Android应用签名的枷锁与革新

张煜龙，韦韬（Lenx）
{ylzhang, lenx}@baidu.com

百度安全实验室 Baidu X-Lab
2017.06

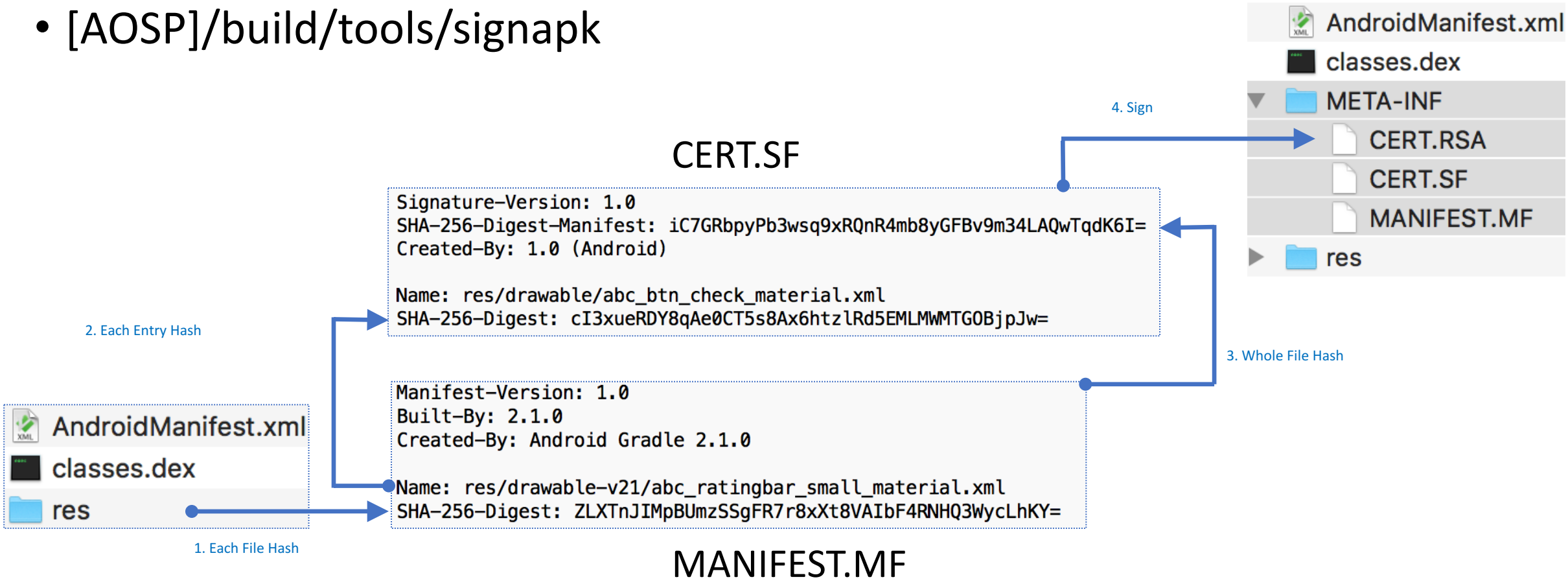
概要

- Android应用签名简介
- 签名对Android生态的意义
- 应用签名的局限和生态乱象
- 既有解决方案
- OASP: 救赎与革新

Android应用签名简介

签名原理

- [AOSP]/build/tools/signapk

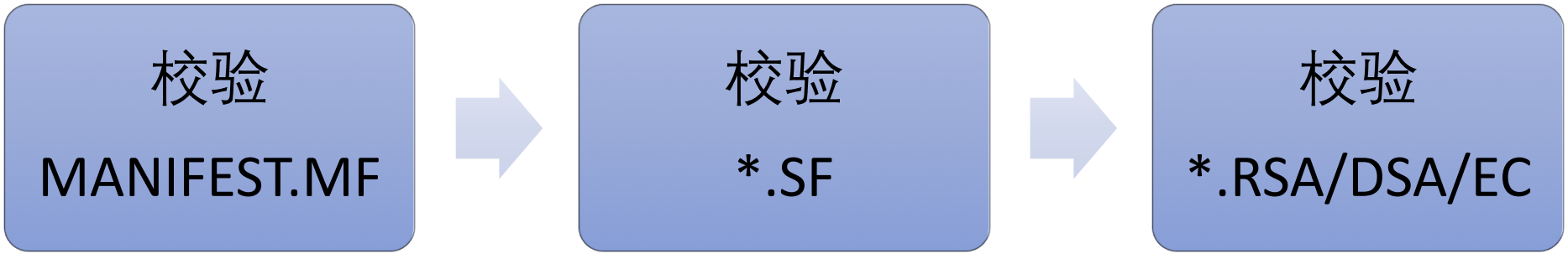


允许多密钥、多种算法签名

```
synchronized boolean readCertificates() {
    if (metaEntries.isEmpty()) {
        return false;
    }

    Iterator<String> it = metaEntries.keySet().iterator();
    while (it.hasNext()) {
        String key = it.next();
        if (key.endsWith(".DSA") || key.endsWith(".RSA") || key.endsWith(".EC")) {
            verifyCertificate(key);
            it.remove();
        }
    }
    return true;
}
```

参考：libcore/luni/src/main/java/java/util/jar/JarVerifier.java

安装校验签名



- 如果MANIFEST.MF上的entry没有对应的文件存在——可以
- 如果MANIFEST.MF上没有entry但文件存在——不可以

签名校验的那些坑

- Master Key漏洞
  - Bug 8219321, 9695860, 9950697
- META-INF目录下的文件不进入MANIFEST.MF
  - 被广泛用于隐藏文件或签名后插入渠道信息
- Android安装、运行不校验证书是否过期
  - 但是一些App Store会校验
- 不校验证书链，只要每一级的Issuer等于(strcmp)上一级的Subject就行
  - Self-signed情况也不进行自我验证
  - 因此可以随意篡改Subject
- Android签名证书没有PKI体系，最上一级的证书（Self-signed证书）就是可信CA
  - Android没有CRL/OCSP，有个证书黑名单，但仅用于SSL/HTTPS

