、暗号化された電子メールなどの暗号化されたデータが使用され、そのデータのロックを解除するキーを失うと、データ自体が失われます。会社のポリシーによっては、交換用の証明書を再生成または再インポートするためにキーをリカバリーする方法が必要になる可能性があり、どちらの操作にも秘密キーが必要です。

これには、キーを特別にアーカイブおよび取得するサブシステムである KRA が必要です。

図5.2 CA および KRA

キーリカバリー機関は暗号鍵 (キーアーカイブ) を保存し、CA が証明書 (キーのリカバリー) を再発行できるようにこれらのキーを取得できます。KRA は、通常の証明書に対して実行される場合でも、スマートカードを登録する場合でも、Certificate Systemによって生成された証明書のキーを格納できます。

キーのアーカイブおよびリカバリーのプロセスは、「キーのアーカイブ、リカバリー、およびローテーション」 で詳細に説明されています。



注記

KRA は、秘密鍵のアーカイブおよび復元を目的としています。したがって、エンドユーザーは、公開鍵と秘密鍵のペアを格納するために、デュアルキー生成をサポートするブラウザーを使用する必要があります。

5.1.3. 証明書要求の処理の分散

サブシステムを連携させる方法のもう 1 つが負荷分散です。サイトのトラフィックが多い場合は、相互のクローンとして、またはフラット階層 (各 CA が独立している場合) またはツリー階層 (一部の CA が他の CA に従属している場合) として、多数の CA を簡単にインストールできます。詳細は、「認証局階層の定義」 を参照してください。

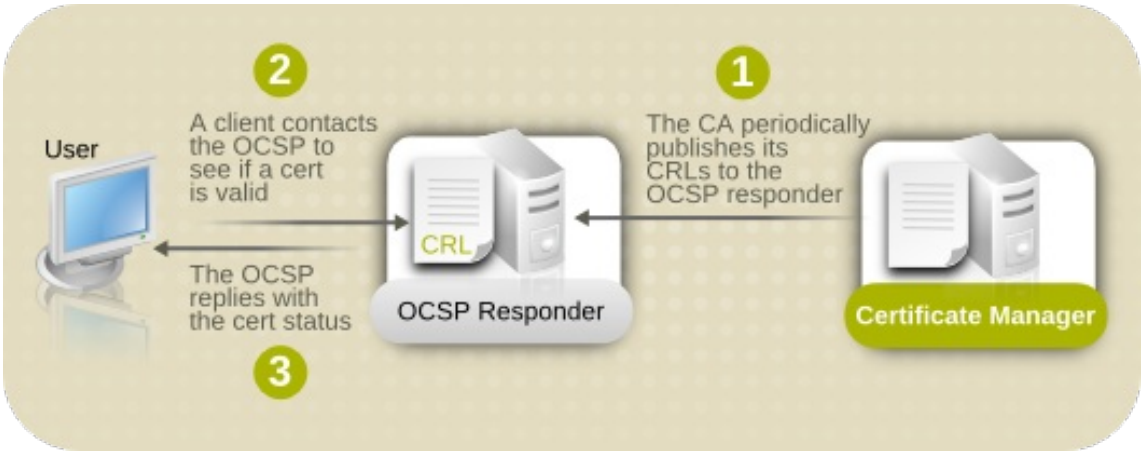
5.1.4. クライアント OCSP 要求の分散

証明書が有効期間内にありますが無効にする必要がある場合は、取り消すことができます。Certificate Manager は、取り消された証明書の一覧を公開することができます。これにより、クライアントが証明書が有効であることを確認する必要がある場合は、一覧を確認できます。これらの要求は、**オンライン証明書ステータスプロトコル要求** で、特定の要求と応答の形式を持ちます。Certificate Manager には、OCSP レスポンダーが組み込まれており、OCSP 要求を自分で検証できます。

ただし、証明書要求トラフィックと同様に、サイトには、証明書のステータスを確認するためのクライアントの要求が多数ある可能性があります。Example Corp. には大規模な Web ストアがあり、各顧客のブラウザーは SSL/TLS 証明書の有効性を検証しようとします。ここでも、CA は証明書の発行数を処

理することができますが、要求トラフィックが高いとパフォーマンスに影響します。この場合、Example Corp. は外部の OCSP Manager サブシステムを使用して証明書のステータスを確認し、Certificate Manager は更新された CRL を頻繁に公開するだけで済みます。

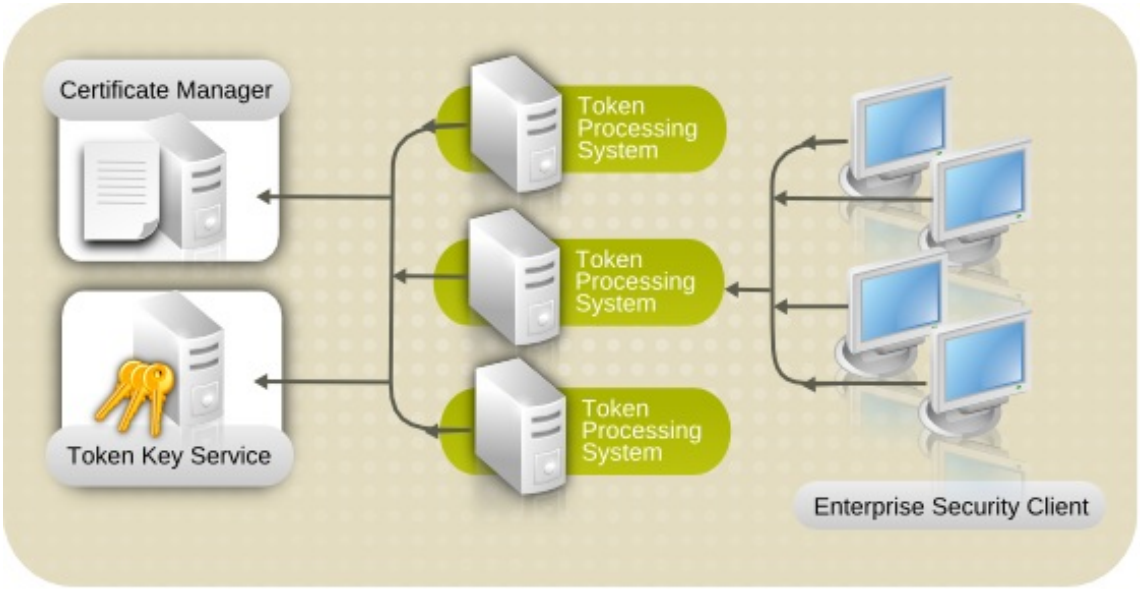
図5.3 CA および OCSP



5.1.5. スマートカードの使用

スマートカードは、キーと証明書のデータを格納するために物理的な媒体を使用するため、通常、直接の登録と承認のプロセスが必要です。つまり、複数のエージェントが TPS にアクセスでき、多くの場合は、異なるオフィスまたは地理的な場所にある複数の TPS サブシステムが必要になります。

トークン管理システムは非常にスケーラブルです。複数の TPS は、単一の CA、TKS、または KRA インスタンスと連携するように設定できますが、複数の Enterprise Security Clients は 1 つの TPS と通信できます。追加のクライアントがインストールされると、TPS を再設定せずに TPS インスタンスを参照できます。同様に、TPS が追加されるため、これらのサブシステムを再設定せずに、同じ CA、TKS、および KRA インスタンスを参照することができます。



インストール後に、TPS 設定を編集して、フェイルオーバーのサポートに追加の CA インスタンス、KRA インスタンス、および TKS インスタンスを使用できます。そのため、プライマリーサブシステムが利用できない場合、TPS はトークンサービスを中断せずに次に利用可能なシステムに切り替えます。

5.2. 認証局階層の定義

CA は PKI の中心となるため、CA システムの相互関係 (CA 階層) や他のサブシステム (セキュリティードメイン) の両方は、Certificate System PKI の計画に不可欠です。

PKI に複数の CA がある場合、CA は階層またはチェーンに構築されます。チェーン内の別の CA は、**ルート CA** と呼ばれます。チェーン内の別の CA の背後にある CA は **下位の CA** と呼ばれます。CA は、Certificate System デプロイメント以外のルートに従属させることもできます。たとえば、Certificate System デプロイメント内でルート CA として動作する CA は、サードパーティーの CA に従属できます。

証明書マネージャー (または CA) は、CA 署名証明書 (証明書の発行を許可する証明書) が別の CA によって発行されるため、別の CA に従属します。下位 CA 署名証明書を発行した CA は、CA 署名証明書の内容を使用して CA を制御します。CA は、発行できる証明書の種類、証明書に含めることができる拡張機能、従属 CA が作成できる従属 CA のレベル数、ならびに発行できる証明書の有効期間および下位 CA 署名証明書の有効期間を通じて、従属 CA を制約できます。

注記

下位 CA はこれらの制約に違反する証明書を作成できますが、これらの制約に違反する証明書を認証するクライアントはその証明書を受け入れません。

自己署名ルート CA は、独自の CA 署名証明書に署名し、独自の制約を設定するとともに、発行する下位 CA 署名証明書に制約を設定します。

Certificate Manager は、ルート CA または下位 CA として設定できます。最初の CA が自己署名ルートをインストールし、サードパーティーに適用して証明書を発行するのを待つのが最も簡単な方法です。ただし、完全な PKI をデプロイする前に、ルート CA を用意するかどうか、いくつ持つか、ルート CA と従属 CA の両方を置く場所を検討してください。

5.2.1. パブリック CA への従属

Certificate System CA をサードパーティー公開 CA にチェーンすると、公開 CA が、下位 CA が発行できる証明書の種類と証明書チェーンの性質に課す制限が導入されます。たとえば、サードパーティー CA にチェーンする CA は、SSL/TLS サーバー証明書ではなく、S/MIME (Secure Multipurpose Internet Mail Extensions) および SSL/TLS クライアント認証証明書のみの発行に制限される可能性があります。パブリック CA を使用する方法は他にもあります。これは一部の PKI デプロイメントに許可されないことがあります。

パブリック CA にチェーンする利点の 1 つは、サードパーティーがルート CA 証明書を Web ブラウザーまたは他のクライアントソフトウェアに送信することです。これは、管理者が制御できないブラウザーを使用してさまざまな企業がアクセスする証明書を持つエクストラネットにとって大きな利点になる可能性があります。CA 階層にルート CA を作成するということは、ローカル組織が、Certificate System によって発行された証明書を使用するすべてのブラウザーにルート証明書を取得する必要があることを意味します。イントラネット内でこれを行うためのツールはありますが、エクストラネットでは実現が難しい場合があります。

5.2.2. Certificate System CA への従属

Certificate System CA は **ルート CA** として機能するため、サーバーは独自の CA 署名証明書、およびその他の CA 署名証明書に署名し、組織固有の CA 階層を作成します。このサーバーは、**下位 CA** として設定することもできます。つまり、サーバーの CA 署名鍵が既存の CA 階層にある別の CA により署名されます。

Certificate System CA をルート CA として設定すると、発行した CA 署名証明書の内容を制御するポリシーを設定し、Certificate System 管理者がすべての下位 CA を制御することができます。下位 CA は、ルート CA の設定とは対照的に、独自の認証および証明書プロファイル設定を評価することで証明

書を発行します。

5.2.3. リンクされた CA

Certificate System の Certificate Manager は、**リンクされた CA** として機能し、検証用のサードパーティーまたはパブリック CA までチェーンできます。これにより、会社の証明書階層外で証明書チェーンを検証できます。Certificate Manager は、サードパーティーの CA から Certificate Manager の **CA 署名証明書** を要求して、サードパーティーの CA にチェーンされます。

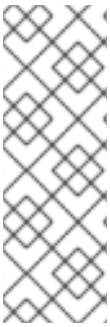
これに関連し、Certificate Manager は、**クロスペア** または **クロス署名の証明書** を発行することもできます。これら 2 つの CA 間でクロス署名証明書を発行および格納することにより、2 つの異なる CA 間で信頼される関係を作成できます。クロス署名証明書ペアを使用すると、組織の PKI 以外で発行された証明書は、システム内で信頼できます。

これらは、連邦ブリッジ認証局 (FBCA) の定義に関連して、**ブリッジ証明書** とも呼ばれます。

5.2.4. CA クローン

ルート CA と従属 CA の階層を作成する代わりに、Certificate Manager の複数のクローンを作成し、シリアル番号の範囲内で証明書を発行するように各クローンを設定することができます。

クローン として作成された Certificate Manager は、同じ CA 署名鍵と、別の Certificate Manager (**マスター** Certificate Manager) を使用します。



注記

サブシステムが複製される可能性がある場合は、設定プロセス中にそのキーペアをエクスポートして、安全な場所に保存するのが最も簡単です。クローンが元の Certificate Manager のキーから証明書を生成できるように、クローンインスタンスの設定時に、元の Certificate Manager のキーペアが使用可能である必要があります。

pk12util または **PKCS12Export** コマンドを使用して、後でセキュリティーデータベースから鍵をエクスポートすることもできます。

クローン CA と元の CA は、同じ CA 署名鍵と証明書を使用して、発行する証明書に署名するため、すべての証明書の **発行者名** は同じです。クローン CA と元の Certificate Manager は、単一の CA であるかのように証明書を発行します。高可用性フェイルオーバーのサポートのために、これらのサーバーは異なるホストに置くことができます。

クローン作成の利点は、Certificate Manager の負荷を複数のプロセスまたは複数の物理マシンに分散することです。登録需要の高い CA の場合は、クローン作成から得られる配布により、指定された時間間隔でより多くの証明書に署名して発行できます。

クローン作成された Certificate Manager には、エージェントやエンドエンティティーゲートウェイ機能など、通常の Certificate Manager と同じ機能があります。

クローンが発行する証明書のシリアル番号は、動的に分散されます。各クローンとマスターのデータベースが複製されるため、すべての証明書要求と発行された証明書の両方も複製されます。これにより、シリアル番号の競合が発生せず、クローンされた Certificate Manager にシリアル番号の範囲を手動で割り当てる必要がなくなります。

5.3. セキュリティードメインの計画

セキュリティードメイン は PKI サービスのレジストリーです。CA などの PKI サービスは、これらのドメインに独自の情報を登録するため、PKI サービスのユーザーはレジストリーを確認して他のサービス

を検索できます。Certificate System のセキュリティードメインサービスは、Certificate System サブシステムの PKI サービスの登録と、共有信頼ポリシーのセットの両方を管理します。

レジストリーは、ドメイン内でサブシステムによって提供されるすべての PKI サービスの完全なビューを提供します。各 Certificate System サブシステムは、ホストまたはセキュリティードメインのメンバーのいずれかである必要があります。

CA サブシステムは、セキュリティードメインをホストできる唯一のサブシステムです。セキュリティードメインは、特権ユーザーおよびグループ情報の CA 内部データベースを共有して、セキュリティードメインを更新し、新しい PKI サービスを登録し、証明書を発行できるユーザーを決定します。

セキュリティードメインは CA の設定中に作成され、セキュリティードメインの CA の LDAP ディレクトリーにエントリーが自動的に作成されます。各エントリーには、ドメインに関する重要な情報がすべて含まれます。セキュリティードメインを登録する CA を含む、ドメイン内のすべてのサブシステムは、セキュリティードメインコンテナーエントリーの下に記録されます。

CA の URL はセキュリティードメインを一意に識別する。セキュリティードメインには、**Example Corp Intranet PKI** などの分かりやすい名前も付与されます。他のすべてのサブシステム (KRA、TPS、TKS、OCSP、およびその他の CA) は、サブシステムの設定時にセキュリティードメイン URL を指定して、セキュリティードメインのメンバーになる必要があります。

セキュリティードメイン内の各サブシステムは、異なるサーバーやブラウザーから取得できる同じ信頼ポリシーと信頼されたルートを共有します。セキュリティードメインで利用可能な情報は、新しいサブシステムの設定中に使用されます。これにより、設定プロセスが合理化および自動化されます。たとえば、TPS が CA に接続する必要がある場合、TPS はセキュリティードメインを参照して、使用可能な CA のリストを取得できます。

各 CA には独自の LDAP エントリーがあります。セキュリティードメインは、CA エントリーの下にある組織グループです。

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

次に、セキュリティードメイン組織グループの下に各サブシステムタイプのリストがあり、グループタイプを識別するための特別なオブジェクトクラス (**pkiSecurityGroup**) があります。

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

各サブシステムインスタンスは、エントリータイプを識別するための特別な **pkiSubsystem** オブジェクトクラスを使用してそのグループのメンバーとして保存されます。

```
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
```

サブシステムが操作を実行するために別のサブシステムに接続する必要がある場合は、(CA の管理ポートを介して接続するサーブレットを呼び出すことにより) サブシステムがセキュリティードメインをホストする CA に接続します。次に、セキュリティードメイン CA は、LDAP データベースからサブシステムに関する情報を取得し、その情報を要求元のサブシステムに返します。

サブシステムはサブシステム証明書を使用してセキュリティードメインに対して認証します。

セキュリティードメインを計画するときに以下を考慮してください。

- セキュリティードメインをホストする CA は外部認証局によって署名できます。
- 複数のセキュリティードメインを組織内に設定することができます。ただし、各サブシステムは 1 つのセキュリティードメインにのみ属できます。
- ドメイン内のサブシステムをクローンできます。サブシステムインスタンスは、システム負荷を分散し、フェイルオーバーポイントを提供します。
- セキュリティードメインは、CA と KRA との間の設定を合理化します。KRA は、管理者が証明書を手動で CA にコピーする必要がなく、KRA コネクター情報をプッシュして証明書を CA に自動的に転送できます。
- Certificate System セキュリティードメインを使用すると、オフラインの CA を設定できます。このシナリオでは、オフラインルートには独自のセキュリティードメインがあります。オンラインの下位 CA はすべて、異なるセキュリティードメインに属します。
- セキュリティードメインは、CA と OCSP の間の設定を簡素化します。OCSP は、CA が OCSP 公開を設定するために、その情報を CA にプッシュし、CA から CA 証明書チェーンを取得して、内部データベースに格納することもできます。

5.4. サブシステム証明書の要件の決定

CA 設定は、発行される証明書の実際のタイプに関係なく、発行する証明書の特性の多くを決定します。CA 自体の有効期間、識別名、および許可される暗号化アルゴリズムに対する制約は、発行された証明書の同じ特性に影響を与えます。さらに、Certificate Manager には、発行するさまざまな種類の証明書のルールを設定する事前定義されたプロファイルがあり、追加のプロファイルを追加または変更できます。これらのプロファイル設定は、発行した証明書にも影響します。

5.4.1. インストールする証明書の決定

Certificate System サブシステムが最初にインストールおよび設定されると、それにアクセスして管理するために必要な証明書が自動的に作成されます。これには、エージェントの証明書、サーバー証明書、およびサブシステム固有の証明書が含まれます。これらの初期証明書は 表5.1「初期サブシステム証明書」 に表示されます。

表5.1 初期サブシステム証明書

サブシステム	証明書

サブシステム	証明書
Certificate Manager	• CA 署名証明書 • OCSP 署名証明書 • SSL/TLS サーバー証明書の場合 • サブシステム証明書 • ユーザー (エージェント/管理者) 証明書 • 監査ログ署名証明書
OCSP	• OCSP 署名証明書 • SSL/TLS サーバー証明書の場合 • サブシステム証明書 • ユーザー (エージェント/管理者) 証明書 • 監査ログ署名証明書
KRA	• トランスポート証明書 • ストレージ証明書 • SSL/TLS サーバー証明書の場合 • サブシステム証明書 • ユーザー (エージェント/管理者) 証明書 • 監査ログ署名証明書
TKS	• SSL/TLS サーバー証明書の場合 • ユーザー (エージェント/管理者) 証明書 • 監査ログ署名証明書
TPS	• SSL/TLS サーバー証明書の場合 • ユーザー (エージェント/管理者) 証明書 • 監査ログ署名証明書

既存のサブシステム証明書を置き換える際の注意点がいくつかあります。

- ルート CA の新しい自己署名 CA 証明書を作成するときに新しいキーペアを生成すると、以前の CA 証明書で発行されたすべての証明書が無効になります。

  これは、古いキーを使用して CA によって発行または署名された証明書が機能しないことを意味します。下位 Certificate Manager、KRA、OCSP、TKS、および TPS は機能しなくなり、エージェントはエージェントインターフェイスにアクセスできなくなります。

  これと同じ状況は、下位 CA の CA 証明書が新しいキーペアを持つものに置き換えられた場合に発生します。その CA によって発行されたすべての証明書は無効になり、機能しなくなります。

  新しいキーペアから新しい証明書を作成する代わりに、既存の CA 署名証明書を更新することを検討してください。

- CA が OCSP に公開するように設定されていて、新しい CA 署名証明書または新しい CRL 署名証明書がある場合は、CA を OCSP に対して再識別する必要があります。

- KRA 用に新しいトランスポート証明書が作成された場合は、CA の設定ファイル **CS.cfg** で KRA 情報を更新する必要があります。既存のトランスポート証明書は、**ca.connector.KRA.transportCert** パラメーターの新しい証明書に置き換える必要があります。

- CA のクローンが作成されている場合は、マスター Certificate Manager の新しい SSL/TLS サーバー証明書を作成するときに、クローン CA の証明書データベースが新しい SSL/TLS サーバーの全証明書で更新する必要があります。

- Certificate Manager が証明書と CRL を LDAP ディレクトリーに公開するように設定されており、SSL/TLS クライアント認証に SSL/TLS サーバー証明書を使用する場合は、適切な拡張子を付けて新しい SSL/TLS サーバー証明書を要求する必要があります。証明書をインストールしたら、公開ディレクトリーが新しいサーバー証明書を使用するように設定する必要があります。

- サブシステムインスタンスに対して任意の数の SSL/TLS サーバー証明書を発行できますが、実際には 1 つの SSL/TLS 証明書のみが必要です。この証明書は、必要に応じて何度でも更新または交換できます。

5.4.2. CA 識別名の計画

CA のコア要素は署名ユニットと Certificate Manager ID です。署名ユニットは、終了エンティティーが要求した証明書に署名します。Certificate Manager には、発行するすべての証明書に記載されている、独自の識別名 (DN) が必要です。

他の証明書と同様に、CA 証明書は DN を公開鍵にバインドします。DN は、エンティティーを一意に識別する一連の名前と値のペアです。たとえば、次の DN は、Example Corporation という名前の企業のエンジニアリング部門の Certificate Manager を識別します。

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

Certificate Manager の DN には、名前と値のペアの組み合わせが多数あります。エンドエンティティーが検証できるため、DN は一意で識別する必要があります。

5.4.3. CA 署名証明書の有効期間の設定

Certificate Manager 署名証明書を含むすべての証明書には有効期間が必要です。Certificate System は、指定可能な有効期間を制限しません。証明書の更新の要件、証明書階層内の CA の位置、および PKI に含まれる公開 CA の要件に応じて、有効期間をできるだけ長く設定します。

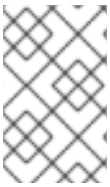
Certificate Manager は、CA 署名証明書の有効期間よりも有効期間が長い証明書を発行することはできません。CA 証明書の有効期間よりも長い期間要求が行われた場合、要求された有効日は無視され、CA 署名証明書の有効期間が使用されます。

5.4.4. 署名キーの種類と長さの選択

署名鍵は、サブシステムが何かを検証して封印するために使用されます。CA は、CA 署名証明書を使用して、発行する証明書または CRL に署名します。OCSP は、署名証明書を使用して、証明書ステータス要求への応答を検証します。すべてのサブシステムは、ログファイル署名証明書を使用して監査ログに署名します。

署名キーは、署名操作の保護とセキュリティーを提供するために、暗号的に強力である必要があります。以下の署名アルゴリズムがセキュアであるとみなされます。

- SHA256withRSA
- SHA512withRSA
- SHA256withEC
- SHA512withEC



注記

Certificate System には、ネイティブの ECC サポートが含まれています。ECC 対応サードパーティーの PKCS #11 モジュールを読み込み、使用することも可能です。詳細は9 章*ECC システム証明書を使用するインスタンスのインストール*を参照してください。

キー **タイプ** とともに、各キーには特定の **ビット長** があります。キーは、より短い鍵よりも暗号で強力なと見なされます。ただし、キーの署名操作にはより長い時間がかかります。

設定ウィザードのデフォルトの RSA キーの長さは 2048 ビットです。機密性の高いデータまたはサービスへのアクセスを提供する証明書の場合は、長さを 4096 ビットに増やすことを検討してください。ECC キーは RSA キーよりもはるかに強力であるため、ECC キーの推奨長は 256 ビットであり、これは 2048 ビットの RSA キーと同等の強度です。

5.4.5. 証明書の拡張の使用

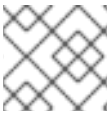
X.509 v3 証明書には、証明書に任意の数の追加フィールドを追加できる拡張フィールドが含まれています。証明書拡張機能は、代替サブジェクト名や使用制限などの情報を証明書に追加する方法を提供します。Red Hat Directory Server や Red Hat Certificate System などの古い Netscape サーバーは、PKIX パート 1 標準が定義される前に開発されたため、Netscape 固有の拡張機能が必要です。

X.509 v1 証明書の仕様は、公開鍵を X.500 ディレクトリーの名前にバインドするように設計されました。証明書がインターネットおよびエクストラネットで使用されるようになり、ディレクトリールックアップを常に実行できるとは限らなかったため、元の仕様ではカバーされていなかった問題領域が発生しました。

- **信用**。X.500 仕様は、厳密なディレクトリー階層を使用して信頼を確立します。対照的に、インターネットおよびエクストラネットのデプロイメントには、階層的な X.500 アプローチに準拠していない分散信頼モデルが含まれることがよくあります。
- **証明書の使用**。一部の組織では、証明書の使用方法を制限しています。たとえば、一部の証明書はクライアント認証のみに制限される場合があります。

- **複数の証明書**証明書ユーザーが、同じサブジェクト名でキーマテリアルが異なる複数の証明書を所有していることは珍しくありません。この場合、どのキーと証明書をどの目的に使用するかを特定する必要があります。
- **代替名**。目的によっては、証明書の公開鍵にもバインドされている代替サブジェクト名があると便利です。
- **追加属性**。一部の組織では、ディレクトリーで情報を検索できない場合など、追加情報を証明書に保存します。
- **CA に関連**。証明書チェーンに中間 CA が含まれる場合は、CA 間の関係に関する情報を証明書に埋め込むと便利です。
- **CRL チェック**。証明書の失効ステータスをディレクトリーに対して、または元の認証局で常に確認できるとは限らないため、証明書に CRL を確認する場所に関する情報を含めると便利です。

X.509 v3 仕様は、証明書の拡張機能の一般的な形式を定義し、証明書に含めることができる拡張機能を指定することにより、証明書の形式を変更して証明書内に追加情報を含めることにより、これらの問題に対処しました。X.509 v3 証明書用に定義された拡張機能により、追加の属性をユーザーまたは公開鍵に関連付け、証明書階層を管理できます。『インターネット X.509 公開鍵インフラストラクチャー証明書および CRL プロファイル』は、インターネット証明書に使用する一連の拡張機能と、証明書または CA 情報の標準的な場所を推奨しています。これらの拡張機能は **標準拡張機能** と呼ばれます。



注記

標準拡張の詳細は、RFC 2459、RFC 3280、および RFC 3279 を参照してください。

証明書の X.509 v3 標準を使用すると、組織はカスタム拡張機能を定義して証明書に含めることができます。これらの拡張機能は、**プライベート** 拡張機能、**プロプライエタリー** 拡張機能、または**カスタム** 拡張機能と呼ばれ、組織またはビジネスに固有の情報を伝達します。アプリケーションは、プライベートの重要な拡張機能を含む証明書を検証できない可能性があるため、これらを広範囲の状況で使用することは推奨されません。

X.500 および X.509 の仕様は、主に大手通信事業者や政府機関などの国際的な通信ネットワーク関係者を対象とした国際機関である ITU (International Telecommunication Union) によって管理されています。インターネットの基礎となる標準の多くを管理する IETF (Internet Engineering Task Force) は、現在、公開鍵インフラストラクチャー X.509 (PKIX) 標準を開発しています。これらの提案された標準は、インターネットで使用するための拡張機能に対する X.509v3 アプローチをさらに改良します。証明書および CRL の推奨事項は、提案された標準ステータスに達しており、**PKIX パート 1** と呼ばれるドキュメントに記載されています。

他の 2 つの標準、Abstract Syntax Notation One (ASN.1) および Distinguished Encoding Rules (DER) は、Certificate System と一般的な証明書で使用されます。これらは CCITT 勧告 X.208 および X.209 で指定されます。ASN.1 および DER の概要については、RSA Laboratories の Web サイト (http://www.rsa.com) で入手できる『A Layman's Guide to a Subset of ASN.1, BER, and DER 』を参照してください。

5.4.5.1. 証明書の拡張機能の構造

RFC 3280 では、X.509 証明書拡張は以下のように定義されます。

```
Extension  ::=  SEQUENCE  {

extnID OBJECT IDENTIFIER,
```

```
critical BOOLEAN DEFAULT FALSE,
extnValue OCTET STRING  }
```

証明書の拡張機能が次のもので設定されていることを意味します。

- 拡張のオブジェクト識別子 (OID)。この識別子は、拡張子を一意に識別します。また、値フィールドの ASN.1 タイプの値や、値が解釈される方法も決定します。拡張機能が証明書に表示されると、OID は拡張機能 ID フィールド (**extnID**) として示されます。また、対応する ASN.1 エンコード構造は、オクテット文字列の値 (**extnValue**) として示されます。
- **critical** というフラグまたはブール値フィールド

  このフィールドに割り当てられた値 (**true** または **false** のいずれか) は、拡張子が証明書にとって重要かどうかを示します。

  - 拡張機能が重要であり、拡張機能の ID に基づいて拡張機能を理解しないアプリケーションに証明書が送信された場合は、アプリケーションが証明書を拒否する必要があります。
  - 拡張機能が重要ではなく、拡張機能の ID に基づいて拡張機能を理解しないアプリケーションに証明書が送信された場合、アプリケーションは拡張機能を無視して証明書を受け入れることができます。
- 拡張の値の DER エンコーディングを含む octet 文字列。

通常、証明書を受信するアプリケーションは、拡張 ID をチェックして、ID を認識できるかどうかを判断します。可能であれば、エクステンション ID を使用して、使用する値のタイプを判断します。

X.509 v3 標準仕様で定義されている標準拡張には、以下が含まれます。

- 証明書の署名に使用されるキーである CA の公開キーを識別する Authority Key Identifier 拡張。
- サブジェクトの公開キーを識別するサブジェクトキー識別子拡張。キーは認証されています。

注記

すべてのアプリケーションがバージョン 3 拡張機能の証明書をサポートするわけではありません。これらの拡張機能をサポートするアプリケーションは、これらの特定の拡張機能の一部またはすべてを解釈できない場合があります。

5.4.6. 証明書プロファイルの使用およびカスタマイズ

証明書には、タイプとアプリケーションが異なります。これらは、企業ネットワークのシングルサインオン環境の確立、VPN のセットアップ、電子メールの暗号化、または Web サイトへの認証に使用できます。異なる種類のユーザーに対して同じタイプの証明書に対して異なる要件が存在する可能性があるのと同様に、これらすべての証明書の要件は異なる可能性があります。これらの証明書の特性は、**証明書プロファイル** で設定されます。Certificate Manager は、ユーザーまたはマシンが証明書を要求するときに登録フォームとして使用する一連の証明書プロファイルを定義します。

証明書プロファイル
証明書プロファイルは、認証方法、証明書の内容 (デフォルト)、内容の値の制約、証明書プロファイルの入力と出力の内容など、特定の種類の証明書の発行に関連するすべてを定義します。登録要求は証明書プロファイルに送信され、その証明書プロファイルで設定されたデフォルトと制約の対象となります。これらの制約は、要求が証明書プロファイルに関連付けられた入力フォームを介して送信される

か、他の手段を介して送信されるかに関係なく適用されます。証明書プロファイル要求から発行される証明書には、デフォルトで必要なコンテンツと、デフォルトのパラメーターで必要な情報が含まれています。制約は、証明書で許可されるコンテンツに対するルールを提供します。

たとえば、ユーザー証明書の証明書プロファイルは、証明書の有効期間を含む、その証明書のすべての側面を定義します。デフォルトの有効期間は 2 年に設定でき、この証明書プロファイルを介して要求された証明書の有効期間が 2 年を超えてはならないという制約をプロファイルに設定できます。ユーザーがこの証明書プロファイルに関連付けられた入力フォームを使用して証明書を要求すると、発行された証明書にはデフォルトで指定された情報が含まれ、2 年間有効です。ユーザーが、有効期間が 4 年の証明書を事前にフォーマットされた要求を提出した場合、制約により、このタイプの証明書の有効期間は最大 2 年となるため、要求は拒否されます。

発行する最も一般的な証明書プロファイルのセットが事前定義されました。これらの証明書プロファイルは、デフォルトと制約を定義し、認証方法を関連付け、証明書プロファイルに必要な入力と出力を定義します。

証明書プロファイルパラメーターの変更

デフォルトの証明書プロファイルのパラメーターは変更できます。これには、認証方法、デフォルト、各プロファイルで使用される制約、プロファイル内の任意のパラメーターに割り当てられた値、入力、および出力が含まれます。他のタイプの証明書用に新しい証明書プロファイルを作成したり、証明書タイプ用に複数の証明書プロファイルを作成したりすることもできます。特定のタイプの証明書に複数の証明書プロファイルがあり、同じタイプの証明書を異なる認証方法またはデフォルトと制約の異なる定義で発行できます。たとえば、SSL/TLS サーバー証明書の登録用に 2 つの証明書プロファイルがあり、1 つの証明書プロファイルは 6 カ月の有効期間の証明書を発行し、もう 1 つの証明書プロファイルは 2 年の有効期間の証明書を発行します。

入力は、登録フォームのテキストフィールドと、エンドエンティティーから収集する必要のある情報の種類を設定します。これには、証明書要求を貼り付けるためのテキスト領域の設定が含まれます。これにより、必要な要求情報を使用して、入力フォームの外部で要求を作成できます。入力値は、証明書の値として設定されます。デフォルトの入力は、Certificate System で設定できません。

出力は、正常な登録に対する応答ページがどのように表示されるかを指定します。通常は、ユーザーが読み取り可能な形式で証明書を表示します。デフォルトの出力には、結果の証明書の出力可能なバージョンが表示されます。他の出力は、PKCS #7 など、登録の最後に生成される情報のタイプを設定します。

ポリシーセットは、プロファイルを通じて処理されるすべての証明書に付加される制約とデフォルトの拡張機能のセットです。拡張機能は、有効期間やサブジェクト名の要件などの証明書の内容を定義します。プロファイルは 1 つの証明書要求を処理しますが、1 つの要求に複数の証明書の情報を含めることができます。PKCS#10 リクエストには、公開鍵が 1 つ含まれています。CRMF 要求には複数の公開鍵 (つまり、複数の証明書要求) を含めることができます。プロファイルには複数のポリシーセットが含まれる場合があり、各セットは CRMF 要求内で 1 つの証明書要求を処理する方法を指定します。

証明書プロファイルの管理

管理者は、既存の認証プラグインまたはメソッドを証明書プロファイルに関連付けることにより、証明書プロファイルを設定します。デフォルトと制約を有効にして設定し、入力と出力を定義します。管理者は、既存の証明書プロファイルを使用したり、既存の証明書プロファイルを変更したり、新しい証明書プロファイルを作成したり、この PKI で使用されない証明書プロファイルを削除したりできます。

証明書プロファイルが設定されると、エージェントサービスページの **Manage Certificate Profiles** ページに表示され、エージェントは証明書プロファイルを承認して有効にすることができます。証明書プロファイルを有効にすると、エンドエンティティーページの **Certificate Profile** タブに表示され、エンドエンティティーは証明書プロファイルを使用して証明書を登録できます。

エンドエンティティーインターフェイスの証明書プロファイル登録ページには、エージェントによって有効にされた各証明書プロファイルへのリンクが含まれています。エンドエンティティーがこれらのリンクの 1 つを選択すると、その証明書プロファイルに固有の登録フォームを含む登録ページが表示され

ます。登録ページは、プロファイルに定義された入力から動的に生成されます。認証プラグインが設定されている場合、ユーザーを認証するために追加のフィールドが追加される場合があります。

エンドエンティティーが、エージェントが承認した (手動) 登録 (認証プラグインが設定されていない登録) に関連付けられた証明書プロファイル要求を送信すると、証明書要求はエージェントサービスインターフェイスのキューに入れられます。エージェントは、登録のいくつかの側面を変更、要求、検証、キャンセル、拒否、更新、または承認することができます。エージェントは、リクエストを送信せずに更新したり、リクエストがプロファイルのデフォルトと制約に準拠していることを検証したりできます。この検証手順は検証のみを目的としており、リクエストが送信されることはありません。エージェントは、設定された制約に拘束されます。制約に違反するような方法でリクエストを変更することはできません。署名付き承認は即座に処理され、証明書が発行されます。

証明書プロファイルが認証方法に関連付けられている場合は、要求がすぐに承認され、ユーザーが認証に成功し、必要なすべての情報が提供され、要求が証明書プロファイルに設定された制約に違反しない場合は、証明書が自動的に生成されます。サブジェクト名や有効期間など、ユーザーが指定した設定を許可するプロファイルポリシーがあります。証明書プロファイルフレームワークは、発行した証明書の元の証明書要求に設定されたユーザー定義のコンテンツを保持することもできます。

発行された証明書には、証明書の延長や有効期間など、この証明書プロファイルのデフォルトで定義されたコンテンツが含まれています。証明書の内容は、デフォルトごとに設定された制約によって制限されます。1つのプロファイルに複数のポリシー (デフォルトと制約) を設定し、ポリシーセット ID で同じ値を使用して各セットを区別できます。これは、暗号鍵と署名鍵が同じプロファイルに送信されるデュアルキー登録を処理する場合に特に便利です。サーバーは、受け取る各リクエストで各セットを評価します。1つの証明書が発行されると、1つのセットが評価され、その他のセットは無視されます。デュアルキーペアが発行されると、最初のセットは最初の証明書要求で評価され、2 番目のセットは 2 番目の証明書要求で評価されます。単一の証明書を発行するために複数のセット、またはデュアルキーペアを発行するために複数のセットは必要ありません。

証明書プロファイルのカスタマイズガイドライン
組織のプロファイルを、組織が使用する実際のニーズと予想される証明書の種類に合わせて調整します。

- PKI で必要となる証明書プロファイルを決定します。発行される証明書のタイプごとに、少なくとも1つのプロファイルが必要です。証明書の種類ごとに複数の証明書プロファイルを用意して、特定の種類の証明書に対して異なる認証方法や異なるデフォルトや制約を設定することができます。管理インターフェイスで使用可能な証明書プロファイルはすべて、エージェントによって承認され、エンドエンティティーによって登録に使用されます。
- 使用されていない証明書プロファイルを削除します。
- 会社の証明書の特定の特性について、既存の証明書プロファイルを変更します。
    - 証明書プロファイルに設定されているデフォルト、デフォルトに設定されているパラメーターの値、または証明書の内容を制御する制約を変更します。
    - パラメーターの値を変更して、設定された制約を変更します。
    - 認証方法を変更します。
    - 入力ページのフィールドを制御する証明書プロファイルの入力を追加または削除して、入力を変更します。
    - 出力を追加または削除します。

5.4.6.1. SSL サーバー証明書への SAN 拡張機能の追加

Certificate System を使用すると、ルート以外の CA またはその他の Certificate System インスタンス

のインストール時に、SSL サーバー証明書にサブジェクト代替名 (SAN) 拡張機能を追加できます。これを行うには、**/usr/share/pki/ca/profiles/ca/caInternalAuthServerCert.cfg** ファイルの指示に従い、**pkispawn** ユーティリティーに提供されている設定ファイルに次のパラメーターを追加します。

pki_san_inject

このパラメーターの値を **True** に設定します。

pki_san_for_server_cert

必要な SAN 拡張機能の一覧をコンマで区切って指定します。

以下に例を示します。

```
pki_san_inject=True
pki_san_for_server_cert=intca01.example.com,intca02.example.com,intca.example.com
```

5.4.7. 認証方法の計画

「証明書プロファイルの使用およびカスタマイズ」 で暗示されるように、証明書プロセスの **認証** とは、証明書を要求するユーザーまたはエンティティーが、自分が本人であることを証明する方法を意味します。Certificate System がエンティティーを認証できる方法は 3 つあります。

- **エージェントの承認** 登録では、承認のためにエンドエンティティーがエージェントに送信されます。エージェントは証明書要求を承認します。
- **自動** 登録では、エンドエンティティー要求はプラグインを使用して認証され、証明書要求が処理されます。エージェントは登録プロセスに関与していません。
- **CMC 登録** では、サードパーティーのアプリケーションが、エージェントによって署名されてから自動的に処理される要求を作成できます。

Certificate Manager は、最初にエージェント承認の登録と CMC 認証用に設定されています。自動登録を有効にするには、認証プラグインモジュールのいずれかを設定します。サブシステムの単一インスタンスで、1 つ以上の複数の認証方法を設定できます。HTML 登録ページには、使用されるメソッドを指定する非表示値が含まれます。証明書プロファイルを使用すると、有効なプロファイルごとにエンドエンティティー登録ページが動的に生成されます。この証明書プロファイルに関連付けられた認証方法は、動的に生成される登録ページで指定します。

認証プロセスは単純です。

1. エンドエンティティーは登録のリクエストを送信します。リクエストの送信に使用されるフォームは、認証および登録のメソッドを識別します。すべての HTML フォームはプロファイルにより動的に生成され、適切な認証方法をフォームと自動的に関連付けます。
2. 認証方法がエージェント承認の登録である場合、要求は CA エージェントの要求キューに送信されます。キューの要求に対する自動通知が設定されている場合は、新しい要求が受信された適切なエージェントにメールが送信されます。エージェントは、そのフォームに対して許可される要求とそのプロファイル制約を修正できます。承認されると、要求は Certificate Manager に設定された証明書プロファイルに合格する必要があり、合格すると証明書が発行されます。証明書が発行されると、内部データベースに保存され、シリアル番号または要求 ID によってエンドエンティティーページのエンドエンティティーによって取得できます。
3. 認証方法が自動化されている場合、エンドエンティティーは、LDAP ユーザー名やパスワードなど、ユーザーを認証するために必要な情報とともに要求を送信します。ユーザーが正常に認証されると、リクエストはエージェントのキューに送信されずに処理されます。要求が

Certificate Manager の証明書プロファイル設定に合格すると、証明書が発行され、内部データベースに保管されます。HTML フォームを介して即座に終了エンティティーに配信されます。

証明書要求の認証方法の要件は、必要なサブシステムとプロファイル設定に直接影響を与える可能性があります。たとえば、エージェントが承認した登録で、エージェントが要求者に直接会い、裏付けされた書類を使用して身元を確認する必要がある場合、認証プロセスは時間がかかるだけでなく、エージェントと要求者の両方の物理的な可用性に制約を受ける可能性があります。

5.4.8. 証明書および CRL の公開

CA は、証明書と CRL の両方を公開できます。証明書は、プレーンファイルまたは LDAP ディレクトリーに公開できます。CRL は、ファイルまたは LDAP ディレクトリーに公開することもできます。また、証明書の検証を処理するために OCSP レスポンダーに公開することもできます。

パブリッシュの設定は非常にシンプルで、簡単に調整できます。ただし、継続性とアクセシビリティーのためには、証明書および CRL をどこで公開する必要があるか、およびクライアントがそれらにアクセスできるようにする必要があるかを計画することが推奨されます。

LDAP ディレクトリーに公開するには、公開が機能するようにディレクトリーの特別な設定が必要です。

- 証明書がディレクトリーに公開されている場合、証明書が発行されるすべてのユーザーまたはサーバーに、LDAP ディレクトリーに対応するエントリーがある必要があります。
- CRL がディレクトリーに公開されている場合は、CRL を発行した CA のエントリーに公開する必要があります。
- SSL/TLS の場合、ディレクトリーサービスは SSL/TLS で設定する必要があり、任意で、Certificate Manager が証明書ベースの認証を使用できるように設定する必要があります。
- ディレクトリー管理者は、LDAP ディレクトリーへの DN (エントリー名) とパスワードベースの認証を制御するための適切なアクセス制御ルールを設定する必要があります。

5.4.9. CA 署名証明書の更新または再発行

CA 署名証明書の有効期限が切れると、CA の対応する署名キーで署名されたすべての証明書が無効になります。エンドエンティティーは CA 証明書の情報を使用して、証明書の信頼性を検証します。CA 証明書自体が期限切れになると、アプリケーションは証明書を信頼できる CA にチェーンできません。

CA 証明書の有効期限を解決する方法は 2 つあります。

- CA 証明書を **更新** するには、古い CA 証明書と同じサブジェクト名と公開鍵および秘密鍵の内容を使用し、有効期間を延長した新しい CA 証明書を発行する必要があります。古い CA 証明書の有効期限が切れる前に、新しい CA 証明書がすべてのユーザーに配布される限り、証明書を更新すると、古い CA 証明書で発行された証明書は、有効期間の全期間にわたって機能し続けることができます。
- CA 証明書を **再発行すること** には、新しい名前、公開鍵と秘密鍵の内容、および有効期限を持つ新しい CA 証明書を発行することが含まれます。これにより、CA 証明書の更新に関連するいくつかの問題が回避されますが、管理者とユーザーの両方が実装するためにより多くの作業が必要になります。有効期限が切れていないものも含め、古い CA によって発行されたすべての証明書は、新しい CA によって更新する必要があります。

CA 証明書の更新または再発行には、問題があり、利点があります。Certificate Manager をインストールする前に、CA 証明書の更新または再発行の計画を開始し、計画された手順が PKI デプロイメントの拡張、ポリシー、およびその他の側面に与える影響を検討します。

注記

拡張機能 (たとえば **authorityKeyIdentifier** 拡張機能) の正しい使用は、古い CA 証明書から新しい CA 証明書への移行に影響を与える可能性があります。

5.5. ネットワークおよび物理セキュリティーの計画

Certificate System サブシステムをデプロイするときは、サブシステムによって生成および保存されるデータの機密性のために、サブシステムインスタンスの物理的およびネットワークセキュリティーを考慮する必要があります。

5.5.1. ファイアウォールの考慮事項

Certificate System サブシステムでファイアウォールを使用する方法は 2 つあります。

- 機密サブシステムを不正アクセスから保護
- ファイアウォール外部のその他のサブシステムおよびクライアントへの適切なアクセスを許可する

CA、KRA、および TKS には、セキュリティーが侵害された場合に壊滅的なセキュリティー結果を引き起こす可能性のある重要な情報が含まれているため、常にファイアウォールの内側に置かれます。

TPS および OCSP は、ファイアウォールの外に置くことができます。同様に、Certificate System で使用される他のサービスやクライアントは、ファイアウォールの外側にある別のマシンに置くことができます。この場合、ファイアウォールの背後にあるサブシステムとファイアウォールの外側にあるサービスとの間のアクセスを許可するようにローカルネットワークを設定する必要があります。

LDAP データベースは、それを使用するサブシステムとは異なるネットワーク上であっても、異なるサーバー上に配置できます。この場合、ディレクトリーサービスへのトラフィックを許可するために、すべての LDAP ポート (デフォルトでは LDAP の場合は **389**、LDAPS の場合は **636**) を開く必要があります。LDAP データベースにアクセスしないと、すべてのサブシステム操作が失敗します。

ファイアウォールの設定の一環として、iptables が有効になっている場合は、適切な Certificate System ポートを介した通信を許可するようにポリシーを設定する必要があります。iptables の設定は、『firewalld の使用および設定』ガイドを参照してください。

5.5.2. 物理セキュリティーと場所を考慮事項

それらが保持する機密データのため、CA、KRA、および TKS を適切な物理的アクセス制限のある安全な施設に保管することを検討してください。システムへのネットワークアクセスを制限する必要があるのと同様に、物理アクセスも制限する必要があります。

安全な場所を見つけることに加えて、サブシステムのエージェントと管理者への近さを考慮してください。たとえば、キーリカバリーには、複数のエージェントの承認が必要となる場合があります。これらのエージェントが広い地域に分散している場合は、時間差がキーの取得能力に悪影響を及ぼす可能性がある。サブシステムの物理的な場所を計画し、次にサブシステムを管理するエージェントと管理者の場所に従って計画します。

5.5.3. ポートの計画

各 Certificate System サーバーは、最大 4 つのポートを使用します。

- 認証を必要としないエンドエンティティーサービスのセキュアではない HTTP ポート

- 認証を必要とするエンドエンティティーサービス、エージェントサービス、管理コンソール、および管理サービス用の安全な HTTP ポート
- Tomcat Server Management ポート
- Tomcat AJP コネクターポート

『Red Hat Certificate System 管理ガイド』の『Red Hat Certificate System ユーザーインターフェイス』の章で説明されているすべてのサービスページとインターフェイスは、インスタンスの URL と対応するポート番号を使用して接続されます。以下に例を示します。

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

管理コンソールにアクセスするには、URL は管理ポートを指定します。

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

すべてのエージェントおよび管理機能に、SSL/TLS クライアント認証が必要です。エンドエンティティーからの要求の場合、Certificate System は SSL/TLS (暗号化) ポートと非 SSL/TLS ポートの両方でリッスンします。

ポートは **server.xml** ファイルで定義します。ポートを使用しない場合は、そのファイルで無効にできます。以下に例を示します。

```
<Service name="Catalina">
   <!--Connector port="8080" ... /-->  unused standard port
   <Connector port="8443" ... />
```

新しいインスタンスをインストールするときは常に、新しいポートがホストシステム上で一意であることを確認してください。

ポートが使用できることを確認するには、オペレーティングシステムに適切なファイルを確認します。ネットワークアクセス可能なサービスのポート番号は通常、**services** という名前のファイルで維持されます。Red Hat Enterprise Linux では、現在 SELinux コンテキストを持っているすべてのポートを一覧表示する **semanage port -l** コマンドを実行して、ポートが SELinux によって割り当てられていないことを確認することも役立ちます。

新しいサブシステムインスタンスが作成されると、1〜65535 の任意の番号をセキュアポート番号として指定できます。

5.6. CERTIFICATE SYSTEM サブシステムのキーおよび証明書を保存するトークン

トークン は、暗号化機能を実行し、公開鍵証明書、暗号化鍵、およびその他のデータを格納するハードウェアまたはソフトウェアデバイスです。 Certificate System は、Certificate System サブシステムに属するキーペアと証明書を格納するために、**内部** と **外部** の 2 種類のトークンを定義します。

内部 (ソフトウェア) トークンは、通常は **証明書データベース** (**cert9.db**) および **キーデータベース** (**key4.db**) と呼ばれるファイルのペアで、Certificate System がキーペアと証明書を生成および保存するために使用します。Certificate System は、最初に内部トークンを使用するときに、ホストマシンのファイルシステムにこれらのファイルを自動的に生成します。これらのファイルは、キーペアの生成に内部トークンが選択されている場合、Certificate System サブシステムの設定時に作成されます。

これらのセキュリティーデータベースは、**/var/lib/pki/*instance_name*/alias** ディレクトリーにあります。

外部トークンとは、スマートカードやハードウェアセキュリティーモジュール (HSM) など、Certificate System がキーペアと証明書を生成および保存するために使用する外部ハードウェアデバイスを指します。Certificate System は、PKCS #11 に準拠するハードウェアトークンに対応します。

PKCS#11 は、暗号化デバイスの詳細からアプリケーションを分離する API と共有ライブラリーの標準セットです。これにより、アプリケーションは PKCS #11 準拠の暗号化デバイスに統合されたインターフェイスを提供できます。

Certificate System に実装されている PKCS #11 モジュールは、さまざまな製造元が提供する暗号化デバイスをサポートしています。このモジュールを使用すると、Certificate System は、外部暗号化デバイスの製造元が提供する共有ライブラリーをプラグインし、それを使用して、Certificate System マネージャーのキーおよび証明書を生成および保存できます。

