功能

本页中包含与 Android 6.0 及更高版本中 Keystore (/security/keystore/index.html) 加密功能相关的信息。

注意：标记和功能以 Keymaster 3 样式进行编写。要了解详情，请参阅 HIDL 概览 (/security/keystore/index#hidl-overview)。

加密基元

Keystore 能够提供以下类别的操作：

- 生成密钥
- 导入和导出不对称密钥（无密钥封装）
- 导入原始对称密钥（无密钥封装）
- 使用适当的填充模式进行不对称加密和解密
- 使用摘要和适当的填充模式进行不对称签名和验证
- 以适当模式（包括 AEAD 模式）进行对称加密和解密
- 生成和验证对称消息验证码

生成或导入密钥时，系统会指定协议元素（例如，用途、模式和填充，以及访问控制限制 (#key_access_control)），这些元素会永久绑定到相应密钥，以确保无法以任何其他方式使用相应密钥。

除了上面列出的操作外，Keymaster 实现还提供另外一项服务，即随机数生成服务，但该服务并不作为 API 进行提供。该服务仅供在内部使用，用于生成密钥、初始化矢量 (IV)、随机填充，以及其他需要具有随机性的安全协议元素。

必要基元

所有 Keymaster 实现都提供：

- RSA (http://en.wikipedia.org/wiki/RSA_(cryptosystem))
    - 支持 2048 位、3072 位和 4096 位密钥

- 支持公开指数 F4 (2^16+1)
- RSA 签名所需的填充模式：
  - RSASSA-PSS (`PaddingMode::RSA_PSS`)
  - RSASSA-PKCS1-v1_5 (`PaddingMode::RSA_PKCS1_1_5_SIGN`)
- RSA 签名所需的摘要模式：
  - SHA-256
- RSA 加密/解密所需的填充模式：
  - 无填充
  - RSAES-OAEP (`PaddingMode::RSA_OAEP`)
  - RSAES-PKCS1-v1_5 (`PaddingMode::RSA_PKCS1_1_5_ENCRYPT`)

- ECDSA (http://en.wikipedia.org/wiki/Elliptic_Curve_DSA)
  - 支持 224 位、256 位、384 位和 521 位密钥，分别使用 NIST P-224、P-256、P-384 和 P-521 曲线
  - ECDSA 所需的摘要模式：
    - 无摘要（已弃用，将于日后移除）
    - SHA-256
- AES (http://en.wikipedia.org/wiki/Advanced_Encryption_Standard)
  - 支持 128 位和 256 位密钥
  - CBC (http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher-block_chaining_.28CBC.29)、CTR、ECB 和 GCM。GCM 实现不允许使用少于 96 位的标记，也不允许使用 96 位以外的随机数长度。
  - CBC 和 ECB 模式支持填充模式 `PaddingMode::NONE` 和 `PaddingMode::PKCS7`。采用“无填充”时，如果输入的不是分块大小的倍数，CBC 或 ECB 模式的加密会失败。
- HMAC (http://en.wikipedia.org/wiki/Hash-based_message_authentication_code) SHA-256 (http://en.wikipedia.org/wiki/SHA-2)，其中任意密钥均不得短于 32 个字节。

对于 Keymaster 实现，强烈建议提供 SHA1，以及 SHA2 系列的其他成员（SHA-224、SHA384 和 SHA512）。如果硬件 Keymaster 实现未提供这些内容，则 Keystore 会在软件中

提供它们。

此外，为了实现与其他系统的互用性，还建议提供以下基元：

- 适用于 RSA 的较小密钥大小
- 适用于 RSA 的任意公开指数

密钥访问控制

如果攻击者可以随意使用在任何情况下都无法从设备获取的基于硬件的密钥，那么此类密钥将无法提供太多的安全性（尽管它们比可被窃取的密钥更安全）。因此，Keystore 强制执行访问控制至关重要。

访问控制指的是由“标记/值”对组成的“授权列表”。授权标记是 32 位整数，其值有多种类型。有些标记可以重复使用，以指定多个值。某个标记是否可重复使用是在关于该标记的文档(/security/keystore/tags)中指定的。密钥创建好后，调用程序会指定一个授权列表。Keymaster 实现使用的底层 Keystore 会修改该列表，以指定一些额外的信息（例如，密钥是否有防回滚保护），并且会返回一个“最终”授权列表（编码到返回的密钥 Blob 中）。如果最终授权列表被修改了，那么任何尝试使用相应密钥进行任何加密操作的行为都会失败。

