在 Oracle® Solaris 11.2 中管理加密和证书

文件号码 E53961-02
2014 年 9 月

ORACLE

目录

使用本文档

《在 *Oracle*® *Solaris 11.2* 中管理加密和证书》介绍了如何管理和使用加密，以及如何创建和管理私钥/公钥证书。

- **概述** – 介绍与加密框架和密钥管理框架有关的概念以及使用这些技术保护文件安全的任务。
- **目标读者** – 必须在企业中实现安全的系统管理员。
- **必备知识** – 熟悉安全概念和术语。

产品文档库

有关本产品的最新信息和已知问题均包含在文档库中，网址为：http://www.oracle.com/pls/topic/lookup?ctx=E56344。

获得 Oracle 支持

Oracle 客户可通过 My Oracle Support 获得电子支持。有关信息，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info；如果您听力受损，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs。

反馈

可以在 http://www.oracle.com/goto/docfeedback 上提供有关本文档的反馈。

♦ ♦ ♦ 第 1 章

1 加密框架

本章介绍了 Oracle Solaris 的加密框架功能，并涵盖了以下主题：

要管理和使用加密框架，请参见第 3 章 加密框架。

Oracle Solaris 11.2 的新增加密功能

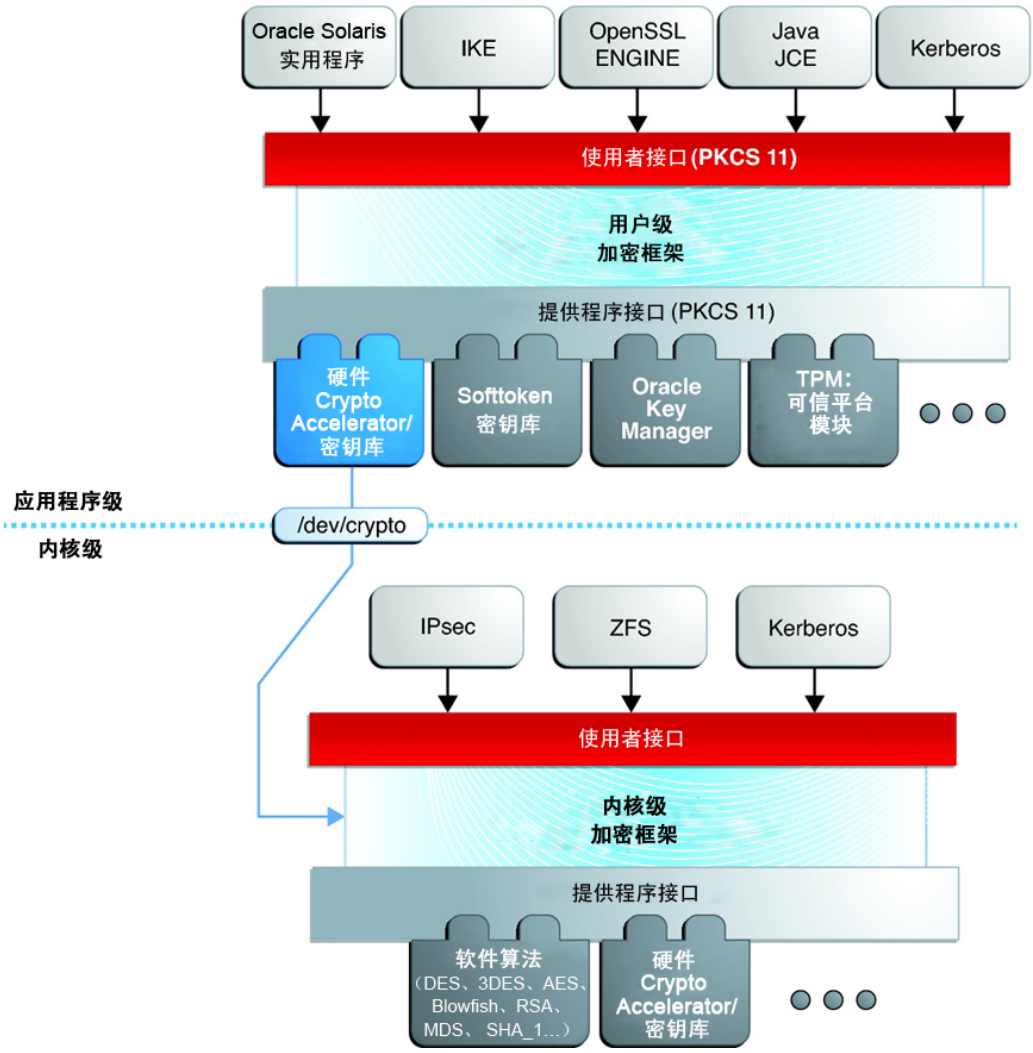
本部分重点介绍此发行版中针对现有客户的有关新的加密功能的信息。

- Oracle Solaris 支持 OpenSSL 的两个版本，包括支持 FIPS 和不支持 FIPS 的版本。
- 在经过加密优化的 SPARC T4 系统上，加密指令可直接在硬件中使用，从而加快加密运算的速度。
- 加密框架支持 Camellia，这是一种类似于 AES 的 128 位块加密算法，主要在日本市场中使用。

加密框架简介

加密框架提供了一个包含算法和 PKCS #11 库的公共存储区来处理加密要求。PKCS #11 库依据 RSA Security Inc. 的 PKCS #11 加密令牌接口 (Cryptographic Token Interface, Cryptoki) 标准实现。

图 1-1 加密框架级别



在内核级别，该框架目前可处理 ZFS、Kerberos、IPsec 以及硬件的加密要求。用户级的使用者包括 OpenSSL 引擎、Java 加密扩展 (Java Cryptographic Extensions, JCE)、`libsasl` 和 Internet 密钥协议 (Internet Key Protocol, IKE)。内核 SSL (`kssl`) 代理使用加密框架。有关更多信息，请参阅《在 Oracle Solaris 11.2 中确保网络安全 》中的“SSL 内核代理加密 Web 服务器通信”和 `ksslcfg(1M)` 手册页。

美国出口法要求应获得许可才能使用开放式加密接口。加密框架通过要求对内核加密服务提供者和 PKCS #11 加密服务提供者进行签名来满足现行法律规定。有关进一步讨论，请参阅“加密框架中的用户级命令” [11]中有关 `elfsign` 命令的信息。

该框架允许加密服务提供者让 Oracle Solaris 中的许多使用者使用其服务。提供者又称为插件。该框架支持以下三种类型的插件：

- 用户级插件 – 使用 PKCS #11 库（例如 `/var/user/$USER/pkcs11_softtoken.so.1`）提供服务的共享目标。
- 内核级插件 – 在软件中提供加密算法（如 AES）实现的内核模块。

  框架中的许多算法针对 x86（使用 SSE2 指令集）和 SPARC 硬件进行了优化。有关 T 系列优化，请参见“加密框架和 SPARC T 系列服务器” [15]。
- 硬件插件 – 设备驱动程序及其关联的硬件加速器。Niagara 芯片和 Oracle 的 `ncp` 和 `n2cp` 设备驱动程序就是一个示例。硬件加速器承担了操作系统的高开销加密功能负载。Sun Crypto Accelerator 6000 板就是一个示例。

该框架为用户级提供者实现了标准接口 PKCS #11 v2.20 修订 3 库。第三方应用程序可以使用该库来访问提供者。第三方也可以向框架中添加签名库、签名内核算法模块和签名设备驱动程序。映像包管理系统 (Image Packaging System, IPS) 安装第三方软件时将添加这些插件。有关该框架的主要组件的示意图，请参见图 1-1 “加密框架级别”。

加密框架中的概念

请注意以下概念介绍和对应的示例，它们在处理加密框架时会很有用。

- **算法** – 加密算法是已确立的用于对输入执行加密或散列操作的递归计算过程。加密算法可以是对称算法，也可是非对称算法。对称算法对于加密和解密使用同一个密钥。非对称算法用于公钥加密中，要求使用两个不同的密钥。散列函数也是算法。

  算法示例包括：

  - 对称算法，如 AES 和 ECC
  - 非对称算法，例如 Diffie-Hellman 和 RSA
  - 散列函数，如 SHA256
- **使用者** – 提供者提供的加密服务的用户。使用者可以是应用程序、最终用户或内核操作。

  使用者示例包括：

  - 应用程序，如 IKE
  - 最终用户，如运行 `encrypt` 命令的一般用户。
  - 内核操作，例如 IPsec
- **密钥库** – 在加密框架中，密钥库是令牌对象的持久性存储，通常可与**令牌**互换使用。有关保留的密钥库的信息，请参见此定义列表中的 **Metaslot**。
- **机制** – 针对特定目的的算法模式的应用。

例如，应用于验证的 DES 机制（如 CKM_DES_MAC）与应用于加密的 DES 机制（如 CKM_DES_CBC_PAD）不同。

- **Metaslot** – 提供框架中装入的其他插槽的功能组合的单个插槽。Metaslot 简化了处理可通过框架使用的提供者全部功能的工作。使用 metaslot 的应用程序请求某个操作时，metaslot 将确定应执行该操作的实际插槽。可以配置 metaslot 功能，但这不是必须的。缺省情况下，metaslot 处于打开状态。有关更多信息，请参见 `cryptoadm(1M)` 手册页。

  metaslot 没有自己的密钥库。相反，metaslot 会保留使用来自加密框架中的实际插槽之一的密钥库。缺省情况下，metaslot 保留 `Sun Crypto Softtoken` 密钥库。metaslot 使用的密钥库不会显示为可用插槽之一。

  通过设置环境变量 `${METASLOT_OBJECTSTORE_SLOT}` 和 `${METASLOT_OBJECTSTORE_TOKEN}`，或者通过运行 `cryptoadm` 命令，用户可以为 metaslot 指定备用密钥库。有关更多信息，请参见 `libpkcs11(3LIB)`、`pkcs11_softtoken (5)` 和 `cryptoadm(1M)` 手册页。
- **模式** – 加密算法的版本。例如，模式 CBC（Cipher Block Chaining，密码块链接）与模式 ECB（Electronic Code Book，电子源码书）是不同的模式。AES 算法有两种模式：CKM_AES_ECB 和 CKM_AES_CBC。
- **策略** – 由管理员做出的要使哪些机制可用的选择。缺省情况下，所有提供者和所有机制都可用。启用或禁用任何机制是对策略的应用。有关设置和应用策略的示例，请参见“管理加密框架” [30]。
- **提供者** – 使用者使用的加密服务。提供者插入到框架中，因此也称为插件。

  提供者示例包括：

  - PKCS #11 库，例如 `/var/user/$USER/pkcs11_softtoken.so`
  - 加密算法模块，例如 `aes` 和 `arcfour`
  - 设备驱动程序及其关联的硬件加速器，例如 Sun Crypto Accelerator 6000 的 `mca` 驱动程序
- **插槽** – 到一个或多个加密设备的接口。对应于物理读取器或其他设备接口的每个插槽可能包含令牌。令牌提供框架中加密设备的逻辑视图。
- **令牌** – 在插槽中，令牌提供框架中加密设备的逻辑视图。

加密框架命令和插件

框架为管理员、用户和开发提供者的开发者提供命令。

- 管理命令 – `cryptoadm` 命令提供了 `list` 子命令，用于列出可用提供者及其功能。一般用户可以运行 `cryptoadm list` 和 `cryptoadm --help` 命令。

  所有其他 `cryptoadm` 子命令要求您承担拥有 Crypto Management（加密管理）权限配置文件的角色或者成为超级用户。子命令（如 `disable`、`install` 和 `uninstall`）可用于管理框架。有关更多信息，请参见 `cryptoadm(1M)` 手册页。

svcadm 命令用于管理 kcfd 守护进程和在内核中刷新加密策略。有关更多信息，请参见 svcadm(1M) 手册页。

- 用户级命令 – digest 和 mac 命令可提供文件完整性服务。encrypt 和 decrypt 命令，用于保护文件不被窃听。有关使用这些命令的信息，请参见表 3-1 “使用加密框架保护文件（任务列表）”。

加密框架中的管理命令

cryptoadm 命令可管理正在运行的加密框架。该命令包括在 Crypto Management（加密管理）权限配置文件中。可以将此配置文件指定给角色，以实现加密框架的安全管理。使用 cryptoadm 命令可执行以下操作：

- 禁用或启用提供者机制
- 禁用或启用 metaslot

svcadm 命令用于启用、刷新和禁用加密服务守护进程 kcfd。此命令是 Oracle Solaris 的服务管理工具 (Service Management Facility, SMF) 的一部分。svc:/system/cryptosvcs 是加密框架的服务实例。有关更多信息，请参见 smf(5) 和 svcadm(1M) 手册页。