外部トークンを使用して、Certificate System が使用する鍵ペアと証明書を生成および保存することを検討します。ハードウェアトークンはソフトウェアトークンよりも安全であると見なされることがあるため、これらのデバイスは秘密鍵を保護するためのもう 1 つのセキュリティー対策です。

外部トークンを使用する前に、サブシステムで外部トークンをどのように使用するかを計画します。

- サブシステムのすべてのシステムキーは、同じトークンで生成する必要があります。
- サブシステムは空の HSM スロットにインストールする必要があります。HSM スロットが以前に他のキーを格納するために使用されていた場合は、HSM ベンダーのユーティリティーを使用してスロットの内容を削除します。Certificate System は、デフォルトのニックネームを持つスロットで証明書および鍵を作成できます。適切にクリーンアップされない場合、これらのオブジェクトの名前は以前のインスタンスと共存する可能性があります。

Certificate System は、外部トークンで **ハードウェア暗号化プレーヤー** を使用することもできます。アクセラレーターの多くは、以下のセキュリティー機能を提供します。

- 高速 SSL/TLS 接続。多数の同時登録またはサービス要求に対応するには、速度が重要です。
- 秘密鍵のハードウェア保護これらのデバイスは、ハードウェアトークンから秘密鍵をコピーまたは削除できないようにすることで、スマートカードのように動作します。これは、オンラインの Certificate Manager のアクティブな攻撃から鍵の盗難に対する予防措置として重要です。

Certificate System は、デフォルトで nCipher nShield Connect XC ハードウェアセキュリティーモジュール (HSM) をサポートします。Certificate System がサポートする HSM は、PKCS #11 ライブラリーモジュールがデフォルトのインストールパスにある場合は、インストールの事前設定段階で **modutil** を使用して **pkcs11.txt** データベースに自動的に追加されます。

設定中、**Security Modules** パネルには、サポートされているモジュールと、NSS 内部ソフトウェア PKCS #11 モジュールが表示されます。検出された、サポートされているすべてのモジュールは、**Found** のステータスを示し、**ログイン済み** または **未ログイン** のいずれかとして個別にマークされます。トークンが見つかり、ログインしていない場合には、**Operations** の下にある **Login** を使用してログインできます。管理者がトークンを正常にログインできる場合、パスワードは設定ファイルに保存されます。Certificate System インスタンスの次回の起動時または再起動時に、パスワードストア内のパスワードを使用して、対応する各トークンのログインが試行されます。

管理者は、システムキーの生成に使用されるデフォルトトークンとしてログインしている任意のトークンを選択できます。

5.7. PKI の計画に関するチェックリスト

問： いくつのセキュリティードメインが作成され、どのサブシステムインスタンスが各ドメインに配置されますか。

答： サブシステムは、信頼できる関係がある場合にのみ別のサブシステムと通信できます。セキュリティードメイン内のサブシステムは相互に自動的に信頼できる関係にあるため、サブシステムがどのドメインに参加するかが重要です。セキュリティードメインには、さまざまな証明書発行ポリシー、ドメイン内のさまざまな種類のサブシステム、またはさまざまな Directory Server データベースを含めることができます。各サブシステムが属する場所 (物理マシン上および相互の関係の両方) をマップし、それに応じてセキュリティードメインに割り当てます。

問： 各サブシステムに割り当てるポート。単一の SSL/TLS ポートが必要ですか。それともセキュリティーを強化するためにポートを分離する方がよいでしょうか。

答： 最も安全なソリューションとして、SSL/TLS インターフェイスごとに個別のポートを使用します。ただし、複数のポートを実装する可能性は、ネットワークセットアップ、ファイアウォールルール、およびその他のネットワーク条件によって異なる場合があります。

問： ファイアウォールの内側に配置するサブシステムは何ですか。これらのファイアウォールで保護されたサブシステムにアクセスするために必要なクライアントまたは他のサブシステムと、アクセスが許可される方法LDAP データベースにファイアウォールへのアクセスが許可されているか。

答： サブシステムのインストール場所は、ネットワーク設計によって異なります。OCSP サブシステムは、ユーザーの便宜のためにファイアウォールの外側で動作するように特別に設計されていますが、CA、KRA、および TPS はすべて、セキュリティーを強化するためにファイアウォールの背後で保護する必要があります。

サブシステムの場所を決定するときは、サーバーがアクセスする必要のある **他の** サブシステムまたはサービス (内部データベース、CA、または外部ユーザーを含む) を計画し、ファイアウォール、サブネット、および VPN 設定を確認することが重要です。

問： 物理的にセキュア化する必要があるサブシステムアクセスが付与される方法と、アクセスを誰に付与するか。

答： CA、TKS、および KRA はすべて、非常に機密性の高いキーと証明書の情報を格納します。デプロイメントによっては、サブシステムが実行するマシンへの物理アクセスを制限しないといけない場合があります。その場合、サブシステムとそのホストマシンの両方を、より大規模なインフラストラクチャーインベントリーとセキュリティー設計に含める必要があります。

問： 全エージェントと管理者の物理的な場所サブシステムの物理的な場所。管理者またはエージェントは、サブシステムサービスに適時どのようにアクセスしますか。各地理的な場所またはタイムゾーンにサブシステムが必要ですか。

答： Certificate System ユーザーの物理的な場所は、ネットワークの遅延時間のために、要求の承認やトークンの登録、システムのパフォーマンスなどに影響を与える可能性があります。インストールするサブシステムの場所と数を決定するときは、証明書操作を処理するための応答時間の重要性を考慮に入れる必要があります。

問： インストールが必要なサブシステムの数

答： 主な考慮事項は、各サブシステムの予想される負荷であり、次に、地理的または部門的な分割です。負荷がかなり低いサブシステム (KRA や TKS など) では、PKI 全体に対して単一のインスタンスのみが必要になる場合があります。高負荷のシステム (CA)、またはローカルエージェント (TPS など) のローカルインスタンスの恩恵を受ける可能性のあるシステムでは、複数のインスタンスが必要になる場合があります。

サブシステムは複製できます。つまり、サブシステムは基本的にクラスター化され、単一のユニットとして動作します。これは、負荷分散と高可用性に適しています。クローン作成は、CA と KRA に特に適しています。この場合、同じキーと証明書のデータに多数のユーザーがアクセスしたり、信頼性の高いフェイルオーバーを行う必要があります。

複数のサブシステムインスタンスを計画するときは、サブシステムが確立されたセキュリティードメイン内にどのように適合するかを覚えておいてください。セキュリティードメインは、サブシステム間に信頼できる関係を構築し、サブシステムが連携して、差し迫ったニーズに対応するために利用可能なサブシステムを見つけることを可能にします。あらゆる種類のサブシステムの複数のインスタンスを使用して、単一の PKI で複数のセキュリティードメインを使用することも、すべてのサブシステムに単一のセキュリティードメインを使用することもできます。

問： サブシステムのクローンを作成する必要がありますか。その場合、キーのマテリアルを安全に保管する方法は何ですか。

答： クローン作成されたサブシステムは、基本的に単一のインスタンスとして動作します。これは、需要の高いシステム、フェイルオーバー、または負荷分散には適していますが、保守が困難になる可能性があります。たとえば、複製された CA には、発行する証明書のシリアル番号の範囲があり、クローンはその範囲の終わりに達する可能性があります。

問： サブシステムの証明書とキーは、Certificate System の内部ソフトウェアトークンまたは外部ハードウェアトークンに保存されますか。

答： Certificate System は、nCipher nShield Connect XC と Thales Luna HSM の 2 つのハードウェアセキュリティーモジュール (HSM) をサポートします。ハードウェアトークンを使用すると、サブシステムをインストールする前に追加のセットアップと設定が必要になる場合がありますが、セキュリティーの別のレイヤーも追加されます。

問： CA 署名証明書の要件Certificate System で、有効期間などの属性を制御する必要があるか。CA 証明書が配布される方法

答： ここでの本当の問題は、CA が証明書の発行に関する独自のルールを設定するルート CA になるのか、それとも他の CA (PKI 内の別の CA または外部 CA) が発行できるどの種類の証明書に制限を設定するのかということです。

Certificate Manager は、ルート CA または下位 CA として設定できます。ルート CA と下位 CA の相違点は、CA 署名証明書に署名することです。ルート CA は、独自の証明書に署名します。下位 CA には、別の CA (内部または外部) がその証明書に署名します。

自己署名ルート CA の問題であり、独自の CA 署名証明書に署名します。これにより、CA は、有効期間や許可される従属 CA の数など、独自の設定ルールを設定できます。

下位 CA には、パブリック CA または別の Certificate System ルート CA によって発行された証明書があります。この CA は、他の CA が発行できる証明書の種類、証明書に含めることができる拡張子、下位 CA が作成できる下位 CA のレベルなど、証明書の設定や証明書の使用方法に関するルールに **従属** しています。

1 つの方法として、Certificate Manager をパブリック CA に従属させる方法があります。これは、下位 CA が発行できる証明書の種類と証明書チェーンの性質にパブリック CA が課す制限を導入するため、非常に制限される可能性があります。一方、パブリック CA にチェーンする利点の 1 つは、サードパーティーがルート CA 証明書を Web ブラウザーまたは他のクライアントソフトウェアに送信する責任があることです。これは、管理者が制御することはできないブラウザーを使用してさまざまな企業がアクセスする証明書の主な利点です。

もう 1 つのオプションは、CA を Certificate System CA に従属させることです。Certificate System CA をルート CA として設定すると、発行した CA 署名証明書の内容を制御するポリシーを設定し、Certificate System 管理者がすべての下位 CA を制御することができます。

最初の CA が自己署名ルートをインストールし、サードパーティーに適用して証明書を発行するのを待つのが最も簡単な方法です。ルート CA の数と、ルート CA と従属 CA の両方を配置する場所を必ず決めておいてください。

問： 発行する証明書の種類どのような特性が必要であり、それらの特性に使用できるプロファイル設定は何ですか。証明書に必要な制限

答： 「証明書プロファイルの使用およびカスタマイズ」で触れたように、プロファイル (CA によって発行される各タイプの証明書を設定するフォーム) をカスタマイズできます。これは、サブジェクト DN、有効期間、SSL/TLS クライアントのタイプ、およびその他の要素を、特定の目的のために管理者が定義できることを意味します。セキュリティーを確保するため、プロファイルは PKI で必要な機能のみを提供する必要があります。不要なプロファイルは無効にする必要があります。

問： 証明書要求を承認するための要件。要求側が自身を認証する方法と、要求を承認するのに必要なプロセスの種類。

答： 要求承認プロセスは、証明書のセキュリティーに直接影響を及ぼします。自動化されたプロセスははるかに高速ですが、リクエスターの ID は表面的にしか検証されないため、安全性は低くなります。同様に、エージェントが承認した登録はより安全です (最も安全なシナリオでは、直接の確認とサポート ID が必要になる可能性があるため) が、最も時間がかかる可能性もあります。

まず、ID 検証プロセスの安全性を判断し、次にその ID を検証するために必要な認証 (承認) メカニズムを判断します。

問： 多くの外部クライアントが証明書のステータスを検証する必要があるか。Certificate Manager の内部 OCSP が負荷を処理しているか。

答： CRL の公開および証明書の検証は重要です。証明書のステータスをチェックするクライアントの種類 (主に内部クライアントなのか、外部クライアントなのか、ユーザー証明書またはサーバー証明書を検証するのか) を決定し、実行される OCSP チェックの数も決定します。CA には内部チェックや頻度の低いチェックに使用できる内部 OCSP がありますが、多数の OCSP チェックを行うと CA のパフォーマンスが低下する可能性があります。多数の OCSP チェック、特に大規模な CRL の場合は、別の OCSP マネージャーを使用して CA の負荷を軽減することを検討してください。

問： PKI が代替キーを許可するか ?キーアーカイブとリカバリーが必要になりますか。

答： 失われた証明書またはトークンが単に取り消されて置き換えられた場合は、それを回復するためのメカニズムは必要ありません。ただし、証明書を使用して電子メール、ファイル、ハードドライブなどのデータに署名または暗号化を行う場合は、データを回復できるように、キーが常に使用可能である必要があります。この場合、データが常にアクセスできるように KRA を使用してキーをアーカイブします。

問： 組織はスマートカードを使用しますか ?その場合は、スマートカードが置き忘れられ、キーのアーカイブとリカバリーが必要になった場合、一時的なスマートカードは許可されますか。

答： スマートカードが使用されていない場合、これらのサブシステムはトークン管理にのみ使用されるため、TPS または TKS を設定する必要はありません。ただし、スマートカードを使用する場合は、TPS および TKS が必要です。トークンが失われて置き換えられる可能性がある場合は、トー

問： **証明書および CRL を公開する場所パブリッシュが機能するには、受信側でどのような設定が必要ですか。どのような種類の証明書または CRL を公開する必要があり、どのくらいの頻度で公開する必要がありますか。**

答： 決定すべき重要なことは、証明書または CRL 情報にアクセスできるようにするために必要なクライアントと、そのアクセスを許可する方法です。そこから、公開ポリシーを定義します。

5.8. 任意のサードパーティーサービス

5.8.1. ロードバランサー

5.8.2. バックアップハードウェアおよびソフトウェア

パート II. RED HAT CERTIFICATE SYSTEM のインストール

本セクションでは、Red Hat Certificate System をインストールするための要件および手順を説明します。

第6章 インストールの前提条件および準備

Red Hat Certificate System のインストールプロセスでは、環境の準備が必要です。この章では、Certificate System サブシステムをインストールする際の要件、依存関係、およびその他の前提条件を説明します。

6.1. RED HAT ENTERPRISE LINUX のインストール

Red Hat Certificate System 10.4 には、Red Hat Enterprise Linux 8.6 バージョンが必要です。Red Hat Enterprise Linux のインストールの詳細は、『標準的な RHEL インストールの実行』を参照してください。

Red Hat Enterprise Linux で FIPS (Federal Information Processing Standard) を有効にするには、『Red Hat セキュリティーの強化ガイド』を参照してください。

重要

Certificate System をインストールする前に、FIPS モードを RHEL ホストで有効にする必要があります。FIPS モードが有効になっていることを確認するには、以下を実行します。

```
# sysctl crypto.fips_enabled
```

返された値が **1** の場合は、FIPS モードが有効になります。

6.2. SELINUX を使用したシステムのセキュリティー保護

SELinux (Security-Enhanced Linux) は、Linux カーネルに必須のアクセス制御メカニズムを実装しておりで、標準の任意アクセス制御がチェックされた後、許可された操作をチェックします。SELinux は、定義されたポリシーに基づいて、Linux システム内のファイルとプロセス、およびそれらのアクションにルールを適用できます。

ほとんどの場合、**enforcing** モードで SELinux を使用して Certificate System を実行する必要はありません。ハードウェアセキュリティーモジュール (HSM) を使用する場合など、Certificate System のドキュメントの手順で SELinux 関連の設定を手動で行う必要がある場合は、対応するセクションに記載されています。

SELinux の詳細は『SELinux の使用』を参照してください。

6.2.1. SELinux が Enforcing モードで実行していることの確認

デフォルトでは、Red Hat Enterprise Linux をインストールした後、SELinux が有効になり、**enforcing** モードで実行します。それ以上のアクションは必要ありません。

現在の SELinux モードを表示するには、次のコマンドを実行します。

```
# getenforce
```

6.3. ファイアウォールの設定

以下の表は、Certificate System サブシステムで使用されるデフォルトのポートを示しています。

表6.1 Certificate System のデフォルトポート

サービス	ポート	プロトコル
HTTP	8080	TCP
HTTPS	8443	TCP
Tomcat 管理	8005	TCP

pkispawn ユーティリティーを使用して Certificate System を設定すると、ポート番号をカスタマイズできます。上記のデフォルトとは異なるポートを使用する場合は、「ファイアウォールで必要なポートを開く」の説明に従ってファイアウォールで対応して開きます。ポートの詳細は、「ポートの計画」を参照してください。

Directory Server にアクセスするために必要なポートについては、『Directory Server インストールガイド』の該当するセクションを参照してください。

6.3.1. ファイアウォールで必要なポートを開く

クライアントと Certificate System と間の通信を有効にするには、ファイアウォールで必要なポートを開きます。

1. **firewalld** サービスが実行中である必要があります。

```
# systemctl status firewalld
```

2. **firewalld** を起動し、システム起動時に自動的に起動するように設定するには、次のコマンドを実行します。

```
# systemctl start firewalld
# systemctl enable firewalld
```

3. **firewall-cmd** ユーティリティーを使用して必要なポートを開きます。たとえば、デフォルトのファイアウォールゾーンで Certificate System のデフォルトポートを開くには、次のコマンドを実行します。

```
# firewall-cmd --permanent --add-port={8080/tcp,8443/tcp,8009/tcp,8005/tcp}
```

firewall-cmd を使用してシステムでポートを開く方法は、『ネットワークのセキュリティー保護』 または firewall-cmd(1) の man ページを参照してください。

4. firewall-cmd 設定を再度読み込んで、変更が即座に反映されるようにします。

```
# firewall-cmd --reload
```

6.4. ハードウェアセキュリティーモジュール

ハードウェアセキュリティーモジュール (HSM) を使用するには、FIPS (Federal Information Processing Standard) 140-2 で検証された HSM が必要です。HSM をインストール、設定、および FIPS モードでセットアップする方法については、HSM のドキュメントを参照してください。

6.4.1. HSM 用の SELinux の設定

特定の HSM では、Certificate System をインストールする前に、手動で SELinux 設定を更新する必要があります。

以下のセクションでは、対応している HSM に必要なアクションを説明します。

nCipher nShield

HSM をインストールし、Certificate System をインストールする前に、以下を行います。

1. **/opt/nfast/** ディレクトリーのファイルのコンテキストをリセットします。

```
# restorecon -R /opt/nfast/
```

2. **nfast** ソフトウェアを再起動します。

```
# /opt/nfast/sbin/init.d-ncipher restart
```

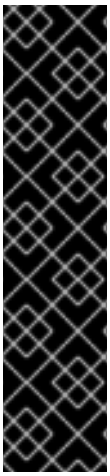
Thales Luna HSM

Certificate System をインストールする前に、SELinux 関連のアクションは必要ありません。

対応している HSM の詳細は、「サポート対象のハードウェアセキュリティーモジュール」を参照してください。

6.4.2. HSM での FIPS モードの有効化

HSM で FIPS モードを有効にするには、特定の手順については、HSM ベンダーのドキュメントを参照してください。



重要

nCipher HSM

nCipher HSM では、FIPS モードが Security World を生成する場合にのみ有効にできます。これは後で変更することはできません。Security World を生成するにはさまざまな方法がありますが、常に **new-world** コマンドを使用することが推奨されます。FIPS 準拠の Security World を生成する方法は、nCipher HSM ベンダーのドキュメントを参照してください。

LunaSA HSM

同様に、Luna HSM で FIPS モードを有効にするには、初期設定時に行う必要があります。これは、このポリシーを変更すると、セキュリティー対策として HSM がゼロになるためです。詳細は、Luna HSM ベンダーのドキュメントを参照してください。

6.4.3. FIPS モードが HSM で有効になっているかどうかの確認

本セクションでは、特定の HSM に対して FIPS モードが有効になっているかどうかを確認する方法を説明します。その他の HSM は、ハードウェアの製造元のドキュメントを参照してください。

6.4.3.1. FIPS モードが nCipher HSM で有効にされているかどうかの確認

注記

完全な手順については、HSM ベンダーのドキュメントを参照してください。

FIPS モードが nCipher HSM で有効になっているかどうかを確認するには、次のコマンドを実行します。

```
# /opt/nfast/bin/nfkminfo
```

古いバージョンのソフトウェアでは、**StrictFIPS140** が state フラグに一覧表示されると、FIPS モードが有効になります。ただし、新しいバージョンでは、新しい **mode** の行を確認して **fips1402level3** を検索することが推奨されます。すべてのケースで、**nfkminfo** 出力には **hkfips** キーも存在しているはずです。

6.4.3.2. FIPS モードが Luna SA HSM で有効にされているかどうかの確認

注記

完全な手順については、HSM ベンダーのドキュメントを参照してください。

FIPS モードが Luna SA HSM で有効になっているかどうかを確認するには、次を実行します。

1. **lunash** 管理コンソールを開きます。
2. **hsm show** コマンドを使用して、出力に **The HSM is in FIPS 140-2 approved operation mode.** の文字が含まれていることを確認します。

```
lunash:> hsm show
...
    FIPS 140-2 Operation:
    =====================
    The HSM is in FIPS 140-2 approved operation mode.
...
```

6.4.4. HSM を使用した Certificate System のインストールの準備

「**pkispawn** ユーティリティーの概要」 では、HSM を使用して Certificate System をインストールする場合は、**pkispawn** ユーティリティーに渡す設定ファイルで以下のパラメーターを使用します。

```
...
[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*** パラメーターおよび ***pki_token_name*** パラメーターの値は、特定の HSM インストールにより異なります。これらの値により、**pkispawn** ユーティリティーで HSM をセットアップし、Certificate System が接続できるようになります。
- ***pki_token_password*** の値は、特定の HSM トークンのパスワードによって異なります。パスワードは、HSM で新しいキーを作成するための **pkispawn** ユーティリティーの読み取りおよび書き込み権限を付与します。
- ***pki_hsm_modulename*** の値は、HSM を識別するため、後続の **pkispawn** 操作で使用される名前です。文字列は、任意のものとして設定できる識別子です。これにより、**pkispawn** および Certificate System は、後の操作で HSM および設定情報を名前で参照できます。

以下のセクションでは、各 HSM の設定を説明します。HSM が一覧にない場合は、HSM の製造元のドキュメントを参照してください。

6.4.4.1. NCipher HSM パラメーター

nCipher HSM の場合は、以下のパラメーターを設定します。

```
pki_hsm_libfile=/opt/nfast/toolkits/pkcs11/libcknfast.so
pki_hsm_modulename=nfast
```

pki_hsm_modulename の値を任意の値に設定することができることに注意してください。上記の値は推奨値です。

例6.1 トークン名の特定

トークン名を特定するには、**root** ユーザーで以下のコマンドを実行します。

```
[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 セクションの ***name*** フィールドの値は、トークン名を一覧表示します。

以下のようにトークン名を設定します。

```
pki_token_name=NHSM-CONN-XC
```

6.4.4.2. SafeNet / Luna SA HSM パラメーター

SafeNet Luna Network HSM などの SafeNet / Luna SA HSM の場合は、次のパラメーターを指定します。

```
pki_hsm_libfile=/usr/safenet/lunaclient/lib/libCryptoki2_64.so
pki_hsm_modulename=lunasa
```

pki_hsm_modulename の値を任意の値に設定することができることに注意してください。上記の値は推奨値です。

例6.2 トークン名の特定

トークン名を特定するには、**root** ユーザーで以下のコマンドを実行します。

```
# /usr/safenet/lunaclient/bin/vtl verify

The following Luna SA Slots/Partitions were found:

Slot    Serial #           Label
====    ================   =====
  0     1209461834772      lunasaQE
```

label 列の値は、トークン名を表示します。

以下のようにトークン名を設定します。

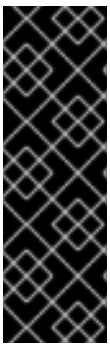
```
pki_token_name=lunasaQE
```

6.4.5. ハードウェアセキュリティーモジュールでのキーのバックアップ

HSM に保存されている鍵と証明書を、**.p12** ファイルにエクスポートすることができません。このようなインスタンスをバックアップする必要がある場合には、HSM の製造元に連絡してサポートしてください。

6.5. RED HAT DIRECTORY SERVER のインストール

Certificate System は、Red Hat Directory Server を使用して、システム証明書とユーザーデータを保存します。Directory Server と Certificate System の両方を、ネットワーク内の同じまたは他のホストにインストールできます。



重要

Directory Server をインストールする前に、FIPS モードを RHEL ホストで有効にする必要があります。FIPS モードが有効になっていることを確認するには、以下を実行します。

```
# sysctl crypto.fips_enabled
```

返された値が **1** の場合は、FIPS モードが有効になります。

6.5.1. Certificate System 用の Directory Server インスタンスの準備

Red Hat Directory Server をインストールするには、以下の手順を実行します。

1. Directory Server を提供するサブスクリプションがホストに登録されていることを確認してください。
2. Directory Server リポジトリーを有効にします。

```
# subscription-manager repos --enable=dirsrv-11-for-rhel-8-x86_64-rpms
```

3. Directory Server および openldap-clients パッケージをインストールします。

```
# dnf module install redhat-ds
```

```
# dnf install openldap-clients
```

4. Directory Server インスタンスを設定します。
   a. DS 設定ファイルを生成します (例: **/tmp/ds-setup.inf**)。

```
$ dscreate create-template /tmp/ds-setup.inf
```

   b. DS 設定ファイルを次のようにカスタマイズします。

```
$ 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. **setup** 設定ファイルを指定して **dscreate** コマンドを使用してインスタンスを作成します。

```
# dscreate from-file /tmp/ds-setup.inf
```

詳細な手順は、『Red Hat Directory Server インストールガイド』を参照してください。

6.5.2. Certificate System の設定の準備

「**pkispawn** ユーティリティーの概要」 では、Certificate System と Directory Server の間に TLS を設定することを選択した場合は、Certificate System をインストールする際に、**pkispawn** ユーティリティーに渡す設定ファイルで次のパラメーターを使用します。

注記

最初に基本的な TLS サーバー認証接続を作成する必要があります。最後に、インストール後、接続に戻り、クライアント認証証明書を Directory Server に提示する必要があります。クライアント認証が設定されていると、***pki_ds_password*** は関連なくなります。

```
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 パラメーターの値は、**pkispawn** ユーティリティーによって使用される名前で、Directory Server インスタンスに対応するサブシステムデータベースを作成します。

pki_ds_hostname パラメーターの値は、Directory Server インスタンスのインストール場所によって異なります。これは、「Certificate System 用の Directory Server インスタンスの準備」 で使用される値によって異なります。

pki_ds_secure_connection=True を設定する場合は、以下のパラメーターを設定する必要があります。

- ***pki_ds_secure_connection_ca_pem_file***: Directory Server の CA 証明書のエクスポートされたコピーを含むファイルのファイル名を含む完全修飾パスを設定します。このファイルが先に存在していないと、**pkispawn** は使用できません。
- ***pki_ds_ldaps_port***: Directory Server がリッスンしているセキュアな LDAPS ポートの値を設定します。デフォルトは **636** です。

6.6. DIRECTORY SERVER (CA) での一時的な自己署名証明書の置き換え

内部 LDAP サーバーが一時的な自己署名サーバー証明書で最初に作成された場合は、インストール後のセクションの 「一時的な証明書の置き換え」 を参照して、一時証明書を CA が発行した新しい証明書に置き換えます。

6.7. 内部 LDAP サーバーの TLS クライアント認証の有効化

Red Hat Certificate System は、TLS 相互認証を介して内部 LDAP サーバーと通信できます。インストールが完了したら、有効化の方法について、インストール後のセクションの 「TLS クライアント認証の有効化」 を参照してください。

6.8. RED HAT サブスクリプションの添付および CERTIFICATE SYSTEM パッケージリポジトリーの有効化

Certificate System をインストールおよび更新する前に、対応するリポジトリーを有効にする必要があります。

1. Red Hat サブスクリプションをシステムに割り当てます。

   システムがすでに登録されているか、Certificate Server を提供するサブスクリプションが割り当てられている場合は、この手順をスキップしてください。

   a. システムを Red Hat サブスクリプション管理サービスに登録する

      --auto-attach オプションを使用して、オペレーティングシステムに利用可能なサブスクリプションを自動的に適用します。

      ```
      # 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. 利用可能なサブスクリプションをリストし、Red Hat Certificate System を提供するプール ID をメモします。以下に例を示します。

```
# subscription-manager list --available --all
...
Subscription Name:   Red Hat Enterprise Linux Developer Suite
Provides:            ...
                     Red Hat Certificate System
                     ...
Pool ID:             7aba89677a6a38fc0bba7dac673f7993
Available:           1
...
```

サブスクリプションが多数ある場合は、コマンドの出力が非常に長くなることがあります。必要に応じて、出力をファイルにリダイレクトできます。

```
# subscription-manager list --available --all > /root/subscriptions.txt
```

c. 前の手順でプール ID を使用して、Certificate System のサブスクリプションをシステムに割り当てます。

```
# subscription-manager attach --pool=7aba89677a6a38fc0bba7dac673f7993
Successfully attached a subscription for: Red Hat Enterprise Linux Developer Suite
```

2. Certificate System リポジトリーを有効にします。

```
# subscription-manager repos --enable certsys-10.x-for-rhel-8-x86_64-rpms
```

ここで、**x** は最新の Certificate System バージョンを示します。たとえば、RHCS 10.4 の Certificate System リポジトリーを有効にするには、次のコマンドを使用します。

```
# 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. Certificate System モジュールストリームを有効にします。

```
# dnf module enable redhat-pki
```

必要なパッケージのインストールについては、*7章Certificate System のインストールおよび設定* に記載されています。

注記

コンプライアンスのために、Red Hat で承認したリポジトリーのみを有効にします。**subscription-manager** ユーティリティーを使用して有効にできるのは、Red Hat 承認済みのリポジトリーのみです。

6.9. CERTIFICATE SYSTEM のオペレーティングシステムのユーザーおよびグループ

Certificate System をインストールする場合は、**pkiuser** アカウントと対応する **pkiuser** グループは自動的に作成されます。Certificate System は、このアカウントとグループを使用してサービスを起動します。

インストール後の手順の一部として、オペレーティングシステム上でインスタンスを起動、停止、または直接設定できるように、各 Certificate System 管理者のオペレーティングシステムユーザーを作成するように指示されます。詳細は、「インストール後のタスク」 を参照してください。

第7章 CERTIFICATE SYSTEM のインストールおよび設定

Red Hat Certificate System は、個別にインストールできるさまざまなサブシステムを提供します。たとえば、複数のサブシステムインスタンスを 1 つのサーバーにインストールすることも、異なるホストで個別に実行することもできます。これにより、インストールを環境に合わせて適応させることができ、より高い可用性、スケーラビリティー、フェイルオーバーのサポートを提供することができます。本章では、パッケージのインストールと、個別のサブシステムの設定方法を説明します。

Certificate System には以下のサブシステムが含まれます。

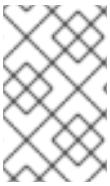
- 認証局 (CA)
- キーリカバリー認証局 (KRA)
- OCSP (Online Certificate Status Protocol) レスポンダーの使用
- トークンキーサービス (TKS)
- トークン処理システム (TPS)

各サブシステムは、スタンドアロン Tomcat Web サーバーインスタンスとして独立してインストールおよび設定されます。ただし、Red Hat Certificate System はさらに、各サブシステムに 1 つまで共有の Tomcat Web サーバーインスタンスの実行をサポートしています。

7.1. サブシステムの設定順序

異なるサブシステム間の関係のため、個々のサブシステムがセットアップされる順序は重要です。

1. 他の公開鍵インフラストラクチャー (PKI) サブシステムをインストールする前に、セキュリティードメインとして実行されている少なくとも 1 つの CA が必要です。
2. CA の設定後に OCSP をインストールします。
3. KRA サブシステムおよび TKS サブシステムは、CA および OCSP の設定後にいつでもインストールできます。
4. TPS サブシステムは CA および TKS に依存し、任意で KRA サブシステムおよび OCSP サブシステムに依存します。



注記

特定の状況では、管理者は、セキュリティードメインとして実行している CA を必要としないスタンドアロンの KRA または OCSP をインストールしたいと考えます。詳細は、「スタンドアロン KRA または OCSP の設定」 を参照してください。

7.2. CERTIFICATE SYSTEM パッケージ

Certificate System パッケージをインストールする場合は、サブシステムごとに個別にインストールすることも、一度にすべてインストールすることもできます。

重要

Certificate Server パッケージをインストールおよび更新するには、対応するリポジトリーを有効にする必要があります。詳細は、「Red Hat サブスクリプションの添付および Certificate System パッケージリポジトリーの有効化」 を参照してください。

Red Hat Certificate System では、以下のサブシステムパッケージとコンポーネントを利用できます。

- pki-ca: 認証局 (CA) サブシステムを提供します。
- pki-kra: Key Recovery Authority (KRA) サブシステムを提供します。
- pki-ocsp: OCSP (Online Certificate Status Protocol) レスポンダーを提供します。
- pki-tks: Token Key Service (TKS) を提供します。
- pki-tps: Token Processing Service (TPS) を提供します。
- pki-console および redhat-pki-console-theme: Java ベースの Red Hat PKI コンソールを提供します。両方のパッケージをインストールする必要があります。
- pki-server および redhat-pki-server-theme: Web ベースの Certificate System インターフェイスを提供します。両方のパッケージをインストールする必要があります。

  pki-ca、pki-kra、pki-ocsp、pki-tks、pki-tps のパッケージのいずれかをインストールすると、このパッケージは依存関係としてインストールされます。

7.2.1. Certificate System パッケージのインストール

- redhat-pki モジュールとともに使用すると、すべての Certificate System サブシステムパッケージとコンポーネントを RHEL 8 システムに一度にインストールできます。redhat-pki モジュールは、Red Hat Certificate System の 5 つのサブシステムをインストールします。Red Hat Identity Management (IdM) の一部である pki-core モジュール (CA、KRA) に加え、RHCS 固有のサブシステム (OCSP、TKS、および TPS) と必要な依存関係を処理する pki-deps モジュールが含まれています。

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

- パッケージを個別にインストールできます。たとえば、CA サブシステムおよび任意の Web インターフェイスをインストールするには、次のコマンドを実行します。

```
# yum install pki-ca redhat-pki-server-theme
```

  その他のサブシステムでは、pki-ca のパッケージ名を、インストールするサブシステムの 1 つに置き換えます。

- オプションの PKI コンソールが必要な場合は、以下を行います。

```
# yum install pki-console redhat-pki-console-theme
```

注記

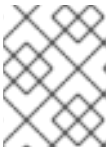
pkiconsole ツールは非推奨になりました。

7.2.2. Certificate System パッケージの更新

Certificate System パッケージおよびオペレーティングシステムパッケージを更新するには、以下の手順に従います。

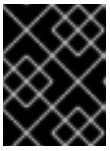
1. 「Certificate System の製品バージョンの特定」 の手順に従って、製品バージョンを確認します。
2. **# yum update** を実行します。

   上記のコマンドは、RHCS パッケージを含むシステム全体を更新します。



注記

PKI インフラストラクチャーをオフラインにして更新をインストールできるメンテナンスウィンドウをスケジュールすることが推奨します。



重要

Certificate System を更新するには、PKI インフラストラクチャーを再起動する必要があります。

3. 次に、「Certificate System の製品バージョンの特定」 でバージョンを再度確認します。

   バージョン番号は、更新が正常にインストールされていることを確認する必要があります。

インストールせずに更新をダウンロードするには、上記の手順で **--downloadonly** オプションを使用します。

```
yum update --downloadonly
```

ダウンロードしたパッケージは **/var/cache/yum/** ディレクトリーに保存されます。最新バージョンであれば、 **yum update** は後でパッケージを使用します。

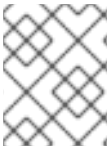
7.2.3. Certificate System の製品バージョンの特定

Red Hat Certificate System の製品バージョンは、**/usr/share/pki/CS_SERVER_VERSION** ファイルに保存されます。バージョンを表示するには、以下を実行します。

```
# cat /usr/share/pki/CS_SERVER_VERSION
Red Hat Certificate System 10.0 (Batch Update 1)
```

実行中のサーバーの製品バージョンを検索するには、ブラウザーから以下の URL にアクセスします。

- **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**



注記

各コンポーネントは個別のパッケージであるため、バージョン番号は異なります。上記は、現在実行中の各コンポーネントのバージョン番号を示しています。

7.3. PKISPAWN ユーティリティーの概要

Red HatCertificate System では、**pkispawn** ユーティリティーを使用して個々の公開鍵インフラストラクチャー (PKI) サブシステムをセットアップします。設定中に **pkispawn** は以下を実行します。

1. **/etc/pki/default.cfg** ファイルからデフォルト値を読み取ります。詳細は、pki_default.cfg(5) の man ページを参照してください。

重要

/etc/pki/default.cfg ファイルは編集しないでください。代わりに、設定ファイルを作成し、デフォルトを上書きして、**pkispawn** ユーティリティーに渡します。設定ファイルの使用方法は、「2 ステップインストール」を参照してください。

2. 設定モードに応じて、パスワードやその他のデプロイメント固有の情報を使用します。
   - インタラクティブモード: セットアップ中、ユーザーは設定時に個々の設定を要求します。このモードは、単純なデプロイメントに使用します。
   - バッチモード: ユーザーは提供する設定ファイルから値が読み込まれます。設定ファイルで設定されていないパラメーターは、デフォルト値を使用します。
3. 要求された PKI サブシステムのインストールを実行します。
4. 設定に基づいて設定を行う Java サーブレットに設定を渡します。

pkispawn ユーティリティーを使用してインストールします。

- ルート CA。詳細は、「ルート認証局の設定」 を参照してください。
- 下位 CA またはその他のサブシステム。詳細は、「追加のサブシステムの設定」 を参照してください。

注記

pkispawn ユーティリティーを使用してルート CA を設定する方法は 「ルート認証局の設定」 を参照してください。下位 CA または非 CA サブシステムの設定は、「外部 CA でのサブシステムの設定」を参照してください。

pkispawn および例の詳細は、pkispawn(8) の man ページを参照してください。

7.4. ルート認証局の設定

認証局 (CA) サブシステムは、その他のすべての認証局サブシステムの前提条件です。そのため、他のサブシステムを設定する前に CA を設定します。

Certificate System にルート CA を設定するには、以下のオプションがあります。

- 設定ファイルベースのインストール:

  この方法は高度なカスタマイズに使用します。このインストール方法は、デフォルトのインストールパラメーターを上書きする設定ファイルを使用します。

  1 つの手順または 2 つの手順で、設定ファイルを使用して Certificate System をインストールできます。詳細および例は、以下を参照してください。

- man ページの pkispawn(8) (単一ステップのインストールについて)
- 「2 ステップインストール」 (2 ステップインストール用)

- 対話型インストール::

```
# pkispawn -s CA
```

必要最小限の設定オプションのみを設定する場合は、対話型インストーラーを使用してください。

7.5. インストール後の設定

以下の手順を実施します。

- 「署名監査ログの有効化」
- 「TLS 暗号の設定」
- 「サブシステムの証明書失効チェックの有効化」

上記の手順を完了したら、インストール後の操作について 「インストール後のタスク」 に従ってください。

7.6. 追加のサブシステムの設定

「ルート認証局の設定」 の説明に従ってルート認証局 (CA) をインストールした後、追加の Certificate System サブシステムをインストールできます。

前提条件
追加のサブシステムにはルート認証局 (CA) が必要になります。ルート Certificate System CA をインストールしていない場合は、「ルート認証局の設定」を参照してください。

サブシステムのインストール
追加のサブシステムを設定するには、以下のオプションを使用できます。

- 設定ファイルベースのインストール:

  この方法は高度なカスタマイズに使用します。このインストール方法は、デフォルトのインストールパラメーターを上書きする設定ファイルを使用します。

  1 つの手順または 2 つの手順で、設定ファイルを使用して Certificate System をインストールできます。詳細および例は、以下を参照してください。

  - man ページの pkispawn(8) (単一ステップのインストールについて)
  - 「2 ステップインストール」 (2 ステップインストール用)

- 対話型インストール::

  必要最小限の設定オプションのみを設定する場合は、対話型インストーラーを使用してください。

  以下に例を示します。

```
# pkispawn -s subsystem
```

subsystem は、**KRA**、**OCSP**、**TKS** または **TPS** のいずれかに置き換えます。

対話型インストーラーは、下位 CA のインストールをサポートしません。下位 CA をインストールするには、2 ステップのインストールを使用します。「2 ステップインストール」 を参照してください。

7.7. 2 ステップインストール

インストール中に特定の設定パラメーターをカスタマイズするには、インストールプロセスを 2 つのステップで実行し、そのステップの間に設定を行う必要があります。このため、**pkispawn** ユーティリティーを使用すると、2 つの手順でサブシステムのインストールを実行できます。

7.7.1. 2 ステップインストールを使用するタイミング

次のようなシナリオでは、2 つのステップのインストールを使用します。

- セキュリティーの強化。
- サブシステム証明書のカスタマイズ。
- 既存の Certificate System に接続する新しい Certificate System インスタンスをインストールする場合に、**/etc/pki/*instance_name*/server.xml** ファイルの ***sslRangeCiphers*** パラメーターの暗号リストをカスタマイズ。
- FIPS モードで CA クローン、KRA、OCSP、TKS、および TPS をインストールします。
- FIPS モードでハードウェアセキュリティーモジュール (HSM) を使用した Certificate System のインストール

7.7.2. 2 ステップインストールの 2 つの主要部分

2 ステップのインストールは、以下の 2 つの主要な部分で設定されます。

1. **インストール**

   このステップでは、**pkispawn** は、**/usr/share/pki/** ディレクトリーから、インスタンス固有のディレクトリー **/etc/pki/*instance_name*/** に設定ファイルをコピーします。さらに、**pkispawn** は、デプロイメント設定ファイルで定義されている値に基づいて設定を行います。

   インストールのこの部分には、以下のサブステップが含まれます。

   1. 「インストールの最初ステップ用の設定ファイルの作成」
   2. 「インストール手順の起動」

2. **設定**

   このステップでは、**pkispawn** インスタンス固有の **/etc/pki/*instance_name*/** ディレクトリーの設定ファイルに基づいてインストールを続行します。

   インストールのこの部分には、以下のサブステップが含まれます。

   1. 「インストール手順間の設定のカスタマイズ」

2. 「設定手順の開始」

7.7.3. インストールの最初ステップ用の設定ファイルの作成

/root/config.txtなどの設定設定用のテキストファイルを作成し、これを以下の設定で入力します。

重要

本セクションでは、Certificate System と同じホストで実行している Directory Server の最低限の設定を説明します。環境に応じて、追加パラメーターが必要になる場合があります。その他の例は、pkispawn(8) の man ページの 『EXAMPLES』 セクションを参照してください。

本セクションで説明するパラメーターの説明は、pki_default.cfg(5) の man ページを参照してください。

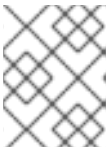
サブシステムに依存しない設定
インストールするサブシステムとは関係なく、設定ファイルには次の設定が必要です。

1. Certificate System **admin** ユーザー、PKCS #12 ファイル、および Directory Server のパスワードを設定します。

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

2. 同じホストで実行している Directory Server への LDAPS 接続を使用するには、設定ファイルの **[DEFAULT]** セクションに以下のパラメーターを追加します。

```
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
```

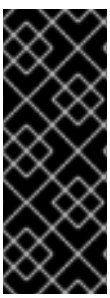


注記

セキュリティー上の理由から、Red Hat は、Directory Server への暗号化された接続を使用することを推奨します。

Directory Server で自己署名証明書を使用する場合は、以下のコマンドを使用して Directory Server の Network Security Services (NSS) データベースからエクスポートします。

```
# certutil -L -d /etc/dirsrv/slapd-instance_name/ \
    -n "server-cert" -a -o /root/ds.crt
```



重要

デフォルトでは、Certificate System は、インストール後に **~/.dogtag/*instance_name*/*subsystem*/alias** クライアントデータベースを削除します。セキュリティー上の理由から、Red Hat は、設定ファイルの ***pki_client_database_purge*** パラメーターを有効にしないことをお勧めします。このパラメーターを手動で **True** に設定した場合、Certificate System は、インストール後にクライアントデータベースを削除しません。

初期設定ファイルを作成したら、サブシステム固有の設定を追加します。以下を参照してください。

- 「CA 設定」
- 「他のサブシステムの設定」

CA 設定

「サブシステムに依存しない設定」 に加え、CA をインストールするために以下の設定が必要になります。

1. セキュリティーを強化するには、設定ファイルに以下が設定された **[CA]** を追加して、ランダムなシリアル番号を有効にします。

```
[CA]
pki_random_serial_numbers_enable=true
```

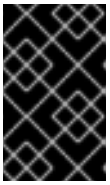
2. 必要に応じて、**[CA]** セクションに以下のパラメーターを設定して、**admin** ユーザーのデータを指定します。これは、インストール時に自動的に作成されます。

```
pki_admin_nickname=caadmin
pki_admin_name=CA administrator account
pki_admin_password=password
pki_admin_uid=caadmin
pki_admin_email=caadmin@example.com
```

   Certificate System は、このアカウントに管理者権限を割り当てます。インストール後にこのアカウントを使用して、Certificate System を管理し、さらにユーザーアカウントを作成します。

3. Certificate System が一意のニックネームを生成できるようにするには、**[DEFAULT]** セクションで次のパラメーターを設定します。

```
pki_instance_name=instance_name
pki_security_domain_name=example.com Security Domain
pki_host=server.example.com
```



重要

ネットワーク共有ハードウェアセキュリティーモジュール (HSM) を使用して Certificate System をインストールする場合は、一意の証明書のニックネームを使用する必要があります。

4. 必要に応じて、証明書の生成時に、RSA ではなく Elliptic Curve Cryptography (ECC) を使用するには、以下を実行します。

   a. **[DEFAULT]** セクションに以下のパラメーターを追加します。

```
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. **[CA]** セクションに以下のパラメーターを追加します。

```
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. **[CA]** セクションに以下のパラメーターを追加して、ECC プロファイルで RSA プロファイルを上書きします。

```
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.profile
```

他のサブシステムの設定

「サブシステムに依存しない設定」 に加え、下位 CA、KRA、OCSP、TKS、または TPS をインストールする必要があります。

1. 以下のエントリーを設定ファイルの **[DEFAULT]** セクションに追加します。

```
pki_client_database_password=password
```

2. TPS をインストールしている場合は、以下を行います。

a. 次のセクションに次のセクションを追加します。

```
[TPS]
pki_authdb_basedn=basedn_of_the_TPS_authentication_database
```

b. 必要に応じて、TPS が共有 CA インスタンスにすでにインストールされている KRA を利用してサーバー側のキー生成を使用するように設定するには、**[TPS]** セクションに次のエントリーを追加します。

```
pki_enable_server_side_keygen=True
```

7.7.4. インストール手順の起動

「インストールの最初ステップ用の設定ファイルの作成」の説明に従って設定ファイルを準備したら、インストールの最初のステップを開始します。

```
# pkispawn -f /root/config.txt -s subsystem --skip-configuration
```

subsystem は、**CA**、**KRA**、**OCSP**、**TKS** または **TPS** のいずれかに置き換えます。

7.7.5. インストール手順間の設定のカスタマイズ

「インストール手順の起動」 で説明されているインストール手順が修了したら、実際の設定を開始する前に、インスタンス固有の設定ファイルを手動で更新できます。このセクションでは、インストールの最初のステップと 2 番目のステップの間でカスタマイズできるものの特定の例を示します。

7.7.5.1. 証明書プロファイルの設定

多くの場合、サイトは特定の証明書登録プロファイルをカスタマイズしたり (たとえば、証明書のデフォルトの有効期間を変更したり)、一部のプロファイルを追加/削除したいと思うでしょう。オプションの完全なリストは、16章*証明書プロファイルの設定*を参照してください。

7.7.5.2. 署名監査ログの有効化

署名された監査ロギング機能を使用すると、承認されていないログ操作の検出が可能になります。詳細は、「署名監査ログの有効化および設定」 を参照してください。

7.7.5.3. 暗号一覧の更新

特定の状況では、管理者が暗号の一覧を更新する必要があります。以下に例を示します。

- Certificate System インスタンスのセキュリティーを保護するには
- Certificate System インスタンスをインストールし、特定の暗号のみをサポートする既存のサイトに追加するには

暗号一覧の更新に関する詳細は、『Red Hat Certificate System 管理ガイド』の該当するセクションを参照してください。

RSA 暗号化のデフォルトの FIPS モードが有効になっている暗号
デフォルトでは、Certificate System には、RSA 暗号化に以下の FIPS モードが有効な暗号が含まれます。

- 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 暗号化のデフォルトの FIPS モードが有効になっている暗号
デフォルトでは、Certificate System には、ECC (Elliptic Curve Cryptography) 暗号化に対して、以下の FIPS モードが有効な暗号が含まれます。

- 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

FIPS モードが有効になっているシステムで HSM を実行する際に必要な RSA 暗号
RSA で FIPS モードが有効になっているシステムに LunaSA または Hardware Security Module (HSM) のいずれかを使用した Certificate System をインストールする場合は、HSM ではサポートされていないため、以下の暗号を無効にします。

- 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. PKI コンソールタイムアウトの設定

PKI コンソールタイムアウトの設定に関する詳細は、「セッションのタイムアウト」 の該当するセクションを参照してください。

7.7.5.5. KRA の暗号化モードへの設定

Hardware Security Module (HSM) を使用している場合は、特定の状況で Key Recovery Authority (KRA) を暗号化モードに設定する必要があります。詳細は、「KRA の暗号化モードへの設定」 を参照してください。

7.7.5.6. OCSP の有効化

OCSP (Online Certificate Status Protocol) を有効にする方法は、「サブシステムの証明書失効チェックの有効化」を参照してください。

7.7.5.7. リクエスト番号とシリアル番号の範囲の設定

リクエストおよびシリアル番号の範囲の設定に関する詳細は、「リクエスト番号とシリアル番号の範囲の設定」を参照してください。

7.7.6. 設定手順の開始

「インストール手順間の設定のカスタマイズ」に従って設定ファイルをカスタマイズしたら、インストールの 2 つ目のステップを開始します。

```
# pkispawn -f /root/config.txt -s subsystem --skip-installation
```

subsystem は、**CA**、**KRA**、**OCSP**、**TKS** または **TPS** のいずれかに置き換えます。

設定手順に成功すると、**pkispawn** ユーティリティーにインストールの概要が表示されます。以下に例を示します。

```
==========================================================================
                    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. インストール後の設定

上記の手順を完了したら、インストール後の操作について「インストール後のタスク」に従ってください。

7.8. 外部 CA でのサブシステムの設定

7.8.1. 内部 CA と外部 CA の相違点

Red Hat Certificate System では、**pkispawn** ユーティリティーがサブシステム証明書署名要求 (CSR) を以前にインストールされた Certificate System に送信すると、上記の証明書が **pkispawn** で受信されて使用され、その CSR が送られる CA は **Internal CA** と呼ばれます。

一方、**External CA** は以下のいずれかになります。

- Certificate System の下位 CA の署名証明書を発行する RedHat Certificate System 以外の CA。
- CSR の直接送信を許可しない Red Hat Certificate System CA がインストールされていました。たとえば、ご使用の環境で、下位 CA、KRA、OCSP、TKS、または TPS からの CSR を必要とする場合、これは PKCS #10 以外の形式である必要があります。

7.8.2. 外部 CA を使用したサブシステムのインストール

本セクションでは、証明書が外部 CA により署名される下位 CA または他のサブシステムを設定する方法を説明します。

外部 CA インストール用の設定ファイルの準備
サブシステムを Certificate System またはスタンドアロンに統合するかどうかに応じて、設定ファイルを準備します。

- 既存の Certificate System インストールに統合されていて、外部 CA により署名された証明書を使用するサブシステムをインストールする場合は、以下を行います。

  1. サブシステムの設定ファイルを作成します。「インストールの最初ステップ用の設定ファイルの作成」 を参照してください。

  2. 各 [CA] セクション、[KRA] セクション、または [OCSP] セクションの設定ファイルに以下の設定を追加します。

     - CA インストールの場合:

       ```
       [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
       ```

     - KRA インストールの場合:

       ```
       [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
       ```

     - OCSP インストールの場合:

       ```
       [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
       ```

- 既存の Certificate System インストールに統合されていないスタンドアロンの KRA または OCSP をインストールする場合は、「スタンドアロン KRA または OCSP の設定」 に記載されている手順を実行します。

