Oracle® Solaris 11.2 での暗号化と証明書の管理

Part No: E53959-02
2014 年 9 月

ORACLE

目次

このドキュメントの使用

『*Oracle® Solaris 11.2 での暗号化および証明書の一元管理*』では、暗号化を管理および使用する方法、および非公開/公開鍵証明書を作成および管理する方法について説明します。

- **概要** – 暗号化フレームワークおよび鍵管理フレームワークに関する概念と、これらのテクノロジを使用してファイルをセキュリティー保護するタスクについて説明します。
- **対象読者** – 企業でセキュリティーを実装する必要のあるシステム管理者。
- **前提知識** – セキュリティーに関する概念と用語に精通していること。

製品ドキュメントライブラリ

この製品の最新情報や既知の問題は、ドキュメントライブラリ (http://www.oracle.com/pls/topic/lookup?ctx=E56342) に含まれています。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support を通じて電子的なサポートを利用することができます。詳細は、http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info (聴覚に障害をお持ちの場合は http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs) を参照してください。

フィードバック

このドキュメントに関するフィードバックを http://www.oracle.com/goto/docfeedback からお聞かせください。

◆ ◆ ◆ 第 1 章

暗号化フレームワーク

この章では、Oracle Solaris の暗号化フレームワーク機能について説明します。この章の内容は次のとおりです。

暗号化フレームワークを管理および使用するには、第3章「暗号化フレームワーク」を参照してください。

Oracle Solaris 11.2 の暗号化の新機能

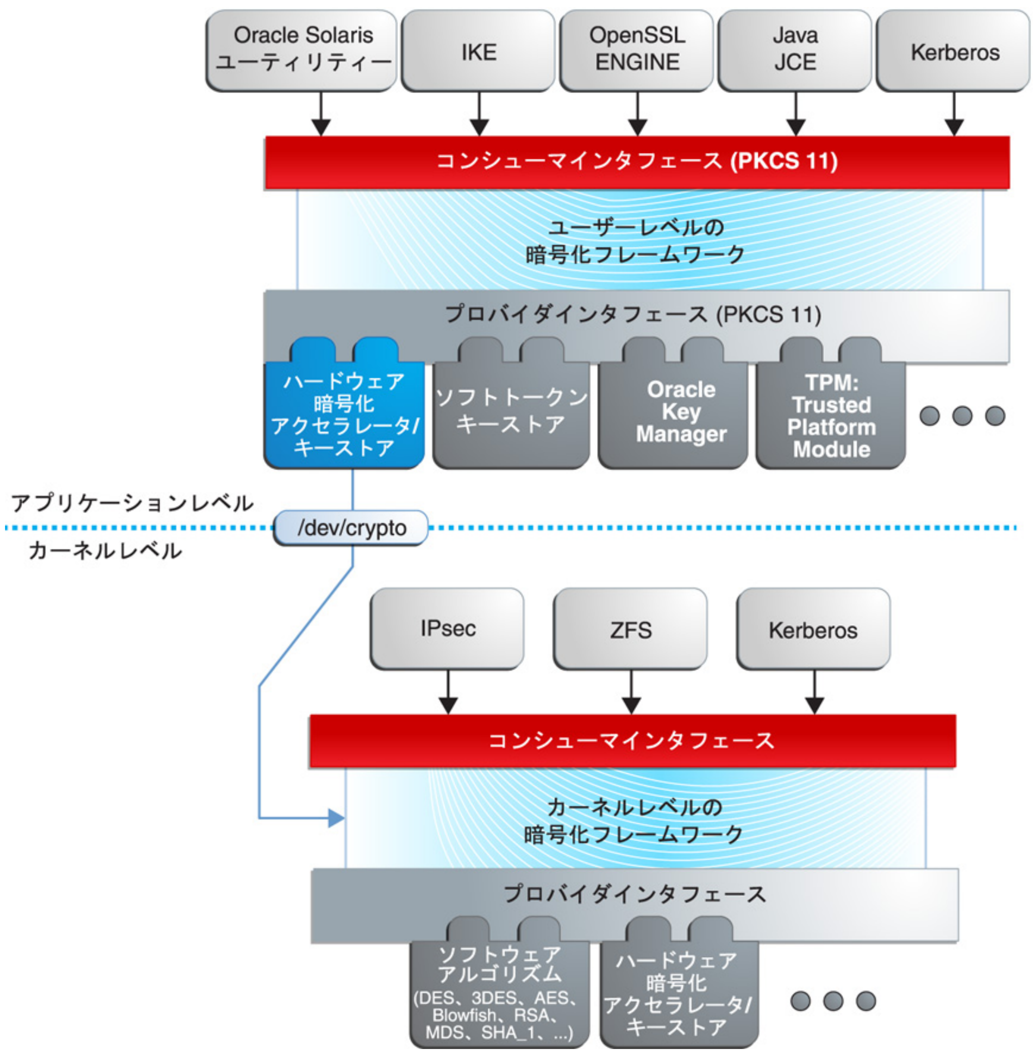
このセクションでは、このリリースの暗号化サポートの新機能について、既存の顧客に重点的に説明します。

- Oracle Solaris では、OpenSSL の FIPS 対応および FIPS 非対応の両方のバージョンをサポートしています。
- 暗号化が最適化された SPARC T4 システムでは、迅速な暗号化操作の実行を可能にする暗号化命令をハードウェアで直接使用できます。
- 暗号化フレームワークでは、128 ビットブロック暗号である Camellia をサポートしており、これは、AES に似ており、日本市場でもっとも使用されています。

暗号化フレームワークの紹介

暗号化フレームワークは、暗号化要求を処理するアルゴリズムと PKCS #11 ライブラリの共通の格納場所を提供します。PKCS #11 ライブラリは、RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki) 標準に従って実装されます。

図 1-1 暗号化フレームワークのレベル



暗号化フレームワークは、現在、ZFS、Kerberos、IPsec、およびハードウェアに対する暗号化要求をカーネルレベルで処理します。ユーザーレベルのコンシューマには、OpenSSL エンジン、Java Cryptographic Extensions (JCE)、`libsasl`、および IKE (Internet Key Protocol) が含まれます。カーネルの SSL (`kssl`) プロキシでは、暗号化フレームワークを使用します。詳細は、『Oracle Solaris 11.2 でのネットワークのセキュリティー保護 』の「SSL カーネルプロキシ は Web サーバー通信を暗号化する」および `ksslcfg`(1M) のマニュアルページを参照してください。

米国の輸出法では、公開された暗号化インタフェースの使用はライセンス供与が義務付けられています。暗号化フレームワークは、カーネル暗号化プロバイダおよび PKCS #11 暗号化プロバイダの署名を義務付けることにより、この現行法を満たしています。詳細は、12 ページの「暗号化フレームワークのユーザーレベルコマンド」の `elfsign` コマンドに関する情報を参照してください。

このフレームワークにより、暗号化サービスの*プロバイダ*は、そのサービスが Oracle Solaris の多数の*コンシューマ*に使用されるようにすることができます。プロバイダは*プラグイン*とも言います。フレームワークでは、3 種類のプラグインがサポートされます。

- ユーザーレベルプラグイン – `/var/user/$USER/pkcs11_softtoken.so.1` など、PKCS #11 ライブラリを使用することによってサービスを提供する共有オブジェクト。
- カーネルレベルプラグイン –AES など、ソフトウェアの暗号化アルゴリズムの実装を提供するカーネルモジュール。

  暗号化フレームワークのアルゴリズムの多くは、SSE2 命令セットを備えた x86 および SPARC ハードウェア用に最適化されます。T シリーズの最適化については、19 ページの「暗号化フレームワークと SPARC T シリーズサーバー」を参照してください。
- ハードウェアプラグイン – デバイスドライバおよび関連するハードウェアアクセラレータ。たとえば、Niagara チップ、Oracle の `ncp` および `n2cp` デバイスドライバです。ハードウェアアクセラレータにより、オペレーティングシステムの高価な暗号化機能が肩代わりされます。たとえば、Sun Crypto Accelerator 6000 ボードがあります。

このフレームワークは、ユーザーレベルプロバイダ用に標準インタフェースとして PKCS #11 v2.20 Amendment 3 ライブラリを実装しています。このライブラリは、サードパーティーのアプリケーションがプロバイダに到達するために使用できます。サードパーティーは、署名付きライブラリ、署名付きカーネルアルゴリズムモジュール、および署名付きデバイスドライバを暗号化フレームワークに追加することもできます。これらのプラグインは、Image Packaging System (IPS) によってサードパーティーのソフトウェアがインストールされると追加されます。フレームワークの主要コンポーネントの図については、図1-1「暗号化フレームワークのレベル」 を参照してください。

暗号化フレームワークの概念

次の概念の説明および対応する例は、暗号化フレームワークでの作業時に役立ちます。

- **アルゴリズム** – 暗号化アルゴリズムは、入力を暗号化またはハッシュする確立された再帰的な計算手続きです。暗号化アルゴリズムには対称と非対称があります。対称アルゴリズムでは、暗号化と復号化に同じ鍵が使用されます。非対称アルゴリズムは、公開鍵暗号化で使用され、2 つの鍵を必要とします。ハッシングもアルゴリズムです。

  アルゴリズムの例として、次のものがあります。

  - AES や ECC などの対称アルゴリズム
  - Diffie-Hellman や RSA などの非対称アルゴリズム
  - SHA256 などのハッシング機能
- **コンシューマ** – プロバイダから提供される暗号化サービスのユーザー。コンシューマになりえるものとして、アプリケーション、エンドユーザー、カーネル処理などが挙げられます。

  コンシューマの例として、次のものがあります。

  - IKE などのアプリケーション
  - `encrypt` コマンドを実行する通常のユーザーなどのエンドユーザー
  - IPsec などのカーネル操作
- **キーストア** – 暗号化フレームワークでは、トークンオブジェクトのための永続的なストレージであり、多くの場合は**トークン**と同じ意味で使用されます。予約されたキーストアについては、この定義のリストにある**メタスロット**を参照してください。
- **メカニズム** – 特定の目的でアルゴリズムのモードを応用したもの。

  たとえば、認証に適用される DES メカニズム (CKM_DES_MAC など) は、暗号化に適用されるメカニズム (CKM_DES_CBC_PAD) とは別です。
- **メタスロット** – フレームワークにロードされているほかのスロットの機能を統合した単一のスロット。メタスロットは、フレームワークを通じて利用可能なプロバイダのすべての機能を取り扱う際の作業を軽減します。メタスロットを使用するアプリケーションから処理リクエストを受けると、実際のスロットのうちどのスロットがその処理を行うのかをメタスロットが決定します。メタスロットの機能は構成可能ですが、構成が必須ではありません。メタスロットはデフォルトでは有効になっています。詳細は、`cryptoadm`(1M) のマニュアルページを参照してください。

  メタスロットには、独自のキーストアはありません。代わりに、メタスロットは、暗号化フレームワーク内の実際のスロットのいずれかからキーストアの使用を予約します。デフォルトでは、

メタスロットは Sun Crypto Softtoken キーストアを予約します。メタスロットによって使用されているキーストアは、使用可能なスロットの 1 つとしては示されません。

ユーザーは、環境変数 ${METASLOT_OBJECTSTORE_SLOT} および ${METASLOT_OBJECTSTORE_TOKEN} を設定するか、または cryptoadm コマンドを実行することによって、メタスロットのための代わりのキーストアを指定できます。詳細は、libpkcs11(3LIB)、pkcs11_softtoken (5)、および cryptoadm(1M) の各マニュアルページを参照してください。

- **モード** – 暗号化アルゴリズムのバージョン。たとえば、CBC (Cipher Block Chaining) は、ECB (Electronic Code Book) とは別のモードです。AES アルゴリズムには、CKM_AES_ECB と CKM_AES_CBC の 2 つのモードがあります。
- **ポリシー** – どのメカニズムを使用できるようにするかについての管理者による選択。デフォルトでは、すべてのプロバイダとすべてのメカニズムが使用可能です。メカニズムを有効または無効にすることは、ポリシーの適用です。ポリシーの設定および適用の例については、38 ページの「暗号化フレームワークの管理」を参照してください。
- **プロバイダ** – コンシューマが使用する暗号化サービス。プロバイダは、フレームワークに組み込まれる(プラグインする)ので、*プラグイン*とも呼ばれます。

  プロバイダの例として、次のものがあります。

  - /var/user/$USER/pkcs11_softtoken.so などの PKCS #11 ライブラリ
  - aes や arcfour などの暗号化アルゴリズムのモジュール
  - Sun Crypto Accelerator 6000 の mca ドライバなど、デバイスドライバおよび関連するハードウェアアクセラレータ
- **スロット** – 1 つまたは複数の暗号化デバイスとのインタフェース。各スロットは物理的なリーダー (読み取り器) またはその他のデバイスインタフェースに相当し、スロットにはトークンが 1 つ搭載されていることがあります。トークンは、フレームワーク内の暗号化デバイスを論理的に表示します。
- **トークン** – スロット内で、トークンはフレームワーク内の暗号化デバイスの論理表示を提供します。

暗号化フレームワークのコマンドとプラグイン

フレームワークには、管理者、ユーザー、およびプロバイダを提供する開発者向けのコマンドが用意されています。

- 管理コマンド – cryptoadm コマンドは、使用可能なプロバイダとその機能を一覧表示する list サブコマンドを提供します。通常のユーザーは、cryptoadm list コマンドおよび cryptoadm --help コマンドを実行できます。

  それ以外の cryptoadm サブコマンドでは、Crypto Management 権利プロファイルを含む役割になるか、スーパーユーザーになる必要があります。disable、install、および uninstall などのサブコマンドを使用して、暗号化フレームワークを管理できます。詳細は、cryptoadm(1M) のマニュアルページを参照してください。

  svcadm コマンドを使用して、kcfd デーモンの管理やカーネルの暗号化ポリシーのリフレッシュを行うことができます。詳細は、svcadm(1M) のマニュアルページを参照してください。
- ユーザーレベルコマンド –digest コマンドおよび mac コマンドによって、ファイル整合性サービスが提供されます。encrypt および decrypt コマンドは、ファイルが傍受されるのを防ぎます。これらのコマンドを使用するには、表3-1「暗号化フレームワークによるファイルの保護のタスクマップ」を参照してください。

暗号化フレームワークの管理コマンド

cryptoadm コマンドは、動作中の暗号化フレームワークを管理します。このコマンドは、Crypto Management 権利プロファイルの一部です。このプロファイルは、暗号化フレームワークのセキュアな管理のための役割に割り当てることができます。cryptoadm コマンドを使用して、次を実行します。

- プロバイダメカニズムを無効または有効にする
- メタスロットを無効または有効にする

svcadm コマンドは、暗号化サービスデーモン kcfd を有効化、リフレッシュ、および無効化するために使用されます。このコマンドは、Oracle Solaris のサービス管理機能 (SMF) の機能の一部です。svc:/system/cryptosvcs は、暗号化フレームワークのサービスインスタンスです。詳細は、smf(5) および svcadm(1M) のマニュアルページを参照してください。

暗号化フレームワークのユーザーレベルコマンド

暗号化フレームワークは、ファイルの整合性の確認、ファイルの暗号化、およびファイルの復号化を行うユーザーレベルコマンドを提供します。

- `digest` コマンド – 1 つまたは複数のファイルまたは標準入力のメッセージダイジェストを計算します。ダイジェストは、ファイルの整合性を検証するのに便利です。SHA1 および MD5 は、ダイジェスト機能の例です。
- `mac` コマンド – 1 つまたは複数のファイルまたは標準入力のメッセージ認証コード (MAC) を計算します。MAC は、データを認証されたメッセージに関連付けます。MAC によって、受信者は、メッセージの送信者、およびメッセージが改ざんされていないことを検証できるようになります。`sha1_mac` メカニズムおよび `md5_hmac` メカニズムが MAC を計算します。
- `encrypt` コマンド – 対称暗号でファイルまたは標準入力を暗号化します。`encrypt -l` コマンドは、使用可能なアルゴリズムを一覧表示します。ユーザーレベルライブラリで一覧表示されるメカニズムは、`encrypt` コマンドで使用可能です。暗号化フレームワークでは、ユーザーの暗号化のために AES、DES、3DES (Triple-DES)、および ARCFOUR メカニズムが用意されています。
- `decrypt` コマンド – `encrypt` コマンドで暗号化されたファイルまたは標準入力を復号化します。`decrypt` コマンドは、元のファイルの暗号化に使用されたのと同一の鍵とメカニズムを使用します。
- `elfsign` コマンド – 暗号化フレームワークでの使用のためにプロバイダに署名する手段を提供します。一般に、このコマンドはプロバイダの開発者によって実行されます。`elfsign` コマンドには、証明書のリクエスト、バイナリへの署名、およびバイナリ上の署名の検証を行うためのサブコマンドがあります。署名されていないバイナリは、暗号化フレームワークで使用できません。検証可能な署名付きのバイナリを持っているプロバイダは、そのフレームワークを使用できます。

暗号化フレームワークのプラグイン

サードパーティーは、暗号化フレームワークに自身のプロバイダを接続できます。サードパーティーのプロバイダとは、次のオブジェクトのいずれかです。

- PKCS #11 共有ライブラリ
- 暗号化アルゴリズム、MAC 機能、ダイジェスト機能など、ロード可能なカーネルソフトウェアモジュール
- ハードウェアアクセラレータ用のカーネルデバイスドライバ

プロバイダのオブジェクトは、Oracle からの証明書を使用して署名されている必要があります。証明書要求は、サードパーティーが選択する非公開鍵と Oracle が提供する証明書に基づきます。証明書要求は Oracle に送信され、サードパーティーが登録されたあと、証明書が発行

されます。次にサードパーティーは Oracle からの証明書を使用してそのプロバイダオブジェクトに署名します。

ロード可能なカーネルソフトウェアモジュールおよびハードウェアアクセラレータ用のカーネルデバイスドライバも、カーネルに登録する必要があります。登録は、暗号化フレームワークの SPI (サービスプロバイダインタフェース) 経由で行います。

暗号化サービスとゾーン

大域ゾーンと各非大域ゾーンには、それぞれの `/system/cryptosvc` サービスが用意されています。大域ゾーンで暗号化サービスが有効になったりリフレッシュされたりすると、大域ゾーンで `kcfd` デーモンが起動され、大域ゾーンに対するユーザーレベルポリシーが設定され、システムに対するカーネルポリシーが設定されます。非大域ゾーンでサービスが有効になったりリフレッシュされたりすると、その非大域ゾーンで `kcfd` デーモンが起動され、そのゾーンに対するユーザーレベルポリシーが設定されます。カーネルポリシーは、大域ゾーンによって設定されています。

ゾーンの詳細は、『Oracle Solaris ゾーンの紹介 』を参照してください。SMF を使用して永続的なアプリケーションを管理する方法の詳細は、『Oracle Solaris 11.2 でのシステムサービスの管理 』の第 1 章「サービス管理機能の概要」および `smf(5)` のマニュアルページを参照してください。

暗号化フレームワークおよび FIPS 140

FIPS 140 は、暗号化モジュールのための米国政府のコンピュータセキュリティー標準です。Oracle Solaris システムでは、FIPS 140-2 レベル 1 に対して承認された、暗号化アルゴリズムの 2 つのプロバイダが提供されます。

これらのプロバイダは次のとおりです。

- Oracle Solaris の暗号化フレームワークでは、FIPS 140 承認の 2 つのモジュールが提供されます。*ユーザーランド*モジュールは、ユーザー空間で実行されるアプリケーションのための暗号化を提供します。*カーネル*モジュールは、カーネルレベルのプロセスのための暗号化を提供します。
- OpenSSL オブジェクトモジュールは、SSH および Web アプリケーションのための FIPS 140 承認の暗号化を提供します。