加密框架中的用户级命令

加密框架提供了用户级命令，用于检查文件完整性、加密和解密文件。

- digest 命令 – 计算一个或多个文件或 stdin 的 message digest（消息摘要）。摘要对检验文件的完整性非常有用。摘要功能的示例有 SHA1 和 MD5。
- mac 命令 – 计算一个或多个文件或 stdin 的 message authentication code, MAC（消息验证代码）。MAC 可将数据与通过验证的消息相关联。MAC 使接收者可以验证消息是否来自发送者，以及消息是否被篡改。使用 sha1_mac 和 md5_hmac 机制可以计算 MAC。
- encrypt 命令 – 使用对称密码算法加密文件或 stdin。encrypt -l 命令可以列出可用的算法。用户级库下列出的机制可用于 encrypt 命令。框架提供了 AES、DES、3DES（Triple-DES，三重 DES）和 ARCFOUR 机制来实现用户加密。
- decrypt 命令 – 解密使用 encrypt 命令加密的文件或 stdin。decrypt 命令使用的密钥和机制与用于加密原始文件的密钥和机制相同。
- elfsign 命令 – 提供对要用于加密框架的提供者进行签名的方法。通常，此命令由提供者的开发者运行。elfsign 命令包含一些用于申请证书、对二进制文件进行签名以及验证二进制文件上签名的子命令。加密框架无法使用未签名的二进制文件。具有可验证已签名二进制文件的提供者可以使用框架。

加密框架的插件

第三方可以将其提供者插入到加密框架中。第三方提供者可以是以下对象之一：

- PKCS #11 共享库
- 可装入内核软件模块，如加密算法、MAC 函数或摘要功能
- 硬件加速器的内核设备驱动程序

必须使用 Oracle 提供的证书对提供者的目标文件进行签名。证书申请基于第三方选择的私钥，以及 Oracle 提供的证书。证书申请将会发送到 Oracle，Oracle 注册第三方然后颁发该证书。之后第三方使用 Oracle 提供的证书对其提供者目标文件进行签名。

可装入内核软件模块和硬件加速器的内核设备驱动程序也必须在内核中注册。注册通过加密框架 SPI（service provider interface，服务提供者接口）进行。

加密服务和区域

全局区域和每个非全局区域都有自己的 `/system/cryptosvc` 服务。在全局区域中启用或刷新加密服务时，`kcfd` 守护进程会在该全局区域中启动，并且设置此全局区域的用户级策略和系统的内核策略。在非全局区域中启用或刷新服务时，`kcfd` 守护进程将在该区域中启动，并且设置该区域的用户级策略。内核策略由全局区域设置。

有关区域的更多信息，请参见《Oracle Solaris Zones 介绍 》。有关使用 SMF 管理持久性应用程序的更多信息，请参见《在 Oracle Solaris 11.2 中管理系统服务 》中的第 1 章 “服务管理工具简介”和 `smf(5)` 手册页。

加密框架和 FIPS-140

FIPS 140 是针对加密模块的美国政府计算机安全标准。Oracle Solaris 系统提供两个 FIPS 140-2 级别 1 认可的加密算法提供者。

这些提供者是：

- Oracle Solaris 的加密框架提供两个 FIPS 140 认可的模块。用户级模块为在用户空间中运行的应用程序提供加密。内核模块为内核级别的进程提供加密。
- OpenSSL 对象模块为 SSH 和 Web 应用程序提供 FIPS 140 认可的加密。

请注意以下关键事项：

- 因为 FIPS 140-2 提供者模块占用大量 CPU，所以缺省情况下不启用它们。作为系统管理员，您负责在 FIPS 140 模式下启用提供者并配置使用 FIPS 认可的算法的应用程序。

- 如果有严格的要求，只能使用 FIPS 140-2 验证的加密，则必须运行 Oracle Solaris11.1 SRU5.5 发行版或 Oracle Solaris11.1 SRU3 发行版。Oracle 已在这两个特定发行版中针对 Solaris 加密框架完成了 FIPS 140-2 验证。Oracle Solaris11.2 基于此验证的基础而构建并在性能、功能和可靠性方面进行了软件改进。应当尽可能在 FIPS 140-2 模式下配置 Oracle Solaris11.2 以利用这些改进。

有关信息，请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》。本文章包含以下主题：

- Oracle Solaris 中的 FIPS 140-2 级别 1 加密概述
- 启用 FIPS 140 提供者
- 启用 FIPS 140 使用者
- 在 FIPS 140 模式下启用两个应用程序的示例
- FIPS 140 认可的算法和证书参考

提供了以下附加信息：

- 如何切换到支持 FIPS 140 的 OpenSSL 实现方式 [13]
- “创建启用了 FIPS 140 的引导环境 ” [38]

Oracle Solaris 中的 OpenSSL 支持

Oracle Solaris 支持两种 OpenSSL 实现方式：

- 支持 FIPS 140 的 OpenSSL
- 不支持 FIPS 140 的 OpenSSL

两种实现方式都已升级，与 OpenSSL 项目的最新 OpenSSL 版本（OpenSSL 1.0.1）兼容。对于版本库，两种实现方式都与 API/ABI 兼容。

虽然操作系统中两种实现方式都有，但一次只有一种实现方式有效。要确定系统上哪种 OpenSSL 实现方式有效，可使用 `pkg mediator openssl` 命令。

▼ 如何切换到支持 FIPS 140 的 OpenSSL 实现方式

缺省情况下，不支持 FIPS 140 的 OpenSSL 实现方式在 Oracle Solaris 中有效。但是，您可以选择系统的安全设置，选择您需要的实现方式。

1. **成为管理员。**
2. **确保系统上两种实现方式都有。**

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

注意 - 要切换到的 OpenSSL 实现方式必须在系统中存在。否则，如果切换到系统中不存在的实现方式，系统可能会变得无法使用。

3. 切换到不同的 OpenSSL 实现方式。

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

其中 *implementation* 是 `default` 或 `fips-140`，*name* 是当前引导环境的新克隆的名称。该克隆将使指定的实现方式有效。

注 - 指定 *--be-name* 时，该命令创建当前引导环境的备份。重新引导时，系统将运行包含新实现方式的克隆的新引导环境。

有关 `pkg set-mediator` 命令的更多信息，请参见《在 Oracle Solaris 11.2 中添加和更新软件 》中的“更改首选应用程序”。

4. 重新引导系统。

5. （可选） 验证切换是否成功，以及您的首选 OpenSSL 实现方式是否有效。

```
# pkg mediator openssl
```

例 1-1 切换到支持 FIPS 140 的 OpenSSL 实现方式

此示例将系统的 OpenSSL 实现方式更改为支持 FIPS 140。

```
# 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

2 关于 SPARC T 系列系统和加密框架

本章介绍 SPARC T 系列服务器上的加密框架以及 Oracle Solaris 11 中增强加密函数性能的优化。

加密框架和 SPARC T 系列服务器

加密框架为 SPARC T 系列系统提供了加密机制，并针对这些服务器优化了某些机制。针对处于静止和活动状态的数据优化了三种加密机制：AES-CBC、AES-CFB128 和 ARCFOUR。DES 加密机制针对 OpenSSL 进行了优化，RSA 和 DSA 也通过优化任意精度算法 (bignum) 针对 OpenSSL 进行了优化。其他优化包括针对握手和活动中数据的小数据包性能。

在此发行版中提供了以下加密机制：

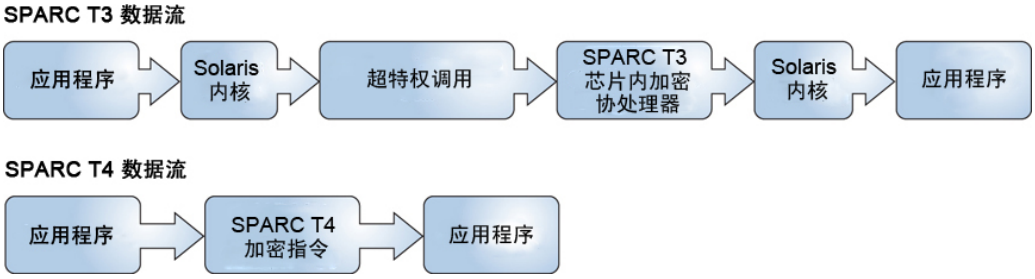
- AES-XTS – 用于静止数据
- SHA-224 – SHA2 机制
- AES-XCBC-MAC – 用于 IPsec

SPARC T-4 系统中的加密优化

从 SPARC T4 微处理器开始，可以在硬件中直接使用执行加密函数的新指令。这些指令是非特权指令。因此，任何程序都可以使用这些指令，无需任何内核环境、root 权限或其他特殊设置。加密直接在硬件上执行，而不使用大量的低级指令。因此，与以前采用有单独的加密处理单元的 SPARC 处理器的系统相比，加密运算速度更快。

以下比较显示了 SPARC T-3 系统与经过加密优化的 SPARC T-4 系统之间的数据流差异。

图 2-1 SPARC T 系统之间的数据流比较



下表提供了与特定的 Oracle Solaris 发行版相结合的 SPARC T 微处理器单元中加密函数的详细比较。

表 2-1 SPARC T 系列服务器上的加密性能

功能/软件使用者	T-3 和以前的系统	运行 Oracle Solaris 10 的 T-4 系统	运行 Oracle Solaris 11 的 T-4 系统
SSH	在 Solaris 10 5/09 及更高版本中自动启用。 在 `/etc/ssh/sshd_config` 中使用 `UseOpenSSLEngine` 子句禁用/启用。	需要修补程序 147707-01。 在 `/etc/ssh/sshd_config` 中使用 `UseOpenSSLEngine` 子句禁用/启用。	自动启用。 在 `/etc/ssh/sshd_config` 中使用 `UseOpenSSLEngine` 子句禁用/启用。
Java/JCE	自动启用。 在 `$JAVA_HOME/jre/lib/security/java.security` 中配置。	自动启用。 在 `$JAVA_HOME/jre/lib/security/java.security` 中配置。	自动启用。 在 `$JAVA_HOME/jre/lib/security/java.security` 中配置。
ZFS 加密	不可用。	不可用。	如果加密了数据集，将自动启用硬件加密。
IPsec	自动启用。	自动启用。	自动启用。
OpenSSL	使用 `-engine pkcs11`	需要修补程序 147707-01 使用 `-engine pkcs11`	自动使用 T4 优化。 （可选择使用 `-engine pkcs11`。）当前建议对 RSA/DSA 使用 `pkcs11`。
KSSL（Kernel SSL proxy，内核 SSL 代理）	自动启用。	自动启用。	自动启用。
Oracle TDE	不支持。	待定修补程序。	在 Oracle DB 11.2.0.3 和 ASO 中自动启用。
Apache SSL	使用 `SSLCryptoDevice pkcs11` 配置	使用 `SSLCryptoDevice pkcs11` 配置	使用 `SSLCryptoDevice pkcs11` 配置

功能/软件使用者	T-3 和以前的系统	运行 Oracle Solaris 10 的 T-4 系统	运行 Oracle Solaris 11 的 T-4 系统
逻辑域	将加密单元分配到域。	功能始终可用，不需要配置。	功能始终可用，不需要配置。

T4 加密指令包括以下指令：

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

  这些指令执行密钥扩展。它们将 128 位、192 位或 256 位用户提供的密钥扩展为在加密和解密过程中在内部使用的密钥排程。`aes_kexpand2` 指令仅用于 AES-256。另两种 `aes_kexpand` 指令用于全部三种密钥长度：AES-128、AES-192 和 AES-256。
- `aes_eround01`、`aes_eround23`、`aes_eround01_l`、`aes_eround_23_l`

  这些指令用于 AES 加密循环或变换。根据 FIPS 197 中的 AES 标准，所用的循环数量（例如 10、12 或 14）会因为 AES 密钥长度而有差异，因为使用较大的密钥大致上表示需要牺牲更多的计算性能来实现更强大的加密。
- `aes_dround01`、`aes_dround23`、`aes_dround01_l`、`aes_dround_23_l`

  这些指令以类似于加密的方式用于 AES 解密循环。
- DES/DES-3、Kasumi、Camellia、Montgomery 乘/开方（针对 RSA Bignum）和 CRC32c 校验和指令
- MD5、SHA1 和 SHA2 摘要指令

SPARC T4 硬件加密指令在运行 Oracle Solaris 11 的 SPARC T4 系统上可用，并通过系统的 T4 微处理器上的内置 t4 引擎自动使用。从 Oracle Solaris 11.2 开始，这些指令已嵌入到 OpenSSL 上游代码中。因此，在此发行版中，OpenSSL 1.0.1e 随附有一个使其可使用这些指令的修补程序。

有关 T4 指令的更多信息，请参考以下文章。

- “SPARC T4 OpenSSL 引擎” (https://blogs.oracle.com/DanX/entry/sparc_t4_openssl_engine)
- “如何知道是否正在使用 SPARC T4 加密？” (https://blogs.oracle.com/DanX/entry/how_to_tell_if_sparc)
- “T4 处理器和 Oracle Solaris 11 中激动人心的加密技术进展” (http://bubbva.blogspot.com/2011/11/exciting-crypto-advances-with-t4.html)
- “Solaris 11.1 中的 SPARC T4 摘要和加密优化” (https://blogs.oracle.com/DanX/entry/sparc_t4_digest_and_crypto)

确定系统是否支持 SPARC T4 优化

要确定是否正在使用 T4 优化，可使用 `isainfo` 命令。在输出中包括 `sparcv9` 和 `aes` 表示系统正在使用优化。

```
$ 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
```

确定系统的 OpenSSL 版本

要检查正在系统上运行的 OpenSSL 的版本，请键入 `openssl version`。输出以下类似内容：

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

验证您的系统是否有带 SPARC T4 优化的 OpenSSL

要确定您的系统是否支持带 SPARC T4 优化的 OpenSSL，请按如下方式检查 `libcrypto.so` 库：