外部 CA を使用したサブシステムのインストールの開始

設定ファイルを使用してインストールを開始するには、以下を行います。

1. **pkispawn** ユーティリティーを使用してインストールを開始します。

```
# pkispawn -f /root/config.txt -s subsystem
```

subsystem は、インストールするサブシステム (**CA**、**KRA**または**OCSP**) に置き換えます。

このステップでは、セットアップでは、設定で指定されたファイルに CSR が保管されます。

2. 外部 CA に CSR を送信します。CA が対応する証明書を発行した後に続行します。

特定の環境では、外部 CA が Certificate System インスタンスでもある場合は、PKCS#10 形式の CSR を、CA に送信する前に ESP 形式に変換する必要があります。証明書の発行に関する詳細は、『Red Hat Certificate System 管理ガイド』『UMC を使用した証明書の発行』セクションを参照してください。

3. オプションで、インストールをカスタマイズします。詳細は、「インストール手順間の設定のカスタマイズ」 を参照してください。

4. 外部 CA が証明書を発行したら、デプロイメント設定ファイルを編集します。

a. ***pki_external_step_two*** を **True** に設定します。

```
pki_external_step_two=True
```

b. インストールするサブシステムに応じて、以下のパラメーターを追加します。

- CA の場合は、証明書ファイルへのパスを設定します。以下に例を示します。

```
pki_ca_signing_cert_path=/home/user_name/ca_signing.crt
```

指定したファイルに証明書チェーンを含む証明書が含まれていない場合は、さらに証明書チェーンファイルへのパスとニックネームを指定します。以下に例を示します。

```
pki_cert_chain_path=/home/user_name/cert_chain.p7b
pki_cert_chain_nickname=CA Signing Certificate
```

- KRA の場合には、証明書ファイルへのパスを設定します。以下に例を示します。

```
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
```

指定したファイルに証明書チェーンを含む証明書が含まれていない場合は、署名証明書ファイルと証明書チェーンファイルへのパスと、そのニックネームを指定します。以下に例を示します。

```
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
```

- OCSP の場合には、証明書ファイルへのパスを設定します。以下に例を示します。

```
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
```

指定したファイルに証明書チェーンを含む証明書が含まれていない場合は、署名証明書ファイルと証明書チェーンファイルへのパスと、そのニックネームを指定します。以下に例を示します。

```
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. 必要に応じて、設定ファイルをカスタマイズします。例については、「インストール手順間の設定のカスタマイズ」を参照してください。

6. 設定手順を開始します。

```
# pkispawn -f /root/config.txt -s subsystem
```

subsystem は、インストールするサブシステム (**CA**、**KRA**または**OCSP**) に置き換えます。

7.8.3. インストール後の設定

上記の手順を完了したら、インストール後の操作について 「インストール後のタスク」 に従ってください。

7.9. スタンドアロン KRA または OCSP の設定

本セクションでは、スタンドアロンの KRA および OCSP をインストールする方法を説明します。スタンドアロンインストールでは、Certificate System 以外の CA を使用して証明書を発行する柔軟性が提供されます。これは、インストール中に生成された CSR が CA に自動的に送信され、サブシステムにインポートされないためです。また、スタンドアロンモードにインストールされた KRA または OCSP は CA のセキュリティードメインの一部ではなく、キーのアーカイブ CA のコネクターは設定されません。

スタンドアロン KRA または OCSP をインストールするには、以下を行います。

1. 以下の内容で、**/root/config.txt** などの設定ファイルを作成します。

```
[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. スタンドアロンの KRA の場合は、設定ファイルに以下のセクションを追加します。

```
[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. スタンドアロン OCSP の場合は、設定ファイルに以下のセクションを追加します。

```
[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. 同じホストで実行している Directory Server への LDAPS 接続を使用するには、設定ファイルの **DEFAULT** セクションに以下のパラメーターを追加します。

```
pki_ds_secure_connection=True
pki_ds_secure_connection_ca_pem_file=path_to_CA_or_self-signed_certificate
```

注記

セキュリティー上の理由から、Red Hat は、Directory Server への暗号化された接続を使用することを推奨します。

Directory Server で自己署名証明書を使用する場合は、以下のコマンドを使用して Directory Server の Network Security Services (NSS) データベースからエクスポートします。

```
# certutil -L -d /etc/dirsrv/slapd-instance_name/ \
    -n "server-cert" -a -o /root/ds.crt
```

5. 「外部 CA を使用したサブシステムのインストールの開始」 に記載されている手順に進みます。

7.10. インストール後のタスク

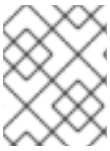
pkispawn ユーティリティーを使用したインストールが完了したら、サイトの設定に応じて、さらに設定をカスタマイズすることができます。詳細は、パートIII「Certificate System の設定」で説明してください。

本セクションでは、デプロイメントのセキュリティーを強化するために、パートIII「Certificate System の設定」 からの操作の一覧を紹介します。

7.10.1. RHCS の日付/時刻の設定

RHCS を実行するための時間を正しく設定しておくことが重要です。『Red Hat Certificate System 管理ガイド』の『日時の設定』セクションを参照してください。

7.10.2. 一時的な証明書の置き換え



注記

本セクションでは、ルート CA をインストールして実行する必要があります。以下の手順は、インストールの完了後に実行します。

以下の手順では、一時的な自己署名 Directory Server 証明書を新しくインストールした CA によって発行された永続的な Directory Server 証明書に置き換えるプロセス、または古い Directory Server 証明書を新しいものに置き換えるプロセスを説明します。

1. 新しい Directory Server サーバー証明書を取得します。CA が署名した新規証明書の要求を送信します。CMC メソッドを使用して新しい Directory Server 証明書を取得するには、『Red Hat Certificate System 管理ガイド』 の 『システム証明書およびサーバー証明書の取得』 を参照してください。

   上記のセクションで、TLS サーバー証明書の作成に関するガイダンスに従ってください。証明書を作成したら、Directory Server 証明書データベースにインポートし直す必要があります。

2. NSS データベースにアクセスする前に、Directory Server インスタンスを停止します。

```
# systemctl stop dirsrv@instance_name
```

3. 古い Directory Server 証明書を削除します。

```
# certutil -F -d /etc/dirsrv/slapd-instance_name -f
/etc/dirsrv/slapd-instance_name/password.txt -n "DS Certificate"
```

4. 先ほどダウンロードした CA 証明書をインポートします。

```
# 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. 先にダウンロードした新しい Directory Server 証明書をインポートします。

```
# 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
```

「HSM への証明書のインポート」 も併せて参照してください。

6. Directory Server インスタンスを停止します。

```
# systemctl start dirsrv@instance_name
```

7. PKI CA から古い Directory Server 証明書を削除します。

   a. Certificate System インスタンスを停止します。

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

   b. 証明書を削除します。

   ```
   # certutil -D -d /var/lib/pki/instance_name/alias/ -n "DS Certificate"
   ```

   c. Certificate System インスタンスを起動します。

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

8. 新しい Directory Server 証明書が、NSS データベースにインストールされている CA で署名されていることを確認します。

   a. Directory Server 証明書の表示

   ```
   $ certutil -L -d /etc/dirsrv/slapd-instance_name -n "DS Certificate"

     Issuer: "CN=CA Signing Certificate,O=EXAMPLE"
     Subject: "CN=server.example.com"
   ```

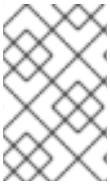
   b. 古い Directory Server 証明書が PKI NSS データベースに存在しなくなったことを確認します。

   ```
   $ certutil -L -d /var/lib/pki/instance_name/alias
   ```

   c. Certificate System が、新しい証明書を使用して Directory Server に接続できることを確認します。

   ```
   $ pki cert-find
   ```

7.10.3. TLS クライアント認証の有効化



注記

本セクションでは、ルート CA をインストールして実行する必要があります。一時的な LDAP サーバー証明書を使用している場合は、最初に 「一時的な証明書の置き換え」 で置き換えます。以下の手順は、インストールの完了後に実行します。

TLS クライアント認証を有効にすることを選択した場合は、CS サブシステムのインストール時に基本的な TLS サーバー認証を設定して実行すると、逆戻りして、特定のサブシステムから LDAP サーバーへのクライアント認証の有効化を試みることができます。

クライアント認証の設定には 2 つの部分があります。最初の部分は、TLS の相互認証を必要とする LDAP ディレクトリーを設定します。この手順は、『Red Hat Directory Server 管理ガイド』 の 証明書ベースのクライアント認証の使用 を参照してください。

以下の点に注意してください。

- **pkispawn** は、すでに内部ディレクトリーサーバー上に **pkidbuser** を自動的に作成しており、そこにはアウトバウンド TLS クライアント認証に使用される CS インスタンスのサブシステム証明書 (たとえば **subsystemCert cert-pki-ca**) がユーザーエントリーに格納されています。したがって、TLS クライアント認証用に別の LDAP ユーザーまたは別の証明書を作成する必要はありません。

- **/etc/dirsrv/slapd-instance_name/certmap.conf** のコンテンツを作成する場合は、以下の形式を使用します。

  ```
  certmap rhcs <certificate issuer DN>
    rhcs:CmapLdapAttr seeAlso
    rhcs:verifyCert on
  ```

  以下に例を示します。

  ```
  certmap rhcs CN=CA Signing Certificate,OU=pki-tomcat-ca,O=pki-tomcat-ca-SD
    rhcs:CmapLdapAttr seeAlso
    rhcs:verifyCert on
  ```

- 設定後に、Directory Server を再起動します。

2 番目の部分は、Red Hat Certificate System インスタンスに設定を追加して、TLS 相互認証を使用して内部 LDAP サーバーと通信するために使用するポートと証明書を認識できるようにすることです。これには、***<instance directory>*/*<subsystem type>*/conf/CS.cfg** にある RHCS インスタンスの **CS.cfg** ファイルを編集する必要があります。たとえば、**/var/lib/pki/*instance_name*/ca/conf/CS.cfg** です。

CS.cfg で、RHCS インスタンスのサブシステム証明書のニックネームを **internaldb.ldapauth.clientCertNickname** に追加し、未使用のエントリーを 2 つ削除します。

```
internaldb.ldapauth.bindDN
  internaldb.ldapauth.bindPWPrompt
```

以下に例を示します。

```
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
```

インストール後のステップの最後に CS インスタンスを再起動します。

注記

特定の LDAP インスタンスに一致するように、***internaldb.basedn*** パラメーターおよび ***internaldb.database*** パラメーターを設定する必要があります。

コンプライアンスのために、**internaldb.ldapauth.authtype=SslClientAuth** および **internaldb.ldapconn.secureConn=true** を設定する必要があり、***internaldb.ldapauth.clientCertNickname*** の値は、NSS DB で LDAP に対して認証するには、TLS クライアント証明書のニックネームと一致させる必要があります。

その他の値はすべて、環境または可用性の要件を反映するために必要に応じて変更できます。

7.10.4. セッションタイムアウトの設定

さまざまなタイムアウト設定がシステムに存在し、終了前に TLS セッションがアイドル状態の意地する時間に影響を与える可能性があります。詳細は、「セッションのタイムアウト」 を参照してください。

7.10.5. CRL または証明書の発行

CRL 公開は、OCSP サービスを提供する際に重要です。証明書の公開は任意になりますが、多くの場合、サイトで必要になります。詳細は、『Red Hat Certificate System 管理ガイド』 の 『証明書および CRL の公開』 セクションを参照してください。

7.10.6. 証明書登録プロファイル (CA) の設定

RHCS には、証明書登録プロファイルのカスタマイズを可能にするリッチプロファイルフレームワークがあります。システムで使用できるデフォルトのプロファイルを有効またはにしたり、既存のプロファイルを変更したり、独自のプロファイルを作成することが非常に一般的です。詳細は、16章*証明書プロファイルの設定* を参照してください。

7.10.7. アクセスバナーの有効化

ユーザーインターフェイスバナーを有効にするには、「アクセスバナーの有効化」を参照してください。

7.10.8. Watchdog サービスの有効化

ウォッチドッグ (**nuxwdog**) サービスは、セキュアなシステムパスワード管理を提供します。詳細は、「Watchdog サービスの有効化」 を参照してください。

7.10.9. CMC 登録および失効 (CA) の設定

証明書の登録と失効は CMC で行うことができます。

- CMC Shared Token Feature の有効化に関する詳細は、「CMC 共有シークレット機能の有効化」を参照してください。

- **PopLinkWittness** 機能を有効にする方法は、「**PopLinkWittnessV2** 機能の有効化」 を参照してください。
- Web ユーザーインターフェイスの **CMCRevoke** を有効にする方法は、「Web ユーザーインターフェイスの CMCRevoke の有効化」 を参照してください。

7.10.10. Java コンソールの TLS クライアント認証

Certificate System 管理者が Java コンソールにログインするときにユーザー TLS クライアント証明書を提示するように要求するには、「TLS クライアント証明書認証を使用するための **pkiconsole** 要件の設定」を参照してください。

注記

pkiconsole が非推奨になりました。

7.10.11. ロールユーザーの作成

ブートストラップユーザーを削除できるように、実際のロールユーザーを作成します。

ユーザーを作成して異なる特権ロールに割り当てて、Certificate System を管理するには、19章*ロールユーザーの作成*を参照してください。

7.10.12. Bootstrap ユーザーの削除

実際のロールユーザーが作成されると、インストール時に自動的に作成されたブートストラップユーザーは不要になりました。このアカウントを削除するには、個人個人に割り当てられている新しい管理者アカウントを作成したことを確認してから、20章*Bootstrap ユーザーの削除* を参照してください。

7.10.13. マルチロールサポートの無効化

ブートストラップユーザーが削除された後にマルチロールサポートを無効にするには、「マルチロールサポートの無効化」を参照してください。

7.10.14. KRA の設定

7.10.14.1. KRA (Key Recovery Authority) に複数のエージェント承認の要件の追加

複数の KRA エージェントがキーリカバリーを承認するための要件を設定するには、『Red Hat Certificate System 管理ガイド』の『コマンドラインでのエージェント承認キーリカバリーの設定』を参照してください。

7.10.14.2. KRA 暗号化設定の設定

キーの暗号化/ラップアルゴリズムを設定する場合は、「KRA 操作の暗号化」を参照してください。

7.10.15. ユーザーインターフェイスを使用するようにユーザーを設定

ユーザーが承認されたユーザーインターフェイスを使用する前に、初期化を行う必要があります。ユーザー (管理ロールなど) は、ユーザーインターフェイスにアクセスするためにクライアントを設定するために必要です。『Red Hat Certificate System 管理ガイド』の『クライアント NSS データベースの初期化』セクションを参照してください。

第8章 サブシステムセキュリティーデータベース用のハードウェアセキュリティーモジュールの使用

サブシステムインスタンスは、**セキュリティーモジュール** や **トークン** と呼ばれるキーストアにキー情報を生成し、保存します。内部 NSS トークンを使用するか、別の暗号化デバイス (ハードウェアトークン) を使用してキーを生成および保存するようにサブシステムインスタンスを設定できます。

8.1. HSM を使用した CERTIFICATE SYSTEM のインストール

HSM を使用して Certificate System をインストールする場合は、**pkispawn** ユーティリティーに渡す設定ファイルで以下のパラメーターを使用します。

```
[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. サブシステムでのハードウェアセキュリティーモジュールの使用

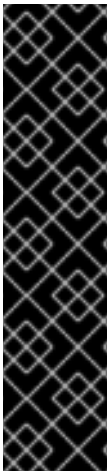
Certificate System は、デフォルトで nCipher nShield Connect XC ハードウェアセキュリティーモジュール (HSM) および Thales Luna HSM をサポートします。Certificate System がサポートする HSM は、PKCS #11 ライブラリーモジュールが指定されているインストールパスにある場合は、インストールの事前設定段階で **modutil** を使用して **pkcs11.txt** データベースに自動的に追加されます。

重要

特定のデプロイメントでは、FIPS モードを使用するように HSM を設定する必要があります。

8.2.1. HSM での FIPS モードの有効化

HSM で FIPS モードを有効にするには、特定の手順については、HSM ベンダーのドキュメントを参照してください。



重要

nCipher HSM

nCipher HSM では、FIPS モードが Security World を生成する場合にのみ有効にできます。これは後で変更することはできません。Security World を生成するにはさまざまな方法がありますが、常に **new-world** コマンドを使用することが推奨されます。FIPS 準拠の Security World を生成する方法は、nCipher HSM ベンダーのドキュメントを参照してください。

LunaSA HSM

同様に、Luna HSM で FIPS モードを有効にするには、初期設定時に行う必要があります。これは、このポリシーを変更すると、セキュリティー対策として HSM がゼロになるためです。詳細は、Luna HSM ベンダーのドキュメントを参照してください。

8.2.2. FIPS モードが HSM で有効になっているかどうかの確認

本セクションでは、特定の HSM に対して FIPS モードが有効になっているかどうかを確認する方法を説明します。その他の HSM は、ハードウェアの製造元のドキュメントを参照してください。

8.2.2.1. FIPS モードが nCipher HSM で有効にされているかどうかの確認

注記

完全な手順については、HSM ベンダーのドキュメントを参照してください。

FIPS モードが nCipher HSM で有効になっているかどうかを確認するには、次のコマンドを実行します。

```
# /opt/nfast/bin/nfkminfo
```

古いバージョンのソフトウェアでは、**StrictFIPS140** が state フラグに一覧表示されると、FIPS モードが有効になります。ただし、新しいバージョンでは、新しい **mode** の行を確認して **fips1402level3** を検索することが推奨されます。すべてのケースで、**nfkminfo** 出力には **hkfips** キーも存在しているはずです。

8.2.2.2. FIPS モードが Luna SA HSM で有効にされているかどうかの確認

注記

完全な手順については、HSM ベンダーのドキュメントを参照してください。

FIPS モードが Luna SA HSM で有効になっているかどうかを確認するには、次を実行します。

1. **lunash** 管理コンソールを開きます。
2. **hsm show** コマンドを使用して、出力に **The HSM is in FIPS 140-2 approved operation mode.** の文字が含まれていることを確認します。

```
lunash:> hsm show
...
   FIPS 140-2 Operation:
```

```
    ====================
    The HSM is in FIPS 140-2 approved operation mode.
...
```

8.2.3. サブシステムの HSM エントリーの追加および管理

インストール時に、**pkispawn** コマンドに渡された設定ファイルに適切な HSM 固有のパラメーターが設定されているデフォルトトークンとして HSM が選択された場合は、以下のパラメーターが、HSM パスワードの **/var/lib/pki/*instance_name*/conf/password.conf** ファイルに追加されます。

```
hardware-HSM_token_name=HSM_token_password
```

8.2.4. HSM 用の SELinux の設定

Hardware Security Modules (HSM) で Certificate System をインストールし、SELinux が **enforcing** モードで実行している場合は、Certificate System をインストールする前に、特定の HSM を手動で SELinux 設定を更新する必要があります。

以下のセクションでは、対応している HSM に必要なアクションを説明します。

nCipher nShield Connect XC

HSM をインストールし、Certificate System をインストールする前に、以下を行います。

1. **/opt/nfast/** ディレクトリーのファイルのコンテキストをリセットします。

```
# restorecon -R /opt/nfast/
```

2. **nfast** ソフトウェアを再起動します。

```
# /opt/nfast/sbin/init.d-ncipher restart
```

Thales Luna HSM

Certificate System をインストールする前に、SELinux 関連のアクションは必要ありません。

8.2.5. nCipher nShield Connect XC HSM を使用するサブシステムのインストール

nCipher nShield Connect XC HSM を使用するサブシステムインスタンスをインストールするには、以下の手順に従います。

1. 特定のデプロイメントに対応するオーバーライドファイルを準備します。以下の **default_hms.txt** ファイルは、このような上書きファイルの例になります。

注記

このファイルは、nCipher HSM 上書き設定ファイルのサンプルとしてのみ提供されます。その他の多くの値は、デフォルトのハッシュアルゴリズムを含む上書きできます。また、**pkispawn** コマンドラインで指定されたサブシステムの呼び出しに応じて、[CA] セクション、[KRA] セクション、[OCSP] セクション、[TKS] セクション、または [TPS] セクションの 1 つだけが使用されます。

例8.1 nCipher HSM で使用するオーバーライドファイルのサンプル

```
############################################################################
#######
############################################################################
#######
############################################################################
#######
##                                                        ##
## 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. 「2 ステップインストール」に記載されているように、設定ファイルを使用します。

- # 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. HSM に次の証明書が含まれていることを確認します。

```
$ 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. Thales Luna HSM を使用するサブシステムのインストール

Thales Luna HSM を使用するサブシステムインスタンスをインストールするには、「nCipher nShield Connect XC HSM を使用するサブシステムのインストール」 に詳述されている手順に従います。オーバーライドファイルは、例8.1「nCipher HSM で使用するオーバーライドファイルのサンプル」 で提供されるサンプルと似ていますが、特定のデプロイメントに関連する値とは異なります。以下の例では、前述の nCipher 例で提供された [DEFAULT]、[Tomcat]、[CA]、[KRA]、[OCSP]、[TKS]、[TPS] の各セクションで使用される nCipher オーバーライドファイルのヘッダーの代わりに使用される LunaSA ヘッダーのサンプルを提供します。

例8.2 LunaSA オーバーライドファイルヘッダーの例

```
#############################################################################

#############################################################################

#############################################################################

##                                                         ##
## 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. ハードウェアセキュリティーモジュールでのキーのバックアップ

HSM に保存されている鍵と証明書を、**.p12** ファイルにエクスポートすることができません。このようなインスタンスをバックアップする必要がある場合には、HSM の製造元に連絡してサポートしてください。

8.4. HSM を使用したクローンサブシステムのインストール

通常、クローンサブシステムは、マスターサブシステムのシステムキーを含む PKCS #12 ファイルを使用して作成されます。このファイルは、インストールに使用する設定ファイルで ***pki_backup_keys*** を **True** に設定して ***pki_backup_password*** オプションの定義値と一緒にインストールするか、または

PKCS12Export ツールを使用して鍵をエクスポートすることで、マスターサブシステムのインストール中に生成されます。

HSM を使用する場合は、HSM からキーを取得できないため、**PKCS12Export** で、PKCS #12 ファイルを生成することはできません。代わりに、クローンサブシステムは、マスターサブシステムが使用する HSM を指すようにして、同じキーにアクセスできるようにする必要があります。別の HSM を使用する場合は、HSM ベンダーが提供するユーティリティーを使用して、この HSM に鍵のクローンを作成します。**pkispawn** ユーティリティーを実行する前に、master サブシステムのキーへのアクセスを提供する必要があります。

Certificate System は、HSM を使用してインストールする際に PKCS #12 を使用しないため、PKCS #12 ファイルの場所とそのパスワードについて、設定ファイルでパラメーターを設定する必要はありません。次の例では、CA クローンの生成に必要な設定パラメーターと、それぞれの PKI 設定ファイルの CA セクションのオプションを示します。

非 HSM CA のクローンを生成するには、以下を行います。

```
[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>
```

HSM を使用して CA クローンを生成するには、以下を行います。

```
[CA]
pki_clone=True
pki_clone_replicate_schema=True
pki_clone_uri=https://<master_ca_host_name>:<master_ca_https_port>
```

注記

マスターサブシステムが同じ証明書と鍵をそのクローンと共有できるようにするには、HSM が共有モードにあり、すべてのサブシステムからアクセスできるネットワーク上にある必要があります。

8.5. トークンの表示

Certificate System インスタンスに現在インストールされているトークンの一覧を表示するには、**modutil** ユーティリティーを使用します。

1. インスタンスの **alias** ディレクトリーに移動します。以下に例を示します。

```
# cd /var/lib/pki/pki-tomcat/alias
```

2. インストールされている PKCS #11 モジュールに関する情報と、**modutil** ツールを使用して、対応するトークンに関する情報を表示します。

```
# modutil -dbdir . -nocertdb -list
```

8.6. トークンの検出

Certificate System でトークンを検出できるかどうかを確認するには、**TokenInfo** ユーティリティーを使用し、サブシステムインスタンスの **alias** ディレクトリーを参照します。これは、Certificate System パッケージのインストール後に利用できる Certificate System ツールです。

以下に例を示します。

```
# TokenInfo /var/lib/pki/pki-tomcat/alias
```

このユーティリティーは、Certificate System にインストールされたトークンだけでなく、Certificate System で検出できるすべてのトークンを返します。

第9章 ECC システム証明書を使用するインスタンスのインストール

楕円曲線暗号 (ECC) は、RSA スタイルの暗号化よりも優先される場合があります。これは、使用するキーの長さがはるかに短く、証明書の生成が高速になるためです。ECC 対応の CA は、ECC 署名証明書を使用して、RSA 証明書と ECC 証明書の両方を発行できます。

Certificate System には ECC 機能に対するネイティブサポートが含まれています。このサポートはデフォルトで NSS 3.16 以降で有効になっています。ハードウェアセキュリティーモジュール (HSM) などのサードパーティーの PKCS #11 モジュールを読み込み、使用することも可能です。ECC モジュールを使用するには、サブシステムインスタンスを設定する前に読み込む必要があります。

重要

サードパーティーの ECC モジュールには SELinux ポリシーが設定されている必要があります。もしくは、このモジュールが機能するには、SELinux を **enforcing** モードから **permissive** モードに変更する必要があります。それ以外の場合は、ECC モジュールを必要とするサブシステム操作は失敗します。

9.1. サードパーティーの ECC モジュールの読み込み

サードパーティーの ECC モジュールの読み込みは、Certificate System でサポートされる HSM の読み込みと同じ原則に従います。これは 8章 *サブシステムセキュリティーデータベース用のハードウェアセキュリティーモジュールの使用*で説明されています。詳細は、本章を参照してください。

9.2. HSM での ECC の使用

Certificate System がサポートする HSM は、独自のネイティブ ECC モジュールに対応します。ECC システム証明書を使用するインスタンスを作成するには、以下を行います。

1. 製造元の指示に従って HSM をセットアップします。複数のホストが HSM を共有している場合は、必要に応じて、サイトポリシーで許可されている場合は、すべてのホストが同じパーティションにアクセスできることを確認してください。

2. **pkispawn** ユーティリティー設定ファイルに必要なパラメーターを定義し、**pkispawn** を実行します。たとえば、設定ファイルが **ecc.inf** の場合に、Certificate System が ECC CA を作成するように設定するには、次を行います。

    1. **ecc.inf** を編集して、適切な設定を指定します。設定ファイルの例は、pkispawn(8) の man ページを参照してください。

    2. **ecc.inf** に対して **pkispawn** を実行します。

    ```
    $ script -c 'pkispawn -s CA -f /root/pki/ecc.inf -vvv'
    ```

第10章 サブシステムのクローン作成

新しいサブシステムインスタンスが最初に設定されている場合、Red Hat Certificate System は Certificate System の高可用性のためにサブシステムをクローンするか、または複製できます。クローンが作成されたインスタンスは、単一障害点を回避するために異なるマシンで実行され、それらのデータベースはレプリケーションを通じて同期されます。

マスター CA およびそのクローンは機能的に同一で、シリアル番号の割り当てと CRL 生成でのみ異なります。したがって、本章はマスターまたはそのクローンを *複製 CA* として参照します。

10.1. ソフトウェアデータベースからのサブシステムキーのバックアップ

理想的には、インスタンスの初回作成時に、マスターインスタンスの鍵がバックアップされます。キーがバックアップされていない場合や、バックアップファイルが失われた場合は、**PKCS12Export**ユーティリティーを使用してサブシステムインスタンスの内部ソフトウェアデータベースからキーを抽出することができます。以下に例を示します。

```
PKCS12Export -debug -d /var/lib/pki/instance_name/alias -w p12pwd.txt -p internal.txt -o master.p12
```

次に、クローンインスタンス設定で使用するクローンマシンに PKCS #12 ファイルをコピーします。詳細は、「クローンおよびキーストア」を参照してください。

注記

キーは HSM からエクスポートできません。ただし、標準的なデプロイメントでは、マスターと同じ HSM を使用してクローンインスタンスがインストールされている限り、HSM はネットワークアクセスをサポートします。両方のインスタンスが同じキーストアを使用する場合、このキーは基本的にクローンで利用可能になります。

HSM からキーをバックアップする必要がある場合は、HSM の製造元に問い合わせてください。

10.2. CA のクローン作成

1. マスター CA を設定し、キーのバックアップを作成します。
2. マスター CA の **CS.cfg** ファイルで、**ca.listenToCloneModifications** パラメーターを追加して、マスター CA がレプリケーションデータベースの変更を監視できるようにします。

   ```
   ca.listenToCloneModifications=true
   ```

3. クローンサブシステムインスタンスを作成します。

   CA サブシステムのクローン作成時に **pkispawn** で必要な設定ファイルの例は、pkispawn(8) の man ページの **Installing a CA clone** セクションおよび **Installing a CA clone on the same host** セクションを参照してください。

4. クローンが使用する Directory Server インスタンスを再起動します。

   ```
   # systemctl restart pki-tomcatd@kra-clone-ds-instance.service
   ```

注記

Directory Server を再起動すると、更新されたスキーマが再読み込みされます。これは、パフォーマンスを適切に行うために必要です。

5. クローンインスタンスを再起動します。

```
# pki-server restart instance_name
```

クローンの設定後に、テストを実行して、master-clone 関係が機能していることを確認します。

1. クローン作成された CA から証明書を要求します。
2. 要求を承認します。
3. ブラウザーに証明書をダウンロードします。
4. 証明書を取り消します。
5. マスター CA の CRL で取り消された証明書を確認します。マスター Certificate Manager のエージェントサービスページで、**Update Certificate Revocation List** をクリックします。一覧で CRL を検索します。

   CRL には、クローン作成された Certificate Manager が失効した証明書が表示されます。証明書が一覧にない場合は、ログを確認して問題を解決します。

10.3. クローン後の CA-KRA コネクター情報の更新

「カスタム設定およびクローン」で説明されているとおり、クローンの作成 **後** に行われる場合には、クローンインスタンスで設定情報が更新されません。同様に、クローンに加えられた変更はマスターインスタンスにはコピーされません。

クローン CA の作成後に新しい KRA をインストールまたはクローンした場合、クローン CA には設定に新しい KRA コネクター情報がありません。つまり、クローン CA はアーカイブされたリクエストを KRA に送信しないことを意味します。

新しい KRA が作成またはクローンされるたびに、そのコネクター情報をデプロイメントで **クローン** 作成されたすべての CA にコピーします。これには、**pki ca-kraconnector-add** コマンドを使用します。

手動で行う必要がある場合は、次の手順を実行します。

1. マスタークローンマシンで、マスター CA の **CS.cfg** ファイルを開き、新しい KRA コネクターの ***ca.connector.KRA.**** 行をすべてコピーします。

```
[root@master ~]# vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

2. クローン CA インスタンスを停止します。以下に例を示します。

```
[root@clone-ca ~]# pki-server stop instance_name
```

3. クローン CA の **CS.cfg** ファイルを開きます。

```
[root@clone-ca ~]# vim /var/lib/pki/instance_name/ca/conf/CS.cfg
```

4. 新しい KRA インスタンスまたはクローンのコネクター情報をコピーします。

```
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. クローン CA を起動します。

```
[root@clone-ca ~]# pki-server start instance_name
```

10.4. OCSP サブシステムのクローン作成

1. マスター OCSP を設定し、キーをバックアップします。
2. マスター OCSP の **CS.cfg** ファイルで、**OCSP.Responder.store.defStore.refreshInSec** パラメーターを 21600 以外の番号に設定します。21600 はクローンの設定になります。

```
# vim /etc/instance_name/CS.cfg

OCSP.Responder.store.defStore.refreshInSec=15000
```

3. **pkispawn** ユーティリティーを使用して、クローンサブシステムインスタンスを作成します。

   OCSP サブシステムのクローン時に **pkispawn** で必要な設定ファイルの例は、pkispawn(8) の man ページを参照してください。

4. クローンが使用する Directory Server インスタンスを再起動します。

```
# systemctl dirsrv@instance_name.service
```

注記

Directory Server を再起動すると、更新されたスキーマが再読み込みされます。これは、パフォーマンスを適切に行うために必要です。

5. クローンインスタンスを再起動します。

```
# pki-server restart instance_name
```

クローンの設定後に、テストを実行して、master-clone 関係が機能していることを確認します。

1. CRL がマスター OCSP に公開されるように、マスター CA で OCSP 公開を設定します。
2. CRL が正常に公開されたら、エージェントページのマスターおよびクローン作成された OCSP の **List Certificate Authority** リンクの両方を確認します。リストは同一でなければなりません。
3. **OCSPClient** ツールを使用して、マスターおよびクローン作成された Online Certificate Status Manager に OCSP 要求を送信します。このツールは、両方のマネージャーから同じ OCSP 応答を受け取る必要があります。

10.5. KRA サブシステムのクローン作成

1. master サブシステムを設定し、キーをバックアップします。
2. **pkispawn** ユーティリティーを使用して、クローンサブシステムインスタンスを作成します。

```
$ pkispawn -s <subsystem> -f myconfig.txt
```

KRA サブシステムのクローンを作成するときに **pkispawn** に必要な設定ファイルの例:

```
[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. クローンが使用する Directory Server インスタンスを再起動します。

```
# systemctl dirsrv@instance_name.service
```

注記

Directory Server を再起動すると、更新されたスキーマが再読み込みされます。これは、パフォーマンスを適切に行うために必要です。

4. クローンインスタンスを再起動します。

```
# pki-server restart instance_name
```

クローンの設定後に、テストを実行して、マスターとクローンの関係が機能していることを確認します。

1. KRA エージェントのページに移動します。
2. **List Requests** をクリックします。
3. リクエストタイプおよびステータスの **Show all requests** の表示を選択します。
4. **送信** をクリックします。

5. クローン作成された KRA とマスター KRA の結果を比較します。結果は同一であることが見なされます。

10.6. TKS サブシステムのクローン作成

1. master サブシステムを設定し、キーをバックアップします。
2. **pkispawn** ユーティリティーを使用して、クローンサブシステムインスタンスを作成します。

```
$ pkispawn -s <subsystem> -f myconfig.txt
```

TKS サブシステムのクローンを作成するときに **pkispawn** に必要な設定ファイルの例:

```
[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. クローンインスタンスを再起動します。

```
# pki-server restart instance_name
```

TKS の場合は、スマートカードを登録してから **ldapsearch** を実行して、同じキー情報が両方のデータベースに含まれていることを確認します。

10.7. マスターとクローンの変換

CRL を生成する 1 つのアクティブな CA のみが、同じトポロジー内に存在できます。同様に、CRL を受信する OCSP は 1 つだけ同じトポロジー内に存在できます。そのため、クローンはいくつでも存在できますが、CA および OCSP 用に設定されたマスターは 1 つだけです。

KRA と TKS の場合、マスターとクローンの間に設定の違いはありませんが、CA と OCSP にはいくつかの設定の違いがあります。これは、マスターがオフラインになった場合、障害やメンテナンスのため、または PKI 内のサブシステムの機能を変更するために、既存のマスターをクローンに再設定し、クローンの 1 つをマスターに昇格させなければならないことを意味します。

10.7.1. CA クローンおよびマスターの変換

1. マスター CA が実行中の場合は停止します。

2. 既存のマスター CA 設定ディレクトリーを開きます。

```
# cd /var/lib/pki/instance_name/ca/conf
```

3. マスターの **CS.cfg** ファイルを編集し、CRL およびメンテナンススレッド設定をクローンとして設定します。

- データベースメンテナンススレッドの制御を無効にします。

```
ca.certStatusUpdateInterval=0
```

- データベースのレプリケーション変更の監視を無効にします。

```
ca.listenToCloneModifications=false
```

- CRL キャッシュのメンテナンスを無効にします。

```
ca.crl.IssuingPointId.enableCRLCache=false
```

- CRL 生成を無効にします。

```
ca.crl.IssuingPointId.enableCRLUpdates=false
```

- CRL 要求を新規マスターにリダイレクトするように CA を設定します。

```
master.ca.agent.host=new_master_hostname
master.ca.agent.port=new_master_port
```

4. クローン作成された CA サーバーを停止します。

```
# pki-server stop instance_name
```

5. クローン CA の設定ディレクトリーを開きます。

```
# cd /etc/instance_name
```

6. **CS.cfg** ファイルを編集して、クローンを新規マスターとして設定します。

a. **ca.crl.** 接頭辞で開始する各行を削除します。

b. **ca.crl.** で始まる各行を、以前のマスター CA **CS.cfg** ファイルから、クローン作成された CA の **CS.cfg** ファイルにコピーします。

c. データベースメンテナンススレッドの制御を有効にします。マスター CA のデフォルト値は **600** です。

```
ca.certStatusUpdateInterval=600
```

d. データベースのレプリケーションのモニタリングを有効にします。

```
ca.listenToCloneModifications=true
```

e. CRL キャッシュのメンテナンスを有効にします。

```
ca.crl.IssuingPointId.enableCRLCache=true
```

f. CRL 生成を有効にします。

```
ca.crl.IssuingPointId.enableCRLUpdates=true
```

g. CRL 生成要求のリダイレクト設定を無効にします。

```
master.ca.agent.host=hostname
master.ca.agent.port=port number
```

7. 新規マスター CA サーバーを起動します。

```
# pki-server start instance_name
```

10.7.2. OCSP クローンの変換

1. OCSP マスターが稼働している場合は停止します。
2. 既存のマスター OCSP 設定ディレクトリーを開きます。

```
# cd /etc/instance_name
```

3. **CS.cfg** を編集し、**OCSP.Responder.store.defStore.refreshInSec** パラメーターを **21600** にリセットします。

```
OCSP.Responder.store.defStore.refreshInSec=21600
```

4. オンラインのクローン作成された OCSP サーバーを停止します。

```
# pki-server stop instance_name
```

5. クローン作成された OCSP レスポンダーの設定ディレクトリーを開きます。

```
# cd /etc/instance_name
```

6. **CS.cfg** ファイルを開き、**OCSP.Responder.store.defStore.refreshInSec** パラメーターを削除するか、その値をゼロ以外の数字に変更します。

```
OCSP.Responder.store.defStore.refreshInSec=15000
```

7. 新規マスター OCSP レスポンダーサーバーを起動します。

```
# pki-server start instance_name
```

10.8. 再キーが設定された CA のクローン作成

証明書の期限が切れたら、置き換える必要があります。これは、元のキーペアを再利用して新しい証明書を生成する証明書を更新するか、新しいキーペアと証明書を生成することによって実行できます。2つ目の方法は **再キーイング** と呼ばれます。

CA のキーが再設定されると、新しいキーペアが証明書データベースに保存されます。これらは通常の操作のキー参照です。ただし、サブシステムを複製する場合、複製プロセスは、**CS.cfg** 設定ファイルに格納されている CA 秘密鍵 ID をチェックし、証明書データベースの鍵が変更されても、これらの鍵 ID は更新されません。

CA のキーが再設定された後、管理者がそのクローンを作成しようとすると、クローンされた CA は、キーが再設定された証明書の証明書の生成に失敗し、次のエラーとともにエラーログに表示されます。

```
CertUtil::createSelfSignedCert() - CA private key is null!
```

再登録した CA のクローンを作成するには、以下を実行します。

1. **CS.cfg** ファイルで、秘密鍵 ID をすべて検索します。

```
# 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. NSS データベースに保存されている現在の秘密鍵 ID をすべて出力し、それを **CS.cfg** ファイルに保存されている秘密鍵 ID と比較します。

```
# 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
```

この例では、監査署名キーのみが同じで、他は変更になりました。

3. ステップ 2 でキーを取得し、これらを署名なし値 (**certutil** を返すもの) から署名済み Java BigInteger (Certificate System データベースに格納) に変換します。

これは、calculator または 例10.1「certutil から BigInteger 変換プログラム」 でスクリプトを使用して実行できます。

4. 新しいキーの値を **CS.cfg** ファイルにコピーします。

```
# 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. 「CA のクローン作成」の説明に従って CA のクローンを作成します。

例10.1 certutil から BigInteger 変換プログラム

この Java プログラムは、**certutil** からのキー出力を必要な BigInteger 形式に変換できます。

これを **Test.java** などの **.java** ファイルとして保存します。

```
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));
  }
}
```

次に、ファイルを再コンパイルします。

```
# javac Test.java
```

第11章 PKI ACME RESPONDER の設定

本章では、CA サブシステムがすでにある PKI サーバーの ACME レスポンダーへのインストールおよび初期設定を説明します。

注記

以下は、デフォルトのインスタンス名で CA をインストールしたことを前提としています (つまり **pki-tomcat**)。

PKI ACME Responder の管理方法は、『Red Hat Certificate System 管理ガイド』の『PKI ACME Responder の管理』の章を参照してください。

11.1. PKI ACME RESPONDER のインストール

PKI サーバーに **PKI ACME Responder** をインストールするには、以下を行います。

1. 最初に **pki-acme** RPM パッケージをダウンロードしてインストールします。

```
$ dnf install pki-acme
```

2. 以下のコマンドを使用して、PKI サーバーインスタンスに ACME レスポンダーを作成します。

```
$ pki-server acme-create
```

これにより、**/etc/pki/pki-tomcat/acme** ディレクトリーに初期設定ファイルが作成されます。

詳細は、**pki-server-acme** の man ページを参照してください。

11.2. ACME データベースの設定

本セクションでは、ACME レスポンダーにデータベースを設定する方法を説明します。データベース設定は **/etc/pki/pki-tomcat/acme/database.conf** にあります。

- **pki-server acme-database-mod** コマンドを使用すると、コマンドラインでデータベースを設定できます。パラメーターを指定せずにこのコマンドを呼び出すと、インタラクティブモードが起動します。以下に例を示します。

```
$ 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]:
```

- **--type** パラメーターでコマンドを呼び出すと、指定したタイプに基づいて新しい設定が作成されます。
- 他のパラメーターを指定してコマンドを呼び出すと、指定したパラメーターが更新されます。

特定の ACME 設定プロパティーはデータベースに保存され、クラスター内のすべての ACME レスポンダーを一貫して設定できます。デフォルトでは、ACME 設定プロパティーを取得または更新する際に、ACME レスポンダーがデータベースに直接アクセスし、データベースの負荷を増やす可能性があります。一部のデータベースは、この負荷を軽減するために ACME 設定モニターを提供する場合があります。

11.2.1. DS データベースの設定

ACME レスポンダーが DS データベースを使用するように設定できます。DS データベース設定のサンプルは、**/usr/share/pki/acme/database/ds/database.conf** にあります。

DS データベースを設定するには、次のコマンドを実行します。

1. 以下のコマンドで **/usr/share/pki/acme/database/database/ds/schema.ldif** ファイルをインポートして ACME DS スキーマを追加します。

```
$ ldapmodify -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/database/ds/schema.ldif
```

2. 次に LDIF ファイルを準備して ACME サブツリーを作成します。LDIF ファイルのサンプルは、**usr/share/pki/acme/database/ds/create.ldif** で利用できます。この例では、ベース DN に **dc=acme,dc=pki,dc=example,dc=com** を使用しています。
3. **ldapadd** コマンドを使用して LDIF ファイルをインポートします。

```
$ ldapadd -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/database/ds/create.ldif
```

4. データベース設定ファイルのサンプルを **/usr/share/pki/acme/database/database.conf** から **/etc/pki/pki-tomcat/acme** ディレクトリーにコピーします。または、以下のコマンドを実行して一部のパラメーターをカスタマイズします。

```
$ pki-server acme-database-mod --type ds \
        -DbindPassword=Secret.123
```

5. 必要に応じて設定をカスタマイズします。
   - スタンドアロンの ACME デプロイメントでは、**database.conf** は以下のようになります。

```
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
```

- 共有 CA および ACME デプロイメントでは、database.conf は以下のようになります。

```
class=org.example.acme.database.DSDatabase
        configFile=conf/ca/CS.cfg
        baseDN=dc=acme,dc=pki,dc=example,dc=com
```

DS データベースは、検索の永続性を使用して ACME 設定モニターを提供します。**monitor.enabled=true** パラメーターを設定してこれを有効にできます。

11.3. ACME ISSUER の設定

本セクションでは、PKI ACME Responder に発行者を設定する方法を説明します。**ACME Issuer** の設定は、**/etc/pki/pki-tomcat/acme/issuer.conf** にあります。

pki-server acme-issuer-mod コマンドを使用して、コマンドラインから発行者を設定できます。

- パラメーターを指定せずにこのコマンドを呼び出すと、インタラクティブモードが起動します。以下に例を示します。

```
$ 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]:
```

- **--type** パラメーターでコマンドを呼び出すと、指定したタイプに基づいて新しい設定が作成されます。
- 他のパラメーターを指定してコマンドを呼び出すと、指定したパラメーターが更新されます。

11.3.1. PKI 発行者の設定

PKI 発行者 を使用して証明書を発行するように **PKIACME 応答者** を設定できます。サンプル設定は **/usr/share/pki/acme/issuer/pki/issuer.conf** で確認できます。

- PKI 発行者を設定するには、このサンプル **issuer.conf** を **/etc/pki/pki-tomcat/acme** ディレクトリーにコピーするか、以下のコマンドを実行して一部のパラメーターをカスタマイズします。

```
$ pki-server acme-issuer-mod --type pki \
        -Dusername=caadmin \
        -Dpassword=Secret.123
```

必要に応じて設定をカスタマイズします。**issuer.conf** ファイルは以下のようになります。

```
class=org.example.acme.issuer.PKIIssuer
url=https://localhost.localdomain:8443
profile=acmeServerCert
username=caadmin
password=Secret.123
```

- **url** パラメーターは PKI 発行者の場所を指定します。
- **profile** パラメーターは、使用する証明書プロファイルを指定します。
- クライアント証明書認証を使用するには、**nickname** パラメーターにクライアント証明書のニックネームを指定します。
- Basic 認証を使用するには、**username** パラメーターと **password** パラメーターにそれぞれユーザー名とパスワードを指定します。

11.4. ACME レルムの設定

本セクションでは、**PKI ACME レスポンダー** のレルムを設定する方法を説明します。レルム設定は **/etc/pki/pki-tomcat/acme/realm.conf** にあります。

pki-server acme-realm-mod コマンドを使用すると、コマンドラインで **ACME レルム** を設定できます。

- パラメーターを指定せずにこのコマンドを呼び出すと、インタラクティブモードが起動します。以下に例を示します。

```
$ 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]:
```

- **--type** パラメーターでコマンドを呼び出すと、指定したタイプに基づいて新しい設定が作成されます。
- 他のパラメーターを指定してコマンドを呼び出すと、指定したパラメーターが更新されます。

11.4.1. DS レルムの設定

DS レルムを使用するように **PKI ACME Responder** を設定することができます。**DS Realm** のサンプル設定は、**/usr/share/pki/acme/realm/ds/realm.conf** にあります。

DS レルムを設定するには、次のコマンドを実行します。

1. DS の ACME ユーザーおよびグループのサブツリーを準備します。サンプル LDIF ファイルは **/usr/share/pki/acme/realm/ds/create.ldif** にあります。この例では、ベース DN に **dc=acme,dc=pki,dc=example,dc=com** を使用しています。
2. **ldapadd** コマンドを使用して LDIF ファイルをインポートします。

```
$ ldapadd -h $HOSTNAME -x -D "cn=Directory Manager" -w Secret.123 \
        -f /usr/share/pki/acme/realm/ds/create.ldif
```

3. **/usr/share/pki/acme/realm/ds/realm.conf** から **/etc/pki/pki-tomcat/acme** ディレクトリーに設定例をコピーするか、以下のコマンドを実行して一部のパラメーターをカスタマイズします。

```
$ pki-server acme-realm-mod --type ds \
        -DbindPassword=Secret.123
```

4. 必要に応じて設定をカスタマイズします。
   - スタンドアロンの ACME デプロイメントでは、**realm.conf** ファイルは以下のようになるはずです。

```
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
```

- 共有 CA および ACME デプロイメントでは、**realm.conf** ファイルは以下のようになります。

```
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. ACME RESPONDER のデプロイ

1. ACME レスポンダーを設定したら、以下のコマンドを使用してデプロイします。

```
$ pki-server acme-deploy
```

これにより、**/etc/pki/pki-tomcat/Catalina/localhost/acme.xml** にデプロイメント記述子が作成されます。

PKI サーバーは、数秒後に ACME レスポンダーを自動的に起動します。サーバーを再起動する必要はありません。

2. ACME レスポンダーが実行中であることを確認するには、以下のコマンドを使用します。

```
$ 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"
}
```

詳細は、**pki-server-acme** の man ページを参照してください。

第12章 その他のインストールオプション

pkispawn で作成したすべての Red Hat Certificate System インスタンスは、インストールされているインスタンスについて、CA 署名証明書に使用するデフォルトの署名アルゴリズムやホストの IPv6 アドレスを許可するかどうかなど、インストールされているインスタンスについて仮定します。

本章では、新しいインスタンスのインストールと設定に影響を与える追加の設定オプションについて説明します。そのため、これらの手順の多くは、インスタンスが作成される前に実行されます。

12.1. 軽量サブ CA

デフォルト設定を使用すると、軽量のサブ CA を作成できます。これにより、1 つのサブ CA が発行する証明書のみを受け入れるように、仮想プライベートネットワーク (VPN) ゲートウェイなどのサービスを設定できます。同時に、別のサブ CA またはルート CA が発行する証明書のみを受け入れるように他のサービスを設定できます。

サブ CA の中間証明書を破棄する場合には、このサブ CA で発行された証明書はすべて無効になります。

Certificate System で CA サブシステムを設定すると、ルート CA は自動的に実行されます。作成するサブ CA はすべてこのルート CA の下位局になります。

12.1.1. 軽量サブ CA の設定

ご自分の環境に応じて、サブ CA のインストールは **内部** CA と **外部** CA によって異なります。詳細は、「外部 CA でのサブシステムの設定」 を参照してください。

12.1.2. 軽量サブ CA の作成の無効化

特定の状況では、管理者は軽量のサブ CA を無効にする必要があります。サブ CA の追加、変更、または削除を防ぐには、Certificate System が使用する Directory Server インスタンスで以下のコマンドを入力します。

```
# 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
```

このコマンドは、サブ CA を管理するパーミッションを付与するデフォルトのアクセス制御リスト (ACL) エントリーを削除します。

注記

軽量のサブ CA 作成に関連する ACL を変更または追加する場合は、関連する値を削除します。

12.1.3. 軽量サブ CA の作成の再有効化

以前、軽量のサブ CA の作成を無効にしている場合は、Certificate System が使用する Directory Server インスタンスで以下のコマンドを入力して、機能を再度有効にできます。

```
# 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
```

このコマンドは、アクセス制御リスト (ACL) エントリーを追加します。このエントリーは、サブ CA を管理するパーミッションを付与します。

12.2. サブシステムの IPV6 の有効化

Certificate System は、サブシステム間の接続を自動的に設定および管理します。すべてのサブシステムは、セキュリティードメインのメンバーとして CA と対話し、PKI 操作を実行する必要があります。

これらの接続では、Certificate System サブシステムはホストの完全修飾ドメイン名または IP アドレスで認識できます。デフォルトでは、Certificate System は IPv4 アドレスとホスト名を自動的に解決しますが、Certificate System は接続に IPv6 を使用することもできます。IPv6 は、他のサブシステムへの接続、管理コンソール (**pkiconsole**) への接続、または **tpsclient** などのコマンドラインスクリプトを介した接続など、すべてのサーバー接続でサポートされています。

```
op=var_set name=ca_host value=IPv6 address
```

1. Red Hat Certificate System パッケージをインストールします。
2. **/etc/hosts** ファイルに IPv4 アドレスおよび IPv6 アドレスを設定します。以下に例を示します。

   ```
   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
   ```

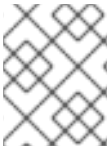
3. 次に、環境変数をエクスポートして、サーバーの IPv6 アドレスを使用します。以下に例を示します。

