新一代开源Android渠道包生成工具Walle (https://tech.meituan.com/2017/01/13/android-apk-v2-signature-scheme.html)

2017年01月13日 作者: 建帅 陈潼 **文章链接 (https://tech.meituan.com/2017/01/13/android-apk-v2-signature-scheme.html)**
4985字 10分钟阅读

在Android 7.0（Nougat）推出了新的应用签名方案APK Signature Scheme v2后，之前快速生成渠道包的方式（ **美团Android自动化之旅—生成渠道包 (http://tech.meituan.com/mt-apk-packaging.html)** ）已经行不通了，在此应用签名方案下如何快速生成渠道包呢?

本文会对新的应用签名方案APK Signature Scheme v2以及新一代渠道生成工具进行详细深入的介绍。

新的应用签名方案APK Signature Scheme v2

Android 7.0（Nougat）引入一项新的应用签名方案 **APK Signature Scheme v2 (https://source.android.com/security/apksigning/v2.html)** ，它是一个对全文件进行签名的方案，能提供更快的应用安装时间、对未授权APK文件的更改提供更多保护，在默认情况下，Android Gradle 2.2.0插件会使用APK Signature Scheme v2和传统签名方案来签署你的应用。

下面以 `新的应用签名方案` 来指APK Signature Scheme v2。

目前该方案不是强制性的，在 `build.gradle` 添加 `v2SigningEnabled false` ，就能使用传统签名方案来签署我们的应用（见下面的代码片段）。

```
android {
  ...
  defaultConfig { ... }
  signingConfigs {
    release {
      storeFile file("myreleasekey.keystore")
      storePassword "password"
      keyAlias "MyReleaseKey"
      keyPassword "password"
      v2SigningEnabled false
    }
  }
}
```

但新的应用签名方案有着良好的向后兼容性，能完全兼容低于Android 7.0（Nougat）的版本。对比旧签名方案，它有更快的验证速度和更安全的保护，因此新的应用签名方案可能会被采纳成一个强制配置，笔者认为现在有必要对现有的渠道包生成方式进行检查、升级或改造来支持新的应用签名方案。

新的签名方案对已有的渠道生成方案有什么影响呢？下图是新的应用签名方案和旧的签名方案的一个对比：

新的签名方案会在ZIP文件格式的 `Central Directory` 区块所在文件位置的前面添加一个APK Signing Block区块，下面按照ZIP文件的格式来分析新应用签名方案签名后的APK包。

整个APK（ZIP文件格式）会被分为以下四个区块： 1. Contents of ZIP entries（from offset 0 until the start of APK Signing Block） 2. APK Signing Block 3. ZIP Central Directory 4. ZIP End of Central Directory

新应用签名方案的签名信息会被保存在区块2（APK Signing Block）中， 而区块1（`Contents of ZIP entries`）、区块3（`ZIP Central Directory`）、区块4（`ZIP End of Central Directory`）是受保护的，在签名后任何对区块1、3、4的修改都逃不过新的应用签名方案的检查。

之前的渠道包生成方案是通过在META-INF目录下添加空文件，用空文件的名称来作为渠道的唯一标识，之前在META-INF下添加文件是不需要重新签名应用的，这样会节省不少打包的时间，从而提高打渠道包的速度。但在新的应用签名方案下META-INF已经被列入了保护区了，向META-INF添加空文件的方案会对区块1、3、4都会有影响，新应用签名方案签署的应用经过我们旧的生成渠道包方案处理后，在安装时会报以下错误：

```
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES:
Failed to collect certificates from base.apk: META-INF/CERT.SF
indicates base.apk is signed using APK Signature Scheme v2,
but no such signature was found. Signature stripped?]
```

目前另外一种比较流行的 **渠道包快速生成方案**(http://linghaolu.github.io/apk/2016/04/02/apk-comment.html) （往APK中添加ZIP Comment）也因为上述原因，无法在新的应用签名方案下进行正常工作。

如果新的应用签名方案后续改成强制要求，那我们现有的生成渠道包的方式就会无法工作，那我们难道要退回到解放前，通过传统的方式（例如：使用APKTool逆向工具、采用Flavor + BuildType等比较耗时的方案来进行渠道包打包）来生成支持新应用签名方案的渠道包吗？