次の重要な考慮事項に留意してください。

- FIPS 140-2 プロバイダモジュールは CPU を集中的に使用するため、デフォルトでは有効になっていません。システム管理者には、FIPS 140 モードでこれらのプロバイダを有効にし、FIPS 承認アルゴリズムを使用するアプリケーションを構成する責任があります。
- FIPS 140-2 で検証された暗号化のみを使用するという厳格な要件がある場合は、Oracle Solaris11.1 SRU5.5 リリースまたは Oracle Solaris11.1 SRU3 リリースを実行する必要があります。Oracle は、これらの 2 つの特定のリリースでの Solaris 暗号化フレームワークに対する FIPS 140-2 の検証を完了しました。Oracle Solaris11.2 は、この検証された基盤の上に構築されており、パフォーマンス、機能、および信頼性に対応するソフトウェアの機能強化を含んでいます。これらの機能強化を利用するために、可能な場合は常に、FIPS 140-2 モードで Oracle Solaris11.2 を構成するようにしてください。

詳細は、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』を参照してください。この記事では、次のトピックを扱っています。

- Oracle Solaris での FIPS 140-2 レベル 1 暗号化の概要
- FIPS 140 プロバイダの有効化
- FIPS 140 コンシューマの有効化
- FIPS 140 モードでの 2 つのアプリケーションの有効化の例
- FIPS 140 承認アルゴリズムおよび証明書リファレンス

次の追加情報を参照できます。

- 16 ページの「FIPS 140 対応の OpenSSL 実装に切り替える方法」
- 48 ページの「FIPS 140 が有効になったブート環境の作成 」

Oracle Solaris での OpenSSL のサポート

Oracle Solaris では、OpenSSL の 2 つの実装をサポートしています。

- FIPS 140 対応の OpenSSL
- FIPS 140 非対応の OpenSSL

両方の実装はアップグレードされており、OpenSSL プロジェクトの最新の OpenSSL バージョン (OpenSSL 1.0.1) と互換性があります。このバージョンライブラリに関しては、両方とも API/ABI 互換です。

実装は両方とも OS 内に存在し、一度に 1 つのみの実装をアクティブにできます。どの OpenSSL 実装がシステムでアクティブかを判断するには、`pkg mediator openssl` コマンドを使用します。

▼ FIPS 140 対応の OpenSSL 実装に切り替える方法

デフォルトでは、FIPS 140 非対応の OpenSSL 実装が Oracle Solaris でアクティブになります。ただし、システムのセキュリティーを選択して、必要な実装を選択できます。

1. **管理者になります。**

2. **両方の実装がシステムにあることを確認します。**

```
$ pkg mediator -a openssl
```

注意 - 切り替える OpenSSL 実装がシステムに存在する必要があります。そうでない場合、システムに存在しない実装に切り替えると、システムを使用できなくなる可能性があります。

3. **異なる OpenSSL 実装に切り替えます。**

```
# pkg set-mediator [--be-name name] -I implementation openssl
```

ここで、*implementation* は `default` または `fips-140` のどちらかであり、*name* は現在のブート環境の新しいクローンの名前です。このクローンでは、指定された実装がアクティブになります。

注記 - *--be-name* が指定された場合、このコマンドは、現在のブート環境のバックアップを作成します。リブートすると、システムは、この新しいクローニングされたブート環境を新しい実装で実行します。

`pkg set-mediator` コマンドの詳細は、『Oracle Solaris 11.2 ソフトウェアの追加と更新 』の「優先アプリケーションの変更」を参照してください。

4. **システムをリブートします。**

5. **(オプション) 切り替えが成功し、優先する OpenSSL 実装がアクティブになっていることを確認します。**

```
# pkg mediator openssl
```

例 1-1 FIPS 140 対応の OpenSSL 実装への切り替え

この例では、システムの OpenSSL 実装を FIPS 140 対応になるように変更します。

```
# pkg mediator -a openssl
MEDIATOR    VER. SRC.    VERSION IMPL.    SRC. IMPLEMENTATION
openssl        vendor           vendor              default
openssl        system          system              fips-140
```

```
# pkg set-mediator --be-name BE2 -I fips-140 openssl
# reboot

# pkg mediator openssl
MEDIATOR   VER. SRC.   VERSION IMPL.   SRC. IMPLEMENTATION
openssl      vendor           vendor              default
```

♦ ♦ ♦ 第 2 章

SPARC T シリーズシステムおよび暗号化フレームワークについて

この章では、SPARC T シリーズサーバーの暗号化フレームワークと、暗号化機能のパフォーマンスを強化する Oracle Solaris 11 の最適化について説明します。

暗号化フレームワークと SPARC T シリーズサーバー

暗号化フレームワークは、SPARC T シリーズシステムに暗号化メカニズムを提供し、これらのサーバーのために一部のメカニズムを最適化します。保存されたデータと移動中のデータのために、AES-CBC、AES-CFB128、ARCFOUR の 3 つの暗号化メカニズムが最適化されます。DES 暗号化メカニズムは OpenSSL 用に最適化されており、任意精度演算 (bignum) を最適化することによって、RSA と DSA も最適化されます。その他の最適化には、ハンドシェークや移動中のデータのための小さなパケットのパフォーマンスが含まれます。

このリリースでは、次の暗号化メカニズムが使用できます。

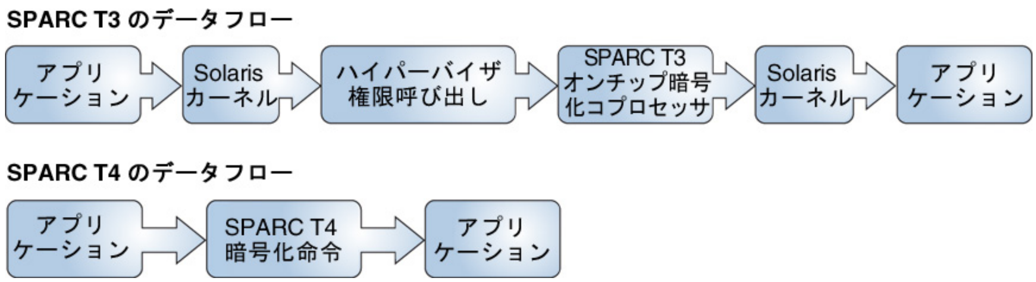
- AES-XTS – 保存されたデータに使用
- SHA-224 – SHA2 メカニズム
- AES-XCBC-MAC – IPsec に使用

SPARC T-4 システムの暗号化の最適化

SPARC T4 マイクロプロセッサから、暗号化機能を実行するための新しい命令がハードウェアで直接使用できるようになりました。命令には特権が不要です。したがって、プログラムはカーネル環境、root 権限、またはその他の特別な設定なしで命令を使用できます。暗号化は、低レベルの多数の命令を使用する代わりに、ハードウェア上で直接実行されます。したがって、以前の SPARC プロセッサでは、暗号化用に個別の処理ユニットを備えていましたが、このようなシステムと比べて、暗号化操作が高速になりました。

次の比較では、暗号化を最適化した SPARC T-3 システムと SPARC T-4 システム間のデータフローの相違点を示します。

図 2-1 SPARC T システム間のデータフローの比較



次の表は、特定の Oracle Solaris リリースと組み合わせた場合の SPARC T マイクロプロセッサユニットの暗号化機能の詳細な比較を示しています。

表 2-1 SPARC T シリーズサーバーの暗号化のパフォーマンス

機能/ソフトウェアコンシューマ	T-3 および以前のシステム	Oracle Solaris 10 を実行する T-4 システム	Oracle Solaris 11 を実行する T-4 システム
SSH	Solaris 10 5/09 以降では自動的に有効化。 `/etc/ssh/sshd_config` の `UseOpenSSLEngine` 句で無効化/有効化。	パッチ 147707-01 が必要。 `/etc/ssh/sshd_config` の `UseOpenSSLEngine` 句で無効化/有効化。	自動的に有効化。 `/etc/ssh/sshd_config` の `UseOpenSSLEngine` 句で無効化/有効化。
Java/JCE	自動的に有効化。 `$JAVA_HOME/jre/lib/security/java.security` で構成。	自動的に有効化。 `$JAVA_HOME/jre/lib/security/java.security` で構成。	自動的に有効化。 `$JAVA_HOME/jre/lib/security/java.security` で構成。
ZFS 暗号化	使用不可。	使用不可。	データセットが暗号化されている場合は、HW 暗号化が自動的に有効化。
IPsec	自動的に有効化。	自動的に有効化。	自動的に有効化。
OpenSSL	`-engine pkcs11` を使用	パッチ 147707-01 が必要 `-engine pkcs11` を使用	T4 最適化を自動的に使用。

機能/ソフトウェアコンシューマ	T-3 および以前のシステム	Oracle Solaris 10 を実行する T-4 システム	Oracle Solaris 11 を実行する T-4 システム
			(オプションで `-engine pkcs11` を使用) この時点では、RSA/DSA に `pkcs11` を推奨。
KSSL (カーネル SSL プロキシ)	自動的に有効化。	自動的に有効化。	自動的に有効化。
Oracle TDE	サポート対象外。	保留中のパッチ。	Oracle DB 11.2.0.3 および ASO で自動的に有効化。
Apache SSL	`SSLCryptoDevice pkcs11` で構成	`SSLCryptoDevice pkcs11` で構成	`SSLCryptoDevice pkcs11` で構成
論理ドメイン	暗号化ユニットをドメインに割り当て。	構成不要で機能を常に使用可能。	構成不要で機能を常に使用可能。

T4 暗号化命令には次のものが含まれます。

- `aes_kexpand0`、`aes_kexpand1`、`aes_kexpand2`

  これらの命令は、*鍵拡張*を実行します。128 ビット、192 ビット、または 256 ビットのユーザー指定の鍵を、暗号化および復号化中に内部で使用される鍵スケジュールに展開します。`aes_kexpand2` 命令は AES-256 専用です。それ以外の 2 つの `aes_kexpand` 命令は、AES-128、AES-192、AES-256 の 3 つの鍵の長さすべてに使用されます。
- `aes_eround01`、`aes_eround23`、`aes_eround01_l`, `aes_eround_23_l`

  これらの命令は、AES 暗号化*ラウンド*または変換に使用されます。FIPS 197 の AES 標準に従って、使用するラウンドの数 (10、12、14 など) は AES 鍵の長さに応じて変化しますが、この理由は、大きい鍵を使用すると、より堅牢な暗号化には、多くの計算が必要になる可能性があるためです。
- `aes_dround01`、`aes_dround23`、`aes_dround01_l`、`aes_dround_23_l`

  これらの命令は、暗号化と同様の方法で、AES 復号化ラウンドに使用されます。
- DES/DES-3、Kasumi、Camellia、Montgomery 多重/平方根 (RSA Bignum 用)、および CRC32c チェックサム用の命令
- MD5、SHA1、および SHA2 ダイジェスト命令

SPARC T4 ハードウェア暗号化命令は、Oracle Solaris 11 を実行する SPARC T4 システムで使用可能で、システムの T4 マイクロプロセッサの組み込み t4 エンジンにより自動的に使用されます。Oracle Solaris 11.2 以降は、これらの命令が OpenSSL アップストリームコー

ドに埋め込まれました。したがって、このリリースでは、OpenSSL 1.0.1e がパッチとともに提供され、これらの命令を使用できます。

T4 命令の詳細は、次の記事を参照してください。

- 「SPARC T4 OpenSSL Engine」(https://blogs.oracle.com/DanX/entry/sparc_t4_openssl_engine)
- 「How to tell if SPARC T4 crypto is being used?」(https://blogs.oracle.com/DanX/entry/how_to_tell_if_sparc)
- 「Exciting Crypto Advances with the T4 processor and Oracle Solaris 11」(http://bubbva.blogspot.com/2011/11/exciting-crypto-advances-with-t4.html)
- 「SPARC T4 Digest and Crypto Optimizations in Solaris 11.1」(https://blogs.oracle.com/DanX/entry/sparc_t4_digest_and_crypto)

システムが SPARC T4 最適化をサポートしているかどうかの確認

T4 最適化が使用されているかどうかを確認するには、isainfo コマンドを使用します。sparcv9 および aes が出力に含まれている場合は、システムが最適化を使用していることを示します。

```
$ isainfo -v
64-bit sparcv9 applications
        crc32c cbcond pause mont mpmul sha512 sha256 sha1 md5 camellia kasumi
        des aes ima hpc vis3 fmaf asi_blk_init vis2 vis popc
```

システムの OpenSSL バージョンの確認

システムで実行されている OpenSSL のバージョンを確認するには、openssl version と入力します。出力は、次のようになります。

```
OpenSSL 1.0.0j 10 May 2012
```

SPARC T4 最適化を備えた OpenSSL がシステムにあることの確認

SPARC T4 最適化を備えた OpenSSL をシステムがサポートしているかどうかを確認するには、次のように libcrypto.so ライブラリを確認します。

```
# nm /lib/libcrypto.so.1.0.0 | grep des_t4
[5239]  |     504192|       300|FUNC |GLOB |3    |12     |des_t4_cbc_decrypt
[5653]  |     503872|       300|FUNC |GLOB |3    |12     |des_t4_cbc_encrypt
[4384]  |     505024|       508|FUNC |GLOB |3    |12     |des_t4_ede3_cbc_decrypt
```

```
[2963]  |    504512|       508|FUNC |GLOB |3    |12     |des_t4_ede3_cbc_encrypt
[4111]  |    503712|       156|FUNC |GLOB |3    |12     |des_t4_key_expand
```

コマンドで出力が生成されない場合、システムは OpenSSL の SPARC T4 最適化をサポートしていません。

◆ ◆ ◆ 第 3 章

暗号化フレームワーク

この章では、暗号化フレームワークの使用方法について説明します。この章の内容は次のとおりです。

- 25 ページの「暗号化フレームワークによるファイルの保護」
- 38 ページの「暗号化フレームワークの管理」

暗号化フレームワークによるファイルの保護

このセクションでは、対称鍵を生成する方法、ファイルの整合性のためにチェックサムを作成する方法、およびファイルが傍受されるのを防ぐ方法について説明します。このセクションのコマンドは、通常のユーザーが実行することができます。開発者は、これらのコマンドを使用するスクリプトを作成することができます。

FIPS 140 モードでシステムを設定するには、FIPS で検証されたアルゴリズム、モード、および鍵の長さを使用する必要があります。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithm Lists and Certificate References for Oracle Solaris Systems」を参照してください。

暗号化フレームワークは、ファイルの保護に役立ちます。次のタスクマップでは、使用可能なアルゴリズムを一覧表示する手順、および暗号化によってファイルを保護する手順を示します。

表 3-1 暗号化フレームワークによるファイルの保護のタスクマップ

タスク	説明	参照先
対称鍵を生成する。	ユーザーが指定した長さの鍵を生成します。任意で、ファイル、PKCS #11 キーストア、または NSS キーストアに鍵を格納します。 FIPS 140 承認のモードの場合は、FIPS に対して検証された鍵のタイプ、モード、およ	26 ページの「`pktool` コマンドを使用して対称鍵を生成する方法」

タスク	説明	参照先
	び鍵の長さを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。	
ファイルの整合性を保証するチェックサムを提供します。	受信者のファイルのコピーが送信されたファイルと同一のものであることを検証します。	32 ページの「ファイルのダイジェストを計算する方法」
メッセージ認証コード (MAC) でファイルを保護します。	自分がメッセージの送信者であることを受信者に証明します。	33 ページの「ファイルの MAC を計算する方法」
ファイルを暗号化したあと、暗号化されたファイルを復号化します。	ファイルを暗号化することによりファイルの内容を保護します。ファイルを復号化するための暗号化パラメータを指定します。	36 ページの「ファイルを暗号化および復号化する方法」

▼ pktool コマンドを使用して対称鍵を生成する方法

アプリケーションによっては、通信の暗号化および復号化に対称鍵が必要です。この手順では、対称鍵を作成して格納します。

乱数発生関数がある場合、この関数を使用して鍵の乱数を作成できます。この手順は乱数発生関数を使用しません。

1. **(オプション) キーストアを使用する場合は、作成します。**
   - **PKCS #11 キーストアを作成して初期化する方法については、68 ページの「`pktool setpin` コマンドを使ってパスフレーズを生成する方法」を参照してください。**
   - **NSS データベースを作成して初期化するには、例4-5「キーストアをパスフレーズで保護する」 のサンプルコマンドを参照してください。**
2. **対称鍵として使用する乱数を生成します。**

   次のいずれかを実行します。

   - **鍵を生成してファイルに格納します。**

鍵をファイルに格納する利点は、このファイルから鍵を抽出して、/etc/inet/secret/ipseckeys ファイルや IPsec など、アプリケーションの鍵ファイルで使用できることです。使用法の文は引数を示しています。

```
% pktool genkey keystore=file
...genkey keystore=file
outkey=key-fn
[ keytype=aes|arcfour|des|3des|generic ]
[ keylen=key-size (AES, ARCFOUR or GENERIC only)]
[ print=y|n ]
```

outkey=*key-fn*

鍵が格納されているファイル名。

keytype=*specific-symmetric-algorithm*

任意の長さの対称鍵の場合、値は generic になります。特定のアルゴリズムとして、aes、arcfour、des、または 3des を指定します。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵のタイプを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

keylen=*size-in-bits*

鍵のビット長。8 で割り切れる数にする必要があります。des または 3des には指定*しない*でください。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵の長さを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

print=n

鍵を端末ウィンドウに印刷します。デフォルトでは、print の値は n です。

- **鍵を生成して PKCS #11 キーストアに格納します。**

PKCS #11 キーストアの利点は、ラベルに基づいて鍵を取得できることです。この方法は、ファイルを暗号化および復号化する鍵の場合に便利です。この方法を使用するには、ステップ 1 を完了する必要があります。使用法の文は引数を示しています。キーストア引数の前後の角括弧は、キーストア引数が指定されていないときは鍵が PKCS #11 キーストア内に格納されることを示しています。

```
$ pktool genkey keystore=pkcs11
...genkey [ keystore=pkcs11 ]
label=key-label
[ keytype=aes|arcfour|des|3des|generic ]
```

```
[ keylen=key-size (AES, ARCFOUR or GENERIC only)]
[ token=token[:manuf[:serial]]]
[ sensitive=y|n ]
[ extractable=y|n ]
[ print=y|n ]
```

label=*key-label*

鍵についてユーザーが指定したラベル。ラベルに基づいてキーストアから鍵を取得できます。

keytype=*specific-symmetric-algorithm*

任意の長さの対称鍵の場合、値は generic になります。特定のアルゴリズムとして、aes、arcfour、des、または 3des を指定します。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵のタイプを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

keylen=*size-in-bits*

鍵のビット長。8 で割り切れる数にする必要があります。des または 3des には指定*しない*でください。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵の長さを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

token=*token*

トークン名。デフォルトでは、トークンは Sun Software PKCS#11 softtoken です。

sensitive=n

鍵の重要度を指定します。値が y の場合、鍵は print=y 引数を使用して出力することはできません。デフォルトでは、sensitive の値は n です。

extractable=y

鍵がキーストアから抽出できることを指定します。鍵が抽出されないようにするには、n を指定します。

print=n

鍵を端末ウィンドウに印刷します。デフォルトでは、print の値は n です。

- **鍵を生成して NSS キーストアに格納します。**

この方法を使用するには、ステップ 1 を完了する必要があります。使用法の文は引数を示しています。

```
$ pktool genkey keystore=nss
...genkey keystore=nss
label=key-label
[ keytype=aes|arcfour|des|3des|generic ]
[ keylen=key-size (AES, ARCFOUR or GENERIC only)]
[ token=token[:manuf[:serial]]]
[ dir=directory-path ]
[ prefix=DBprefix ]
```

`label=`*key-label*

鍵についてユーザーが指定したラベル。ラベルに基づいてキーストアから鍵を取得できます。

`keytype=`*specific-symmetric-algorithm*

任意の長さの対称鍵の場合、値は `generic` になります。特定のアルゴリズムとして、`aes`、`arcfour`、`des`、または `3des` を指定します。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵のタイプを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

`keylen=`*size-in-bits*

鍵のビット長。8 で割り切れる数にする必要があります。`des` または `3des` には指定*しない*でください。

FIPS 140 承認アルゴリズムの場合は、FIPS に対して検証された鍵の長さを選択します。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」を参照してください。