   ```
   export PKI_HOSTNAME=server6.example.com
   ```

4. **pkispawn** を実行して、新規インスタンスを作成します。**CS.cfg** ファイル内のサーバーのホスト名の値は、IPv6 アドレスに設定されます。

12.3. LDAP ベースの登録プロファイルの有効化

LDAP ベースのプロファイルを使用してインストールするには、**pkispawn** 設定ファイルの **[CA]** セクションに **pki_profile_in_ldap=True** オプションを指定します。



注記

この場合、プロファイルファイルは **/var/lib/pki/*instance_name*/ca/profiles/ca/** にそのまま表示されますが、無視されます。

既存のインスタンスで LDAP ベースのプロファイルを有効にするには、インスタンスの **CS.cfg** で以下を変更します。

```
subsystem.1.class=com.netscape.cmscore.profile.LDAPProfileSubsystem
```

次に、**pki** コマンドラインユーティリティーまたはカスタムスクリプトを使用して、プロファイルを手動でデータベースにインポートします。

12.4. TLS 暗号のカスタマイズ

インストール時に TLS 暗号を適用することができます。『Red Hat Certificate System Administration Guide』の『Configuring Ciphers』セクションを参照してください。

第13章 インストールとクローン作成のトラブルシューティング

本章では、Certificate System のインストール時に発生するより一般的なインストールと移行の問題のトラブルシューティングを説明します。

13.1. インストール

問： Certificate System パッケージまたは更新は表示されません。

答： お使いのシステムが Red Hat サブスクリプション管理サービスに正しく登録され、有効なサブスクリプションが割り当てられ、Certificate System のリポジトリーが有効になっていることを確認します。詳細は、「Red Hat サブスクリプションの添付および Certificate System パッケージリポジトリーの有効化」 を参照してください。

問： init スクリプトは OK ステータスを返しましたが、その CA インスタンスは応答しません。理由

答： これは起こらないはずです。通常 (常にではありませんが)、これは CA のリスナーの問題を示しますが、さまざまな原因が考えられます。発生したエラーを確認するには、以下のコマンドを実行して **journal** ログを確認します。

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

または、**/var/log/pki/*instance_name*/*subsystem_type*/debug** でデバッグログファイルを調べます。

1 つの状況は、CA の PID があり、プロセスが実行されているが、サーバーのリスナーが開かれていないことを示している場合です。これにより、Java 呼び出しクラスエラーが **catalina.out** ファイルに返されます。

```
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
```

これは、JSS または NSS の誤ったバージョンがあることを意味します。プロセスに **libnss3.so** が必要です。以下のコマンドでこれを確認します。

```
ldd /usr/lib64/libjss4.so
```

libnss3.so が見つからない場合は、**/etc/sysconfig/*instance_name*** 設定ファイルで正しいクラスパスを設定します。次に、**systemctl restart pki-tomcatd@*instance_name*.service** コマンドを使用して CA を再起動します。

問： CA 署名証明書のサブジェクト名をカスタマイズするには、pkispawn インタラクティブインストールモードを使用します。

答： これには、**/etc/pki/default.cfg** ファイルへの差異リンクを表す設定ファイルが必要です。pkispawn(8) および pki_default.cfg(5) の man ページを参照してください。

問： ルート認証局に異なる証明書の有効期間と延長を設定したいのですが、pkispawn を使用して設定する方法がわかりません。

答： 現在、**pkispawn** を使用して設定できません。ただし、**pkispawn**で使用する証明書プロファイルを編集してルート CA 証明書を生成する方法は**あります**。

重要

これは、**pkispawn を** 実行して新しい CA インスタンスを作成する **前に** 行う必要があります。

1. **pkispawn** が使用する元の CA 証明書プロファイルをバックアップします。

```
cp -p /usr/share/pki/ca/conf/caCert.profile /usr/share/pki/ca/conf/caCert.profile.orig
```

2. 設定ウィザードが使用する CA 証明書プロファイルを開きます。

```
vim /usr/share/pki/ca/conf/caCert.profile
```

3. Validity Default の有効期限を任意の値にリセットします。たとえば、期間を 2 年に変更するには、次のコマンドを実行します。

```
2.default.class=com.netscape.cms.profile.def.ValidityDefault
2.default.name=Validity Default
2.default.params.range=7200
```

4. プロファイルに新しいデフォルトエントリーを作成し、これをリストに追加して、エクステンションを追加します。たとえば、基本的な制約拡張を追加するには、デフォルトを追加します (この例ではデフォルトの #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
```

次に、新しいデフォルトを使用するデフォルトの一覧にデフォルトの番号を追加します。

```
list=2,4,5,6,7,8,9
```

5. 新規プロファイルを設定したら、**pkispawn** を実行して新規 CA インスタンスを作成し、設定ウィザードに移動します。

問： サブシステムインスタンスを設定した後、Web サービスページに接続しようとすると、HTTP 500 エラーコードが表示されます。

答： これは予期しない一般的なエラーであり、さまざまな原因が考えられます。**journal**、**system**、および **debug** ログファイルでインスタンスに対して、発生したエラーを確認します。これは、いくつかの一般的なエラーを示していますが、他の方法は多数あります。

エラー #1: LDAP データベースは実行していない
内部データベースに Red Hat Directory Server インスタンスが稼働していない場合は、そのインスタンスに接続できません。これは、インスタンスが準備状態にない **journal** ファイルで、例外が発生することはありません。

```
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 ログは、特に LDAP 接続の問題を特定します。

```
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)
```

インスタンスの**debug**ログは以下のようになります。

```
[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)
```

エラー #2: VPN がアクセスをブロックしている
もう 1 つの可能性として、VPN を介してサブシステムに接続している可能性があります。VPN には、**Use this connection only for resources on its network** などの設定オプションが有効になっている **必要** があります。そのオプションが有効になっていない場合は、インスタンスの Tomcat サービスの **journal** ログファイルには、HTTP 500 エラーが発生する一連の接続エラーが表示されます。

```
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$Http11ConnectionHandler.processConnection(Http11BaseProtocol.java:665)
        at org.apache.tomcat.util.net.PoolTcpEndpoint.processSocket(PoolTcpEndpoint.java:528)
        at org.apache.tomcat.util.net.LeaderFollowerWorkerThread.runIt(LeaderFollowerWorkerThread.java:81)
        at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:685)
 at java.lang.Thread.run(Thread.java:636)
```

13.2. Java コンソール

重要

pkiconsole が非推奨になりました。

問： pkiconsole を開くことができません。標準出力 (stdout) で Java の例外が表示されます。

答： これはおそらく、間違った JRE がインストールされているか、間違った JRE がデフォルトとして設定されていることを意味します。**alternatives --config java** を実行して、選択した JRE を確認します。Red Hat Certificate System には OpenJDK 1.7 が必要です。

問： pkiconsole の実行を試みましたが、標準出力 (stdout) でソケット例外が取得されました。理由

答： これは、ポートに問題があることを意味します。管理ポートの SSL/TLS 設定が間違っている (**server.xml** の設定が間違っている) か、管理者インターフェイスにアクセスするために間違ったポートが付与されたかのいずれかです。

ポートエラーは以下のようになります。

```
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:430)
        at com.netscape.admin.certsrv.connection.JSSConnection.sendRequest(JSSConnection.java:344)

        at com.netscape.admin.certsrv.connection.AdminConnection.processRequest(AdminConnection.java:714)
        at com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java:623)
        at com.netscape.admin.certsrv.connection.AdminConnection.sendRequest(AdminConnection.java: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)
```

問： コンソールの起動を試み、システムはユーザー名とパスワードの入力を要求します。これらの認証情報を入力すると、コンソールが表示されません。

答： 入力したユーザー名とパスワードが有効であることを確認してください。その場合は、デバッグ出力を有効にして確認します。

デバッグ出力を有効にするには、**/usr/bin/pkiconsole** ファイルを開き、以下の行を追加します。

```
============================================
${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.
============================================
```

パート III. CERTIFICATE SYSTEM の設定

第14章 CERTIFICATE SYSTEM の設定ファイル

すべてのサブシステムの主な設定ファイルは **CS.cfg** ファイルです。本章では、**CS.cfg** ファイルの編集に関する基本情報およびそのルールについて説明します。この章では、パスワードや Web サービスファイルなど、サブシステムで使用されるその他の便利な設定ファイルについても説明します。

14.1. CERTIFICATE SYSTEM サブシステムのファイルおよびディレクトリーの場所

Certificate System サーバーは、**1つ以上** のサブシステムを含む Apache Tomcat インスタンスで設定されます。各サブシステムは、特定のタイプの PKI 関数の要求を処理する Web アプリケーションで設定されます。

利用可能なサブシステムは CA、KRA、OCSP、TKS、および TPS です。各インスタンスには、PKI サブシステムのタイプを 1 つのみ含めることができます。

pkispawn コマンドを使用して、特定のインスタンス内にサブシステムをインストールできます。

14.1.1. インスタンス固有の情報

デフォルトインスタンスの情報 (pki-tomcat) については、???を参照してください。

表14.1 証明書サーバーポートの割り当て (デフォルト)

ポートタイプ	ポート番号	注記
セキュアなポート	8443	HTTPS 経由でエンドユーザー、エージェント、および管理者により PKI サービスにアクセスするために使用される主要なポート。
セキュアなポート	8080	HTTP を介した一部のエンドエンティティー機能のために、安全ではないサーバーにアクセスするために使用されます。たとえば、すでに署名されているため暗号化する必要のない CRL を提供するために使用されます。
AJP ポート	8009	フロントエンドの Apache プロキシーサーバーから AJP 接続を介してサーバーにアクセスするために使用されます。HTTPS ポートにリダイレクトします。
Tomcat ポート	8005	Web サーバーによって使用されます。

14.1.2. CA サブシステム情報

このセクションには、CA サブシステムに関する詳細が含まれています。CA サブシステムは、Certificate Server インスタンスに Web アプリケーションとしてインストールできるサブシステムの 1 つです。

表14.2 デフォルトインスタンスの CA サブシステム情報 (pki-tomcat)

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/ca/

設定	値
設定ディレクトリー	/var/lib/pki/pki-tomcat/ca/conf/[a]
設定ファイル	/var/lib/pki/pki-tomcat/ca/conf/CS.cfg
サブシステム証明書	CA 署名証明書
	OCSP 署名証明書 (CA の内部 OCSP サービス用)
	TLS サーバー証明書
	監査ログ署名証明書
	サブシステム証明書[b]
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias/[c]
ログファイル	/var/log/pki/pki-tomcat/ca/logs/[d]
ログのインストール	/var/log/pki/pki-ca-spawn.*date*.log
ログのアンインストール	/var/log/pki/pki-ca-destroy.*date*.log
監査ログ	/var/log/pki/pki-tomcat/ca/signedAudit/
プロファイルファイル	/var/lib/pki/pki-tomcat/ca/profiles/ca/
メール通知テンプレート	/var/lib/pki/pki-tomcat/ca/emails/
Web サービスファイル	**エージェントサービス:** /var/lib/pki/pki-tomcat/ca/webapps/ca/agent/
	管理サービス: /var/lib/pki/pki-tomcat/ca/webapps/ca/admin/
	エンドユーザーサービス: /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/

[a] /etc/pki/pki-tomcat/ca/ のエイリアス

[b] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。

[c] すべてのサブシステム証明書がインスタンスセキュリティーデータベースに保存されることに注意してください。

[d] /var/lib/pki/pki-tomcat/ca へのエイリアス

14.1.3. KRA サブシステム情報

このセクションには、KRA サブシステムに関する詳細が含まれています。CA サブシステムは、Certificate Server インスタンスに Web アプリケーションとしてインストールできるサブシステムの 1 つです。

表14.3 デフォルトインスタンスの KRA サブシステム情報 (pki-tomcat)

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/kra/
設定ディレクトリー	/var/lib/pki/pki-tomcat/kra/conf/[a]
設定ファイル	/var/lib/pki/pki-tomcat/kra/conf/CS.cfg
サブシステム証明書	トランスポート証明書
	ストレージ証明書
	TLS サーバー証明書
	監査ログ署名証明書
	サブシステム証明書[b]
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias/[c]
ログファイル	/var/lib/pki/pki-tomcat/kra/logs/
ログのインストール	/var/log/pki/pki-kra-spawn-*date*.log
ログのアンインストール	/var/log/pki/pki-kra-destroy-*date*.log
監査ログ	/var/log/pki/pki-tomcat/kra/signedAudit/
Web サービスファイル	**エージェントサービス:** /var/lib/pki/pki-tomcat/kra/webapps/kra/agent/
	管理サービス: /var/lib/pki/pki-tomcat/kra/webapps/kra/admin/

[a] /etc/pki/pki-tomcat/kra/ にリンク

[b] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。

[c] すべてのサブシステム証明書がインスタンスセキュリティーデータベースに保存されることに注意してください。

14.1.4. OCSP サブシステム情報

このセクションには、OCSP サブシステムに関する詳細が含まれています。CA サブシステムは、Certificate Server インスタンスに Web アプリケーションとしてインストールできるサブシステムの 1 つです。

表14.4 デフォルトのインスタンスの OCSP サブシステム情報 (pki-tomcat)

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/ocsp/
設定ディレクトリー	/var/lib/pki/pki-tomcat/ocsp/conf/[a]
設定ファイル	/var/lib/pki/pki-tomcat/ocsp/conf/CS.cfg
サブシステム証明書	トランスポート証明書
	ストレージ証明書
	TLS サーバー証明書
	監査ログ署名証明書
	サブシステム証明書[b]
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias/[c]
ログファイル	/var/lib/pki/pki-tomcat/ocsp/logs/
ログのインストール	/var/log/pki/pki-ocsp-spawn-*date*.log
ログのアンインストール	/var/log/pki/pki-ocsp-destroy-*date*.log
監査ログ	/var/log/pki/pki-tomcat/ocsp/signedAudit/
Web サービスファイル	**エージェントサービス:** /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/agent/
	管理サービス: /var/lib/pki/pki-tomcat/ocsp/webapps/ocsp/admin/

[a] /etc/pki/pki-tomcat/ocsp/ へのリンク

[b] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。

[c] すべてのサブシステム証明書がインスタンスセキュリティーデータベースに保存されることに注意してください。

14.1.5. TKS サブシステム情報

このセクションには、TKS サブシステムに関する詳細が含まれています。CA サブシステムは、Certificate Server インスタンスに Web アプリケーションとしてインストールできるサブシステムの 1 つです。

表14.5 サブシステムが初期インストールまたは (pki-tomcat による) 追加のインスタンスの作成のいずれかによって作成されるたび

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/tks/
設定ディレクトリー	/var/lib/pki/pki-tomcat/tks/conf/[a]
設定ファイル	/var/lib/pki/pki-tomcat/tks/conf/CS.cfg
サブシステム証明書	トランスポート証明書
	ストレージ証明書
	TLS サーバー証明書
	監査ログ署名証明書
	サブシステム証明書[b]
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias/[c]
ログファイル	/var/lib/pki/pki-tomcat/tks/logs/
ログのインストール	/var/log/pki/pki-tks-spawn-*date*.log
ログのアンインストール	/var/log/pki/pki-tks-destroy-*date*.log
監査ログ	/var/log/pki/pki-tomcat/tks/signedAudit/
Web サービスファイル	**エージェントサービス:** /var/lib/pki/pki-tomcat/tks/webapps/tks/agent/
	管理サービス: /var/lib/pki/pki-tomcat/tks/webapps/tks/admin/

[a] /etc/pki/pki-tomcat/tks/ にリンク

[b] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。

[c] すべてのサブシステム証明書がインスタンスセキュリティーデータベースに保存されることに注意してください。

14.1.6. TPS サブシステム情報

このセクションには、TPS サブシステムに関する詳細が含まれています。CA サブシステムは、Certificate Server インスタンスに Web アプリケーションとしてインストールできるサブシステムの 1 つです。

表14.6 デフォルトインスタンスの TPS サブシステム情報 (pki-tomcat)

設定	値
メインディレクトリー	/var/lib/pki/pki-tomcat/tps
設定ディレクトリー	/var/lib/pki/pki-tomcat/tps/conf/[a]
設定ファイル	/var/lib/pki/pki-tomcat/tps/conf/CS.cfg
サブシステム証明書	トランスポート証明書
	ストレージ証明書
	TLS サーバー証明書
	監査ログ署名証明書
	サブシステム証明書[b]
セキュリティーデータベース	/var/lib/pki/pki-tomcat/alias/[c]
ログファイル	/var/lib/pki/pki-tomcat/tps/logs/
ログのインストール	/var/log/pki/pki-tps-spawn-*date*.log
ログのアンインストール	/var/log/pki/pki-tps-destroy-*date*.log
監査ログ	/var/log/pki/pki-tomcat/tps/signedAudit/
Web サービスファイル	**エージェントサービス:** /var/lib/pki/pki-tomcat/tps/webapps/tps/agent/
	管理サービス: /var/lib/pki/pki-tomcat/tps/webapps/tps/admin/

[a] /etc/pki/pki-tomcat/tps/ にリンク

[b] サブシステム証明書はセキュリティードメインによって常に発行されるため、クライアント認証を必要とするドメインレベルの操作はこのサブシステム証明書をベースにします。

[c] すべてのサブシステム証明書がインスタンスセキュリティーデータベースに保存されることに注意してください。

14.1.7. 共有 Certificate System サブシステムファイルの場所

サーバー全般操作のために、すべての Certificate System サブシステムインスタンスで使用されるディレクトリーがあります (??? に記載されています)。

表14.7 サブシステムファイルの場所

ディレクトリーの場所	コンテンツ
/var/lib/**instance_name**	ユーザー固有のディレクトリーの場所およびカスタマイズされた設定ファイル、プロファイル、証明書データベース、Web ファイル、およびサブシステムインスタンスの他のファイルの場所であるメインインスタンスディレクトリーが含まれます。
/usr/share/java/pki	Certificate System サブシステムによって共有される Java アーカイブファイルが含まれます。すべてのサブシステムの共有ファイルとともに、サブディレクトリーにサブシステム固有のファイルがあります。 pki/ca/ (CA) pki/kra/ (KRA) pki/ocsp/ (OCSP) pki/tks/ (TKS) **TPS サブシステムが使用していない。**
/usr/share/pki	Certificate System インスタンスの作成に使用する一般的なファイルとテンプレートが含まれます。すべてのサブシステムの共有ファイルとともに、サブディレクトリーにサブシステム固有のファイルがあります。 pki/ca/ (CA) pki/kra/ (KRA) pki/ocsp/ (OCSP) pki/tks/ (TKS) pki/tps (TPS)
/usr/bin	Certificate System サブシステムが共有する **pkispawn** および **pkidestroy** インスタンス設定スクリプトおよびツール (Java、ネイティブ、およびセキュリティー) が含まれます。
/var/lib/tomcat5/common/lib	ローカルの Tomcat Web アプリケーションで共有される Java アーカイブファイルへのリンクが含まれ、Certificate System サブシステムによって共有されます。**TPS サブシステムが使用していない。**

ディレクトリーの場所	コンテンツ
/var/lib/tomcat5/server/lib	ローカルの Tomcat Web サーバーによって使用される Java アーカイブファイルへのリンクが含まれ、Certificate System サブシステムによって共有されます。**TPS サブシステムが使用していない。**
/usr/shared/pki	Tomcat サーバーおよび Certificate System インスタンスが使用するアプリケーションが使用する Java アーカイブファイルが含まれます。**TPS サブシステムが使用していない。**
/usr/lib/httpd/modules /usr/lib64/httpd/modules	TPS サブシステムによって使用される Apache モジュールが含まれます。**CA、KRA、OCSP、または TKS サブシステムでは使用されません。**
/usr/lib/mozldap /usr/lib64/mozldap	TPS サブシステムが使用する Mozilla LDAP SDK ツール。**CA、KRA、OCSP、または TKS サブシステムでは使用されません。**

14.2. CS.CFG ファイル

Certificate System サブシステムのランタイムプロパティーは、一連の設定パラメーターで管理されます。これらのパラメーターは、起動時にサーバーが読み込む **CS.cfg** ファイルに保存されます。

CS.cfg (ASCII ファイル) が作成され、サブシステムの初回インストール時に適切な設定パラメーターが入力されます。インスタンスの機能の変更方法は、サブシステムコンソールで変更することが推奨されます。これが推奨される方法です。管理コンソールで行った変更は、設定ファイルに反映されます。

CS.cfg 設定ファイルを直接編集することもできます。場合によっては、サブシステムを管理する最も簡単な方法となる場合があります。

14.2.1. CS.cfg ファイルの検索

Certificate System サブシステムの各インスタンスには、独自の設定ファイル **CS.cfg** があります。各サブシステムインスタンスのファイルの内容は、サブシステムの設定方法、追加の設定および設定 (新規プロファイルの追加、セルフテストの有効化など)、サブシステムのタイプによって異なります。

CS.cfg ファイルは、インスタンスの設定ディレクトリーにあります。

```
/var/lib/pki/instance_name/subsystem_type/conf
```

以下に例を示します。

```
/var/lib/pki/instance_name/ca/conf
```

14.2.2. 設定ファイルの編集

警告

設定パラメーターに精通していない場合や、変更がサーバーに許容されていることを認識せずに、設定ファイルを直接編集しないでください。設定ファイルが正しく変更されていないと、Certificate System は起動に失敗します。設定が間違っていると、データが失われる可能性があります。

CS.cfg ファイルを変更するには、以下の手順に従います。

1. サブシステムインスタンスを停止します。

```
# pki-server stop instance_name
```

または (**nuxwdog watchdog** を使用している場合)

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

設定ファイルは、インスタンスの起動時にキャッシュに保存されます。コンソールを介してインスタンスに加えた変更は、ファイルのキャッシュバージョンに変更されます。サーバーが停止または再起動されると、キャッシュに保存されている設定ファイルがディスクに書き込まれます。設定ファイルを編集する前にサーバーを停止すると、サーバーが停止するとキャッシュバージョンの変更が上書きされます。

2. **/var/lib/pki/*instance_name*/*subsystem_type*/conf** ディレクトリーを開きます。
3. テキストエディターで **CS.cfg** ファイルを開きます。
4. ファイル内のパラメーターを編集して、変更を保存します。
5. サブシステムインスタンスを開始します。

```
# pki-server start instance_name
```

または (**nuxwdog watchdog** を使用している場合)

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

14.2.3. CS.cfg 設定ファイルの概要

各サブシステムインスタンスには、設定用のプラグインや Java クラスなど、インスタンスのすべての設定が含まれる独自のメイン設定ファイル **CS.cfg** があります。パラメーターと特定の設定はサブシステムのタイプによって異なりますが、一般的に **CS.cfg** ファイルはサブシステムインスタンスの以下の部分を定義します。

- 名前、ポート割り当て、インスタンスディレクトリー、ホスト名などの基本サブシステムインスタンス情報
- ログ

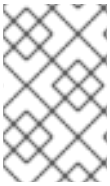
- インスタンスのユーザーディレクトリー (authorization) に対して認証するプラグインおよびメソッド
- インスタンスが属するセキュリティードメイン
- サブシステム証明書
- サブシステムインスタンスによって使用される他のサブシステム
- サブシステムによって使用されるデータベースタイプおよびインスタンス
- TKS のキープロファイル、CA の証明書プロファイル、KRA のキーリカバリーに必要なエージェントなどの PKI 関連タスクの設定

設定パラメーターの多く (PKI タスクのパラメーターを除く) は、すべて Java ベースのコンソールを使用するため、CA、OCSP、KRA、および TKS 間でほとんど同じです。したがって、コンソールで管理できる設定設定には、同様のパラメーターがあります。

CS.cfg ファイルは、基本的な **parameter=value** 形式です。

```
#comment
parameter=value
```

CS.cfg ファイルでは、パラメーターブロックの多くに pound (#) 文字でコメントアウトされた記述的なコメントがあります。コメント、空白行、不明なパラメーター、またはスペルミスのパラメーターはサーバーによって無視されます。



注記

TPS のバグが原因で、**CS.cfg** ファイルでコメントアウトされている行が無視されなくなります。TPS **CS.cfg** ファイルの行をコメントアウトするのではなく、それらの行を削除するだけです。

インスタンスの同じ領域を設定するパラメーターは、同じブロックにグループ化される傾向があります。

機能の一部は、サブシステムにアクセスするためにセルフテスト、ジョブ、認可などのプラグインを使用して実装されます。これらのパラメーターの場合、プラグインインスタンスには、一意の識別子 (サブシステムに対して呼び出される同じプラグインのインスタンスが複数存在する可能性があるため)、実装プラグイン名、および Java クラスがあります。

例14.1 サブシステム認証設定

```
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
```

注記

設定パラメーターの値を適切にフォーマットする必要があるため、2 つのルールを考慮する必要があります。

- ローカライズする必要のある値は、UTF8 文字である必要があります。
- **CS.cfg** ファイルは、パラメーター値のスラッシュ (/) をサポートします。値にバックスラッシュ (\) が必要な場合は、スラッシュでエスケープする必要があります。つまり、2 つのバックスラッシュを続けて使用する必要があります。

以下のセクションでは、**CS.cfg** ファイル設定およびパラメーターのスナップショットです。これらは、完全な参考資料や **CS.cfg** ファイルパラメーターの例ではありません。また、各サブシステム設定ファイルで利用可能なパラメーターと使用されるパラメーターは異なりますが、類似しています。

14.2.3.1. サブシステムの基本設定

基本的な設定は、サブシステムの機能や動作に直接関係することなく、インスタンス自体に固有のものです。これには、インスタンス名、root ディレクトリー、プロセスのユーザー ID、およびポート番号の設定が含まれます。インスタンスの初回インストールまたは設定時に割り当てられる設定の多くは、**pkispawn** で示されます。

例14.2 CA の基本インスタンスパラメーター: pkispawn ファイル 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

ポート設定などの情報は **CS.cfg** ファイルに **含まれ** ますが、**CS.cfg**には **設定** されません。サーバー設定が **server.xml** ファイルに設定されます。

CS.cfg および **server.xml** のポートは、稼働中の RHCS インスタンスと一致している**必要**があります。

14.2.3.2. ログ設定

サブシステムのタイプによって、設定可能なログにはいくつかのタイプがあります。各タイプのログには、**CS.cfg** ファイルに独自の設定エントリーがあります。

ロギング設定の詳細は、「Certificate System ログの設定」 を参照してください。

14.2.3.3. 認証および認可の設定

CS.cfg ファイルには、ユーザーがサブシステムインスタンス (認証) にアクセスする方法や、認証された各ユーザーの承認 (認証) されるアクションを設定します。

CS サブシステムは、認証プラグインを使用して、サブシステムにログインする方法を定義します。

以下の例は、**SharedSecret**という名前の JAVA プラグインをインスタンス化する **SharedToken** という名前の認証インスタンスを示しています。

```
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
```

一部の認証設定では、LDAP データベースを使用してユーザーエントリーを保存する認証方法を選択できます。その場合、データベース設定は、以下に示すようにプラグインとともに設定されます。

```
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
```

LDAP のセキュアな設定とパラメーターの説明は、「TLS クライアント認証の有効化」 を参照してください。パラメーターパスは表示される内容とは異なりますが、両方の場所で同じ名前と値が許可されます。

CA は、ユーザーリクエストの承認メカニズムも必要です。これは、承認の設定と同様に、適切な認証プラグインを特定し、そのためにインスタンスを設定して行います。

```
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. サブシステム証明書の設定

複数のサブシステムには、設定ファイルの各サブシステム証明書のエントリーがあります。

```
ca.sslserver.cert=MIIDmDCCAoCgAwIBAgIBAzANBgkqhkiG9w0BAQUFADBAMR4wHAYDVQQKExVS
ZWR...
ca.sslserver.certreq=MIICizCCAXMCAQAwRjEeMBwGA1UEChMVUmVkYnVkY29tcHV0ZXIgRG9tYWl
uMSQwIgYDV...
ca.sslserver.nickname=Server-Cert cert-pki-ca
ca.sslserver.tokenname=Internal Key Storage Token
```

14.2.3.5. 必要なサブシステムの設定

少なくとも、各サブシステムは CA に依存するため、CA (およびその他の必要なサブシステム) をサブシステムの設定で設定する必要があります。別のサブシステムへの接続の前に **conn.** を付けてから、サブシステムのタイプと番号が付けられます。

```
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. データベース設定

すべてのサブシステムは LDAP ディレクトリーを使用してそれらの情報を保存します。この内部データベースは **internaldb** パラメーターで設定されますが、その他の設定オプションが多数ある **tokendb** パラメーターで設定した TPS を除きます。

```
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
```

LDAP のセキュアな設定とパラメーターの説明は、「TLS クライアント認証の有効化」を参照してください。「TLS クライアント認証の有効化」と 以外の設定は必要ありません。

14.2.3.7. キューの有効化および設定

登録プロセスには、発行した証明書をディレクトリーまたはファイルに公開します。したがって、基本的には、最初の証明書要求を閉じます。ただし、外部ネットワークに証明書を公開すると、発行プロセスが大幅に遅くなり、リクエストが開いたままになります。

この状況を回避するために、管理者は **公開キュー** を有効にできます。パブリッシュキューは、別のリクエストキューを使用するリクエスト操作および登録操作 (外部の LDAP ディレクトリーを伴う可能性がある) を分離します。要求キューはすぐに更新され、登録プロセスが完了したことを示します。一方、公開キューがネットワークトラフィックの一時停止時に情報を送信します。

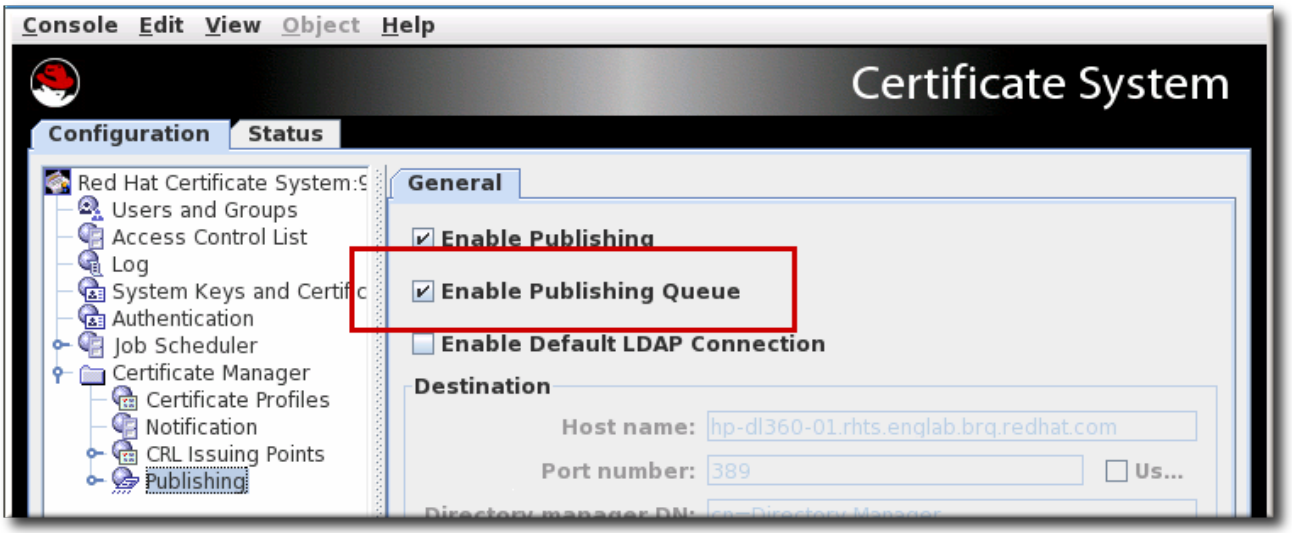
公開キューは、承認された証明書ごとに新しいスレッドを開くのではなく、生成された証明書を公開する定義済みの制限された数のスレッドを設定します。

パブリッシュキューはデフォルトで無効になっています。公開を有効にするとともに、CA コンソールで有効にすることができます。

注記

パブリッシュキューはデフォルトで無効になっていますが、**コンソールで** LDAP 公開が有効な場合にはキューが自動的に有効になります。それ以外の場合は、キューを手動で有効にできます。

図14.1 公開キューの有効化



14.2.3.7.1. CS.cfg ファイルを編集して、Queue の有効化と設定

CS.cfg ファイルを編集してパブリッシュキューを有効にすると、管理者はパブリッシュ操作に使用するスレッドの数やキューページサイズなど、パブリッシュする他のオプションを設定できます。

1. 設定ファイルを編集できるように CA サーバーを停止します。

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

2. CA の **CS.cfg** ファイルを開きます。

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

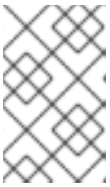
3. ***ca.publish.queue.enable*** を true に設定します。パラメーターが存在しない場合は、パラメーターで行を追加します。

```
ca.publish.queue.enable=true
```

4. その他の関連するパブリッシュキューパラメーターを設定します。

- ***ca.publish.queue.maxNumberOfThreads*** パブリッシュ操作に対して開くことができるスレッドの最大数を設定します。デフォルトは 3 です。
- ***ca.publish.queue.priorityLevel*** は、パブリッシュ操作の優先度を設定します。優先度の値の範囲は、**-2** (最も低い優先度) から **2** (最も高い優先度) までです。ゼロ (0) は通常の優先度で、デフォルトはデフォルトです。
- ***ca.publish.queue.pageSize*** は、パブリッシュキューページに格納できるリクエストの最大数を設定します。デフォルトは 40 です。
- ***ca.publish.queue.saveStatus*** は、指定された公開操作の数ごとにステータスを保存する間隔を設定します。これにより、CA が再起動またはクラッシュした場合にパブリッシュキューを復元できます。デフォルトは 200 ですが、CA の再起動時にゼロ以外の番号がキューを復旧します。このパラメーターを 0 に設定すると、キューの復元が無効になります。

```
ca.publish.queue.maxNumberOfThreads=1
ca.publish.queue.priorityLevel=0
ca.publish.queue.pageSize=100
ca.publish.queue.saveStatus=200
```



ヒント

ca.publish.queue.enable を false に設定し、***ca.publish.queue.maxNumberOfThreads*** を 0 に設定すると、公開キューと、発行した証明書の公開に別のスレッドが使用されます。

5. CA サーバーを起動します。

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

14.2.3.8. PKI タスクの設定

CS.cfg ファイルは、すべてのサブシステムに PKI タスクを設定するのに使用されます。パラメーターは、重複のないサブシステムごとに異なります。

たとえば、KRA には、鍵を復元するために必要な数のエージェントの設定があります。

```
kra.noOfRequiredRecoveryAgents=1
```

各サブシステムの **CS.cfg** ファイルを確認して、PKI タスク設定について理解してください。ファイル内のコメントは、さまざまなパラメーターが何であるかを学習するための適切なガイドです。

- CA 設定ファイルには、すべての証明書プロファイルおよびポリシー設定と、CRL を生成するルールが一覧表示されます。
- TPS は、さまざまなトークン操作を設定します。
- TKS は、さまざまな鍵タイプからの鍵を取得するプロファイルを一覧表示します。
- OCSP は、さまざまな鍵セットの鍵情報を設定します。

14.2.3.9. CA 発行証明書の DN 属性の変更

Certificate System が発行した証明書で、DN は証明書を所有するエンティティーを特定します。いずれの場合も、Certificate System が Directory Server に接続されている場合、証明書内の DN の形式はディレクトリー内の DN の形式と一致する必要があります。名前が完全に一致する必要はありません。証明書マッピングにより、証明書のサブジェクト DN をディレクトリー内の DN とは異なるものにすることができます。

Certificate System の DN は、X.509 標準で定義されたコンポーネントまたは属性に基づいています。表14.8「値タイプに許可される文字」 は、デフォルトでサポートされる属性を一覧表示しています。属性のセットは拡張可能です。

表14.8 値タイプに許可される文字

属性	値のタイプ	オブジェクト識別子
cn	**DirectoryString**	2.5.4.3
ou	**DirectoryString**	2.5.4.11
o	**DirectoryString**	2.5.4.10
c	**PrintableString**、2 文字	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

属性	値のタイプ	オブジェクト識別子
mail	**IA5String**	1.2.840.113549.1.9.1
dc	**IA5String**	0.9.2342.19200300.100.1.2.25
serialnumber	**PrintableString**	2.5.4.5
unstructuredname	**IA5String**	1.2.840.113549.1.9.2
unstructuredaddress	**PrintableString**	1.2.840.113549.1.9.8

デフォルトでは、Certificate System は、表14.8「値タイプに許可される文字」 で特定される属性に対応します。サポートされる属性の一覧は、新しい属性を作成または追加することで拡張できます。**X.500Name** 属性またはコンポーネントを追加する構文は次のとおりです。

```
X500Name.NEW_ATTRNAME.oid=n.n.n.n
X500Name.NEW_ATTRNAME.class=string_to_DER_value_converter_class
```

値コンバータークラスは文字列を ASN.1 値に変換します。このクラスは **netscape.security.x509.AVAValueConverter** インターフェイスを実装する必要があります。文字列から値へのコンバータークラスは、次のいずれかになります。

- **Netscape.security.x509.PrintableConverter** は、文字列を **PrintableString** 値に変換します。この文字列は、出力可能な文字のみでなければなりません。
- **Netscape.security.x509.IA5StringConverter** は文字列を **IA5String** 値に変換します。この文字列は IA5String 文字のみである必要があります。
- **netscape.security.x509.DirStrConverter** は文字列を **DirectoryString** に変換します。この文字列は RFC 2253 に従って **DirectoryString** 形式になることが予想されます。
- **netscape.security.x509.GenericValueConverter** は、最小の文字セットから最大の文字セットへ、次の順序で文字列を文字ごとに変換します。
  - PrintableString
  - IA5String
  - BMPString
  - 汎用文字列

属性エントリーは、以下のようになります。

```
X500Name.MY_ATTR.oid=1.2.3.4.5.6
X500Name.MY_ATTR.class=netscape.security.x509.DirStrConverter
```

14.2.3.9.1. 新規属性またはカスタム属性の追加

Certificate System スキーマに新規またはプロプライエタリー属性を追加するには、以下を行います。

1. Certificate Manager を停止します。

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

2. **/var/lib/pki/*cs_instance*/conf/** ディレクトリーを開きます。

3. 設定ファイル **CS.cfg** を開きます。

4. 設定ファイルに新しい属性を追加します。

   たとえば、**DirectoryString** である **MYATTR1**、**IA5String** である **MYATTR2**、**PrintableString** である **MYATTR3** の 3 つのプロプライエタリー属性を追加するには、設定ファイルの最後に以下の行を追加します。

```
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. 変更を保存して、ファイルを閉じます。

6. Certificate Manager を再起動します。

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

7. 登録ページを再読み込みして変更を確認します。フォームに新しい属性が表示されるはずです。

8. 新しい属性が有効になっていることを確認するには、手動登録フォームを使用して証明書を要求します。

   新しい属性に値を入力します。これにより、証明書のサブジェクト名に表示されることを確認できます。たとえば、以下の値を入力して、新しい属性の値を入力し、サブジェクト名で確認します。

```
MYATTR1: a_value
MYATTR2: a.Value
MYATTR3: aValue
cn: John Doe
o: Example Corporation
```

9. エージェントサービスページを開き、要求を承認します。

10. 証明書の発行時に、発行先名を確認します。証明書には、サブジェクト名に新しい属性値が表示されるはずです。

14.2.3.9.2. DER エンコード順序の変更

異なるクライアントが異なるエンコーディングをサポートしているため、**DirectoryString** の DER エンコーディングの順序を変更して、文字列を設定できます。

DirectoryString の DER-エンコーディング順序を変更する構文は以下の通りです。

```
X500Name.directoryStringEncodingOrder=encoding_list_separated_by_commas
```

エンコーディング値は以下のとおりです。

- **PrintableString**
- **IA5String**
- **UniversalString**
- **BMPString**
- **UTF8String**

たとえば、DER エンコーディング順序は次のようになります。

```
X500Name.directoryStringEncodingOrder=PrintableString,BMPString
```

DirectoryString エンコーディングを変更するには、以下を行います。

1. Certificate Manager を停止します。

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

2. **/var/lib/pki/*cs_instance*/conf/** ディレクトリーを開きます。
3. **CS.cfg** 設定ファイルを開きます。
4. 設定ファイルにエンコーディング順序を追加します。

   たとえば、**PrintableString** と **UniversalString** の 2 つのエンコーディング値を指定し、エンコーディング順序が最初に **PrintableString**、次に **UniversalString** を指定するには、設定ファイルの最後に以下の行を追加します。

   ```
   X500Name.directoryStringEncodingOrder=PrintableString,UniversalString
   ```

5. 変更を保存して、ファイルを閉じます。
6. Certificate Manager を開始します。

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

7. エンコーディングの順序が有効であることを確認するには、手動登録フォームを使用して証明書を登録します。**cn** には **John_Doe** を使用します。
8. エージェントサービスページを開き、要求を承認します。
9. 証明書が発行されたら、**dumpasn1** ツールを使用して証明書のエンコーディングを検証します。

   サブジェクト名の **cn** コンポーネントは、**UniversalString** としてエンコードする必要があります。

10. **cn** に **John Smith** を使用して新しい要求を作成して提出します。

サブジェクト名の **cn** コンポーネントは、**PrintableString** としてエンコードする必要があります。

14.2.3.10. 異なる証明書を使用するように CA を設定して CRL を署名

Certificate Manager は、証明書および証明書失効リスト (CRL) の署名に OCSP 署名証明書に対応するキーペアを使用します。別のキーペアを使用して Certificate Manager が生成する CRL に署名するには、CRL 署名証明書を作成できます。Certificate Manager の CRL 署名証明書は、署名するか、または自己発行する必要があります。

Certificate Manager が異なるキーペアで CRL に署名できるようにするには、以下を行います。

1. Certificate Manager の CRL 署名証明書を要求します。

   または、次のようなキーペアを生成できるツールを使用します。たとえば、**certutil** ツールを使用してキーペアを生成し、キーペアの証明書を要求し、Certificate Manager の証明書データベースに証明書をインストールします。**certutil** ツールの詳細は http://www.mozilla.org/projects/security/pki/nss/tools/ を参照してください。

2. 証明書要求を作成したら、Certificate Manager エンドページを介して証明書を送信し、Manual OCSP Manager Signing Certificate registration プロファイルなどの適切なプロファイルを選択します。このページには、次の形式の URL が含まれます。

   ```
   https://hostname:port/ca/ee/ca
   ```

3. 要求が送信されたら、エージェントサービスページにログインします。

4. 必要な拡張機能について要求を確認します。CRL 署名証明書には、**crlSigning** ビットが設定された **キー使用法** 拡張が含まれている必要があります。

5. 要求を承認します。

6. CRL 署名証明書が生成されたら、コンソールの **システムキーおよび証明書** を使用して、Certificate Manager のデータベースに証明書をインストールします。

7. Certificate Manager を停止します。

   ```
   # pki-server stop instance_name
   ```

8. Certificate Manager の設定を更新して、新しいキーペアと証明書を認識します。

   a. Certificate Manager インスタンス設定ディレクトリーに移動します。

      ```
      # cd /var/lib/pki/instance-name/ca/conf/
      ```

   b. **CS.cfg** ファイルを開き、以下の行を追加します。

      ```
      ca.crl_signing.cacertnickname=nickname cert-instance_ID
      ca.crl_signing.defaultSigningAlgorithm=signing_algorithm
      ca.crl_signing.tokenname=token_name
      ```

      ニックネーム は、CRL 署名証明書に割り当てられた名前です。

      instance_ID は、Certificate Manager インスタンスの名前です。

インストールされた CA が RSA ベースの CA の場合には、*signing_algorithm* は **SHA256withRSA**、**SHA384withRSA** または **SHA512withRSA** になります。インストールされている CA が EC ベースの CA の場合、*signing_algorithm* は **SHA256withEC**、**SHA384withEC**、**SHA512withEC** になります。

token_name は、キーペアの生成に使用されるトークンの名前と証明書です。内部/ソフトウェアトークンを使用する場合は、**内部キーストレージトークン** を値として使用します。

たとえば、エントリーは以下のようになります。

```
ca.crl_signing.cacertnickname=crlSigningCert cert-pki-ca
ca.crl_signing.defaultSigningAlgorithm=SHAMD512withRSA
ca.crl_signing.tokenname=Internal Key Storage Token
```

c. 変更を保存して、ファイルを閉じます。

9. Certificate Manager を再起動します。

```
# pki-server restart instance_name
```

これで、Certificate Manager は CRL 署名証明書を使用して、生成した CRL に署名できるようになりました。

14.2.3.11. CS.cfg のキャッシュからの CRL 生成の設定

CRL キャッシュは、メモリーに保持されている失効情報のコレクションから証明書失効情報を取得できるようにする単純なメカニズムです。最適なパフォーマンスを得るには、すでにデフォルトの動作を表すこの機能を有効にすることが推奨されます。次の設定情報 (デフォルト) は、情報提供の目的で、または変更が必要な場合に表示されます。

1. 認証局サーバーを停止します。

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

2. CA 設定ディレクトリーを開きます。

```
# cd /var/lib/instance_name/conf/
```

3. **CS.cfg** ファイルを編集して、***enableCRLCache*** パラメーターおよび ***enableCacheRecovery*** パラメーターを true に設定します。

```
ca.crl.MasterCRL.enableCRLCache=true
ca.crl.MasterCRL.enableCacheRecovery=true
```

4. CA サーバーを起動します。

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

14.2.3.12. CS.cfg での CRL の更新間隔の設定

以下では、CRL システムを柔軟に設定して目的の動作を反映する方法について説明します。ゴールは、2 種類のスケジュールに応じて CRL 更新を設定することが目的です。1 つのタイプは明示的な時間のリ

ストを許可し、もう 1 つのタイプは更新間の時間間隔の長さで設定されます。また、共にドリフトを考慮して考慮できるハイブリッドシナリオもあります。以下の注記のエントリーは、実際にボックスシナリオのデフォルトを示しています。

デフォルトのシナリオは以下のとおりです。

```
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
```

より詳細で具体的な更新スケジュールが必要な場合にのみ、これから逸脱してください。残りのセクションでは、その実行方法を示します。

CS.cfg ファイルで完全な CRL および delta CRL の設定を行うには、パラメーターを編集する必要があります。

表14.9 CRL 拡張間隔パラメーター

パラメーター	説明	許可される値
updateSchema	完全な CRL ごとに生成されるデルタ CRL 数の比率を設定します。	整数値
enableDailyUpdates	設定された時間に基づいて CRL 更新の設定を有効または無効にします。	true または false
enableUpdateInterval	設定された間隔に基づいて CRL 更新の設定を有効または無効にします。	true または false
dailyUpdates	CRL の更新時間を設定します。	コンマ区切りの時間リスト
autoUpdateInterval	CRL を更新する間隔を分単位で設定します。	整数値
autoUpdateInterval.effectiveAtStart	現在スケジュールされている nextUpdate の時刻を待つのではなく、システムが自動更新の新しい値をすぐに使用できるようにします	true または false
nextUpdateGracePeriod	CRL の有効期間に分単位を追加し、公開またはレプリケーション期間全体で CRL が有効である状態にする期間を追加します。	整数値
refreshInSec	クローン OCSP のスレッドの周期を秒単位で設定して、LDAP で CRL の更新を確認します。	整数値

重要

autoUpdateInterval.effectiveAtStart パラメーターでは、新しい値を適用するためにシステムを再起動する必要があります。このパラメーターのデフォルト値は false です。これは、自分が何をしているかを確信しているユーザーのみが変更する必要があります。

手順14.1 CS.cfg での CRL 更新間隔の設定方法

1. 認証局サーバーを停止します。

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

2. CA 設定ディレクトリーに移動します。

```
# cd /var/lib/instance_name/conf/
```

3. **CS.cfg** ファイルを編集し、次の行を追加して更新間隔を設定します。

```
ca.crl.MasterCRL.updateSchema=3
```

デフォルトの間隔は 1 です。これは、CRL が生成されるたびに完全な CRL が生成されることを意味します。***updateSchema*** の間隔は、任意の整数に設定できます。

4. 周期的な間隔を指定するか、更新を実行する時間を設定して、更新頻度を設定します。

- ***enableDailyUpdates*** パラメーターを有効にして設定する時間を指定し、***dailyUpdates*** パラメーターに希望する時間を追加します。

```
ca.crl.MasterCRL.enableDailyUpdates=true
ca.crl.MasterCRL.enableUpdateInterval=false
ca.crl.MasterCRL.dailyUpdates=0:50,04:55,06:55
```

このフィールドは、CRL を更新する必要がある毎日の時間を設定します。複数回指定するには、**01:50,04:55,06:55** などのコンマ区切りリストを入力します。複数日のスケジュールを入力するには、コンマ区切りのリストを入力して同じ日の時間を設定し、セミコロンで区切ったリストを入力して異なる日の時間を識別します。たとえば、**01:50,04:55,06:55;02:00,05:00,17:00** を設定して、サイクルの 1 日目の 午前 1:50、4:55、および 6:55、そして 2 日目の午前 2:00、5:00、および午後 5:00 に失効を設定します。

enableUpdateInterval パラメーターを有効にして間隔を指定し、***autoUpdateInterval*** パラメーターに必要な間隔を分単位で追加します。

```
ca.crl.MasterCRL.enableDailyUpdates=false
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
```

5. 環境に応じて以下のパラメーターを設定します。

- OCSP サブシステムなしで CA を実行する場合は、以下を設定します。

```
ca.crl.MasterCRL.nextUpdateGracePeriod=0
```

- OCSP サブシステムで CA を実行する場合は、以下を設定します。

```
ca.crl.MasterCRL.nextUpdateGracePeriod=time_in_minutes
```

ca.crl.MasterCRL.nextUpdateGracePeriod パラメーターは時間 (分単位) を定義します。この値は、CA が新しい CRL を OCSP に伝播するのに十分な大きさである必要があります。パラメーターをゼロ以外の値に設定する必要があります。

お使いの環境に OCSP クローンが追加されている場合は、以下も設定します。

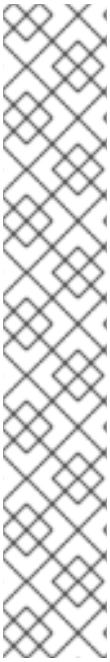
```
ocsp.store.defStore.refreshInSec=time_in_seconds
```

ocsp.store.defStore.refreshInSec パラメーターは、マスター OCSP インスタンスからの LDAP レプリケーション更新を使用して、クローン OCSP インスタンスが CRL 更新に通知する頻度を秒単位で設定します。

パラメーターの詳細は、表14.9「CRL 拡張間隔パラメーター」を参照してください。

6. CA サーバーを起動します。

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



注記

間隔ごとに CRL を更新するとドリフトが発生する場合があります。通常、ドリフトは手動更新と CA の再起動時に実行されます。

ドリフトのスケジュールを防ぐには、***enableDailyUpdates*** パラメーターと ***enableUpdateInterval*** パラメーターを true に設定し、必要な値を ***autoUpdateInterval*** および ***dailyUpdates*** に追加します。

```
ca.crl.MasterCRL.enableDailyUpdates=true
ca.crl.MasterCRL.enableUpdateInterval=true
ca.crl.MasterCRL.autoUpdateInterval=240
ca.crl.MasterCRL.dailyUpdates=1:00
```

間隔別に CRL を更新する場合は、***dailyUpdates*** 値を 1 つだけ受け入れます。

間隔を更新すると、24 時間ごとに ***dailyUpdates*** の値と再同期され、スケジュールのドリフトを防ぐことができます。

14.2.3.13. サブシステムのアクセス制御設定の変更

デフォルトでは、アクセス制御ルールは最初に拒否ルールを評価してから、許可ルールを評価することで適用されます。この順序を変更するには、**CS.cfg** の ***authz.evaluateOrder*** パラメーターを変更します。