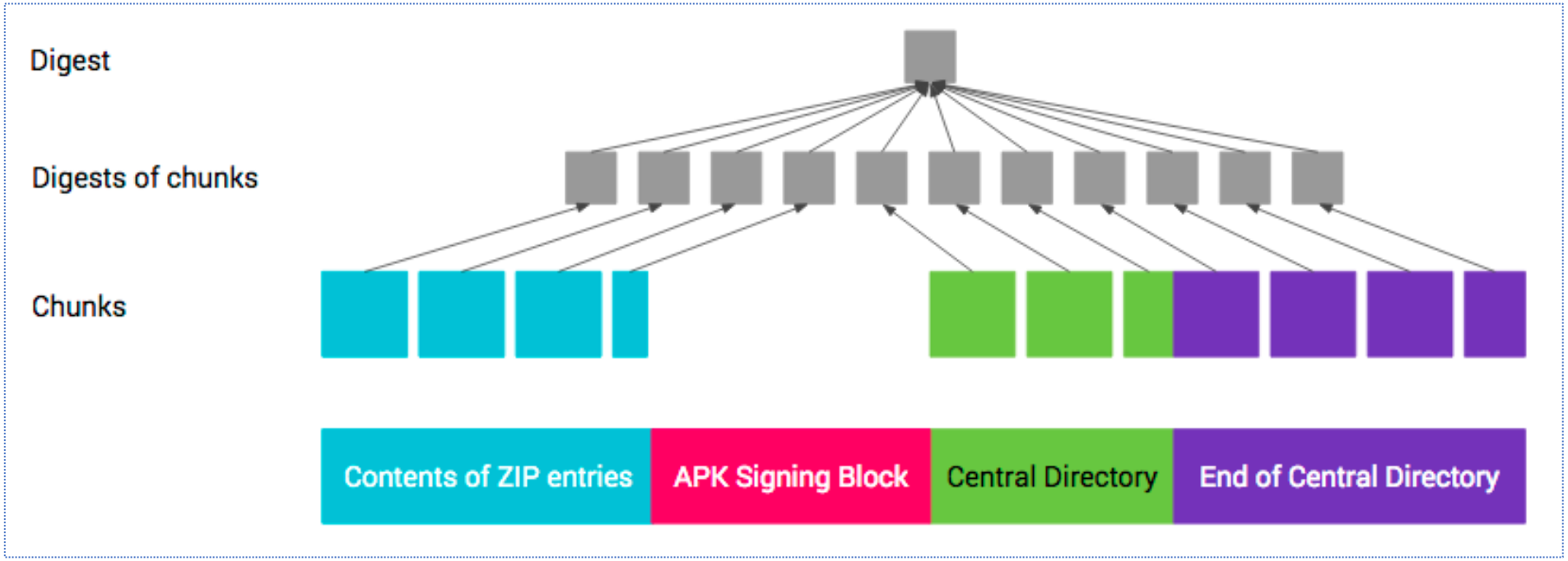
有些patch一直没有合入

```
commit  2bc5e811a817a8c667bca4318ae98582b0ee6dc6        [log] [tgz]
author  Kenny Root <kroot@google.com>                   Thu Apr 17 11:
committer  Kenny Root <kroot@google.com>                Wed Apr 30 16:
tree  7e8e824bd964e1a7a45d013e0a007cfbbed22e40
parent  afd7d9472e5d850a8e1a6d02abaaa9f94579a77f [diff]
```

```
Add API to check certificate chain signatures

Add hidden API to check certificate chain signatures when needed. The
getCertificates implementation returns a list of all the certificates and
chains and would expect any caller interested in verifying actual chains
to call getCodeSigners instead.

We add this hidden constructor as a stop-gap until we can switch callers
over to getCodeSigners.

Bug: 13678484
Change-Id: I01cddef287767422454de4c5fd266c812a04d570
```

The “Fake ID” patch

Scheme v2 (from Android 7)



- 更快（区块化、可并行）
- 为兼容考虑，仍需先用旧的方式先签一遍再在外面用Scheme v2签名

注意：不要混淆"证书"和"签名"

- *.RSA/DSA/EC
  - PKCS #7格式的签名（包含证书）

- PackageManager.getPackageInfo(pkgName, GET_SIGNATURES)
  - X.509格式的证书

AOSP导致了社区开发者们的混淆

```
 * This class name is slightly misleading, since it's not actually a signature.
 */
public class Signature implements Parcelable {
    private final byte[] mSignature;
```

下面的讨论中，引用代码里的”signature”很多实际上是certificate，注意区分

签名对Android生态的意义

防止恶意篡改

如果APK内文件被篡改：

java.lang.SecurityException: META-INF/MANIFEST.MF has invalid digest for res/drawable/haha

如果进而篡改MANIFEST.MF：

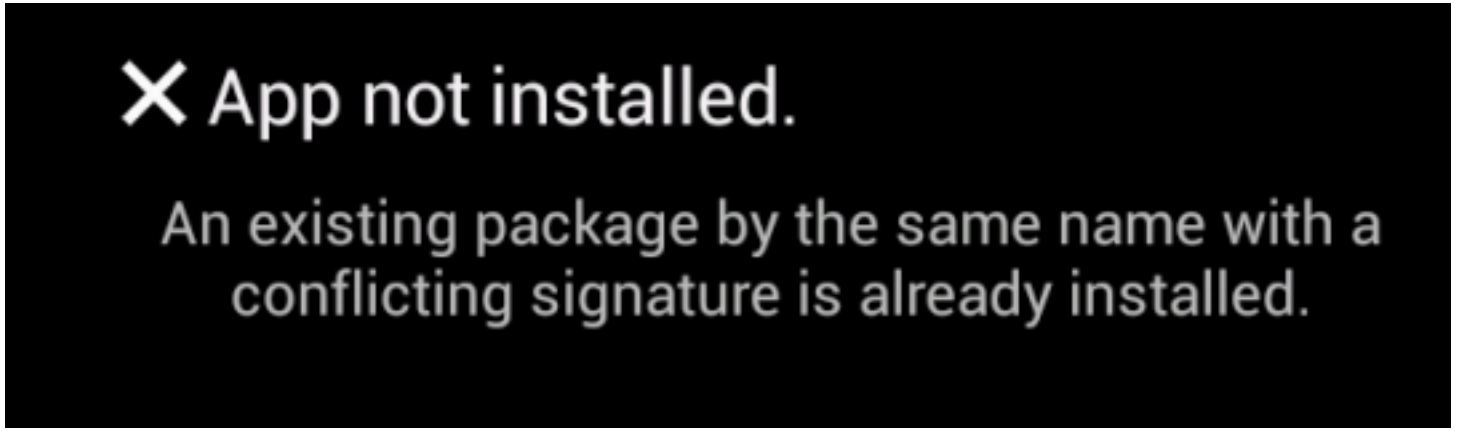
java.lang.SecurityException: META-INF/CERT.SF has invalid digest for res/drawable/haha

如果进而篡改CERT.SF：

java.lang.SecurityException: Incorrect signature

无法再进一步篡改*.RSA/DSA/EC，除非获得开发者私钥

防止恶意替换


App not installed.
An existing package by the same name with a conflicting signature is already installed.

应用上架与安装

- You must use the same certificate throughout the lifespan of your app in order for users to be able to install new versions as updates to the app.