`token=`*token*

トークン名。デフォルトでは、トークンは NSS 内部トークンです。

`dir=`*directory*

NSS データベースへのディレクトリパス。デフォルトでは、*directory* は現在のディレクトリです。

`prefix=`*directory*

NSS データベースの接頭辞。デフォルトは接頭辞なしです。

3. **(オプション) 鍵が存在することを確認します。**

鍵を格納した場所に応じて、次のコマンドのいずれかを使用します。

- ***key-fn* ファイル内の鍵を確認します。**

```
% pktool list keystore=file objtype=key [infile=key-fn]
```

```
Found n keys.
Key #1 - keytype:location (keylen)
```

- **PKCS #11 または NSS キーストア内の鍵を確認します。**

For PKCS #11, use the following command:

```
$ pktool list keystore=pkcs11 objtype=key
Enter PIN for keystore:
Found n keys.
Key #1 - keytype:location (keylen)
```

次に、このコマンドの keystore=pkcs11 を keystore=nss に置き換えます。

例 3-1 pktool コマンドを使用して対称鍵を作成する

次の例では、ユーザーは最初に PKCS #11 キーストアを作成し、次にアプリケーション用の大きな対称鍵を生成します。最後に、ユーザーはその鍵がキーストアに格納されていることを確認します。

PKCS #11 キーストアの初期パスワードは changeme です。NSS キーストアの最初のパスワードは空のパスワードです。

```
# pktool setpin
Create new passphrase:     Type password
Re-enter new passphrase:     Retype password
Passphrase changed.
% pktool genkey label=specialappkey keytype=generic keylen=1024
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password

% pktool list objtype=key
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password
No.      Key Type      Key Len.      Key Label
----------------------------------------------------
Symmetric keys:
1        Symmetric     1024          specialappkey
```

例 3-2 pktool コマンドを使用して FIPS 承認の AES 鍵を作成する

次の例では、AES アルゴリズムの秘密鍵が FIPS 承認のアルゴリズムと鍵の長さを使用して作成されます。鍵は、あとで復号化するためにローカルファイルに格納されます。コマンドは、400 のアクセス権でファイルを保護します。鍵が作成されると、print=y オプションにより、生成された鍵が端末ウィンドウに表示されます。

鍵ファイルを所有するユーザーは、od コマンドを使用して鍵を取得します。

```
% pktool genkey keystore=file outkey=256bit.file1 keytype=aes keylen=256 print=y
```

```
Key Value ="aaa2df1d10f02eaee2595d48964847757a6a49cf86c4339cd5205c24ac8c8873"
% od -x 256bit.file1

0000000 aaa2 df1d 10f0 2eae e259 5d48 9648 4775
0000020 7a6a 49cf 86c4 339c d520 5c24 ac8c 8873
0000040
```

例 3-3 IPsec セキュリティーアソシエーション用の対称鍵を作成する

次の例では、管理者は IPsec SA 用の鍵データを手動で作成し、それらをファイルに格納します。次に、管理者はそれらの鍵を /etc/inet/secret/ipseckeys ファイルにコピーし、元のファイルを破棄します。

最初に、管理者は IPsec ポリシーで要求される鍵を作成して表示します。

```
# pktool genkey keystore=file outkey=ipencrin1 keytype=generic keylen=192 print=y
Key Value ="294979e512cb8e79370dabecadc3fcbb849e78d2d6bd2049"
# pktool genkey keystore=file outkey=ipencrout1 keytype=generic keylen=192 print=y
Key Value ="9678f80e33406c86e3d1686e50406bd0434819c20d09d204"
# pktool genkey keystore=file outkey=ipspi1 keytype=generic keylen=32 print=y
Key Value ="acbeaa20"
# pktool genkey keystore=file outkey=ipspi2 keytype=generic keylen=32 print=y
Key Value ="19174215"
# pktool genkey keystore=file outkey=ipsha21 keytype=generic keylen=256 print=y
Key Value ="659c20f2d6c3f9570bcee93e96d95e2263aca4eeb3369f72c5c786af4177fe9e"
# pktool genkey keystore=file outkey=ipsha22 keytype=generic keylen=256 print=y
Key Value ="b041975a0e1fce0503665c3966684d731fa3dbb12fcf87b0a837b2da5d82c810"
```

そのあと、管理者は次の /etc/inet/secret/ipseckeys ファイルを作成します。

```
##   SPI values require a leading 0x.
##   Backslashes indicate command continuation.
##
## for outbound packets on this system
add esp spi 0xacbeaa20 \
src 192.168.1.1 dst 192.168.2.1 \
encr_alg aes auth_alg sha256  \
encrkey  294979e512cb8e79370dabecadc3fcbb849e78d2d6bd2049 \
authkey  659c20f2d6c3f9570bcee93e96d95e2263aca4eeb3369f72c5c786af4177fe9e
##
## for inbound packets
add esp spi 0x19174215 \
src 192.168.2.1 dst 192.168.1.1 \
encr_alg aes auth_alg sha256  \
encrkey 9678f80e33406c86e3d1686e50406bd0434819c20d09d204 \
authkey b041975a0e1fce0503665c3966684d731fa3dbb12fcf87b0a837b2da5d82c810
```

ipseckeys ファイルの構文が有効であることを確認したあと、管理者は元の鍵ファイルを破棄します。

```
# ipseckey -c /etc/inet/secret/ipseckeys
```

```
# rm ipencrin1 ipencrout1 ipspi1 ipspi2 ipsha21 ipsha22
```

管理者は、ssh コマンドや別のセキュアなメカニズムを使用して、ipseckeys ファイルを通信先のシステムにコピーします。通信先のシステムでは、それらの保護は取り消されます。ipseckeys ファイルの最初のエントリによってインバウンドパケットが保護され、2 番目のエントリによってアウトバウンドパケットが保護されます。通信先のシステムでは鍵は生成されません。

次の手順 鍵を使用してファイルのメッセージ認証コード (MAC) の作成を続行するには、33 ページの「ファイルの MAC を計算する方法」を参照してください。

▼ ファイルのダイジェストを計算する方法

ファイルのダイジェストを作成すると、ダイジェストの出力を比較することによって、ファイルが改ざんされていないことを確認することができます。ダイジェストによって元のファイルが変更されることはありません。

1. **使用可能なダイジェストアルゴリズムを一覧表示します。**

```
% digest -l
md5
sha1
sha224
sha256
sha384
sha512
```

注記 - 可能な場合は常に、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」にあるリストに従って FIPS 承認アルゴリズムを選択してください。

---

2. **ファイルのダイジェストを計算し、ダイジェストのリストを保存します。**

digest コマンドでアルゴリズムを指定します。

% **digest -v -a** *algorithm input-file* **>** *digest-listing*

-v　次の形式で出力を表示します。

algorithm (*input-file*) = *digest*

-a *algorithm*　ファイルのダイジェストの計算に使用するアルゴリズム。ステップ 1 の出力のようにアルゴリズムを入力します。

注記 - 可能な場合は常に、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」に記載されている FIPS 承認アルゴリズムを選択してください。

---

input-file `digest` コマンドの入力ファイル。

digest-listing `digest` コマンドの出力ファイル。

例 3-4 SHA1 メカニズムでダイジェストを計算する

次の例では、`digest` コマンドが SHA1 メカニズムを使用してディレクトリ一覧を提供します。結果はファイルに格納されます。

```
% digest -v -a sha1 docs/* > $HOME/digest.docs.legal.05.07
% more ~/digest.docs.legal.05.07
sha1 (docs/legal1) = 1df50e8ad219e34f0b911e097b7b588e31f9b435
sha1 (docs/legal2) = 68efa5a636291bde8f33e046eb33508c94842c38
sha1 (docs/legal3) = 085d991238d61bd0cfa2946c183be8e32cccf6c9
sha1 (docs/legal4) = f3085eae7e2c8d008816564fdf28027d10e1d983
```

▼ ファイルの MAC を計算する方法

メッセージ認証コード (MAC) は、ファイルのダイジェストを計算し、秘密鍵を使用してさらにダイジェストを保護します。MAC によって元のファイルが変更されることはありません。

1. **使用可能なメカニズムを一覧表示します。**

```
% mac -l
Algorithm       Keysize:  Min   Max
-----------------------------------
des_mac                    64    64
sha1_hmac                   8   512
md5_hmac                    8   512
sha224_hmac                 8   512
sha256_hmac                 8   512
sha384_hmac                 8  1024
sha512_hmac                 8  1024
```

注記 - サポートされている各アルゴリズムは、もっともよく使用され、もっとも制限が少ない特定のアルゴリズムタイプの別名です。上の出力は、使用可能なアルゴリズム名と各アルゴリズムのキーサイズを示しています。可能な場合は常に、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」に記載されている、FIPS 承認の鍵の長さを備えた FIPS 承認アルゴリズムに一致するサポートされたアルゴリズムを使用してください。

---

2. **該当する長さの対称鍵を生成します。**

   鍵が生成されるパスフレーズを指定したり、鍵を指定したりできます。

   - パスフレーズを指定する場合、指定したパスフレーズを格納するか覚えておく必要があります。パスフレーズをオンラインで格納する場合、そのパスフレーズファイルは自分だけが読み取ることができるようにします。
   - 鍵を指定する場合、それはメカニズムの現在のサイズである必要があります。`pktool` コマンドを使用できます。手順およびいくつかの例については、26 ページの「`pktool` コマンドを使用して対称鍵を生成する方法」を参照してください。

3. **ファイルの MAC を作成します。**

   `mac` コマンドで、鍵を指定して対称鍵アルゴリズムを使用します。

   `% mac [-v] -a` *algorithm* `[-k` *keyfile* `| -K` *key-label* `[-T` *token*`]]` *input-file*

-v	次の形式で出力を表示します。 *algorithm* (*input-file*) = *mac*
-a *algorithm*	MAC の計算に使用するアルゴリズム。`mac -l` コマンドの出力のようにアルゴリズムを入力します。
-k *keyfile*	アルゴリズムで指定された長さの鍵を含むファイル。
-K *key-label*	PKCS #11 キーストア内の鍵のラベル。
-T *token*	トークン名。デフォルトでは、トークンは `Sun Software PKCS#11 softtoken` です。-K *key-label* オプションが使用された場合にのみ使用されます。
input-file	MAC の入力ファイル。

例 3-5 SHA1_HMAC およびパスフレーズで MAC を計算する

次の例では、電子メールの添付ファイルを、SHA1_HMAC メカニズムとパスフレーズから生成される鍵で認証します。MAC の一覧はファイルに保存されます。パスフレーズをファイルに格納する場合、そのファイルはユーザー以外は読み取ることができないようにします。

```
% mac -v -a sha1_hmac email.attach
Enter passphrase:      Type passphrase
sha1_hmac (email.attach) = 2b31536d3b3c0c6b25d653418db8e765e17fe07b
% echo "sha1_hmac (email.attach) = 2b31536d3b3c0c6b25d653418db8e765e17fe07b" \
>> ~/sha1hmac.daily.05.12
```

例 3-6 SHA1_HMAC および鍵ファイルで MAC を計算する

次の例では、ディレクトリの一覧を、SHA1_HMAC メカニズムと秘密鍵で認証します。結果はファイルに格納されます。

```
% mac -v -a sha1_hmac \
-k $HOME/keyf/05.07.mack64 docs/* > $HOME/mac.docs.legal.05.07
% more ~/mac.docs.legal.05.07
sha1_hmac (docs/legal1) = 9b31536d3b3c0c6b25d653418db8e765e17fe07a
sha1_hmac (docs/legal2) = 865af61a3002f8a457462a428cdb1a88c1b51ff5
sha1_hmac (docs/legal3) = 076c944cb2528536c9aebd3b9fbe367e07b61dc7
sha1_hmac (docs/legal4) = 7aede27602ef6e4454748cbd3821e0152e45beb4
```

例 3-7 SHA1_HMAC および鍵ラベルで MAC を計算する

次の例では、ディレクトリの一覧を、SHA1_HMAC メカニズムと秘密鍵で認証します。結果は、ユーザーの PKCS #11 キーストアに格納されます。ユーザーは、最初に `pktool setpin` コマンドを使用してキーストアとそのキーストアのパスワードを作成しました。

```
% mac -a sha1_hmac -K legaldocs0507 docs/*
Enter pin for Sun Software PKCS#11 softtoken:     Type password
```

キーストアから MAC を取り出すために、ユーザーは冗長オプションを使用し、鍵ラベルと認証されたディレクトリの名前を指定します。

```
% mac -v -a sha1_hmac -K legaldocs0507  docs/*
Enter pin for Sun Software PKCS#11 softtoken:     Type password
sha1_hmac (docs/legal1) = 9b31536d3b3c0c6b25d653418db8e765e17fe07a
sha1_hmac (docs/legal2) = 865af61a3002f8a457462a428cdb1a88c1b51ff5
sha1_hmac (docs/legal3) = 076c944cb2528536c9aebd3b9fbe367e07b61dc7
sha1_hmac (docs/legal4) = 7aede27602ef6e4454748cbd3821e0152e45beb4
```

▼ ファイルを暗号化および復号化する方法

ファイルを暗号化しても、元のファイルが削除されたり変更されたりすることはありません。出力ファイルが暗号化されます。

encrypt コマンドに関する一般的なエラーを解決するには、例のあとのセクションを参照してください。

注記 - ファイルを暗号化および復号化するとき、可能な場合は常に、承認された鍵の長さで FIPS 承認アルゴリズムを使用してみてください。『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」にあるリストを参照してください。使用可能なアルゴリズムとその鍵の長さを表示するには、encrypt -l コマンドを実行します。

---

1. **該当する長さの対称鍵を作成します。**

   鍵が生成されるパスフレーズを指定したり、鍵を指定したりできます。

   - パスフレーズを指定する場合、指定したパスフレーズを格納するか覚えておく必要があります。パスフレーズをオンラインで格納する場合、そのパスフレーズファイルは自分だけが読み取ることができるようにします。
   - 鍵を指定する場合、それはメカニズムの現在のサイズである必要があります。pktool コマンドを使用できます。手順およびいくつかの例については、26 ページの「pktool コマンドを使用して対称鍵を生成する方法」を参照してください。

2. **ファイルを暗号化します。**

   encrypt コマンドで、鍵を指定して対称鍵アルゴリズムを使用します。

```
% encrypt -a algorithm [-v] \
[-k keyfile | -K key-label [-T token]] [-i input-file] [-o output-file]
```

-a *algorithm*　ファイルを暗号化するために使用するアルゴリズム。encrypt -l コマンドの出力のようにアルゴリズムを入力します。可能な場合は常に、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」にあるリストに従って FIPS 承認アルゴリズムを選択してください。

-k *keyfile*　アルゴリズムで指定された長さの鍵を含むファイル。アルゴリズムごとの鍵の長さが、ビット単位で encrypt -l コマンドの出力に一覧表示されます。

-K *key-label* PKCS #11 キーストア内の鍵のラベル。

-T *token* トークン名。デフォルトでは、トークンは Sun Software PKCS#11 softtoken です。-K *key-label* オプションが使用された場合にのみ使用されます。

-i *input-file* 暗号化する入力ファイル。このファイルは、コマンドによって変更されることはありません。

-o *output-file* 入力ファイルと同じ暗号化形式の出力ファイル。

例 3-8 ファイルを暗号化するための AES 鍵を作成する

次の例では、ユーザーは暗号化と復号化用に AES 鍵を作成して、既存の PKCS #11 キーストアに格納します。ユーザーは、その鍵が存在し、使用できることを確認することはできますが、その鍵自体を表示することはできません。

```
% pktool genkey label=MyAESkeynumber1 keytype=aes keylen=256
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password

% pktool list objtype=key
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password
No.      Key Type      Key Len.      Key Label
----------------------------------------------------
Symmetric keys:
1        AES           256           MyAESkeynumber1
```

その鍵を使用してファイルを暗号化するには、ユーザーはそのラベルで鍵を取り出します。

```
% encrypt -a aes -K MyAESkeynumber1 -i encryptthisfile -o encryptedthisfile
```

encryptedthisfile ファイルを復号化するには、ユーザーはその鍵をラベルで取り出します。

```
% decrypt -a aes -K MyAESkeynumber1 -i encryptedthisfile -o sameasencryptthisfile
```

例 3-9 AES およびパスフレーズで暗号化および復号化する

次の例では、ファイルを AES アルゴリズムで暗号化します。鍵はパスフレーズから生成されます。パスフレーズをファイルに格納する場合、そのファイルはユーザー以外は読み取ることができないようにします。

```
% encrypt -a aes -i ticket.to.ride -o ~/enc/e.ticket.to.ride
Enter passphrase:        Type passphrase
Re-enter passphrase:     Type passphrase again
```

入力ファイル ticket.to.ride は、元の形式で存在します。

出力ファイルを復号化する場合、ユーザーはファイルを暗号化したときと同じパスフレーズと暗号化メカニズムを使用します。

```
% decrypt -a aes -i ~/enc/e.ticket.to.ride -o ~/d.ticket.to.ride
Enter passphrase:      Type passphrase
```

例 3-10 AES および鍵ファイルで暗号化および復号化する

次の例では、ファイルを AES アルゴリズムで暗号化します。AES メカニズムでは、128 ビット (16 バイト) の鍵が使用されます。

```
% encrypt -a aes -k ~/keyf/05.07.aes16 \
-i ticket.to.ride -o ~/enc/e.ticket.to.ride
```

入力ファイル `ticket.to.ride` は、元の形式で存在します。

出力ファイルを復号化するとき、ユーザーはファイルを暗号化したときと同じ鍵と暗号化メカニズムを使用します。

```
% decrypt -a aes -k ~/keyf/05.07.aes16  \
-i ~/enc/e.ticket.to.ride -o ~/d.ticket.to.ride
```

注意事項 次のメッセージは、`encrypt` コマンドに指定した鍵が、使用しているアルゴリズムで許可されないことを示しています。

- `encrypt: unable to create key for crypto operation: CKR_ATTRIBUTE_VALUE_INVALID`
- `encrypt: failed to initialize crypto operation: CKR_KEY_SIZE_RANGE`

アルゴリズムの条件を満たしていない鍵を渡した場合は、次のいずれかの方法を使用して、条件を満たす鍵を指定する必要があります。

- パスフレーズを使用します。それによって、条件を満たす鍵が暗号化フレームワークで指定されます。
- アルゴリズムが使用できる鍵サイズを渡します。たとえば、DES アルゴリズムでは 64 ビットの鍵が必要です。3DES アルゴリズムでは 192 ビットの鍵が必要です。

暗号化フレームワークの管理

このセクションでは、暗号化フレームワークでのソフトウェアプロバイダとハードウェアプロバイダの管理方法について説明します。必要に応じて、ソフトウェアプロバイダおよびハードウェアプロ

バイダの使用を解除することができます。たとえば、ソフトウェアプロバイダのアルゴリズムの実装を無効にすることができます。その後、別のソフトウェアプロバイダのアルゴリズムがシステムで使用されるようにすることができます。

注記 - 暗号化フレームワークの管理の重要なコンポーネントは、暗号化モジュールのための米国政府のコンピュータセキュリティー標準である FIPS 140 に関連したポリシーの計画および実装です。

FIPS 140-2 で検証された暗号化のみを使用するという厳格な要件がある場合は、Oracle Solaris11.1 SRU5.5 リリースまたは Oracle Solaris11.1 SRU3 リリースを実行する必要があります。Oracle は、これらの 2 つの特定のリリースでの Solaris 暗号化フレームワークに対する FIPS 140-2 の検証を完了しました。Oracle Solaris11.2 は、この検証された基盤の上に構築されており、パフォーマンス、機能、および信頼性に対応するソフトウェアの機能強化を含んでいます。これらの機能強化を利用するために、可能な場合は常に、FIPS 140-2 モードで Oracle Solaris11.2 を構成するようにしてください。

『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』を確認し、システムの全体的な FIPS ポリシーを計画してください。