```
# 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
```

如果该命令不生成任何输出，则您的系统不支持针对 OpenSSL 的 SPARC T4 优化。

♦ ♦ ♦ 第 3 章

3 加密框架

本章介绍了如何使用加密框架，并涵盖了以下主题：

使用加密框架保护文件

本节介绍如何生成对称密钥、如何创建校验和来检验文件完整性以及如何避免文件遭到窃听。本节中的命令可由一般用户运行。开发者可以编写使用这些命令的脚本。

要在 FIPS 140 模式下设置系统，必须使用 FIPS 验证的算法、模式和密钥长度。请参阅《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithm Lists and Certificate References for Oracle Solaris Systems”。

加密框架可帮助您保护文件。以下任务列表列出了用于列出可用算法以及通过加密方式保护文件的过程。

表 3-1 使用加密框架保护文件（任务列表）

任务	说明	参考
生成对称密钥。	生成具有用户指定长度的密钥。或者，可以选择将密钥存储在文件、PKCS #11 密钥库或 NSS 密钥库中。 对于 FIPS 140 认可的模式，选择已针对 FIPS 验证的密钥类型、模式和密钥长度。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。	如何使用 pktool 命令生成对称密钥 [20]
提供可确保文件完整性的校验和。	检验接收者的文件副本是否与发送的文件完全相同。	如何计算文件摘要 [25]
使用消息验证代码 (message authentication code, MAC) 保护文件。	向消息接收者确认您是发送者。	如何计算文件的 MAC [26]

任务	说明	参考
对文件加密，然后将加密文件解密。	通过对文件进行加密来保护文件内容。提供加密参数用于解密文件。	如何加密和解密文件 [28]

▼ 如何使用 pktool 命令生成对称密钥

某些应用程序要求使用对称密钥对通信进行加密和解密。在此过程中，应创建一个对称密钥并存储它。

如果您的站点有随机数生成器，可使用此生成器为该密钥创建随机数。此过程不使用您站点的随机数生成器。

1. （可选） 如果计划使用密钥库，需要创建一个。

   - 有关创建和初始化 PKCS #11 密钥库的信息，请参见如何使用 **pktool setpin** 命令生成口令短语 [56]。
   - 要创建和初始化 NSS 数据库，请参见例 4-5 “使用口令短语保护密钥库” 中的示例命令。

2. 生成一个随机数以用作对称密钥。
   使用以下方法之一。

   - 生成一个密钥并将其存储于文件中。
     以文件存储的密钥的优点是，您可以从该文件提取要在应用程序的密钥文件中使用的密钥，如 /etc/inet/secret/ipseckeys 文件或 IPsec。用法语句显示了参数。

     ```
     % 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*

     存储密钥的文件名。

     keytype=*specific-symmetric-algorithm*

     无论对称密钥的长度如何，值均为 generic。对于特定算法，请指定 aes、arcfour、des 或 3des。

     对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥类型。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

keylen=*size-in-bits*

密钥的长度（位）。该数字必须可以被 8 整除。请勿对 des 或 3des 指定此值。

对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥长度。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

print=n

将密钥显示至终端窗口。缺省情况下，print 的值为 n。

- **生成一个密钥并将其存储于 PKCS #11 密钥库中。**

  PKCS #11 密钥库的优点是，您可以按密钥标签检索密钥。此方法适用于加密和解密文件所用的密钥。在使用此方法之前，必须先完成步骤 1。用法语句显示了参数。包围着 keystore 参数的括号表示未指定 keystore 参数时，密钥将存储在 PKCS #11 密钥库中。

```
$ 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*

密钥的用户指定标签。可以从密钥库中按密钥标签检索密钥。

keytype=*specific-symmetric-algorithm*

无论对称密钥的长度如何，值均为 generic。对于特定算法，请指定 aes、arcfour、des 或 3des。

对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥类型。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

keylen=*size-in-bits*

密钥的长度（位）。该数字必须可以被 8 整除。请勿对 des 或 3des 指定此值。

对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥长度。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

token=*token*

令牌名称。缺省情况下，令牌是 Sun Software PKCS#11 softtoken。

`sensitive=n`

指定密钥的敏感度。当此值为 y 时，不可使用 `print=y` 参数显示密钥。缺省情况下，`sensitive` 的值为 `n`。

`extractable=y`

指定可从密钥库提取密钥。指定 `n` 可阻止提取密钥。

`print=n`

将密钥显示至终端窗口。缺省情况下，`print` 的值为 `n`。

- **生成一个密钥并将其存储于 NSS 密钥库中。**

  在使用此方法之前，必须先完成步骤 1。用法语句显示了参数。

```
$ 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*

密钥的用户指定标签。可以从密钥库中按密钥标签检索密钥。

`keytype=`*specific-symmetric-algorithm*

无论对称密钥的长度如何，值均为 `generic`。对于特定算法，请指定 `aes`、`arcfour`、`des` 或 `3des`。

对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥类型。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

`keylen=`*size-in-bits*

密钥的长度（位）。该数字必须可以被 8 整除。请勿对 `des` 或 `3des` 指定此值。

对于 FIPS 140 认可的算法，选择已针对 FIPS 验证的密钥长度。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”。

`token=`*token*

令牌名称。缺省情况下，该令牌为 NSS 内部令牌。

`dir=`*directory*

NSS 数据库的目录路径。缺省情况下，*directory* 是当前目录。

prefix=*directory*

NSS 数据库的前缀。缺省为无前缀。

3. （可选） 检验密钥是否存在。
根据密钥的存储位置，使用以下命令之一。

- 检验 *key-fn* 文件中的密钥。

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

- 检验 PKCS #11 或 NSS 密钥库中的密钥。

```
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)
```

或者，在命令中将 keystore=pkcs11 替换为 keystore=nss。

例 3-1 使用 pktool 命令创建对称密钥

在以下示例中，用户首先创建一个 PKCS #11 密钥库，随后为应用程序生成一个较大的对称密钥。最后，用户验证密钥是否位于密钥库中。

请注意，PKCS #11 密钥库的初始口令是 changeme。NSS 密钥库的初始口令是空口令。

```
# 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 使用 pktool 命令创建 FIPS 认可的 AES 密钥

在下面的示例中，将使用 FIPS 认可的算法和密钥长度创建一个用于 AES 算法的密钥。该密钥存储在本地文件中以供日后解密之用。该命令使用 400 权限保护文件。创建了密钥后，print=y 选项将在终端窗口显示生成的密钥。

拥有密钥文件的用户将使用 od 命令检索密钥。

```
% 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 为 IPsec 安全关联创建对称密钥

在以下示例中，管理员手动为 IPsec SA 创建密钥材料并将其存储在文件中。然后，管理员将密钥复制到 /etc/inet/secret/ipseckeys 文件，并销毁原始文件。

首先，管理员创建并显示 IPsec 策略要求的密钥：

```
# 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"
```

然后，管理员创建以下 /etc/inet/secret/ipseckeys 文件：

```
##   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
```

在验证 ipseckeys 文件的语法有效之后，管理员将销毁原始密钥文件。

```
# ipseckey -c /etc/inet/secret/ipseckeys
# rm ipencrin1 ipencrout1 ipspi1 ipspi2 ipsha21 ipsha22
```

管理员通过使用 ssh 命令或其他安全机制，将 ipseckeys 文件复制到通信系统。在通信系统中，将反转保护。ipseckeys 文件中的第一个条目保护传入数据包，第二个条目保护传出数据包。通信系统中没有生成任何密钥。

接下来的步骤 要继续使用该密钥创建文件的消息验证代码 (Message Authentication Code, MAC)，请参阅如何计算文件的 MAC [26]。

▼ 如何计算文件摘要

计算文件摘要时，可以通过比较摘要输出来检查文件是否被篡改。摘要不会修改原始文件。

1. **列出可用的摘要算法。**

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

注 - 请尽可能按照《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中的列表选择 FIPS 认可的算法。

2. **计算文件摘要并保存摘要列表。**

为 digest 命令提供一种算法。

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

-v 显示以下格式的输出：

algorithm（*input-file*）= *digest*

-a *algorithm* 用于计算文件摘要的算法。键入步骤 1 的输出中显示的算法。

注 - 请尽可能选择《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中列出的 FIPS 认可算法。

input-file digest 命令的输入文件。

digest-listing digest 命令的输出文件。

例 3-4 使用 SHA1 机制计算摘要

在下面的示例中，digest 命令使用 SHA1 机制提供目录列表。结果存放于文件中。

```
% 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
```

▼ 如何计算文件的 MAC

消息验证代码 (message authentication code, MAC) 可计算文件的摘要并使用密钥进一步保护该摘要。MAC 不会修改原始文件。

1. **列出可用机制。**

```
% 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 - 每个受支持的算法都是特定算法类型的最常用和受限最小版本的别名。上面的输出显示了可用算法名称和每种算法的密钥大小。请尽可能使用受支持的算法，该算法与具有 FIPS 认可的密钥长度的 FIPS 认可算法相匹配，列在《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中。

2. **生成长度适当的对称密钥。**

可以提供据以生成密钥的 passphrase（口令短语），也可以提供一个密钥。

- 如果提供口令短语，则必须存储或记住该口令短语。如果您联机存储口令短语，则只有您才可以读取该口令短语文件。
- 如果提供密钥，则其大小必须是适用于相应机制的正确大小。您可以使用 pktool 命令。有关该过程和一些示例，请参见如何使用 pktool 命令生成对称密钥 [20]。

3. **创建文件的 MAC。**

为 mac 命令提供密钥并使用对称密钥算法。

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

-v	显示以下格式的输出： *algorithm*（*input-file*）= *mac*
-a *algorithm*	用于计算 MAC 的算法。键入 `mac -l` 命令的输出中显示的算法。
-k *keyfile*	包含算法所指定长度的密钥的文件。
-K *key-label*	PKCS #11 密钥库中的密钥标签。
-T *token*	令牌名称。缺省情况下，令牌是 `Sun Software PKCS#11 softtoken`。仅在使用 -K *key-label* 选项时才使用。
input-file	MAC 的输入文件。

例 3-5 使用 SHA1_HMAC 和口令短语计算 MAC

在下面的示例中，将使用 SHA1_HMAC 机制和基于口令短语的密钥对电子邮件附件进行验证。MAC 列表将保存到文件中。如果口令短语存储在某个文件中，则除了该用户之外，其他任何人都不能读取该文件。

```
% 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 使用 SHA1_HMAC 和密钥文件计算 MAC

在下面的示例中，将使用 SHA1_HMAC 机制和密钥对目录清单进行验证。结果存放于文件中。

```
% 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 使用 SHA1_HMAC 和密钥标签计算 MAC。

在下面的示例中，将使用 SHA1_HMAC 机制和密钥对目录清单进行验证。结果将放置在用户的 PKCS #11 密钥库中。用户最初通过使用 `pktool setpin` 命令创建了密钥库及其口令。

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

要从密钥库中检索 MAC，用户应使用详细选项，并提供密钥标签和通过验证的目录名称。

```
% 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
```

▼ 如何加密和解密文件

加密文件时，不会删除或更改原始文件。输出文件将被加密。

有关 `encrypt` 命令相关常见错误的解决方案，请参见示例后面的部分。

注 - 加密和解密文件时，请尽可能尝试使用具有认可的密钥长度的 FIPS 认可算法。请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中的列表。运行 `encrypt -l` 命令查看可用算法及其密钥长度。

---

1. **创建长度适当的对称密钥。**
   可以提供据以生成密钥的 passphrase（口令短语），也可以提供一个密钥。
   - 如果提供口令短语，则必须存储或记住该口令短语。如果您联机存储口令短语，则只有您才可以读取该口令短语文件。
   - 如果提供密钥，则其大小必须是适用于相应机制的正确大小。您可以使用 `pktool` 命令。有关该过程和一些示例，请参见如何使用 pktool 命令生成对称密钥 [20]。
2. **对文件加密。**
   提供密钥并使用 `encrypt` 命令及对称密钥算法参数。

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

-a *algorithm*	用于对文件加密的算法。键入 `encrypt -l` 命令输出中显示的算法。请尽可能按照《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中的列表选择 FIPS 认可的算法。
-k *keyfile*	包含算法所指定长度的密钥的文件。在 `encrypt -l` 命令的输出中会列出每种算法的密钥长度（位）。

-K *key-label*	PKCS #11 密钥库中的密钥标签。
-T *token*	令牌名称。缺省情况下，令牌是 Sun Software PKCS#11 softtoken。仅在使用 -K *key-label* 选项时才使用。
-i *input-file*	要加密的输入文件。该命令不会更改此文件。
-o *output-file*	输出文件，是输入文件的加密形式。

例 3-8 创建 AES 密钥以加密文件

在以下示例中，用户创建 AES 密钥并将其存储在现有 PKCS #11 密钥库中以用于加密和解密。用户可以验证密钥是否存在，也可以使用密钥，但是无法查看密钥本身。

```
% 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
```