对于 Keymaster 2 及更早版本，枚举 `keymaster_authorization_tag_t` 中定义了一组可能的标记，这组标记永远保持不变（不过可进行扩展）。 这些标记的名称带有 `KM_TAG` 前缀。标记 ID 的前四位用于指明类型。

Keymaster 3 将 `KM_TAG` 前缀改为 `Tag::`。

可能的类型包括：

`ENUM`：很多标记的值都是在枚举中定义的。例如，`TAG::PURPOSE` 的可能值是在枚举 `keymaster_purpose_t` 中定义的。

`ENUM_REP`：与 `ENUM` 相同，不过此标记可在授权列表中重复使用。重复使用此标记表明有多个已获授权的值。例如，加密密钥可能具有 `KeyPurpose::ENCRYPT` 和 `KeyPurpose::DECRYPT`。

`UINT`：32 位未签名整数。例如：`TAG::KEY_SIZE`

`UINT_REP`：与 `UINT` 相同，不过此标记可在授权列表中重复使用。重复使用此标记表明有多个已获授权的值。

`ULONG`：64 位未签名整数。例如：`TAG::RSA_PUBLIC_EXPONENT`

ULONG_REP：与 **ULONG** 相同，不过此标记可在授权列表中重复使用。重复使用此标记表明有多个已获授权的值。

DATE：日期/时间值，以距 1970 年 1 月 1 日的毫秒数表示。 例如：**TAG::PRIVKEY_EXPIRE_DATETIME**

BOOL：true 或 false。对于 **BOOL** 类型的标记，如果不存在则被视为“false”，如果存在则被视为“true”。例如：**TAG::ROLLBACK_RESISTANT**

BIGNUM：任意长度的整数，以字节数数组表示（采用大端字节序）。例如：**TAG::RSA_PUBLIC_EXPONENT**

BYTES：一系列字节数。例如：**TAG::ROOT_OF_TRUST**

硬件与软件强制执行

并非所有安全硬件实现都包含相同的功能。为了支持多种方法，Keymaster 会对安全域和非安全域访问控制强制执行（分别称为硬件强制执行和软件强制执行）加以区分。

所有实现：

- 强制执行所有授权完全匹配（不是强制执行所有授权）。密钥 Blob 中的授权列表与密钥生成期间返回的授权完全匹配（包括顺序）。如有任何不匹配，都会导致进行错误诊断。
- 声明语义值会被强制执行的授权。

用于声明由硬件强制执行的授权的 API 机制位于 **keymaster_key_characteristics_t** 结构中。它将授权列表划分成两个子列表：**hw_enforced** 和 **sw_enforced**。安全硬件负责根据它可以强制执行的内容在每个子列表中放入适当的值。

此外，Keystore 会实现基于软件强制执行所有授权，无论它们是否由安全硬件强制执行。

让我们以一个不支持密钥过期日期且基于 TrustZone 的实现为例。实现仍可能会创建一个具有过期日期的密钥。该密钥的授权列表将包含具有过期日期的 **TAG::ORIGINATION_EXPIRE_DATETIME** 标记。向 Keystore 发出的密钥特性请求将会在 **sw_enforced** 列表中找到此标记，并且安全硬件不会强制执行过期日期要求。不过，如果尝试在过期日期之后使用该密钥，则会被 Keystore 拒绝。

如果设备随后进行了升级，采用了不支持过期日期的安全硬件，那么密钥特性请求将会在 **hw_enforced** 列表中找到 **TAG::ORIGINATION_EXPIRE_DATETIME**，并且即使以某种方式破坏或规避 Keystore，尝试在过期日期之后使用相应密钥也会失败。

要详细了解如何确定密钥是否受硬件支持，请参阅密钥认证 (/security/keystore/attestation)。

加密消息构建授权

以下标记用于定义使用关联密钥的操作的加密特性：`TAG::ALGORITHM`、`TAG::KEY_SIZE`、`TAG::BLOCK_MODE`、`TAG::PADDING`、`TAG::CALLER_NONCE` 和 `TAG::DIGEST`