次のタスクマップは、暗号化フレームワークでのソフトウェアプロバイダとハードウェアプロバイダの管理の手順を示しています。

表 3-2 暗号化フレームワークの管理のタスクマップ

タスク	説明	参照先
システムの FIPS ポリシーを計画します。	FIPS 承認のプロバイダおよびコンシューマを有効にするための計画を決定し、その計画を実装します。	『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』
暗号化フレームワークのプロバイダを一覧表示します。	暗号化フレームワークで使用可能なアルゴリズム、ライブラリ、およびハードウェアデバイスを一覧表示します。	40 ページの「使用可能なプロバイダの一覧表示」
FIPS 140 モードを有効にします。	暗号化モジュールのための米国政府の標準に従って暗号化フレームワークを実行します。	48 ページの「FIPS 140 が有効になったブート環境を作成する方法」
ソフトウェアプロバイダを追加します。	PKCS #11 ライブラリまたはカーネルモジュールを暗号化フレームワークに追加します。プロバイダは署名されている必要があります。	46 ページの「ソフトウェアプロバイダを追加する方法」
ユーザーレベルのメカニズムが使用されないようにします.	ソフトウェアメカニズムの使用を解除します。ソフトウェアメカニズムは、再度有効にすることができます。	50 ページの「ユーザーレベルのメカニズムが使用されないようにする方法」

タスク	説明	参照先
カーネルモジュールのメカニズムを一時的に無効にします。	一時的にメカニズムの使用を解除します。通常はテストのために使用します。	52 ページの「カーネルソフトウェアメカニズムが使用されないようにする方法」
ライブラリをアンインストールします。	ユーザーレベルソフトウェアプロバイダの使用を解除します。	例3-17「ユーザーレベルライブラリを永続的に削除する」
カーネルプロバイダをアンインストールします。	カーネルソフトウェアプロバイダの使用を解除します。	例3-19「カーネルソフトウェアプロバイダの使用を一時的に削除する」
ハードウェアプロバイダのメカニズムを無効にします。	ハードウェアアクセラレータ上の選択したメカニズムが使用されないようにします。	55 ページの「ハードウェアプロバイダのメカニズムと機能を無効にする方法」
暗号化サービスを再起動またはリフレッシュします。	暗号化サービスを使用できるようにします。	57 ページの「すべての暗号化サービスをリフレッシュまたは再起動する方法」

使用可能なプロバイダの一覧表示

ハードウェアプロバイダは、自動的に配置されロードされます。詳細は、`driver.conf(4)` のマニュアルページを参照してください。

暗号化フレームワークへの接続が想定されているハードウェアがある場合、そのハードウェアはカーネルの SPI に登録されます。暗号化フレームワークでは、ハードウェアドライバが署名されていることが確認されます。特に、ドライバのオブジェクトファイルが Oracle が発行する証明書付きで署名されていることが確認されます。

たとえば、Sun Crypto Accelerator 6000 ボード (`mca`)、UltraSPARC T1 および T2 プロセッサの暗号化アクセラレータ用 `ncp` ドライバ (`ncp`)、UltraSPARC T2 プロセッサ用 n2cp ドライバ (`n2cp`)、T シリーズシステム用の `/dev/crypto` ドライバは、ハードウェアのメカニズムをフレームワークに接続します。

プロバイダの署名については、12 ページの「暗号化フレームワークのユーザーレベルコマンド」の `elfsign` に関する情報を参照してください。

使用可能なプロバイダを一覧表示するには、取得する情報に応じて、`cryptoadm list` コマンドにさまざまなオプションを付けて使用します。

- システムのすべてのプロバイダを一覧表示する。

  プロバイダリストの内容と書式は、Oracle Solaris のリリースやプラットフォームによって異なります。使用しているシステムでサポートされるプロバイダを表示するには、システムで

`cryptoadm list` コマンドを実行します。通常のユーザーは、ユーザーレベルのメカニズムのみを直接使用できます。

```
% cryptoadm list
User-level providers:        /* for applications */
Provider: /usr/lib/security/$ISA/pkcs11_kernel.so
Provider: /usr/lib/security/$ISA/pkcs11_softtoken.so
Provider: /usr/lib/security/$ISA/pkcs11_tpm.so

Kernel software providers:        /* for IPsec, kssl, Kerberos */
des
aes
arcfour
blowfish
camellia
ecc
sha1
sha2
md4
md5
rsa
swrand
n2rng/0        /* for hardware */
ncp/0
n2cp/0
```

- 暗号化フレームワークのプロバイダとそのメカニズムを一覧表示する。

  利用可能なメカニズムの強度やモード (ECB、CBC など) を表示できます。ただし、一覧表示されたメカニズムのいくつかは使用できない場合があります。使用可能なメカニズムを一覧表示する方法については、次の項目の手順を参照してください。

  次の出力は、表示用に切り詰められています。

```
% cryptoadm list -m [provider=provider]
User-level providers:
=====================

Provider: /usr/lib/security/$ISA/pkcs11_kernel.so
```

```
Mechanisms:
CKM_DSA
CKM_RSA_X_509
CKM_RSA_PKCS
...
CKM_SHA256_HMAC_GENERAL
CKM_SSL3_MD5_MAC

Provider: /usr/lib/security/$ISA/pkcs11_softtoken.so
Mechanisms:
CKM_DES_CBC
CKM_DES_CBC_PAD
CKM_DES_ECB
CKM_DES_KEY_GEN
CKM_DES_MAC_GENERAL
...
CKM_ECDSA_SHA1
CKM_ECDH1_DERIVE

Provider: /usr/lib/security/$ISA/pkcs11_tpm.so
/usr/lib/security/$ISA/pkcs11_tpm.so: no slots presented.

Kernel providers:
==========================
des: CKM_DES_ECB,CKM_DES_CBC,CKM_DES3_ECB,CKM_DES3_CBC
aes: CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR,CKM_AES_CCM, \
     CKM_AES_GCM,CKM_AES_GMAC,
CKM_AES_CFB128,CKM_AES_XTS,CKM_AES_XCBC_MAC
arcfour: CKM_RC4
blowfish: CKM_BLOWFISH_ECB,CKM_BLOWFISH_CBC
ecc: CKM_EC_KEY_PAIR_GEN,CKM_ECDH1_DERIVE,CKM_ECDSA, \
     CKM_ECDSA_SHA1
sha1: CKM_SHA_1,CKM_SHA_1_HMAC,CKM_SHA_1_HMAC_GENERAL
sha2: CKM_SHA224,CKM_SHA224_HMAC,...CKM_SHA512_256_HMAC_GENERAL

md4: CKM_MD4
md5: CKM_MD5,CKM_MD5_HMAC,CKM_MD5_HMAC_GENERAL
```

```
rsa: CKM_RSA_PKCS,CKM_RSA_X_509,CKM_MD5_RSA_PKCS, \
     CKM_SHA1_RSA_PKCS,CKM_SHA224_RSA_PKCS,
CKM_SHA256_RSA_PKCS,CKM_SHA384_RSA_PKCS,CKM_SHA512_RSA_PKCS
swrand: No mechanisms presented.
n2rng/0: No mechanisms presented.
ncp/0: CKM_DSA,CKM_RSA_X_509,CKM_RSA_PKCS,CKM_RSA_PKCS_KEY_PAIR_GEN,
CKM_DH_PKCS_KEY_PAIR_GEN,CKM_DH_PKCS_DERIVE,CKM_EC_KEY_PAIR_GEN,
CKM_ECDH1_DERIVE,CKM_ECDSA
n2cp/0: CKM_DES_CBC,CKM_DES_CBC_PAD,CKM_DES_ECB,CKM_DES3_CBC, \
     ...CKM_SSL3_SHA1_MAC
```

- 使用可能な暗号化メカニズムを一覧表示する。

  使用可能なメカニズムをポリシーで決定します。ポリシーは、管理者によって設定されます。管理者は、特定のプロバイダのメカニズムを無効にすることができます。-p オプションを指定すると、管理者が設定したポリシーによって許可されているメカニズムのリストが表示されます。

```
% cryptoadm list -p [provider=provider]
User-level providers:
=====================
/usr/lib/security/$ISA/pkcs11_kernel.so: \
     all mechanisms are enabled.random is enabled.
/usr/lib/security/$ISA/pkcs11_softtoken.so: \
     all mechanisms are enabled, random is enabled.
/usr/lib/security/$ISA/pkcs11_tpm.so: all mechanisms are enabled.

Kernel providers:
==========================
des: all mechanisms are enabled.
aes: all mechanisms are enabled.
arcfour: all mechanisms are enabled.
blowfish: all mechanisms are enabled.
ecc: all mechanisms are enabled.
sha1: all mechanisms are enabled.
sha2: all mechanisms are enabled.
md4: all mechanisms are enabled.
md5: all mechanisms are enabled.
rsa: all mechanisms are enabled.
```

```
swrand: random is enabled.
n2rng/0: all mechanisms are enabled. random is enabled.
ncp/0: all mechanisms are enabled.
n2cp/0: all mechanisms are enabled.
```

次の例は、cryptoadm list コマンドの追加の使用を示しています。

例 3-11 特定のプロバイダの暗号化情報の一覧表示

cryptoadm *options* コマンドでプロバイダを指定すると、プロバイダに適用可能な情報のみに出力が制限されます。

```
# cryptoadm enable provider=dca/0 random
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_MD5, CKM_MD5_HMAC,...
random is enabled.
```

次の出力は、メカニズムのみが有効であることを示しています。乱数発生関数は無効にしておきます。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_MD5,CKM_MD5_HMAC,....

# cryptoadm enable provider=dca/0 mechanism=all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is disabled.
```

次の出力は、ボードのすべての機能とメカニズムが有効であることを示しています。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms ar enabled, except CKM_DES_ECB,CKM_DES3_ECB.
random is disabled.

# cryptoadm enable provider=dca/0 all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is enabled.
```

例 3-12 ユーザーレベルの暗号化メカニズムのみを検索する

次の例では、ユーザーレベルライブラリ pkcs11_softtoken が提供するすべてのメカニズムを一覧表示します。

```
% cryptoadm list -m provider=/usr/lib/security/\
   $ISA/pkcs11_softtoken.so
Mechanisms:
CKM_DES_CBC
CKM_DES_CBC_PAD
CKM_DES_ECB
```

```
CKM_DES_KEY_GEN
CKM_DES_MAC_GENERAL
CKM_DES_MAC
…
CKM_ECDSA
CKM_ECDSA_SHA1
CKM_ECDH1_DERIVE
```

例 3-13 暗号化メカニズムで実行される機能を確認する

メカニズムでは、署名や鍵の生成など、特定の暗号化機能を実行します。-v -m オプションによって、すべてのメカニズムとその機能が表示されます。

この場合、管理者はどの機能に対して CKM_ECDSA* メカニズムが使用できるかを確認します。

```
% cryptoadm list -vm
User-level providers:
=====================
Provider: /usr/lib/security/$ISA/pkcs11_kernel.so
Number of slots: 3
Slot #2
Description: ncp/0 Crypto Accel Asym 1.0
...
CKM_ECDSA                    163  571  X  .  .  .  X  .  X  .  .  .  .  .  .  .  .
...

Provider: /usr/lib/security/$ISA/pkcs11_softtoken.so
...
CKM_ECDSA       112 571  .  .  .  .  X  .  X  .  .  .  .  .  .  .  .
CKM_ECDSA_SHA1  112 571  .  .  .  .  X  .  X  .  .  .  .  .  .  .  .
...
Kernel providers:
=================
...
ecc: CKM_EC_KEY_PAIR_GEN,CKM_ECDH1_DERIVE,CKM_ECDSA,CKM_ECDSA_SHA1
...
```

このリストは、これらのメカニズムが次のユーザーレベルプロバイダから使用可能であることを示しています。

- CKM_ECDSA および CKM_ECDSA_SHA1 – /usr/lib/security/$ISA/pkcs11_softtoken.so ライブラリ内のソフトウェア実装として
- CKM_ECDSA – /usr/lib/security/$ISA/pkcs11_kernel.so ライブラリ内の ncp/0 Crypto Accel Asym 1.0 による高速化

エントリの各項目は、メカニズムに関する情報を表しています。これらの ECC メカニズムについて、一覧は次を示しています。

- 最小の長さ – 112 バイト
- 最大の長さ – 571 バイト

- ハードウェア – ハードウェア上で使用できるかどうか。
- 暗号化 – データの暗号化に使用されません。
- 復号化 – データの復号化に使用されません。
- ダイジェスト – メッセージダイジェストの作成に使用されません。
- 署名 – データの署名に使用されます。
- 署名 + 回復 – データの署名に使用されません。そのデータはシグニチャーから回復できます。
- 検証 – 署名付きデータの検証に使用されます。
- 検証 + 回復 – シグニチャーから回復できるデータの検証に使用されません。
- 鍵の生成 – 非公開鍵の生成に使用されません。
- ペアの生成 – 鍵ペアの生成に使用されません。
- ラップ - ラップに使用されません。既存の鍵の暗号化。
- ラップ解除 – ラップされた鍵のラップ解除に使用されません。
- 派生 – ベース鍵からの新しい鍵の派生に使用されません。
- EC 機能 – 前の項目には含まれていない存在しない EC 機能

ソフトウェアプロバイダの追加

次の手順では、プロバイダをシステムに追加する方法について説明します。Crypto Management 権利プロファイルが割り当てられている管理者になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

▼ ソフトウェアプロバイダを追加する方法

1. **システムで使用可能なソフトウェアプロバイダを一覧表示します。**

```
% cryptoadm list
User-level providers:
Provider: /usr/lib/security/$ISA/pkcs11_kernel.so
Provider: /usr/lib/security/$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_tpm.so: all mechanisms are enabled.

Kernel software providers:
des
aes
arcfour
blowfish
```

```
camellia
sha1
sha2
md4
md5
rsa
swrand
n2rng/0
ncp/0
n2cp/0
```

2. **リポジトリからプロバイダを追加します。**

   既存のプロバイダソフトウェアには Oracle によって証明書が発行されています。

3. **プロバイダをリフレッシュします。**

   ソフトウェアプロバイダを追加した場合や、ハードウェアおよびそのハードウェアに指定されているポリシーを追加した場合は、プロバイダをリフレッシュする必要があります。

   ```
   # svcadm refresh svc:/system/cryptosvc
   ```

4. **新しいプロバイダをリストに追加します。**

   この場合、新しいカーネルソフトウェアプロバイダがインストールされています。

   ```
   # cryptoadm list
   …
   Kernel software providers:
   des
   aes
   arcfour
   blowfish
   camellia
   ecc
   sha1
   sha2
   md4
   md5
   rsa
   swrand
   sha3      <-- added provider
   …
   ```

例 3-14 ユーザーレベルソフトウェアプロバイダを追加する

次の例では、署名された PKCS #11 ライブラリをインストールします。

```
# pkgadd -d /cdrom/cdrom0/PKCSNew
    Answer the prompts
# svcadm refresh system/cryptosvc
# cryptoadm list
```

```
user-level providers:
==========================
/usr/lib/security/$ISA/pkcs11_kernel.so
/usr/lib/security/$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_tpm.so
/opt/lib/$ISA/libpkcs11.so.1      <-- added provider
```

暗号化フレームワークによってライブラリをテストする開発者は、ライブラリを手動でインストールできます。

```
# cryptoadm install provider=/opt/lib/\$ISA/libpkcs11.so.1
```

FIPS 140 が有効になったブート環境の作成

Oracle Solaris では、デフォルトでは FIPS 140 モードが無効になっています。この手順では、FIPS 140 モードのための新しいブート環境 (BE) を作成したあと、FIPS 140 を有効にして新しい BE にブートします。この方法を使用すると、バックアップ BE が提供されるため、FIPS 140 準拠テストによって発生する可能性のあるシステムパニックからすばやく回復できます。

FIPS の概要については、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』を参照してください。また、cryptoadm(1M) のマニュアルページおよび14 ページの「暗号化フレームワークおよび FIPS 140」も参照してください。

▼ FIPS 140 が有効になったブート環境を作成する方法

始める前に root 役割になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

1. **システムが FIPS 140 モードにあるかどうかを判定します。**

```
% cryptoadm list fips-140
User-level providers:
=====================
/usr/lib/security/$ISA/pkcs11_softtoken: FIPS-140 mode is disabled.

Kernel software providers:
==========================
des: FIPS-140 mode is disabled.
aes: FIPS-140 mode is disabled.
ecc: FIPS-140 mode is disabled.
sha1: FIPS-140 mode is disabled.
sha2: FIPS-140 mode is disabled.
rsa: FIPS-140 mode is disabled.
```

```
swrand: FIPS-140 mode is disabled.

Kernel hardware providers:
=========================:
```

2. **FIPS 140 バージョンの暗号化フレームワークのための新しい BE を作成します。**

   FIPS 140 モードを有効にする前に、まず beadm コマンドを使用して新しい BE を作成し、アクティブにしてから、ブートする必要があります。FIPS 140 が有効になったシステムでは、失敗した場合はパニックを引き起こす可能性のある準拠テストを実行します。そのため、FIPS 140 の境界に関する問題をデバッグしている間、ブートしてシステムを稼働状態にすることのできる使用可能な BE を確保しておくことが重要です。

   a. **現在の BE に基づいて BE を作成します。**

      この例では、S11.1-FIPS という名前の BE を作成します。

      ```
      # beadm create S11.1-FIPS-140
      ```

   b. **その BE をアクティブにします。**

      ```
      # beadm activate S11.1-FIPS-140
      ```

   c. **システムをリブートします。**

   d. **新しい BE で FIPS 140 モードを有効にします。**

      ```
      # cryptoadm enable fips-140
      ```

注記 - このサブコマンドは、ユーザーレベルの pkcs11_softtoken ライブラリおよびカーネルソフトウェアプロバイダの FIPS 140 未承認アルゴリズムは無効にしません。このフレームワークのコンシューマは、FIPS 140 承認アルゴリズムのみを使用することに責任を負っています。

FIPS 140 モードの影響の詳細は、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』を参照してください。また、cryptoadm(1M) のマニュアルページも参照してください。

---

3. **FIPS 140 を有効にしないで実行する場合は、FIPS 140 モードを無効にします。**

   元の BE にリブートするか、または現在の BE で FIPS 140 を無効にできます。

   ■ **元の BE にブートします。**

   ```
   # beadm list
   BE             Active Mountpoint Space   Policy Created
   ```

```
--              ------ ---------- -----   ------ -------
S11.1           -      -          48.22G   static 2012-10-10 10:10
S11.1-FIPS-140  NR     /          287.01M  static 2012-11-18 18:18
# beadm activate S11.1
# beadm list
BE              Active Mountpoint Space   Policy Created
--              ------ ---------- -----   ------ -------
S11.1           R      -          48.22G   static 2012-10-10 10:10
S11.1-FIPS-140  N      /          287.01M  static 2012-11-18 18:18
# reboot
```

■ **現在の BE で FIPS 140 モードを無効にして、リブートします。**

```
# cryptoadm disable fips-140
```

FIPS 140 モードは、システムがリブートされるまで動作を継続します。

```
# reboot
```

メカニズムの使用の防止

ライブラリプロバイダの暗号化メカニズムに使用すべきでないものが存在する場合、選択したメカニズムを削除できます。たとえば、別のライブラリ内の同じメカニズムの方がパフォーマンスが向上する場合や、セキュリティーの脆弱性を調査中の場合など、メカニズムを使用できないようにすることを検討する場合などです。

暗号化フレームワークが AES などのプロバイダの複数のモードを提供する場合、遅いメカニズムの使用を解除したり、破壊されたメカニズムを削除したりする場合があります。この手順を使用して、既知のセキュリティー脆弱性を持つアルゴリズムを削除することもできます。

ハードウェアプロバイダのメカニズムや乱数機能を選択して無効にすることができます。それらを再び有効にする場合は、例3-22「ハードウェアプロバイダのメカニズムと機能を有効にする」を参照してください。この例のハードウェア Sun Crypto Accelerator 1000 ボードは、乱数発生関数を提供します。

▼ ユーザーレベルのメカニズムが使用されないようにする方法

始める前に Crypto Management 権利プロファイルが割り当てられている管理者になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