如果只有少量渠道包的场景下，这种耗时时长还能够勉强接受。但是目前我们有将近900个渠道，如果采用传统方式打完所有的渠道包需要近3个小时，这是不能接受的。

那我们有没有其他更好的渠道包生成方式，既能支持新的应用签名方案，又能体验毫秒级的打包耗时呢？我们来分析一下新方案中的区块2——Block。

可扩展的APK Signature Scheme v2 Block

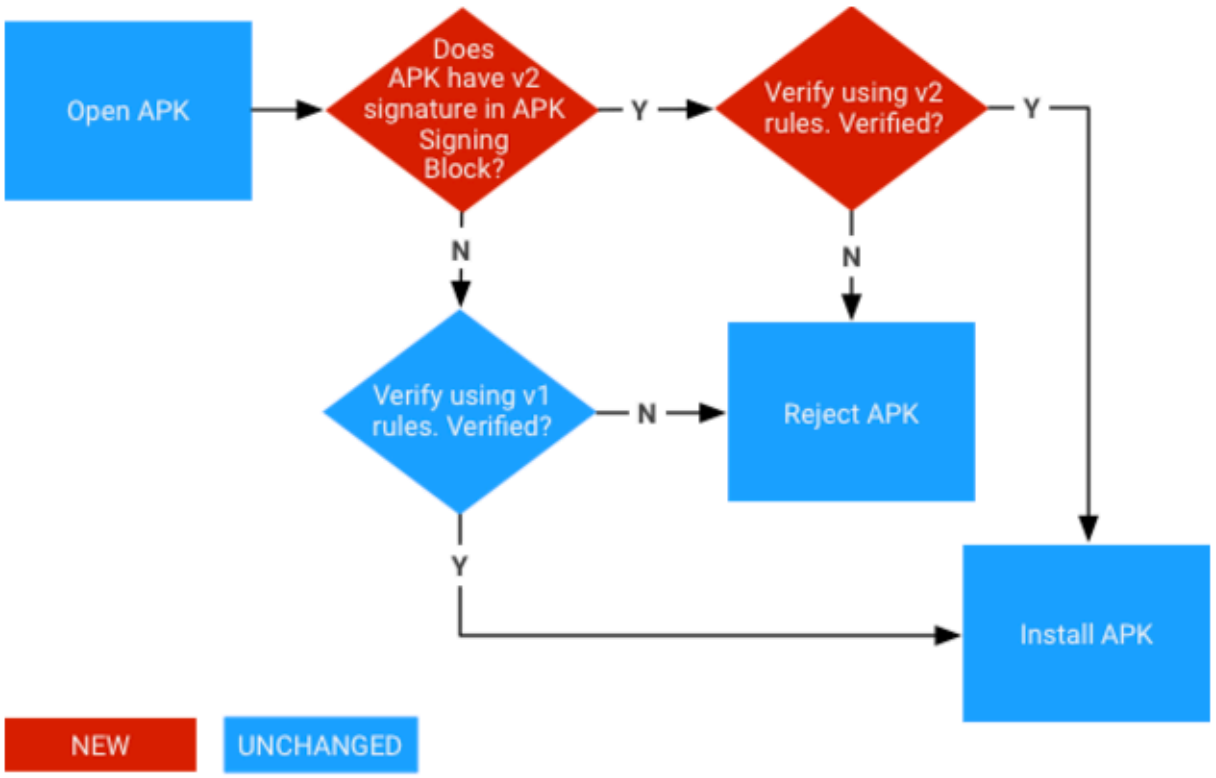
通过上面的描述，可以看出因为APK包的区块1、3、4都是受保护的，任何修改在签名后对它们的修改，都会在安装过程中被签名校验检测失败，而区块2（APK Signing Block）是不受签名校验规则保护的，那是否可以在这个不受签名保护的区块2（APK Signing Block）上做文章呢？我们先来看看对区块2格式的描述：