要使用密钥来加密文件，用户应通过密钥标签检索密钥。

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

要解密 encryptedthisfile 文件，用户应通过密钥标签检索密钥。

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

例 3-9 使用 AES 和口令短语进行加密和解密

在下面的示例中，将使用 AES 算法来加密文件。密钥基于口令短语生成。如果口令短语存储在某个文件中，则除了该用户之外，其他任何人都不能读取该文件。

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

输入文件 ticket.to.ride 仍然以其原始形式存在。

要解密输出文件，用户应使用加密该文件的相同口令短语和加密机制。

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

例 3-10 使用 AES 和密钥文件进行加密和解密

在下面的示例中，将使用 AES 算法来加密文件。AES 机制使用 128 位（即 16 字节）密钥。

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

输入文件 `ticket.to.ride` 仍然以其原始形式存在。

要解密输出文件，用户应使用加密文件时所用的密钥和加密机制。

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

故障排除 以下消息表明，所使用的算法不接受您提供给 `encrypt` 命令的密钥。

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

如果传递的密钥不满足算法的要求，则必须使用以下方法之一提供更好的密钥：

- 使用口令短语。然后，框架提供满足要求的密钥。
- 传递算法接受的密钥大小。例如，DES 算法要求 64 位的密钥。3DES 算法要求 192 位的密钥。

管理加密框架

本节介绍如何管理加密框架中的软件提供者和硬件提供者。如果需要，可以禁用软件提供者和硬件提供者。例如，可以禁用某个软件提供者的算法实现。然后，可以强制系统使用其他软件提供者的算法。

注 - 管理密钥框架的一个重要组成部分是规划和实现有关 FIPS 140 的策略、加密模块的美国政府计算机安全标准。

如果有严格的要求，只能使用 FIPS 140-2 验证的加密，则必须运行 Oracle Solaris11.1 SRU5.5 发行版或 Oracle Solaris11.1 SRU3 发行版。Oracle 已在这两个特定发行版中针对 Solaris 加密框架完成了 FIPS 140-2 验证。Oracle Solaris11.2 基于此验证的基础而构建并在性能、功能和可靠性方面进行了软件改进。应当尽可能在 FIPS 140-2 模式下配置 Oracle Solaris11.2 以利用这些改进。

查看《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》并规划系统的总体 FIPS 策略。

以下任务列表列出了管理加密框架中的软件和硬件提供者的过程。

表 3-2 管理加密框架（任务列表）

任务	说明	参考
为系统规划 FIPS 策略。	确定用于启用 FIPS 认可的提供者和使用者的规划并实现规划。	《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》
列出加密框架中的提供者。	列出可在加密框架中使用的算法、库和硬件设备。	“列出可用的提供程者” [31]
启用 FIPS 140 模式。	根据针对加密模块的美国政府标准运行加密框架。	如何创建启用了 FIPS 140 的引导环境 [38]
添加软件提供者。	将 PKCS #11 库或内核模块添加到加密框架中。必须对提供者进行签名。	如何添加软件提供者 [36]
禁止使用用户级机制。	禁用软件机制。可以再次启用该机制。	如何禁止使用用户级机制 [40]
临时禁用内核模块的机制。	临时禁用某个机制。通常用于测试。	如何禁止使用内核软件机制 [41]
卸载库。	禁用用户级软件提供者。	例 3-17 “永久删除用户级库”
卸载内核提供者。	禁用内核软件提供者。	例 3-19 “临时禁用内核软件提供者”
禁用硬件提供者的机制。	确保未使用硬件加速器的所选机制。	如何禁用硬件提供者机制和功能 [44]
重新启动或刷新加密服务。	确保加密服务可用。	如何刷新或重新启动所有加密服务 [46]

列出可用的提供程者

硬件提供者将自动定位和装入。有关更多信息，请参见 `driver.conf(4)` 手册页。

如果您的硬件预计会插入加密框架中，该硬件将通过 SPI 在内核中注册。框架将检查是否已对硬件驱动程序签名。具体地说，框架将检查是否已使用 Oracle 颁发的证书对驱动程序的目标文件进行签名。

例如，Sun Crypto Accelerator 6000 板 (`mca`)、UltraSPARC T1 和 T2 处理器上加密加速器的 `ncp` 驱动程序 (`ncp`)、UltraSPARC T2 处理器的 n2cp 驱动程序 (`n2cp`) 以及 T 系列系统的 `/dev/crypto` 驱动程序将硬件机制插入到框架中。

有关对提供者签名的信息，请参见“加密框架中的用户级命令” [11]中的 `elfsign` 命令。

要列出可用的提供者，可使用包含不同选项的 `cryptoadm list` 命令，具体取决于要获取的具体信息。

- 列出系统上的所有提供者。

  对于不同的 Oracle Solaris 发行版和不同的平台，提供者列表的内容和格式会有所不同。在您的系统上运行 `cryptoadm list` 命令可查看系统支持的提供者。一般用户只能直接使用用户级机制。

  ```
  % 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
```

- 列出加密框架中的提供者及其机制。

  可以查看可用机制的强度和模式（例如 ECB 和 CBC）。但是，所列出的一些机制可能无法使用。有关如何列出可使用的机制的说明，请参见下一个项目。

  为了便于显示，以下输出被截断。

```
% 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
```

- 列出可用的加密机制。

  策略确定可使用的机制。管理员负责设置策略。管理员可以选择禁用特定提供者的机制。-p 选项显示管理员设置的策略允许的机制列表。

```
% 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.
```

以下示例说明 `cryptoadm list` 命令的其他特定用途。

例 3-11 列出特定提供者的加密信息

在 `cryptoadm` *options* 命令中指定提供者可将输出限制为仅输出适用于提供者的信息。

```
# 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.
```

以下输出显示仅启用了机制。随机数生成器将继续处于禁用状态。

```
# 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.
```

以下输出显示启用了板上的每个功能和机制。

```
# 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 仅查找用户级加密机制

在以下示例中，列出了用户级库 pkcs11_softtoken 提供的所有机制。

```
% 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 确定哪些加密机制执行哪些功能

机制执行特定加密函数，如签名或密钥生成。-v -m 选项显示每个机制及其功能。

在本实例中，管理员要确定 CKM_ECDSA* 机制可用于哪些功能。

```
% 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
...
```

列表内容表明这些机制可从用户级提供者获得：

- CKM_ECDSA 和 CKM_ECDSA_SHA1 – 作为 /usr/lib/security/$ISA/pkcs11_softtoken.so 库中的软件实现
- CKM_ECDSA – 通过 /usr/lib/security/$ISA/pkcs11_kernel.so 库中的 ncp/0 Crypto Accel Asym 1.0 加速

条目中的每一项代表机制的一条相关信息。对于这些 ECC 机制，该列表指示以下内容：

- 最小长度 – 112 字节
- 最大长度 – 571 字节
- 硬件 – 硬件是否可用。
- 加密 – 不可用于加密数据。
- 解密 – 不可用于解密数据。
- 摘要 – 不可用于创建消息摘要。
- 签名 – 用于对数据签名。
- 签名 + 恢复 – 不可用于对能从签名中恢复的数据签名。
- 验证 – 用于验证签名数据。
- 验证 + 恢复 – 不可用于验证能从签名中恢复的数据。
- 密钥生成 – 不可用于生成私钥。
- 密钥对生成 – 不可用于生成密钥对。
- 包装 – 不可用于包装。即加密，现有密钥。
- 解包 – 不可用于将已包装的密钥解包。
- 派生 – 不可用于从基本密钥派生新密钥。
- EC 功能 – 前面的项未涵盖的缺少的 EC 功能

添加软件提供者

以下过程说明了如何将提供者添加到系统。您必须是指定有 Crypto Management（加密管理）权限配置文件的管理员。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

▼ 如何添加软件提供者

1. **列出系统可用的软件提供者。**

```
% 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. 从系统信息库添加提供者。
   Oracle 已向现有提供者软件颁发证书。

3. 刷新提供者。
   如果添加了软件提供者或者添加了硬件并为该硬件指定了策略，则需要刷新提供者。

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

4. 在列表中找到新增的提供者。
   在本例中，安装了一个新的内核软件提供者。

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

例 3-14 添加用户级软件提供者

在下面的示例中，将安装签名的 PKCS #11 库。

```
# 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
```

使用加密框架测试库的开发者可以手动安装该库。

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

创建启用了 FIPS 140 的引导环境

缺省情况下，Oracle Solaris 中禁用了 FIPS 140 模式。在此过程中，您将为 FIPS 140 模式创建一个新的引导环境 (boot environment, BE)，然后启用 FIPS 140 并引导进入新的 BE。通过为您提供备份 BE，此方法允许您从由于 FIPS 140 符合性测试导致的系统紧急情况中快速恢复。

有关 FIPS 的概述，请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》。另请参见 `cryptoadm`(1M) 手册页和“加密框架和 FIPS-140” [12]。

▼ 如何创建启用了 FIPS 140 的引导环境

开始之前 您必须承担 `root` 角色。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

1. 确定系统是否处于 FIPS 140 模式。

```
% 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. 为加密框架的 FIPS 140 版本创建一个新的 BE。

   在启用 FIPS 140 模式之前，您必须先使用 `beadm` 命令创建、激活并引导新的 BE。启用了 FIPS 140 的系统将运行符合性测试，如果这些测试失败，会导致出现紧急情况。因

此，在调试 FIPS 140 边界问题时，具有可用于引导以启动系统并让系统运行的 BE 很重要。

a. 基于您当前的 BE 创建 BE。

在此示例中，您将创建一个名为 S11.1-FIPS 的 BE。

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

b. 激活该 BE。

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

c. 重新引导系统。

d. 在新的 BE 中启用 FIPS 140 模式。

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

注 - 此子命令不会禁用用户级 pkcs11_softtoken 库以及内核软件提供者中的未经 FIPS 140 认可的算法。框架使用者只能使用经 FIPS 140 认可的算法。

有关 FIPS 140 模式的效果的更多信息，请参见《Using a FIPS 140 Enabled System in Oracle Solaris 11.2》。另请参见 cryptoadm(1M) 手册页。

---

3. 如果希望在不启用 FIPS 140 的情况下运行，请禁用 FIPS 140 模式。

您可以重新引导至原始 BE 或在当前 BE 中禁用 FIPS 140。

- 引导至原始 BE。

```
# 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
```

- 在当前 BE 中禁用 FIPS 140 模式并重新引导。

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

FIPS 140 模式将保持运行，直到系统重新引导。

```
# reboot
```

阻止使用机制

如果不应使用库提供者的某些加密机制，可以删除这些所选的机制。例如，如果另一个库中的相同机制表现得更好，或者调查发现了安全漏洞，则可能要考虑阻止使用机制。

如果加密框架提供了某个提供者（如 AES）的多种模式，则可以禁用速度较慢的机制或已损坏的机制。还可以使用本过程删除经证明存在安全漏洞的算法。

可以有选择地禁用硬件提供者的机制和随机数功能。有关重新启用它们的信息，请参见例 3-22 “启用硬件提供者的机制和功能”。此示例中的硬件 Sun Crypto Accelerator 1000 板提供了一个随机数生成器。

▼ 如何禁止使用用户级机制

开始之前 您必须是指定有 Crypto Management（加密管理）权限配置文件的管理员。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

1. 列出特定用户级软件提供者提供的机制。

```
% 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. 列出可使用的机制。

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

3. 禁用不应使用的机制。

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

4. 列出可使用的机制。

```
$ 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 启用一个用户级软件提供者机制

在下面的示例中，将使禁用的 DES 机制再次可用。

```
$ 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 启用所有用户级软件提供者机制

在下面的示例中，将启用用户级库的所有机制。

```
$ 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 永久删除用户级库

在下面的示例中，将从 `/opt` 目录中删除 `libpkcs11.so.1` 库。

```
$ 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:
…
```

▼ 如何禁止使用内核软件机制

开始之前 您必须是指定有 Crypto Management（加密管理）权限配置文件的管理员。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

1. 列出特定内核软件提供者提供的机制。

```
$ 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. 列出可使用的机制。

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

3. 禁用不应使用的机制。

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

4. 列出可使用的机制。

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

例 3-18 启用内核软件提供者机制

在下面的示例中，将使禁用的 AES 机制再次可用。

```
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 临时禁用内核软件提供者

在下面的示例中，将临时禁用 AES 提供者。`unload` 子命令可用于禁止在卸载某个提供者时自动装入该提供者。例如，修改此提供者的机制时，可能使用 `unload` 子命令。

```
$ 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 提供者只有在加密框架刷新之后才会可用。

```
$ 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
```

如果内核使用者正在使用内核软件提供者，则不会卸载该软件。此时将显示错误消息，不过该提供者仍然可供使用。

例 3-20 永久禁用软件提供者

在下面的示例中，将禁用 AES 提供者。禁用后，该 AES 提供者将不会出现在内核软件提供者的策略列表中。

```
$ 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
```

如果内核使用者正在使用内核软件提供者，将显示一条错误消息，不过该提供者仍然可供使用。

例 3-21 重新安装已删除的内核软件提供者

在下面的示例中，将重新安装 AES 内核软件提供者。要重新安装已禁用的内核提供者，必须枚举要安装的机制。