1. **特定のユーザーレベルのソフトウェアプロバイダによって提供されるメカニズムを一覧表示します。**

```
% cryptoadm list -m provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so:
CKM_DES_CBC,CKM_DES_CBC_PAD,CKM_DES_ECB,CKM_DES_KEY_GEN,
CKM_DES3_CBC,CKM_DES3_CBC_PAD,CKM_DES3_ECB,CKM_DES3_KEY_GEN,
CKM_AES_CBC,CKM_AES_CBC_PAD,CKM_AES_ECB,CKM_AES_KEY_GEN,
…
```

2. **使用可能なメカニズムを一覧表示します。**

```
$ cryptoadm list -p
user-level providers:
=====================
…
/usr/lib/security/$ISA/pkcs11_softtoken.so: all mechanisms are enabled.
random is enabled.
…
```

3. **使用すべきでないメカニズムを無効にします。**

```
$ cryptoadm disable provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so \
> mechanism=CKM_DES_CBC,CKM_DES_CBC_PAD,CKM_DES_ECB
```

4. **使用可能なメカニズムを一覧表示します。**

```
$ cryptoadm list -p provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so: all mechanisms are enabled,
except CKM_DES_ECB,CKM_DES_CBC_PAD,CKM_DES_CBC. random is enabled.
```

例 3-15 ユーザーレベルソフトウェアプロバイダのメカニズムを有効にする

次の例では、無効になっている DES メカニズムを再び使用可能にします。

```
$ cryptoadm list -m provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so:
CKM_DES_CBC,CKM_DES_CBC_PAD,CKM_DES_ECB,CKM_DES_KEY_GEN,
CKM_DES3_CBC,CKM_DES3_CBC_PAD,CKM_DES3_ECB,CKM_DES3_KEY_GEN,
…
$ cryptoadm list -p provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so: all mechanisms are enabled,
except CKM_DES_ECB,CKM_DES_CBC_PAD,CKM_DES_CBC. random is enabled.
$ cryptoadm enable provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so \
> mechanism=CKM_DES_ECB
$ cryptoadm list -p provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so: all mechanisms are enabled,
except CKM_DES_CBC_PAD,CKM_DES_CBC. random is enabled.
```

例 3-16 ユーザーレベルソフトウェアプロバイダのメカニズムをすべて有効にする

次の例では、ユーザーレベルライブラリのメカニズムをすべて有効にします。

```
$ cryptoadm enable provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so all
```

```
$ cryptoadm list -p provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_softtoken.so: all mechanisms are enabled.
random is enabled.
```

例 3-17 ユーザーレベルライブラリを永続的に削除する

次の例では、/opt ディレクトリの libpkcs11.so.1 ライブラリが削除されます。

```
$ cryptoadm uninstall provider=/opt/lib/\$ISA/libpkcs11.so.1
$ cryptoadm list
user-level providers:
/usr/lib/security/$ISA/pkcs11_kernel.so
/usr/lib/security/$ISA/pkcs11_softtoken.so
/usr/lib/security/$ISA/pkcs11_tpm.so

kernel providers:
…
```

▼ カーネルソフトウェアメカニズムが使用されないようにする方法

始める前に Crypto Management 権利プロファイルが割り当てられている管理者になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

1. **特定のカーネルソフトウェアプロバイダによって提供されるメカニズムを一覧表示します。**

```
$ cryptoadm list -m provider=aes
aes: CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR,CKM_AES_CCM,CKM_AES_GCM,
CKM_AES_GMAC,CKM_AES_CFB128,CKM_AES_XTS,CKM_AES_XCBC_MAC
```

2. **使用可能なメカニズムを一覧表示します。**

```
$ cryptoadm list -p provider=aes
aes: all mechanisms are enabled.
```

3. **使用しないメカニズムを無効にします。**

```
$ cryptoadm disable provider=aes mechanism=CKM_AES_ECB
```

4. **使用可能なメカニズムを一覧表示します。**

```
$ cryptoadm list -p provider=aes
aes: all mechanisms are enabled, except CKM_AES_ECB.
```

例 3-18 カーネルソフトウェアプロバイダのメカニズムを有効にする

次の例では、無効になっている AES メカニズムを再び使用可能にします。

```
cryptoadm list -m provider=aes
aes: CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR,CKM_AES_CCM,
CKM_AES_GCM,CKM_AES_GMAC,CKM_AES_CFB128,CKM_AES_XTS,CKM_AES_XCBC_MAC
$ cryptoadm list -p provider=aes
aes: all mechanisms are enabled, except CKM_AES_ECB.
$ cryptoadm enable provider=aes mechanism=CKM_AES_ECB
$ cryptoadm list -p provider=aes
aes: all mechanisms are enabled.
```

例 3-19 カーネルソフトウェアプロバイダの使用を一時的に削除する

次の例では、AES プロバイダの使用を一時的に解除します。`unload` サブコマンドは、プロバイダのアンインストール中にプロバイダが自動的に読み込まれないようにするために使用します。たとえば、このプロバイダのメカニズムを修正するときに `unload` サブコマンドを使用できます。

```
$ cryptoadm unload provider=aes
```

```
$ cryptoadm list
…
Kernel software providers:
des
aes (inactive)
arcfour
blowfish
ecc
sha1
sha2
md4
md5
rsa
swrand
n2rng/0
ncp/0
n2cp/0
```

AES プロバイダは、暗号化フレームワークがリフレッシュされるまでは使用できません。

```
$ svcadm refresh system/cryptosvc
```

```
$ cryptoadm list
…
Kernel software providers:
des
aes
arcfour
blowfish
camellia
ecc
sha1
sha2
md4
md5
rsa
```

```
swrand
n2rng/0
ncp/0
n2cp/0
```

カーネルコンシューマがカーネルソフトウェアプロバイダを使用している場合は、ソフトウェアは読み込み解除されません。エラーメッセージが表示され、プロバイダを使用し続けることができます。

例 3-20 ソフトウェアプロバイダの使用を永続的に解除する

次の例では、AES プロバイダの使用を解除します。いったん削除すると、AES プロバイダはカーネルソフトウェアプロバイダのポリシー一覧に表示されません。

```
$ cryptoadm uninstall provider=aes
```

```
$ cryptoadm list
…
Kernel software providers:
des
arcfour
blowfish
camellia
ecc
sha1
sha2
md4
md5
rsa
swrand
n2rng/0
ncp/0
n2cp/0
```

カーネルコンシューマがカーネルソフトウェアプロバイダを使用している場合は、エラーメッセージが表示され、プロバイダを使用し続けることができます。

例 3-21 削除されたカーネルソフトウェアプロバイダを再インストールする

次の例では、AES カーネルソフトウェアプロバイダを再インストールします。削除されたカーネルプロバイダを再インストールするには、インストールするメカニズムを列挙する必要があります。

```
$ cryptoadm install provider=aes \
mechanism=CKM_AES_ECB,CKM_AES_CBC,CKM_AES_CTR,CKM_AES_CCM,
CKM_AES_GCM,CKM_AES_GMAC,CKM_AES_CFB128,CKM_AES_XTS,CKM_AES_XCBC_MAC
```

```
$ cryptoadm list
…
Kernel software providers:
des
```

```
aes
arcfour
blowfish
camellia
ecc
sha1
sha2
md4
md5
rsa
swrand
n2rng/0
ncp/0
n2cp/0
```

▼ ハードウェアプロバイダのメカニズムと機能を無効にする方法

始める前に Crypto Management 権利プロファイルが割り当てられている管理者になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

● **無効にするメカニズムまたは機能を選択します。**

ハードウェアプロバイダを一覧表示します。

```
# cryptoadm list
...
Kernel hardware providers:
dca/0
```

- **選択したメカニズムを無効にします。**

```
# cryptoadm list -m provider=dca/0
dca/0: CKM_RSA_PKCS, CKM_RSA_X_509, CKM_DSA, CKM_DES_CBC, CKM_DES3_CBC
random is enabled.
# cryptoadm disable provider=dca/0 mechanism=CKM_DES_CBC,CKM_DES3_CBC
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled except CKM_DES_CBC,CKM_DES3_CBC.
random is enabled.
```

- **乱数発生関数を無効にします。**

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is enabled.
# cryptoadm disable provider=dca/0 random
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is disabled.
```

- **すべてのメカニズムを無効にします。乱数発生関数は無効にしません。**

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is enabled.
# cryptoadm disable provider=dca/0 mechanism=all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are disabled. random is enabled.
```

■ **ハードウェアのすべての機能とメカニズムを無効にします。**

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is enabled.
# cryptoadm disable provider=dca/0 all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are disabled. random is disabled.
```

例 3-22 ハードウェアプロバイダのメカニズムと機能を有効にする

次の例では、一部のハードウェアの無効になっているメカニズムを選択して有効にします。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled except CKM_DES_ECB,CKM_DES3_ECB

.
random is enabled.
# cryptoadm enable provider=dca/0 mechanism=CKM_DES3_ECB
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled except CKM_DES_ECB.
random is enabled.
```

次の例では、乱数発生関数のみを有効にします。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_MD5,CKM_MD5_HMAC,….
random is disabled.
# cryptoadm enable provider=dca/0 random
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_MD5,CKM_MD5_HMAC,….
random is enabled.
```

次の例では、メカニズムのみを有効にします。乱数発生関数は無効にしておきます。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_MD5,CKM_MD5_HMAC,….
random is disabled.
# cryptoadm enable provider=dca/0 mechanism=all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is disabled.
```

次の例では、ボードのすべての機能とメカニズムを有効にします。

```
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled, except CKM_DES_ECB,CKM_DES3_ECB.
```

```
random is disabled.
# cryptoadm enable provider=dca/0 all
# cryptoadm list -p provider=dca/0
dca/0: all mechanisms are enabled. random is enabled.
```

すべての暗号化サービスのリフレッシュまたは再開

デフォルトでは、暗号化フレームワークは有効になっています。なんらかの理由で kcfd デーモンが失敗した場合は、サービス管理機能 (SMF) を使用すると暗号化サービスを再起動できます。詳細は、smf(5) および svcadm(1M) のマニュアルページを参照してください。暗号化サービスの再起動がゾーンに与える影響については、14 ページの「暗号化サービスとゾーン」を参照してください。

▼ すべての暗号化サービスをリフレッシュまたは再起動する方法

始める前に Crypto Management 権利プロファイルが割り当てられている管理者になる必要があります。詳細は、『Oracle Solaris 11.2 でのユーザーとプロセスのセキュリティー保護 』の「割り当てられている管理権利の使用」を参照してください。

1. **暗号化サービスのステータスを確認します。**

```
% svcs cryptosvc
STATE          STIME    FMRI
offline         Dec_09   svc:/system/cryptosvc:default
```

2. **暗号化サービスを有効にします。**

```
# svcadm enable svc:/system/cryptosvc
```

例 3-23 暗号化サービスをリフレッシュする

次の例では、暗号化サービスを大域ゾーンでリフレッシュします。その結果、すべての非大域ゾーンのカーネルレベルの暗号化ポリシーもリフレッシュされます。

```
# svcadm refresh system/cryptosvc
```

♦ ♦ ♦ 第 4 章

鍵管理フレームワーク

Oracle Solaris の鍵管理フレームワーク (KMF) 機能は、公開鍵オブジェクトを管理するためのツールとプログラミングインタフェースを提供します。公開鍵オブジェクトには、X.509 証明書や公開鍵/非公開鍵ペアが含まれます。これらのオブジェクトの格納形式としては、さまざまなものが使えます。また、KMF では、アプリケーションによる X.509 証明書の使用方法を定義したポリシーを管理するためのツールも提供されます。KMF では、サードパーティーのプラグインがサポートされています。

この章で扱う内容は、次のとおりです。

公開鍵技術の管理

KMF は、公開鍵技術 (PKI) の管理を集中化します。Oracle Solaris には、PKI テクノロジを使用する異なるアプリケーションがいくつか含まれています。各アプリケーションはそれぞれ独自のプログラミングインタフェース、鍵格納メカニズム、および管理ユーティリティーを提供します。アプリケーションがあるポリシー適用メカニズムを提供する場合、そのメカニズムはそのアプリケーションにしか適用されません。KMF では、アプリケーションは統一された管理ツール群、単一のプログラミングインタフェース群、および単一のポリシー適用メカニズムを使用します。これらのインタフェースを採用したすべてのアプリケーションの PKI ニーズは、これらの機能によって管理されます。

KMF では、次のインタフェースによって公開鍵技術の管理が統一されます。

- `pktool` コマンド – さまざまなキーストア内の PKI オブジェクト (証明書など) を管理します。
- `kmfcfg` コマンド –PKI ポリシーデータベースとサードパーティーのプラグインを管理します。

PKI ポリシー決定には、ある操作の検証方法などの操作が含まれます。また、PKI ポリシーは証明書の適用範囲を制限することもできます。たとえば、ある証明書が特定の目的にしか使用できないことを主張する PKI ポリシーを定義できます。そのようなポリシーは、その証明書がほかの要求のために使用されるのを禁止します。

- KMF ライブラリ – ベースとなるキーストアメカニズムを抽象化するプログラミングインタフェースが含まれています。

  アプリケーションは特定のキーストアメカニズムを選択する必要がなく、あるメカニズムから別のメカニズムへ移行できます。サポートされているキーストアは、PKCS #11、NSS、および OpenSSL です。このライブラリに含まれるプラグイン可能フレームワークを使えば、新しいキーストアメカニズムを追加できます。したがって、アプリケーションがある新しいメカニズムを使用する場合、アプリケーションを若干変更するだけで、その新しいキーストアを使用できるようになります。

鍵管理フレームワークのユーティリティー

KMF は、鍵の格納を管理するための手段を提供するとともに、それらの鍵の使用に関する包括的なポリシーを提供します。KMF は、次の 3 つの公開鍵技術のポリシー、鍵、および証明書を管理できます。

- PKCS #11 プロバイダ、つまり暗号化フレームワークからのトークン
- NSS、つまりネットワークセキュリティーサービス
- OpenSSL、ファイルベースのキーストア

`kmfcfg` ツールを使えば、KMF ポリシーエントリの作成、変更、または削除を行えます。また、このツールでは、フレームワークへのプラグインも管理します。KMF は、`pktool` コマンド経由でキーストアを管理します。詳細は、`kmfcfg(1)` と `pktool(1)` のマニュアルページ、および次の各セクションを参照してください。

KMF のポリシー管理

KMF のポリシーはデータベース内に格納されます。このポリシーデータベースは、KMF プログラミングインタフェースを使用するすべてのアプリケーションによって内部的にアクセスされます。このデータベースを使えば、KMF ライブラリによって管理される鍵や証明書の使用に、制約を設けることができます。アプリケーションは、証明書の検証を試みる際に、ポリシーデータベースをチェックします。`kmfcfg` コマンドはポリシーデータベースを変更します。

KMF プラグイン管理

kmfcfg コマンドは、プラグインのための次のサブコマンドを提供します。

- list plugin – KMF で管理されているプラグインを一覧表示します。
- install *plugin* – モジュールのパス名でプラグインをインストールし、そのプラグインのためのキーストアを作成します。KMF からプラグインを削除するには、キーストアを削除します。
- uninstall *plugin* – プラグインのキーストアを削除することによって、KMF からプラグインを削除します。
- modify *plugin* – プラグインのコードで定義されているオプション (debug など) を使用してプラグインを実行できるようにします。

詳細は、kmfcfg(1) のマニュアルページを参照してください。手順については、75 ページの「KMF でサードパーティーのプラグインを管理する方法」を参照してください。

KMF のキーストア管理

KMF では、PKCS #11 トークン、NSS、OpenSSL の 3 つの公開鍵技術に対してキーストアを管理します。pktool コマンドを使えば、これらすべての技術について次のことが行えます。

- 自己署名付き証明書を生成します
- 証明書リクエストを生成します
- 対称鍵を生成します
- 公開鍵と非公開鍵のペアを生成します
- 署名のために外部の認証局 (CA) に送信される PKCS #10 証明書署名リクエスト (CSR) を生成します
- PKCS #10 CSR に署名します
- キーストアにオブジェクトをインポートします
- キーストア内のオブジェクトを一覧表示します
- キーストアからオブジェクトを削除します
- CRL をダウンロードします

PKCS #11 および NSS 技術の場合には、pktool コマンドで、キーストアまたはキーストアのオブジェクトのパスフレーズを生成して PIN を設定することもできます。

pktool ユーティリティーの使用の詳細は、pktool(1) のマニュアルページおよび 表4-1「鍵管理フレームワークの使用のタスクマップ」を参照してください。

鍵管理フレームワークの使用

このセクションでは、パスワード、パスフレーズ、ファイル、キーストア、証明書、CRL などの公開鍵オブジェクトを、pktool コマンドを使って管理する方法について説明します。

鍵管理フレームワーク (KMF) を使うと公開鍵技術を集中管理できます。

表 4-1 鍵管理フレームワークの使用のタスクマップ

タスク	説明	参照先
証明書を作成します。	PKCS #11、NSS、または OpenSSL で使用される証明書を作成します。	63 ページの「pktool gencert コマンドを使って証明書を作成する方法」
証明書をエクスポートします。	証明書とそれをサポートする鍵を含むファイルを作成します。このファイルはパスワードで保護できます。	66 ページの「証明書と非公開鍵を PKCS #12 形式でエクスポートする方法」
証明書をインポートします。	別のシステムから取得した証明書をインポートします。	65 ページの「証明書をキーストアにインポートする方法」
	別のシステムから取得した PKCS #12 形式の証明書をインポートします。	例4-2「PKCS #12 ファイルをキーストアにインポートする」
パスフレーズを生成します。	PKCS #11 キーストアまたは NSS キーストアにアクセスするためのパスフレーズを生成します。	68 ページの「pktool setpin コマンドを使ってパスフレーズを生成する方法」
対称鍵を生成する。	ファイルの暗号化やファイルの MAC の作成で、またはアプリケーションのために使用する対称鍵を生成します。	26 ページの「pktool コマンドを使用して対称鍵を生成する方法」
鍵のペアを生成します。	アプリケーションで使用する公開鍵と非公開鍵のペアを生成します。	69 ページの「pktool genkeypair コマンドを使用して鍵のペアを生成する方法」
PKCS #10 CSR を生成します。	外部の認証局 (CA) が署名するための PKCS #10 証明書署名要求 (CSR) を生成します。	pktool(1) マニュアルページ
PKCS #10 CSR に署名します。	PKCS #10 CSR に署名します。	74 ページの「pktool signcsr コマンドを使用して証明書要求に署名する方法」

タスク	説明	参照先
KMF にプラグインを追加します。	プラグインをインストール、変更、および一覧表示します。また、KMF からプラグインを削除します。	75 ページの「KMF でサードパーティーのプラグインを管理する方法」

▼ pktool gencert コマンドを使って証明書を作成する方法

この手順では、自己署名付き証明書を作成し、その証明書を PKCS #11 キーストアに格納します。また、この処理の一部として、RSA 公開鍵/非公開鍵ペアも作成されます。非公開鍵は、証明書とともにキーストアに格納されます。

1. **自己署名付き証明書を生成する。**

```
% pktool gencert [keystore=keystore] label=label-name \
subject=subject-DN serial=hex-serial-number keytype=rsa/dsa keylen=key-size
```

`keystore=`*keystore*
公開鍵オブジェクトのタイプを指定することでキーストアを指定します。この値には、`nss`、`pkcs11`、または `file` を指定できます。このキーワードはオプションです。

`label=`*label-name*
発行者が証明書に与える一意の名前を指定します。

`subject=`*subject-DN*
証明書の識別名を指定します。

`serial=`*hex-serial-number*
16 進形式のシリアル番号を指定します。証明書の発行者が、`0x0102030405` などの番号を選択します。

`keytype=`*key type*
証明書に関連付けられた非公開鍵のタイプを指定するオプション変数。選択されたキーストアに使用できる鍵のタイプを見つけるには、`pktool(1)` のマニュアルページを確認してください。

FIPS 140 承認の鍵を使用するには、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」で、承認された鍵のタイプを確認してください。

