APK 签名方案 v2

APK 签名方案 v2 是一种全文件签名方案，该方案能够发现对 APK 的受保护部分进行的所有更改，从而有助于加快验证速度并增强完整性保证 (#integrity-protected-contents)。

使用 APK 签名方案 v2 进行签名时，会在 APK 文件中插入一个 APK 签名分块 (#apk-signing-block)，该分块位于“ZIP 中央目录”部分之前并紧邻该部分。在“APK 签名分块”内，v2 签名和签名者身份信息会存储在 APK 签名方案 v2 分块 (#apk-signature-scheme-v2-block)中。

图 1. 签名前和签名后的 APK

APK 签名方案 v2 是在 Android 7.0 (Nougat) 中引入的。为了使 APK 可在 Android 6.0 (Marshmallow) 及更低版本的设备上安装，应先使用 JAR 签名 (index.html#v1)功能对 APK 进行签名，然后再使用 v2 方案对其进行签名。

APK 签名分块

为了保持与 v1 APK 格式向后兼容，v2 及更高版本的 APK 签名会存储在“APK 签名分块”内，该分块是为了支持 APK 签名方案 v2 而引入的一个新容器。在 APK 文件中，“APK 签名分块”位于“ZIP 中央目录”（位于文件末尾）之前并紧邻该部分。

该分块包含多个“ID-值”对，所采用的封装方式有助于更轻松地在 APK 中找到该分块。APK 的 v2 签名会存储为一个“ID-值”对，其中 ID 为 0x7109871a。

格式

“APK 签名分块”的格式如下（所有数字字段均采用小端字节序）：

- `size of block`，以字节数（不含此字段）计 (uint64)
- 带 uint64 长度前缀的“ID-值”对序列：
  - `ID` (uint32)
  - `value`（可变长度：“ID-值”对的长度 - 4 个字节）
- `size of block`，以字节数计 - 与第一个字段相同 (uint64)
- `magic`“APK 签名分块 42”（16 个字节）

在解析 APK 时，首先要通过以下方法找到“ZIP 中央目录”的起始位置：在文件末尾找到“ZIP 中央目录结尾”记录，然后从该记录中读取“中央目录”的起始偏移量。通过 `magic` 值，可以快速确定“中央目录”前方可能是“APK 签名分块”。然后，通过 `size of block` 值，可以高效地找到该分块在文件中的起始位置。

在解译该分块时，应忽略 ID 未知的“ID-值”对。

APK 签名方案 v2 分块

APK 由一个或多个签名者/身份签名，每个签名者/身份均由一个签名密钥来表示。该信息会以“APK 签名方案 v2 分块”的形式存储。对于每个签名者，都会存储以下信息：

- （签名算法、摘要、签名）元组。摘要会存储起来，以便将签名验证和 APK 内容完整性检查拆开进行。
- 表示签名者身份的 X.509 证书链。
- 采用键值对形式的其他属性。

对于每位签名者，都会使用收到的列表中支持的签名来验证 APK。签名算法未知的签名会被忽略。如果遇到多个支持的签名，则由每个实现来选择使用哪个签名。这样一来，以后便能够以向后兼容的方式引入安全系数更高的签名方法。建议的方法是验证安全系数最高的签名。

格式

“APK 签名方案 v2 分块”存储在“APK 签名分块”内，ID 为 `0x7109871a`。

“APK 签名方案 v2 分块”的格式如下（所有数字值均采用小端字节序，所有带长度前缀的字段均使用 uint32 值表示长度）：

- 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` (uint32)
      - （带长度前缀）`digest` - 请参阅受完整性保护的内容 (#integrity-protected-contents)
    - 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）
    - 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）
  - 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）

签名算法 ID

- 0x0101 - 采用 SHA2-256 摘要、SHA2-256 MGF1、32 个字节的盐且尾部为 0xbc 的 RSASSA-PSS 算法
- 0x0102 - 采用 SHA2-512 摘要、SHA2-512 MGF1、64 个字节的盐且尾部为 0xbc 的 RSASSA-PSS 算法
- 0x0103 - 采用 SHA2-256 摘要的 RSASSA-PKCS1-v1_5 算法。此算法适用于需要确定性签名的编译系统。
- 0x0104 - 采用 SHA2-512 摘要的 RSASSA-PKCS1-v1_5 算法。此算法适用于需要确定性签名的编译系统。
- 0x0201 - 采用 SHA2-256 摘要的 ECDSA 算法
- 0x0202 - 采用 SHA2-512 摘要的 ECDSA 算法
- 0x0301 - 采用 SHA2-256 摘要的 DSA 算法