偏移	字节数	描述
@+0	8	这个Block的长度（本字段的长度不计算在内）
@+8	n	一组ID-value
@-24	8	这个Block的长度（和第一个字段一样值）
@-16	16	魔数 “APK Sig Block 42”

区块2中APK Signing Block是由这几部分组成：2个用来标示这个区块长度的8字节 + 这个区块的魔数（`APK Sig Block 42`）+ 这个区块所承载的数据（ID-value）。

我们重点来看一下这个ID-value，它由一个8字节的长度标示+4字节的ID+它的负载组成。V2的签名信息是以ID（`0x7109871a`）的ID-value来保存在这个区块中，不知大家有没有注意这是一组ID-value，也就说它是可以有若干个这样的ID-value来组成，那我们是不是可以在这里做一些文章呢？

为了验证我们的想法，先来看看新的应用签名方案是怎么验证签名信息的，见下图：



通过上图可以看出新的应用签名方案的验证过程： 1. 寻找APK Signing Block，如果能够找到，则进行验证，验证成功则继续进行安装，如果失败了则终止安装 2. 如果未找到APK Signing Block，则执行原来的签名验证机制，也是验证成功则继续进行安装，如果失败了则终止安装

那Android应用在安装时新的应用签名方案是怎么进行校验的呢？笔者通过翻阅Android相关部分的源码，发现下面代码段是用来处理上面所说的ID-value的：

```
    public static ByteBuffer findApkSignatureSchemeV2Block(
            ByteBuffer apkSigningBlock,
            Result result) throws SignatureNotFoundException {
        checkByteOrderLittleEndian(apkSigningBlock);
        // FORMAT:
        // OFFSET       DATA TYPE  DESCRIPTION
        // * @+0  bytes uint64:    size in bytes (excluding thi
s field)
        // * @+8  bytes pairs
        // * @-24 bytes uint64:    size in bytes (same as the o
ne above)
        // * @-16 bytes uint128:   magic
        ByteBuffer pairs = sliceFromTo(apkSigningBlock, 8, apkS
igningBlock.capacity() - 24);

        int entryCount = 0;
        while (pairs.hasRemaining()) {
            entryCount++;
            if (pairs.remaining() < 8) {
                throw new SignatureNotFoundException(
                        "Insufficient data to read size of APK
Signing Block entry #" + entryCount);
            }
            long lenLong = pairs.getLong();
            if ((lenLong < 4) || (lenLong > Integer.MAX_VALUE))
{
                throw new SignatureNotFoundException(
                        "APK Signing Block entry #" + entryCoun
t
                                + " size out of range: " + lenL
ong);
            }
            int len = (int) lenLong;
            int nextEntryPos = pairs.position() + len;
            if (len > pairs.remaining()) {
                throw new SignatureNotFoundException(
                        "APK Signing Block entry #" + entryCoun
t + " size out of range: " + len
                                + ", available: " + pairs.remai
ning());
            }
            int id = pairs.getInt();
            if (id == APK_SIGNATURE_SCHEME_V2_BLOCK_ID) {
                return getByteBuffer(pairs, len - 4);
            }
            result.addWarning(Issue.APK_SIG_BLOCK_UNKNOWN_ENTRY
_ID, id);
            pairs.position(nextEntryPos);
        }

        throw new SignatureNotFoundException(
                "No APK Signature Scheme v2 block in APK Signin
g Block");
    }
```

上述代码中关键的一个位置是 `if (id == APK_SIGNATURE_SCHEME_V2_BLOCK_ID) {return getByteBuffer(pairs, len - 4);}`，通过源代码可以看出Android是通过查找ID为 `APK_SIGNATURE_SCHEME_V2_BLOCK_ID = 0x7109871a` 的ID-value，来获取APK Signature Scheme v2 Block，对这个区块中其他的ID-value选择了忽略。

在 **APK Signature Scheme v2 (https://source.android.com/security/apksigning/v2.html)** 中没有看到对无法识别的ID，有相关处理的介绍。