`keylen=`*key size*
証明書に関連付けられた非公開鍵の長さを指定するオプション変数。

FIPS 140 承認の鍵を使用するには、『Using a FIPS 140 Enabled System in Oracle Solaris 11.2 』の「FIPS 140 Algorithms in the Cryptographic Framework」で選択した鍵のタイプの承認された鍵の長さを確認してください。

2. **キーストアの内容を確認します。**

```
% pktool list
Found number certificates.
1. (X.509 certificate)
Label:  label-name
ID: fingerprint that binds certificate to private key
Subject: subject-DN
Issuer:  distinguished-name
Serial:  hex-serial-number
n. ...
```

このコマンドは、キーストア内のすべての証明書を一覧表示します。次の例では、キーストアには証明書が 1 つしか含まれていません。

例 4-1 pktool を使って自己署名付き証明書を作成する

次の例では、My Company のあるユーザーが自己署名付き証明書を作成し、その証明書を PKCS #11 オブジェクト用のキーストアに格納しています。このキーストアは最初は空になっています。キーストアが初期化されていない場合、ソフトトークンの PIN は changeme であるため、pktool setpin コマンドを使用して PIN をリセットできます。コマンドオプションでは、FIPS 承認の鍵のタイプと鍵の長さ (RSA 2048) が指定されます。

```
% pktool gencert keystore=pkcs11 label="My Cert" \
subject="C=US, O=My Company, OU=Security Engineering Group, CN=MyCA" \
serial=0x000000001 keytype=rsa keylen=2048
Enter pin for Sun Software PKCS#11 softtoken:     Type PIN for token

% pktool list
No.  Key Type  Key Len.  Key Label
----------------------------------------------------
Asymmetric public keys:
1    RSA                 My Cert
Certificates:
1    X.509 certificate
Label: My Cert
ID: d2:7e:20:04:a5:66:e6:31:90:d8:53:28:bc:ef:55:55:dc:a3:69:93
Subject: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
Issuer: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
...
...
Serial: 0x00000010
...
```

▼ 証明書をキーストアにインポートする方法

この手順では、PEM または生の DER を使ってエンコードされた PKI 情報を含むファイルを、キーストアにインポートする方法を説明します。エクスポートの手順については、例4-4「証明書と非公開鍵を PKCS #12 形式でエクスポートする」を参照してください。

1. **証明書をインポートします。**

   % **pktool import keystore=***keystore* **infile=***infile-name* **label=***label-name*

2. **非公開 PKI オブジェクトをインポートする場合、プロンプトが表示されたときにパスワードを入力します。**

   a. **プロンプトで、ファイルのパスワードを入力します。**

   PKCS #12 形式のエクスポートファイルなど、非公開の PKI 情報をインポートする場合、そのファイルはパスワードを必要とします。インポートするファイルの作成者が PKCS #12 パスワードを教えてくれます。

   ```
   Enter password to use for accessing the PKCS12 file:     Type PKCS #12 password
   ```

   b. **プロンプトで、キーストアのパスワードを入力します。**

   ```
   Enter pin for Sun Software PKCS#11 softtoken:     Type PIN for token
   ```

3. **キーストアの内容を確認します。**

   ```
   % pktool list
   Found number certificates.
   1. (X.509 certificate)
   Label:   label-name
   ID: fingerprint that binds certificate to private key
   Subject:  subject-DN
   Issuer:   distinguished-name
   Serial:   hex-serial-number

   2. ...
   ```

例 4-2 PKCS #12 ファイルをキーストアにインポートする

次の例では、サードパーティーから取得した PKCS #12 ファイルをインポートしています。`pktool import` コマンドは、`gracedata.p12` ファイルから非公開鍵と証明書を取り出し、それらを指定されたキーストア内に格納します。

```
% pktool import keystore=pkcs11 infile=gracedata.p12 label=GraceCert
```

```
Enter password to use for accessing the PKCS12 file:    Type PKCS #12 password
Enter pin for Sun Software PKCS#11 softtoken:    Type PIN for token
Found 1 certificate(s) and 1 key(s) in gracedata.p12
% pktool list
No.  Key Type  Key Len.  Key Label
----------------------------------------------------
Asymmetric public keys:
1    RSA                GraceCert
Certificates:
1    X.509 certificate
Label: GraceCert
ID: 71:8f:11:f5:62:10:35:c2:5d:b4:31:38:96:04:80:25:2e:ad:71:b3
Subject: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
Issuer: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
Serial: 0x00000010
```

例 4-3 X.509 証明書をキーストアにインポートする

次の例では、PEM 形式の X.509 証明書を指定されたキーストアにインポートしています。この公開証明書はパスワードで保護されていません。ユーザーの公開キーストアもパスワードで保護されていません。

```
% pktool import keystore=pkcs11 infile=somecert.pem label="TheirCompany Root Cert"
% pktool list
No.  Key Type  Key Len.  Key Label
Certificates:
1    X.509 certificate
Label: TheirCompany Root Cert
ID: ec:a2:58:af:83:b9:30:9d:de:b2:06:62:46:a7:34:49:f1:39:00:0e
Subject: C=US, O=TheirCompany, OU=Security, CN=TheirCompany Root CA
Issuer: C=US, O=TheirCompany, OU=Security, CN=TheirCompany Root CA
Serial: 0x00000001
```

▼ 証明書と非公開鍵を PKCS #12 形式でエクスポートする方法

PKCS #12 形式のファイルを作成し、非公開鍵とそれに関連付けられた X.509 証明書をほかのシステムにエクスポートすることができます。このファイルへのアクセスはパスワードで保護されます。

1. **エクスポートする証明書を見つけます。**

```
% pktool list
Found number certificates.
1. (X.509 certificate)
Label:  label-name
```

```
ID: fingerprint that binds certificate to private key
Subject: subject-DN
Issuer:  distinguished-name
Serial:  hex-serial-number

2. ...
```

2. **鍵と証明書をエクスポートします。**

   pktool list コマンドから得られたキーストアとラベルを使用します。エクスポートファイルのファイル名を指定します。名前に空白が含まれている場合は、名前を二重引用符で囲みます。

   ```
   % pktool export keystore=keystore outfile=outfile-name label=label-name
   ```

3. **エクスポートファイルをパスワードで保護します。**

   プロンプトで、キーストアの現在のパスワードを入力します。ここで、エクスポートファイルのパスワードを作成します。ファイルの受信者は、インポート時にこのパスワードを入力する必要があります。

   ```
   Enter pin for Sun Software PKCS#11 softtoken:     Type PIN for token
   Enter password to use for accessing the PKCS12 file:    Create PKCS #12 password
   ```

   ヒント - パスワードはエクスポートファイルとは別に送ってください。パスワードを伝える最良の方法は、電話上など、通常の通信手段以外の手段を使用する方法です。

例 4-4 証明書と非公開鍵を PKCS #12 形式でエクスポートする

次の例では、ユーザーが非公開鍵を関連付けられた X.509 証明書とともに標準の PKCS #12 ファイルにエクスポートしています。このファイルは、ほかのキーストアにインポートできます。PKCS #11 パスワードはソースのキーストアを保護します。PKCS #12 パスワードは、PKCS #12 ファイル内の非公開データを保護するために使用されます。このパスワードはファイルのインポート時に必要となります。

```
% pktool list
No.  Key Type  Key Len.  Key Label
----------------------------------------------------
Asymmetric public keys:
1    RSA                 My Cert
Certificates:
1    X.509 certificate
Label: My Cert
ID: d2:7e:20:04:a5:66:e6:31:90:d8:53:28:bc:ef:55:55:dc:a3:69:93
Subject: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
Issuer: C=US, O=My Company, OU=Security Engineering Group, CN=MyCA
Serial: 0x000001

% pktool export keystore=pkcs11 outfile=mydata.p12 label="My Cert"
```

```
Enter pin for Sun Software PKCS#11 softtoken:      Type PIN for token
Enter password to use for accessing the PKCS12 file:     Create PKCS #12 password
```

続いて、このユーザーは受信者に電話をかけ、PKCS #12 パスワードを伝えます。

▼ pktool setpin コマンドを使ってパスフレーズを生成する方法

キーストア内のあるオブジェクトに対して、あるいはキーストアそのものに対して、パスフレーズを生成することができます。このパスフレーズは、オブジェクトやキーストアにアクセスする際に必要となります。キーストア内のオブジェクトに対するパスフレーズの生成例については、例 4-4「証明書と非公開鍵を PKCS #12 形式でエクスポートする」を参照してください。

1. **キーストアにアクセスするためのパスフレーズを生成します。**

   ```
   % pktool setpin keystore=nss|pkcs11 [dir=directory]
   ```

   デフォルトの鍵格納ディレクトリは /var/*username* です。

   PKCS #11 キーストアの初期パスワードは changeme です。NSS キーストアの最初のパスワードは空のパスワードです。

2. **プロンプトに答えます。**

   現在のトークンパスフレーズの入力を要求されたら、PKCS #11 キーストアのトークン PIN を入力するか、または NSS キーストアを示す Return キーを押します。

   ```
   Enter current token passphrase:        Type PIN or press the Return key
   Create new passphrase:                Type the passphrase that you want to use
   Re-enter new passphrase:                  Retype the passphrase
   Passphrase changed.
   ```

   これでキーストアがパスフレーズで保護されます。パスフレーズを忘れると、キーストア内のオブジェクトにアクセスできなくなります。

3. **(オプション) トークンのリストを表示します。**

   ```
   # pktool tokens
   ```

   出力は、メタスロットが有効かどうかによって異なります。メタスロットの詳細は、10 ページの「暗号化フレームワークの概念」を参照してください。

   - メタスロットが有効な場合は、pktools token コマンドで次のような出力が生成されます。

```
ID Slot   Name                       Token Name                        Flags
--        ---------                  ----------                        -----
0         Sun Metaslot               Sun Metaslot
1         Sun Crypto Softtoken       Sun Software PKCS#11 softtoken    LIX
2         PKCS#11 Interface for TPM  TPM                               LXS
```

- メタスロットが無効な場合は、`pktools token` コマンドで次のような出力が生成されます。

```
ID Slot   Name                       Token Name                        Flags
--        ---------                  ----------                        -----
1         Sun Crypto Softtoken       Sun Software PKCS#11 softtoken    LIX
2         PKCS#11 Interface for TPM  TPM                               LXS
```

2 つの出力バージョンでは、次のような組み合わせのフラグが可能です。

- `L` – ログインが必要
- `I` – 初期化済み
- `X` – ユーザー PIN の有効期限切れ
- `S` – SO PIN の有効期限切れ

例 4-5 キーストアをパスフレーズで保護する

次の例は、NSS データベースのパスフレーズを設定する方法を示したものです。パスフレーズがまだ作成されていないため、最初のプロンプトで Return キーを押します。

```
% pktool setpin keystore=nss dir=/var/nss
Enter current token passphrase:      Press the Return key
Create new passphrase: has8n0NdaH
Re-enter new passphrase: has8n0NdaH
Passphrase changed.
```

▼ pktool genkeypair コマンドを使用して鍵のペアを生成する方法

アプリケーションによっては、公開鍵と非公開鍵のペアが必要なものがあります。この手順では、これらの鍵のペアを作成して格納します。

1. **(オプション) キーストアの使用を予定している場合は、キーストアを作成します。**

- **PKCS #11 キーストアを作成して初期化する方法については、**68 ページの「pktool setpin コマンドを使ってパスフレーズを生成する方法」**を参照してください。**

- **NSS キーストアを作成して初期化するには、**例4-5「キーストアをパスフレーズで保護する」**を参照してください。**

2. **鍵のペアを作成します。**

   次のいずれかを実行します。

   - **鍵のペアを作成し、その鍵のペアをファイル内に格納します。**

     ディスク上のファイルから直接鍵を読み取るアプリケーションの場合は、ファイルベースの鍵が作成されます。通常、OpenSSL 暗号化ライブラリを直接使用するアプリケーションでは、そのアプリケーションのための鍵と証明書をファイル内に格納する必要があります。

注記 - `file` キーストアは、楕円曲線 (`ec`) 鍵および証明書をサポートしていません。

```
% pktool genkeypair keystore=file outkey=key-filename \
[format=der|pem] [keytype=rsa|dsa] [keylen=key-size]
```

`keystore=file`

`file` の値は、鍵の格納場所のファイルタイプを指定します。

`outkey=`*key-filename*

鍵のペアが格納されるファイルの名前を指定します。

`format=der|pem`

鍵のペアのエンコード形式を指定します。`der` 出力はバイナリであり、`pem` 出力は ASCII です。

`keytype=rsa|dsa`

`file` キーストア内に格納できる鍵のペアのタイプを指定します。定義については、DSA と RSA を参照してください。

`keylen=`*key-size*

鍵の長さをビット数で指定します。8 で割り切れる数にする必要があります。指定できるキーサイズを確認するには、`cryptoadm list -vm` コマンドを使用します。

   - **鍵のペアを作成して PKCS #11 キーストア内に格納します。**

この方法を使用するには、ステップ 1 を完了する必要があります。

PKCS #11 キーストアは、オブジェクトをハードウェアデバイス上に格納するために使用されます。これらのデバイスには、Sun Crypto Accelerator 6000 カード、TPM (Trusted Platform Module) デバイス、または暗号化フレームワークに組み込まれたスマートカードがあります。また、PKCS #11 を使用すると、オブジェクトを `softtoken` (ソフトウェアベースのトークン) 内にも格納できます。このトークンでは、オブジェクトがディスク上の個人のサブディレクトリ内に格納されます。詳細は、`pkcs11_softtoken` (5) のマニュアルページを参照してください。

指定したラベルによって、鍵のペアをキーストアから取得できます。

```
% pktool genkeypair label=key-label \
[token=token[:manuf[:serial]]] \
[keytype=rsa|dsa|ec]  [curve=ECC-Curve-Name]]\
[keylen=key-size] [listcurves]
```

`label=`*key-label*

鍵のペアのラベルを指定します。鍵のペアは、そのラベルによってキーストアから取得できます。

`token=`*token*[:*manuf*[:*serial*]]

トークン名を指定します。デフォルトでは、トークン名は `Sun Software PKCS#11 softtoken` です。

`keytype=rsa|dsa|ec [curve=`*ECC-Curve-Name*`]`

鍵のペアのタイプを指定します。楕円曲線 (`ec`) タイプの場合は、必要に応じて曲線名を指定します。曲線名は、`listcurves` オプションに対する出力として一覧表示されます。

`keylen=`*key-size*

鍵の長さをビット数で指定します。8 で割り切れる数にする必要があります。

`listcurves`

`ec` 鍵タイプの `curve=` オプションの値として使用できる楕円曲線の名前を一覧表示します。

- **鍵のペアを生成し、それを NSS キーストア内に格納します。**

NSS キーストアは、NSS を主要な暗号化インタフェースとして使用するサーバーによって使用されます。

この方法を使用するには、ステップ 1 を完了する必要があります。

```
% pktool keystore=nss genkeypair label=key-nickname \
[token=token[:manuf[:serial]]] \
[dir=directory-path] [prefix=database-prefix] \
[keytype=rsa|dsa|ec] [curve=ECC-Curve-Name]] \
[keylen=key-size] [listcurves]
```

keystore=nss

nss の値は、鍵の格納場所の NSS タイプを指定します。

label=*nickname*

鍵のペアのラベルを指定します。鍵のペアは、そのラベルによってキーストアから取得できます。

token=*token*[:*manuf*[:*serial*]]

トークン名を指定します。デフォルトでは、トークンは Sun Software PKCS#11 softtoken です。

dir=*directory*

NSS データベースのディレクトリパスを指定します。デフォルトでは、*directory* は現在のディレクトリです。

prefix=*database-prefix*

NSS データベースの接頭辞を指定します。デフォルトは接頭辞なしです。

keytype=rsa|dsa|ec [curve=*ECC-Curve-Name*]

鍵のペアのタイプを指定します。楕円曲線タイプの場合は、必要に応じて曲線名を指定します。曲線名は、listcurves オプションに対する出力として一覧表示されます。

keylen=*key-size*

鍵の長さをビット数で指定します。8 で割り切れる数にする必要があります。

listcurves

ec 鍵タイプの curve= オプションの値として使用できる楕円曲線の名前を一覧表示します。

3. **(オプション) 鍵が存在することを確認します。**

鍵を格納した場所に応じて、次のコマンドのいずれかを使用します。

- *key-filename* **ファイル内の鍵を確認します。**

```
% pktool list keystore=file objtype=key infile=key-filename
Found n keys.
Key #1 - keytype:location (keylen)
```

- **PKCS #11 キーストア内の鍵を確認します。**

```
$ pktool list objtype=key
Enter PIN for keystore:
Found n keys.
Key #1 - keytype:location (keylen)
```

- **NSS キーストア内の鍵を確認します。**

```
% pktool list keystore=nss dir=directory objtype=key
```

例 4-6 pktool コマンドを使用して鍵のペアを作成する

次の例では、ユーザーがはじめて PKCS #11 キーストアを作成します。RSA 鍵のペアのキーサイズを確認したあと、ユーザーは、アプリケーションのための鍵のペアを生成します。最後に、ユーザーは、キーストア内に鍵のペアが存在することを確認します。ユーザーは、RSA 鍵のペアの 2 番目のインスタンスをハードウェア上に格納できることに気付きました。ユーザーは token 引数を指定しなかったため、鍵のペアは Sun Software PKCS#11 softtoken として格納されます。

```
# pktool setpin
Create new passphrase:
Re-enter new passphrase:     Retype password
Passphrase changed.
% cryptoadm list -vm | grep PAIR
...
CKM_DSA_KEY_PAIR_GEN       512  3072 .  .  .  .  .  .  .  .  .  X  .  .  .  .
CKM_RSA_PKCS_KEY_PAIR_GEN  256  8192 .  .  .  .  .  .  .  .  .  X  .  .  .  .
...
CKM_RSA_PKCS_KEY_PAIR_GEN  256  2048 X  .  .  .  .  .  .  .  .  X  .  .  .  .
ecc: CKM_EC_KEY_PAIR_GEN,CKM_ECDH1_DERIVE,CKM_ECDSA,CKM_ECDSA_SHA1
% pktool genkeypair label=specialappkeypair keytype=rsa keylen=2048
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password

% pktool list
Enter PIN for Sun Software PKCS#11 softtoken  :     Type password
No.      Key Type      Key Len.      Key Label
----------------------------------------------------
Asymmetric public keys:
1        RSA                         specialappkeypair
```

例 4-7 楕円曲線アルゴリズムを使用する鍵のペアを作成する

次の例では、ユーザーが楕円曲線 (ec) 鍵のペアをキーストアに追加し、曲線名を指定したあと、キーストア内に鍵のペアが存在することを確認します。

```
% pktool genkeypair listcurves
secp112r1, secp112r2, secp128r1, secp128r2, secp160k1
```

```
.
.
.
c2pnb304w1, c2tnb359v1, c2pnb368w1, c2tnb431r1, prime192v2
prime192v3
% pktool genkeypair label=eckeypair keytype=ec curves=c2tnb431r1
% pktool list
Enter PIN for Sun Software PKCS#11 softtoken  :    Type password
No.  Key Type  Key Len.  Key Label
----------------------------------------------------
Asymmetric public keys:
1    ECDSA              eckeypair
```

▼ pktool signcsr コマンドを使用して証明書要求に署名する方法

この手順は、PKCS #10 証明書署名要求 (CSR) に署名するために使用されます。PEM または DER 形式の CSR を使用できます。署名プロセスによって X.509 v3 証明書が発行されます。PKCS #10 CSR を生成するには、pktool(1) のマニュアルページを参照してください。