Android 平台支持上述所有签名算法。签名工具可能只支持其中一部分算法。

支持的密钥大小和 EC 曲线：

- RSA：1024、2048、4096、8192、16384
- EC：NIST P-256、P-384、P-521
- DSA：1024、2048、3072

受完整性保护的内容

为了保护 APK 内容，APK 包含以下 4 个部分：

1. ZIP 条目的内容（从偏移量 0 处开始一直到“APK 签名分块”的起始位置）
2. APK 签名分块
3. ZIP 中央目录
4. ZIP 中央目录结尾

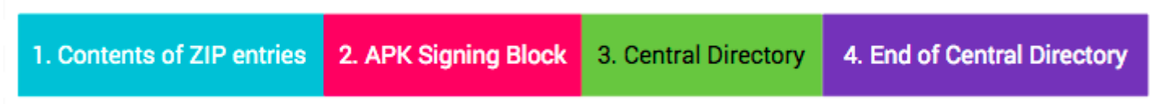

图 2. 签名后的各个 APK 部分

APK 签名方案 v2 负责保护第 1、3、4 部分的完整性，以及第 2 部分包含的“APK 签名方案 v2 分块”中的 `signed data` 分块的完整性。

第 1、3 和 4 部分的完整性通过其内容的一个或多个摘要来保护，这些摘要存储在 `signed data` 分块中，而这些分块则通过一个或多个签名来保护。

第 1、3 和 4 部分的摘要采用以下计算方式，类似于两级 Merkle 树 (https://en.wikipedia.org/wiki/Merkle_tree)。 每个部分都会被拆分成多个大小为 1 MB（2^{20} 个字节）的连续块。每个部分的最后一个块可能会短一些。每个块的摘要均通过字节 `0xa5` 的连接、块的长度（采用小端字节序的 uint32 值，以字节数计）和块的内容进行计算。顶级摘要通过字节 `0x5a` 的连接、块数（采用小端字节序的 uint32 值）以及块的摘要的连接（按照块在 APK 中显示的顺序）进行计算。摘要以分块方式计算，以便通过并行处理来加快计算速度。

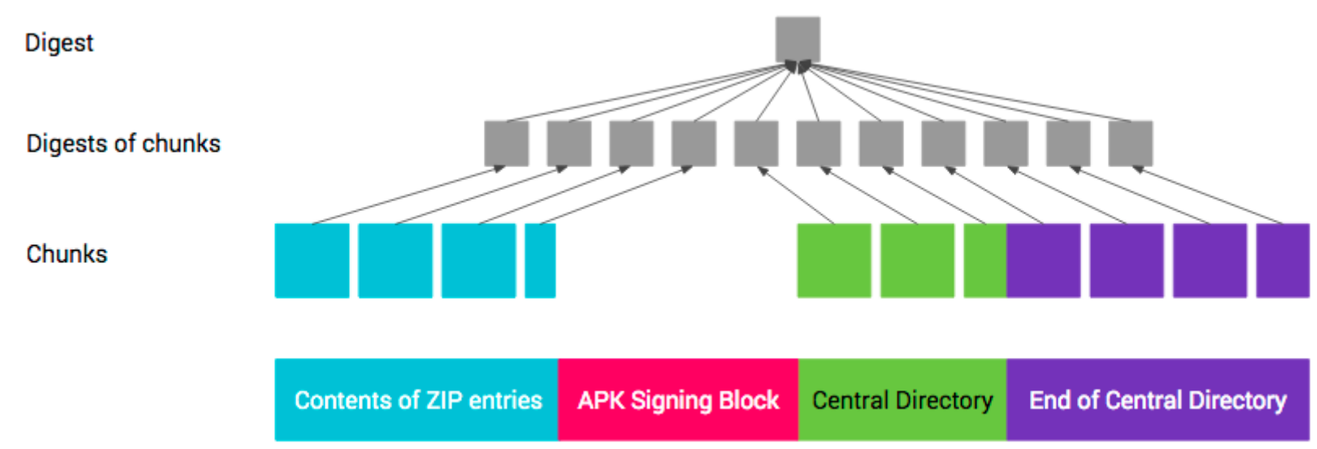

图 3. APK 摘要

由于第 4 部分（ZIP 中央目录结尾）包含“ZIP 中央目录”的偏移量，因此该部分的保护比较复杂。当“APK 签名分块”的大小发生变化（例如，添加了新签名）时，偏移量也会随之改变。因此，在通过“ZIP 中央目录结尾”计算摘要时，必须将包含“ZIP 中央目录”偏移量的字段视为包含“APK 签名分块”的偏移量。

防回滚保护

攻击者可能会试图在支持对带 v2 签名的 APK 进行验证的 Android 平台上将带 v2 签名的 APK 作为带 v1 签名的 APK 进行验证。为了防范此类攻击，带 v2 签名的 APK 如果还带 v1 签名，其 META-INF/*.SF 文件的主要部分中必须包含 X-Android-APK-Signed 属性。该属性的值是一组以英文逗号分隔的 APK 签名方案 ID（v2 方案的 ID 为 2）。在验证 v1 签名时，对于此组中验证程序首选的 APK 签名方案（例如，v2 方案），如果 APK 没有相应的签名，APK 验证程序必须要拒绝这些 APK。此项保护依赖于内容 META-INF/*.SF 文件受 v1 签名保护这一事实。请参阅 JAR 已签名的 APK 的验证 (#v1-verification)部分。