来源：https://developer.android.com/studio/publish/app-signing.html

- 注意key一样不够、需要证书完全一致（same certificate hash）

应用上架与安装

Upload failed

You uploaded an APK that is signed with a different certificate to your previous APKs. You must use the same certificate. Your existing APKs are signed with the certificate(s) with fingerprint(s):

```
[ SHA1:
A7:18:CB:09:D3:93:34:4A:ED:89:29:78:75:CA:6F:08:44:E4:B2:3B ]
```

and the certificate(s) used to sign the APK you uploaded have fingerprint(s):

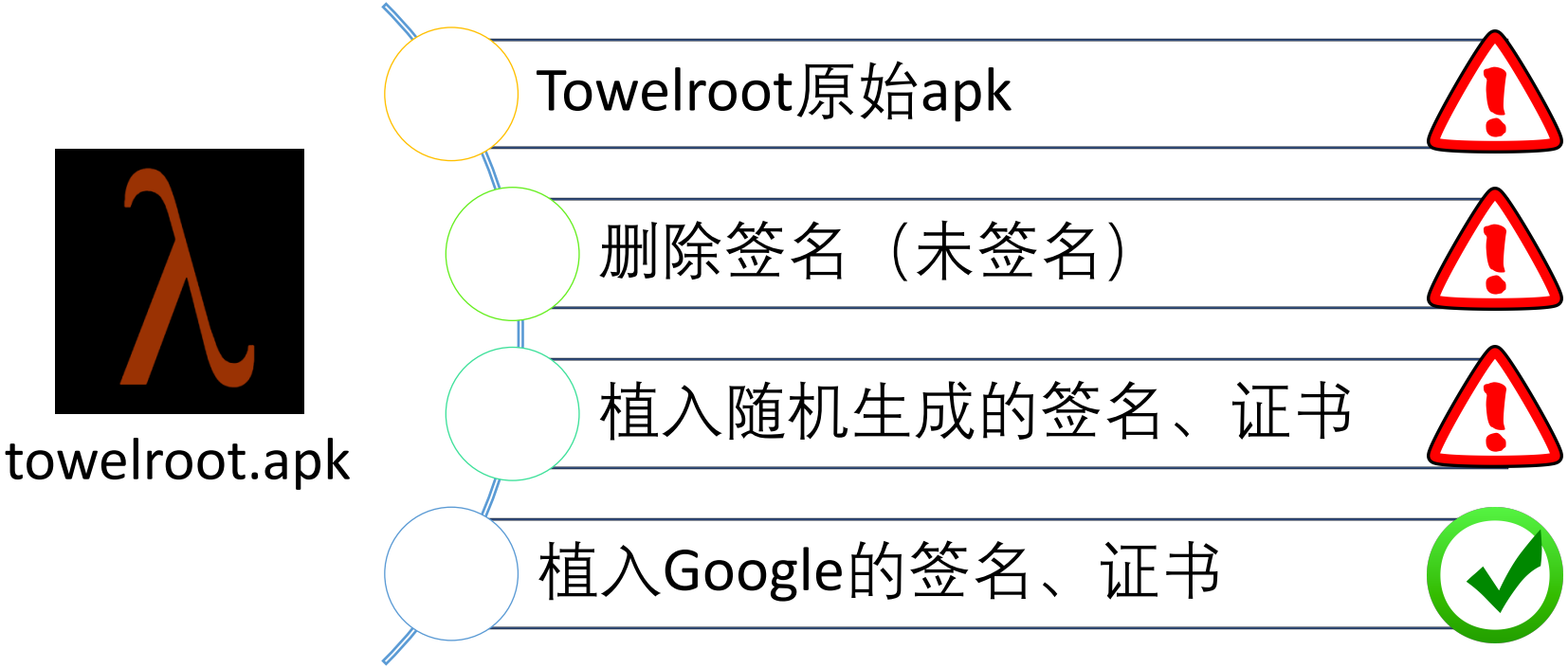
```
[ SHA1:
7C:C9:F9:A7:B7:9E:7C:62:CB:58:C3:FB:3A:18:66:C1:80:9F:49:02 ]
```

安全厂商检测——黑名单

- Google Play数个下载量1M+的安全app有证书白名单逻辑

安全厂商检测——白名单

- Google Play数个下载量1M+的安全app有证书白名单逻辑



权限授予

- frameworks/base/core/java/android/content/pm/PackageManager.java

```
/**
 * Compare the signatures of two packages to determine if the same
 * signature appears in both of them.  If they do contain the same
 * signature, then they are allowed special privileges when working
 * with each other: they can share the same user-id, run instrumentation
 * against each other, etc.
 */
@CheckResult
public abstract int checkSignatures(String pkg1, String pkg2);
```

- 注意如前文所提及，这里signature实际上是certificate
- 需要整条证书链一致

Webview插件注入

- frameworks/base/core/java/android/webkit/PluginManager.java
  - （仅适用于Android <= 4.3）

```
// Only plugin matches one of the signatures in the list can be loaded
// inside the WebView process
private static final String SIGNATURE_1 = "308204c5308203ada003020102020
```

特殊资源访问

/etc/nfcee_access.xml，控制谁能访问NFC Execution Environment：

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <!-- Applications granted NFCEE access on user builds

    See packages/apps/Nfc/etc/sample_nfcee_access.xml for full documentation.
     -->

    <!--  Google wallet release signature -->
    <signer android:signature="3082044c30820334a0030201020900a8cd17c93da5d99

</resources>
```

小结

- 应用签名是Android生态的基石
- 应用签名证书（私钥）具有“不可遗失”、无法更换的特点

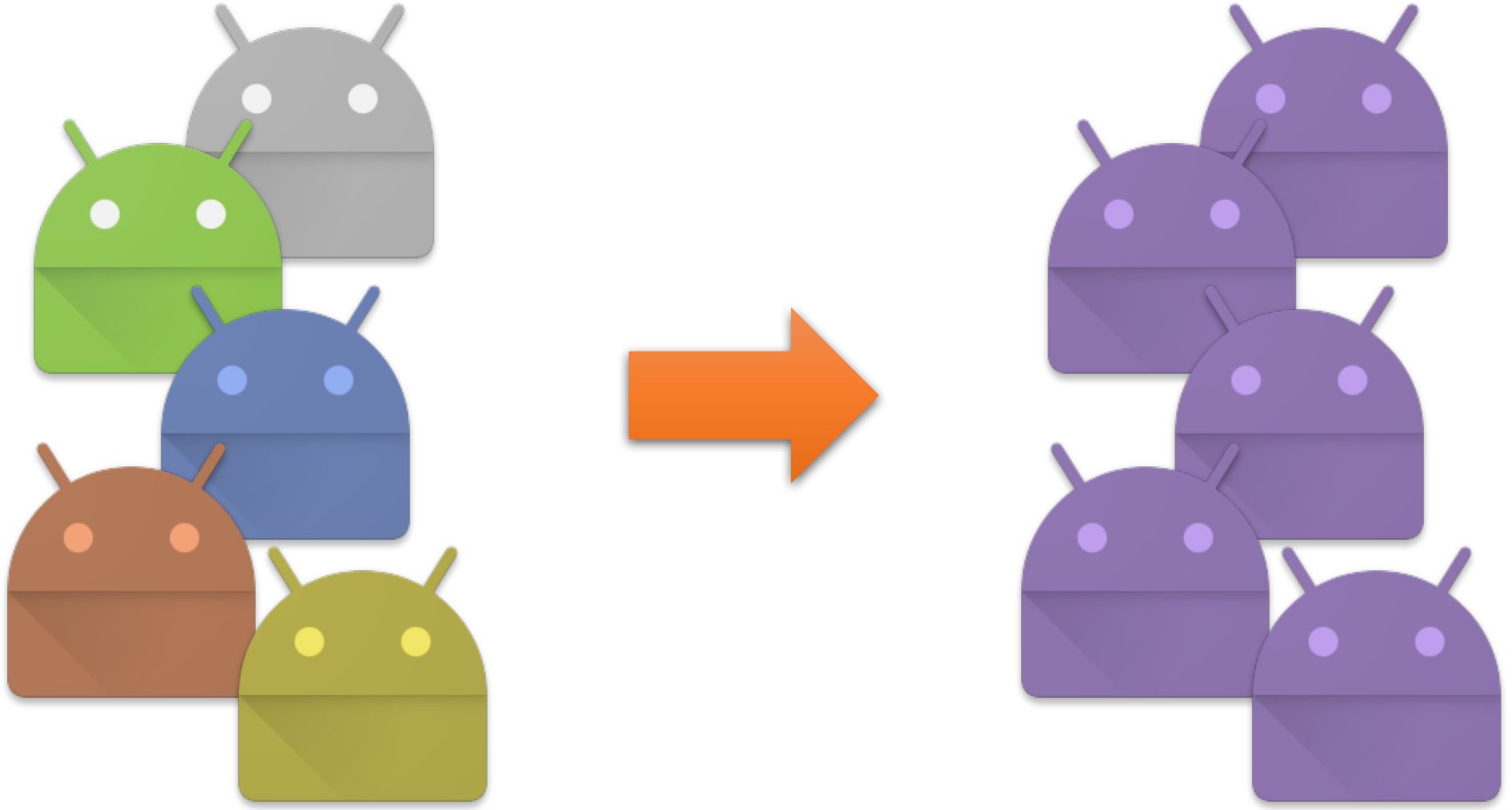
应用签名的局限和生态乱象

使用公开的Debug签名

- [AOSP]/build/target/product/security
  - media.pk8
  - platform.pk8
  - shared.pk8
  - testkey.pk8