```
$ 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
```

▼ 如何禁用硬件提供者机制和功能

开始之前 您必须是指定有 Crypto Management（加密管理）权限配置文件的管理员。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

- **选择要禁用的机制或功能。**

  列出硬件提供者。

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

- 禁用选定的机制。

```
# 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.
```

- 禁用随机数生成器。

```
# 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.
```

- 禁用所有机制。不禁用随机数生成器。

```
# 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.
```

- 禁用该硬件上所有功能和机制。

```
# 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 启用硬件提供者的机制和功能

在下面的示例中，将有选择地启用单个硬件的已禁用机制。

```
# 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.
```

在下面的示例中，将仅启用随机数生成器。

```
# 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.
```

在下面的示例中，将仅启用机制。随机数生成器将继续处于禁用状态。

```
# 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.
```

在下面的示例中，将启用板上的所有功能和机制。

```
# 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.
```

刷新或重新启动所有加密服务

缺省情况下，将启用加密框架。无论 `kcfd` 守护进程因何种原因失败，都可使用服务管理工具 (Service Management Facility, SMF) 重新启动加密服务。有关更多信息，请参见 `smf(5)` 和 `svcadm(1M)` 手册页。有关重新启动加密服务对区域的影响，请参见“加密服务和区域” [12]。

▼ 如何刷新或重新启动所有加密服务

开始之前 您必须是指定有 Crypto Management（加密管理）权限配置文件的管理员。有关更多信息，请参见《在 Oracle Solaris 11.2 中确保用户和进程的安全 》中的“使用所指定的管理权限”。

1. **检查加密服务的状态。**

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

2. **启用加密服务。**

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

例 3-23 刷新加密服务

在下面的示例中，将在全局区域中刷新加密服务。因此，每个非全局区域中的内核级加密策略也将被刷新。

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

♦ ♦ ♦ 第 4 章

密钥管理框架

Oracle Solaris 的密钥管理框架 (Key Management Framework, KMF) 功能提供了用于管理公钥对象的工具及编程接口。公钥对象包括 X.509 证书和公钥/私钥对。存储这些对象所用的格式可能有所不同。KMF 还提供了一种工具，用于管理定义应用程序如何使用 X.509 证书的策略。KMF 支持第三方插件。

本章包含以下主题：

管理公钥技术

KMF 集中管理公钥技术 (PKI)。Oracle Solaris 具有多种可使用 PKI 技术的不同应用程序。每种应用程序都提供自己的编程接口、密钥存储机制和管理实用程序。如果某个应用程序提供了策略执行机制，则此机制将仅适用于该应用程序。借助 KMF，各应用程序可以使用一套统一的管理工具，单一一组编程接口，以及同一个策略执行机制。这些功能可以管理采用这些接口的所有应用程序的 PKI 需求。

KMF 通过以下接口统一管理公钥技术：

- `pktool` 命令 – 管理各种密钥库中的 PKI 对象（如证书）。
- `kmfcfg` 命令 – 管理 PKI 策略数据库和第三方插件。

  PKI 策略决策包括一些操作，如某项操作的验证方法。此外，PKI 策略还可以限制证书的范围。例如，PKI 策略可以声明某个证书仅可用于特定的目的。这种策略可防止将证书用于其他请求。
- KMF 库 – 包含将底层密钥库机制抽象化的编程接口。

  应用程序不必选择某种特定的密钥库机制，但可以从一种机制迁移到另一种机制。支持的密钥库包括 PKCS #11、NSS 和 OpenSSL。该库具有可插接式框架，因此可添加新的密钥库机制。由于这一原因，使用新机制的应用程序将只需要微小的修改就能使用新密钥库。

密钥管理框架实用程序

KMF 提供了管理密钥存储的方法，并且提供了使用这些密钥的整体策略。KMF 可管理以下三种公钥技术的策略、密钥和证书：

- 来自 PKCS #11 提供者（即来自加密框架）的令牌
- NSS，即网络安全服务 (Network Security Services)
- OpenSSL，一种基于文件的密钥库

`kmfcfg` 工具可以创建、修改或删除 KMF 策略项。该工具还可以管理框架的插件。KMF 通过 `pktool` 命令管理密钥库。有关更多信息，请参阅 `kmfcfg(1)` 和 `pktool(1)` 手册页以及以下各部分。

KMF 策略管理

KMF 策略存储在数据库中。使用 KMF 编程接口的所有应用程序在内部访问此策略数据库。该数据库可以约束由 KMF 库管理的密钥和证书的使用。当某个应用程序试图验证证书时，该应用程序将检查策略数据库。使用 `kmfcfg` 命令可修改策略数据库。

KMF 插件管理

`kmfcfg` 命令提供了以下插件子命令：

- `list plugin` – 列出由 KMF 管理的插件。
- `install` *plugin* – 按模块的路径名安装插件并为插件创建密钥库。要从 KMF 删除插件，请删除密钥库。
- `uninstall` *plugin* – 通过删除插件的密钥库从 KMF 删除插件。
- `modify` *plugin* – 使插件通过在插件的代码中定义的选项（如 `debug`）运行。

有关更多信息，请参见 `kmfcfg(1)` 手册页。有关过程，请参见如何在 KMF 中管理第三方插件 [62]。

KMF 密钥库管理

KMF 可管理以下三种公钥技术的密钥库：PKCS #11 令牌、NSS 和 OpenSSL。对于所有这些技术，`pktool` 命令可用于执行以下操作：

- 生成自签名证书
- 生成证书请求
- 生成对称密钥

- 生成公钥/私钥对
- 生成要发送到外部证书颁发机构 (certificate authority, CA) 进行签署的 PKCS #10 证书签名请求 (certificate signing request, CSR)
- 签署 PKCS #10 CSR
- 将对象导入密钥库
- 列出密钥库中的对象
- 删除密钥库中的对象
- 下载 CRL

对于 PKCS #11 和 NSS 技术，还可使用 `pktool` 命令为密钥库或密钥库中的对象生成口令短语来设置 PIN。

有关使用 `pktool` 实用程序的示例，请参见 `pktool(1)` 手册页以及表 4-1 “使用密钥管理框架任务列表”。

使用密钥管理框架

本节介绍如何使用 `pktool` 命令管理公钥对象，例如口令、口令短语、文件、密钥库、证书和 CRL。

通过密钥管理框架 (Key Management Framework, KMF) 可以集中管理公钥技术。

表 4-1 使用密钥管理框架任务列表

任务	说明	参考
创建证书。	创建供 PKCS #11、NSS 或 OpenSSL 使用的证书。	如何使用 `pktool gencert` 命令创建证书 [52]
导出证书。	创建包含证书及其支持密钥的文件。可以使用口令保护该文件。	如何以 PKCS #12 格式导出证书和私钥 [55]
导入证书。	从其他系统导入证书。	如何将证书导入密钥库 [53]
	从其他系统导入 PKCS #12 格式的证书。	例 4-2 “将 PKCS #12 文件导入密钥库”
生成口令短语。	生成访问 PKCS #11 密钥库或 NSS 密钥库所用的口令短语。	如何使用 `pktool setpin` 命令生成口令短语 [56]
生成对称密钥。	生成用于加密文件、创建文件的 MAC 和应用程序的对称密钥。	如何使用 `pktool` 命令生成对称密钥 [20]
生成密钥对。	生成用于应用程序的公钥/私钥对。	如何使用 `pktool genkeypair` 命令生成密钥对 [57]
生成 PKCS #10 CSR。	生成要外部证书颁发机构 (CA) 签署的 PKCS #10 证书签名请求 (certificate signing request, CSR)。	`pktool(1)` 手册页
签署 PKCS #10 CSR。	签署 PKCS #10 CSR。	如何使用 `pktool signcsr` 命令签署证书请求 [61]

任务	说明	参考
将插件添加到 KMF 中。	安装、修改和列出插件。此外，还从 KMF 中删除插件。	如何在 KMF 中管理第三方插件 [62]

▼ 如何使用 pktool gencert 命令创建证书

此过程可创建自签名证书，并将该证书存储在 PKCS #11 密钥库中。在此操作过程中，还将创建一个 RSA 公钥/私钥对。私钥与该证书一起存储在密钥库中。

1. **生成自签名证书。**

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

keystore=*keystore*	按公钥对象类型指定密钥库。该值可以是 nss、pkcs11 或 file。此关键字是可选的。
label=*label-name*	指定颁发者为证书提供的唯一名称。
subject=*subject-DN*	指定证书的标识名。
serial=*hex-serial-number*	指定十六进制格式序列号。证书的颁发者可以选择该序列号，例如 0x0102030405。
keytype=*key type*	可选变量，指定与证书关联的私钥的类型。请查看 pktool(1) 手册页以了解选定密钥库的可用密钥类型。 要使用 FIPS 140 认可的密钥，请查看《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中的认可密钥类型。
keylen=*key size*	可选变量，指定与证书关联的私钥的长度。 要使用 FIPS 140 认可的密钥，请查看在《Using a FIPS 140 Enabled System in Oracle Solaris 11.2 》中的“FIPS 140 Algorithms in the Cryptographic Framework”中选择的密钥类型的认可密钥长度。

2. **检验密钥库的内容。**

```
% 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. ...
```

此命令可列出密钥库中的所有证书。在下面的示例中，密钥库只包含一个证书。

例 4-1 使用 pktool 创建自签名证书

在下面的示例中，My Company 的用户创建了一个自签名证书，并将该证书存储在 PKCS #11 对象的密钥库中。该密钥库最初是空的。如果尚未初始化此密钥库，则软令牌的 PIN 为 changeme，并且您可以使用 pktool setpin 命令重置该 PIN。请注意，在命令选项中指定 FIPS 认可的密钥类型和密钥长度 RSA 2048。

```
% 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
...
```

▼ 如何将证书导入密钥库

此过程介绍如何将包含以 PEM 或原始 DER 编码的 PKI 信息的文件导入密钥库。有关导出过程的信息，请参见例 4-4 “以 PKCS #12 格式导出证书和私钥”。

1. 导入证书。

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

2. 如果要导入私有 PKI 对象，需要在系统提示时提供口令。

   a. 在提示符下键入文件的口令。
   如果要导入的是私有 PKI 信息（例如 PKCS #12 格式的导出文件），则需要为该文件提供口令。您要导入的文件的创建者会向您提供 PKCS #12 口令。

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

b. 在提示符下键入密钥库的口令。

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

3. 检验密钥库的内容。

```
% 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 将 PKCS #12 文件导入密钥库

在下面的示例中，用户从第三方导入了一个 PKCS #12 文件。`pktool import` 命令可从 `gracedata.p12` 文件提取私钥和证书，并将它们存储在用户的首选密钥库中。

```
% 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 将 X.509 证书导入密钥库

在下面的示例中，用户将 PEM 格式的 X.509 证书导入用户的首选密钥库。此公共证书不受口令保护。用户的公共密钥库也不受口令保护。

```
% 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
```

▼ 如何以 PKCS #12 格式导出证书和私钥

可以创建一个 PKCS #12 格式的文件，用于将私钥及其关联的 X.509 证书导出到其他系统。对该文件的访问受口令保护。

1. **找到要导出的证书。**

```
% 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. **导出密钥和证书。**

使用 `pktool list` 命令生成的密钥库和标签。为导出文件提供文件名。如果文件名包含空格，需要用双引号将其括起来。

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

3. **使用口令保护导出文件。**

在提示符下键入密钥库的当前口令。此时，即会为导出文件创建口令。接收者在导入该文件时必须提供此口令。

```
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
```

提示 - 将口令与导出文件分开发送。建议的最佳做法是不通过发送数据来提供口令，例如通过打电话。

例 4-4 以 PKCS #12 格式导出证书和私钥

在下面的示例中，用户将私钥及其关联的 X.509 证书导出到一个标准的 PKCS #12 文件中。可将此文件导入到其他密钥库中。PKCS #11 口令可保护源密钥库。PKCS #12 口令用于保护 PKCS #12 文件中的私有数据。导入该文件时需要提供此口令。

```
% 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
```

然后，用户打电话给接收者，提供 PKCS #12 口令。

▼ 如何使用 pktool setpin 命令生成口令短语

可以为密钥库中的对象以及密钥库本身生成口令短语。访问该对象或密钥库时需要提供此口令短语。有关为密钥库中的对象生成口令短语的示例，请参见例 4-4 “以 PKCS #12 格式导出证书和私钥”。

1. 生成访问密钥库所用的口令短语。

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

   密钥存储的缺省目录为 /var/*username*。

   PKCS #11 密钥库的初始口令为 changeme。NSS 密钥库的初始口令是空口令。

2. 回答提示。

   当系统提示输入当前令牌口令短语时，对于 PKCS #11 密钥库请键入令牌 PIN，对于 NSS 密钥库请按回车键。

   ```
   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.
   ```

   密钥库现在受口令短语保护了。如果丢失了口令短语，您将无法访问该密钥库中的对象。

3. （可选） 显示令牌列表。

   ```
   # pktool tokens
   ```

输出取决于是否启用了 metaslot。有关 metaslot 的更多信息，请参阅“加密框架中的概念” [9]。

- 如果启用了 metaslot，则 `pktools token` 命令将生成类似以下内容的输出：