`TAG::PADDING`、`TAG::DIGEST` 和 `PaddingMode::BLOCK_MODE` 可重复使用，这意味着可以将多个值与一个密钥相关联，并且要使用的值在操作时指定。

用途

密钥有一组关联的用途，这些用途以一个或多个带有 `TAG::PURPOSE` 标记（用于定义可以如何使用相应密钥）的授权条目表示。这些用途是：

- `KeyPurpose::ENCRYPT`
- `KeyPurpose::DECRYPT`
- `KeyPurpose::SIGN`
- `KeyPurpose::VERIFY`

任意密钥都可以具有这些目的任意组合。请注意，有些组合会带来安全问题。例如，如果某个 RSA 密钥可用于加密和签名，那么能够诱使系统解密任意数据的攻击者就可以利用该密钥来生成签名。

导入和导出

Keymaster 仅支持以 X.509 格式导出公钥，并支持：

- 以未采用密码加密的 DER 编码 PKCS#8 格式导入公钥和私钥对
- 以原始字节形式导入对称密钥

为了确保导入的密钥可与安全生成的密钥区分开来，相应密钥授权列表中会包含 `TAG::ORIGIN`。例如，如果密钥是在安全硬件中生成的，`hw_enforced` 密钥特性列表中将有值为 `KeyOrigin::GENERATED` 的 `TAG::ORIGIN`；如果密钥是导入到安全硬件中的，值将为 `KeyOrigin::IMPORTED`。

用户身份验证

安全的 Keymaster 实现不会实现用户身份验证，但会依赖于其他实现用户身份验证的可信应用。对于这些应用实现的接口，请参阅 Gatekeeper 页面 (/security/authentication/gatekeeper)。

用户身份验证要求是通过两组标记指定的。第一组用于指明哪些用户可以使用相应密钥：

- `TAG::ALL_USERS` 表示所有用户都可以使用相应密钥。如果该标记存在，则 `TAG::USER_ID` 和 `TAG::USER_SECURE_ID` 不存在。
- `TAG::USER_ID` 有一个数字值，用于指定已获授权用户的 ID。请注意，此值是 Android 用户 ID（适用于多用户环境）而非应用 UID，且仅由非安全软件强制执行。如果该标记存在，则 `TAG::ALL_USERS` 不存在。
- `TAG::USER_SECURE_ID` 有一个 64 位数字值，用于指定安全用户 ID。要在安全身份验证令牌中提供该 ID，才能获得使用相应密钥的授权。在重复使用此标记的情况下，只要在安全身份验证令牌中提供了此标记的任何一个值，即可使用相应密钥。

第二组用于指明是否需要对用户进行身份验证以及何时进行验证。如果不存在以下任一标记，但有 `TAG::USER_SECURE_ID`，则表示每次使用相应密钥时均需要经过身份验证。

- `NO_AUTHENTICATION_REQUIRED` 表示无需进行任何用户身份验证，不过仍只有以通过 `TAG::USER_ID` 指定的用户身份运行的应用可以使用相应密钥。
- `TAG::AUTH_TIMEOUT` 是一个数字值，用于指定用户身份验证需要多新（以秒数计）才能授权使用相应密钥。此标记仅适用于私钥/密钥操作。公钥操作不需要进行身份验证。超时会在设备重新启动后失效；设备重新启动后，所有密钥的状态均为“从未经过身份验证”。可将超时设为一个较大的值，以指明每次设备启动后只需进行一次身份验证（2^32 秒约为 136 年；Android 设备的重新启动时间间隔一般不会超过该值）。

客户端绑定

客户端绑定（即将密钥与特定客户端应用相关联）是通过一个可选客户端 ID 和一些可选客户端数据（分别是 `TAG::APPLICATION_ID` 和 `TAG::APPLICATION_DATA`）实现的。Keystore 会将这些值视为不透明 Blob，仅用于确保密钥生成/导入期间存在的 Blob 在每次使用相应密钥时都存在，并且每个字节都完全相同。客户端绑定数据不是由 Keymaster 返回的。调用程序必须知道这些数据，才能使用相应密钥。

此功能未提供给应用。

过期日期

Keystore 支持按日期限制密钥的使用。可以将密钥有效期开始日期和过期日期同密钥相关联，这样一来，如果当前日期/时间不在有效期范围内，Keymaster 会拒绝执行密钥操作。密钥有效期范围是使用 `TAG::ACTIVE_DATETIME`、`TAG::ORIGINATION_EXPIRE_DATETIME` 和 `TAG::USAGE_EXPIRE_DATETIME` 标记指定的。“ORIGINATION”和“USAGE”之间的区别在于使