```
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 12941516320735154170 (0xb3998086d056cffa)
Signature Algorithm: md5WithRSAEncryption
    Issuer:
        emailAddress               = android@android.com
        commonName                 = Android
        organizationalUnitName     = Android
        organizationName           = Android
        localityName               = Mountain View
        stateOrProvinceName        = California
        countryName                = US
    Validity
        Not Before: Apr 15 22:40:50 2008 GMT
        Not After : Sep  1 22:40:50 2035 GMT
    Subject:
        emailAddress               = android@android.com
        commonName                 = Android
        organizationalUnitName     = Android
        organizationName           = Android
        localityName               = Mountain View
        stateOrProvinceName        = California
        countryName                = US
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:9c:78:05:92:ac:0d:5d:38:1c:de:aa:65:ec:c8:
                a6:00:6e:36:48:0c:6d:72:07:b1:20:11:be:50:86:
                3a:ab:e2:b5:5d:00:9a:df:71:46:d6:f2:20:22:80:
```

外包签名、ROM重签名、Store重签名



Prod证书控制不严，Dev随意使用

- 开发测试
  - com.product与com.product.test拥有同样的签名
  - com.product与com.example.xxx拥有同样的签名
- QA测试
  - com.product与com.product.qa拥有同样签名

注意以上现象不止开发者有，某些出货很大的手机厂商也有

私钥泄露

blogs.adobe.com

Search Blogs

Security @ Adobe

HOME › Inappropriate Use of Adobe Code Signing Certificate

Inappropriate Use of Adobe Code Signing Certificate

We recently received two malicious utilities that appeared to be digitally signed using a valid Adobe code signing certificate. The discovery of these utilities was isolated to a single source. As soon as we verified the signatures, we immediately decommissioned the existing Adobe code signing infrastructure and initiated a forensics investigation to determine how these signatures were created. We have identified a compromised build server with access to the Adobe code signing infrastructure. We are proceeding with plans to revoke the certificate and publish updates for existing Adobe software signed using the impacted certificate. This only

Inadvertently Disclosed Digital Certificate Could Allow Spoofing

Published: December 8, 2015

Microsoft

Executive Summary

Microsoft is aware of an SSL/TLS digital certificate for *.xboxlive.com for which the private keys were inadvertently disclosed. The certificate could be used in attempts to perform man-in-the-middle attacks. It cannot be used to issue other certificates, impersonate other domains, or sign code. This issue affects all supported releases of Microsoft Windows. Microsoft is

D-Link spilled its private key onto the web – letting malware dress up as Windows apps

Friday firmware facepalm

18 Sep 2015 at 19:01, Chris Williams

Updated Taiwanese networking kit maker D-Link leaked a private code-signing key onto the internet for anyone to download.

Yahoo Axis Chrome Extension Leaks Private Certificate File

Preamble: The tl;dr for users is to not install (in my opinion) the Yahoo! Axis extenion for Chrome until this issue is clarified. See update below about disclosing this issue.

Yahoo! today announced their new

DigiNotar reports security incident

OAKBROOK TERRACE, Illinois and ZURICH, Switzerland – August 30, 2011 – VASCO Data Security International, Inc. (Nasdaq: VDSI; www.vasco.com) today comments on DigiNotar's reported security incident. DigiNotar is a wholly owned subsidiary of VASCO.
On July 19th 2011, DigiNotar detected an intrusion into its Certificate Authority (CA) infrastructure, which resulted in the fraudulent issuance of public key certificate requests for a number of domains, including Google.com.

The Bit9 incident

We see in the news another example of cyber criminals successfully stealing a private certificate and using it to their nefarious advantage. In this instance, cyber criminals allegedly exploited perimeter defences and web application security to gain access to one of Bit9's private certificates –

即使顶级软硬件公司，甚至安全公司和CA，都不能幸免

证书过期

- You must use the same certificate throughout the lifespan of your app in order for users to be able to install new versions as updates to the app.

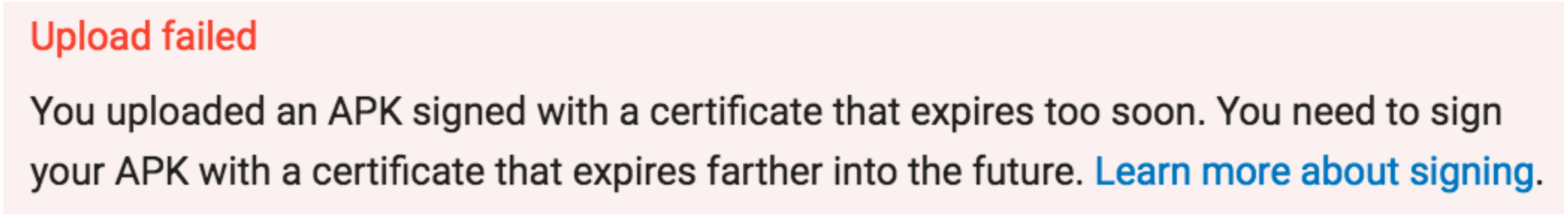
来源：https://developer.android.com/studio/publish/app-signing.html

- 注意key一样不够、需要证书完全一致（same certificate fingerprint）

这就意味着过期证书无法延期

Google’s Recommendation

- If you plan to support upgrades for an app, ensure that your app signing key has a validity period that exceeds the expected lifespan of that app.
- A validity period of 25 years or more is recommended. When your key's validity period expires, users will no longer be able to seamlessly upgrade to new versions of your app.

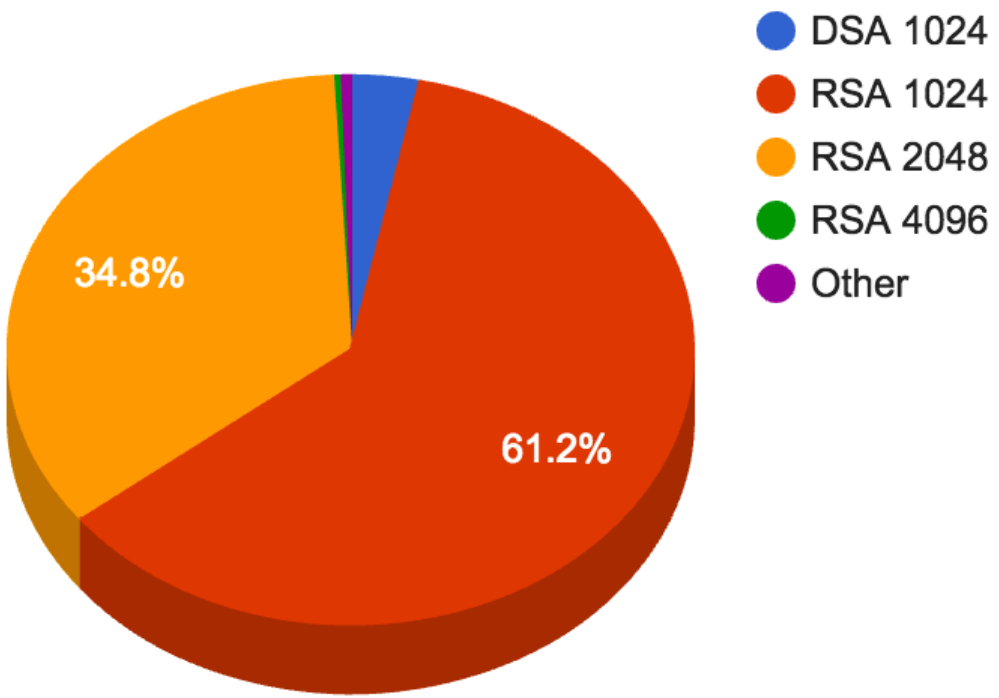


产品线转卖

Signed by Obama

Weak Signing Algorithm

- 许多APP仍然在用RSA-1024签名
- NIST: RSA-1024 is considered deprecated for use after 2010
  - Equivalent in strength to 80bit symmetric keys



参考：https://guardianproject.info/2015/12/29/how-to-migrate-your-android-apps-signing-key

Weak Digest Algorithm

- openssl pkcs7 -print_certs -inform der -in CERT.RSA -out cert_file
- openssl x509 -in cert_file –text