```
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
```

- 如果禁用了 metaslot，则 `pktools token` 命令将生成类似以下内容的输出：

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

在两种输出版本中，标志可以是以下标志的任意组合：

- `L` – 需要登录
- `I` – 已初始化
- `X` – 用户 PIN 过期
- `S` – SO PIN 过期

例 4-5 使用口令短语保护密钥库

下面的示例说明了如何为 NSS 数据库设置口令短语。由于尚未创建口令短语，用户需要在第一个提示处按回车键。

```
% 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.
```

▼ 如何使用 pktool genkeypair 命令生成密钥对

某些应用程序需要公钥/私钥对。在此过程中，将创建这些密钥对并存储它们。

1. （可选） 如果计划使用密钥库，请创建密钥库。

   - 有关创建和初始化 PKCS #11 密钥库的信息，请参见如何使用 pktool setpin 命令生成口令短语 [56]。

- 要创建和初始化 NSS 密钥库，请参见例 4-5 “使用口令短语保护密钥库”。

2. 创建密钥对。

   使用以下方法之一。

   - 创建密钥对，然后将该密钥对存储在文件中。

     基于文件的密钥是为从磁盘上的文件中直接读取密钥的应用程序而创建的。通常，直接使用 OpenSSL 加密库的应用程序要求您将应用程序的密钥和证书存储在文件中。

     注 - file 密钥库不支持椭圆曲线 (elliptic curve, ec) 密钥和证书。

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

     keystore=file

     值 file 指定密钥存储位置的文件类型。

     outkey=*key-filename*

     指定存储密钥对的文件的名称。

     format=der|pem

     指定密钥对的编码格式。der 输出是二进制的，pem 输出为 ASCII。

     keytype=rsa|dsa

     指定可存储在 file 密钥库中的密钥对的类型。有关定义，请参见 DSA 和 RSA。

     keylen=*key-size*

     指定密钥的长度（位）。该数字必须可以被 8 整除。要确定可能的密钥大小，请使用 cryptoadm list -vm 命令。

   - 创建密钥对并将其存储在 PKCS #11 密钥库中。

     在使用此方法之前，必须先完成步骤 1。

     PKCS #11 密钥库用于将对象存储在硬件设备上。该设备可以是插入到加密框架的 Sun Crypto Accelerator 6000 卡、受信任的平台模块 (trusted platform module, TPM) 设备或智能卡。PKCS #11 还可以用于将对象存储在 softtoken，或基于软件的令牌中，从而将这些对象存储在磁盘上的专用子目录中。有关更多信息，请参见 pkcs11_softtoken (5) 手册页。

     可以从密钥库中按指定的标签检索密钥对。

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

label=*key-label*

为密钥对指定标签。可以从密钥库中按其标签检索密钥对。

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

指定令牌名称。缺省情况下，令牌名称是 Sun Software PKCS#11 softtoken。

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

指定密钥对类型。对于椭圆曲线 (elliptic curve, ec) 类型，可选择性地指定曲线名称。曲线名称作为输出列出到 listcurves 选项。

keylen=*key-size*

指定密钥的长度（位）。该数字必须可以被 8 整除。

listcurves

列出可用作 ec 密钥类型的 curve= 选项值的椭圆曲线名称。

- **生成密钥对并将其存储在 NSS 密钥库中。**

  NSS 密钥库由依赖 NSS 作为其主加密接口的服务器使用。

  在使用此方法之前，必须先完成步骤 1。

```
% 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

值 nss 指定密钥存储位置的 NSS 类型。

label=*nickname*

为密钥对指定标签。可以从密钥库中按其标签检索密钥对。

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

指定令牌名称。缺省情况下，令牌是 Sun Software PKCS#11 softtoken。

dir=*directory*

指定 NSS 数据库的目录路径。缺省情况下，*directory* 是当前目录。

`prefix=`*database-prefix*

指定 NSS 数据库的前缀。缺省为无前缀。

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

指定密钥对类型。对于椭圆曲线类型，（可选）指定曲线名称。曲线名称作为输出列出到 `listcurves` 选项。

`keylen=`*key-size*

指定密钥的长度（位）。该数字必须可以被 8 整除。

`listcurves`

列出可用作 `ec` 密钥类型的 `curve=` 选项值的椭圆曲线名称。

3. **（可选） 检验密钥是否存在。**

根据密钥的存储位置，使用以下命令之一：

- **检验 *key-filename* 文件中的密钥。**

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

- **检验 PKCS #11 密钥库中的密钥。**

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

- **检验 NSS 密钥库中的密钥。**

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

例 4-6 使用 `pktool` 命令创建密钥对

在以下示例中，用户首次创建 PKCS #11 密钥库。确定 RSA 密钥对的密钥大小后，用户生成应用程序的密钥对。最后，用户验证该密钥对是否在密钥库中。用户注意到 RSA 密钥对的第二个实例可以存储在硬件上。由于用户未指定 `token` 参数，因此该密钥对存储为 `Sun Software PKCS#11 softtoken`。

```
# 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 创建使用椭圆曲线算法的密钥对

在以下示例中，用户将椭圆曲线 (elliptic curve, ec) 密钥对添加到密钥库中，指定曲线名称并验证该密钥对是否在密钥库中。

```
% 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
```

▼ 如何使用 pktool signcsr 命令签署证书请求

此过程用于签署 PKCS #10 证书签名请求 (certificate signing request, CSR)。CSR 可以是 PEM 或 DER 格式。签署过程颁发 X.509 v3 证书。要生成 PKCS #10 CSR，请参见 pktool(1) 手册页。

开始之前 此过程假设您是证书颁发机构 (certificate authority, CA)，您收到了一个存储在文件中的 CSR。

1. 为 pktool signcsr 命令所需的参数收集以下信息：

   signkey　　如果已将签名者的密钥存储在 PKCS #11 密钥库中，则 signkey 是检索此私钥的标签。

如果已将签名者的密钥存储在 NSS 密钥库或文件密钥库中，则 `signkey` 是保存此私钥的文件名。

`csr` 指定 CSR 的文件名。

`serial` 指定已签名证书的序列号。

`outcer` 指定已签名证书的文件名。

`issuer` 指定采用标识名 (distinguished name, DN) 格式的 CA 颁发者名称。

有关 `signcsr` 子命令的可选参数的信息，请参见 `pktool(1)` 手册页。

2. 签署请求和颁发证书。
例如，以下命令可使用 PKCS #11 系统信息库中的签名者密钥签署证书。

```
# 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"
```

以下命令使用文件中的签名者密钥签署证书。

```
# 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. 将该证书发送给请求方。
您可以使用电子邮件、Web 站点或其他机制将该证书传送给请求方。
例如，您可以使用电子邮件将 `ExampleCoCert2010` 文件发送给请求方。

▼ 如何在 KMF 中管理第三方插件

通过向插件提供密钥库名称来标识该插件。将插件添加到 KMF 中时，软件会按其密钥库名称标识它。可以定义该插件以接受选项。此过程包括如何从 KMF 中删除插件。

1. 安装插件。

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

其中

keystore-name 为您提供的密钥库指定一个唯一名称。

path-to-plugin 指定 KMF 插件的共享库对象的完整路径。

option-string 指定共享库对象的可选参数。

2. 列出插件。

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

3. 要删除插件，请先卸载该插件，然后检验是否已删除。

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

例 4-8 使用选项调用 KMF 插件

在以下示例中，管理员将 KMF 插件存储在站点专用目录中。定义该插件以接受 `debug` 选项。管理员添加插件，然后验证该插件是否已安装。

```
# /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
```

该插件现在在调试模式下运行。

安全词汇表

Access Control List, ACL（访问控制列表）	与传统的 UNIX 文件保护相比，访问控制列表 (access control list, ACL) 可提供更为精细的文件安全性。例如，通过 ACL 可以让组获得对某个文件的读取权限，而仅允许该组中的一个成员获得对该文件的写入权限。
admin principal（管理主体）	名称形式为 *username*`/admin` 的用户主体（如 `jdoe/admin`）。与一般用户主体相比，管理主体可以拥有更多特权（例如，可以更改策略）。另请参见 principal name（主体名称）和 user principal（用户主体）。
AES	Advanced Encryption Standard（高级加密标准）。一种对称的 128 位块数据加密技术。美国政府在 2000 年 10 月采用该种算法的 Rijndael 变体作为其加密标准。AES 从而取代了 user principal（用户主体）加密方法成为政府的加密标准。
algorithm（算法）	加密算法。这是一种确立的递归计算过程，用于对输入执行加密或散列操作。
application server（应用服务器）	请参见 network application server（网络应用服务器）。
asynchronous audit event（异步审计事件）	异步事件在系统事件中属于少数。这些事件不与任何进程关联，因此没有任何进程可供阻塞并在以后唤醒。例如，初始系统引导和 PROM 进入和退出事件就属于异步事件。
audit files（审计文件）	二进制审计日志。审计文件单独存储在一个审计文件系统中。
audit policy（审计策略）	决定要记录的审计事件的全局设置和按用户设置。通常，应用于审计服务的全局设置会影响审计迹所包括的可选信息。`cnt` 和 `ahlt` 这两个设置会影响系统在填充审计队列时执行的操作。例如，审计策略可能要求每条审计记录都包含一个序列号。
audit trail（审计迹）	来自所有主机的所有审计文件的集合。
authenticated rights profile（需要验证权限配置文件）	rights profile（权限配置文件），需要指定的用户或角色键入口令后才能执行配置文件中的操作。此行为类似于 `sudo` 行为。口令的有效期可配置。

authentication（验证）	验证主体所声明的身份的过程。
authenticator（验证者）	当客户机从 KDC 请求票证以及从服务器请求服务时，会传递验证者。这些验证者包含使用仅对客户机和服务器公开的会话密钥所生成的信息，这些信息可以作为最新来源进行检验，从而表明事务是安全的。验证者可与票证一起使用来验证用户主体。验证者中包括用户的主体名称、用户主机的 IP 地址，以及时间戳。与票证不同，验证者只能使用一次，通常在请求访问服务时使用。验证者是使用特定客户机和服务器的会话密钥进行加密的。
authorization（授权）	1. 在 Kerberos 中，是指决定主体是否可以使用服务，允许主体访问哪些对象，以及可对每个对象执行的访问操作类型的过程。 2. 在用户权限管理中，是指可以分配给角色或用户（或在权限配置文件中嵌入）的一种权限，允许执行安全策略原本禁止的某类操作。授权在用户应用程序级别（而不是内核级别）实施。
basic set（基本特权集）	登录时为用户进程指定的特权集。在未修改的系统上，每个用户的初始可继承特权集等同于登录时获取的基本特权集。
Blowfish	一种对称块加密算法，它采用 32 位到 448 位的可变长度密钥。其作者 Bruce Schneier 声称 Blowfish 已针对密钥不经常更改的应用程序进行优化。
certificate（证书）	公钥证书是一组对公钥值进行编码的数据，含有关于生成证书的信息，例如名称和签名人、证书的散列或校验和以及散列的数字签名。这些值共同构成了证书。数字签名可以确保证书不被修改。 有关更多信息，请参见 key（密钥）。
client principal（客户机主体）	(RPCSEC_GSS API) 是指使用受 RPCSEC_GSS 保护的网络服务的客户机（用户或应用程序）。客户机主体名称将以 `rpc_gss_principal_t` 结构的形式进行存储。
client（客户机）	狭义上讲，是指代表用户使用网络服务的进程，例如，使用 `rlogin` 的应用程序。在某些情况下，服务器本身即可是其他某个服务器或服务的客户机。 广义上讲，是指 a) 接收 Kerberos 凭证的主机，以及 b) 使用由服务器提供的服务的主机。 非正式地讲，是指使用服务的主体。
clock skew（时钟相位差）	所有参与 Kerberos 验证系统的主机上的内部系统时钟可以相差的最大时间量。如果任意两台参与主机之间的时间偏差超过了时钟相位差，则请求会被拒绝。可以在 `krb5.conf` 文件中指定时钟相位差。
confidentiality（保密）	请参见 privacy（保密性）。