用相应密钥是为了“生成”新的密文/签名/等，还是“使用”现有密文/签名/等。请注意，此区别未提供给应用。

`TAG::ACTIVE_DATETIME`、`TAG::ORIGINATION_EXPIRE_DATETIME` 和 `TAG::USAGE_EXPIRE_DATETIME` 标记是可选的。如果缺少这些标记，相应密钥会被视为可随时用于解密/验证消息。

由于挂钟时间是由非安全域提供的，因此与过期日期相关的标记不可能位于由硬件强制执行的列表中。如果由硬件强制执行过期日期，将需要安全域以某种方式获取可信时间和数据，例如通过具有可信远程时间服务器的质询响应协议。

信任根绑定

Keystore 要求将密钥绑定到一个信任根。信任根是在启动期间提供给 Keymaster 安全硬件的一个位串（最好由引导加载程序提供）。该位串会以加密形式绑定到由 Keymaster 管理的每个密钥。

信任根包含一个公钥，该公钥用于验证启动映像上的签名和设备的锁定状态。如果该公钥被更改了（以允许使用不同的系统映像），或锁定状态发生了变化，之前的系统创建的受 Keymaster 保护的所有密钥都无法再使用，除非之前的信任根已恢复并且通过相应密钥签名的系统已启动。这是为了确保由攻击者安装的操作系统无法使用 Keymaster 密钥，从而提高由软件强制执行的密钥访问控制所发挥的作用。

独立密钥

有些 Keymaster 安全硬件可以将密钥材料存储在内部并返回句柄（而非经过加密的密钥材料）。也可能会存在相应密钥在一些其他非安全域或安全域系统组件可用之前无法使用的其他情况。Keymaster HAL 允许调用程序通过 `TAG::STANDALONE` 标记请求将密钥设为“独立”密钥，这意味着，除了 Blob 和运行中的 Keymaster 系统之外，不需要任何其他资源。要想知道某个密钥是否为独立密钥，可以查看与该密钥关联的标记。目前只为此标记定义了两个值：

- `KeyBlobUsageRequirements::STANDALONE`
- `KeyBlobUsageRequirements::REQUIRES_FILE_SYSTEM`

此功能未提供给应用。

使用时间间隔

密钥创建好后，可以通过 `TAG::MIN_SECONDS_BETWEEN_OPS` 指定使用时间间隔上限。如果距离上次使用相应密钥执行操作的时间还没有超过 `TAG::MIN_SECONDS_BETWEEN_OPS` 秒，TrustZone 实现会拒绝再次使用相应密钥执行加密操作。

要实现使用时间间隔上限，一种非常简单的方法是创建一个用于存放密钥 ID 和上次使用时间戳的表格。该表格可能有大小限制，但能够容纳至少 16 个条目。如果该表格已被占满，并且没有任何可以更新或舍弃的条目，那么安全硬件实现会“安全失败”，最好是拒绝所有受密钥使用时间间隔限制的密钥操作，直到其中一个条目过期为止。可设为所有条目在设备重新启动时过期。

也可以通过 `TAG::MAX_USES_PER_BOOT` 将密钥限制为每次设备启动后最多使用 n 次。这也需要一个跟踪表格（能够容纳至少 4 个密钥），并且也能够安全失败。请注意，应用无法创建按设备启动限制使用次数的密钥。该功能不会在 Keystore 之外提供，而且仅用于系统操作。

此功能未提供给应用。

随机数生成器补种

由于安全硬件会生成随机数（在密钥材料中使用）和初始化矢量 (IV)，而且硬件随机数生成器可能并非始终可信，因此 Keymaster HAL 会提供一个接口，以便客户端提供额外的熵（将与生成的随机数混合在一起）。

使用硬件随机数生成器作为主要种子来源。 通过外部 API 提供的种子数据不能是生成数字时所用随机数据的唯一来源。此外，如果有任何一个种子来源不可预测，则所使用的混合操作需要确保随机输出不可预测。

此功能未提供给应用，但可供框架使用。框架会定期为安全硬件提供从 Java SecureRandom 实例获取的其他熵。