```
Signature Algorithm: md5WithRSAEncryption
```

➢即使Google自己的App证书也有这个现象

Weak Digest Algorithm

```
/**
 * HISTORICAL NOTE:
 *
 * Prior to the keylimepie release, SignApk ignored the signature
 * algorithm specified in the certificate and always used SHA1withRSA.
 *
 * Starting with keylimepie, we support SHA256withRSA, and use the
 * signature algorithm in the certificate to select which to use
 * (SHA256withRSA or SHA1withRSA).
 *
 * Because there are old keys still in use whose certificate actually
 * says "MD5withRSA", we treat these as though they say "SHA1withRSA"
 * for compatibility with older releases.  This can be changed by
 * altering the getAlgorithm() function below.
 */
```

```
if ("SHA1withRSA".equals(sigAlg) ||
    "MD5withRSA".equals(sigAlg)) {
    return USE_SHA1;
```

Weak Digest Algorithm

- openssl pkcs7 -print_certs -inform der -in CERT.RSA -out cert_file
- openssl x509 -in cert_file –text

```
Signature Algorithm: md5WithRSAEncryption
```

√ apksign碰到这种scheme其实是用SHA1来做APK签名

X 但是级联证书之间仍然可以通过MD5碰撞来替换签名证书

小结

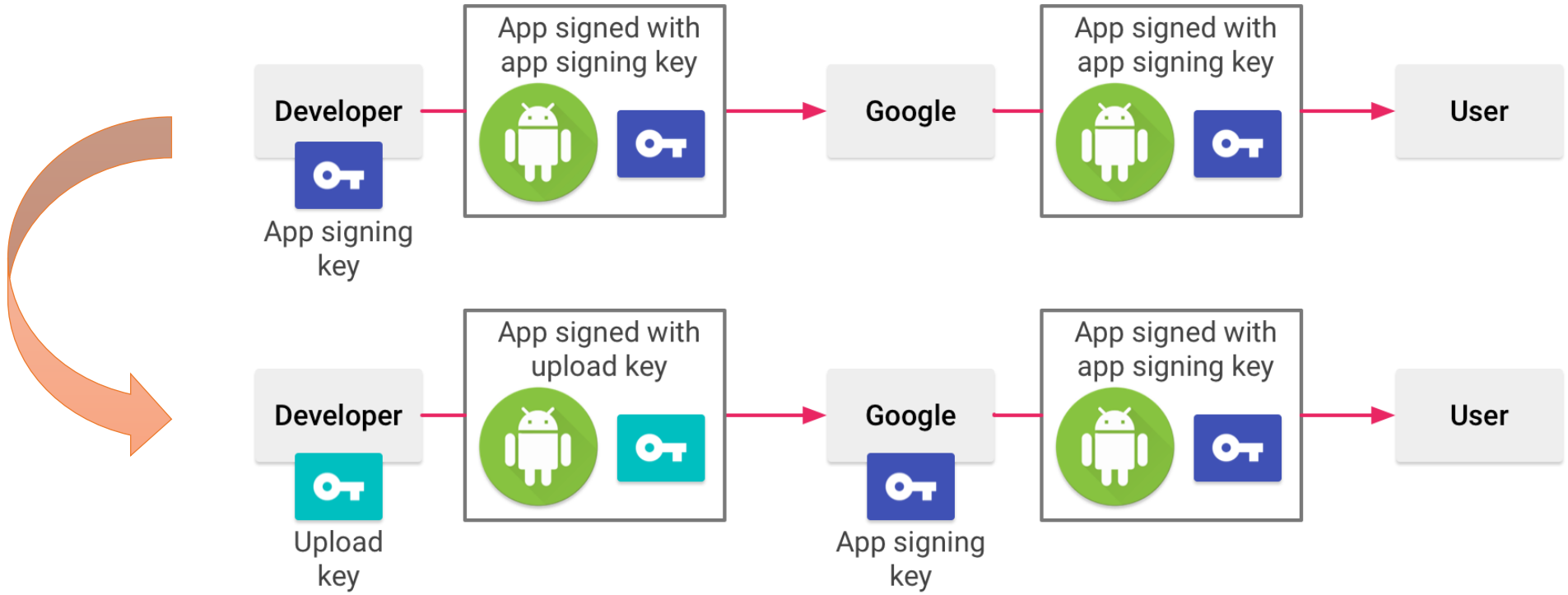
- 应用签名是Android生态的基石
- 应用签名证书（私钥）具有“不可遗失”、无法更换的特点

然而有时候却不得不换。。。