consumer（使用者）	在 Oracle Solaris 的加密框架功能中，使用者是指使用提供者提供的加密服务的用户。使用者可以是应用程序、最终用户或内核操作。例如，Kerberos、IKE 和 IPsec 便属于使用者。有关提供者的示例，请参见 provider（提供者）。
credential cache（凭证高速缓存）	包含从 KDC 接收的凭证的存储空间（通常为文件）。
credential（凭证）	包括票证及匹配的会话密钥的信息软件包。用于验证主体的身份。另请参见 ticket（票证）和 session key（会话密钥）。
cryptographic algorithm（密码算法）	请参见 algorithm（算法）。
DES	Data Encryption Standard（数据加密标准）。一种对称密钥加密方法，开发于 1975 年，1981 年由 ANSI 标准化为 ANSI X.3.92。DES 使用 56 位密钥。
device allocation（设备分配）	用户级别的设备保护。设备分配强制规定一次只能由一个用户独占使用一台设备。重用设备之前，将清除设备数据。可以使用授权来限制允许分配设备的用户。
device policy（设备策略）	内核级别的设备保护。设备策略在设备上作为两个特权集实现。一个特权集控制对设备的读取权限，另一个特权集控制对设备的写入权限。另请参见 policy（策略）。
Diffie-Hellman protocol（Diffie-Hellman 协议）	也称为公钥密码学。Diffie 和 Hellman 于 1976 年开发的非对称密钥一致性协议。使用该协议，两个用户可以在以前没有任何密钥的情况下通过不安全的介质交换密钥。Diffie-Hellman 由 Kerberos 使用。
digest（摘要）	请参见 message digest（消息摘要）。
DSA	Digital Signature Algorithm（数字签名算法）。一种公钥算法，采用大小可变（512 位到 4096 位）的密钥。美国政府标准 DSS 可达 1024 位。DSA 的输入依赖于 SHA1。
ECDSA	Elliptic Curve Digital Signature Algorithm（椭圆曲线数字签名算法）。一种基于椭圆曲线数学运算的公钥算法。在生成相同长度的签名时，所需的 ECDSA 密钥大小明显小于 DSA 公钥大小。
effective set（有效特权集）	当前对进程有效的特权集。
flavor（特性）	以前，*security flavor*（安全特性）和 *authentication flavor*（验证特性）具有相同的含义，都是表示验证类型 (AUTH_UNIX, AUTH_DES, AUTH_KERB) 的特性。RPCSEC_GSS 也是一种安全特性，虽然它除了验证之外还提供完整性和保密性服务。
forwardable ticket（可转发票证）	一种票证，可供客户机在不需要完成远程主机上的完整验证过程的情况下用于请求此主机票证。例如，如果用户 `david` 登录到用户

	jennifer 的计算机时获取了一张可转发票证，则 david 不必获取新的票证（从而对自身进行重新验证）即可登录到自己的计算机。另请参见 proxiable ticket（可代理票证）。
FQDN	Fully qualified domain name（全限定域名）。例如，central.example.com（与简单的 denver 相对）。
GSS-API	Generic Security Service Application Programming Interface（通用安全服务应用编程接口）。为各种模块化安全服务（包括 Kerberos 服务）提供支持的网络层。GSS-API 可用于安全验证服务、完整性服务和保密性服务。另请参见 authentication（验证）、integrity（完整性）和 privacy（保密性）。
hardening（强化）	为了删除主机中固有的安全漏洞而对操作系统的缺省配置进行的修改。
hardware provider（硬件提供者）	在 Oracle Solaris 的加密框架功能中，是指设备驱动程序及其硬件加速器。硬件提供者使计算机系统不必执行开销很大的加密操作，从而可释放 CPU 资源以用于其他用途。另请参见 provider（提供者）。
host principal（主机主体）	服务主体的一个特定实例，其中将主体（由主名称 host 表示）设置为提供一系列网络服务，如 ftp、rcp 或 rlogin。例如，host/central.example.com@EXAMPLE.COM 便是一个主机主体。另请参见 server principal（服务器主体）。
host（主机）	可通过网络进行访问的系统。
inheritable set（可继承特权集）	进程可以通过调用 exec 而继承的特权集。
initial ticket（初始票证）	直接颁发（即，不基于现有的票证授予票证）的票证。某些服务（如用于更改口令的应用程序）可能需要将票证标记为 initial，以便使其自身确信客户机知晓其密钥。这种保证非常重要，因为初始票证表明客户机最近已进行了自我验证（而并非依赖于存在时间可能较长的票证授予票证）。
instance（实例）	实例是主体名称的第二个部分，用于限定主体的主名称。对于服务主体，实例是必需的。实例就是主机的全限定域名，例如 host/central.example.com。对于用户主体，实例是可选的。但是请注意，jdoe 和 jdoe/admin 都是唯一的主体。另请参见 primary（主）、principal name（主体名称）、service principal（服务主体）和 user principal（用户主体）。
integrity（完整性）	一种安全服务，除了用于用户验证之外，还用于通过加密校验和来验证传输数据的有效性。另请参见 authentication（验证） 和 privacy（保密性）。
invalid ticket（无效票证）	尚未成为可用票证的以后生效的票证。应用服务器将拒绝无效票证，直到此票证生效为止。要使无效票证生效，必须在其开始时间已过

后，由客户机通过 TGS 请求将其提供给 KDC，同时设置 VALIDATE 标志。另请参见 postdated ticket（以后生效的票证）。

KDC

Key Distribution Center（密钥分发中心）。具有以下三个 Kerberos V5 组件的计算机：

- 主体和密钥数据库
- 验证服务
- 票证授予服务

每个领域都具有一个主 KDC，并且应该具有一个或多个从 KDC。

Kerberos

是指一种验证服务、此服务所使用的协议或者用于实现此服务的代码。

Oracle Solaris 中的 Kerberos 实现主要基于 Kerberos V5 实现。

虽然在技术方面有所不同，但是在 Kerberos 文档中经常会互换使用 "Kerberos" 和 "Kerberos V5"。

Kerberos（也可写成 Cerberus）在希腊神话中是指守护地狱之门的三头凶悍猛犬。

Kerberos policy（Kerberos 策略）

管理 Kerberos 服务中口令的使用的规则集合。这些策略可以控制主体的访问权限或票证参数（如生命周期）。

key（密钥）

1. 通常是指以下两种主要密钥类型之一：

- 对称密钥 – 与解密密钥相同的加密密钥。对称密钥用于对文件进行加密。
- 非对称密钥或公钥 – 在公钥算法（如 Diffie-Hellman 或 RSA）中使用的密钥。公钥包括仅对一个用户公开的私钥、服务器或通用资源所使用的公钥，以及包含这两者的私钥/公钥对。私钥 (private key) 也称为密钥 (secret key)。公钥也称为共享密钥或公用密钥。

2. 密钥表文件中的项（主体名称）。另请参见 keytab file（密钥表文件）。

3. 在 Kerberos 中，是指加密密钥，此类密钥分为以下三种类型：

- 私钥 – 由主体和 KDC 共享并在系统范围之外分发的加密密钥。另请参见 private key（私钥）。
- 服务密钥 – 此密钥与私钥的用途相同，但由服务器和服务使用。另请参见 service key（服务密钥）。
- 会话密钥 – 在两个主体之间使用的临时加密密钥，其生命周期仅限于单个登录会话的持续时间。另请参见 session key（会话密钥）。

keystore（密钥库）	密钥库包含用于应用程序检索的口令、口令短语、证书，以及其他验证对象。密钥库可特定于一种技术，或特定于多个应用程序使用的一个位置。
keytab file（密钥表文件）	包含一个或多个密钥（主体）的密钥表文件。主机或服务使用密钥表文件的方式与用户使用口令的方式大致相同。
kvno	Key version number（密钥版本号）。按照生成顺序跟踪特定密钥的序列号。kvno 最高则表示密钥最新。
least privilege（最小特权）	一种安全模型，该模型仅向指定进程提供超级用户功能的某个子集。最小特权模型为一般用户指定可以用来执行个人管理任务（如挂载文件系统和更改文件的所有权）的足够特权。另一方面，仅使用完成该任务所需的特权运行进程，而不是使用超级用户的完全功能模式（即所有特权）。对非 root 用户而言，可以包含由于编程错误而导致的损坏（如缓冲区溢出），该用户对重要功能（如读取或写入受保护的系统文件或停止计算机）没有访问权限。
limit set（限制特权集）	对哪些特权可用于进程及其子进程的外部限制。
MAC	1. 请参见 message authentication code, MAC（消息验证代码）。 2. 也称为标签设置操作。在政府安全术语中，MAC 是指 Mandatory Access Control（强制访问控制）。例如，Top Secret（绝密）和 Confidential（机密）之类的标签便是 MAC。MAC 与 DAC 相对，后者是指 Discretionary Access Control（自主访问控制）。例如，UNIX 权限便是一个 DAC。 3. 在硬件中，是指 LAN 中的唯一系统地址。如果系统位于以太网中，则 MAC 是指以太网地址。
master KDC（主 KDC）	每个领域中的主要 KDC，包括 Kerberos 管理服务器 `kadmind`，以及验证和票证授予守护进程 `krb5kdc`。每个领域至少都必须具有一个主 KDC，可以具有多个 KDC 副本或从 KDC，这些 KDC 为客户机提供验证服务。
MD5	一种重复加密散列函数，用于进行消息验证（包含数字签名）。该函数于 1991 年由 Rivest 开发。其使用已过时。
mechanism（机制）	1. 指定加密技术以实现数据验证或保密的软件包。例如：Kerberos V5、Diffie-Hellman 公钥。 2. 在 Oracle Solaris 的加密框架功能中，是指用于特殊用途的算法的实现。例如，应用于验证的 DES 机制（如 CKM_DES_MAC）与应用于加密的 DES 机制（如 CKM_DES_CBC_PAD）不同。
message authentication code, MAC（消息验证代码）	MAC 可确保数据的完整性，并验证数据的来源。MAC 不能防止窃听。

message digest（消息摘要）	消息摘要是从消息中计算所得的散列值。此散列值几乎可唯一地标识消息。摘要对检验文件的完整性非常有用。
minimization（最小安装）	运行服务器所需的最小操作系统安装。不安装与服务器操作不直接相关的任何软件，或者在安装之后即删除。
name service scope（名称服务范围）	允许角色在其中执行操作的范围，即，由指定的命名服务（如 NIS 或 LDAP）提供服务的单个主机或所有主机。
network application server（网络应用服务器）	提供网络应用的服务器，如 `ftp`。一个领域可以包含多个网络应用服务器。
network policies（网络策略）	网络实用程序为了保护网络通信而配置的设置。有关网络安全性的信息，请参见《在 Oracle Solaris 11.2 中确保网络安全 》。
nonattributable audit event（无归属审计事件）	无法确定其触发者的审计事件，如 AUE_BOOT 事件。
NTP	Network Time Protocol（网络时间协议）。由特拉华大学开发的软件，可用于在网络环境中管理准确时间或网络时钟同步，或者同时管理这两者。可以使用 NTP 在 Kerberos 环境中维护时钟相位差。另请参见 clock skew（时钟相位差）。
PAM	Pluggable Authentication Module（可插拔验证模块）。一种框架，允许使用多种验证机制而不必重新编译运行这些机制的服务。PAM 可用于在登录时初始化 Kerberos 会话。
passphrase（口令短语）	一种短语，用于验证某个私钥是否是由口令短语用户创建。理想的口令短语应包含 10-30 个字符，请混合使用字母和数字字符，并且避免简单的文本结构和名称。使用私钥对通信执行加密和解密操作时，系统会提示您提供口令短语进行验证。
password policy（口令策略）	可用于生成口令的加密算法，还可以指与口令有关的更普遍的问题，如必须对口令进行更改的频率，允许的口令尝试次数以及其他安全注意事项。安全策略需要口令。口令策略可能要求使用 AES 算法对口令进行加密，并可能对口令强度提出进一步要求。
permitted set（允许特权集）	可供进程使用的特权集。
policy for public key technologies（公钥技术的策略）	在密钥管理框架 (Key Management Framework, KMF) 中，所实现的策略是管理证书的使用。KMF 策略数据库可以对由 KMF 库管理的密钥和证书的使用施加约束。
policy in the Cryptographic Framework（加密框架中的策略）	在 Oracle Solaris 的加密框架功能中，所实现的策略是禁用现有的加密机制。从而使这些机制不可使用。加密框架中的策略可能会阻止使用提供者（如 DES）提供的特殊机制，如 `CKM_DES_CBC`。
policy（策略）	一般而言，是指影响或决定决策和操作的操作规划或操作过程。对于计算机系统，策略通常表示安全策略。站点的安全策略是规则集合和相关措施，可用于定义所处理信息的敏感度并防止信息受到未经授权

	的访问。例如，安全策略可能要求对系统进行审计，必须分配设备才能使用，以及每六周必须更改一次口令。 有关在 Oracle Solaris OS 特定区域中实施策略的信息，请参见 audit policy（审计策略）、policy in the Cryptographic Framework（加密框架中的策略）、device policy（设备策略）、Kerberos policy（Kerberos 策略）、password policy（口令策略）和rights policy（权限策略）。
postdated ticket（以后生效的票证）	以后生效的票证直到创建之后的某一指定时间才能开始生效。此类票证对于计划在深夜运行的批处理作业等情况非常有用，因为在运行批处理作业之前无法使用该票证（即使被盗）。颁发以后生效的票证时，将以 `invalid` 状态颁发该票证，并在出现以下情况之前一直保持此状态：a) 票证开始时间已过，并且 b) 客户机请求 KDC 进行验证。通常，以后生效的票证在票证授予票证的截止时间之前会一直有效。但是，如果将以后生效的票证标记为 `renewable`，则通常会将其生命周期设置为等于票证授予票证的整个生命周期的持续时间。另请参见 invalid ticket（无效票证）和 renewable ticket（可更新票证）。
primary（主）	主体名称的第一部分。另请参见 instance（实例）、principal name（主体名称）和 realm（领域）。
principal name（主体名称）	1. 主体的名称，格式为 *primary/instance@REALM*。另请参见 instance（实例）、primary（主）和 realm（领域）。 2.(RPCSEC_GSS API) 请参见 client principal（客户机主体）和 server principal（服务器主体）。
principal（主体）	1. 参与网络通信并且具有唯一名称的客户机/用户或服务器/服务实例。Kerberos 事务涉及主体之间（服务主体与用户主体）或主体与 KDC 之间的交互。换言之，主体是 Kerberos 可为其指定票证的唯一实体。另请参见 principal name（主体名称）、service principal（服务主体）和 user principal（用户主体）。 2.(RPCSEC_GSS API) 请参见 client principal（客户机主体）和 server principal（服务器主体）。
principle of least privilege（最小特权原则）	请参见least privilege（最小特权）。
privacy（保密性）	一种安全服务，其中传输的数据加密之后才会发送。保密性还包括数据完整性和用户验证。另请参见 authentication（验证）、integrity（完整性）和 service（服务）。
private key（私钥）	为每个用户主体提供的密钥，并且只对主体的用户和 KDC 公开。对于用户主体，密钥基于用户的口令。另请参见 key（密钥）。
private-key encryption（私钥加密）	采用私钥加密时，发送者和接收者使用相同的加密密钥。另请参见 public-key encryption（公钥加密）。

privilege escalation（特权升级）

可以访问在所指定权限（包括覆盖缺省设置的权限）允许的资源范围以外的资源。特权升级的结果是某个进程可以执行未经授权的操作。

privilege model（特权模型）

计算机系统上比超级用户模型更为严格的安全模型。在特权模型中，进程需要具有相应的特权才能运行。系统管理可以分为多个独立的部分，这些部分基于管理员在其进程中所具有的特权。可以将特权指定给管理员的登录过程。或者，可以指定特权只对特定命令有效。

privilege set（特权集）

特权的集合。每个进程都有四个特权集，用于确定进程是否可以使用特定特权。请参见 limit set（限制特权集）、effective set（有效特权集）、permitted set（允许特权集）和 inheritable set（可继承特权集）。

