APK 签名方案 v3

Android 9 支持 APK 密钥轮转 (https://developer.android.com/preview/features/security#apk-key-rotation)，这使应用能够在 APK 更新过程中更改其签名密钥。为了实现轮转，APK 必须指示新旧签名密钥之间的信任级别。为了支持密钥轮转，我们将 APK 签名方案 (/security/apksigning/v2)从 v2 更新为 v3，以允许使用新旧密钥。v3 在 APK 签名分块中添加了有关受支持的 SDK 版本和 proof-of-rotation 结构的信息。

APK 签名分块

为了保持与 v1 APK 格式向后兼容，v2 和 v3 APK 签名存储在“APK 签名分块”内紧邻 ZIP Central Directory 前面。

v3 APK 签名分块的格式与 v2 相同 (/security/apksigning/v2#apk-signing-block-format)。APK 的 v3 签名会存储为一个“ID-值”对，其中 ID 为 0xf05368c0。

APK 签名方案 v3 分块

v3 方案的设计与 v2 方案 (/security/apksigning/v2#apk-signature-scheme-v2-block)非常相似，采用相同的常规格式，并支持相同的签名算法 ID (/security/apksigning/v2#signature-algorithm-ids)、密钥大小和 EC 曲线。

但是，v3 方案增添了有关受支持的 SDK 版本和 proof-of-rotation 结构的信息。

格式

“APK 签名方案 v3 分块”存储在“APK 签名分块”内，ID 为 `0xf05368c0`。

“APK 签名方案 v3 分块”采用 v2 的格式：

- 带长度前缀的 `signer`（带长度前缀）序列：
  - 带长度前缀的 `signed data`：
    - 带长度前缀的 `digests`（带长度前缀）序列：
      - `signature algorithm ID`（4 个字节）
      - `digest`（带长度前缀）
    - 带长度前缀的 X.509 `certificates` 序列：
      - 带长度前缀的 X.509 `certificate`（ASN.1 DER 形式）
    - `minSDK` (uint32) - 如果平台版本低于此数字，则应忽略该签名者。
    - `maxSDK` (uint32) - 如果平台版本高于此数字，则应忽略该签名者。
    - 带长度前缀的 `additional attributes`（带长度前缀）序列：
      - `ID` (uint32)
      - `value`（可变长度：附加属性的长度 - 4 个字节）
      - `ID - 0x3ba06f8c`
      - `value` - Proof-of-rotation 结构
  - `minSDK` (uint32) - 签名数据部分中 minSDK 值的副本 - 用于在当前平台不在相应范围内时跳过对此签名的验证。必须与签名数据值匹配。
  - `maxSDK` (uint32) - 签名数据部分中 maxSDK 值的副本 - 用于在当前平台不在相应范围内时跳过对此签名的验证。必须与签名数据值匹配。
  - 带长度前缀的 `signatures`（带长度前缀）序列：
    - `signature algorithm ID` (uint32)
    - `signed data` 上带长度前缀的 `signature`
  - 带长度前缀的 `public key`（SubjectPublicKeyInfo，ASN.1 DER 形式）

Proof-of-rotation 和 self-trusted-old-certs 结构

proof-of rotation 结构允许应用轮转其签名证书，而不会使这些证书在与这些应用通信的其他应用上被屏蔽。为此，应用签名包含两个新数据块：

- 告知第三方应用的签名证书可信（只要其先前证书可信）的断言
- 应用的旧签名证书（应用本身仍信任这些证书）

签名数据部分中的 proof-of-rotation 属性包含一个单链表，其中每个节点都包含用于为之前版本的应用签名的签名证书。此属性旨在包含概念性 proof-of-rotation 和 self-trusted-old-certs 数据结构。该单链表按版本排序，最旧的签名证书对应于根节点。在构建 proof-of-rotation 数据结构时，系统会让每个节点中的证书为列表中的下一个证书签名，从而为每个新密钥提供证据来证明它应该像旧密钥一样可信。

在构造 self-trusted-old-certs 数据结构时，系统会向每个节点添加标记来指示它在组中的成员资格和属性。例如，可能存在一个标记，指示给定节点上的签名证书可信，可获得 Android 签名权限。此标记允许由旧证书签名的其他应用仍被授予由使用新签名证书签名的应用所定义的签名权限。由于整个 proof-of-rotation 属性都位于 v3 `signer` 字段的签名数据部分中，因此用于为所含 APK 签名的密钥会保护该属性。

此格式排除了多个签名密钥 (#multiple-certificates)的情况和将不同祖先签名证书 (#multiple-ancestors)收敛到一个证书的情况（多个起始节点指向一个通用接收器）。