始める前に この手順では、自身が認証局 (CA) であり、CSR を受信しており、それがファイル内に格納されてると想定します。

1. **pktool signcsr コマンドへの必要な引数に関する次の情報を収集します。**

signkey	署名者の鍵を PKCS #11 キーストア内に格納した場合、signkey は、この非公開鍵を取得するラベルです。 署名者の鍵を NSS キーストアまたはファイルキーストア内に格納した場合、signkey は、この非公開鍵を保持するファイル名です。
csr	CSR のファイル名を指定します。
serial	署名される証明書のシリアル番号を指定します。
outcer	署名される証明書のファイル名を指定します。
issuer	自分の CA 発行者名を識別名 (DN) 形式で指定します。

   signcsr サブコマンドのオプションの引数については、pktool(1) のマニュアルページを参照してください。

2. **要求に署名し、証明書を発行します。**

たとえば、次のコマンドは、PKCS #11 リポジトリにある署名者の鍵を使用して証明書に署名します。

```
# pktool signcsr signkey=CASigningKey \
csr=fromExampleCoCSR \
serial=0x12345678 \
outcert=ExampleCoCert2010 \
issuer="O=Oracle Corporation, \
OU=Oracle Solaris Security Technology, L=Redwood City, ST=CA, C=US, \
CN=rootsign Oracle"
```

次のコマンドは、ファイルにある署名者の鍵を使用して証明書に署名します。

```
# pktool signcsr signkey=CASigningKey \
csr=fromExampleCoCSR \
serial=0x12345678 \
outcert=ExampleCoCert2010 \
issuer="O=Oracle Corporation, \
OU=Oracle Solaris Security Technology, L=Redwood City, ST=CA, C=US, \
CN=rootsign Oracle"
```

3. **証明書を要求者に送信します。**

   電子メール、Web サイト、またはその他のメカニズムを使用すると証明書を要求者に配信できます。

   たとえば、電子メールを使用して `ExampleCoCert2010` ファイルを要求者に送信できます。

▼ KMF でサードパーティーのプラグインを管理する方法

プラグインは、キーストア名を指定することによって識別します。KMF にプラグインを追加すると、ソフトウェアは、そのプラグインをキーストア名で識別します。プラグインは、オプションを受け入れるように定義できます。この手順には KMF からプラグインを削除する方法が含まれています。

1. **プラグインをインストールします。**

```
% /usr/bin/kmfcfg install keystore=keystore-name \
modulepath=path-to-plugin [option="option-string"]
```

各情報の意味は次のとおりです

keystore-name	指定したキーストアの一意の名前を指定します。

path-to-plugin KMF プラグインの共有ライブラリオブジェクトへのフルパスを指定します。

option-string 共有ライブラリオブジェクトへのオプションの引数を指定します。

2. **プラグインを一覧表示します。**

```
% kmfcfg list plugin
keystore-name:path-to-plugin [(built-in)] | [;option=option-string]
```

3. **プラグインを削除するには、そのプラグインをアンインストールしたあと、削除を確認します。**

```
% kmfcfg uninstall keystore=keystore-name
% kmfcfg plugin list
```

例 4-8 オプションを指定して KMF プラグインを呼び出す

次の例では、管理者が KMF プラグインをサイト固有のディレクトリ内に格納します。このプラグインは、debug オプションを受け入れるように定義されています。管理者はプラグインを追加したあと、そのプラグインがインストールされていることを確認します。

```
# /usr/bin/kmfcfg install keystore=mykmfplug \
modulepath=/lib/security/site-modules/mykmfplug.so
# kmfcfg list plugin
KMF plugin information:
-----------------------
pkcs11:kmf_pkcs11.so.1 (built-in)
file:kmf_openssl.so.1 (built-in)
nss:kmf_nss.so.1 (built-in)
mykmfplug:/lib/security/site-modules/mykmfplug.so
# kmfcfg modify plugin keystore=mykmfplug option="debug"
# kmfcfg list plugin
KMF plugin information:
-----------------------
...
mykmfplug:/lib/security/site-modules/mykmfplug.so;option=debug
```

このプラグインは現在、デバッグモードで動作しています。

セキュリティー用語集

アクセス制御リスト (ACL)
アクセス制御リスト (ACL) を使用すると、従来の UNIX ファイル保護よりもきめ細かな方法でファイルセキュリティーを確立できます。たとえば、特定のファイルにグループ読み取り権を設定し、そのグループ内の 1 人のメンバーだけにそのファイルへの書き込み権を与えることが可能です。

アプリケーションサーバー
ネットワークアプリケーションサーバーを参照してください。

アルゴリズム
暗号化アルゴリズム。これは、入力を暗号化 (ハッシング) する既成の再帰的な計算手続きです。

暗号化アルゴリズム
アルゴリズムを参照してください。

暗号化フレームワークにおけるポリシー
Oracle Solaris の暗号化フレームワーク機能では、ポリシーは既存の暗号化メカニズムの無効化です。無効に設定されたメカニズムは使用できなくなります。暗号化フレームワークにおけるポリシーにより、プロバイダ (DES など) からの特定のメカニズム (`CKM_DES_CBC`など) を使用できなくなることがあります。

インスタンス
主体名の 2 番目の部分。インスタンスは、主体の主ノード指定します。サービス主体の場合、インスタンスは必ず指定する必要があります。`host/central.example.com` にあるように、インスタンスはホストの完全修飾ドメイン名です。ユーザー主体の場合、インスタンスは省略することができます。ただし、`jdoe` と `jdoe/admin` は、一意の主体です。プライマリ、主体名、サービス主体、ユーザー主体も参照してください。

オーセンティケータ
オーセンティケータは、KDC にチケットを要求するときおよびサーバーにサービスを要求するときに、クライアントから渡されます。オーセンティケータには、クライアントとサーバーだけが知っているセッション鍵を使用して生成された情報が含まれます。オーセンティケータは、最新の識別として検査され、そのトランザクションが安全であることを示します。これをチケットとともに使用すると、ユーザー主体を認証できます。オーセンティケータには、ユーザーの主体名、ユーザーのホストの IP アドレス、タイムスタンプが含まれます。チケットとは異なり、オーセンティケータは一度しか使用できません。通常、サービスへのアクセスが要求されたときに使用されます。オーセンティケータは、そのクライアントとそのサーバーのセッション鍵を使用して暗号化されます。

鍵
1. 一般には、次に示す 2 種類の主要鍵のどちらか一方です。

- *対称鍵* – 復号化鍵とまったく同じ暗号化鍵。対称鍵はファイルの暗号化に使用されます。

- *非対称鍵*または*公開鍵* – Diffie-Hellman や RSA などの公開鍵アルゴリズムで使用される鍵。公開鍵には、1 人のユーザーしか知らない非公開鍵、サーバーまたは一般リソースによって使用される公開鍵、およびこれらの 2 つを組み合わせた公開鍵と非公開鍵のペアがあります。非公開鍵は、「秘密鍵」とも呼ばれます。公開鍵は、「共有鍵」や「共通鍵」とも呼ばれます。

2. キータブファイルのエントリ (主体名)。キータブファイルも参照してください。

3. Kerberos では暗号化鍵であり、次の 3 種類があります。

- 「非公開鍵」 – 主体と KDC によって共有される暗号化鍵。システムの外部に配布されます。非公開鍵も参照してください。
- 「サービス鍵」– 非公開鍵と同じ目的で使用されますが、この鍵はサーバーとサービスによって使用されます。サービス鍵も参照してください。
- 「セッション鍵」 – 一時的な暗号化鍵。2 つの主体の間で使用され、その有効期限は 1 つのログインセッションの期間に制限されます。セッション鍵も参照してください。

仮想プライベートネットワーク (VPN)
暗号化とトンネルを使用して、セキュアな通信を提供するネットワーク。公開ネットワークを通してユーザーを接続します。

関係
`kdc.conf` または `krb5.conf` ファイルに定義される構成変数または関係の 1 つ。

監査トレール
すべてのホストから収集した一連の監査ファイル。

監査ファイル
バイナリ形式の監査ログ。監査ファイルは、監査ファイルシステム内に個別に格納されます。

監査ポリシー
どの監査イベントが記録されるかを決定する設定であり、大域の設定とユーザーごとの設定があります。大域の設定は監査サービスに適用され、一般にどのオプション情報を監査トレールに含めるかを決定します。2 つの設定 `cnt` と `ahlt` は、監査キューがいっぱいになった時点でのシステムの処理を決定します。たとえば、各監査レコードにシーケンス番号を含めるように監査ポリシーを設定できます。

キーストア
キーストアは、アプリケーションによる取得のために、パスワード、パスフレーズ、証明書、およびその他の認証オブジェクトを保持します。キーストアはテクノロジ固有にすることも、複数のアプリケーションで使用される場所にすることもできます。

キータブファイル
1 つまたは複数の鍵 (主体) が含まれるキーテーブル。ホストまたはサービスとキータブファイルとの関係は、ユーザーとパスワードの関係と似ています。

基本セット
ログイン時にユーザーのプロセスに割り当てられる一連の特権。変更されていないシステムの場合、各ユーザーの初期の継承可能セットはログイン時の基本セットと同じです。

機密性
プライバシを参照してください。

強化 ホストが本来抱えるセキュリティー上の脆弱性を解決するためにオペレーティングシステムのデフォルト構成を変更すること。

許可されたセット プロセスによって使用できる一連の特権。

クライアント 狭義では、rlogin を使用するアプリケーションなど、ユーザーの代わりにネットワークサービスを使用するプロセスを指します。サーバー自身が他のサーバーやサービスのクライアントになる場合もあります。

広義では、a) Kerberos 資格を受け取り、b) サーバーから提供されたサービスを利用するホストを指します。

広義では、サービスを使用する主体を指します。

クライアント主体 (RPCSEC_GSS API) RPCSEC_GSS で保護されたネットワークサービスを使用するクライアント (ユーザーまたはアプリケーション)。クライアント主体名は、rpc_gss_principal_t 構造体の形式で格納されます。

クロックスキュー Kerberos 認証システムに参加しているすべてのホスト上の内部システムクロックに許容できる最大時間。参加しているホスト間でクロックスキューを超過すると、要求が拒否されます。クロックスキューは、krb5.conf ファイルに指定できます。

継承可能セット プロセスが exec の呼び出しを通して継承できる一連の特権。

権利 すべての機能を持つスーパーユーザーの代替アカウント。ユーザー権利の管理およびプロセス権利の管理で、組織はスーパーユーザーの特権を分割して、ユーザーまたは役割に割り当てることができます。Oracle Solaris の権利は、カーネル特権、承認、または特定の UID や GID としてプロセスを実行する機能として実装されています。権利は、権利プロファイルおよび役割で収集できます。

権利プロファイル プロファイルとも呼ばれます。役割またはユーザーに割り当てることができるセキュリティーオーバーライドの集合。権利プロファイルには、承認、特権、セキュリティー属性が割り当てられたコマンド、および補足プロファイルと呼ばれるその他の権利プロファイルを含めることができます。

権利ポリシー コマンドに関連付けられるセキュリティーポリシー。現在、Oracle Solaris で有効なポリシーは solaris です。solaris ポリシーでは、特権と拡張特権ポリシー、承認、および setuid セキュリティー属性が認識されます。

公開オブジェクト root ユーザーによって所有され、すべてのユーザーが読み取ることのできるファイル。たとえば、/etc ディレクトリ内のファイルです。

公開鍵技術のポリシー 鍵管理フレームワーク (KMF) におけるポリシーは、証明書の使用を管理します。KMF ポリシーデータベースを使えば、KMF ライブラリによって管理される鍵や証明書の使用に、制約を設けることができます。

公開鍵の暗号化
暗号化スキームの 1 つ。各ユーザーが 1 つの公開鍵と 1 つの非公開鍵を所有します。公開鍵の暗号化では、送信者は受信者の公開鍵を使用してメッセージを暗号化し、受信者は非公開鍵を使用してそれを復号化します。Kerberos サービスは非公開鍵システムです。非公開鍵の暗号化も参照してください。

更新可能チケット
有効期限の長いチケットは、セキュリティーを低下させることがあるため、「更新可能」チケットに指定することができます。更新可能チケットには 2 つの有効期限があります。 a) チケットの現在のインスタンスの有効期限と、b) 任意のチケットの最長有効期限です。クライアントがチケットの使用を継続するときは、最初の有効期限が切れる前にチケットの有効期限を更新します。たとえば、すべてのチケットの最長有効期限が 10 時間のときに、あるチケットが 1 時間だけ有効だとします。このチケットを保持するクライアントが 1 時間を超えて使用する場合は、チケットの有効期限を更新する必要があります。チケットが最長有効期限に達すると、チケットの有効期限が自動的に切れ、それ以上更新できなくなります。

コンシューマ
Oracle Solaris の暗号化フレームワーク機能では、コンシューマはプロバイダが提供する暗号化サービスのユーザー。コンシューマになりえるものとして、アプリケーション、エンドユーザー、カーネル処理などが挙げられます。Kerberos、IKE、IPsec などはコンシューマの例です。プロバイダの例は、プロバイダを参照してください。

サーバー
ネットワーククライアントにリソースを提供する主体。たとえば、システム `central.example.com` に `ssh` で接続する場合、そのシステムが `ssh` サービスを提供するサーバーになります。サービス主体も参照してください。

サーバー主体
(RPCSEC_GSS API) サービスを提供する主体。サーバー主体は、*service*@*host* という書式で ASCII 文字列として格納されます。クライアント主体も参照してください。

サービス
1. ネットワーククライアントに提供されるリソース。多くの場合、複数のサーバーから提供されます。たとえば、マシン `central.example.com` に `rlogin` で接続する場合、そのマシンが `rlogin` サービスを提供するサーバーになります。

2. 認証以外の保護レベルを提供するセキュリティーサービス (整合性またはプライバシ)。整合性とプライバシも参照してください。

サービス鍵
サービス主体と KDC によって共有される暗号化鍵。システムの外部に配布されます。鍵も参照してください。

サービス主体
1 つまたは複数のサービスに Kerberos 認証を提供する主体。サービス主体では、プライマリ名はサービス名 (`ftp` など) で、インスタンスはサービスを提供するシステムの完全指定ホスト名になります。ホスト主体、ユーザー主体も参照してください。

最小化
サーバーを稼働させる上で必要な最小限のオペレーティングシステムをインストールすること。サーバーの処理に直接影関係がないソフトウェアはすべて、インストールされないか、あるいはインストール後削除されます。

最少特権
指定されたプロセスにスーパーユーザー権限のサブセットのみを提供するセキュリティーモデル。最小特権モデルでは、通常のユーザーに、ファイルシステムのマウントやファイルの所有権の変更などの個人の管理タスクを実行できる十分な特権を割り当てます。これに対して、プロ

セスは、スーパーユーザーの完全な権限 (つまり、すべての特権) ではなく、タスクを完了するために必要な特権のみで実行されます。バッファーオーバーフローなどのプログラミングエラーによる損害を、保護されたシステムファイルの読み取りまたは書き込みやマシンの停止などの重要な機能にはアクセスできない root 以外のユーザーに封じ込めることができます。

最少特権の原則 最少特権を参照してください。

再認証 コンピュータ操作を実行するためにパスワードを指定する際の要件。通常、`sudo` 操作では再認証が必要です。認証済み権利プロファイルには、再認証が必要なコマンドを含めることができます。認証済み権利プロファイルを参照してください。

シード 乱数生成のスターター (元になる値)。この値から生成が開始されます。このスターターがランダムソースから生じる場合、このシードは「ランダムシード」と呼ばれます。

資格 チケットと照合セッション鍵を含む情報パッケージ。主体の識別情報を認証するときに使用します。チケットとセッション鍵も参照してください。

資格キャッシュ KDC から受信した資格を含むストレージ領域。通常はファイルです。

主体 1. ネットワーク通信に参加する、一意の名前を持つ「クライアントまたはユーザー」あるいは「サーバーまたはサービス」のインスタンス。Kerberos トランザクションでは、主体 (サービス主体とユーザー主体) 間、または主体と KDC の間で対話が行われます。つまり、主体とは、Kerberos がチケットを割り当てることができる一意のエンティティーのことです。主体名、サービス主体、ユーザー主体も参照してください。

2. (RPCSEC_GSS API) クライアント主体、サーバー主体を参照してください。

主体名 1. 主体の名前。書式は、*primary/instance@REALM*。インスタンス、プライマリ、レルムも参照してください。

2. (RPCSEC_GSS API) クライアント主体、サーバー主体を参照してください。

承認 1. Kerberos では、主体がサービスを使用できるかどうか、主体がアクセスできるオブジェクト、各オブジェクトに許可するアクセスの種類を決定するプロセス。

2. ユーザー権利の管理で、役割またはユーザーに割り当てる (権利プロファイルに埋め込む) ことができる一連の操作 (そうしない場合、セキュリティーポリシーによって拒否される) を実行するための権利。承認はカーネルではなく、ユーザーアプリケーションレベルで適用されます。

証明書 公開鍵証明書は、公開鍵の値 (名前や署名者などの証明書の生成に関するいくつかの情報を含む)、証明書のハッシュまたはチェックサム、およびハッシュのデジタル署名をエンコードする一連のデータです。これらの値が一体となって証明書を形成します。デジタル署名は証明書が変更されていないことを保証します。

詳細は、鍵を参照してください。

初期チケット
直接発行されるチケット (既存のチケット許可チケットは使用されない)。パスワードを変更するアプリケーションなどの一部のサービスでは、クライアントが非公開鍵を知っていることを確認するために、「初期」と指定されたチケットを要求することができます。初期チケットを使用した検査は、クライアントが最近認証されたことを証明するときに重要になります。チケット許可チケットの場合は、取得してから時間が経過していることがあります。

信頼できるユーザー
ある程度の信頼レベルで管理タスクを実行できるように決定されたユーザー。一般に、管理者は最初に信頼できるユーザーのログインを作成してから、ユーザーの信頼および能力レベルに合致した管理者権利を割り当てます。その後、これらのユーザーはシステムの構成および保守を支援します。*特権ユーザー*とも呼ばれます。

スーパーユーザーモデル
コンピュータシステムにおける典型的な UNIX セキュリティーモデル。スーパーユーザーモデルでは、管理者は絶対的なシステム制御権を持ちます。一般に、マシン管理のために 1 人のユーザーがスーパーユーザー (`root`) になり、すべての管理作業を行える状態となります。

スキャンエンジン
既知のウイルスがないかどうかファイルを検査する、外部ホスト上に存在するサードパーティーのアプリケーション。

スレーブ KDC
マスター KDC のコピー。マスター KDC のほとんどの機能を実行できます。各レルムには通常、いくつかのスレーブ KDC (と 1 つのマスター KDC) を配置します。KDC、マスター KDCも参照してください。

制限セット
プロセスとその子プロセスでどの特権が利用できるかを示す上限。

整合性
ユーザー認証に加えて、暗号チェックサムを使用して、転送されたデータの有効性を提供するセキュリティーサービス。認証、プライバシも参照してください。

責務分離
最少特権の概念の一部。責務分離により、1 人のユーザーが、トランザクションを完了するためのすべての操作を実行または承認することが回避されます。たとえば、RBAC では、セキュリティーオーバーライドの割り当てからログインユーザーの作成を分離できます。1 つの役割がユーザーを作成します。個別の役割により、権利プロファイル、役割、特権などのセキュリティー属性を既存のユーザーに割り当てることができます。

セキュリティーサービス
サービスを参照してください。

セキュリティー属性
セキュリティーポリシーをオーバーライドし、スーパーユーザー以外のユーザーによって実行されても成功する管理コマンドを有効にします。スーパーユーザーモデルでは、`setuid root` プログラムと `setgid` プログラムがセキュリティー属性です。これらの属性がコマンドで指定されると、そのコマンドがどのようなユーザーによって実行されているかにかかわらず、コマンドは正常に処理されます。特権モデルでは、セキュリティー属性として `setuid root` プログラムがカーネル特権およびその他の権利によって置き換えられます。特権モデルは、スーパーユーザーモデルと互換性があります。このため、特権モデルは `setuid` プログラムと `setgid` プログラムをセキュリティー属性として認識します。

セキュリティーフレーバ
フレーバを参照してください。

セキュリティーポリシー ポリシーを参照してください。

セキュリティーメカニズム メカニズムを参照してください。

セッション鍵 認証サービスまたはチケット認可サービスによって生成される鍵。セッション鍵は、クライアントとサービス間のトランザクションのセキュリティーを保護するために生成されます。セッション鍵の有効期限は、単一のログインセッションに制限されます。鍵も参照してください。