- 私钥(被)共享
- 私钥被滥用
- 私钥泄漏
- 证书过期
- 产品线转卖
- 弱签名算法

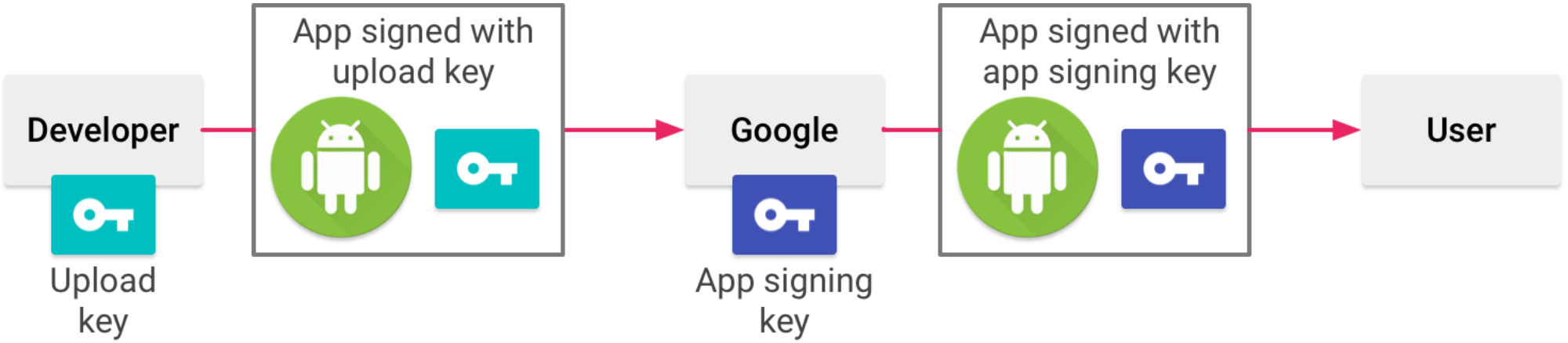
既有解决方案

New Google Play App Signing



1. You sign your app with your upload key.
2. Google verifies and removes the upload key signature.
3. Google re-signs the app with the signing key you provided.

New Google Play App Signing

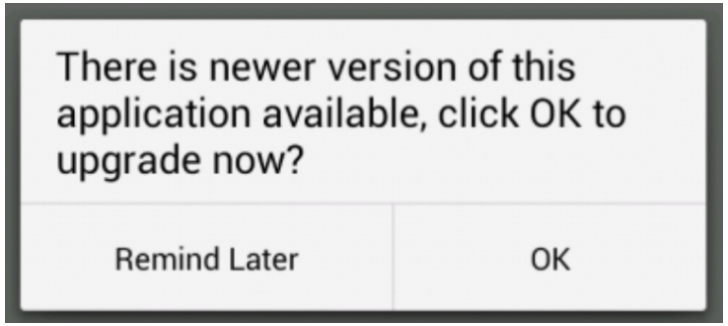


√ Signing key由Google代为保护
√ Upload key遗失可以找回

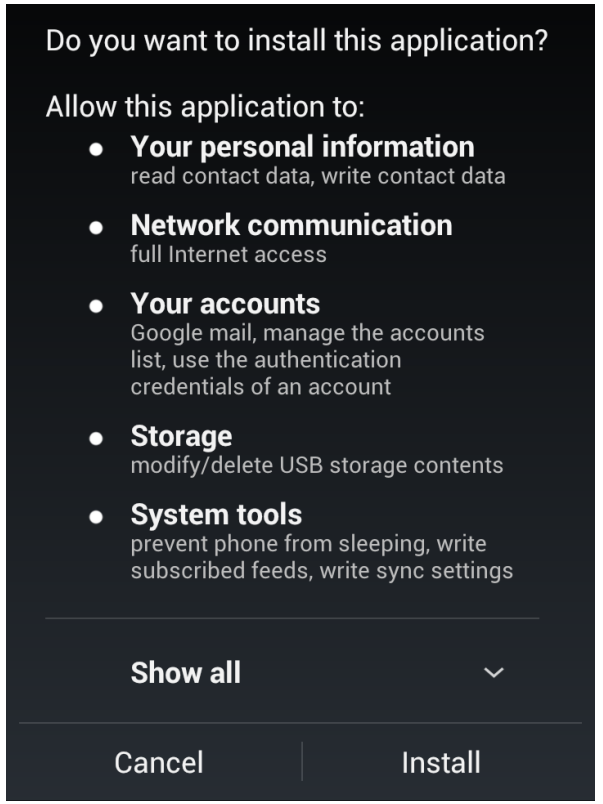
x 仍然无法更换签名证书
x 你真的愿意把私钥交给Google？
x 如何证明你是你？（钓鱼撞库、偷天换日）

手动升级

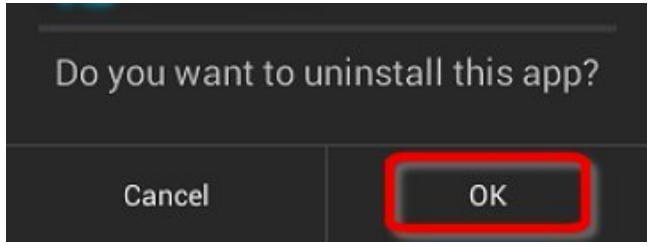
- 无法直接覆盖升级，影响用户体验



1. 旧APP弹窗引导用户安装新APP



2. 用户像安装不相干APP一样安装新APP



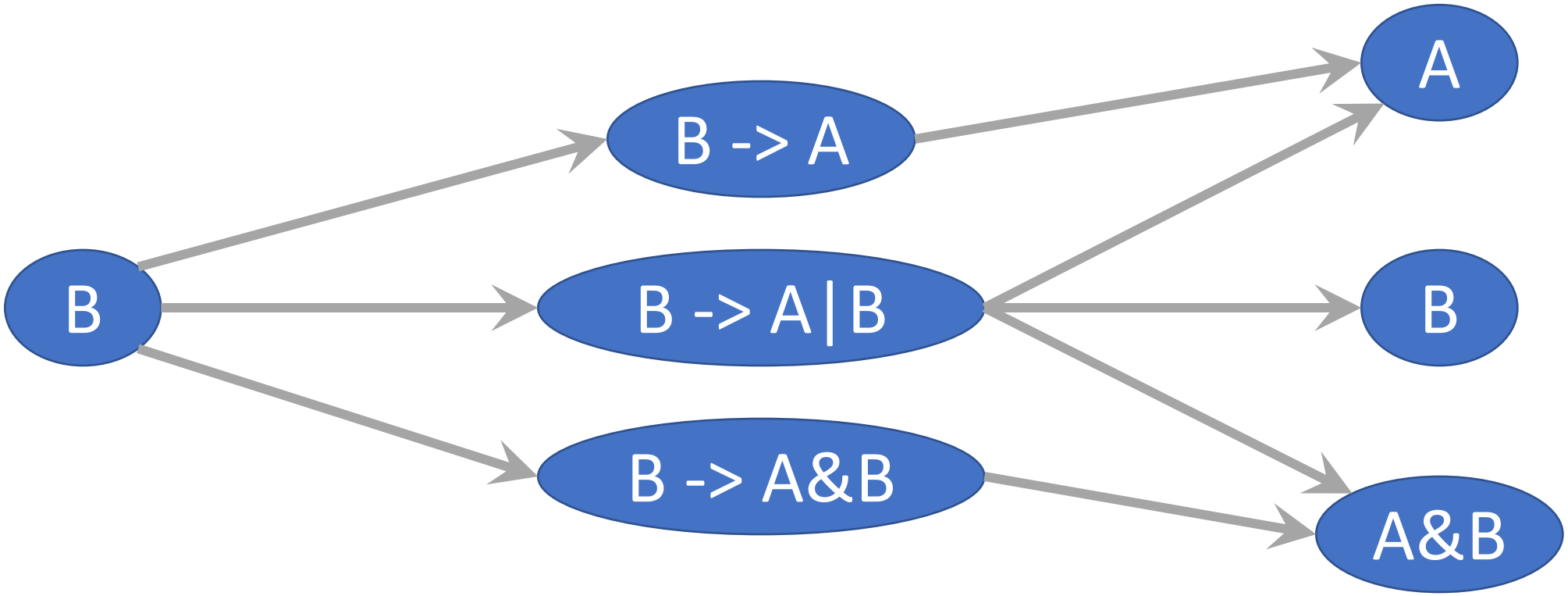
3. 新APP引导用户删除旧APP

自动升级

- 需要Android >= 5.0，不兼容老平台

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.android.frameworks.coretests.keysets">
    <application android:hasCode="false">
    </application>
    <key-sets>
        <key-set android:name="A" >
          <public-key android:name="keyA"
                      android:value="MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
        </key-set>
        <upgrade-key-set android:name="A"/>
    </key-sets>
</manifest>
```