格式

proof-of-rotation 存储在“APK 签名方案 v3 分块”内，ID 为 `0x3ba06f8c`。其格式为：

- 带长度前缀的 `levels`（带长度前缀）序列：
  - 带长度前缀的 `signed data`（由上一个证书签名 - 如果存在）

    - 带长度前缀的 X.509 `certificate`（ASN.1 DER 形式）
    - `signature algorithm ID` (uint32) - 上一级证书使用的算法
  - `flags` (uint32) - 这些标记用于指示此证书是否应该在 self-trusted-old-certs 结构中，以及用于哪些操作。
  - `signature algorithm ID` (uint32) - 必须与签名数据部分下一级中的相应 ID 一致。
  - `signed data` 上带长度前缀的 `signature`

多个证书

Android 目前将使用多个证书签名的 APK 视为具有与所含证书不同的签名身份。因此，签名数据部分中的 proof-of-rotation 属性构成了一个有向无环图，最好将其视为单链表，其中给定版本的每组签名者都表示一个节点。这为 proof-of-rotation 结构（下面的多签名者版本）带来了额外的复杂性。排序成为一个特别突出的问题。更重要的是，无法再单独为 APK 签名，因为 proof-of-rotation 结构必须让旧签名证书为新的证书集签名，而不是逐个签名。例如，如果希望由两个新密钥 B 和 C 签名的 APK 是由密钥 A 签名的，则它不能让 B 签名者仅包含 A 或 B 的签名，因为这是与 B 和 C 不同的签名身份。这意味着签名者必须在构建此类结构之前进行协调。

多个签名者 proof-of-rotation 属性

- 带长度前缀的 `sets`（带长度前缀）序列：
  - `signed data`（由上一组证书签名 - 如果存在）
    - 带长度前缀的 `certificates` 序列
      - 带长度前缀的 X.509 `certificate`（ASN.1 DER 形式）
    - `signature algorithm IDs` (uint32) 序列 - 上一组证书中的每个证书对应一个序列，且采用相同顺序。
  - `flags` (uint32) - 这些标记用于指示这组证书是否应该在 self-trusted-old-certs 结构中，以及用于哪些操作。
  - 带长度前缀的 `signatures`（带长度前缀）序列：
    - `signature algorithm ID` (uint32) - 必须与签名数据部分中的相应 ID 一致
    - `signed data` 上带长度前缀的 `signature`

proof-of-rotation 结构中的多个祖先实体

v3 方案也无法处理轮转到同一个应用的同一签名密钥的两个不同密钥。这不同于收购情形，在收购情形中，收购公司希望转移收购的应用以使用其签名密钥来共享权限。收购被视为受支持的用例，因为新应用将通过其软件包名称来区分，并且可能包含自己的 proof-of-rotation 结构。不受支持的用例是，同一应用有两个不同的路径指向相同的证书，这打破了在密钥轮转设计中做出的许多假设。

验证

在 Android 9 及更高版本中，可以根据 APK 签名方案 v3、v2 方案或 v1 方案验证 APK。较旧的平台会忽略 v3 签名并尝试验证 v2 签名，然后验证 v1。