攻击者可能会试图从“APK 签名方案 v2 分块”中删除安全系数较高的签名。为了防范此类攻击，对 APK 进行签名时使用的签名算法 ID 的列表会存储在通过各个签名保护的 `signed data` 分块中。

验证

在 Android 7.0 及更高版本中，可以根据 APK 签名方案 v2+ 或 JAR 签名（v1 方案）验证 APK。更低版本的平台会忽略 v2 签名，仅验证 v1 签名。

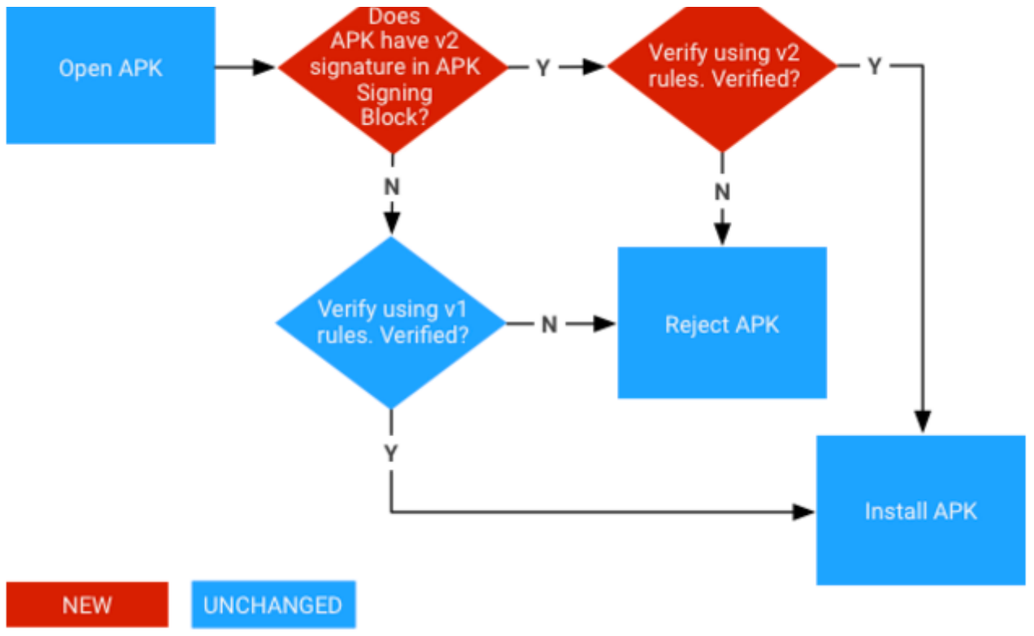

图 4. APK 签名验证过程（新步骤以红色显示）

APK 签名方案 v2 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 v2 分块”。如果 v2 分块存在，则继续执行第 3 步。否则，回退至使用 v1 方案 (#v1-verification)验证 APK。
3. 对“APK 签名方案 v2 分块”中的每个 `signer` 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 `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。）
    d. 使用签名算法所用的同一种摘要算法计算 APK 内容的摘要 (#integrity-protected-contents)。
    e. 验证计算出的摘要是否与 `digests` 中对应的 `digest` 相同。
    f. 验证 `certificates` 中第一个 `certificate` 的 SubjectPublicKeyInfo 是否与 `public key` 相同。
4. 如果找到了至少一个 `signer`，并且对于每个找到的 `signer`，第 3 步都取得了成功，APK 验证将会成功。

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

JAR 已签名的 APK 的验证（v1 方案）

JAR 已签名的 APK 是一种标准的已签名 JAR (https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html#Signed_JAR_File)，其中包含的条目必须与 META-INF/MANIFEST.MF 中列出的条目完全相同，并且所有条目都必须已由同一组签名者签名。其完整性按照以下方式进行验证：

1. 每个签名者均由一个包含 META-INF/<signer>.SF 和 META-INF/<signer>.(RSA|DSA|EC) 的 JAR 条目来表示。
2. <signer>.(RSA|DSA|EC) 是具有 SignedData 结构的 PKCS #7 CMS ContentInfo (https://tools.ietf.org/html/rfc5652)，其签名通过 <signer>.SF 文件进行验证。
3. <signer>.SF 文件包含 META-INF/MANIFEST.MF 的全文件摘要和 META-INF/MANIFEST.MF 各个部分的摘要。需要验证 MANIFEST.MF 的全文件摘要。如果该验证失败，则改为验证 MANIFEST.MF 各个部分的摘要。
4. 对于每个受完整性保护的 JAR 条目，META-INF/MANIFEST.MF 都包含一个具有相应名称的部分，其中包含相应条目未压缩内容的摘要。所有这些摘要都需要验证。
5. 如果 APK 包含未在 MANIFEST.MF 中列出且不属于 JAR 签名一部分的 JAR 条目，APK 验证将会失败。

因此，保护链是每个受完整性保护的 JAR 条目的 <signer>.(RSA|DSA|EC) -> <signer>.SF -> MANIFEST.MF -> 每个受完整性保护的 JAR 条目的内容。