此外，特权的 basic set（基本特权集）是指登录时为用户进程指定的特权集合。

privilege-aware（特权识别）

在其代码中启用和禁用特权的程序、脚本和命令。在生产环境中，启用的特权必须提供给进程，例如，通过要求程序的用户使用将特权添加到程序中的权限配置文件。有关特权的完整说明，请参见 `privileges(5)` 手册页。

privilege（特权）

1. 通常是指在某个计算机系统上执行一般用户所无法执行的操作的能力。超级用户特权是向超级用户授予的所有 rights（权限）。特权用户或特权应用程序是指获得了额外权限的用户或应用程序。

2. Oracle Solaris 系统中的进程具有的独立权限。与 `root` 相比，特权可提供更为精细的进程控制。特权是在内核中定义和实施的。特权也称为进程特权或内核特权。有关特权的完整说明，请参见 `privileges(5)` 手册页。

privileged application（特权应用程序）

可以覆盖系统控制的应用程序。该应用程序可以检查安全属性（如特定的 UID、GID、授权或特权）。

privileged user（特权用户）

计算机系统上为其指定的权限高于一般用户权限的用户。另请参见 trusted users（可信用户）。

profile shell（配置文件 shell）

在权限管理中，角色（或用户）可通过该 shell 从命令行运行指定给角色权限配置文件的任何特权应用程序。配置文件 shell 版本与系统上可用的 shell 对应（例如 `bash` 的 `pfbash` 版本）。

provider（提供者）

在 Oracle Solaris 的加密框架功能中，是指为使用者提供的加密服务。例如，PKCS #11 库、内核加密模块和硬件加速器便是提供者。提供者可插入到加密框架中，因此也称为插件。有关使用者的示例，请参见 consumer（使用者）。

proxiable ticket（可代理票证）

可供服务用于代表客户机执行客户机操作的票证。因此，可以说服务充当客户机的代理。使用该票证，服务便可具有客户机的身份。服务

	可以使用可代理票证来获取其他服务的服务票证，但是不能获取票证授予票证。可代理票证与可转发票证之间的区别在于可代理票证只对单项操作有效。另请参见 forwardable ticket（可转发票证）。
public object（公共对象）	root 用户所拥有且全局可读的文件，如 /etc 目录中的任何文件。
public-key encryption（公钥加密）	一种加密方案，其中每个用户都有两个密钥：一个是公钥，一个是私钥。采用公钥加密时，发送者使用接收者的公钥对消息进行加密，而接收者则使用私钥对其进行解密。Kerberos 服务是一种私钥系统。另请参见 private-key encryption（私钥加密）。
QOP	Quality of Protection（保护质量）。用于选择与完整性服务或保密性服务结合使用的加密算法的参数。
RBAC	Role-based access control（基于角色的访问控制），Oracle Solaris 的一项用户权限管理功能。请参见 rights（权限）。
RBAC policy（RBAC 策略）	请参见 rights policy（权限策略）。
realm（领域）	1. 由单个 Kerberos 数据库以及一组密钥分发中心 (Key Distribution Center, KDC) 提供服务的逻辑网络。 2. 主体名称的第三部分。对于主体名称 jdoe/admin@CORP.EXAMPLE.COM，领域为 CORP.EXAMPLE.COM。另请参见 principal name（主体名称）。
reauthentication（重新验证）	执行计算机操作需要提供口令。通常，sudo 操作要求重新验证。需要验证权限配置文件可包含需要重新验证的命令。请参见 authenticated rights profile（需要验证权限配置文件）。
relation（关系）	在 kdc.conf 或 krb5.conf 文件中定义的配置变量或关系。
renewable ticket（可更新票证）	由于票证的生命周期过长会存在安全风险，因此可以将票证指定为 renewable。可更新票证有两个截止时间：a) 票证的当前实例的截止时间，b) 任意票证的最长生命周期。如果客户机需要继续使用某票证，则可在首次失效之前更新此票证。例如，某个票证的有效期为 1 小时，所有票证的最长生命周期为 10 小时。如果持有票证的客户机希望保留此票证的时间长于 1 小时，则必须更新此票证。当某个票证达到最长票证生命周期时，便会自动到期，并且无法更新。
rights policy（权限策略）	与命令关联的安全策略。当前，solaris 是 Oracle Solaris 的有效策略。solaris 策略可识别特权和扩展特权策略、授权及 setuid 安全属性。
rights profile（权限配置文件）	也称为配置文件。指的是可以分配给角色或用户的安全设置覆盖值的集合。权限配置文件可包括授权、特权、具有安全属性的命令和称为补充配置文件的其他权限配置文件。

rights（权限）	对超级用户模型（管理员对系统要么具有全部控制权要么毫无控制权）的替代。通过用户权限管理和进程权限管理，组织可划分超级用户的特权并将其指定给用户或角色。Oracle Solaris 中的权限实施方式有内核特权、授权和以特定 UID 或 GID 运行进程的能力。可在 rights profile（权限配置文件）和 role（角色）中收集权限。
role（角色）	一种用于运行特权应用程序的特殊身份，仅指定用户才能承担此身份。
RSA	获取数字签名和公钥密码系统的方法。该方法于 1978 年首次由其开发者 Rivest、Shamir 和 Adleman 介绍。
scan engine（扫描引擎）	第三方应用程序，驻留在外部主机上，可检查文件中是否含有已知病毒。
SEAM	这是 Solaris 系统上的 Kerberos 初始版本的产品名。该产品基于麻省理工学院开发的 Kerberos V5 技术。SEAM 现在称为 Kerberos 服务。其特性与 MIT 版本仍稍有不同。
secret key（密钥）	请参见 private key（私钥）。
Secure Shell（安全 Shell）	一种特殊协议，用于在不安全的网络中进行安全远程登录并提供其他安全网络服务。
security attributes（安全属性）	是指当超级用户以外的用户运行管理命令时，可使此命令成功执行的安全策略覆盖项。在超级用户模型中，`setuid root` 和 `setgid` 程序都是安全属性。将这些属性应用于某命令时，此命令便会成功执行，而与运行它的用户无关。在 privilege model（特权模型）中，内核特权及其他 rights（权限）会将 `setuid root` 程序替换为安全属性。特权模型与超级用户模型兼容，因为特权模型也可将 `setuid` 和 `setgid` 程序识别为安全属性。
security flavor（安全特性）	请参见 flavor（特性）。
security mechanism（安全机制）	请参见 mechanism（机制）。
security policy（安全策略）	请参见 policy（策略）。
security service（安全服务）	请参见 service（服务）。
seed（种子）	用于生成随机数的数字起动器。当起动器来自随机源时，种子称为随机种子。
separation of duty（职责分离）	least privilege（最小特权）的部分概念。职责分离可阻止一个用户执行或批准完成事务的所有操作。例如，在 RBAC 中，可以将登录用户的创建与安全覆盖的指定分隔开来。一个角色创建该用户。另一个

	角色可以将安全属性（如权限配置文件、角色和特权）指定给现有用户。
server principal（服务器主体）	(RPCSEC_GSS API) 提供服务的主体。服务器主体以 *service@host* 形式的 ASCII 字符串进行存储。另请参见 client principal（客户机主体）。
server（服务器）	为网络客户机提供资源的主体。 例如，如果通过 `ssh` 远程登录到系统 `central.example.com`，则该系统便是提供 `ssh` 服务的服务器。另请参见 service principal（服务主体）。
service key（服务密钥）	由服务主体和 KDC 共享，并在系统范围之外分发的加密密钥。另请参见 key（密钥）。
service principal（服务主体）	为一项或多项服务提供 Kerberos 验证的主体。对于服务主体，主名称是服务的名称（如 `ftp`），其实例是提供服务的系统的全限定主机名。另请参见 host principal（主机主体）和 user principal（用户主体）。
service（服务）	1. 通常由多台服务器提供给网络客户机的资源。例如，如果通过 `rlogin` 远程登录到计算机 `central.example.com`，则该计算机便是提供 `rlogin` 服务的服务器。 2. 除验证之外，还提供其他保护级别的安全服务（完整性或保密性）。另请参见 integrity（完整性）和 privacy（保密性）。
session key（会话密钥）	由验证服务或票证授予服务生成的密钥。生成会话密钥的目的是在客户机与服务之间提供安全事务。会话密钥的生命周期仅限于单个登录会话的持续时间。另请参见 key（密钥）。
SHA1	Secure Hashing Algorithm（安全散列算法）。该算法可以针对长度小于 2^{64} 的任何输入进行运算，以生成消息摘要。SHA1 算法是 DSA 的输入。
single-system image（单系统映像）	单系统映像用在 Oracle Solaris 审计中来描述使用相同命名服务的一组受审计系统。这些系统将其审计记录发送给某个中心审计服务器，可在该服务器中对记录进行比较，就像这些记录来自一个系统一样。
slave KDC（从 KDC）	主 KDC 的副本，可以执行主 KDC 的大多数功能。每个领域通常都具有若干个从 KDC（但仅有一个主 KDC）。另请参见 KDC 和 master KDC（主 KDC）。
software provider（软件提供者）	在 Oracle Solaris 的加密框架功能中，是指提供加密服务的内核软件模块或 PKCS #11 库。另请参见 provider（提供者）。
stash file（存储文件）	存储文件包含 KDC 主密钥的已加密副本。当重新引导服务器以便在 KDC 启动 `kadmind` 和 `krb5kdc` 进程之前自动验证 KDC 时，将使用此主密钥。由于存储文件中包含主密钥，因此，应该保证存储文件及其任何备份的安全。如果加密受到威胁，则可以使用此密钥来访问或修改 KDC 数据库。

superuser model（超级用户模型）	计算机系统上的典型 UNIX 安全模型。在超级用户模型中，管理员对系统要么具有全部的控制权要么毫无控制权。通常，为了管理计算机，用户可成为超级用户 (`root`)，并可执行所有管理活动。
synchronous audit event（同步审计事件）	审计事件中的大多数事件属于同步审计事件。这些事件与系统中的某个进程关联。与某个进程关联的无归属事件属于同步事件，如失败的登录。
TGS	Ticket-Granting Service（票证授予服务）。负责颁发票证的那部分 KDC。
TGT	Ticket-Granting Ticket（票证授予票证）。由 KDC 颁发的票证，客户机可使用此票证来请求其他服务的票证。
ticket file（票证文件）	请参见 credential cache（凭证高速缓存）。
ticket（票证）	用于安全地将用户身份传递给服务器或服务的信息包。一个票证仅对一台客户机以及某台特定服务器上的一项特殊服务有效。票证包含服务的主体名称、用户的主体名称、用户主机的 IP 地址、时间戳以及定义此票证生命周期的值。票证是通过由客户机和服务使用的随机会话密钥创建的。一旦创建了票证，便可重复使用此票证，直到其到期为止。票证与新的验证者同时出现时，仅用于验证客户机。另请参见 authenticator（验证者）、credential（凭证）、service（服务）和 session key（会话密钥）。
trusted users（可信用户）	指的是您决定允许其在一定信任级别下执行管理任务的用户。通常，管理员先为可信用户创建登录名，并分配与此类用户的信任级别和能力匹配的管理权限。之后，这些用户便能帮助配置和维护系统。此类用户也称为特权用户。
user principal（用户主体）	属于某个特定用户的主体。用户主体的主名称是用户名，其可选实例是用于说明相应凭证预期用法的名称（例如 `jdoe` 或 `jdoe/admin`）。也称为用户实例。另请参见 service principal（服务主体）。
virtual private network, VPN（虚拟专用网络）	通过使用加密和隧道连接公共网络上的用户来提供安全通信的网络。

索引

T

V

W

X

Y

Z