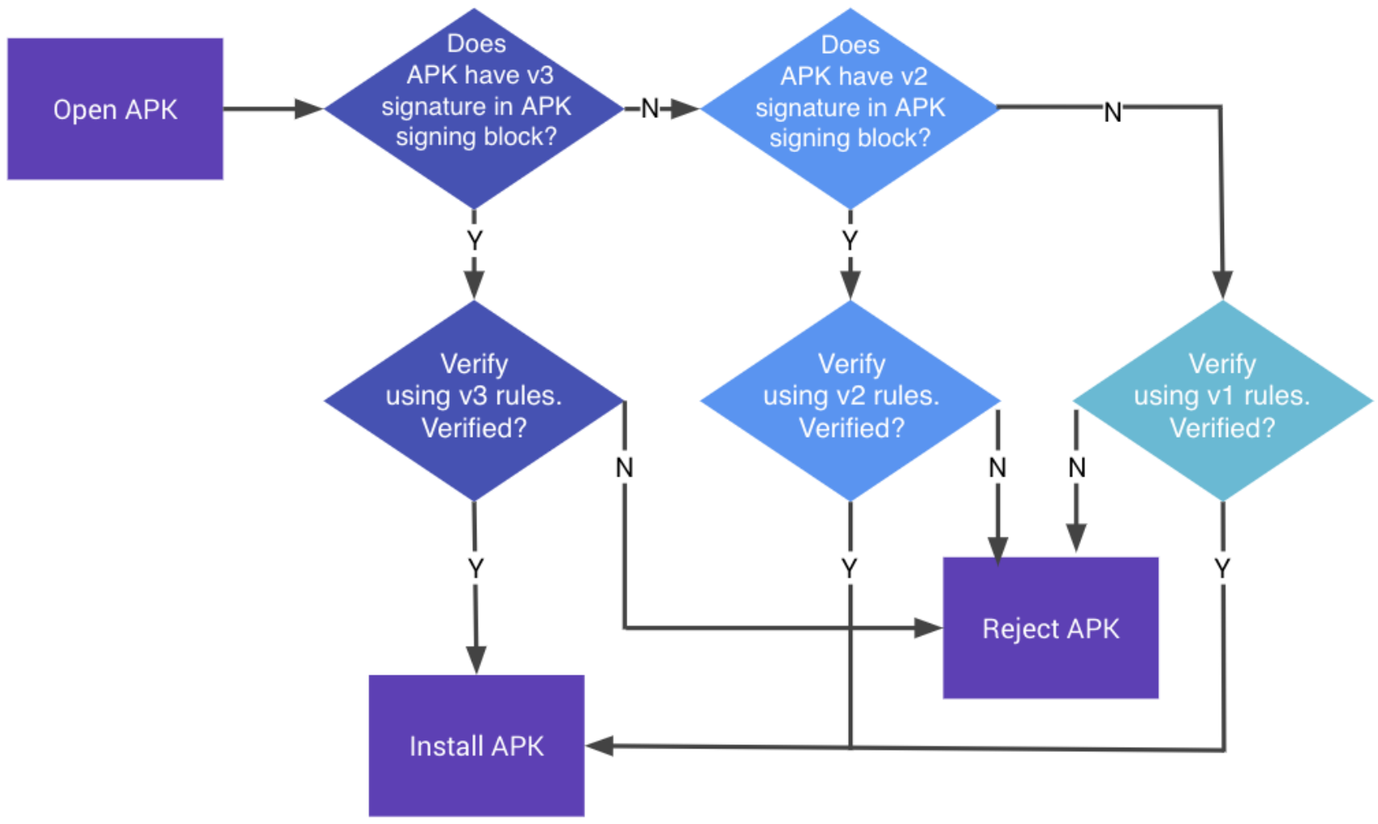

图 1. APK 签名验证过程

APK 签名方案 v3 验证

1. 找到“APK 签名分块”并验证以下内容：
   a. “APK 签名分块”的两个大小字段包含相同的值。
   b. “ZIP 中央目录结尾”紧跟在“ZIP 中央目录”记录后面。
   c. “ZIP 中央目录结尾”之后没有任何数据。
2. 找到“APK 签名分块”中的第一个“APK 签名方案 v3 分块”。如果 v3 分块存在，则继续执行第 3 步。否则，回退至使用 v2 方案 (/security/apksigning/v2#v2-verification)验证 APK。
3. 对“APK 签名方案 v3 分块”中的每个 `signer`（最低和最高 SDK 版本在当前平台的范围内）执行以下操作：
   a. 从 `signatures` 中选择安全系数最高的受支持 `signature algorithm ID`。安全系数排序取决于各个实现/平台版本。
   b. 使用 `public key` 并对照 `signed data` 验证 `signatures` 中对应的 `signature`。（现在可以安全地解析 `signed data` 了。）
   c. 验证签名数据中的最低和最高 SDK 版本是否与为 `signer` 指定的版本匹配。
   d. 验证 `digests` 和 `signatures` 中的签名算法 ID 列表（有序列表）是否相同。（这是为了防止删除/添加等

d. 验证 `digests` 和 `signatures` 中的签名算法 ID 列表（有序列表）是否相同。（这是为了防止删除/添加签名。）

e. 使用签名算法所用的同一种摘要算法计算 APK 内容的摘要 (/security/apksigning/v2#integrity-protected-contents)。

f. 验证计算出的摘要是否与 `digests` 中对应的 `digest` 相同。

g. 验证 `certificates` 中第一个 `certificate` 的 SubjectPublicKeyInfo 是否与 `public key` 相同。

h. 如果 `signer` 存在 proof-of-rotation 属性，则验证结构是否有效，以及此 `signer` 是否为列表中的最后一个证书。

4. 如果在当前平台范围内仅找到了一个 `signer`，并且对该 `signer` 成功执行第 3 步，则验证成功。

注意：如果第 3 步或第 4 步失败，则不得使用 v1 或 v2 方案验证 APK。

验证

要测试您的设备是否正确支持 v3，请在 `cts/hostsidetests/appsecurity/src/android/appsecurity/cts/` 中运行 `PkgInstallSignatureVerificationTest.java` CTS 测试。