```
authz.evaluateOrder=deny,allow
```

さらに、アクセス制御ルールは、ローカルの **web.xml** ファイル (基本 ACL) から評価することも、LDAP データベースを確認すると、より複雑な ACL にアクセスすることもできます。***authz.sourceType*** パラメーターは、使用する承認のタイプを特定します。

```
authz.sourceType=web.xml
```

注記

CS.cfg ファイルの編集後には、更新された設定を読み込むためにサブシステムを常に再起動します。

14.2.3.14. リクエスト番号とシリアル番号の範囲の設定

クローンシステムでランダムなシリアル番号を使用しない場合、管理者は **/etc/pki/*instance_name*/*subsystem*/CS.cfg** ファイルで、リクエストおよびシリアル番号に Certificate System が使用する範囲を指定できます。

```
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
```

注記

Certificate System は、範囲で **BigInteger** の値をサポートします。

14.2.3.15. TLS クライアント証明書認証を使用するための pkiconsole 要件の設定

注記

pkiconsole が非推奨になりました。

各サブシステムの **CS.cfg** ファイルを編集し、***authType*** パラメーターを検索して、以下のように設定します。

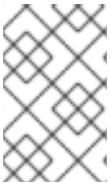
```
authType=sslclientauth
```

14.3. システムパスワードの管理

「パスワードおよびウォッチドッグ (nuxwdog)」で説明されているように、Certificate System は、サーバーにバインドするパスワードを使用するか、サーバーの起動時にトークンのロックを解除します。

password.conf ファイルは、システムパスワードをプレーンテキストで保存します。ただし、一部の管理者は、パスワードファイルを完全に削除して、**nuxwdog** が各パスワードの手動入力を要求し、予定外のシャットダウンの場合は自動再起動を保存することを希望します。

Certificate System インスタンスが起動すると、サブシステムは自動的に **password.conf** ファイルを確認します。ファイルが存在する場合は、それらのパスワードを使用して内部 LDAP データベースなどの他のサービスに接続します。そのファイルが存在しない場合は、ウォッチドッグデーモンは、PKI サーバーが起動するのに必要なすべてのパスワードを要求します。



注記

password.conf ファイルが存在する場合、サブシステムでは必要なパスワードがすべて存在し、適切にフォーマットされたことを前提としています。パスワードが欠落しているか、フォーマットが間違っていると、システムは正しく起動できません。

必要なパスワードは、**CS.cfg** ファイルの ***cms.passwordlist*** パラメーターに一覧表示されます。

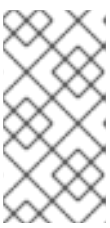
```
cms.passwordlist=internaldb,replicationdb,CA LDAP Publishing
cms.password.ignore.publishing.failure=true
```

注記

cms.password.ignore.publishing.failure パラメーターを使用すると、LDAP 公開ディレクトリーのいずれかへの接続に失敗しても、CA サブシステムが正常に起動できるようになります。

CA、KRA、OCSP、および TKS サブシステムでは、デフォルトの予想されるパスワードは以下のとおりです。

- **internal** (NSS データベースの場合)
- **internaldb** (内部 LDAP データベースの場合)
- **replicationdb** (レプリケーションパスワードの場合)
- パブリッシュ用に外部 LDAP データベースにアクセスするためのパスワード (CA のみ)

  注記

  パブリッシャーが **password.conf** ファイルの削除後に設定されている場合は、**password.conf** ファイルに書き込まれません。**nuxwdog** が設定されていない場合、サーバーは、次回インスタンスを再起動したときに新しい公開パスワードを要求するプロンプトにアクセスできません。

- 外部のハードウェアトークンのパスワード

TPS の場合は、これにより 3 つのパスワードが要求されます。

- **internal** (NSS データベースの場合)
- **tokendbpass** (内部 LDAP データベースの場合)
- 外部のハードウェアトークンのパスワード

本セクションでは、Certificate System がこれらのパスワードを取得するメカニズムを 2 つ説明します。

- **password.conf** ファイル (デフォルト)
- nuxwdog (watchdog)

14.3.1. password.conf ファイルの設定

注記

本セクションは参照用途としてのみ提供されています。正しい操作とセキュアな操作には、**nuxwdog** ウォッチドッグを使用する必要があります。完全なコンプライアンスに必要なため、**nuxwdog** を有効にするには 「Certificate System の Watchdog サービスの使用」 を参照してください。

デフォルトでは、パスワードは、サブシステム **conf/** ディレクトリーにプレーンテキストファイル **password.conf** に保存されます。したがって、テキストエディターを使用して変更できます。

このファイルに保存されているパスワードの一覧には、以下が含まれます。

- 内部データベースへのアクセスおよび更新に Certificate System インスタンスによって使用されるバインドパスワード。
- HSM のパスワード。
- 認証ディレクトリーにアクセスするために Certificate System インスタンスによって使用されるバインドパスワード (CMC Shared Token の場合)。
- LDAP 公開ディレクトリーにアクセスして更新するためにサブシステムによって使用されるバインドパスワード。これは、証明書および CRL を LDAP 準拠のディレクトリーに公開するように設定されている場合にのみ必要です。
- サブシステムがレプリケーションデータベースにアクセスするために使用するバインドパスワード。
- TPS インスタンスでは、トークンデータベースへのアクセスおよび更新に使用されるバインドパスワード。

password.conf ファイルには、サブシステムの秘密鍵を開くのに必要なトークンパスワードも含まれます。

サブシステムに使用する名前と場所のパスワードファイルは **CS.cfg** ファイルに設定されています。

```
passwordFile=/var/lib/pki/instance_name/conf/password.conf
```

内部パスワードストアとレプリケーションデータベースには、サブシステムのインストールと設定時に設定されたランダムに生成された PIN があります。内部 LDAP データベースのパスワードは、インスタンスの設定時に管理者によって定義されました。

password.conf ファイルのパスワードエントリーの形式は、以下のとおりです。

```
name=password
```

以下に例を示します。

```
internal=413691159497
```

HSM トークンが必要な場合は、以下の形式を使用します。

```
hardware-name=password
```

以下に例を示します。

```
hardware-NHSM-CONN-XC=MyHSM$S8cret
```

password.conf ファイルの内容の例:

```
internal=376577078151
internaldb=secret12
replicationdb=1535106826
hardware-NHSM-CONN-XC=MyHSM$S8cret
```

14.3.2. Certificate System の Watchdog サービスの使用

Certificate System では、ウォッチドッグサービスを使用して、開始するためにセキュリティーデータベースにアクセスするためにパスワードを必要とするサービスを開始します。暗号化されていないパスワードをシステムに保存しないという要件がある場合、ウォッチドッグサービスは次のようになります。

- サーバーの起動時に関連するパスワードを要求し、そのメッセージをキャッシュします。
- クラッシュが原因でサーバーが自動的に再起動される場合に、キャッシュされたパスワードを使用します。

14.3.2.1. Watchdog サービスの有効化

ウォッチドッグサービスを有効にするには、以下を実行します。

1. このホストで Shared Secret 機能を使用する場合は、「CMC 共有シークレット機能の有効化」の説明に従って Shared Secret 機能を有効にします。
2. **/var/lib/pki/*instance_name*/conf/** ディレクトリーから **server.xml** および **password.conf** ファイルをバックアップします。以下に例を示します。

   ```
   # cp -p /var/lib/pki/instance_name/conf/server.xml /root/
   # cp -p /var/lib/pki/instance_name/conf/password.conf /root/
   ```

3. Certificate System インスタンスのサービスを停止し、無効にします。

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

4. Hardware Security Module (HSM) を使用する場合は、ウォッチドッグサービスを有効にして、ハードウェアトークンのパスワードを入力するようにします。

   a. ハードウェアトークンの名前を表示します。

   ```
   # egrep "^hardware-" /var/lib/pki/instance_name/conf/password.conf
   hardware-HSM_token_name=password
   ```

   上記の例で強調表示された文字列は、ハードウェアトークン名です。

   b. **/var/lib/pki/*instance_name*/conf/ca/CS.cfg** ファイルに ***cms.tokenList*** パラメーターを追加して、ハードウェアトークンの名前に設定します。以下に例を示します。

   ```
   cms.tokenList=HMS_token_name
   ```

5. インスタンスのウォッチドッグ設定を有効にします。

   ```
   # pki-server instance-nuxwdog-enable instance_name
   ```

   または、すべてのインスタンスでウォッチドッグを有効にします。

   ```
   # pki-server nuxwdog-enable
   ```

   詳細は、pki-server-nuxwdog(8) の man ページを参照してください。

6. デフォルトでは、**nuxwdog** は、**/etc/sysconfig/pki-tomcat** ファイルの ***TOMCAT_USER*** 変数で設定したユーザーとしてサーバーを起動します。必要に応じて、ユーザーおよびグループを変更する場合は、次のコマンドを実行します。

   a. インスタンスの watchdog **systemd** ユニットファイルを、**/etc/systemd/system/** ディレクトリーにコピーします。

   ```
   # cp -p /usr/lib/systemd/system/instance_name-nuxwdog@.service /etc/systemd/system/
   ```

   注記

   /etc/systemd/system/ ディレクトリーのユニットファイルの優先度は高く、更新中は置き換えられません。

   b. **/etc/pki/*instance_name*/nuxwdog.conf** ファイルの ***[Service]*** セクションに以下のエントリーを追加します。

   ```
   User new_user_name
   ```

   c. **systemd** 設定をリロードします。

   ```
   # systemctl daemon-reload
   ```

7. ウォッチドッグを使用する Certificate System サービスを有効にします。

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

8. 必要に応じて、「Certificate System の Watchdog が有効になっていることの確認」を参照してください。

9. Certificate System インスタンスを起動するには、以下のコマンドを実行してプロンプトを入力します。

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

14.3.2.2. Watchdog が有効になっている Certificate System の起動および停止

Certificate System インスタンスの管理方法は、「実行管理 (systemctl)」を参照してください。

14.3.2.3. Certificate System の Watchdog が有効になっていることの確認

ウォッチドッグサービスが有効になっていることを確認するには、次のコマンドを実行します。

1. **pki-tomcatd-nuxwdog** サービスが有効になっていることを確認します。

```
# systemctl is-enabled pki-tomcatd-nuxwdog@instance_name.service
enabled
```

2. **pki-tomcatd** サービスが無効になっていることを確認します。

```
# systemctl is-disabled pki-tomcatd@instance_name.service
disabled
```

3. **/etc/pki/*instance_name*/server.xml** ファイルで以下を行います。

   a. ***passwordFile*** パラメーターが **CS.cfg** ファイルを参照することを確認します。以下に例を示します。

```
passwordFile="/var/lib/pki/instance_name/ca/CS.cfg"
```

   b. ***passwordClass*** パラメーターが **com.netscape.cms.tomcat.NuxwdogPasswordStore** に設定されていることを確認します。

```
passwordClass="com.netscape.cms.tomcat.NuxwdogPasswordStore"
```

14.3.2.4. Watchdog サービスの無効化

ウォッチドッグサービスを無効にするには、次のコマンドを実行します。

1. ウォッチドッグを使用する Certificate System インスタンスのサービスを停止して無効にします。

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

2. インスタンスのウォッチドッグなしで通常のサービスを有効にします。

```
# pki-server instance-nuxwdog-disable instance_name
```

3. インスタンスのウォッチドッグ設定を無効にします。

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

詳細は、pki-server-nuxwdog(8) の man ページを参照してください。

4. **password.conf** ファイルを元の場所に復元します。以下に例を示します。

```
# cp /root/password.conf.bak /var/lib/pki/instance_name/conf/password.conf
```

5. Certificate System インスタンスを起動します。

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

14.4. TOMCAT ENGINE および WEB サービスの設定ファイル

サブシステムのユーザーおよび管理 (管理者、エージェント、および監査者) サービスはすべて、Web プロトコルを介してアクセスされます。

本セクションでは、すべての Red Hat Certificate System サブシステム (CA、KRA、OCSP、TKS、および TPS) に適用される設定ファイルの 2 つの主要なセットを説明します。

- **/var/lib/pki/*instance_name*/conf/server.xml** で Tomcat エンジンの設定を指定します。
- **/usr/share/pki/*subsystem_type*/webapps/WEB-INF/web.xml** は、このインスタンスが提供する Web サービスの設定を指定します。

14.4.1. Tomcatjss

注記

以降のサブセクションには、パラメーター値で必要な変更に関する重要な設定情報が含まれます。必ず厳密なコンプライアンスを確保してください。

サンプルの **pki-tomcat/conf** ディレクトリーにある **server.xml** ファイルの以下の設定は、**Tomcatjss** が Certificate System エコシステム全体にどのように適合するかを説明するために使用できます。シークレットポートの Connector エントリーの一部を以下に示します。

```
<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"

/>
```

Tomcat エンジンの **server.xml** 設定ファイルに、この Connector 設定要素として、この Connector オブジェクトの **sslImplementation** プロパティーにプラグインできる tomcatjss 実装へのポインターが含まれるこの Connector 設定要素があります。

各キーパラメーター要素は、以下のサブセクションで説明します。

14.4.1.1. TLS 暗号の設定

server.xml ファイルで設定した TLS 暗号は、Red Hat Certificate システムがクライアントおよびサーバーとして機能する際にシステム全体のデフォルトを提供します。これには、サーバーとして機能する場合 (たとえば、Tomcat から HTTPS 接続を提供する場合) およびクライアントとして機能する場合 (たとえば、LDAP サーバーと通信する場合または別の Certificate System インスタンスと通信する場合) が含まれます。

サーバー TLS 暗号の設定は、Red Hat Certificate System インスタンス固有の **/var/lib/pki/*instance_name*/conf/server.xml** ファイルにあります。以下のパラメーターは、提供される暗号を制御します。

- ***strictCiphers*** を **true** に設定すると、***sslRangeCiphers*** に **+** 記号が付いた暗号のみが有効になります。

  ```
  strictCiphers="true"
  ```

  デフォルト値 (**true**) を変更しないでください。

- ***sslVersionRangeStream*** および ***sslVersionRangeDatagram*** は、サーバーがサポートする TLS バージョンを設定します。パラメーターのデフォルト値は以下のとおりです。

  ```
  sslVersionRangeStream="tls1_1:tls1_2"
  sslVersionRangeDatagram="tls1_1:tls1_2"
  ```

  パラメーターのデフォルト値は変更しないでください。

- ***sslRangeCiphers*** は、有効または無効にする暗号を設定します。**+** 記号の付いた暗号は有効になり、**-** 記号の付いた暗号は無効になります。

  RSA 暗号を以下のように設定します。

  ```
  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
  ```

  EC 暗号を以下のように設定します。

  ```
  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
  ```

  許可される暗号の一覧は、「TLS、ECC、および RSA」を参照してください。

- RSA で FIPS モードが有効になっているシステムに LunaSA または nCipher の Hardware Security Module (HSM) のいずれかを使用した Certificate System をインストールする場合は、FIPS モードの HSM ではサポートされていないため、以下の暗号を無効にします。

  - 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. クライアント TLS 暗号の設定

Red Hat Certificate System は、別の CS システムへのクライアントとして動作している場合にシステムでの暗号化設定を可能にします。

リストの暗号はコンマで区切ります。

CA で (KRA と CA の通信用) 、以下を行います。

```
ca.connector.KRA.clientCiphers=your selected cipher list
```

以下に例を示します。

```
ca.connector.KRA.clientCiphers=TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_
CBC_SHA
```

TPS の場合 (CA、KRA、および TKS との通信用):

```
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
```

以下に例を示します。

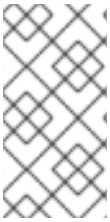
```
tps.connector.ca1.clientCiphers=TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_
CBC_SHA
```

14.4.1.2. CA での自動失効チェックの有効化

CA は、SSL/TLS クライアントの認証中にサーバーが受信する証明書 (エージェント、管理者、登録を含む) の失効ステータスを確認するように設定できます。つまり、CA がクライアント認証要求を受け取ると、OCSP が自動的にチェックされます。OCSP レスポンダーの設定に関する詳細は、『Red Hat Certificate システム管理ガイド』の『OCSP (Online Certificate Status Protocol) レスポンダーの使用』を参照してください。

CA は、失効チェックの一環としてクライアント認証をキャッシュし、検証された証明書の一覧を保持するようにします。これにより、CA は内部データベースまたは OCSP をチェックする前にキャッシュされた結果をチェックできるようになり、全体的な操作パフォーマンスが向上します。***revocationChecking.enabled*** パラメーターで、自動失効チェックが有効になります。

失効ステータスの結果は、指定された期間 (***revocationChecking.validityInterval***) のみ有効になります。CA がキャッシュにある証明書の状態を再検証する方法がない場合は、有効期間外であっても、以前キャッシュされていた状態が有効とみなされる猶予間隔 (***revocationChecking.unknownStateInterval***) があります。



注記

キャッシュされた証明書はバッファー (***revocationChecking.bufferSize***) に保持されます。バッファー設定がない場合やゼロに設定されている場合は、バッファーは保持されません。つまり、失効チェックの結果はキャッシュされません。この場合、失効チェックはすべて CA 内部データベースに対して直接実行されます。

注記

サブシステム **CS.cfg** 設定ファイルにはパラメーター **jss.ocspcheck.enable** が含まれています。このパラメーターは、Certificate Manager が OCSP を使用して、SSL/TLS クライアントまたはサーバー認証の一部として受信する証明書の失効状態を検証するかどうかを設定します。このパラメーターの値を **true** に変更すると、Certificate Manager は証明書の Authority Information Access 拡張を読み取り、拡張に指定された OCSP レスポンダーから証明書の失効状態を検証します。

1. サブシステムインスタンスを停止します。

```
# pki-server stop instance_name
```

2. **CS.cfg** ファイルを開きます。

```
# vim /var/lib/pki/instance-name/ca/conf/CS.cfg
```

3. 失効関連のパラメーターを編集します。

```
auths.revocationChecking.bufferSize=50
auths.revocationChecking.ca=ca
auths.revocationChecking.enabled=true
auths.revocationChecking.unknownStateInterval=0
auths.revocationChecking.validityInterval=120
```

- ***revocationChecking.ca***。OCSP 応答、CA または OCSP レスポンダーを提供するサービスを設定します。
- ***revocationChecking.enabled***。失効チェックを設定します。**true** は、チェックを有効にします。**false は** チェックを無効にします。デフォルトでは、この機能は有効になっています。
- ***revocationChecking.bufferSize***。サーバーがキャッシュで維持する必要のある、最後に確認された証明書の合計数を設定します。たとえば、バッファーサイズが 2 の場合、サーバーはキャッシュでチェックされた最後の 2 つの証明書を保持します。デフォルトでは、サーバーは最後の 50 個の証明書をキャッシュします。
- ***revocationChecking.unknownStateInterval***。サーバーが失効ステータスを確認する頻度を設定します。デフォルトの間隔は 0 秒です。unknownStateInterval は、CA に証明書のステータスを確認する手段がない (情報へのアクセスが許可されていない) 場合に、キャッシュ結果が true であると見なされる猶予期間です。
- ***revocationChecking.validityInterval***。キャッシュされた証明書が有効とみなされる期間を設定します。間隔を選択するときは慎重に行ってください。たとえば、有効期間が 60 秒の場合、サーバーは 1 分ごとにキャッシュ内の証明書を破棄し、ソースから証明書を取得しようとします。Certificate Manager は内部データベースを使用して、証明書の失効ステータスを取得して確認します。デフォルトの有効期間は 120 秒 (2 分) です。

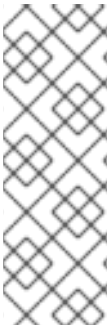
4. Certificate System インスタンスを起動します。

```
# pki-server start instance_name
```

14.4.1.3. サブシステムの証明書失効チェックの有効化

Certificate System サブシステムには、デフォルトで OCSP チェックを有効にしてサブシステム証明書を検証します。つまり、失効した証明書を使用して管理インターフェイスまたはエージェントインターフェイスにログインすることができます。

server.xml ファイルを編集して、すべてのサブシステムに対して OCSP チェックを有効にできます。エージェントインターフェイスと管理インターフェイスは別々に設定されるため、この設定で両方のセクションを編集する必要があります。



注記

サブシステムが内部データベースで SSL/TLS 接続を使用するように設定されている場合は、LDAP 内部データベースの SSL/TLS サーバー証明書が OCSP レスポンダーによって認識される必要があります。OCSP レスポンダーが LDAP サーバー証明書を認識しない場合は、サブシステムが適切に起動しません。この設定は、サブシステムの SSL/TLS サーバー接続がサブシステム設定の一部として設定されているため、『Red Hat Certificate System 10 のプランニング、インストール、およびデプロイメントガイド』で説明されています。

1. 証明書ステータスの確認に使用される OCSP または CA の OCSP 署名証明書の名前を取得します。以下に例を示します。

```
# 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. サブシステムの **server.xml** ファイルを開きます。以下に例を示します。

```
# vim /etc/pki/instance-name/server.xml
```

3. OCSP 署名証明書がインスタンスのセキュリティーデータベースにない場合は、インポートします。

```
# certutil -d /etc/pki/instance-name/alias -A -n "ocspSigningCert cert-pki-ca" -t "C,," -a -i
ocspCert.b64
```

4. OCSP チェックを有効にするには、以下の 3 つの重要なパラメーターがあります。

- ***enableOCSP***: OCSP チェックを有効にするには、**true** に設定する必要があります。

  これはグローバル設定です。単一のインターフェイスに設定されている場合には、インスタンスのすべてのインターフェイスに適用されます。ただし、通常は、**server.xml** ファイルに記載されている最初のインターフェイスで設定する必要があります。別のインターフェイスの設定は無視されます。

- ***ocspResponderURL***: OCSP レスポンダーの URL に OCSP リクエストを送信します。

  OCSP Manager の場合は、別の OCSP または CA の別の OCSP サービスになります。その他のサブシステムでは、これは常に OCSP または CA の外部 OCSP サービスを参照します。

- ***ocspResponderCertNickname***: レスポンスの署名に使用する署名証明書を提供します。CA OCSP サービスの場合、これは CA の OCSP 署名証明書であり、OCSP レスポンダーについては OCSP 署名証明書になります。

その他のパラメーターを使用して OCSP 通信を定義できます。OCSP チェックパラメーターはすべて 表14.10「server.xml の OCSP パラメーター」 に記載されています。

エージェントインターフェイスと管理者インターフェイス用に、ファイルには 2 つの異なるセクションがあります。OCSP チェックを有効にして設定するには、両方のセクションに OCSP パラメーターを追加する必要があります。以下に例を示します。

例14.3 エージェントインターフェイスの OCSP 設定

```
<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. 指定の OCSP サービスが CA ではない場合は、OCSP サービスの署名証明書がサブシステムの NSS データベースにインポートする必要があります。これは、コンソールまたは **certutil** を使用して行うことができます。いずれのオプションも、『Red Hat Certificate System 管理ガイド』 の 『Certificate System データベース への Certificates のインストール 』 で説明されています。

6. サブシステムを再起動します。

```
# pki-server restart instance_name
```

表14.10 server.xml の OCSP パラメーター

パラメーター	説明
enableOCSP	サブシステムの OCSP チェックを有効 (または無効) にします。
ocspResponderURL	OCSP リクエストが送信される URL を設定します。OCSP Manager の場合は、別の OCSP または CA の別の OCSP サービスになります。TKS または KRA の場合、これは常に OCSP または CA の外部 OCSP サービスを指します。

パラメーター	説明
ocspResponderCertNickname	レスポンダーの署名証明書のニックネームを設定します。OCSP 署名証明書または CA の OCSP 署名証明書のいずれかです。証明書はサブシステムの NSS データベースにインポートされ、適切な信頼設定が設定されている必要があります。
ocspCacheSize	キャッシュエントリーの最大数を設定します。
ocspMinCacheEntryDuration	別のフェッチを試行するまでの最小秒数を設定します。たとえば、これが 120 に設定された場合は、最後の有効期間をチェックから 2 分以上経たないと証明書の有効性を再度確認することができません。
ocspMaxCacheEntryDuration	次のフェッチを試みる前に待機する最大秒数を設定します。これにより、有効期間チェック間でウィンドウが大きくなり過ぎないようにできます。
ocspTimeout	OCSP 要求のタイムアウト期間を秒単位で設定します。

14.4.1.4. AIA 拡張機能の登録プロファイルへの追加

外部 OCSP を使用する際にプロファイルに AIA URL を設定するには、証明書プロファイルに正しい URL を追加します。以下に例を示します。

```
policyset.cmcUserCertSet.5.default.params.authInfoAccessADLocation_0=http://example.com:8080/ocsp/ee/ocsp
```

14.4.2. セッションのタイムアウト

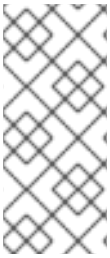
ユーザーがクライアントアプリケーションを介して PKI サーバーに接続すると、サーバーはユーザーを追跡するセッションを作成します。ユーザーがアクティブな状態である限り、ユーザーは再認証せずに同じセッションで複数の操作を実行できます。

セッションタイムアウトは、アクティブでないためにセッションを終了するまで最後の操作からサーバーが待機する期間を決定します。セッションが終了すると、ユーザーはサーバーへのアクセスを継続するためにユーザーを再認証し、サーバーが新しいセッションを作成します。

タイムアウトには、2 つのタイプがあります。

- TLS セッションのタイムアウト
- HTTP セッションのタイムアウト

クライアントの仕組みが異なるため、クライアントはこれらのタイムアウトによって影響を及ぼします。



注記

特定のクライアントには独自のタイムアウト設定があります。たとえば、Firefox には keep-alive タイムアウト設定があります。詳細は、http://kb.mozillazine.org/Network.http.keep-alive.timeout を参照してください。値が TLS セッションタイムアウトまたは HTTP セッションタイムアウトのサーバーの設定と異なる場合は、異なる動作を確認できます。

14.4.2.1. TLS セッションのタイムアウト

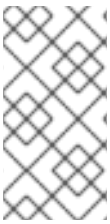
TLS セッションは、TLS ハンドシェイクプロトコルを介して確立された TLS 接続におけるセキュアな通信チャネルです。

PKI サーバーは、TLS セッションアクティビティーの監査イベントを生成します。サーバーは、接続の作成時に、**Outcome=Success** の **ACCESS_SESSION_ESTABLISH** 監査イベントを生成します。接続の作成に失敗した場合、サーバーは **Outcome=Failure** を指定した **ACCESS_SESSION_ESTABLISH** 監査イベントを生成します。接続が閉じられると、サーバーは **ACCESS_SESSION_TERMINATED** 監査イベントを生成します。

TLS セッションタイムアウト (TLS 接続タイムアウト) は、**/etc/pki/<instance>/server.xml** ファイルの **Secure** <Connector> 要素の **keepAliveTimeout** パラメーターで設定されます。

```
...
<Server>
 <Service>
  <Connector name="Secure"
   ...
   keepAliveTimeout="300000"
   ...
   />
 </Service>
</Server>
...
```

デフォルトではタイムアウト値は 300000 ミリ秒 (5 分) に設定されます。この値を変更するには、**/etc/pki/<instance>/server.xml** ファイルを編集し、サーバーを再起動します。

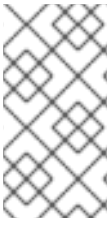


注記

この値は、サーバーへのすべての TLS 接続に影響することに注意してください。期限切れでない既存の接続を再利用できるため、クライアントの効率が大きくなる可能性があります。ただし、放棄された接続の有効期限が切れるまでに時間がかかるため、サーバーが同時にサポートする必要のある接続の数も増える可能性があります。

14.4.2.2. HTTP セッションのタイムアウト

HTTP セッションは、HTTP クッキーを使用して複数の HTTP リクエストでユーザーを追跡するメカニズムです。PKI サーバーは、HTTP セッションの監査イベントを生成しません。



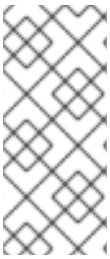
注記

一貫性を監査するために、本セクションの **<session-timeout>** 値を、「TLS セッションのタイムアウト」 の **keepAliveTimeout** 値と一致するように設定します。たとえば、**keepAliveTimeout** が 300000 (5 分) に設定されている場合は、**<session-timeout>** を 30 に設定します。

HTTP セッションのタイムアウトは、**/etc/pki/<instance>/web.xml** ファイルの **<session-timeout>** 要素で設定できます。

```
...
<web-app>
 <session-config>
  <session-timeout>30</session-timeout>
 </session-config>
</web-app>
...
```

デフォルトでは、タイムアウト値は 30 分に設定されています。値を変更するには、**/etc/pki/<instance>/web.xml** ファイルを編集し、サーバーを再起動します。



注記

この値は、サーバー上のすべての Web アプリケーションにあるすべてのセッションに影響することに注意してください。アクセスバナーを再度認証したり、表示したりする必要がないため、大きな値がユーザーのエクスペリエンスが向上します。ただし、破棄された HTTP セッションが期限切れになるまでの時間があるため、セキュリティーリスクも高まります。

14.4.2.3. PKI Web UI のセッションタイムアウト

PKI Web UI は、ブラウザーで実行されるインタラクティブな Web ベースのクライアントです。現在、クライアント証明書認証のみをサポートしています。

Web UI が開くと、ブラウザーはサーバーに複数の TLS 接続を作成できます。これらの接続は単一の HTTP セッションに関連付けられます。

Web UI のタイムアウトを設定するには、「HTTP セッションのタイムアウト」を参照してください。ブラウザーがクライアント証明書をキャッシュし、TLS セッションを自動的に再作成できるようにするため、TLS セッションのタイムアウトは通常無関係です。

HTTP セッションの期限が切れると、Web UI は即座に表示されません。ただし、Web UI は、ユーザーが操作を実行する前にアクセスバナー (有効な場合) を表示します。

14.4.2.4. PKI コンソールのセッションタイムアウト

PKI コンソールは、インタラクティブなスタンドアロングラフィカル UI クライアントです。ユーザー名/パスワードおよびクライアント証明書認証をサポートします。

コンソールの起動時に、サーバーへの 1 つの TLS 接続が作成されます。コンソールは、グラフィカルインターフェイスを開く前に、アクセスバナー (有効な場合) を表示します。Web UI とは異なり、コンソールはサーバーで HTTP セッションを維持しません。

コンソールのタイムアウトを設定するには、「TLS セッションのタイムアウト」を参照してください。コンソールは HTTP セッションを使用しないため、HTTP セッションタイムアウトは無関係です。

TLS セッションの期限が切れると、TLS 接続が閉じられ、コンソールがシステムにすぐに終了します。ユーザーが続行する場合は、コンソールを再起動する必要があります。

14.4.2.5. PKI CLI のセッションタイムアウト

PKI CLI は、一連の操作を実行するコマンドラインクライアントです。ユーザー名/パスワードおよびクライアント証明書認証をサポートします。

CLI の起動時に、サーバーと HTTP セッションへの単一の TLS 接続を作成します。CLI は、操作を実行する前にアクセスバナー (有効な場合) を表示します。

通常、操作は遅延なく順次実行され、CLI が完了するとすぐに CLI を終了するため、両方のタイムアウトは PKI CLI とは無関係です。ただし、CLI がユーザー入力を待機するか、速度が低下するか、応答しなくなると、TLS セッションまたは HTTP セッションが期限切れになり、残りの操作が失敗する場合があります。このような遅延が予想される場合は、「TLS セッションのタイムアウト」 と 「HTTP セッションのタイムアウト」 を参照して、想定される遅延に対応します。

14.5. WEB.XML

14.5.1. web.xml からの未使用インターフェイスの削除 (CA のみ)

(一括発行やポリシーフレームワークなどの機能用の) レガシーインターフェイスは、CA の **web.xml** ファイルに依然として含まれています。ただし、これらの機能は非推奨となり、使用されなくなりました。したがって、CA 設定から削除してセキュリティーを強化することができます。

1. CA を停止します。

```
# pki-server stop instance_name
```

   または (**nuxwdog watchdog** を使用している場合)

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

2. CA の Web ファイルディレクトリーを開きます。以下に例を示します。

```
# cd /var/lib/pki/instance_name/ca/webapps/ca/WEB-INF
```

3. 現在の **web.xml** ファイルをバックアップします。

```
# cp web.xml web.xml.servlets
```

4. **web.xml** ファイルを編集し、非推奨となった以下の各サーブレットの **<servlet>** エントリー全体を削除します。

   - caadminEnroll
   - cabulkissuance
   - cacertbasedenrollment
   - caenrollment
   - caProxyBulkIssuance

   たとえば、**caadminEnroll** サーブレットエントリーを削除します。

```
<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. サーブレットエントリーを削除したら、該当する **<servlet-mapping>** エントリーを削除します。

```
<servlet-mapping>
    <servlet-name> caadminEnroll </servlet-name>
    <url-pattern>  /admin/ca/adminEnroll </url-pattern>
</servlet-mapping>
```

6. エンドエンティティー要求インターフェイスについて、**<filter-mapping>** エントリーを 3 つ削除します。

```
  <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. CA を再度起動します。

```
# pki-server start instance_name
```

または (**nuxwdog watchdog** を使用している場合)

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

14.6. WEB サービスのカスタマイズ

すべてのサブシステム (TKS を除く) には、エージェント用の Web ベースのサービスページや、その他のロール (管理者やエンドエンティティーなど) 用の Web ベースのサービスページがあります。これらの Web ベースのサービスページは基本的な HTML や JavaScript を使用しており、既存のサイトやイントラネットに合わせて、さまざまな色、ロゴ、その他のデザイン要素を使用するようにカスタマイズできます。

14.6.1. サブシステム Web アプリケーションのカスタマイズ

各 PKI サブシステムには以下が含まれる対応する web アプリケーションがあります。

- テキスト、JavaScript コード、ページレイアウト、CSS フォーマットなどを含む HTML ページ
- サーブレット、パス、セキュリティー制約などを定義する **web.xml** ファイル
- PKI ライブラリーへのリンク

サブシステムの Web アプリケーションは、**/var/lib/pki/pki-tomcat/conf/Catalina/localhost/** のディレクトリーにあるコンテキストファイル (**ca.xml** ファイル など) を使用してデプロイされます。

```
<Context docBase="/usr/share/pki/ca/webapps/ca" crossContext="true" allowLinking="true">
    ...
</Context>
```

docBase は、デフォルトの Web アプリケーションディレクトリー **/usr/share/pki/** の場所を参照します。

Web アプリケーションをカスタマイズするには、Web アプリケーションディレクトリーをインスタンスの **webapps** ディレクトリーにコピーします。

```
$ cp -r /usr/share/pki/ca/webapps/ca /var/lib/pki/pki-tomcat/webapps
```

次に、**webapps** ディレクトリーから相対的なカスタム Web アプリケーションのディレクトリーを参照するように、**docBase** を変更します。

```
<Context docBase="ca" crossContext="true" allowLinking="true">
    ...
</Context>
```

変更は、サーバーを再起動しなくてもすぐに有効になります。

カスタム Web アプリケーションを削除するには、**docBase** を元に戻し、カスタムの Web アプリケーションディレクトリーを削除します。

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/ca
```

14.6.2. Web UI テーマのカスタマイズ

同じインスタンスのサブシステムの web アプリケーションは、同じテーマを共有します。これには以下が含まれます。

- CSS ファイル (グローバルの外観を決定する)
- ロゴ、アイコン、およびその他のイメージファイル
- ページタイトル、ロゴリンク、タイトルの色などを判断するブランディングプロパティー

Web UI のテーマは、**/var/lib/pki/pki-tomcat/conf/Catalina/localhost/** ディレクトリーの **pki.xml** コンテキストファイルを使用してデプロイされます。

```
<Context docBase="/usr/share/pki/common-ui" crossContext="true" allowLinking="true">
    ...
</Context>
```

docBase は、デフォルトのテーマディレクトリー **/usr/share/pki/** の場所を参照します。

テーマをカスタマイズするには、デフォルトのテーマディレクトリーをインスタンスの **webapps** ディレクトリーにある **pki** ディレクトリーにコピーします。

```
$ cp -r /usr/share/pki/common-ui /var/lib/pki/pki-tomcat/webapps/pki
```

次に、**webapps** ディレクトリーから相対的なカスタムテーマのディレクトリーを参照するように、**docBase** を変更します。

```
<Context docBase="pki" crossContext="true" allowLinking="true">
    ...
</Context>
```

変更は、サーバーを再起動しなくてもすぐに有効になります。

カスタムテーマを削除するには、**docBase** を元に戻して、カスタムテーマのディレクトリーを削除します。

```
$ rm -rf /var/lib/pki/pki-tomcat/webapps/pki
```

14.6.3. TPS トークンの状態ラベルのカスタマイズ

デフォルトのトークンの状態ラベルは **/usr/share/pki/tps/conf/token-states.properties** ファイルに保存され、「トークンの状態および遷移ラベル」 で説明されています。

ラベルをカスタマイズするには、ファイルをインスタンスディレクトリーにコピーします。

```
$ cp /usr/share/pki/tps/conf/token-states.properties /var/lib/pki/pki-tomcat/tps/conf
```

変更は、サーバーを再起動しなくてもすぐに有効になります。

カスタマイズされたラベルを削除するには、カスタマイズされたファイルを削除するだけです。

```
$ rm /var/lib/pki/pki-tomcat/tps/conf/token-states.properties
```

14.7. アクセスバナーの使用

Certificate System では、管理者はカスタマイズ可能なテキストでバナーを設定できます。以下の状況ではバナーが表示されます。

アプリケーション	バナーが表示されるタイミング
PKI コンソール	• コンソールが表示される前に、 • セッションが期限切れになる後。[a]
Web インターフェイス	• Web インターフェイスに接続する場合。 • セッションの有効期限が切れた後。[a]
pki コマンドラインユーティリティー	• 実際の操作を開始する前。
[a] セッションタイムアウトの変更に関する詳細は、「セッションのタイムアウト」を参照してください。	

バナーを使用すると、Certificate System を使用する前にユーザーに重要な情報を表示できます。続行するには、表示されたテキストに同意する必要があります。

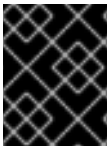
例14.4 アクセスバナーの表示時

以下の例は、**pki** ユーティリティーを使用している場合にアクセスバナーが表示されるタイミングを示しています。

```
$ 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. アクセスバナーの有効化

アクセスバナーを有効にするには、**/etc/pki/*instance_name*/banner.txt** ファイルを作成し、表示するテキストを入力します。



重要

/etc/pki/*instance_name*/banner.txt ファイルのテキストは、UTF-8 形式を使用する必要があります。検証するには、「バナーの検証」を参照してください。

14.7.2. アクセスバナーの無効化

アクセスバナーを無効にするには、**/etc/pki/*instance_name*/banner.txt** ファイルを削除するか、または名前を変更します。以下に例を示します。

```
# mv /etc/pki/instance_name/banner.txt /etc/pki/instance_name/banner.txt.UNUSED
```

14.7.3. バナーの表示

現在設定されているバナーを表示するには、次のコマンドを実行します。

```
# pki-server banner-show -i instance_name
```

14.7.4. バナーの検証

バナーに無効な文字が含まれていないことを確認するには、次のコマンドを実行します。

```
# pki-server banner-validate -i instance_name
---------------
Banner is valid
---------------
```

14.8. CMC の設定

本セクションでは、CMS (CMC) で証明書管理に Certificate System を設定する方法を説明します。

14.8.1. CMC の仕組み

CMC を設定する前に、以下のドキュメントを参照してこのトピックの詳細を確認してください。

- 「CMC を使用した登録」
- 『Red Hat Certificate System 管理ガイド』の『CMC を使用した証明書の発行』
- 『Red Hat Certificate System 管理ガイド』の『証明書 (証明書プロファイル) を発行するルールの作成』

14.8.2. PopLinkWittnessV2 機能の有効化

認証局 (CA) の高レベルのセキュリティーの場合は、**/var/lib/pki/*instance_name*/ca/conf/CS.cfg** ファイルで以下のオプションを有効にします。

```
cmc.popLinkWitnessRequired=true
```

14.8.3. CMC 共有シークレット機能の有効化

認証局 (CA) で共有トークン機能を有効にするには、以下を実行します。

1. ホストでウォッチドッグサービスが有効になっている場合は、そのサービスを一時的に無効にします。「Watchdog サービスの無効化」を参照してください。

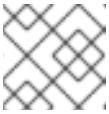
2. Directory Server のスキーマに **shrTok** 属性を追加します。

```
# 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. システムキーが Hardware Security Module (HSM) に保存されている場合は、**/var/lib/pki/*instance_name*/ca/conf/CS.cfg** ファイルに ***cmc.token*** パラメーターを設定します。以下に例を示します。

```
cmc.token=NHSM-CONN-XC
```

4. 以下の方法のいずれかを使用して、共有トークン認証プラグインを有効にします。

- 

  注記

  pkiconsole が非推奨になりました。

  pkiconsole ユーティリティーを使用してプラグインを有効にするには、次のコマンドを実行します。

  1. **pkiconsole** ユーティリティーを使用してシステムにログインします。以下に例を示します。

     ```
     # pkiconsole https:host.example.com:8443/ca
     ```

  2. **Configuration** タブで、**Authentication** を選択します。

  3. **Add** をクリックして、**SharedToken** を選択します。

  4. **Next** をクリックします。

  5. 以下の情報を入力します。

     ```
     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. **OK** をクリックします。

- プラグインを手動で有効にするには、以下の設定を **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** ファイルに追加します。

```
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. **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** ファイルの ***ca.cert.issuance_protection.nickname*** パラメーターで、RSA 発行保護証明書のニックネームを設定します。以下に例を示します。

```
ca.cert.issuance_protection.nickname=issuance_protection_certificate
```

このステップは以下のとおりです。

- ***ca.cert.subsystem.nickname*** パラメーターで RSA 証明書を使用する場合はオプションになります。
- ***ca.cert.subsystem.nickname*** パラメーターで ECC 証明書を使用する場合に必須です。

重要

ca.cert.issuance_protection.nickname パラメーターが設定されていない場合、Certificate System は ***ca.cert.subsystem.nickname***で指定されたサブシステムの証明書を自動的に使用します。ただし、発行保護証明書は RSA 証明書である必要があります。

6. Certificate System を再起動します。

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

CA が起動すると、Certificate System は Shared Token プラグインが使用する LDAP パスワードを要求します。

7. この手順の最初に watchdog サービスを一時的に無効にした場合は、再度有効にします。「Watchdog サービスの有効化」を参照してください。

14.8.4. Web ユーザーインターフェイスの CMCRevoke の有効化

『Red Hat Certificate System 管理ガイド』の『CMC 失効の実行』セクションで説明されているように、失効リクエストを送信する方法は 2 つあります。

CMCRevoke ユーティリティーを使用して、Web UI で送信される失効要求を作成する場合は、以下の設定を **/var/lib/pki/*instance_name*/ca/conf/CS.cfg** ファイルに追加します。

```
cmc.bypassClientAuth=true
```

14.9. CA EE PORTAL を使用した証明書の登録のサーバー専用鍵生成の設定

本セクションでは、CA EE ポータルを使用して、証明書登録の Server-Side Key Generation を設定する方法を説明します。

14.9.1. インストール設定

注記

サーバー側 Keygen を設定するには、CA に加えて KRA インスタンスが必要です。

注記

CA と KRA が Tomcat インスタンスを共有している場合は、以下の手順を実行してトランスポート証明書をインポートする必要はありません。

CA インスタンスおよび KRA インスタンスをインストールした後に、スタンドアロンの Tomcat Web サーバーインスタンスの場合には、CA の nssdb に KRA トランスポート証明書を追加する必要があります。

1. まず、CA を停止します。

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

   以下に例を示します。

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

2. CA の **CS.cfg** に以下の行を追加します。

   ```
   ca.connector.KRA.transportCertNickname=KRA transport cert
   ```

   以下に例を示します。

   ```
   ca.connector.KRA.transportCertNickname=transportCert cert-topology-02-KRA KRA
   ```

   topology-02-KRA は、KRA のインスタンスの名前に置き換えます。

3. KRA トランスポート証明書を検索し、ファイルにエクスポートします。

   ```
   # grep "kra.transport.cert=" /var/lib/pki/kra_instance_name/kra/conf/CS.cfg | sed
   's/kra.transport.cert=//' > kra transport cert file
   ```

以下に例を示します。

```
# grep "kra.transport.cert=" /var/lib/pki/pki-kra/kra/conf/CS.cfg | sed 's/kra.transport.cert=//' >
/tmp/kraTransport.cert
```

4. CA の **CS.cfg** ファイルで指定されているニックネームを使用して、KRA トランスポート証明書を CA の nssdb にインポートします。

   1. トランスポート証明書のニックネームを一覧表示します。

```
grep "ca.connector.KRA.transportCertNickname"
/var/lib/pki/ca_instance_name/ca/conf/CS.cfg
```

   以下に例を示します。

```
# grep "ca.connector.KRA.transportCertNickname" /var/lib/pki/pki-ca/ca/conf/CS.cfg
        ca.connector.KRA.transportCertNickname=KRA transport cert
```

   - 前の手順に記載されているニックネームを使用して証明書をインポートします。

```
certutil -d /var/lib/pki/ca_instance_name/alias -A -t “,,” -n transportNickName -i kra transport
cert file
```

   以下に例を示します。

```
# certutil -d /var/lib/pki/pki-ca/alias -A -t “,,” -n "KRA transport cert" -i /tmp/kraTransport.cert
```

5. CA を起動します。

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

   以下に例を示します。

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

14.9.2. プロファイル設定

サーバー側で鍵が生成される証明書の登録を許可するために、**caServerKeygen_UserCert** および **caServerKeygen_DirUserCert** の 2 つのデフォルトプロファイルがデフォルトで提供されます。ただし、正しい入力、出力、およびポリシー設定を持つプロファイルは、サーバー側の keygen プロファイルに切り替えることができます。

Server-Side Keygen プロファイルには、以下のコンポーネントが含まれている必要があります。

入力

```
input.i1.class_id=serverKeygenInputImpl
```

出力

```
output.o1.class_id=pkcs12OutputImpl
```

Policyset
キータイプとキーサイズのパラメーターは、以下に例示するように設定できます。

```
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,nist
p521
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
```

認証
デフォルトのサーバー側の keygen 登録プロファイルは認証メカニズムによって異なります。ここでは、以下のようになります。

- **caServerKeygen_UserCert.cfg**

  "auth.class_id=" への空の値が含まれます。つまり、このプロファイルを介した登録要求には CA エージェントからの承認が必要になります。

- **caServerKeygen_DirUserCert.cfg**

  これには、**"auth.instance_id=UserDirEnrollment"** が含まれます。これは、ユーザーが LDAP uid/password 認証を渡すのに必要なことを意味します。このような認証は、CA エージェントからの要求ごとの承認を必要としないため、自動証明書として見なされます。

  caServerKeygen_DirUserCert.cfg プロファイルで説明されるように、**auth.instance_id** ディレクティブを互換性のある認証プラグインクラスに設定すると、自動承認を設定できます。**CS.cfg** ファイルでこのような設定の例を、以下に示します。

  ```
  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. 証明書の透過性の設定

Certificate System は、証明書 Certificate Transparency (CT) V1 サポートの基本バージョン (rfc 6962) を提供します。各デプロイメントサイトがルート CA 証明書を含めることを選択した信頼できるログから、Signed Certificate Time スタンプ (SCT) が埋め込まれた証明書を発行する機能があります。また、複数の CT ログに対応するようにシステムを設定することもできます。この機能を使用するには、少なくとも 1 つの信頼できる CT ログが必要です。

重要

デプロイメントサイトが、信頼できる CT ログサーバーとの信頼関係を確立します。

証明書の透過性を設定するには、**/var/lib/pki/*instance name*/ca/conf/CS.cfg** ディレクトリーにある CA の **CS.cfg** ファイルを編集します。

証明書の透過性設定をテストする方法は、『Red Hat Certificate System 計画、インストール、およびデプロイメントのガイド』の『証明書の透明性のテスト』セクションを参照してください。

14.10.1. ca.certTransparency.mode

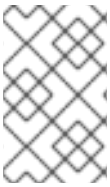
ca.certTransparency.mode は、3 つの証明書の透過性モードのいずれかを指定します。

- **disabled**: 発行した証明書には SCT 拡張がありません。
- **enabled**: 発行した証明書に SCT 拡張が含まれます。
- **perProfile**: 以下の policyset が含まれるプロファイルで登録された証明書は SCT 拡張を保持します: SignedCertificateTimestampListExtDefaultImpl

デフォルトの値は 0 です。

14.10.2. ca.certTransparency.log.num

ca.certTransparency.log.num は、設定で定義された CT ログの合計数を指定します。



注記

定義されている CT ログエントリーがすべてアクティブとみなされているわけではありません。「ca.certTransparency.log.<id>.*」 の **ca.certTransparency.log.<id>.enable** を参照してください。

14.10.3. ca.certTransparency.log.<id>.*

ca.certTransparency.log.<id>.* は、ログ <id> に関する情報を指定します。ここで、<id> は CT ログサーバーに割り当てる一意の ID で、他の CT ログと区別します。

パラメーター名はそれぞれの **ca.certTransparency.log.<id>.** に準拠しており、<id> に属します。

- **ca.certTransparency.log.<id>.enable** は、<id> CT ログが有効 (true) または無効 (false) であるかを指定します。
- **ca.certTransparency.log.<id>.pubKey** には、CT ログの公開鍵の base64 エンコーディングが含まれます。
- **ca.certTransparency.log.<id>.url** には、CT ログ URL の base64 エンコーディングが含まれます。
- **ca.certTransparency.log.<id>.version** は、 (CT ログサーバーと同様) CT がサポートする CT バージョン番号を指定します。現在、バージョン 1 のみをサポートしています。

第15章 証明書/キー暗号トークンの管理

本章では、さまざまなシナリオの証明書のインポートおよび検証方法など、暗号化トークンで証明書/鍵のデータベースを管理する方法を説明します。

暗号トークンについて
NSS ソフトトークンの詳細は、「NSS Soft Token (内部トークン)」を参照してください。

HSM の詳細は、「ハードウェアセキュリティーモジュール (HSM、外部トークン)」を参照してください。

15.1. CERTUTIL および PKICERTIMPORT

certutil コマンドは、Network Security Services (NSS) によって提供されます。 **certutil** は、証明書の検証およびインポートに使用されます。ただし、**certutil** の使用方法に関する基本的な概要を以下に示します。ただし、**PKICertImport** は、証明書を安全に検証およびインポートするために選択するためのラッパースクリプトです。これには、**certutil** を使用して、複数のコマンドの呼び出しが必要で、正しい使用方法は本書の範囲外です。

15.1.1. certutil の基本的な使用法

certutil [command] [options]

各 **certutil** 呼び出しはコマンドフラグを取ります。通常は大文字で表されます。また、コマンドの動作を制御する一連のオプションを利用できます。オプションが値を取る場合、その値の名前は<と>間に付けられます。

15.1.2. PKICertImport の基本的な使用方法

PKICertImport [options]

各 **PKICertImport** 呼び出しは、指定された証明書を検証およびインポートする一連のオプションを取ります。**certutil** の広範なユースケースとは異なり、**PKICertImport** は、証明書を安全にインポートおよび検証することのみに重点を置いています。利用可能なオプションの詳細は、「一般的な **certutil** および **PKICertImport** オプション」を参照してください。

注記

PKICertImport は、実行中に NSS DB または HSM のパスワードを複数回要求します。これは、**PKICertImport** と NSS DB との対話を複数回使用する必要があるためです。NSS DB パスワードを繰り返し入力しなくてもよいように、**-f <filename>** でパスワードファイルを指定します。完了したら、パスワードファイルを必ず削除してください。