当看到这里时，我们可不可以设想一下，提供一个自定义的ID-value并写入该区域，从而为快速生成渠道包服务呢？

怎么向ID-value中添加信息呢？通过阅读ZIP的文件格式和APK Signing Block格式的描述，笔者通过编写下面的代码片段进行验证，发现通过在已经被新的应用签名方案签名后的APK中添加自定义的ID-value，是不需要再次经过签名就能安装的，下面是部分代码片段。

```
  public void writeApkSigningBlock(DataOutput dataOutput) {
        long length = 24;
        for (int index = 0; index < payloads.size(); ++index) {
            ApkSigningPayload payload = payloads.get(index);
            byte[] bytes = payload.getByteBuffer();
            length += 12 + bytes.length;
        }

        ByteBuffer byteBuffer = ByteBuffer.allocate(Long.BYTE
S);
        byteBuffer.order(ByteOrder.LITTLE_ENDIAN);
        byteBuffer.putLong(length);
        dataOutput.write(byteBuffer.array());

        for (int index = 0; index < payloads.size(); ++index) {
            ApkSigningPayload payload = payloads.get(index);
            byte[] bytes = payload.getByteBuffer();

            byteBuffer = ByteBuffer.allocate(Integer.BYTES);
            byteBuffer.order(ByteOrder.LITTLE_ENDIAN);
            byteBuffer.putInt(payload.getId());
            dataOutput.write(byteBuffer.array());

            dataOutput.write(bytes);
        }
        ...
    }
```

新一代渠道包生成工具

到这里为止一个新的渠道包生成方案逐步清晰了起来，下面是新一代渠道包生成工具的描述：

1. 对新的应用签名方案生成的APK包中的ID-value进行扩展，提供自定义ID - value（渠道信息），并保存在APK中
2. 而APK在安装过程中进行的签名校验，是忽略我们添加的这个ID-value的，这样就能正常安装了
3. 在App运行阶段，可以通过ZIP的 `EOCD（End of central directory）`、`Central directory` 等结构中的信息（会涉及ZIP格式的相关知识，这里不做展开描述）找到我们自己添加的ID-value，从而实现获取渠道信息的功能

新一代渠道包生成工具完全是基于ZIP文件格式和APK Signing Block存储格式而构建，基于文件的二进制流进行处理，有着良好的处理速度和兼容性，能够满足不同的语言编写的要求，目前笔者采用的是Java + Groovy开发， 该工具主要有四部分组成： 1. 用于写入ID-value信息的Java类库 2. Gradle构建插件用来和Android的打包流程进行结合 3. 用于读取ID-value信息的Java类库 4. 用于供 `com.android.application` 使用的读取渠道信息的AAR

这样，每打一个渠道包只需复制一个APK，然后在APK中添加一个ID-value即可，这种打包方式速度非常快，对一个30M大小的APK包只需要100多毫秒（包含文件复制时间）就能生成一个渠道包，而在运行时获取渠道信息只需要大约几毫秒的时间。

这个项目我们取名为Walle（瓦力），已经开源，项目的Github地址是：**https://github.com/Meituan-Dianping/walle (https://github.com/Meituan-Dianping/walle)** （求Issue、PR、Star）。希望业内有类似需求的团队能够在APK Signature Scheme V2签名下愉快地生成渠道包，同时也期待大家一起对该项目进行完善和优化。

总结

以上就是我们对新的应用签名方案进行的分析，并根据它所带来的文件存储格式上的变化，找到了可以利用的ID-value，然后基于这个ID-value来构建我们新一代渠道包生成工具。

新一代渠道包生成工具能够满足新应用签名方案对安全性的要求，同时也能满足对渠道包打包时间的要求，至此大家生成渠道包的方式需要升级了！

文章中引用的图片来源于： **https://source.android.com/security/apksigning/v2.html (https://source.android.com/security/apksigning/v2.html)**

参考文献

1. **APK Signature Scheme v2 (https://source.android.com/security/apksigning/v2.html)**
2. **ApkSigner的源代码 (https://android.googlesource.com/platform/build/+/8740e9d)**
3. **apksig的源代码 (https://android.googlesource.com/platform/tools/apksig/)**
4. [ZIP Format](**https://en.wikipedia.org/wiki/Zip_(file_format (https://en.wikipedia.org/wiki/Zip_(file_format)**)