自动升级 – 几种场景



自动升级 – 缺点

- 不兼容Android<5.0
- 之前所述不校验证书过期、不校验证书链等问题仍然存在
- 不能隔代升级（影响用户体验）
  - 只能B升级B->A再升级到A，不能直接B升级到A
- 对于B -> A|B的情况，攻击者可以劫持升级回旧签名

APP签名横向比较

Android

- 签名：开发者证书签名
- 校验："校验"证书链到自签名证书
- 过期检查：不检查
- 吊销检查：（没有机制）
- 换证书覆盖升级：不允许

iOS App Store

- 签名：开发者签名提交，下发由苹果统一签
- 校验：校验证书链到苹果的根证书
- 过期检查：不检查
- 吊销检查：（不适用）
- 换证书覆盖升级：（没有机会）

❖有远程禁用App黑名单：

https://iphone-services.apple.com/clbl/unauthorizedApps

iOS App Store – 过于强势的封闭生态掌控

- 安全App下架事件
- IAP封锁事件
- 禁止hot patch
- ……

- 你没有话语权
- 你可能是下一个

iOS Enterprise

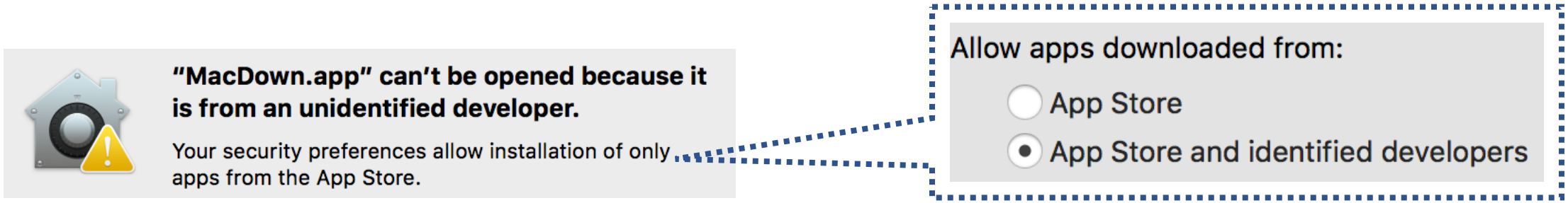
- 签名：开发者证书签名
- 校验：校验证书链到苹果的根证书
- 过期检查：安装检查，运行不检查
- 吊销检查：安装检查，运行不检查
- 换证书覆盖升级：允许（Masque Attack，已修复）

Mac App Store

- 签名：开发者签名提交，下发由苹果统一签
- 校验：校验证书链到苹果的根证书
- 过期检查：不检查
- 吊销检查：不检查
- 换证书覆盖升级：（没有机会）

Mac Applications

- 签名：开发者证书签名
- 校验：校验证书链到本地可信CA
- 过期检查：不检查
- 吊销检查：检查
- 换证书覆盖升级：允许

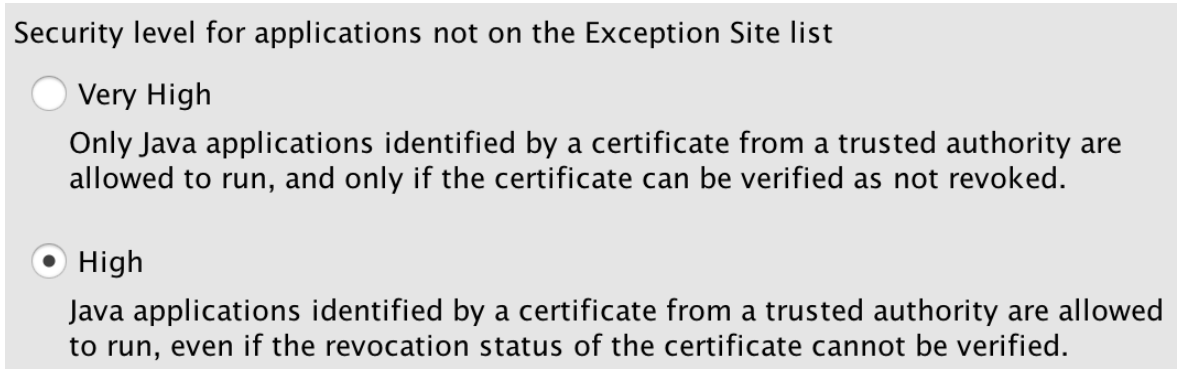


Windows Application/Driver

- 签名：开发者证书签名
- 校验：校验证书链到本地可信CA
- 过期检查：只要是在证书有效期内签的名即可，证书过期没关系
- 吊销检查：检查
- 换证书覆盖升级：允许

Java (JNLP)