15.1.3. certutil の一般的なコマンド

以下のコマンドは **certutil** 固有のもので、いくつかの一般的なコマンドの概要を示します。**PKICertImport** は、これらのコマンドフラグと互換性がなく、これらのコマンドフラグも必要ありません。

certutil -A
-A コマンドは、証明書の追加を示しています。インポートするには証明書 (**-i**)、その証明書のニックネーム (**-n**)、および証明書の一連の信頼フラグ (**-t**) が必要です。

certutil -V
-V コマンドは、証明書の検証を示しています。検証には、証明書のニックネーム (**-n**) と、実行する検証 (**-u**) のタイプが必要です。

certutil -D
-D コマンドは、証明書の削除を示しています。削除する証明書のニックネーム (**-n**) が必要です。

証明書の公開鍵部分のみが削除され、秘密鍵が存在する場合は削除されないことに注意してください。

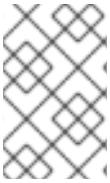
certutil -M
-M コマンドは、証明書の変更を示しています。変更する証明書ニックネーム (**-n**) と、証明書を提供する一連の信頼フラグ (**-t**) が必要です。

certutil -L
-L コマンドは、証明書またはすべての証明書へのリストを示しています。ニックネームオプション (**-n**) を指定すると、その証明書に関する詳細情報が表示されます。それ以外の場合は、存在するすべての証明書に関する一般的な情報が一覧表示されます。

certutil -L の結果として、各証明書のニックネームとその信頼情報が表示されます。以下に例を示します。

表15.1 証明書のニックネームと信頼情報

証明書のニックネーム	Trust Attributes SSL, S/MIME, JAR/XPI
caSigningCert pki-ca1	CT、C、C



注記

certutil -L で表示される信頼属性は、**-t** オプションで指定した内容に対応します。

certutil -L はデータベースを変更しないため、必要な回数だけ安全に実行できます。

15.1.4. 一般的な certutil および PKICertImport オプション

以下の手順に従って、値は特定のデプロイメントシナリオに適し、正しい値であることを確認します。これらのオプションは **PKICertImport** でも利用できます。

-n <nickname>
-n <nickname> オプションは、証明書のニックネームを指定します。これは任意のテキストを指定でき、証明書への参照としてのみ使用されます。一意である必要があります。

設定に応じて、この値を更新してください。

-d <directory>
-d <directory> オプションは、使用中の NSS DB ディレクトリーへのパスを指定します。通常、このディレクトリーはすでに存在し、.を使用して現在のディレクトリーを参照します。

設定に応じて、この値を更新してください。

-t <trust>
-t <trust> オプションは、証明書の信頼レベルを指定します。

信頼には、以下の 3 つのカテゴリーがあります。

- TLS の信頼
- メールの信頼
- オブジェクト署名の信頼

各信頼の位置には、信頼のレベルを指定する信頼文字を 1 つまたは複数追加できます。以下のトラスト文字は、**c**、**C** および **T** です。

- **c** は、この証明書が認証局 (CA) である必要があります。
- **C** は、サーバー証明書に署名するための信頼できる認証局であることを示しています (**C** は小文字の **c** であるため、両方を指定する必要はありません)。
- **T** は、この証明書が、クライアント証明書に署名するための信頼できる機関であることを示しています (**T** は小文字の **c** であるため、**T** と **c** の両方を指定する必要はありません)。

各位置の信頼フラグを指定するには、文字をコンマで区切ります。たとえば、オプション **-t CT,C,c** は、クライアントとサーバーの TLS 証明書への署名、サーバーの電子メール証明書 (S/MIME) への署名に信頼されており、オブジェクトの署名に有効な CA (信頼されていませんが) であることを意味します。

- これにより、この証明書が別の証明書に署名し、それがオブジェクト署名に使用されると、その証明書が無効になります。

信頼なし (または信頼の欠如) は、**-t ,,** を使用して指定できます。

データベース内のすべての証明書のトラストレベルを表示するには、以下を実行します。

- **certutil -L -d**
- 各証明書のニックネームが一覧表示され、行の最後に信頼フラグが指定されます。

-h オプションの HSM に関する注記を参照してください。

信頼レベルは、certutil の man ページで指定されていることに注意してください。このドキュメントを参照するには、certutil が正しくインストールされているシステムで **man certutil** コマンドを実行します。

-h <HSM>
-h <HSM> オプションは、操作を実行する HSM の名前を指定します。

HSM は信頼を保管できないため、**-h** オプションは、**-t** オプションと互換性がありません。信頼を保存できるのは NSS DB だけなので、**-h <HSM>** と一緒に **certutil -A** コマンドや **certutil -M** コマンドを使用しても失敗します。代わりに、**-h** オプションなしで、別の **certutil -M** コマンドで必要な信頼レベルを指定します。

設定に応じて、この値を更新してください。

-e
-e オプションは、**certutil -V** コマンドとともに使用する場合に、署名の有効性もチェックされるように指定します。**PKICertImport** は、証明書署名の検証を常に実行し、**-e** オプションを理解しません。

-a
-a オプションは、問題のキーが PEM (ASCII) 形式であることを指定します。

-i <certificate>
-i <certificate> オプションは、証明書へのパスを指定します。これは、インポートする証明書へのパスを指定するために **certutil -A** コマンドで使用されます。

設定に応じて、この値を更新してください。

-u <usage>
-u <usage> オプションは、**certutil -V** コマンドと併用する際に、検証する証明書の使用を指定します。

次のセクションで参照されるいくつかの使用法の文字があります。

- **-u C** は、クライアントの TLS 証明書検証を表します。これは証明書を許可しますが、有効期限と署名を確認することに注意してください。
- **-u V** は、サーバーの TLS 証明書検証を表します。これにより CA 証明書が拒否され、有効期限と署名を確認することに注意してください。
- **-u L** は、CA TLS 証明書の検証に使用します。これにより信頼フラグが検証され (**c** が存在するか確認)、鍵の使用方法を確認して、キーが CA キーであることを確認します。これは、有効期限および署名も確認します。
- **-u O** は、OCSP ステータスレスポンダー証明書を検証することを表します。これにより、期限切れと署名を確認することに注意してください。
- **-u J** オブジェクト署名証明書の検証を表します。これにより、期限切れと署名を確認することに注意してください。

誤った使用方法オプションが指定されているり、証明書の信頼フラグが正しくない場合 (CA TLS 証明書の **c** フラグがないなど)、**certutil -V** は誤った結果となります。

注記

使用方法の詳細情報は、certutil の man ページに記載されています。このドキュメントを参照するには、certutil が正しくインストールされているシステムで **man certutil** コマンドを実行します。

15.2. ルート証明書のインポート

まず、ディレクトリーを NSS DB に変更します。

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

これらの手順の実行中に Web サービスがオフラインになり (停止、無効化など) し、他のプロセス (ブラウザーなど) による NSS DB への同時アクセスがないことを確認します。これにより、NSS DB が破損したり、これらの証明書の使用が不適切になる場合があります。

新しいルート証明書をインポートする必要がある場合は、証明書がいくつもの証明書に署名できるため、安全な方法でこの証明書を取得するようにしてください。**ca_root.crt** という名前のファイルにすでに存在していることを前提とします。お好みのシナリオに合わせて、正しい名前とパスを置き換えます。

以下の **certutil** オプションおよび **PKICertImport** オプションの詳細は、「**certutil** および **PKICertImport**」 を参照してください。

ルート証明書をインポートするには、次のコマンドを実行します。

- **PKICertImport -d . -n "CA Root" -t "CT,C,C" -a -i ca_root.crt -u L** コマンドを実行します。

  このコマンドは、ルート証明書を NSS DB に検証し、インポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。証明書は通常、有効期限が切れるか、または CA 証明書であるかから検証に失敗します。したがって、証明書ファイルが正しいことを確認し、最新の状態にしてください。発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

15.3. 中間証明書チェーンのインポート

開始する前に、ディレクトリーを NSS DB に変更します。

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

これらの手順の実行中に Web サービスがオフラインであることを確認します (停止、無効化など)、他のプロセス (ブラウザーなど) による NSS DB への同時アクセスがないことを確認します。これにより、NSS DB が破損したり、これらの証明書の使用が不適切になる場合があります。

ルート証明書をインポートして信頼していない場合は、「ルート証明書のインポート」を参照してください。

ルートサーバーとエンドサーバーまたはクライアント証明書間に一連の中間証明書が指定される場合は、ルート CA 証明書から最も適したように、署名済み証明書チェーンをインポートおよび検証する必要があります。中間 CA は **ca_sub_<num>.crt** という名前のファイル (例:**ca_sub_1.crt**、**ca_sub_2.crt** など) を想定しています。デプロイメントに合わせて、証明書の名前とパスを置き換えます。

注記

代わりに、**fullchain.crt**、**fullchain.pem**、または同様の名前の単一ファイルが与えられ、それには複数の証明書が含まれている場合、各ブロック (----BEGIN CERTIFICATE----- マーカーと -----END CERTIFICATE----- マーカーと、その間のテキストを含む) を独自のファイルにコピーして、それを上記のフォーマットに分割します。最初のものは **ca_sub_<num>.crt** という名前で、最後のコンポーネントは **service.crt** という名前のサーバーの証明書です。サーバー証明書については、以降のセクションで説明します。

まず、ルート CA 証明書から最も遠い順に、中間 CA をインポートして検証します。存在していない場合は、次のセクションに進みます。

以下の **certutil** オプションおよび **PKICertImport** オプションの詳細は、「**certutil** および **PKICertImport**」を参照してください。

チェーン内のすべての中間証明書に対して、以下を行います。

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

  このコマンドは、中間 CA 証明書を NSS DB に検証し、インポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

15.4. HSM への証明書のインポート

この手順では、CSR の作成プロセスと同じ HSM トークンでキーが生成された、新しく発行された証明書 (システム証明書の更新時など) を取得した後、証明書を HSM にインポートする方法を説明します。

開始する前に、ディレクトリーを NSS DB に変更します。

- **cd /path/to/nssdb** for example **cd /var/lib/pki/pki-ca/alias**

これらの手順の実行中に Web サービスがオフラインになり (停止、無効化など) し、他のプロセス (ブラウザーなど) による NSS DB への同時アクセスがないことを確認します。これにより、NSS DB が破損したり、これらの証明書の使用が不適切になる場合があります。

ルート証明書をインポートして信頼していない場合は、「ルート証明書のインポート」を参照してください。中間証明書をインポートおよび検証していない場合は、「中間証明書チェーンのインポート」を参照してください。

従う手順セットは、問題の証明書の使用により異なります。

- すべての PKI サブスキームの TLS サーバー証明書については、**サーバー証明書の手順** に従います。
- サブシステムの監査署名証明書については、以下の手順に従って **オブジェクト署名証明書** を検証してください。
- CA サブシステムの署名証明書については、上記の手順に従って **中間証明書チェーン** をインポートして検証しますが、**caSigningCert** でのみ行います。
- CA サブシステムの OCSP 署名証明書については、以下の手順に従って **OCSP 証明書** を検証してください。
- PKI サブシステムのその他のシステム証明書については、**Client Certificate の手順** に従います。

以下の **certutil** オプションおよび **PKICertImport** オプションの詳細は、「**certutil** および **PKICertImport**」 を参照してください。

HSM にサーバー証明書をインポートするには、以下を行います。

- **PKICertImport -d . -h HSM -n "host.name.example.com" -t "," -a -i service.crt -u V** を実行します。

  このコマンドは、サーバー証明書を検証して HSM にインポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。通常、親証明書の有効期限または CA 信頼チェーンの欠落 (中間証明書の欠落や CA ルートの欠落など) が原因で、証明書の検証に失敗します。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

HSM にクライアント証明書をインポートするには、以下を実行します。

- **PKICertImport -d . -h HSM -n "client name" -t "," -a -i client.crt -u C** を実行します。

  このコマンドは、クライアント証明書を HSM に検証し、インポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

HSM にオブジェクト署名証明書をインポートするには、以下を実行します。

- **PKICertImport -d . -h HSM -n "certificate name" -t ",,P" -a -i objectsigning.crt -u J** を実行します。

  このコマンドは、オブジェクト署名証明書を HSM に検証し、インポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

HSM で OCSP 応答署名証明書をインポートするには、次を実行します。

- **PKICertImport -d . -h HSM -n "certificate name" -t ",," -a -i ocsp.crt -u O** を実行します。

  このコマンドは、OCSP レスポンダー証明書を HSM に検証し、インポートします。エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

15.5. NSS データベースでの証明書のインポート

これらの手順の実行中に Web サービスがオフラインになり (停止、無効化など) し、他のプロセス (ブラウザーなど) による NSS データベースへの同時アクセスがないことを確認します。これにより、NSS データベースが破損したり、これらの証明書の使用が不適切になる場合があります。

従う手順セットは、問題の証明書の使用により異なります。

- 任意のサブシステムの **auditSigningCert** で、**オブジェクトの署名証明書** の検証に以下の手順に従ってください。
- CA サブシステムの **caSigningCert** については、**中間証明書チェーン** をインポートして検証するための上記の手順にいますが、caSigningCert でのみ行います。
- CA サブシステムの **ocspSigningCert** については、以下の手順に従って **OCSP 証明書** を検証する必要があります。
- ユーザーのクライアントまたは S/MIME 証明書の場合は、**Client Certificate の手順** を実行します。

以下の **certutil** オプションおよび **PKICertImport** オプションの詳細は、「**certutil** および **PKICertImport**」 を参照してください。

NSS データベースへのクライアント証明書のインポート

NSS データベースへのクライアント証明書のインポートするには、以下を実行します。

1. NSS データベースディレクトリーに移動します。以下に例を示します。

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

2. ルート証明書をまだインポートおよび信頼していない場合は、インポートして信頼します。詳細は、「ルート証明書のインポート」 を参照してください。
3. 中間証明書をインポートおよび検証していない場合は、中間証明書をインポートおよび検証します。詳細は、「中間証明書チェーンのインポート」 を参照してください。

4. クライアント証明書を検証し、インポートします。

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

エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

オブジェクト署名証明書のインポート

オブジェクト署名証明書をインポートするには、以下を実行します。

1. NSS データベースディレクトリーに移動します。以下に例を示します。

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

2. ルート証明書をまだインポートおよび信頼していない場合は、インポートして信頼します。詳細は、「ルート証明書のインポート」 を参照してください。

3. 中間証明書をインポートおよび検証していない場合は、中間証明書をインポートおよび検証します。詳細は、「中間証明書チェーンのインポート」 を参照してください。

4. オブジェクト署名証明書を検証し、インポートします。

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

エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

OCSP レスポンダーのインポート

OCSP レスポンダーをインポートするには、以下を行います。

1. NSS データベースディレクトリーに移動します。以下に例を示します。

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

2. ルート証明書をまだインポートおよび信頼していない場合は、インポートして信頼します。詳細は、「ルート証明書のインポート」 を参照してください。

3. 中間証明書をインポートおよび検証していない場合は、中間証明書をインポートおよび検証します。詳細は、「中間証明書チェーンのインポート」 を参照してください。

4. OCSP レスポンダー証明書を検証およびインポートします。

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

エラーメッセージが出力されず、戻りコードが 0 の場合は、検証が成功します。戻りコードを確認するには、上記のコマンド実行後すぐに **echo $?** を実行します。ほとんどの場合は、視覚的なエラーメッセージが出力されます。検証に成功しなかった場合は、発行者に連絡し、すべての中間証明書およびルート証明書がシステムに存在することを確認します。

第16章 証明書プロファイルの設定

16.1. ファイルシステムで直接証明書プロファイルの作成および編集

CA のインストールプロセスの一部として、プロファイルの設定ファイルを変更することにより、証明書登録プロファイルをファイルシステム上で直接変更できます。インストール時のデフォルトプロファイルにはデフォルトファイルが存在します。新しいプロファイルが必要な場合は、新しいプロファイル設定ファイルを作成します。設定ファイルは CA プロファイルディレクトリーである **instance_directory/ca/profiles/ca/** に保存されます (例: **/var/lib/pki/pki-ca/ca/profiles/ca/**)。ファイル名は **profile_name.cfg** です。プロファイルルールのパラメーターはすべて、これらのプロファイル設定ファイルで設定または変更できます。プロファイルルールは、入力、出力、認証、承認、デフォルト、および制約を使用できます。

CA 証明書の登録プロファイルは、***.profile** という名前の **/var/lib/pki/instance_name/ca/conf** ディレクトリーにあります。

注記

監査上の理由から、この方法は、デプロイメント前の CA のインストール中にのみ使用してください。

プロファイル設定ファイルを編集して変更を有効にした後、サーバーを再起動します。

- 「プロファイル設定パラメーター」
- 「ファイルシステムで直接証明書拡張機能の変更」
- 「ファイルシステムへのプロファイル入力の直接追加」

16.1.1. CA 以外のシステム証明書プロファイルの設定

16.1.1.1. プロファイル設定パラメーター

プロファイルルールのすべてのパラメーター (デフォルト、入力、出力、および制約) は、単一のポリシーセット内で設定されます。プロファイルに設定されたポリシーには **policyset.policyName.policyNumber** という名前があります。以下に例を示します。

```
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
```

一般的なプロファイル設定パラメーターについては、表16.1「プロファイル設定ファイルのパラメーター」 で説明されています。

表16.1 プロファイル設定ファイルのパラメーター

パラメーター	説明

パラメーター	説明
desc	エンドエンティティーページに表示される証明書プロファイルのフリーテキストの説明を提供します。たとえば、**desc=This certificate profile is for enrolling server certificates with agent authentication.** です。
enable	プロファイルを有効にするかどうかを設定します。したがって、エンドエンティティーページからアクセスできます。たとえば、**enable=true** とします。
auth.instance_id	プロファイルを介して送信された証明書要求の認証に使用する認証マネージャープラグインを設定します。自動登録では、認証に成功すると CA は証明書をすぐに発行します。認証が失敗した場合、または認証プラグインが指定されていない場合、リクエストはキューに入れられ、エージェントによって手動で承認されます。たとえば、**auth.instance_id=CMCAuth** となります。認証方法は、**CS.cfg** から登録された認証インスタンスの 1 つである必要があります。
authz.acl	承認制約を指定します。最も一般的なものは、グループ評価 ACL を設定するのに使用されます。たとえば、この caCMCUserCert パラメーターでは、CMC 要求の署名者が Certificate Manager Agents グループに属している必要があります。 **authz.acl=group="Certificate Manager Agents"** ディレクトリーベースのユーザー証明書の更新では、このオプションを使用して、元の要求元と現在の認証ユーザーが同じであることを確認します。 エンティティーは、承認を評価する前に、認証 (バインド、または基本的にシステムへのログイン) を行う必要があります。認証メソッドは、**CS.cfg** から登録された認証インスタンスの 1 つである必要があります。
name	プロファイルの名前を指定します。たとえば、**name=Agent-Authenticated サーバー証明書の登録** などです。この名前は、エンドユーザーの登録または更新ページに表示されます。
input.list	名前でプロファイルに許可されている入力を一覧表示します。たとえば、**input.list=i1,i2** です。
input.*input_id*.class_id	入力 ID (**input.list** にリストされている入力の名前) によって入力の Java クラス名を指定します。たとえば、**input.i1.class_id=cmcCertReqInputImpl** です。

パラメーター	説明
output.list	プロファイルの可能な出力形式を名前でリストします。たとえば、**output.list=o1** です。
output.*output_id*.class_id	**output.list** で指定された出力形式の Java クラス名を指定します。たとえば、**output.o1.class_id=certOutputImpl** です。
policyset.list	設定したプロファイルルールを一覧表示します。二重証明書の場合は、1 セットのルールが署名キーに適用され、もう 1 セットが暗号化キーに適用されます。1 つの証明書で使用するプロファイルは、1 つのプロファイルルールセットだけです。たとえば、**policyset.list=serverCertSet** です。
policyset.*policyset_id*.list	プロファイル用に設定されたポリシーセット内のポリシーを、評価する必要のある順序でポリシー ID 番号別に一覧表示します。たとえば、**policyset.serverCertSet.list=1,2,3,4,5,6,7,8** のようになります。
policyset.*policyset_id.policy_number*.constraint.class_id	プロファイルルールで設定されたデフォルトに設定された制約プラグインの Java クラス名を指定します。たとえば、**policyset.serverCertSet.1.constraint.class_id=subjectNameConstraintImpl** です。
policyset.*policyset_id.policy_number*.constraint.name	制約のユーザー定義の名前を指定します。たとえば、**policyset.serverCertSet.1.constraint.name=Subject Name Constraint** などです。
policyset.*policyset_id.policy_number*.constraint.params.*attribute*	制約に許可される属性値を指定します。設定可能な属性は、制約の種類によって異なります。たとえば、**policyset.serverCertSet.1.constraint.params.pattern=CN=.*** です。
policyset.*policyset_id.policy_number*.default.class_id	プロファイルルールに、デフォルトセットの java クラス名を指定します。たとえば、**policyset.serverCertSet.1.default.class_id=userSubjectNameDefaultImpl** です。
policyset.*policyset_id.policy_number*.default.name	デフォルトのユーザー定義の名前を指定します。たとえば、**policyset.serverCertSet.1.default.name=Subject Name Default** です。

パラメーター	説明
policyset.*policyset_id.policy_number*.default.params.*attribute*	デフォルトに対して許可される属性の値を指定します。設定可能な属性は、デフォルトの種類によって異なります。たとえば、**policyset.serverCertSet.1.default.params.name=CN=(Name)$request.requestor_name$** です。

16.1.1.2. ファイルシステムで直接証明書拡張機能の変更

制約を変更すると、指定できる情報の種類に制限があります。デフォルトと制約を変更すると、証明書要求から受け入れられる、または必要とされる拡張子を追加、削除、または変更することもできます。

たとえば、デフォルトの caFullnagiosUserCert プロファイルは、リクエストの情報からキーの使用拡張を作成するように設定されています。

```
 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
```

デフォルトは、ユーザーが指定するキー拡張機能を許可するように更新されます。

```
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
```

これにより、証明書リクエストで、拡張 OID **2.5.29.15** を受け入れるようにサーバーが設定されます。

その他の制約やデフォルトも同じように変更できます。必要な制約が適切なデフォルト値に含まれていることを確認し、別の制約が必要な場合にデフォルトが変更され、許可される制約のみがデフォルトで使用されます。詳細は、『Red Hat Certificate System 管理ガイド』の『デフォルト参照』および『制約参照』を参照してください。

16.1.1.2.1. 主な使用方法および拡張キー使用の一貫性

Red Hat Certificate System は、環境の要件を満たすように、管理者がカスタム登録プロファイルを作成するための柔軟なインフラストラクチャーを提供します。ただし、プロファイルでは RFC 5280 で定義された要件に違反する証明書を発行できません。キー使用法 (KU) と拡張キー使用法 (EKU) の両方の拡張機能が存在する登録プロファイルを作成する場合は、セクション 4.2.1.12 に従って、2 つの拡張機能間の整合性が維持されていることを確認することが重要です。RFC 5280 の拡張鍵使用。

KU 拡張機能の詳細は、以下を参照してください。

次の表に、一貫したキー使用ビットを各目的の拡張キー使用拡張にマップするガイドラインを示します。

目的/拡張キー使用方法	主な使用方法
TLS サーバー認証コマンド **id-kp-serverAuth**	**digitalSignature**、**keyEncipherment**、または **KeyAgreement**
TLS クライアント (相互) 認証 **id-kp-clientAuth**	**digitalSignature**、**keyEncipherment**、および/または **KeyAgreement**
コード署名 **id-kp-codeSigning**	**digitalSignature**
メール保護 **id-kp-emailProtection**	**digitalSignature**、**nonRepudiation**および/または (**keyEncipherment** または**keyAgreement**)
OCSP レスポンスの署名 **id-kp-OCSPSigning**	**KeyAgreement** および/または **nonRepudiation**

以下は、一貫性のない EKU/KU の例を 2 つ示しています。

- OCSP 応答署名を目的とした登録プロファイルには、拡張キーの使用法 **id-kp-OCSPSigning** が含まれていますが、**keyEncipherment** キー使用法ビットとともに使用されます。

```
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
```

- TLS サーバー認証の目的を目的とする登録プロファイルは、拡張キー **usage id-kp-serverAuth** ですが、CRL 署名鍵の使用ビットを使用します。

```
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 管理ガイド』の『主な使用拡張機能の制約』セクション
- 『Red Hat Certificate System 管理ガイド』の『keyUsage』セクション

EKU 拡張機能の詳細は、以下を参照してください。

- 『Red Hat Certificate System 管理ガイド』の『主な使用拡張機能の制約』セクション
- 『Red Hat Certificate System 管理ガイド』の『extKeyUsage』セクション

16.1.1.2.2. クロスペアプロファイルの設定

クロスペア証明書は、信頼パートナー関係を確立する別の CA 署名証明書であり、これにより、これら 2 つの別個の PKI のエンティティーが相互に信頼します。両方のパートナー CA は、その他の CA 署名証明書をデータベースに保存するため、他の PKI 内で発行されたすべての証明書は信頼され、認識されます。

Certificate System がサポートする 2 つの拡張は、このような信頼パートナー関係 (相互認証) を確立するために使用できます。

- 証明書ポリシー拡張 (**CertificatePoliciesExtension**) は、証明書が該当する条件を指定します。これは、多くの場合、PKI ごとに一意です。
- ポリシーマッピング拡張機能 (**PolicyMappingExtension**) は、2 つの環境の証明書プロファイルをマッピングすることにより、2 つの PKI 間の信頼をシールします。

クロスペアの証明書を発行するには、証明書ポリシー拡張が必要です。これは、『Red Hat Certificate System 管理ガイド』の付録『certificatePoliciesExt』で説明されています。

発行された証明書に CertificatePoliciesExtension が含まれていることを確認するには、登録プロファイルに適切なポリシールールを含める必要があります。次に例を示します。

```
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.explicitText
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=
```

この例の登録プロファイルで発行された証明書には、次の情報が含まれます。

```
Identifier: Certificate Policies: - 2.5.29.32
Critical: no
Certificate Policies:
Policy Identifier: 1.1.1.1
```

クロスペア証明書の使用の詳細は、『Red Hat Certificate System 管理ガイド』の『クロスペアの証明書の使用』セクションを参照してください。

クロスペア証明書の公開の詳細は、『Red Hat Certificate System 管理ガイド』の『クロスペアの証明書の公開』セクションを参照してください。

16.1.1.3. ファイルシステムへのプロファイル入力の直接追加

CA の **profiles/ca** ディレクトリーにある証明書プロファイル設定ファイルには、その特定の証明書プロファイルフォームの入力情報が含まれます。入力は、エンドエンティティーページ登録フォームのフィールドです。そのプロファイルに含まれる入力を一覧表示するパラメーター **input.list** があります。その他のパラメーターは入力を定義します。これらは **input.ID** 形式で識別されます。たとえば、これにより一般的な入力がプロファイルに追加されます。

```
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
```

利用可能な入力やフォームフィールドの詳細は、『Red Hat Certificate System 管理ガイド』の『入力の参照』セクションを参照してください。

16.1.2. 証明書のデフォルト有効期間の変更

認証局 (CA) の各プロファイルで、プロファイルを使用して発行された証明書が有効である期間を設定できます。セキュリティー上の理由から、この値を変更できます。

たとえば、生成された認証局 (CA) 署名証明書の有効性を **825** 日 (約 27 ヶ月) に設定するには、**/var/lib/pki/*instance_name*/ca/profiles/ca/caCACert.cfg** ファイルをエディターで開いて、以下を設定します。

```
policyset.caCertSet.2.default.params.range=825
```

16.1.3. CA システム証明書プロファイルの設定

CA 以外のサブシステムとは異なり、CA 自体のシステム証明書の登録プロファイルは **/var/lib/pki/[instance name]/ca/conf** ファイルに保持されます。これらのプロファイルは以下のとおりです。

- **caAuditSigningCert.profile**
- **eccAdminCert.profile**
- **rsaAdminCert.profile**
- **caCert.profile**
- **eccServerCert.profile**
- **saServerCert.profile**
- **caOCSPCert.profile**
- **eccSubsystemCert.profile**
- **rsaSubsystemCert.profile**

上記のプロファイルのデフォルト値を変更する場合は、「設定手順の開始」 手順を実行する前にプロファイルを変更します。以下は、次のことを示す例です。

- 効力を CA 署名証明書に変更する方法。
- エクステンションの追加方法 (証明書ポリシー拡張機能など)。

1. **pkispawn** が使用する元の CA 証明書プロファイルをバックアップします。

```
# cp -p /usr/share/pki/ca/conf/caCert.profile /usr/share/pki/ca/conf/caCert.profile.orig
```

2. 設定ウィザードが使用する CA 証明書プロファイルを開きます。

```
# vim /usr/share/pki/ca/conf/caCert.profile
```

3. Validity Default の有効期限を任意の値にリセットします。たとえば、期間を 2 年に変更するには、次のコマンドを実行します。

```
2.default.class=com.netscape.cms.profile.def.ValidityDefault
2.default.name=Validity Default
2.default.params.range=7200
```

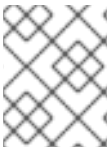
4. プロファイルに新しいデフォルトエントリーを作成し、これをリストに追加して、エクステンションを追加します。たとえば、証明書ポリシー拡張機能を追加するには、デフォルト (この例ではデフォルト #9) を追加します。

```
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=
```

次に、新しいデフォルトを使用するデフォルトの一覧にデフォルトの番号を追加します。

```
list=2,4,5,6,7,8,9
```

16.1.4. スマートカード CA プロファイルの管理



注記

TMS 上の本セクションにある機能は、評価でテストされていません。本セクションは参照用途としてのみ提供されています。

TPS は、証明書要求を生成または承認しません。Enterprise Security Client を介して承認された要求を、設定済みの CA に送信して証明書を発行します。これは、CA にトークンとスマートカードに使用するプロファイルが実際に含まれることを意味します。使用するプロファイルは、カードの種類に基づいて自動的に割り当てられます。

プロファイルの設定ファイルは、他の CA プロファイルが含まれる **/var/lib/pki/*instance_name*/profiles/ca/** ディレクトリーにあります。デフォルトのプロファイルは、表 16.2「デフォルトのトークン証明書プロファイル」 に一覧表示されます。

表16.2 デフォルトのトークン証明書プロファイル

プロファイル名	設定ファイル	説明
通常の登録プロファイル		
トークンデバイスのキー登録	caTokenDeviceKeyEnrollment.cfg	デバイスまたはサーバーに使用するトークンの登録
トークンユーザー暗号化証明書の登録	caTokenUserEncryptionKeyEnrollment.cfg	ユーザーのトークンで暗号化証明書を登録する場合。
トークンユーザーの署名証明書登録	caTokenUserSigningKeyEnrollment.cfg	ユーザーのトークンに署名証明書を登録する場合。
トークンユーザー MS ログイン証明書の登録	caTokenMSLoginEnrollment.cfg	Windows ドメインまたは PC へのシングルサインオンに使用するユーザー証明書を登録する場合。
一時トークンプロファイル		
一時的なデバイス証明書の登録	caTempTokenDeviceKeyEnrollment.cfg	一時的なトークンでデバイスの証明書を登録する場合
一時的なトークン暗号化証明書の登録	caTempTokenUserEncryptionKeyEnrollment.cfg	ユーザーの一時的なトークンに暗号化証明書を登録する場合。
一時的なトークンユーザー署名証明書	caTempTokenUserSigningKeyEnrollment.cfg	ユーザーの一時的なトークンに署名証明書を登録する場合。
プロファイルの更新[a]		
トークンユーザー暗号化証明書の登録 (更新)	caTokenUserEncryptionKeyRenewal.cfg	更新が許可される場合に、ユーザーのトークンで暗号化証明書を更新する場合。
トークンユーザー署名証明書の登録 (更新)	caTokenUserSigningKeyRenewal.cfg	更新が許可される場合に、トークンの署名証明書を更新するため。

[a] 更新プロファイルは、元の証明書を発行したプロファイルと組み合わせてのみ使用できます。メリットには 2 つの設定があります。

- 元の登録プロファイル名は変更されません。
- Renew Grace Period Constraint は、元の登録プロファイルに設定する必要があります。これは、ユーザーが証明書の更新を許可される証明書の有効期限の前後の時間を定義します。デフォルトのプロファイルにはこれらの例がいくつかあり、ほとんどの場合、デフォルトでは有効になっていません。

16.1.4.1. TPS の登録プロファイルの編集

管理者は、TPS で使用されるデフォルトのスマートカード登録プロファイルをカスタマイズできます。たとえば、プロファイルを編集して、サブジェクト代替名拡張子にユーザーの電子メールアドレスを含

めることができます。ユーザーのメールアドレスは認証ディレクトリーから取得されます。LDAP アクセス用に CA を設定するには、プロファイルファイルの次のパラメーターを、適切なディレクトリー情報とともに変更します。

```
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
```

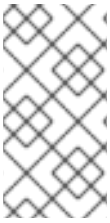
これらの CA プロファイルには、LDAP ルックアップがデフォルトで無効になっています。***ldapStringAttributes*** パラメーターは、会社ディレクトリーから取得する LDAP 属性を CA に指示します。たとえば、ディレクトリーに LDAP 属性名として ***uid*** が含まれており、証明書のサブジェクト名に使用される場合、***uid*** は ***ldapStringAttributes*** パラメーターにリストされ、**request.uid** は、***dnpattern*** のコンポーネントの一つとしてリストされなければなりません。

証明書プロファイルの編集は、『Red Hat Certificate System 管理ガイド』の『証明書プロファイルの設定』で説明されています。

dnpattern パラメーターのフォーマットは、パラメーターについては、『Red Hat Certificate System 管理ガイド』の『サブジェクト名の制約』セクションおよび『サブジェクト名のデフォルト』セクションで説明されています。

16.1.4.2. カスタム TPS プロファイルの作成

証明書プロファイルは通常どおり CA で作成されますが、トークンの登録で使用できるようにするには、TPS で設定する必要もあります。



ヒント

新しいプロファイルは、Red Hat Certificate System の新しいリリースで追加されます。インスタンスが CertificateSystem 10.0 に移行されると、新しいプロファイルは、カスタムプロファイルであるかのように、移行されたインスタンスに追加する必要があります。

1. 発行する CA 用の新しいトークンプロファイルを作成します。プロファイルの設定は、『Red Hat Certificate System 管理ガイド』の『証明書のプロファイルの設定』セクションで説明されています。

2. プロファイルを CA のプロファイルディレクトリー **/var/lib/*instance_name*/ca/profiles/ca/** にコピーします。

3. CA の **CS.cfg** ファイルを編集し、新規プロファイルの参照およびプロファイル名を CA のプロファイル一覧に追加します。以下に例を示します。

```
# 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. TPS **CS.cfg** ファイルを編集し、新しい CA 登録プロファイルを参照する行を追加します。以下に例を示します。

```
# vim /etc/pki/instance_name/tps/CS.cfg

   op.enroll.userKey.keyGen.signing.ca.profileId=tpsExampleEnrollProfile
```

5. スマートカードプロファイルを編集したら、インスタンスを再起動します。

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

CA と TPS が別のインスタンスにある場合は、両方のインスタンスを再起動します。

注記

外部登録の登録プロファイル (**externalReg**) 設定は、ユーザーの LDAP エントリーで設定されます。

16.1.4.3. Windows スマートカードのログオンプロファイルの使用

TPS はプロファイルを使用して、Windows ドメインまたは PC へのシングルサインオンに使用する証明書を生成します。これは、トークンユーザーの MS ログイン証明書の登録プロファイルです (**caTokenMSLoginEnrollment.cfg**)。

ただし、管理者が Windows スマートカードログインを設定するときに考慮する **必要** がある特別な考慮事項がいくつかあります。

- TLS に対して設定されていない場合は、ドメインコントローラーに証明書を発行します。
- グローバルポリシーではなく、ユーザーごとのスマートカードログインを設定して、ドメイン管理者のロックアウトを防止します。
- ドメインコントローラーはログインのたびに CRL をチェックするため、Active Directory サーバーへの CRL 公開を有効にします。

16.1.5. 証明書の登録プロファイルの無効化

本セクションでは、選択したプロファイルを無効にする手順を説明します。

証明書プロファイルを無効にするには、**/var/lib/pki/*instance_name*/ca/profiles/ca/** ディレクトリー内で対応する ***.cfg** ファイルを編集し、***visible*** と ***enable*** のパラメーターを **false** に設定します。

たとえば、CMC プロファイルをすべて無効にするには、次のコマンドを実行します。

1. CMC 以外のプロファイルの一覧を表示します。

```
# ls -l /var/lib/pki/instance_name/ca/profiles/ca/ | grep -v "CMC"
```

2. 表示される各ファイルで、以下のパラメーターを **false** に設定します。

```
visible=false
enable=false
```

さらに、すべての CMC プロファイルで **visible=false** を設定し、エンドエンティティーページに表示されないようにします。

1. CMC プロファイルの一覧を表示します。

```
# ls -l /var/lib/pki/instance_name/ca/profiles/ca/*CMC*
```

2. 表示される各ファイルで、以下を設定します。

```
visible=false
```

第17章 キーリカバリー認証局の設定

17.1. キーアーカイブの手動設定

重要

この手順は、CA および KRA が同じセキュリティードメインにある場合は不要です。この手順は、セキュリティードメインの **外部** にある CA にのみ必要です。

キーアーカイブを手動で設定すると、2 つの項目が可能になります。

- CA と KRA との間に信頼される関係があります。
- キーアーカイブ用に登録フォームが有効になっていると、鍵のアーカイブが設定され、KRA トランスポート証明書がフォームに保存されます。

同じセキュリティードメイン内で、KRA が設定されると、これらの設定手順の両方が **自動的** に実行されます。これは、KRA がセキュリティードメイン内の任意の CA と信頼関係を持つように設定されているためです。信頼関係とプロファイル登録フォームを手動で設定することにより、セキュリティードメイン外の Certificate Manager との信頼関係を作成することができます。

1. 必要に応じて、信頼できるマネージャーを作成して、Certificate Manager と KRA との間に関係を確立します。

   CA が KRA のキーアーカイブを要求できるようにするには、2 つのサブシステムが相互に認識、信頼、および通信するように設定されている必要があります。証明書マネージャーが、KRA の内部データベースで、適切な TLS クライアント認証証明書を使用して特権ユーザーとして設定されていることを確認します。デフォルトでは、Certificate Manager は、KRA への TLS クライアント認証にサブシステム証明書を使用します。

2. KRA の base-64 でエンコードされたトランスポート証明書をコピーします。

   トランスポート証明書は KRA の証明書データベースに保存され、**certutil** ユーティリティーを使用して取得できます。トランスポート証明書が Certificate Manager によって署名されている場合、証明書のコピーは、**Retrieval** タブの Certificate Manager エンドエンティティーページから入手できます。

   または、**pki** ユーティリティーを使用してトランスポート証明書をダウンロードします。

   ```
   # pki cert-find --name "KRA Transport certificate's subject common name"
   # pki cert-show serial_number --output transport.pem
   ```

3. CA の **CS.cfg** ファイルにトランスポート証明書を追加します。

   ```
   ca.connector.KRA.enable=true
   ca.connector.KRA.host=server.example.com
   ca.connector.KRA.local=false
   ca.connector.KRA.nickName=subsystemCert cert-pki-ca
   ca.connector.KRA.port=8443
   ca.connector.KRA.timeout=30
   ca.connector.KRA.transportCert=MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUFADA6
   MRgwFgYDVQQKEw9Eb21haW4gc28gbmFtZWQxHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1d
   Ghvcml0eTAeFw0wNjExMTQxODI2NDdaFw0wODEwMTQxNzQwNThaMD4xGDAWBgNVB
   AoTD0RvbWFpbiBzbyBuYW1lZDEiMCAGA1UEAxMZRFJNIFRyYW5zcG9ydCBDZXJ0aWZpY
   ```

```
2F0ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKnMGB3WkznueouwZjr
WLFZBLpKt6TimNKV9iz5s0zrGUlpdt81/BTsU5A2sRUwNfoZSMs/d5KLuXOHPyGtmC6yVvaY
719hr9EGYuv0Sw6jb3WnEKHpjbUO/vhFwTufJHWKXFN3V4pMbHTkqW/x5fu/3QyyUre/5IhG0
fcEmfvYxIyvZUJx+aQBW437ATD99Kuh+I+FuYdW+SqYHznHY8BqOdJwJ1JiJMNceXYAuAdk+
9t70RztfAhBmkK0OOP0vH5BZ7RCwE3Y/6ycUdSyPZGGc76a0HrKOz+lwVFulFStiuZlaG1pv0
NNivzcj0hEYq6AfJ3hgxcC1h87LmCxgRWUCAwEAAaN5MHcwHwYDVR0jBBgwFoAURShCYt
Sg+Oh4rrgmLFB/Fg7X3qcwRAYIKwYBBQUHAQEEODA2MDQGCCsGAQUFBzABhihodHR
wOi8vY2x5ZGUucmR1LnJlZGhhdC5jb206OTE4MC9jYS9vY3NwMA4GA1UdDwEB/wQEAwIE
8DANBgkqhkiG9w0BAQUFAAOCAQEAFYz5ibujdIXgnJCbHSPWdKG0T+FmR67YqiOtoNlGyI
gJ42fi5lsDPfCbIAe3YFqmF3wU472h8LDLGyBjy9RJxBj+aCizwHkuoH26KmPGntIayqWDH/UG
sIL0mvTSOeLqI3KM0IuH7bxGXjIION83xWbxumW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWy
YdfEOjpp/UQZOhOM1d8GFiw8N8ClWBGc3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwI
FhC82zqeQ1PbxQDL8QLjzca+IUzq6Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVlskXJw
5ZR0Y2zA==
ca.connector.KRA.uri=/kra/agent/kra/connector
```

4. 登録フォームを編集し、**keyTransportCert** メソッドでトランスポート証明書の値を追加または置き換えます。

```
vim /var/lib/pki/pki-tomcat/ca/webapps/ca/ee/ca/ProfileSelect.template

var keyTransportCert =
MIIDbDCCAlSgAwIBAgIBDDANBgkqhkiG9w0BAQUFADA6MRgwFgYDVQQKEw9Eb21haW4
gc28gbmFtZWQxHjAcBgNVBAMTFUNlcnRpZmljYXRlIEF1dGhvcml0eTAeFw0wNjExMTQxO
DI2NDdaFw0wODEwMTQxNzQwNThaMD4xGDAWBgNVBAoTD0RvbWFpbiBzbyBuYW1lZD
EiMCAGA1UEAxMZRFJNIFRyYW5zcG9ydCBDZXJ0aWZpY2F0ZTCCASIwDQYJKoZIhvcNA
QEBBQADggEPADCCAQoCggEBAKnMGB3WkznueouwZjrWLFZBLpKt6TimNKV9iz5s0zrGUl
pdt81/BTsU5A2sRUwNfoZSMs/d5KLuXOHPyGtmC6yVvaY719hr9EGYuv0Sw6jb3WnEKHpjb
UO/vhFwTufJHWKXFN3V4pMbHTkqW/x5fu/3QyyUre/5IhG0fcEmfvYxIyvZUJx+aQBW437ATD
99Kuh+I+FuYdW+SqYHznHY8BqOdJwJ1JiJMNceXYAuAdk+9t70RztfAhBmkK0OOP0vH5BZ7
RCwE3Y/6ycUdSyPZGGc76a0HrKOz+lwVFulFStiuZlaG1pv0NNivzcj0hEYq6AfJ3hgxcC1h87L
mCxgRWUCAwEAAaN5MHcwHwYDVR0jBBgwFoAURShCYtSg+Oh4rrgmLFB/Fg7X3qcwRA
YIKwYBBQUHAQEEODA2MDQGCCsGAQUFBzABhihodHRwOi8vY2x5ZGUucmR1LnJlZGh
hdC5jb206OTE4MC9jYS9vY3NwMA4GA1UdDwEB/wQEAwIE8DANBgkqhkiG9w0BAQUFAA
OCAQEAFYz5ibujdIXgnJCbHSPWdKG0T+FmR67YqiOtoNlGyIgJ42fi5lsDPfCbIAe3YFqmF3w
U472h8LDLGyBjy9RJxBj+aCizwHkuoH26KmPGntIayqWDH/UGsIL0mvTSOeLqI3KM0IuH7bx
GXjIION83xWbxumW/kVLbT9RCbL4216tqq5jsjfOHNNvUdFhWyYdfEOjpp/UQZOhOM1d8GFi
w8N8ClWBGc3mdlADQp6tviodXueluZ7UxJLNx3HXKFYLleewwIFhC82zqeQ1PbxQDL8QLjzca
+IUzq6Cd/t7OAgvv3YmpXgNR0/xoWQGdM1/YwHxtcAcVlskXJw5ZR0Y2zA==;
```

17.2. KRA 操作の暗号化

Certificate System は、キーリカバリー機関 (KRA) で以下の鍵操作を暗号化します。

- アーカイブ:
  - KRA に転送するために Certificate Request Message Format (CRMF) パッケージにアーカイブするキーの暗号化。
  - KRA LDAP データベースのストレージの鍵の暗号化。
- リカバリー:
  - キーに転送するためにユーザーが提供したセッションキーの暗号化。

- シークレットの復号と、ユーザー提供のセッションキーを使用した再暗号化、または PKCS #12 パッケージの作成。

- 生成:
  - ストレージの生成される鍵の暗号化。

17.2.1. クライアントによるキー操作暗号化の管理方法

Certificate System クライアントは、KRA の設定で設定された暗号化アルゴリズムを自動的に使用し、それ以上のアクションは必要ありません。

17.2.2. KRA での暗号化アルゴリズムの設定

注記

以下の設定では、AES CBC(**kra.allowEncDecrypt.archive=true** および **kra.allowEncDecrypt.recovery=true** の場合) および AES Key Wrap(**kra.allowEncDecrypt.archive=false** および **kra.allowEncDecrypt.recovery=false**の場合) だけを使用できます。いずれかのアルゴリズムをサポートする FIPS140-2 で検証された HSM はすべて、KRA が提供する主要なアーカイブおよびリカバリー機能に使用できます。

Certificate System は、**/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** ファイルで鍵操作暗号化に関連する設定パラメーターのグループを定義します。以下のパラメーターのセットを推奨します (他のオプションについては上記を参照)。

```
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
```

各グループ (**kra.storageUnit.wrapping.0.*** 対 **kra.storageUnit.wrapping.1.***) には個別の設定があり、番号はどの設定が同じ設定グループに属するかを定義します。現在の設定グループは、***/var/lib/pki/pki-instance_name/conf/kra/CS.cfg*** ファイル内の ***kra.storageUnit.wrapping.choice*** パラメーターで設定されます。

続行する前に、設定ファイルに **kra.storageUnit.wrapping.choice=1** が設定されていることを確認してください。

注記

Certificate System は、KRA データベースのレコードにデータを復号化するのに必要な情報を追加します。したがって、暗号化アルゴリズムを変更した後でも、Certificate System は、別の暗号化アルゴリズムを使用して、以前に KRA に保存されたデータを復号化できます。

17.2.2.1. パラメーターとその値の説明

各シークレット (payload) はセッションキーで暗号化されます。この暗号化を制御するパラメーターには、**payload** という接頭辞が付けられます。使用するパラメーターは、**kra.allowEncDecrypt.archive** および **kra.allowEncDecrypt.recovery** の値によって異なります。デフォルトでは、これらは両方とも false です。HSM におけるこの 2 つのパラメーターの効果については、「KRA で AES 暗号化を使用する場合の HSM の制約の解決」 を参照してください。

kra.allowEncDecrypt.archive と **kra.allowEncDecrypt.recovery** がいずれも false の場合:

- **payloadWrapAlgorithm** は、使用されるラッピングアルゴリズムを決定します。有効なオプションは **AES KeyWrap** のみです。
- **payloadWrapAlgorithm=AES/CBC/CBC/PKCS5Padding** の場合、**payloadWrapIVLength=16** を指定して IV を生成する必要がある PKI に指示する必要があります (CBC に 1 つ必要)。

kra.allowEncDecrypt.archive と **kra.allowEncDecrypt.recovery** がいずれも true の場合:

- **payloadEncryptionAlgorithm** は、使用される暗号化アルゴリズムを決定します。唯一の有効な選択肢は **AES** です。
- **payloadEncryptionMode** は、ブロックチェーンモードを決定します。唯一の有効な選択肢は **CBC** です。
- **payloadEncryptionPadding** により、パディングスキームが決まります。唯一の有効な選択肢は **PKCS5Padding** です。

次に、セッションキーは KRA Storage Certificate (RSA トークン) でラップされます。セッションキーおよびその暗号化を制御するパラメーターには、**sessionKey** という接頭辞が付けられます。

- **sessionKeyType** は、生成するキーのタイプです。唯一の有効な選択肢は **AES** です。
- **sessionKeyLength** は、生成されたセッションキーの長さです。有効な選択肢は **128** と **256** で、ペイロードをそれぞれ 128 ビット AES または 256 ビット AES で暗号化します。
- **sessionKeyWrapAlgorithm** は、KRA Storage 証明書が使用するキーのタイプです。本ガイドで唯一の有効な選択肢は **RSA** です。

17.2.2.2. KRA で AES 暗号化を使用する場合の HSM の制約の解決

KRA で AES を有効にして Certificate System を実行していても、ハードウェアセキュリティーモジュール (HSM) が AES キーラッピング機能をサポートしていない場合、キーのアーカイブは失敗します。この問題を解決するには、以下のソリューションがサポートされます。

- 「キーラッピングの差分アルゴリズムの選択」
- 「KRA の暗号化モードへの設定」

キーラッピングの差分アルゴリズムの選択

KRA は、デフォルトのキーラッピングアルゴリズムをサポートしていない場合がありますが、他のアルゴリズムはサポートしています。たとえば、**AES-128-CBC** をキーラッピングアルゴリズムとして使用するには、次のコマンドを実行します。

1. **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** ファイルで以下のパラメーターを設定します。

```
kra.storageUnit.wrapping.1.payloadWrapAlgorithm=AES KeyWrap
kra.storageUnit.wrapping.1.payloadWrapIVLen=16
kra.storageUnit.wrapping.1.sessionKeyLength=128
```

2. インスタンスを再起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

KRA が異なるインスタンスで実行されている場合、CA は両方のインスタンスを再起動する必要があります。

キーラッピングに別のアルゴリズムを選択すると、HSM が後で AES キーラッピングのサポートを追加した場合に、キーレコードに関連情報が設定されているため、設定を元に戻すことができるという利点があります。

この設定は、**kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen}** と **kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding}** のパラメーターを使用します。

KRA の暗号化モードへの設定

HSM が KeyWrap アルゴリズムをサポートしていない場合、場合によっては、KRA を暗号化モードにする必要があります。KRA を暗号化モードに設定すると、すべてのキーは、キーラッピングアルゴリズムではなく暗号化アルゴリズムを使用して保存されます。

KRA を暗号化モードに設定するには、以下を行います。

1. **/var/lib/pki/pki-*instance_name*/conf/kra/CS.cfg** ファイルの以下のパラメーターを **true** に設定します。

```
kra.allowEncDecrypt.archive=true
kra.allowEncDecrypt.recovery=true
```

2. サービスを再起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

KRA が CA 以外のインスタンスで実行している場合は、両方のインスタンスを再起動する必要があります。

この設定は、**kra.storageUnit.wrapping.1.payloadEncryption{Algorithm,Mode,Padding}** と **kra.storageUnit.wrapping.1.payloadWrap{Algorithm,IVLen}** パラメーターを使用します。

注記

後で、「キーラッピングの差分アルゴリズムの選択」 に従ってキーラッピングのために別のアルゴリズムに切り替える場合は、KRA を暗号化モードに設定する前に、作成されたレコードに適切なメタデータを手動で追加する必要があります。

17.3. エージェント承認キーリカバリースキームの設定

キーリカバリーエージェントは、PKCS #12 パッケージ内の秘密暗号化キーと関連する証明書をまとめて承認および取得します。キーリカバリーを承認するには、必要な数のリカバリーエージェントが KRA エージェントサービスページにアクセスし、**Authorize Recovery** 領域を使用して各承認を個別に入力します。