ソフトウェアプロバイダ Oracle Solaris の暗号化フレームワーク機能では、暗号化サービスを提供するカーネルソフトウェアモジュールまたは PKCS #11 ライブラリ。プロバイダも参照してください。

ダイジェスト メッセージダイジェストを参照してください。

単一システムイメージ 単一システムイメージは、同じネームサービスを使用する一連の検査対象システムを記述するために、Oracle Solaris 監査で使用されます。これらのシステムは監査レコードを中央の監査サーバーに送信しますが、その監査サーバー上では、それらのレコードがまるで 1 つのシステムからやってきたかのように、レコードの比較を行えます。

遅延チケット 遅延チケットは、作成されても指定された時点まで有効になりません。このようなチケットは、夜遅く実行するバッチジョブなどのために効果的です。そのチケットは盗まれても、バッチジョブが実行されるまで使用できないためです。遅延チケットは、無効チケットとして発行され、a) 開始時間を過ぎて、b) クライアントが KDC による検査を要求したときに有効になります。遅延チケットは通常、チケット認可チケットの有効期限まで有効です。ただし、その遅延チケットが「更新可能」と指定されている場合、その有効期限は通常、チケット認可チケットの有効期限に設定されます。無効チケット、更新可能チケットも参照してください。

チケット ユーザーの識別情報をサーバーやサービスに安全に渡すために使用される情報パケット。チケットは、単一クライアントと特定サーバー上の特定サービスだけに有効です。チケットには、サービスの主体名、ユーザーの主体名、ユーザーのホストの IP アドレス、タイムスタンプ、チケットの有効期限を定義する値などが含まれます。チケットは、クライアントとサービスによって使用されるランダムセッション鍵を使用して作成されます。チケットは、作成されてから有効期限まで再使用できます。チケットは、最新のオーセンティケータとともに提示されなければ、クライアントの認証に使用することができません。オーセンティケータ、資格、サービス、セッション鍵も参照してください。

チケットファイル 資格キャッシュを参照してください。

デバイスの割り当て ユーザーレベルでのデバイス保護。デバイス割り当ては、一度に 1 人のユーザーだけが使用できるようにデバイスを設定する作業です。デバイスデータは、デバイスが再使用される前に消去されます。誰にデバイス割り当てを許可するかは、承認を使用して制限できます。

デバイスポリシー カーネルレベルでのデバイス保護。デバイスポリシーは、2 つの特権セットとしてデバイスに実装されます。この 1 つはデバイスに対する読み取り権を制御し、もう 1 つはデバイスに対する書き込み権を制御します。ポリシーも参照してください。

転送可能チケット
チケットの 1 つ。クライアントがリモートホスト上のチケットを要求するときに使用できます。このチケットを使用すれば、リモートホスト上で完全な認証プロセスを実行する必要がありません。たとえば、ユーザー `david` がユーザー `jennifer` のマシンで転送可能チケットを取得した場合、`david` は自分のマシンにログインできます (新しいチケットを取得する必要はない、自分自身を認証できる)。プロキシ可能チケットも参照してください。

同期監査イベント
監査イベントの大半を占めます。これらのイベントは、システムのプロセスに関連付けられています。失敗したログインなど、あるプロセスに関連付けられた、ユーザーに起因しないイベントは、同期イベントです。

特権
1. 一般に、コンピュータシステム上で通常のユーザーの能力を超える操作を実行する能力または機能。スーパーユーザー特権は、スーパーユーザーに付与されているすべての権利です。特権ユーザーまたは特権アプリケーションは、追加の権利が付与されているユーザーまたはアプリケーションです。

2. Oracle Solaris システムにおいてプロセスに対する個々の権利。特権を使用すると、`root` を使用するよりもきめ細かなプロセス制御が可能です。特権の定義と適用はカーネルで行われます。特権は、*プロセス特権*や*カーネル特権*とも呼ばれます。特権の詳細は、`privileges(5)` のマニュアルページを参照してください。

特権エスカレーション
権利 (デフォルトをオーバーライドして許可する権利を含む) を割り当てられたリソース範囲の外部のリソースへのアクセス権を取得すること。その結果、プロセスは未承認の操作を実行できます。

特権セット
一連の特権。各プロセスには、プロセスが特定の特権を使用できるかどうかを判断する 4 セットの特権があります。詳細は、制限セット、有効セット、許可されたセット、および 継承可能セットを参照してください。

基本セットも、ユーザーのログインプロセスに割り当てられる特権セットです。

特権付きアプリケーション
システム制御をオーバーライドできるアプリケーション。このようなアプリケーションは、セキュリティー属性 (特定の UID、GID、承認、特権など) をチェックします。

特権モデル
コンピュータシステムにおいてスーパーユーザーモデルより厳密なセキュリティーモデル。特権モデルでは、プロセスの実行に特権が必要です。システムの管理は、管理者が各自のプロセスで与えられている特権に基づいて複数の個別部分に分割できます。特権は、管理者のログインプロセスに割り当てることも、特定のコマンドだけで有効なように割り当てることも可能です。

特権ユーザー
コンピュータシステム上で通常ユーザーの権利を超えた権利が割り当てられているユーザー。信頼できるユーザーも参照してください。

特権を認識する
自身のコードでの特権の使用を有効および無効にするプログラム、スクリプト、およびコマンド。本稼動環境では、たとえば、プログラムのユーザーに、その特権をプログラムに追加する権利プロファイルの使用を要求することによって、有効になった特権をプロセスに提供する必要があります。特権の詳細は、`privileges(5)` のマニュアルページを参照してください。

認証
特定の主体の識別情報を検証するプロセス。

認証済み権利プロファイル 権利プロファイルの 1 つ。割り当てられたユーザーまたは役割は、プロファイルから操作を実行する前に、パスワードを入力する必要があります。この動作は、sudo の動作に似ています。パスワードが有効である時間の長さは構成可能です。

ネームサービススコープ 特定の役割が操作を許可されている適用範囲。つまり、NIS LDAP などの指定されたネームサービスからサービスを受ける個々のホストまたはすべてのホスト。

ネットワークアプリケーションサーバー ネットワークアプリケーションを提供するサーバー (ftp など)。レルムは、複数のネットワークアプリケーションサーバーで構成されます。

ネットワークポリシー ネットワークトラフィックを保護するためにネットワークユーティリティーで行われる設定。ネットワークセキュリティーについては、『Oracle Solaris 11.2 でのネットワークのセキュリティー保護 』を参照してください。

ハードウェアプロバイダ Oracle Solaris の暗号化フレームワーク機能では、デバイスドライバとそのハードウェアアクセラレータを指します。ハードウェアプロバイダを使用すると、コンピュータシステムから負荷の高い暗号化処理を解放され、その分 CPU リソースをほかの用途に充てることができます。プロバイダも参照してください。

パスフレーズ 非公開鍵がパスフレーズユーザーによって作成されたことを検証するために使用されるフレーズ。望ましいパスフレーズは、10 - 30 文字の長さで英数字が混在しており、単純な文や名前を避けたものです。通信の暗号化と復号化を行う非公開鍵の使用を認証するため、パスフレーズの入力を求めるメッセージが表示されます。

パスワードポリシー パスワードの生成に使用できる暗号化アルゴリズム。パスワードをどれぐらいの頻度で変更すべきか、パスワードの試行を何回まで認めるかといったセキュリティー上の考慮事項など、パスワードに関連した一般的な事柄を指すこともあります。セキュリティーポリシーにはパスワードが必要です。パスワードポリシーでは、AES アルゴリズムを使用してパスワードを暗号化することを要求したり、パスワードの強度に関連したそれ以上の要件を設定したりすることもできます。

非公開鍵 各ユーザー (主体) に与えられ、主体のユーザーと KDC だけが知っている鍵。ユーザー主体の場合、鍵はユーザーのパスワードに基づいています。鍵も参照してください。

非公開鍵の暗号化 非公開鍵の暗号化では、送信者と受信者は同じ暗号化鍵を使用します。公開鍵の暗号化も参照してください。

非同期監査イベント 非同期イベントは、システムイベントの内の少数です。これらのイベントは、プロセスに関連付けられていないため、ブロックした後に起動できるプロセスはありません。たとえば、システムの初期ブートや PROM の開始および終了のイベントは、非同期イベントです。

秘密鍵 非公開鍵を参照してください。

プライバシ セキュリティーサービスの 1 つ。送信データを送信前に暗号化します。プライバシには、データの整合性とユーザー認証も含まれます。認証、整合性、サービスも参照してください。

プライマリ 主体名の最初の部分。インスタンス、主体名、レルムも参照してください。

フレーバ 従来は、「セキュリティーフレーバ」と「認証フレーバ」は、認証のタイプ (AUTH_UNIX、AUTH_DES、AUTH_KERB) を指すフレーバとして、同じ意味を持っていま

した。RPCSEC_GSS もセキュリティーフレーバですが、これは認証に加えて、整合性とプライバシのサービスも提供します。

プロキシ可能チケット クライアントに代わってクライアント操作を行うためにサービスによって使用されるチケット。このことを、サービスがクライアントのプロキシとして動作するといいます。サービスは、チケットを使用して、クライアントの識別情報を所有できます。このサービスは、プロキシ可能チケットを使用して、ほかのサービスへのサービスチケットを取得できますが、チケット認可チケットは取得できません。転送可能チケットと異なり、プロキシ可能チケットは単一の操作に対してだけ有効です。転送可能チケットも参照してください。

プロバイダ Oracle Solaris の暗号化フレームワーク機能では、コンシューマに提供される暗号化サービス。プロバイダには、PKCS #11 ライブラリ、カーネル暗号化モジュール、ハードウェアアクセラレータなどがあります。プロバイダは暗号化フレームワークに結合 (プラグイン) されるため、*プラグイン*とも呼ばれます。コンシューマの例は、コンシューマを参照してください。

プロファイルシェル 権利の管理で、役割 (またはユーザー) がコマンド行から、その役割の権利プロファイルに割り当てられた任意の特権付きアプリケーションを実行できるようにするシェル。プロファイルシェルのバージョンは、システム上で使用可能なシェルのバージョン (`bash` の `pfbash` バージョンなど) に対応します。

ホスト ネットワークを通じてアクセス可能なシステム。

ホスト主体 サービス主体のインスタンスの 1 つ (プライマリ名は `host`)。さまざまなネットワークサービス (`ftp`、`rcp`、`rlogin` など) を提供するために設定します。`host/central.example.com@EXAMPLE.COM` はホスト主体の例です。サーバー主体も参照してください。

ポリシー 一般には、意思やアクションに影響を与えたり、これらを決定したりする計画や手続き。コンピュータシステムでは、多くの場合セキュリティーポリシーを指します。実際のサイトのセキュリティーポリシーは、処理される情報の重要度や未承認アクセスから情報を保護する手段を定義する規則セットです。たとえば、セキュリティーポリシーが、システムの監査、使用するデバイスの割り当て、6 週ごとのパスワード変更を要求する場合があります。

Oracle Solaris OS の特定の領域におけるポリシーの実装については、監査ポリシー、暗号化フレームワークにおけるポリシー、デバイスポリシー、Kerberos ポリシー、パスワードポリシー、および 権利ポリシーを参照してください。

マスター KDC 各レルムのメイン KDC。Kerberos 管理サーバー `kadmind` と、認証とチケット認可デーモン `krb5kdc` で構成されます。レルムごとに、1 つ以上のマスター KDC を割り当てる必要があります。また、クライアントに認証サービスを提供する複製 (スレーブ) KDC を任意の数だけ割り当てることができます。

無効チケット まだ使用可能になっていない遅延チケット。無効チケットは、有効になるまでアプリケーションサーバーから拒否されます。これを有効にするには、開始時期が過ぎたあと、TGS 要求で `VALIDATE` フラグをオンにしてクライアントがこのチケットを KDC に提示する必要があります。遅延チケットも参照してください。

メカニズム 1. データの認証や機密性を実現するための暗号化技術を指定するソフトウェアパッケージ。たとえば、Kerberos V5、Diffie-Hellman 公開鍵など。

2. Oracle Solaris の暗号化フレームワーク機能では、特定の目的のためのアルゴリズムの実装。たとえば、認証に適用される DES メカニズム (CKM_DES_MAC など) は、暗号化に適用されるメカニズム (CKM_DES_CBC_PAD) とは別です。

メッセージダイジェスト メッセージダイジェストは、メッセージから計算されるハッシュ値です。ハッシュ値によってメッセージはほぼ一意に識別されます。ダイジェストは、ファイルの整合性を検証するのに便利です。

メッセージ認証コード (MAC) データの整合性を保証し、データの出所を明らかにするコード。MAC は盗聴行為には対応できません。

役割 特権付きアプリケーションを実行するための特別な ID。割り当てられたユーザーだけが引き受けられます。

有効セット プロセスにおいて現在有効である一連の特権。

ユーザー主体 特定のユーザーに属する主体。ユーザー主体のプライマリ名はユーザー名であり、その省略可能なインスタンスは対応する資格の使用目的を説明するために使われる名前です (`jdoe`、`jdoe/admin` など)。「ユーザーインスタンス」とも呼ばれます。サービス主体も参照してください。

ユーザーに起因しない監査イベント 開始した人を特定できない監査イベント。AUE_BOOT イベントなど。

レルム 1. 1 つの Kerberos データベースといくつかの鍵配布センター (KDC) を配置した論理ネットワーク。

2. 主体名の 3 番目の部分。主体名が `jdoe/admin@CORP.EXAMPLE.COM` の場合、レルムは `CORP.EXAMPLE.COM` です。主体名も参照してください。

admin 主体 *username*`/admin` という形式 (`jdoe/admin` など) の名前を持つユーザー主体。通常のユーザー主体より多くの特権 (ポリシーの変更など) を持つことができます。主体名とユーザー主体も参照してください。

AES Advanced Encryption Standard。対称 128 ビットブロックのデータ暗号技術。2000 年の 10 月、米国政府は暗号化標準としてこのアルゴリズムの Rijndael 方式を採用しました。ユーザー主体の暗号化に代わる米国政府の標準として、AES が採用されています。

Blowfish 32 ビットから 448 ビットまでの可変長鍵の対称ブロックの暗号化アルゴリズム。その作成者である Bruce Schneier 氏は、鍵を頻繁に変更しないアプリケーションに効果的であると述べています。

DES Data Encryption Standard。1975 年に開発され、1981 年に ANSI X.3.92 として ANSI で標準化された対称鍵の暗号化方式。DES では 56 ビットの鍵を使用します。

Diffie-Hellman プロトコル
公開鍵暗号化としても知られています。1976 年に Diffie 氏と Hellman 氏が開発した非対称暗号鍵協定プロトコルです。このプロトコルを使用すると、セキュアでない伝達手段で、事前の秘密情報がなくても 2 人のユーザーが秘密鍵を交換できます。Diffie-Hellman は Kerberos で使用されます。

DSA
デジタル署名アルゴリズム。512 ビットから 4096 ビットまでの可変長鍵の公開鍵アルゴリズム。米国政府標準である DSS は最大 1024 ビットです。DSA は入力に SHA1 を使用します。

ECDSA
Elliptic Curve Digital Signature Algorithm。楕円曲線数学に基づく公開鍵アルゴリズム。ECDSA 鍵サイズは、同じ長さの署名の生成に必要な DSA 公開鍵のサイズより大幅に小さくなります。

FQDN
完全指定形式のドメイン名。`central.example.com` など (単なる `denver` は FQDN ではない)。

GSS-API
Generic Security Service Application Programming Interface の略。さまざまなモジュールセキュリティーサービス (Kerberos サービスなど) をサポートするネットワーク層。GSS-API は、セキュリティー認証、整合性、およびプライバシサービスを提供します。認証、整合性、プライバシも参照してください。

KDC
鍵配布センター (Key Distribution Center)。次の 3 つの Kerberos V5 要素で構成されるマシン。

- 主体と鍵データベース
- 認証サービス
- チケット許可サービス

レルムごとに、1 つのマスター KDC と、1 つ以上のスレーブ KDC を配置する必要があります。

Kerberos
認証サービス、Kerberos サービスが使用するプロトコル、または Kerberos サービスの実装に使用されるコード。

Oracle Solaris の Kerberos は、Kerberos V5 認証にほぼ準拠して実装されています。

「Kerberos」と「Kerberos V5」は技術的には異なりますが、Kerberos のドキュメントでは多くの場合、同じ意味で使用されます。

Kerberos (または Cerberus) は、ギリシャ神話において、ハデスの門を警護する 3 つの頭を持つどう猛な番犬のことです。

Kerberos ポリシー
Kerberos サービスでのパスワードの使用方法を管理する一連の規則。ポリシーは、主体のアクセスやチケットのパラメータ (有効期限など) を制限できます。

kvno
鍵バージョン番号。特定の鍵に対して、生成順に付けられたシーケンス番号。もっとも大きい kvno が、最新の鍵を示します。

MAC 1. メッセージ認証コード (MAC)を参照してください。

2.「ラベル付け」とも呼ばれます。政府のセキュリティー用語規定では、MAC は「Mandatory Access Control」の略です。「Top Secret」や「Confidential」というラベルは MAC の例です。MAC と対照をなすものに DAC (Discretionary Access Control) があります。UNIX アクセス権は DAC の 1 例です。

3. ハードウェアにおいては、LAN における一意のシステムアドレス。システムが Ethernet 上に存在する場合は、Ethernet アドレスが MAC に相当します。

MD5 デジタル署名などのメッセージ認証に使用する繰り返し暗号化のハッシュ関数。1991 年に Rivest 氏によって開発されました。その使用は非推奨です。

NTP Network Time Protocol (NTP)。デラウェア大学で開発されたソフトウエア。ネットワーク環境で、正確な時間またはネットワーククロックの同期化を管理します。NTP を使用して、Kerberos 環境のクロックスキューを管理できます。「クロックスキュー」も参照してください。

PAM プラグイン可能認証モジュール (Pluggable Authentication Module)。複数の認証メカニズムを使用できるフレームワーク。認証メカニズムを使用するサービスはコンパイルし直す必要がありません。PAM は、ログイン時に Kerberos セッションを初期化できます。

QOP 保護の品質。整合性サービスまたはプライバシサービスで使用する暗号化アルゴリズムを選択するときに使用されるパラメータの 1 つ。

RBAC Oracle Solaris のユーザー権利管理機能である、役割に基づくアクセス制御。権利を参照してください。

RBAC ポリシー 権利ポリシーを参照してください。

RSA デジタル署名と公開鍵暗号化システムを取得するための方法。その開発者である Rivest 氏、Shamir 氏、Adleman 氏によって 1978 年に最初に公開されました。

SEAM Solaris システム上の Kerberos の初期バージョンに対応する製品名。この製品は、マサチューセッツ工科大学 (MIT) で開発された Kerberos V5 テクノロジに基づいています。SEAM は、現在 Kerberos サービスと呼ばれています。引き続き、MIT バージョンとはわずかに異なります。

Secure Shell セキュリティー保護されていないネットワークを通して、セキュアなリモートログインおよびその他のセキュアなネットワークサービスを使用するための特別なプロトコル。

SHA1 セキュアなハッシュアルゴリズム。メッセージ要約を作成するために 2^{64} 文字以下の長さを入力するときに操作します。SHA1 アルゴリズムは DSA に入力されます。

stash ファイル stash ファイルには、KDC のマスター鍵を暗号化したコピーが含まれます。サーバーがリブートされると、このマスター鍵を使用して KDC が自動的に認証されてから、`kadmind` プロセスと `krb5kdc` プロセスがブートされます。stash ファイルにはマスター鍵が入っているため、このファ

イルやこのファイルのバックアップは安全な場所に保管する必要があります。暗号が破られると、この鍵を使用して KDC データベースのアクセスや変更が可能になります。

TGS チケット許可サービス。KDC のコンポーネントの 1 つ。チケットを発行します。

TGT チケット認可チケット (Ticket-Granting Ticket)。KDC によって発行されるチケット。クライアントは、このチケットを使用して、ほかのサービスのチケットを要求することができます。

索引

ま

や

ら

A

C

D

E

F

G

I

K

T

V

X