エージェントの 1 つが、キーリカバリープロセスを開始します。同期リカバリーの場合、各承認エージェントは、(最初のリクエストで返された) 参照番号を使用して要求をオープンにし、その後、個別に鍵のリカバリーを承認します。非同期リカバリーの場合、承認エージェントはすべてキーリカバリー要求を検索してから、キーリカバリーを承認します。すべての承認が指定されている場合に、KRA は情報を確認します。提示された情報が正しい場合は、要求されたキーを取得し、PKCS #12 パッケージの形式で、キーリカバリープロセスを開始したエージェントに、対応する証明書を返します。

キーリカバリーエージェントスキーム は、キーリカバリーエージェントが属するグループを認識するように KRA を設定し、アーカイブされた鍵を復元する前にキーリカバリー要求を承認するために必要なこれらのエージェントの数を指定します。

17.3.1. コマンドラインでのエージェント承認キーリカバリーの設定

エージェントが開始するキーリカバリーを設定するには、KRA 設定で 2 つのパラメーターを編集します。

- リカバリーの承認に必要なリカバリーマネージャーの数を設定します。
- これらのユーザーが属する必要のあるグループを設定します。

これらのパラメーターは、KRA の **CS.cfg** 設定ファイルで設定されます。

1. 設定ファイルを編集する前にサーバーを停止します。

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

または (**nuxwdog watchdog** を使用している場合)

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

2. KRA の **CS.cfg** ファイルを開きます。

```
# vim /var/lib/pki/pki-tomcat/kra/conf/CS.cfg
```

3. 2 つのリカバリースキームパラメーターを編集します。

```
kra.noOfRequiredRecoveryAgents=3
kra.recoveryAgentGroup=Key Recovery Authority Agents
```

4. サービスを再起動します。

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

または

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

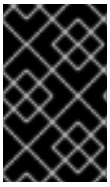
注記

コンソールでエージェント承認済みキーリカバリーを設定する方法は、『Red Hat Certificate System 管理ガイド』の『コンソールでのエージェント承認キーリカバリーの設定』を参照してください。

17.3.2. キーリカバリーフォームのカスタマイズ

デフォルトのキーエージェントスキームでは、キーリカバリー機関エージェントグループの単一のエージェントがキーリカバリーの承認を担当する必要があります。

キーリカバリー形式の外観をカスタマイズすることもできます。キーリカバリーエージェントには、キーリカバリープロセスを開始するための適切なページが必要です。デフォルトでは、KRA のエージェントサービスページには、キーリカバリーエージェントがキーリカバリーを開始し、キーリカバリー要求を承認し、暗号化キーを取得できるようにする適切な HTML フォームが含まれています。このフォームは、**confirmRecover.html** と呼ばれる **/var/lib/pki/pki-tomcat/kra/webapps/kra/agent/kra/** ディレクトリーにあります。



重要

キーリカバリーの確認フォームをカスタマイズするには、応答を生成するための情報を削除しないでください。これは、フォームの機能に不可欠です。コンテンツへの変更とフォームの表示を制限します。

17.3.3. 新しいプライベートストレージキーでのキーの再ラップ

一部の秘密鍵 (主に以前のデプロイメントで) は、KRA でアーカイブされた場合に SHA-1、1024 ビットのストレージキーでラップされました。プロセッサーの速度とアルゴリズムが破損しているため、このアルゴリズムはセキュリティーレベルが低くなります。セキュリティー対策として、新しい強力なストレージ鍵 (SHA-256、2048 ビット鍵) で秘密鍵を再ラップできます。

17.3.3.1. KRATool について

キーの再ラップと移動、およびキーの登録とリカバリー要求は、**KRATool** ユーティリティー (以前のバージョンの Red Hat Certificate System では **DRMTool**) を使用して行われます。

KRATool は、2 つの操作を実行します。新しい秘密鍵で鍵を再ラップでき、登録やリカバリー要求など、キーレコードの LDIF ファイルエントリーで属性を再配列できます。少なくとも 1 つの操作 (再ラップまたは再番号付け) を実行する必要があり、両方を 1 回の呼び出しで実行できます。

キーを再ラップする場合、ツールは元の KRA のエクスポートされた LDIF ファイルのキーエントリーにアクセスし、元の KRA ストレージキーを使用してキーをアンラップしてから、新しい KRA のより強力なストレージキーでキーを再ラップします。

例17.1 キーの再ラップ

```
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"
```

複数の KRA インスタンスが単一のインスタンスにマージされる場合は、キーまたは要求レコードに競合する CN、DN、シリアル番号、または要求 ID 番号がないことを確認することが重要です。これらの値を処理して、既存の値に新しい大きな数値を追加できます。

例17.2 キーの番号変更

```
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 オプションとその設定ファイルの詳細は、**KRATool(1)** の man ページで説明されています。

17.3.3.2.1つまたは複数の KRA から1つの KRA へのキーのラップとマージ

この手順では、1つ以上の Certificate System KRA (たとえば、*sourcekra.example.com*の**pki-tomcat**) に保存されているキーを再ラップし、それを別の Certificate System KRA (たとえば *targetkra.example.com* の **pki-tomcat-2**) に保存します。これが唯一のユースケースではありません。このツールは、ソースとターゲットの両方と同じインスタンスで実行して既存のキーを再ラップすることも、キーをまったく再ラップせずに複数の KRA インスタンスから単一のインスタンスにキーをコピーするために使用することもできます。

重要

pki-tomcat-2 KRA ストレージ鍵のサイズとタイプは、2048 ビットおよび RSA に設定する必要があります。

1. *targetkra.example.com* にログインします。
2. **pki-tomcat-2** KRA を停止します。

```
[root@targetkra ~]# systemctl stop pki-tomcatd@pki-tomcat-2.service
```

3. *sourcekra.example.com* にある **pki-tomcat** KRA インスタンスからインポートされるキーデータを保存するデータディレクトリーを作成します。

```
[root@targetkra ~]# mkdir -p /export/pki
```

4. **pki-tomcat-2** KRA のパブリックストレージ証明書を、新規データディレクトリーのフラットファイルにエクスポートします。

```
[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. 同じマシンにある場合は、**pki-tomcat-2** KRA の Directory Server インスタンスを停止します。

```
[root@newkra ~]# systemctl stop dirsrv.target
```

6. **pki-tomcat-2** KRA の設定情報をエクスポートします。

```
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

LDIF ファイルの最後に 1 行の空白行が含まれていることを確認してください。

7. *sourcekra.example.com* にログインします。
8. **pki-tomcat** KRA を停止します。

```
[root@sourcekra ~]# systemctl stop pki-tomcatd@pki-tomcat.service
```

9. *targetkra.example.com* にある **pki-tomcat-2** KRA インスタンスにエクスポートされるキーデータを保存するデータディレクトリーを作成します。

```
[root@sourcekra ~]# mkdir -p /export/pki
```

10. 同じマシンにある場合は、**pki-tomcat** KRA の Directory Server インスタンスを停止します。

```
[root@sourcekra ~]# systemctl stop dirsrv.target
```

11. **pki-tomcat** KRA の設定情報をエクスポートします。

```
[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

LDIF ファイルの最後に 1 行の空白行が含まれていることを確認してください。

12. このディレクトリーに **pki-tomcat** KRA NSS セキュリティーデータベースをコピーします。

```
[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. パブリックストレージキーを持つファイルを **pki-tomcat-2** KRA マシンからこのマシンにコピーします。以下に例を示します。

```
[root@sourcekra ~]# cd /export/pki

[root@sourcekra ~]# sftp root@targetkra.example.com
  sftp> cd /export/pki
  sftp> get targetKRA.cert
  sftp> quit
```

14. 必要に応じて、このツールで使用するデフォルトの **KRATool.cfg** ファイルを編集します。デフォルトのファイルは、変更せずに使用することもできます。

15. **KRATool** を実行します。これらのパラメーターはすべて 1 行にまとめる必要があります。

```
[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
```

注記

コマンドは、**pki-tomcat** KRA NSS セキュリティーデータベースに保存されているトークンのパスワードの入力を求める場合があります。

完了すると、このコマンドは、**-target_ldif_file** パラメーター **source2targetKRA.ldif** で指定されたファイルを作成します。

16. この LDIF ファイルを **pki-tomcat-2** KRA マシンにコピーします。以下に例を示します。

```
[root@sourcekra ~]# scp /export/pki/source2targetKRA.ldif
root@targetkra.example.com:/export/pki
```

重要

LDIF ファイルの最後に 1 行の空白行が含まれていることを確認してください。

17. 複数の KRA インスタンスをマージしている場合、それらのデータは単一のインポート操作にマージできます。マージされるすべての KRA で同じ手順を実施します。

重要

-target_ldif_file パラメーターが LDIF ファイルを作成するように一意の値を指定し、LDIF ファイルが連結したときに競合が発生しないように一意の **-append_id_offset** 値を指定します。

18. **pki-tomcat-2** KRA マシンに、**pki-tomcat-2** KRA 設定 LDIF ファイル、およびその他の KRA インスタンス用にエクスポートされたすべての LDIF ファイルを連結して、LDIF ファイルを他のキーデータとともにインポートします。以下に例を示します。

```
[root@targetkra ~]# cd /export/pki
[root@targetkra ~]# cat pki-tomcat-2.ldif source2targetKRA.ldif > combined.ldif
```

19. **pki-tomcat-2** KRA インスタンス用に、この組み合わせた LDIF ファイルを Directory Server データベースにインポートします。

```
[root@targetkra ~]# /usr/lib64/dirsrv/slapd-instanceName/ldif2db -n pki-tomcat-2-KRA -i
/export/pki/combined.ldif
```

20. **pki-tomcat-2** KRA の Directory Server インスタンスを起動します。

```
[root@targetkra ~]# systemctl start dirsrv.target
```

21. **pki-tomcat-2** KRA を起動します。

```
[root@targetkra ~]# systemctl start pki-tomcatd@pki-tomcat-2.service
```

17.3.4. クローン後の CA-KRA コネクター情報の更新

クローン後の CA-KRA 情報の更新に関する詳細は、「クローン後の CA-KRA コネクター情報の更新」を参照してください。

第18章 ログの設定

Certificate System サブシステムログファイルは、その特定のサブシステムインスタンス内の操作に関連するイベントを記録します。サブシステムごとに、インストール、アクセス、Web サーバーなどの問題について異なるログが保持されます。

すべてのサブシステムには同様のログ設定、オプション、および管理パスがあります。

インストール後のログ管理の詳細は、『Red Hat Certificate System 管理ガイド』の『サブシステムログの設定』セクションを参照してください。

ログの概要は、「ログ」を参照してください。

18.1. CERTIFICATE SYSTEM ログの設定

ログの設定方法は、Certificate System のパフォーマンスに影響を及ぼす可能性があります。たとえば、ログファイルのローテーションにより、ログが大きくなりすぎてサブシステムのパフォーマンスが低下するのを防ぎます。このセクションでは、Certificate System サブシステムによって記録されるさまざまな種類のログについて説明し、ログファイルのローテーション、バッファーリングされたログ、使用可能なログレベルなどの重要な概念を説明します。

18.1.1. ログに記録されるサービス

Certificate System のすべての主要コンポーネントとプロトコルは、メッセージをログファイルに記録します。表18.1「ログに記録されるサービス」デフォルトでログに記録されるサービスを一覧表示します。特定のサービスがログに記録するメッセージを表示するには、適宜ログ設定をカスタマイズします。

表18.1 ログに記録されるサービス

サービス	説明
ACL	アクセス制御リストに関連するイベントをログに記録します。
管理	コンソールとインスタンス間の HTTPS 通信など、管理アクティビティーに関連するイベントをログに記録します。
すべて	すべてのサービスに関連するイベントをログに記録します。
認証	認証モジュールに関連するアクティビティーに関連するイベントをログに記録します。
認証局	Certificate Manager に関連するイベントをログに記録します。
データベース	内部データベース関連のアクティビティーに関連するイベントをログに記録します。
HTTP	サーバーの HTTP アクティビティーに関連するイベントをログに記録します。HTTP イベントは実際には、HTTP サービスを提供するために Certificate System に組み込まれる Apache サーバーに属するエラーログに記録されることに注意してください。

サービス	説明
キーリカバリー認証局	KRA に関連するイベントをログに記録します。
LDAP	証明書と CRL の公開に使用される LDAP ディレクトリーを使用してアクティビティーに関連するイベントをログに記録します。
OCSP	OCSP ステータスの GET 要求など、OCSP に関連するイベントをログに記録します。
その他	コマンドラインユーティリティーやその他のプロセスなどの他のアクティビティーに関連するイベントをログに記録します。
要求キュー	要求キューアクティビティーに関連するイベントをログに記録します。
ユーザーおよびグループ	インスタンスのユーザーおよびグループに関連するイベントをログに記録します。

18.1.2. ログレベル (メッセージカテゴリー)

Certificate System サービスによってログに記録されるさまざまなイベントは、ログレベルによって決定されるため、イベントの識別とフィルタリングが簡単になります。さまざまな Certificate System のログレベルが、表18.2「ログレベルと対応するログメッセージ」 に一覧表示されます。

ログレベルは数字 **0** から **10** で表されます。各番号は、サーバーによって実行されるログのレベルを示します。このレベルは、ロギングの詳細を設定します。

優先度が高いほど、優先度の高いイベントのみがログに記録されるため、詳細度が低くなります。

注記

デフォルトのログレベルは **1** です。この値は変更しないでください。デバッグロギングを有効にするには、「デバッグログの追加設定」を参照してください。

表18.2「ログレベルと対応するログメッセージ」 は、ログメッセージをよりよく理解するために参照するために提供されます。

表18.2 ログレベルと対応するログメッセージ

ログレベル	メッセージカテゴリー	説明
0	デバッグ	これらのメッセージにはデバッグ情報が含まれます。このレベルは、非常に多くの情報を生成するため、通常の使用には推奨されません。
1	情報提供 (監査ログのデフォルト選択)	これらのメッセージは、**証明書システムの初期化完了** や **成功した操作要求** などのステータスメッセージを含む、証明書システムの状態に関する一般的な情報を提供します。

ログレベル	メッセージカテゴリー	説明
2	警告	これらのメッセージは警告のみであり、サーバーの通常の操作に障害があることを示すものではありません。
3	失敗 - システムおよびエラーログのデフォルト選択	これらのメッセージは、証明書サービス操作の実行の失敗 (**User authentication failed** または **Certificate revoked**) や、取り消せないエラーを引き起こす可能性のある予期しない状況 (**リクエストがクライアントからされた同じチャネルでクライアントに対して処理された要求を返信できない**) など、サーバーが正常に動作することを妨げるエラーおよび障害を示します。
4	誤った設定	これらのメッセージは、サーバーの設定が間違っていることが原因でエラーが発生していることを示します。
5	壊滅的な失敗	これらのメッセージは、エラーにより、サービスの実行を継続できないことを示しています。
6	セキュリティー関連のイベント	これらのメッセージは、サーバーのセキュリティーに影響する発生内容を特定します。たとえば、**リストにない証明書や取り消された証明書を使用するユーザーが特権アクセスを試みる場合など**です。
7	PDU 関連イベント (デバッグ)	これらのメッセージには、PDU イベントのデバッグ情報が含まれます。このレベルは、通常より有用な情報を超える情報を生成するため、通常の使用には推奨していません。
8	PDU 関連イベント	これらのメッセージは、MAC トークンの作成など、PDU で処理されるトランザクションおよびルールに関連するものです。
9	PDU 関連イベント	このログレベルは、MAC トークンの作成など、PDU で処理されるイベントの詳細ログメッセージを提供します。
10	すべてのロギングレベル	このログレベルは、すべてのログレベルを有効にします。

ログレベルを使用すると、イベントの重大度に基づいてログエントリーをフィルターできます。デフォルトでは、すべてのサービスにログレベル 3 (失敗) が設定されます。

ログレベルは連続して行われます。3 の値を指定するとレベル 4、5、および 6 がログに記録されます。ログデータは、特に低い (より冗長な) ログレベルでは広範囲に及ぶ可能性があります。ホストマシンには、すべてのログファイルに十分なディスク領域があることを確認します。また、すべてのログファイルがバックアップされ、ホストシステムが過負荷にならないように、ログレベル、ログローテーション、およびサーバーバックアップポリシーを適切に定義することも重要です。そうしないと、情報が失われる可能性があります。

18.1.3. バッファー付きおよびバッファーなしのロギング

Java サブシステムはすべてのタイプのログに対するバッファーロギングをサポートします。サーバーは、バッファー付きまたはバッファーなしのロギング用に設定できます。

バッファーログが設定されていると、サーバーは対応するログのバッファーを作成し、メッセージを可能な限りバッファーに保持します。サーバーは以下の条件のいずれかが発生した場合に限りログファイルにメッセージをフラッシュします。

- バッファーが満杯になった場合。バッファーサイズが **bufferSize** 設定パラメーターで指定された値以上になると、バッファーが満杯になります。このパラメーターのデフォルト値は 512 KB です。
- バッファーのフラッシュ間隔に到達した場合。最後のバッファーフラッシュからの経過時間、または **flushInterval** 設定パラメーターで指定された値と同じか大きい場合は、フラッシュ間隔に到達します。このパラメーターのデフォルト値は 5 秒です。
- 現在のログがコンソールから読み取られる場合。サーバーは現在のログについてクエリーされる際に最新のログを取得します。

サーバーがバッファーなしロギング用に設定されている場合、サーバーはログファイルに生成されるときにメッセージをフラッシュします。サーバーはメッセージが生成されるたびに I/O 操作 (ログファイルへの書き込み) を実行するため、バッファーなしログ用にサーバーを設定するとパフォーマンスが低下します。

ログパラメーターの設定は、『Red Hat Certificate System 管理ガイド』の『コンソールでログの設定』セクションを参照してください。

18.1.4. ログファイルローテーション

サブシステムログにはオプションのログ設定があり、ログファイルを無期限に拡張する代わりに、ログをローテーションして新しいログファイルを開始できます。ログファイルは、以下のいずれかの場合にローテーションされます。

- 対応するファイルのサイズ制限に到達した場合。対応するログファイルのサイズは、**maxFileSize** 設定パラメーターで指定された値以下である必要があります。このパラメーターのデフォルト値は 100 KB です。
- 対応するファイルの経過時間制限に到達した場合。対応するログファイルは、**rolloverInterval** 設定パラメーターで指定された間隔以上です。このパラメーターのデフォルト値は 2592000 秒 (30 日ごと) です。

ログファイルがローテーションされると、追加したタイムスタンプを持つファイルの名前を使用して古いファイルの名前が指定されます。追加されたタイムスタンプは、対応するアクティブなログファイルがローテーションされた日時を示す整数です。日付と時刻の形式は、YYYYMMDD (年、月、日) および HHMMSS (時、分、秒) です。

ログファイル (特に監査ログファイル)v には重要な情報が含まれています。**log** ディレクトリー全体をアーカイブメディアにコピーして、これらのファイルを定期的に一部のバックアップメディアにアーカイブする必要があります。

注記

Certificate System は、ログファイルをアーカイブするためのツールやユーティリティーを提供していません。

Certificate System は、改ざん検出の手段としてログファイルをアーカイブする前にログファイルに署名するコマンドラインユーティリティー **signtool** を提供します。

ログファイルの署名は、署名された監査ログ機能の代わりに使用されます。署名付き監査ログは、サブシステム署名証明書で自動的に署名される監査ログを作成します。

ローテーションされたログファイルは削除されません。

18.2. オペレーティングシステム (RHCS の外部) のログ設定

18.2.1. OS レベルの監査ログの有効化

警告

以下のセクションではすべての操作は、**sudo** で root または特権ユーザーとして実行する必要があります。

auditd ロギングフレームワークは、多くの監査機能を追加で提供します。これらの OS レベル監査ログは、Certificate System が提供する機能を補完します。本セクションの手順を実行する前に、**audit** パッケージがインストールされていることを確認してください。

```
# sudo yum install audit
```

システムパッケージの更新 (**yum** および **rpm** を使用し、Certificate System を含む) の監査は自動的に実行され、追加の設定は必要ありません。

注記

各監査ルールを追加して **auditd** サービスを再起動したら、以下のコマンドを実行して新しいルールが追加されたことを確認します。

```
# auditctl -l
```

新しいルールの内容が出力に表示されるはずです。

作成された監査ログの表示方法は、『Red Hat Certificate System 管理ガイド』の『オペレーティングシステムレベルの監査ログの表示』セクションを参照してください。

18.2.1.1. 証明書システムの監査ログ削除の監査

監査ログが削除されたときの監査イベントを受信するには、ターゲットが Certificate System ログであるシステムコールを監査する必要があります。

以下の内容で **/etc/audit/rules.d/rhcs-audit-log-deletion.rules** ファイルを作成します。

```
-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
```

次に **auditd** を再起動します。

```
# service auditd restart
```

18.2.1.2. 秘密鍵の使用が承認されていない Certificate System の監査

Certificate System の秘密または秘密鍵へのすべてのアクセスに対する監査イベントを受け取るには、NSS DB へのファイルシステムアクセスを監査する必要があります。

以下の内容で **/etc/audit/rules.d/rhcs-audit-nssdb-access.rules** ファイルを作成します。

```
-w /etc/pki/<instance name>/alias -p warx -k rhcs_audit_nssdb
```

<instance name> は、現在のインスタンスの名前です。**/etc/pki/<instance name>/alias** の各ファイルの **/etc/audit/rules.d/rhcs-audit-nssdb-access.rules** に、以下の行を追加します。

```
-w /etc/pki/<instance name>/alias/<file> -p warx -k rhcs_audit_nssdb
```

たとえば、インスタンス名が **pki-ca121318ec** で、**cert9.db**、**key4.db**、**NHSM-CONN-XCcert9.db**、**NHSM-CONN-XCkey4.db**、および **pkcs11.txt** がファイルである場合、設定ファイルには以下が含まれます。

```
-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
```

次に **auditd** を再起動します。

```
# service auditd restart
```

18.2.1.3. 時間変更イベントの監査

時間変更の監査イベントを受信するには、システム時間を変更する可能性のあるシステムコールアクセスを監査する必要があります。

以下の内容で **/etc/audit/rules.d/rhcs-audit-rhcs_audit_time_change.rules** ファイルを作成します。

```
-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
```

次に **auditd** を再起動します。

```
# service auditd restart
```

時間の設定方法は、『Red Hat Certificate System 管理ガイド』の『Red Hat Enterprise Linux 7 での日時の設定』を参照してください。

18.2.1.4. 証明書システム設定へのアクセスの監査

Certificate System インスタンス設定ファイルへのすべての変更に関する監査イベントを受け取るには、これらのファイルのファイルシステムアクセスを監査します。

以下の内容で **/etc/audit/rules.d/rhcs-audit-config-access.rules** ファイルを作成します。

```
-w /etc/pki/instance_name/server.xml -p wax -k rhcs_audit_config
```

また、**/etc/pki/*instance_name*/** ディレクトリーの各サブシステムに次の内容を追加します。

```
-w /etc/pki/instance_name/subsystem/CS.cfg -p wax -k rhcs_audit_config
```

例18.1 rhcs-audit-config-access.rules 設定ファイル

たとえば、インスタンス名が **pki-ca121318ec** で、CA のみがインストールされている場合に、**/etc/audit/rules.d/rhcs-audit-config-access.rules** ファイルには以下が含まれます。

```
-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
```

PKI NSS データベースへのアクセスは **rhcs_audit_nssdb** の下にすでに監査されていることに注意してください。

18.3. CS.CFG ファイルでのログの設定

インストールの設定中に、インスタンスの **CS.cfg** を直接編集して、ロギングを設定することができます。

1. サブシステムインスタンスを停止します。

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

2. **/var/lib/pki/*instance_name*/*subsystem_type*/conf** ディレクトリーの **CS.cfg** ファイルを開きます。

3. 新しいログを作成します。

4. ログインスタンスを設定するには、そのログに関連付けられたパラメーターを変更します。これらのパラメーターは **log.instance** で始まります。

   表18.3 ログエントリーパラメーター

パラメーター	説明

パラメーター	説明
type	ログファイルのタイプ。**システム** はエラーおよびシステムログを作成します。**トランザクション** は監査ログを記録します。
enable	ログがアクティブかどうかを設定します。有効にするログのみがイベントを記録します。
level	テキストフィールドにログレベルを設定します。このレベルは、フィールドに手動で入力する必要があります。選択メニューはありません。ログレベル設定は、「ログレベル (メッセージカテゴリー)」に記載されている数値です。
fileName	ログファイルへのファイル名を含む完全パス。サブシステムユーザーには、ファイルへの読み書きパーミッションがなければなりません。
bufferSize	ログのキロバイトサイズ (KB) のバッファーサイズ。バッファーがこのサイズに達すると、バッファーの内容はフラッシュされ、ログファイルにコピーされます。デフォルトのサイズは 512 KB です。バッファーロギングの詳細は、「バッファー付きおよびバッファーなしのロギング」 を参照してください。
flushInterval	バッファーの内容がフラッシュされてログファイルに追加されるまでの時間 (秒単位)。デフォルトの間隔は 5 秒です。
maxFileSize	ローテーションされる前に可能なログファイルのサイズをキロバイト (KB) 単位で設定できます。このサイズに達すると、ファイルはローテーションファイルにコピーされ、ログファイルが新たに開始されます。ログファイルのローテーションに関する詳細は、「ログファイルローテーション」を参照してください。デフォルトのサイズは 2000 KB です。
rolloverInterval	サーバーがアクティブなログファイルをローテーションする頻度。利用可能な選択肢は hourly、daily、weekly、monthly、および yearly です。デフォルトの選択は monthly です。詳細は、「ログファイルローテーション」 を参照してください。
logSigning[a]	署名付きロギングを有効にします。このパラメーターが有効な場合は、**signedAuditCertNickname** パラメーターの値を指定します。このオプションは、監査人だけがログを表示できることを意味します。値は **true** または **false** です。
signedAuditCertNickname[a]	監査ログの署名に使用される証明書のニックネーム。この証明書の秘密鍵は、ログに署名するためにサブシステムからアクセスできる必要があります。
events[a]	監査ログにログを記録するイベントを指定します。ログイベントは、空白のないコンマで区切ります。

[a] これは監査ログのみになります。

5. ファイルを保存します。
6. サブシステムインスタンスを開始します。

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

または (**nuxwdog watchdog** を使用している場合)

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

18.3.1. 署名監査ログの有効化および設定

18.3.1.1. 署名監査ログの有効化

デフォルトでは、監査ロギングはインストール時に有効になります。ただし、インストール後に、ログ署名を手動で有効にする必要があります。

現在の監査ロギング設定を表示するには、以下を実行します。

```
# 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
```

署名付き監査ログを有効にするには、以下を実行します。

1. **pki-server** ユーティリティーを使用して、**--logSigning** オプションを **true** に設定します。

```
# pki-server subsystem-audit-config-mod --logSigning True
  ...
  Log Signing: True
  ...
```

2. インスタンスを再起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

18.3.1.2. 監査イベントの設定

18.3.1.2.1. 監査イベントの有効化および無効化

監査イベントの有効化および無効化の詳細は、『Red Hat Certificate System 管理ガイド』の『コンソールでの署名監査ログの設定』セクションを参照してください。

さらに、監査イベントフィルターは、より詳細な選択に設定できます。「監査イベントのフィルタリング」を参照してください。

Certificate System で監査可能なイベントの完全リストは、『Red Hat Certificate System 監査ガイド』の『監査イベント』を参照してください。

18.3.1.2.2. 監査イベントのフィルタリング

Certificate System では、管理者はフィルターを設定して、イベント属性に基づいて監査ファイルに記録される監査イベントを設定できます。

フィルターの形式は LDAP フィルターと同じです。ただし、Certificate System は、以下のフィルターのみをサポートします。

表18.4 サポート対象の Audit イベントフィルター

タイプ	形式	例
Presence	(*attribute*=*)	(ReqID=*)
Equality	(*attribute=value*)	(Outcome=Failure)
Substring	(*attribute=initial*any*...*any*final*)	(SubjectID=*admin*)
AND 演算	(&(*filter_1*)(*filter_2*)...(*filter_n*))	(&(SubjectID=admin)(Outcome=Failure))
OR 演算	(|(*filter_1*)(*filter_2*)...(*filter_n*))	(|(SubjectID=admin)(Outcome=Failure))
NOT 演算	(!(*filter*))	(!(SubjectID=admin))

LDAP フィルターの詳細は、『Red Hat Directory Server Administration Guide』の『Using Compound Search Filters』セクションを参照してください。

> **例18.2 監査イベントのフィルタリング**
>
> プロファイル証明書要求と処理された証明書の現在の設定を表示するには、以下のコマンドを実行します。
>
> ```
> $ 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
> ```

InfoName フィールドが **rejectReason** または **cancelReason** に設定されている処理済み証明書要求のイベントと、プロファイル証明書要求およびイベントの失敗したイベントのみを記録するには、以下を実行します。

1. 以下のコマンドを実行します。

```
$ 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. Certificate System を再起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

18.3.2. セルフテストの設定

セルフテスト機能と個々のセルフテストは、**CS.cfg** ファイルで登録され、設定されます。セルフテストが有効になっている場合、そのセルフテストはオンデマンドまたは起動時に対して一覧表示され、空かまたは **critical** として設定されます。

重要なセルフテストには、セルフテストの名前の後にコロンと単語 **critical** があります。それ以外の場合は、何もありません。オンデマンドセルフテスト中に重要なセルフテストが失敗すると、サーバーはシャットダウンします。起動中に重要なセルフテストが失敗すると、サーバーは起動しません。

インスタンスのインストール時に、実装されたセルフテストが自動的に登録され、設定されます。登録および設定されるセルフテストは、サブシステムタイプに関連するものです。

自己テストの重大度は、**CS.cfg** ファイルのそれぞれの設定を変更することで変更されます。

18.3.2.1. 起動時のデフォルトのセルフテスト

以下の自己テストは、起動時にデフォルトで有効になります。

CA サブシステムでは、起動時に以下のセルフテストがデフォルトで有効になっています。

- **CAPresence**: CA サブシステムが存在することを確認するために使用されます。
- **CAValidity**: CA サブシステムが現在有効であり、期限切れでないことを確認するために使用されます。これには、CA 証明書の有効期限を確認する必要があります。
- **SystemCertsVerification**: システム証明書が現在有効であり、有効期限が切れていないか、または取り消されていないことを確認するために使用されます。CA サブシステムの場合は、各証明書の有効性テストのみが実行され、OCSP 要求が発生する可能性のある証明書検証テストは

除外されます。この動作は、以下の設定パラメーターで上書きできます。

```
selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true
```

デフォルトでは、この設定パラメーターが存在しない場合は **false** と見なされます。

KRA サブシステムの場合は、次のセルフテストが有効になります。

- **KRAPresence**: KRA サブシステムが存在することを確認するために使用されます。
- **KRAValidity**: KRA サブシステムが現在有効であり、期限切れでないことを確認するために使用されます。これには、KRA 証明書の有効期限を確認する必要があります。
- **SystemCertsVerification**: システム証明書が現在有効であり、有効期限が切れていないか、または取り消されていないことを確認するために使用されます。

OCSP サブシステムでは、以下のセルフテストが有効になります。

- **OCSPPresence**: OCSP サブシステムの有無を確認するために使用されます。
- **OCSPValidity**: OCSP サブシステムが現在有効であり、期限切れでないことを確認するために使用されます。これには、OCSP 証明書の有効期限を確認する必要があります。
- **SystemCertsVerification**: システム証明書が現在有効であり、有効期限が切れていないか、または取り消されていないことを確認するために使用されます。OCSP サブシステムの場合は、各証明書の有効性テストのみが実行され、OCSP 要求が発生する可能性のある証明書検証テストは除外されます。この動作は、以下の設定パラメーターで上書きできます。

```
selftests.plugin.SystemCertsVerification.FullCAandOCSPVerify=true
```

デフォルトでは、この設定パラメーターが存在しない場合は **false** と見なされます。

TKS サブシステムの場合は、次のセルフテストが有効になります。

- **TKS knownSessionKey**: TKS サブシステムの既知のセッションキーの検証に使用されます。これにより、TKS は、TPS およびサポート対象のスマートカードの代わりに鍵の作成を適切に支援できます。
- **SystemCertsVerification**: システム証明書が現在有効であり、有効期限が切れていないか、または取り消されていないことを確認するために使用されます。

TPS サブシステムの場合は、次のセルフテストが有効になります。

- **TPSPresence**: TPS サブシステムが存在することを確認するために使用されます。
- **TPSValidity**: TPS サブシステムが現在有効であり、期限切れでないことを確認するために使用されます。これには、TPS 証明書の有効期限を確認する必要があります。
- **SystemCertsVerification**: システム証明書が現在有効であり、有効期限が切れていないか、または取り消されていないことを確認するために使用されます。

18.3.2.2. セルフテスト設定の変更

デフォルトでは、セルフテスト設定は準拠しています。ただし、一部の設定により、自己テストロギングの表示やパフォーマンスを向上させることができます。セルフテストの設定設定を変更するには、以下を実行します。

1. サブシステムインスタンスを停止します。
2. インスタンスの **conf/** ディレクトリーにある **CS.cfg** ファイルを開きます。
3. self-test ログの設定を編集するには、**selftests.container.logger** で始まるエントリーを編集します。指定のない限り、これらのパラメーターはコンプライアンスには影響しません。これには、以下のパラメーターが含まれます。
   - **bufferSize**: ログのバッファーサイズをキロバイト単位 (KB) で指定します。デフォルトのサイズは 512 KB です。バッファーがこのサイズに達すると、バッファーの内容はフラッシュされ、ログファイルにコピーされます。
   - **enable** - 有効にする場合は **true** を指定します。有効にするログのみがイベントを記録します。この値は、コンプライアンスのために有効にする **必要** があります。
   - **filename**: ファイル名を含む、メッセージを書き込むファイルの完全パスを指定します。サーバーに、ファイルへの読み取り/書き込み権限が必要です。デフォルトでは、self-test ログファイルは **/var/log/pki-*name*/logs/selftest.log** です。
   - **flushInterval**: バッファーをファイルにフラッシュする間隔を秒単位で指定します。デフォルトの間隔は 5 秒です。**flushInterval** は、バッファーの内容がフラッシュされログファイルに追加されるまでの時間です。
   - **level**: デフォルトの選択は 1 です。このログは、1 以外のレベルでは設定されません。
   - **maxFileSize**: エラーログのファイルサイズをキロバイト単位 (KB) で指定します。デフォルトのサイズは 100 KB です。**maxFileSize** は、ログファイルがローテーションされる前のサイズを決定します。このサイズに達すると、ファイルはローテーションファイルにコピーされ、新しいログファイルが起動します。
   - **rolloverInterval**: アクティブなエラーログファイルをサーバーがローテーションする頻度を指定します。選択肢は hourly、daily、weekly、monthly、および yearly です。デフォルトの選択は monthly です。
   - **type**: **transaction** に設定してください。これは変更しないでください。
4. セルフテストを実行する順序を編集するには、コンマとスペースで区切った次のパラメーターの値としてセルフテストのいずれかをリストすることにより、順序を指定します。

   セルフテストをクリティカルとしてマークするには、リスト内のセルフテストの名前にコロンとクリティカルという単語を追加します。

   セルフテストを無効にするには、**selftests.container.order.onDemand** または **selftests.container.order.startup** のパラメーターの値の設定を削除します。
5. ファイルを保存します。
6. サブシステムを起動します。

18.3.3. デバッグログの追加設定

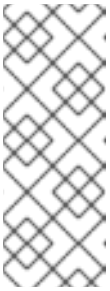
18.3.3.1. デバッグログの有効化および無効化

デフォルトでは、デバッグロギングは Certificate System で有効になっています。ただし、特定の状況では、管理者がこの機能を無効にまたは再有効化する必要があります。

1. **/var/lib/pki/*instance_name*/*subsystem_type*/conf/CS.cfg** ファイルを編集し、***debug.enabled*** のパラメーターを設定します。

   - デバッグのロギングを無効にするには、以下を設定します。

     ```
     debug.enabled=false
     ```

     注記

     デバッグログは監査ロギングの一部では **ありません**。デバッグログは、Certificate System で特定の障害をデバッグする場合や、インストールの失敗時に便利です。

     デフォルトで、デバッグログは有効です。望ましい場合には、管理者はデバッグロギングを安全に無効にして詳細度を下げることができます。

   - デバッグロギングを有効にするには、以下を実行します。

     ```
     debug.enabled=true
     ```

2. Certificate System インスタンスを再起動します。

   ```
   # systemctl restart pki-tomcatd@instance-name.service
   ```

   または (**nuxwdog watchdog** を使用している場合)

   ```
   # systemctl restart pki-tomcatd-nuxwdog@instance-name.service
   ```

18.3.3.2. デバッグログファイルのローテーション設定

Certificate System は、デバッグログをローテーションできません。デバッグロギングはデフォルトで有効になり、これらのログはファイルシステムが満杯になるまで増加します。**logrotate** などの外部ユーティリティーを使用してログをローテーションします。

例18.3 logrotate を使用したデバッグログのローテーション

以下の内容で **/etc/logrotate.d/*rhcs_debug*** などの設定ファイルを作成します。

```
/var/log/pki/instance_name/subsystem/debug {
    copytruncate
    weekly
    rotate 5
    notifempty
    missingok
}
```

1 つの設定ファイルで複数のサブシステムのデバッグログをローテーションするには、ログへのパスを空白で区切って、中括弧の前にリストします。以下に例を示します。

```
/var/log/pki/instance_name/ca/debug /var/log/pki/instance_name/kra/debug {
    ...
}
```

> **logrotate** および、この例で使用するパラメーターの詳細は、logrotate(8) の man ページを参照してください。

18.4. 監査の保持

監査データは、保持カテゴリーに応じた方法で保持する必要があります。

- 拡張監査保持: 証明書の存続期間 (発行から有効期限または失効日まで) の必要な保守のために保持される監査データ。Certificate System の場合は、以下の項目に表示されます。
  - 署名された監査ログ: Red Hat Certificate System 管理ガイドの付録 E. 監査イベントで定義されているすべてのイベント。
  - CA の内部 LDAP サーバーでは、CA が受け取った証明書要求レコードと、要求が承認されたときの証明書レコード。
- 通常の監査保持: 通常は、通常の操作をサポートするためにのみ保持される監査データ。これには、**拡張された監査保持** カテゴリーに記載されないすべてのイベントが含まれます。

注記

Certificate System は、監査データの修正や削除を行うインターフェイスを提供しません。

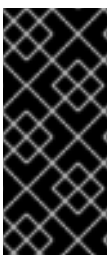
18.4.1. 監査データの場所

本セクションでは、Certificate System が監査データを保存する場所と、保持カテゴリーを決定するために重要なロールを果たす有効期限を見つける場所を説明します。

18.4.1.1. 監査ログの場所

証明書システムは、監査ログを **/var/log/pki-*name*/logs/signedAudit/** ディレクトリーに保存します。たとえば、CA の監査ログは **/var/lib/pki/*instance_name*/ca/logs/signedAudit/** ディレクトリーに保存されます。通常ユーザーは、このディレクトリー内のファイルにアクセスできません。 を参照してください。

拡張された監査保持期間を追跡する必要のある監査ログイベントの一覧は、『Red Hat Certificate System 管理ガイド』の『監査イベント』を参照してください。



重要

証明書要求または有効期限がまだ切れていない証明書については、Extended Audit Events 付録 に記載されているイベントを含む監査ログを削除しないでください。

これらの監査ログは、ディスクパーティションで使用可能なすべてのスペースまでのストレージスペースを消費する可能性があります。

18.4.1.2. 証明書要求および証明書レコードの場所

証明書署名要求 (CSR) が送信されると、CA は CSR を CA の内部ディレクトリーサーバーによって提供される要求リポジトリーに保存します。これらの要求が承認されると、各証明書が正常に発行され、同じ内部ディレクトリーサーバーによって証明書リポジトリーに LDAP レコードが作成されます。

CA の内部ディレクトリーサーバーは、**pkispawn** ユーティリティーを使用して CA が作成されると、以下のパラメーターに指定されました。

- ***pki_ds_hostname***
- ***pki_ds_ldap_port***
- ***pki_ds_database***
- ***pki_ds_base_dn***

証明書要求が正常に承認されると、証明書の有効性は、要求 ID またはシリアル番号のいずれかで CA EE ポータルにアクセスすることで確認できます。

証明書要求レコードの有効性を表示するには、次のコマンドを実行します。

1. **https://*host_name*:*port*/ca/ee/ca/** で CA EE ポータルにログインします。
2. **Check Request Status** をクリックします。
3. Request Identifier を入力します。
4. **Issued Certificate** をクリックします。
5. **Validity** を検索します。

証明書レコードからの有効性を表示するには、以下を実行します。

1. **https://*host_name*:*port*/ca/ee/ca/** で CA EE ポータルにログインします。
2. シリアル番号の範囲を入力します。1 つの特定のレコードを検索する場合は、レコードのシリアル番号を最小値と最大値の両方を、シリアル番号フィールドに入力します。
3. 検索結果をクリックします。
4. **Validity** を検索します。

重要

有効期限が切れていない証明書の証明書レコードの要求を削除しないでください。

第19章 ロールユーザーの作成

「デフォルトの管理ロール」 の説明にあるように、ブートストラップユーザーがインストール時に作成されていました。インストール後に、実際のユーザーを作成して、適切なシステム特権を割り当てます。各ユーザーは、1つのロール (グループ) のみのメンバーである必要があります。

本章では、以下を行う方法を説明します。

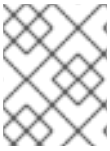
- オペレーティングシステム上での Certificate System の管理ユーザーの作成
- Certificate System での PKI ロールの作成

19.1. オペレーティングシステムでの PKI 管理ユーザーの作成

このセクションは、管理者ロールユーザーを対象にしています。agent および Auditor ロールユーザー。「証明書システムでの PKI ロールユーザーの作成」 を参照してください。

一般に、Certificate System の管理者、エージェント、および監査人は、コマンドラインユーティリティー、Java コンソール、ブラウザーなどのクライアントアプリケーションを使用して、Certificate System インスタンスをリモートで管理できます。大多数の CS 管理タスクでは、Certificate System ロールユーザーは、インスタンスを実行するホストマシンにログインする必要はありません。たとえば、監査人のユーザーは検証のために署名された監査ログをリモートで取得でき、エージェントロールのユーザーはエージェントインターフェイスを使用して証明書の発行を承認でき、管理者ロールのユーザーはコマンドラインユーティリティーを使用してプロファイルを設定できます。

ただし、場合によっては、Certificate System 管理者は、ホストシステムにログインして設定ファイルを直接変更するか、Certificate System インスタンスを開始または停止する必要があります。したがって、オペレーティングシステムでは、管理者ロールユーザーは、設定ファイルに変更を加えたり、Red Hat Certificate System に関連付けられたさまざまなログを読み取ったりできるユーザーである必要があります。



注記

Certificate System の管理者または監査人以外の人が監査ログファイルにアクセスすることを許可しないでください。

1. オペレーティングシステムに **pkiadmin** グループを作成します。

```
# groupadd -r pkiadmin
```

2. **pkiadmin** グループに **pkiuser** を追加します。

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

3. オペレーティングシステムでユーザーを作成します。たとえば、**jsmith** アカウントを作成するには、以下を実行します。

```
# useradd -g pkiadmin -d /home/jsmith -s /bin/bash -c "Red Hat Certificate System
Administrator John Smith" -m jsmith
```

詳細は、useradd(8) の man ページを参照してください。

4. ユーザー **jsmith** を **pkiadmin** グループに追加します。

```
# usermod -a -G pkiadmin jsmith
```

詳細は、usermod(8) の man ページを参照してください。

nCipher ハードウェアセキュリティーモジュール (HSM) を使用している場合は、HSM デバイスを管理するユーザーを **nfast** グループに追加します。

```
# usermod -a -G nfast pkiuser
# usermod -a -G nfast jsmith
```

5. 適切な **sudo** ルールを追加して、**pkiadmin** グループを Certificate System およびその他のシステムサービスに許可します。

   管理とセキュリティーの両方を簡素化するために、Certificate System と Directory Server のプロセスを設定して、(root だけでなく) PKI 管理者がサービスを開始および停止できるようにすることができます。

   サブシステムを設定する際に **pkiadmin** システムグループの使用が推奨されるオプションです。(詳細は 「Certificate System のオペレーティングシステムのユーザーおよびグループ」です)。Certificate System 管理者であるすべてのオペレーティングシステムユーザーがこのグループに追加されます。**pkiadmin** のシステムグループが存在する場合は、特定のタスクを実行するための sudo アクセスを付与することができます。

   1. **/etc/sudoers** ファイルを編集します。Red Hat Enterprise Linux では、 **visudo** コマンドを使用して実行できます。

      ```
      # visudo
      ```

   2. マシンにインストールされているものに応じて、Directory Server、Administration Server、PKI 管理ツール、および各 PKI サブシステムインスタンスの行を追加して、**pkiadmin** グループに **sudo** 権限を付与します。

      ```
      # 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

   マシン上のすべての Certificate System、Directory Server、および Administration Server に対して、またマシン上のそれらのインスタンスに対して**のみ**、sudo パーミッションを設定してください。マシンに同じサブシステムタイプのインスタンスが複数存在する場合と、サブシステムタイプのインスタンスが存在しない場合があります。これはデプロイメントによって異なります。

6. 以下のファイルのグループを **pkiadmin** に設定します。

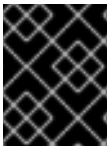
```
# chgrp pkiadmin /etc/pki/instance_name/server.xml
# chgrp -R pkiadmin /etc/pki/instance_name/alias
# chgrp pkiadmin /etc/pki/instance_name/subsystem/CS.cfg
# chgrp pkiadmin /var/log/pki/instance_name/subsystem/debug
```

オペレーティングシステムで管理ユーザーを作成したら、「証明書システムでの PKI ロールユーザーの作成」 の手順に従います。

19.2. 証明書システムでの PKI ロールユーザーの作成

PKI ロールユーザーを作成するには、『Red Hat Certificate System 管理ガイド』の『Certificate System ユーザーおよびグループの管理』セクションを参照してください。

第20章 BOOTSTRAP ユーザーの削除



重要

ブートストラップユーザーを削除する前に、19章 *ロールユーザーの作成* の説明に従って実際の PKI 管理ユーザーを作成します。

ブートストラップユーザーを削除するには、『Red Hat Certificate System 管理ガイド』の『Certificate System ユーザーの削除』セクションで説明されている手順に従います。

20.1. マルチロールサポートの無効化

デフォルトでは、ユーザーは一度に複数のサブシステムグループに属することができ、ユーザーは複数のロールとして機能できます。たとえば、John Smith はエージェントと管理者グループの両方に属する可能性があります。ただし、安全性の高い環境では、サブシステムのロールを制限して、ユーザーが 1 つのロールにしか所属できないようにする必要があります。そのためには、インスタンスの設定で ***multirole*** 属性を無効にします。

すべてのサブシステムの場合:

1. サーバーを停止します。

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

または (**nuxwdog watchdog** を使用している場合)

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

2. **CS.cfg** ファイルを開きます。

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

3. ***multiroles.enable*** パラメーターの値を **true** から **false** に変更します。

4. マルチロール設定に影響のある Certificate System のデフォルトロール一覧を追加または編集します。複数のロールが無効になっており、ユーザーが ***multiroles.false.groupEnforceList*** パラメーターににリストされているロールの **1つ** に属している場合、ユーザーはリスト内の他のロールのグループに追加することができません。

```
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. サービスを再起動します。

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

または (**nuxwdog watchdog** を使用している場合)

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

パート IV. CERTIFICATE SYSTEM 10.X へのアップグレード

第21章 CERTIFICATE SYSTEM 9 から CERTIFICATE SYSTEM 10 へのアップグレード

ユーザーは、Errata-Support チャネルを通じてセキュリティーとバグ修正を入手するために、Red Hat Certificate System の最新バージョンにアップグレードすることを推奨します。現在、最新バージョンは RHEL 7.9 の **RHCS 9.7** と RHEL 8.6 の **RHCS 10.4** です。

21.1. CA の移行

既存の証明書ファイルを使用した Certificate System の移行には、以下の手順が必要です。

1. 「以前のシステムからのデータのエクスポート」
2. 「PKCS12 ファイルの確認」
3. 「新規ホストでの CA の設定」
4. 「古いデータの新規 CA へのインポート」
5. 「デフォルトグループにユーザーの再割り当て」

21.1.1. 以前のシステムからのデータのエクスポート

新しい Certificate System インスタンスをセットアップする前に、以下の手順を使用して、現在の認証局 (CA) のデータをエクスポートします。この例では、CA のインスタンス名は *<pki-rootCA>* で、**/var/lib/pki/<instance_name>** にあります。

1. CA 署名証明書とキーをエクスポートします。

```
# 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. 証明書署名要求 (CSR) をエクスポートします。

```
# 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. 古い CA が (チェーン内に単一のルート CA を持つ) 中間 CA である場合は、NSS データベースからルート CA を抽出します。

```
# certutil -L -d /var/lib/pki/<instance_name>/alias/ -n root_CA_nickname -a > ca_rootca_signing.crt
```

4. 内部 LDAP データベースをバックアップします。
   - CA の **CS.cfg** で ***internaldb.database*** の値を確認して、CA データベースの名前を見つけます。

```
# grep internaldb.database /etc/pki/<instance_name>/ca/CS.cfg
internaldb.database=<CS_database_name>
```

- インスタンスが作成されたら、**setup-ds.pl** によって生成されたインスタンス固有のスクリプトを使用して、このデータベースを **.ldif** ファイルにエクスポートします。

```
# cd /usr/lib64/dirsrv/<instance_name>
# ./db2ldif -n "<CS_database_name>" -a /tmp/old_ca.ldif
```

db2ldif コマンドは DB ユーザーとして実行されるため、書き込みパーミッションを持つ宛先フォルダーが必要になります。

5. **old_ca.ldif**、**ca.p12**、**ca_signing.csr** ファイルを新しい CA マシンに転送します。移行する CA が中間 CA でもある場合は、**ca_rootca_signing.crt** も転送します。

21.1.2. PKCS12 ファイルの確認

新しい CA では、新しいデータベースインスタンスが標準ポート (389) で実行されます。**PKCS12** ファイルには、古いシステムのすべてのシステム証明書が含まれています。ファイルに CA 署名証明書およびキーが含まれていることを確認します。

注記

FIPS モードでは、NSS データベースを使用して **pki pkcs12** コマンドを実行する必要があります。詳細は、**pki-pkcs12** CLI を参照してください。

1. CA 署名証明書およびキーを見つけます。

```
$ 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. CA 署名証明書の信頼フラグが存在することを確認します。フラグが "CTu,Cu,Cu" ではない、またはそれらがない場合は追加します。

```
$ pki pkcs12-cert-mod "<caSigningCert cert-pki-rootCA>" \ --pkcs12-file ca.p12 --pkcs12-
password-file password.txt \ --trust-flags "CTu,Cu,Cu"
```

3. 他の証明書およびキーを削除します。

```
$ 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. 移行する CA も中間 CA の場合は、ルート CA 証明書を **PKCS#12** ファイルから削除します。

```
$ pki pkcs12-cert-del "<Top-level Root CA Signing Certificate>" --pkcs12-file ca.p12 --
pkcs12-password-file password.txt
```

21.1.3. 新規ホストでの CA の設定

既存のインスタンスからデータをエクスポートしたら、新しいホストに認証局 (CA) を作成してセットアップします。

1. 古い CA のパラメーターを使用して、**pkispawn** の設定ファイル (CA.cfg など) を作成します。このインスタンスは標準ポート (389) で実行されます。

```
[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
```

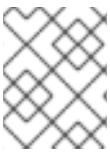
古い CA が中間 CA の場合は、以下の 2 行を設定ファイルに追加します。これらは、ルート CA 証明書へのパスと、NSS データベースに証明書を保存するときに使用するニックネームに対応します。

```
pki_cert_chain_path=<rootca_signing.crt>
pki_cert_chain_nickname=<caSigningCert cert-pki-ca>
```

詳細およびパラメーターの説明は、pkispawn(8) の man ページを参照してください。

2. 新規ホストで **pkispawn** を実行して、新規 CA インスタンスを作成します。

```
# pkispawn -s CA -f ca.cfg -v
```



注記

新規 CA 用に新しい DS インスタンスをセットアップする方法は、Red Hat Directory Server のインストール を参照してください。

21.1.4. 古いデータの新規 CA へのインポート

新しい CA インスタンスを作成した後、データを新しい CA データベースにインポートします。

1. CA サービスを停止します。

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

2. オプション: 新しいホストで CA データベースをバックアップします。

```
# dsctl -v idm-qe-01 db2bak
```

バックアップは、**/var/lib/dirsrv/*<instance_name>*/bak/*<host_name-time_stamp>*/** ディレクトリーに保存されます。

3. 新しい RHEL 8 内部データベースから、署名証明書の証明書エントリーを削除します。

```
# ldapdelete -x -w <password< -D 'cn=Directory Manager'
"cn=<serial_number<,ou=certificateRepository,ou=ca,o=pki-tomcat-CA"
```

エントリーが古いデータからインポートされ、そのエントリーの CRL 属性が正しいエントリーをポイントするようになりました。

4. データを新しいデータベースにインポートします。

```
# ldapmodify -x -W <password> -D 'cn=Directory Manager' -a -c -f <path/to/old_ca.ldif>
```

ldapmodify ユーティリティーは新しいエントリーのみを追加し、CA のインストール時に作成された既存のエントリーを更新しません。

5. オプション: **import.log** ファイルの出力を確認します。**ldap_add: Invalid syntax (21)** など、失敗したアクションを検索できます。

6. 古いセキュリティードメインのディレクトリーエントリーを削除します。

```
# 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. **/etc/pki/*<instance_name>*/ca/CS.cfg** ファイルの ***ca.crl.MasterCRL.enable*** パラメーターを有効にして、CA が証明書失効リスト (CRL) マスターとして機能するようにします。

```
ca.crl.MasterCRL.enable=true
```

8. CA サービスを起動します。

```
# systemctl start pki-tomcatd@<instance_name>
```

21.1.5. デフォルトグループにユーザーの再割り当て

ldapmodify または **pki** ユーティリティーを使用して、デフォルトのグループにメンバーを手動で追加します。

1. クライアントを設定します。

```
# 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. **ユーザー** アカウントを **Administrators** および **Security Domain Administrators** グループに追加します。

```
# pki -n "<PKI Administrator for example.com>" -c <password> \
user-membership-add <user_name> "Certificate Manager Agents"
```

RHCS 9 と RHCS 10 のデフォルト (ブートストラップ) 管理者ユーザーは、同じデフォルトのロール、つまり同じグループメンバーシップを持っています。ただし、デフォルト以外の管理者ユーザーがいる場合や、RHCS 9 で管理者ロールをカスタマイズしている場合は、これらの変更が適切に移行されない可能性があります。この場合は、RHCS 10 でロールを再設定します。

```
# 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"
```

証明書が正常に移行されたことを確認する場合は、**ca-cert-find** コマンドを実行して、新しいホストに存在することを確認します。

21.2. KRA の移行

単純な KRA 移行には、次の手順が必要です。

1. 「新しいホストでの KRA の設定」
2. 「以前のシステムからのコンテンツのエクスポート」
3. 「新規 KRA へのデータのインポート」
4. 「KRA エージェントページから移行したキーの存在の検証」

alpha.example.com にある KRA にはデータが含まれていますが、*omega.example.com* にある KRA はまだ存在しません。

Hostname	PKI KRA バージョン
alpha.example.com	PKI KRA 10.5 on RHCS 9.7
omega.example.com	PKI KRA 10.13 on RHCS 10.4

21.2.1. 新しいホストでの KRA の設定

omega.example.com で **root** ユーザーとして、以下のように設定します。

1. *omega.example.com* に新しい PKI 10.13 KRA をインストールして設定します。

2. KRA インスタンスを停止します。

```
# systemctl stop pki-tomcatd@<pki-kra>
```

3. 必要なファイルをすべてエクスポートするディレクトリーを作成します。

```
# mkdir -p /export/pki
```

4. KRA ストレージ証明書をファイルにエクスポートします。アルファ上の KRA から復号化された古いデータを再暗号化するには、後で KRA ストレージ証明書が必要になります。以下の例では、ファイルの名前は *omega.crt* です。

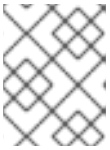
```
# cd /var/lib/pki/<pki-kra>/alias/
# pki-server cert-export kra_storage -i <pki-kra> --cert-file <omega.crt>
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 8 (0x8)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O = example.com Security Domain, OU = topology-02-CA, 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. **omega.crt** ファイルを **/export/pki** ディレクトリーに移動します。

```
# mv omega.crt /export/pki/
```

6. Directory Server を停止します。

```
# systemctl stop dirsrv@omega
```



注記

db2ldif がない場合は、389-ds-base-legacy-tools パッケージをインストールします。

7. KRA LDAP データベース設定を抽出します。古いデータを新しいデータに変換するには、新しい KRA LDAP データベース設定が必要になります。

```
# /usr/lib64/dirsrv/slapd-omega/db2ldif -n <pki-kra-KRA> -a /tmp/omega.ldif
```

8. **/tmp/omega.ldif** ファイルを **/export/pki** ディレクトリーに移動します。

```
# mv /tmp/omega.ldif /export/pki/
```

21.2.2. 以前のシステムからのコンテンツのエクスポート

alpha.example.com で **root** ユーザーとして以下を行います。

1. 共通のディレクトリーを作成します。

```
# mkdir -p /export/pki
```

2. Directory Server を停止します。以下の例では、サーバーの名前は *alpha* です。

```
# systemctl stop dirsrv@alpha
```

3. KRA LDAP データベースから LDIF を生成します。

```
# /usr/lib64/dirsrv/slapd-alpha/db2ldif -n <pki-kra-KRA> -a /tmp/alpha.ldif
```

4. **/tmp/alpha.ldif** ファイルを **/export/pki** ディレクトリーに移動します。

```
# mv /tmp/alpha.ldif /export/pki/
```

5. KRA NSS セキュリティーデータベースをデータ領域にコピーします。

```
# cp -p /var/lib/pki/<pki-kra>/alias/* /export/pki/
```

6. */export/pki* ディレクトリーに移動します。

```
# cd /export/pki
```

7. *omega.example.com* から、新しい KRA のストレージ証明書が含まれるフラットファイを取得します。

```
sftp root@omega.example.com
sftp> cd /export/pki
sftp> get omega.crt
sftp> quit
```

8. 内部パスワードを取得します。

```
# cat /var/lib/<instance_name>/conf/password.conf
```

9. *alpha.example.com* で **KRATool** を実行します。

```
# 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: xxxxxxxxxxxxxxxxxxxxxxxxxx.....
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"
```

注記

あるいは、証明書または証明書データベースによって自動的にアクセスされる **唯一の** パスワードを含む、不正アクセスから保護されたプレーンテキストファイルを作成することもできます。**-source_pki_security_database_pwdfile *<path_to_PKI_password_file>*** コマンドラインオプションを使用して、このファイルを **KRATool** に追加します。

10. **alpha2omega.ldif** ファイルを *omega.example.com* にコピーします。

```
sftp root@omega.example.com
sftp> cd /export/pki
sftp> put alpha2omega.ldif
sftp> quit
```

21.2.3. 新規 KRA へのデータのインポート

omega.example.com で **root** ユーザーとして、以下のように設定します。

1. **/export/pki** ディレクトリーに移動します。

```
# cd /export/pki
```

2. **ldif** ファイルを連結します。

```
# cat omega.ldif alpha2omega.ldif > omega_alpha.ldif
```

3. **omega_alpha.ldif** ファイルを、PKI KRA に関連付けられた LDAP データベースにインポートします。

```
# /usr/lib64/dirsrv/slapd-omega/ldif2db -n <pki-kra-KRA> -i /export/pki/omega_alpha.ldif
```

4. Directory Server を起動します。

```
# systemctl start dirsrv@omega
```

5. KRA インスタンスを起動します。

```
# systemctl start pki-tomcatd@<pki-kra>
```

21.2.4. KRA エージェントページから移行したキーの存在の検証

最後に、KRA エージェントページから移行した鍵のキーリカバリーを実行します。

```
[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. アップグレード後のテスト

アップグレード前にアーカイブされたユーザーキーのキーリカバリー操作を実行します。

手順21.1 キーリカバリーテスト

1. アップグレード前に作成された base64 ユーザー証明書を表示します。

```
# <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. この証明書を使用して、KRA エージェント UI を通じてキーリカバリー要求を生成します。

3. リカバリー要求を承認します。

```
# 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. KRA UI からキーをダウンロードします。

第22章 CERTIFICATE SYSTEM 10 の最新のマイナーバージョンへのアップグレード

Red Hat Certificate System 10.0 を最新のマイナーバージョン (10.1 など) にアップグレードするには、パッケージと設定ファイルを更新する必要があります。

1. サーバーのパッケージをすべてアップグレードします。

```
# yum update
```

更新時に、**/etc/pki/*<instance_name>*/*subsystem*/CS.cfg** や **/etc/pki/*<instance_name>*/server.xml** などの証明書システムの設定ファイルが、新規バージョンに自動的に変更されます。

Certificate System のアップグレード時に生成されるログファイルは以下の通りです。

- **/var/log/pki/pki-server-upgrade-*version*.log**
- **/var/log/pki/pki-upgrade-*version*.log**

ログファイル名のバージョン番号は、Certificate System のバージョンではなく、pki* パッケージのバージョンを表します。

注記

Directory Server が別のホストにインストールされている場合は、そのホストのパッケージも更新します。Directory Server の更新に関する詳細は、以下を参照してください。

- 『Red Hat Directory Server インストールガイド』の関連する章
- Directory Server の注目すべき変更点、バグ修正、既知の問題点については Red Hat Directory Server リリースノート

2. Certificate System インスタンスを再起動します。

```
# systemctl restart pki-tomcatd@<instance_name>.service
```

注記

9.7 などの下位メジャーバージョンから Certificate System 10 に移行するには、21 章*Certificate System 9 から Certificate System 10 へのアップグレード*を参照してください。

パート V. 証明書システムサブシステムのアンインストール

個々のサブシステムを削除するか、サブシステム全体に関連するすべてのパッケージをアンインストールできます。サブシステムは個別にインストールおよびアンインストールされます。たとえば、インストールおよび設定された CA サブシステムを残したまま、KRA サブシステムをアンインストールすることができます。また、単一の CA サブシステムを削除して、他の CA サブシステムをマシン上で残すこともできます。

第23章 サブシステムの削除

サブシステムを削除するには、サブシステムのタイプと、サブシステムが実行されているサーバーの名前を指定する必要があります。このコマンドは、サブシステムに関連付けられているすべてのファイルを削除します (サブシステムパッケージは削除しません)。

```
pkidestroy -s subsystem_type -i instance_name
```

-s オプションは、削除するサブシステムを指定します (CA、KRA、OCSP、TKS、または TPS など)。**-i** オプションは、**pki-tomcat** などのインスタンス名を指定します。

例23.1 CA サブシステムの削除

```
$ 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 ユーティリティーは、サブシステムと、証明書データベース、証明書、キー、関連ユーザーなどの関連ファイルを削除します。サブシステムパッケージをアンインストールしません。サブシステムがサーバーインスタンスの最後のサブシステムである場合は、サーバーインスタンスも削除されます。

第24章 CERTIFICATE SYSTEM のサブシステムパッケージの削除

多くのサブシステム関連のパッケージと依存関係が Red Hat Certificate System とともにインストールされます。一覧は 「Certificate System パッケージ」 に記載されています。サブシステムを削除すると、その特定のサブシステムに関連するファイルおよびディレクトリーのみが削除されます。このインスタンスが使用する、実際にインストールされているパッケージは削除されません。Red Hat Certificate System またはそのサブシステムの 1 つを完全にアンインストールするには、**yum** のようなパッケージ管理ツールを使用して、各パッケージを個別に削除する必要があります。

個々の Certificate System サブシステムパッケージをアンインストールするには、次のコマンドを実行します。

1. 関連するサブシステムをすべて削除します。以下に例を示します。

```
pkidestroy -s CA -i pki-tomcat
```

2. uninstall ユーティリティーを実行します。以下に例を示します。

```
yum remove pki-subsystem_type
```

サブシステムタイプは **ca**、**kra**、**ocsp**、**tks**、または **tps** にすることができます。

3. その他のパッケージおよび依存関係を削除するには、**yum** を使用して、パッケージを具体的に削除します。インストールされているパッケージの完全なリストは、「Certificate System パッケージ」 を参照してください。

用語集

A

agent (エージェント)

Certificate System マネージャーで エージェントサービス (agent services) の管理を許可されたグループに属するユーザー。Certificate Manager エージェント、キーリカバリー認証局エージェント (Key Recovery Authority agent) も併せて参照してください。

APDU

アプリケーションプロトコルデータユニット。スマートカードとスマートカードリーダーとの間の通信に使用される通信ユニット (バイトに類似)。

アクセス制御 (access control)

特定ユーザーが実行できるものを制御するプロセス。たとえば、サーバーへのアクセス制御は通常、パスワードまたは証明書によって確立された ID と、そのエンティティーが実行できることに関するルールに基づいています。アクセス制御リスト (ACL) も併せて参照してください。

アクセス制御リスト (ACL)

サーバーが特定のリソースへのアクセス要求を受け取ったときに評価されるアクセスルールの階層を定義するアクセス制御エントリーのコレクション。アクセス制御手順 (access control instructions, ACI) を参照してください。

アクセス制御手順 (access control instructions, ACI)

アクセスを要求するサブジェクトを識別する方法、または特定のサブジェクトに対して許可または拒否される権限を指定するアクセスルール。アクセス制御リスト (ACL) を参照してください。

エージェントサービス (agent services)

1.エージェントに必要な特権が割り当てられた Certificate System サブシステムが提供する HTML ページを通じて、Certificate System agent (エージェント) が管理できるサービス。

2.このようなサービスを管理するための HTML ページ。

エージェント承認登録の設定 (agent-approved enrollment)

証明書の発行前に、エージェントによる要求を承認するのに必要な登録。

属性値アサーション (attribute value assertion, AVA)

attribute = value の形式のアサーションで、**attribute** は **o** (組織) または **uid** (ユーザー ID) などのタグ、**value** は Red Hat, Inc.やログイン名などの値です。AVA は、証明書の サブジェクト名 (subject name) と呼ばれる、証明書の賢明を識別する 識別名 (distinguished name, DN) を形成するために使用されます。

監査ログ (autdit log)

さまざまなシステムイベントを記録するログこのログは署名して、改ざんされなかった証明を提供でき、auditor ユーザーのみが読み取りできます。

監査者 (auditor)

署名付き監査ログを表示できる特権ユーザー。

管理者 (administrator)

1 つ以上の Certificate System マネージャーをインストールおよび設定し、それらの特権ユーザーまたはエージェントをセットアップする人。agent (エージェント) も併せて参照してください。

自動登録 (automated enrollment)

人の介入なしに、エンドエンティティー登録の自動認証を可能にする Certificate System サブシステムを設定する方法。この形式の認証では、認証モジュールの処理を正常に完了した証明書要求が、プロファイル処理と証明書の発行に対して自動的に承認されます。

認可 (authorization)

サーバーが制御するリソースにアクセスするパーミッション。承認は通常、リソースに関連付けられた ACL がサーバーによって評価された後に行われます。アクセス制御リスト (ACL) を参照してください。

認証 (authentication)

自信を持って識別すること。コンピューター化された送信の当事者が偽者ではないことを保証すること。認証には通常、パスワード、証明書、PIN、またはその他の情報を使用して、コンピューターネットワーク上で ID を検証することが含まれます。パスワードベースの認証 (password-based authentication)、証明書ベースの認証 (certificate-based authentication)、クライアント認証 (client authentication)、サーバー認証 (server authentication) も参照してください。

認証モジュール (authentication module)

ルールのセット (として実装されますJava™クラス) エンドエンティティー、エージェント、管理者、または Certificate System サブシステムと対話する必要があるその他のエンティティーを認証するため。通常のエンドユーザー登録の場合は、ユーザーが登録フォームで要求された情報を入力

した後、登録サーブレットはそのフォームに関連付けられた認証モジュールを使用して情報を検証し、ユーザーの ID を認証します。サーブレット (servlet) を参照してください。

高度暗号化標準 (Advanced Encryption Standard, AES)

Advanced Encryption Standard (AES) は、その前身の Data Encryption Standard (DES) と同様に、FIPS 承認の対称鍵暗号化標準です。AES は 2002 年に米国政府によって採用されました。AES-128、AES-192、および AES-256 の 3 つのブロック暗号を定義します。NIST (National Institute of Standards and Technology) は、米国で AES 標準を定義しています。FIPS PUB 197。詳細は、http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf を参照してください。

B

バインド DN (bind DN)

Red Hat Directory Server への認証にパスワードとともに使用される識別名 (DN) の形式のユーザー ID。

C

CA サーバーキー (CA server key)

CA サービスを提供するサーバーの SSL サーバーキー。

CA 署名鍵 (CA signing key)

CA 証明書の公開鍵に対応する秘密鍵。CA は、その署名キーを使用して証明書および CRL に署名します。

CA 証明書 (CA certificate)

認証局を識別する証明書。認証局 (certificate authority, CA)、下位 CA (subordinate CA)、ルート CA も参照してください。

CA 階層 (CA hierarchy)

ルート CA が下位 CA に証明書を発行する権限を委任する CA の階層。下位 CA は、発行ステータスを他の CA に委譲して階層を拡張することもできます。認証局 (certificate authority, CA)、下位 CA (subordinate CA)、ルート CA も参照してください。

Certificate Manager

認証局として機能する独立した Certificate System サブシステム。Certificate Manager インスタンスは、証明書を発行、更新、および取り消します。証明書は、CRL とともに LDAP ディレクトリーに公開できます。エンドエンティティーからのリクエストを受け入れます。認証局 (certificate authority, CA) を参照してください。

Certificate Manager エージェント

Certificate Manager のエージェントサービスの管理が許可されているグループに所属するユーザーです。これらのサービスには、証明書要求にアクセスして変更 (承認および拒否) し、証明書を発行する機能が含まれています。

Certificate Request Message Format (CRMF)

X.509 証明書の管理に関連するメッセージに使用される形式。この形式は CMMF のサブセットです。証明書管理メッセージ形式 (Certificate Management Message Formats, CMMF) も併せて参照してください。詳細は、http://www.ietf.org/rfc/rfc2511.txt を参照してください。

Certificate System

Red Hat Certificate System、暗号化メッセージ構文 (Cryptographic Message Syntax, CS) を参照してください。

Certificate System コンソール

1 つの Certificate System インスタンスで開くことができるコンソール。Certificate System コンソールを使用すると、Certificate System 管理者は、対応する Certificate System インスタンスの設定設定を制御できます。

Certificate System サブシステム

5 つの Certificate System マネージャーの 1 つ: Certificate Manager、Online Certificate Status Manager、キーリカバリー認証局、Token Key Service、または Token Processing System。

CMC

暗号化メッセージ構文 (CMC) を介した証明書管理メッセージ を参照してください。

CMC 登録 (CMC Enrollment)

署名された登録または署名された失効要求のいずれかを、エージェントの署名証明書を使用して Certificate Manager に送信できるようにする機能。これらの要求は Certificate Manager によって自動的に処理されます。

CMMF

証明書管理メッセージ形式 (Certificate Management Message Formats, CMMF) を参照してください。

CRL

証明書失効リスト (certificate revocation list, CRL) を参照してください。

CRMF

Certificate Request Message Format (CRMF) を参照してください。

CSP

暗号化サービスプロバイダー (cryptographic service provider, CSP) を参照してください。

クライアント SSL 証明書 (client SSL certificate)

SSL プロトコルを使用してサーバーにクライアントを識別するために使用する証明書。セキュアソケットレイヤー (Secure Sockets Layer, SSL) を参照してください。

クライアント認証 (client authentication)

名前とパスワード、または証明書とデジタル署名されたデータなどを使用して、サーバーに対してクライアントを識別するプロセス。証明書ベースの認証 (certificate-based authentication)、パスワードベースの認証 (password-based authentication)、サーバー認証 (server authentication) を参照してください。

チェーン認証局 (chained CA)

リンクされた CA (linked CA) を参照してください。

ペア間の証明書 (cross-pair certificate)

ある CA から別の CA に発行され、信頼の輪を形成するために両方の CA によって保存される証明書。2 つの CA は相互に証明書を発行し、両方のクロスペア証明書を証明書ペアとして格納します。

信頼チェーン (chain of trust)

証明書チェーン (certificate chain) を参照してください。

暗号アルゴリズム (cryptographic algorithm)

暗号化 (encryption) や 復号化 (decryption) などの暗号化操作を実行するために使用される一連のルールまたは方向。

暗号モジュール (cryptographic module)

PKCS #11 モジュール を参照してください。

暗号化 (cipher)

暗号アルゴリズム (cryptographic algorithm) を参照してください。

暗号化サービスプロバイダー (cryptographic service provider, CSP)

PKCS #11 で定義されているような標準インターフェイスを使用してそのようなサービスを要求するソフトウェアに代わって、キー生成、キーストレージ、暗号化などの暗号化サービスを実行する暗号化モジュール。

暗号化メッセージ構文 (CMC) を介した証明書管理メッセージ

Certificate Manager への証明書の要求を伝えるために使用するメッセージ形式。Internet Engineering Task Force (IETF) PKIX の作業グループから提案された標準。詳細は、https://tools.ietf.org/html/draft-ietf-pkix-cmc-02 を参照してください。

暗号化メッセージ構文 (Cryptographic Message Syntax, CS)

CMMF などの任意のメッセージにデジタル署名、ダイジェスト、認証、または暗号化するために使用される構文。

相互認証 (cross-certification)

異なる証明書階層またはチェーン内の 2 つの CA による証明書交換クロス認定は、両方の階層に対応できるように信頼チェーンを拡張します。認証局 (certificate authority, CA) も併せて参照してください。

証明書 (certificate)

X.509 標準に準拠してフォーマットされたデジタルデータで、個人、会社などのエンティティー名 (証明書の サブジェクト名 (subject name)) を指定し、証明書に含まれる 公開鍵 (public key) もそのエンティティーに属することを証明するもの。証明書は発行され、認証局 (certificate authority, CA) により署名されます。証明書の有効性は、公開鍵暗号 (public-key cryptography) の手法で CA の デジタル署名 (digital signature) を確認して検証できます。公開鍵インフラストラクチャー (public-key infrastructure, PKI) 内で信頼されるために、証明書は、PKI に登録されているその他のエンティティーによって信頼されている CA により発行および署名される必要があります。

証明書のフィンガープリント (certificate fingerprint)

証明書に関連付けられた 一方向ハッシュ (one-way hash)。番号は証明書自体の一部ではありませんが、証明書の内容にハッシュ関数を適用することによって生成されます。証明書の内容が 1 文字でも変更されると、同じ関数で異なる番号が生成されます。したがって、証明書のフィンガープリントを使用して、証明書が改ざんされていないことを確認できます。

証明書チェーン (certificate chain)

連続した認証局によって署名された証明書の階層セット。CA 証明書は 認証局 (certificate authority, CA) を識別し、その認証局が発行した証明書の署名に使用されます。CA 証明書は、親 CA の CA 証明書により署名され、ルート CA まで署名できます。Certificate System により、エンドエンティティーが証明書チェーンのすべての証明書を取得できるようになります。

証明書プロファイル (certificate profile)

特定のタイプの登録を定義する一連の設定設定。証明書プロファイルは、証明書プロファイルの認証方法とともに、特定のタイプの登録のポリシーを設定します。

証明書ベースの認証 (certificate-based authentication)

証明書および公開鍵暗号に基づく認証。パスワードベースの認証 (password-based authentication) も併せて参照してください。

証明書失効リスト (certificate revocation list, CRL)

X.509 標準で定義されているように、認証局 (certificate authority, CA) により生成され署名されたシリアル番号ごとに取り消された証明書のリスト。

証明書拡張 (certificate extensions)

X.509 v3 証明書には、証明書に任意の数の追加フィールドを追加できる拡張フィールドが含まれています。証明書拡張は、代替サブジェクト名や使用制限などの情報を証明書に追加する方法を提供します。PKIX ワーキンググループによって、いくつかの標準拡張機能が定義されています。

証明書管理メッセージ形式 (Certificate Management Message Formats, CMMF)

エンドエンティティーから Certificate Manager に証明書要求と失効要求を伝達し、エンドエンティティーにさまざまな情報を送信するために使用されるメッセージ形式。Internet Engineering Task Force (IETF) PKIX の作業グループから提案された標準。CMMF は、別の提案された標準 暗号化メッセージ構文 (CMC) を介した証明書管理メッセージ に含まれています。詳細は、https://tools.ietf.org/html/draft-ietf-pkix-cmmf-02 を参照してください。

認証局 (certificate authority, CA)

証明書が特定を意図している個人またはエンティティーの身元を検証した後、証明書 (certificate) を発行する信頼されたエンティティー。CA は証明書を更新し、取り消し、CRL を生成します。証明書の発行者フィールドで指定されたエンティティーは、常に CA です。認証局は、独立したサードパーティー、または Red Hat Certificate System などの証明書発行サーバーソフトウェアを使用する個人または組織にすることができます。

D

キーリカバリー認証局

エンドエンティティーの RSA 暗号化キーの長期アーカイブとリカバリーを管理する任意の独立した Certificate System サブシステム。Certificate Manager は、新しい証明書を発行する前に、キーリカバリー機関を使用してエンドエンティティーの暗号化キーをアーカイブするように設定できます。キーリカバリー機関は、エンドエンティティーが機密性の高い電子メールなど、組織がいつかリカバリーする必要のあるデータを暗号化している場合にのみ役立ちます。デュアルキーペアをサポートするエンドエンティティーでのみ使用できます。2 つの個別のキーペアで、1 つは暗号化用、もう 1 つはデジタル署名用です。

キーリカバリー認証局のストレージキー (Key Recovery Authority storage key)

キーリカバリー機関の秘密トランスポートキーで復号された後、エンドエンティティーの暗号化キーを暗号化するためにキーリカバリー機関によって使用される特別なキー。ストレージキーがキーリカバリー機関を離れることはありません。

キーリカバリー認証局のトランスポート証明書 (Key Recovery Authority transport certificate)

エンドエンティティーがキーリカバリー機関に転送するためにエンティティーの暗号化キーを暗号化するために使用する公開キーを認証します。キーリカバリー機関は、認証された公開鍵に対応する秘密鍵を使用して、ストレージ鍵で暗号化する前に、エンドエンティティーの鍵を復号します。

キーリカバリー認証局エージェント (Key Recovery Authority agent)

要求キューの管理や HTML ベースの管理ページを使用したリカバリー操作の許可など、キーリカバリー機関のエージェントサービスの管理を許可されたグループに属するユーザー。

キーリカバリー認証局リカバリーエージェント (Key Recovery Authority recovery agent)

キーリカバリー認証局 のストレージキーの一部を所有している **m of n** 人の 1 人。

デジタル ID (digital ID)

証明書 (certificate) を参照してください。

デジタル署名 (digital signature)

デジタル署名を作成するため、署名ソフトウェアは最初に、新しく発行された証明書など、署名するデータから 一方向ハッシュ (one-way hash) を作成します。次に、一方向ハッシュは署名者の秘密鍵で暗号化されます。作成されるデジタル署名は、署名されるデータごとに一意になります。1 つのコンマがメッセージに追加されていても、そのメッセージのデジタル署名が変更されます。署名者の公開鍵を使用したデジタル署名の復号に成功し、同じデータの別のハッシュと比較することで、改ざんの検出 (tamper detection) が可能になります。公開鍵を含む証明書の 証明書チェーン (certificate chain) の検証により、署名側の認証が提供されます。否認防止 (nonrepudiation)、暗号化 (encryption)も参照してください。

デュアルキーペア (dual key pair)

2 つの個別の証明書に対応する、2 つの公開鍵と秘密鍵のペア (合計 4 つの鍵)。一方のペアの秘密鍵は署名操作に使用され、もう一方のペアの公開鍵と秘密鍵は暗号化および復号操作に使用されます。各ペアは個別の 証明書 (certificate) に対応します。暗号鍵 (encryption key)、公開鍵暗号 (public-key cryptography)、署名鍵 (signing key) も参照してください。

デルタ CRL (delta CRL)

最後の完全な CRL が発行されてから取り消された証明書の一覧を含む CRL。

分散ポイント (distribution points)

証明書セットを定義するのに CRL に使用されます。各ディストリビューションポイントは、発行する証明書のセットにより定義されます。CRL は、特定のディストリビューションポイント用に作成できます。

復号化 (decryption)

暗号化されたデータのスクランブル解除。暗号化 (encryption) を参照してください。

識別名 (distinguished name, DN)

証明書の件名を特定する一連の AVA。属性値アサーション (attribute value assertion, AVA) を参照してください。

E

extensions フィールド

証明書拡張 (certificate extensions) を参照してください。

エンドエンティティー (end entity)

公開鍵インフラストラクチャー (public-key infrastructure, PKI)、人、ルーター、サーバー、またはその他のエンティティーで、証明書 (certificate) を使用してそれ自体を特定します。

エンロールメント (enrollment)

公開鍵インフラストラクチャー (public-key infrastructure, PKI) で使用する X.509 証明書を要求および受信するプロセス。**登録** としても知られています。

傍受 (eavesdropping)

情報が意図されていないエンティティーによってネットワークを介して送信された情報の不正な傍受。

暗号化 (encryption)

その意味を偽装する方法で情報をスクランブルします。復号化 (decryption) を参照してください。

暗号鍵 (encryption key)

暗号化のみに使用される秘密鍵。暗号化鍵とその同等の公開鍵、および 署名鍵 (signing key) とその同等の公開鍵は、デュアルキーペア (dual key pair) を設定します。

楕円曲線暗号 (Elliptic Curve Cryptography, ECC)

暗号鍵の基礎となる数学的な問題に対して、楕円曲線を用いて加算対数を作成する暗号アルゴリズム。ECC 暗号は、RSA 暗号よりも効率的に使用でき、本質的に複雑であるため、RSA 暗号よりも小さいビットで強力です。詳細は、https://tools.ietf.org/html/draft-ietf-tls-ecc-12 を参照してください。

F

Federal Bridge Certificate Authority (FBCA)

2 つの CA が相互にクロスペア証明書を発行し、2 つのクロスペア証明書を単一の証明書ペアとして格納することにより、信頼の輪を形成する設定。

FIPS PUBS 140

FIPS PUBS (Federal Information Standards Publications) 140 は、データの暗号化と復号、またはデジタル署名の作成や検証などの他の暗号化操作を実行する暗号化モジュール、ハードウェア、またはソフトウェアの実装に関する米国政府の標準です。米国政府に販売される多くの製品は、1 つ以上の FIPS 標準に準拠する必要があります。http://www.nist.gov/itl/fipscurrent.cfm を参照してください。

ファイアウォール (firewall)

2 つ以上のネットワーク間の境界を強制するシステムまたはシステムの組み合わせ。

フィンガープリント (fingerprint)

証明書のフィンガープリント (certificate fingerprint) を参照してください。

H

Hypertext Transport Protocol (HTTP) および Hypertext Transport Protocol Secure (HTTPS)

Web サーバーとの通信に使用されるプロトコル。HTTP S は、TLS (Transport Layer Security) によって暗号化された接続内の HTTP (Hypertext Transfer Protocol) を介した通信で設定されます。HTTPS の主な目的は、アクセスした Web サイトの認証と、交換されたデータのプライバシーと整合性の保護です。

I

IP スプーフィング (IP spoofing)

クライアント IP アドレスの禁止。

IPv4 および IPv6 (IPv4 and IPv6)

Certificate System は、すべてのサブシステムとツールとの通信と操作、およびクライアント、サブシステムの作成、トークンと証明書の登録のために、IPv4 と IPv6 の両方のアドレス名前空間をサポートします。

なりすまし (impersonation)

ネットワークを介して送信される情報の意図された受信者を装う行為。なりすましには、スプーフィング (spoofing) と 詐称 (misrepresentation) の 2 つの形式があります。

中間認証局 (intermediate CA)

ルート CA と 証明書チェーン (certificate chain) で発行した証明書との間の証明書のある CA。

入力 (input)

証明書プロファイル機能のコンテキストでは、特定の証明書プロファイルの登録フォームを定義します。各入力が設定され、この登録用に設定されたすべての入力から登録フォームが動的に作成されます。

J

JAR ファイル (JAR file)

Java™ アーカイブ (JAR) 形式 (archive (JAR) format) に従って編成されたファイルの圧縮コレクションのデジタルエンベロープ。

Java™ アーカイブ (JAR) 形式 (archive (JAR) format)

デジタル署名、インストーラスクリプト、およびその他の情報をディレクトリー内のファイルに関連付けるための一連の規則。

Java™ セキュリティーサービス (JSS)

あJava™ネットワークセキュリティーサービス (NSS) によって実行されるセキュリティー操作を制御するためのインターフェイス。

Java™ ネイティブインターフェイス (JNI) (Native Interface)

特定のプラットフォーム上の Java™ 仮想マシン (JVM) の異なる実装間でバイナリー互換性を提供する標準的なプログラミングインターフェイスで、単一のプラットフォーム用に C や C++ などの言語で記述された既存のコードを Java™ にバインドできるようにします。http://java.sun.com/products/jdk/1.2/docs/guide/jni/index.html を参照してください。

Java™ 暗号アーキテクチャー (Cryptography Architecture, JCA)

暗号化サービス用に Sun Microsystems によって開発された API 仕様および参照。http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.Introduction を参照してください。

Java™ 開発キット (Development Kit, JDK)

Java™ プログラミング言語を使用してアプリケーションとアプレットを開発するために Sun Microsystems が提供するソフトウェア開発キット。

K

KEA

鍵交換アルゴリズム (Key Exchange Algorithm, KEA) を参照してください。

KEYGEN タグ (tag)

証明書で使用するキーペアを生成する HTML タグ。

鍵 (key)

データを暗号化または復号化するために、暗号アルゴリズム (cryptographic algorithm) が使用する多数の数値。たとえば、あるユーザーの 公開鍵 (public key) にあるユーザーは、そのユーザー専用のメッセージを暗号化できます。その後、メッセージは対応する プライベートキー (private key) を使用して復号化する必要があります。

鍵交換 (key exchange)

クライアントとサーバーが SSL セッション時に両方で使用する対称鍵を決定するために実行する手順。

鍵交換アルゴリズム (Key Exchange Algorithm, KEA)

米国政府が鍵交換に使用するアルゴリズム。

L

Lightweight Directory Access Protocol (LDAP)

TCP/IP および複数のプラットフォームで実行されるためのディレクトリーサービスプロトコル。LDAP は、X.500 ディレクトリーへのアクセスに使用される Directory Access Protocol (DAP) の簡易バージョンです。LDAP は IETF の変更制御下にあり、インターネット要件を満たすために進化しています。

リンクされた CA (linked CA)

公開サードパーティーの CA によって署名される証明書が内部でデプロイされる 認証局 (certificate authority, CA)。内部 CA は、発行する証明書のルート CA として機能し、サードパーティー CA は、同じサードパーティールート CA にリンクされている他の CA によって発行された証明書のルート CA として機能します。チェーン CA としても知られており、異なるパブリック CA で使用される他の用語も使用します。

M

MD5

Ronald Rivest によって開発されたメッセージダイジェストアルゴリズム。一方向ハッシュ (one-way hash) も併せて参照してください。

メッセージダイジェスト (message digest)

一方向ハッシュ (one-way hash) を参照してください。

手動認証 (manual authentication)

各証明書要求の人間による承認を必要とする Certificate System サブシステムを設定する方法。この形式の認証では、サーブレットは、認証モジュールの処理が成功した後、証明書要求を要求キューに転送します。次に、適切な権限を持つエージェントは、プロファイルの処理と証明書の発行を続行する前に、各要求を個別に承認する必要があります。

詐称 (misrepresentation)

そうではない個人または組織としてのエンティティーの提示。たとえば、Web サイトが実際にはクレジットカードでの支払いを行いますが、商品を送信しないサイトである場合、その Web サイトは家具店のふりをする可能性があります。虚偽表示は なりすまし (impersonation) の 1 つの形式です。スプーフィング (spoofing) も併せて参照してください。

N

non-TMS

トークン以外の管理システム。スマートカードを直接処理 **しない** サブシステム (CA、および任意で KRA と OCSP) の設定を指します。

トークン管理システム (token management system, TMS) 参照

ネットワークセキュリティーサービス (Network Security Services, NSS)

セキュリティー対応の通信アプリケーションのクロスプラットフォーム開発をサポートするように設計されたライブラリーのセット。NSS ライブラリーを使用して構築されたアプリケーションは、認証、改ざん検出、および暗号化のための セキュアソケットレイヤー (Secure Sockets Layer, SSL) プロトコル、および暗号化トークンインターフェイスのための PKCS #11 プロトコルをサポートします。NSS は、ソフトウェア開発キットとして個別に利用できます。

否認防止 (nonrepudiation)

メッセージの送信を拒否するためのメッセージの送信者による信頼性。デジタル署名 (digital signature) は、否認防止の形式を 1 つ提供します。

O

OCSP

オンライン証明書ステータスプロトコル

オブジェクトの署名 (object signing)

ソフトウェア開発者が Java コード、JavaScript スクリプト、またはあらゆる種類のファイルに署名し、ユーザーが署名者を識別し、署名されたコードによってローカルシステムリソースへのアクセスを制御できるようにするファイル署名の方法。

オブジェクト署名証明書 (object-signing certificate)

秘密鍵に関連付けられた証明書は、オブジェクトの署名 (object signing) に関連するオブジェクトの署名に使用されます。

一方向ハッシュ (one-way hash)

1.ハッシュアルゴリズムを使用して任意の長さのデータから生成された固定長の数。メッセージダイジェストとも呼ばれる数字は、ハッシュされたデータに固有のものです。1 文字を削除または変更しても、データの変更は異なります。

2.ハッシュされたデータの内容をハッシュから推測することはできません。

出力 (output)

証明書プロファイル機能のコンテキストでは、特定の証明書プロファイルの証明書登録が成功した結果のフォームを定義します。各出力が設定され、この登録に設定されたすべての出力からフォームを動的に作成します。

操作 (operation)

アクセス制御命令で許可または拒否されている、読み取りや書き込みなどの特定の操作。

P

PKCS #10

証明書要求を制御する公開鍵暗号標準規格。

PKCS #11

スマートカードなどの暗号化トークンを管理する公開鍵暗号化標準。

PKCS #11 モジュール

PKCS #11 インターフェイスを介して、暗号化サービス (暗号化や復号など) を提供する暗号化デバイスのドライバー。**暗号化モジュール** または **暗号化サービスプロバイダー** とも呼ばれる PKCS #11 モジュールは、ハードウェアまたはソフトウェアのいずれかで実装できます。PKCS #11 モジュールには、常に 1 つ以上のスロットがあり、スマートカードなどの物理リーダーの形式で物理ハードウェアスロットとして、またはソフトウェアの概念スロットとして実装できます。PKCS #11 モジュールの各スロットには、トークンを追加できます。このトークンは、暗号化サービスを実際に提供し、必要に応じて証明書と鍵を保存するハードウェアまたはソフトウェアのデバイスです。Red Hat は、Certificate System とともに、ビルトイン PKCS #11 モジュールを提供します。

PKCS #12

鍵の移植性を管理する公開鍵暗号化標準。

PKCS #7

署名と暗号化を制御する公開鍵暗号標準。

PKIX 証明書および CRL プロファイル (PKIX Certificate and CRL Profile)

インターネット用の公開鍵インフラストラクチャー向けに IETF により開発された標準規格です。証明書および CRL のプロファイルを指定します。

POA (proof-of-archival)

キーのシリアル番号、キーリカバリー機関の名前、対応する証明書の サブジェクト名 (subject name)、およびアーカイブの日付など、アーカイブされたエンドエンティティーキーに関する情報

を含む秘密キーリカバリー機関のトランスポートキーで署名されたデータ。署名されたアーカイブ証明データは、キーのアーカイブ操作が成功した後、キーリカバリー機関から Certificate Manager に返される応答です。キーリカバリー認証局のトランスポート証明書 (Key Recovery Authority transport certificate) も併せて参照してください。

パスワードベースの認証 (password-based authentication)

名前とパスワードによる確実な識別。認証 (authentication)、証明書ベースの認証 (certificate-based authentication) も併せて参照してください。

プライベートキー (private key)

公開鍵暗号で使用されるキーペアの 1 つ。秘密鍵は秘密を保持し、対応する 公開鍵 (public key) で暗号化されたデータの復号に使用されます。

公開鍵 (public key)

公開鍵暗号で使用されるキーペアの 1 つ。公開鍵は自由に配布され、証明書 (certificate) の一部として公開されます。これは通常、公開鍵の所有者に送信されるデータを暗号化するために使用され、所有者は対応する プライベートキー (private key) でデータを復号化します。

公開鍵インフラストラクチャー (public-key infrastructure, PKI)

ネットワーク環境での公開鍵暗号化と X.509 v3 証明書の使用を容易にする標準とサービス。

公開鍵暗号 (public-key cryptography)

エンティティーがその ID を電子的に検証したり、電子データに署名して暗号化したりできるようにする、確立された技術と標準のセット。公開鍵と秘密鍵の 2 つの鍵が関係します。公開鍵 (public key) は、特定の ID にキーを関連付ける証明書の一部として公開されます。対応する秘密鍵はシークレットに保存されます。公開鍵で暗号化したデータは、秘密鍵でのみ復号できます。

R

RC2, RC4

Rivest による RSA データセキュリティー向けに開発された暗号化アルゴリズム。暗号アルゴリズム (cryptographic algorithm) も併せて参照してください。

Red Hat Certificate System

証明書を作成、デプロイ、および管理するための高度に設定可能なソフトウェアコンポーネントとツールのセット。Certificate System は、さまざまな物理的な場所にあるさまざまな Certificate System インスタンスにインストールできる 5 つの主要なサブシステム (Certificate Manager、Online Certificate Status Manager、 キーリカバリー認証局、Token Key Service, and Token Processing System) で設定されています。

RSA アルゴリズム (RSA algorithm)

Rivest-Shamir-Adleman の略で、暗号化と認証の両方の公開鍵アルゴリズムです。Ronald Rivest、Adi Shamir、および Leonard Adleman により開発され、1978 で導入されました。

RSA キー交換 (RSA key exchange)

RSA アルゴリズムに基づいた SSL の鍵交換アルゴリズム。

ルート CA

証明書チェーンの上部に自己署名証明書が含まれている 認証局 (certificate authority, CA)。CA 証明書 (CA certificate)、下位 CA (subordinate CA) も併せて参照してください。

登録 (registration)

エンロールメント (enrollment) を参照してください。

S

SELinux (Security Enhanced Linux)

SELinux (Security-Enhanced Linux) は、Linux システムカーネルに強制アクセス制御を適用するセキュリティープロトコルのセットです。SELinux は、米国国家安全保障局によって開発され、寛大なアクセス制御または欠陥のあるアクセス制御を介してアプリケーションが機密ファイルまたは保護されたファイルにアクセスするのを防ぎます。

SHA

セキュアなハッシュアルゴリズム (米国政府が使用するハッシュ関数)。

SSL

セキュアソケットレイヤー (Secure Sockets Layer, SSL) を参照してください。

サブジェクト (subject)

証明書 (certificate) で識別されるエンティティー。特に、証明書のサブジェクトフィールドには、認定されたエンティティーを一意に識別する サブジェクト名 (subject name) が含まれます。

サブジェクト名 (subject name)

証明書 (certificate) の サブジェクト (subject) を個別に記述する 識別名 (distinguished name, DN)。

サンドボックス (sandbox)

Java™ コードが動作する必要がある、注意して定義された制限の Java™ 用語。

サーバー SSL 証明書 (server SSL certificate)

セキュアソケットレイヤー (Secure Sockets Layer, SSL) プロトコルを使用して、クライアントにサーバーを識別するために使用する証明書。

サーバー認証 (server authentication)

クライアントにサーバーを特定するプロセス。クライアント認証 (client authentication) も併せて参照してください。

サーブレット (servlet)

Certificate System サブシステムに代わってエンドエンティティーとの特定の種類の相互作用を処理する Java™ コード。たとえば、証明書の登録、失効、およびキーリカバリー要求は、それぞれ別のサーブレットで処理されます。

シングルサインオン (single sign-on)

1.Certificate System において、内部データベースとトークンのパスワードを保管することにより、Red Hat Certificate System へのサインオン方法を簡素化するパスワード。ユーザーがログインするたびに、この単一のパスワードを入力する必要があります。

2.ユーザーが 1 台のコンピューターに一度ログインし、ネットワーク内のさまざまなサーバーによっ

て自動的に認証される機能。部分的なシングルサインオンソリューションは、さまざまなサーバーで使用されるパスワードを自動的に追跡するメカニズムなど、さまざまな形式をとることができます。証明書は、公開鍵インフラストラクチャー (public-key infrastructure, PKI) 内のシングルサインオンをサポートします。ユーザーは、ローカルクライアントの秘密鍵データベースに一度ログインし、クライアントソフトウェアが動作している限り、証明書ベースの認証 (certificate-based authentication) を使用して、ユーザーがアクセスを許可されている組織内の各サーバーにアクセスすることができます。

スプーフィング (spoofing)

他人のふりをします。たとえば、あるユーザーがメールアドレス **jdoe@example.com** やコンピューターから、**www.redhat.com** と呼ばれるサイトとして誤って特定できます。なりすましは、なりすまし (impersonation) の 1 つの形です。詐称 (misrepresentation) も併せて参照してください。

スマートカード (smart card)

マイクロプロセッサーを搭載し、キーや証明書などの暗号化情報を格納し、暗号化操作を実行する小さなデバイス。スマートカードは、一部またはすべて PKCS #11 インターフェイスを実装します。

スロット (slot)

PKCS #11 モジュール の一部 (ハードウェアまたはソフトウェアのいずれか)。これには トークン (token) が含まれています。

セキュアなチャンネル (secure channel)

TPS とスマートカード間のセキュリティーアソシエーション。TKS とスマートカード APDU によって生成された共有マスターキーに基づいて暗号化された通信を可能にします。

セキュアソケットレイヤー (Secure Sockets Layer, SSL)

クライアントとサーバーとの間の相互認証と、認証および暗号化された接続の確立を可能にするプロトコル。SSL は、TCP/IP より上で、HTTP、LDAP、IMAP、NNTP、およびその他の高レベルネットワークプロトコルより下で実行されます。

セキュリティードメイン (security domain)

PKI サブシステムの集中リポジトリーまたはインベントリー。その主な目的は、サブシステム間の信頼できる関係を自動的に確立することにより、新しい PKI サービスのインストールと設定を容易にすることです。

セルフテスト (self tests)

インスタンスの起動時とオンデマンドの両方の Certificate System インスタンスをテストする機能。

下位 CA (subordinate CA)

証明書を行う認証局は、別の下位 CA またはルート CA により署名されます。CA 証明書 (CA certificate)、ルート CA を参照してください。

対象暗号化 (symmetric encryption)

同じ暗号化キーを使用して特定のメッセージを暗号化および復号する暗号化方式。

簡易証明書登録プロトコル (Simple Certificate Enrollment Protocol, SCEP)

ルーターがルーター証明書登録のために CA と通信する方法を指定するためにシスコによって設計されたプロトコル。証明書システムは、要求が CA 署名証明書で暗号化される SCEP の CA 動作モードをサポートします。

署名アルゴリズム (signature algorithm)

デジタル署名の作成に使用される暗号化アルゴリズム。Certificate System は、MD5 および SHA 署名アルゴリズムに対応します。暗号アルゴリズム (cryptographic algorithm)、デジタル署名 (digital signature) も併せて参照してください。

署名付き監査ログ (signed audit log)

監査ログ (autdit log) を参照してください。

署名証明書 (signing certificate)

公開鍵である証明書は、デジタル署名の作成に使用される秘密鍵に対応します。たとえば、Certificate Manager には、発行する証明書の署名に使用する秘密鍵に対応する公開鍵である署名証明書が必要です。

署名鍵 (signing key)

署名用途に使用する秘密鍵署名鍵とその同等の公開鍵、および 暗号鍵 (encryption key) とその同等の公開鍵は、デュアルキーペア (dual key pair) を設定します。

T

token key service (トークンキーサービス、TKS)

スマートカードの APDU およびトークン CUID などの他の共有情報に基づいて、スマートカードごとに特定の個別のキーを取得するトークン管理システムのサブシステム。

ツリー階層 (tree hierarchy)

LDAP ディレクトリーの階層構造。

トランスポートレイヤーセキュリティー (Transport Layer Security, TLS)

サーバー認証、クライアント認証、およびサーバーとクライアントとの間の暗号化通信を管理する一連のルール。

トークン (token)

PKCS #11 モジュール の スロット (slot) に関連付けられたハードウェアまたはソフトウェアのデバイス暗号化サービスを提供し、必要に応じて証明書および鍵を保存します。

トークン処理システム (token processing system, TPS)

Enterprise Security Client とスマートカードを直接対話して、これらのスマートカードのキーと証明書を管理するサブシステム。

トークン管理システム (token management system, TMS)

相互に関連するサブシステム (CA、TKS、TPS、および任意で KRA) は、スマートカード (トークン) の証明書を管理するために使用されます。

信頼 (trust)

個人または他のエンティティーに確定します。公開鍵インフラストラクチャー (public-key infrastructure, PKI) では、信頼とは、証明書のユーザーと、その証明書を発行した 認証局

(certificate authority, CA) との間の関係を指します。CA が信頼されている場合は、その CA が発行する有効な証明書を信頼できます。

改ざんの検出 (tamper detection)

電子形式で受信したデータが同じデータの元のバージョンと完全に対応することを保証するメカニズム。

U

UTF-8

証明書の登録ページは、特定のフィールド (共通名、組織単位、要求者名、および追加のメモ) で UTF-8 文字をサポートします。UTF-8 文字列は、CA、OCSP、および KRA エンドユーザーおよびエージェントサービスページで検索可能かつ正しく表示されます。ただし、UTF-8 のサポートは、電子メールアドレスで使用されるような国際化ドメイン名には拡張されません。

V

仮想プライベートネットワーク (virtual private network, VPN)

企業の地理的に離れた部署を接続する方法。VPN を使用すると、部署間は暗号化されたチャンネルを介して通信できるため、通常はプライベートネットワークに制限される認証済みの機密トランザクションが可能になります。

X

X.509 バージョン 1 およびバージョン 3 (X.509 version 1 and version 3)

国際電気通信連合 (ITU) が推奨するデジタル証明書形式。

索引

シンボル

D

K

O

P

R

S

T

W

付録A 改訂履歴

改訂番号は本ガイドに関するものであり、Red Hat Certificate System のバージョン番号とは関係ありません。

改訂 10.3-0 **Tue Feb 22 2022** **Florian Delehaye**
10.3 のさまざまな修正。

改訂 10.2-0 **Wed Jul 14 2021** **Florian Delehaye**
10.2 のさまざまな修正。

改訂 10.1-2 **Fri Feb 27 2021** **Florian Delehaye**
FIPS モードで HSM を設定するための修正およびさまざまなマイナーな編集。

改訂 10.1-1 **Mon Jan 25 2021** **Florian Delehaye**
さまざまな修正および改善点。

改訂 10.1-0 **Wed Dec 3 2020** **Florian Delehaye**
Red Hat Certificate System 10.1 ガイドのリリース

改訂 10.0-1 **Tues Nov 3 2020** **Florian Delehaye**
RHCS 10 へのアップグレードおよび移行手順を追加。

改訂 10.0-0 **Thur Sep 17 2020** **Florian Delehaye**
Red Hat Certificate System 10.0 ガイドのリリース