- 签名：开发者证书签名
- 校验：校验证书链到本地可信CA
- 过期检查：检查
- 吊销检查：检查
- 换证书覆盖升级：允许

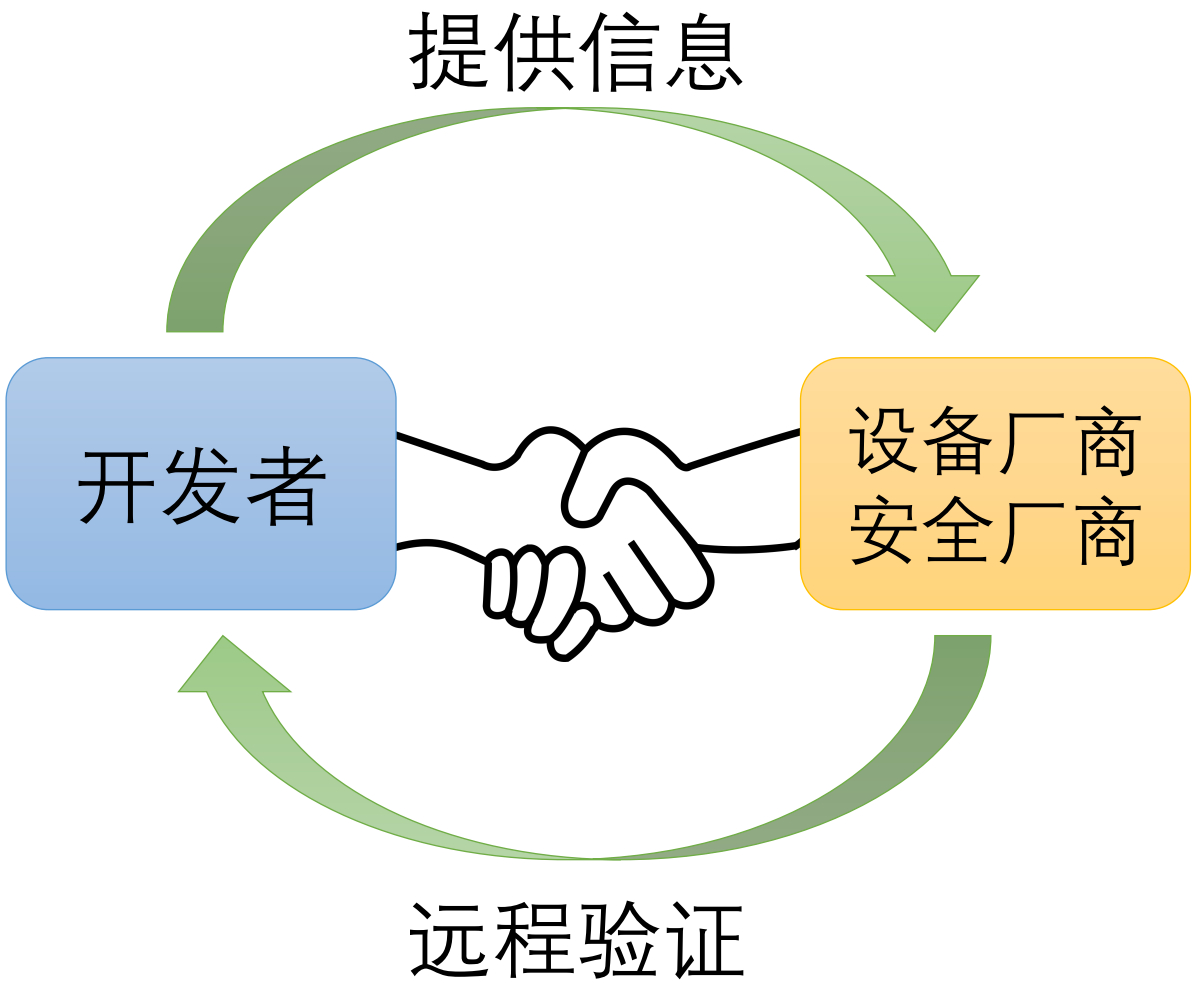


小结

- 应用签名是Android生态的基石
- 应用签名证书（私钥）具有“不可遗失”、无法更换的特点
- 然而有时候却不得不换

➢同时，还需要一个开放、去中心化、可在生态链的每一环进行动态验证的机制

开放、去中心化的签名生态



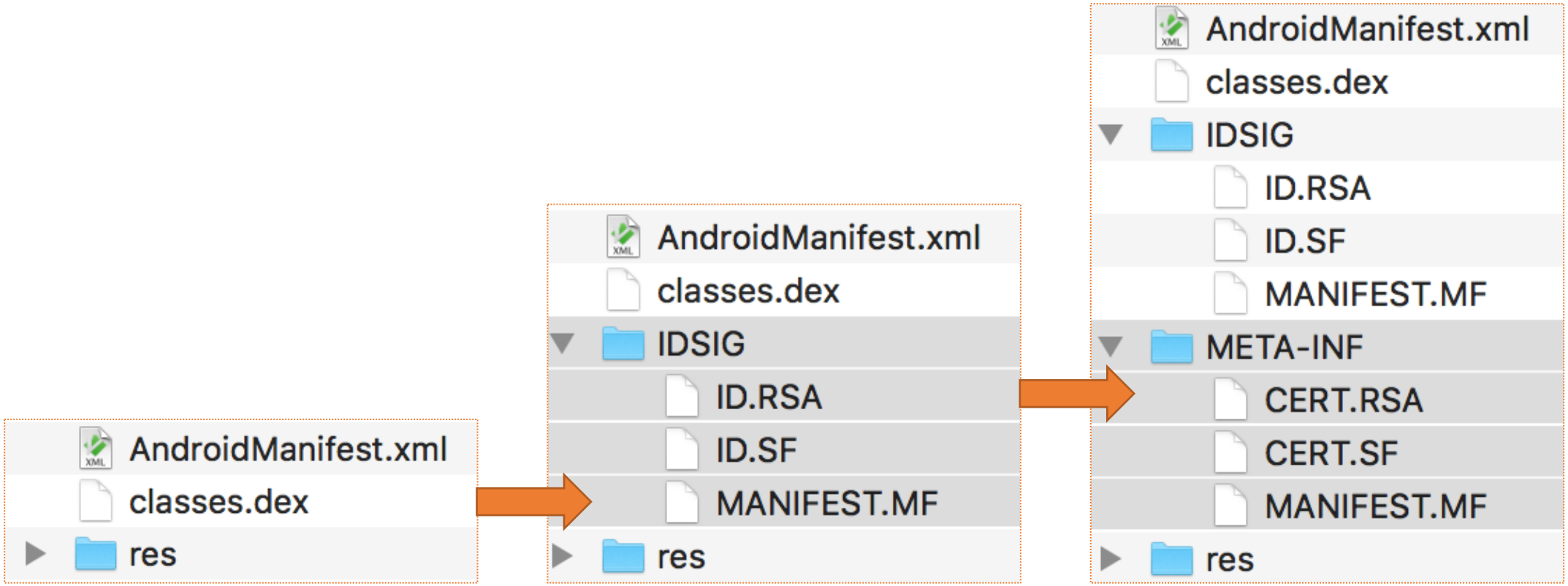
OASP: 救赎与革新

新签名机制

- 兼容
  - 支持既有Android验签机制
- 灵活
  - 支持证书"换新”
  - 支持跨代升级
- 安全
  - 过期、吊销检查
- 动态
  - 可动态更新验证信息
- 生态联防
  - 不产生新的中心依赖，避免生态独裁

新签名机制

新签名机制 -- IDSIG


AndroidManifest.xml
classes.dex
res
AndroidManifest.xml
classes.dex
IDSIG
ID.RSA
ID.SF
MANIFEST.MF
res
AndroidManifest.xml
classes.dex
IDSIG
ID.RSA
ID.SF
MANIFEST.MF
META-INF
CERT.RSA
CERT.SF
MANIFEST.MF
res

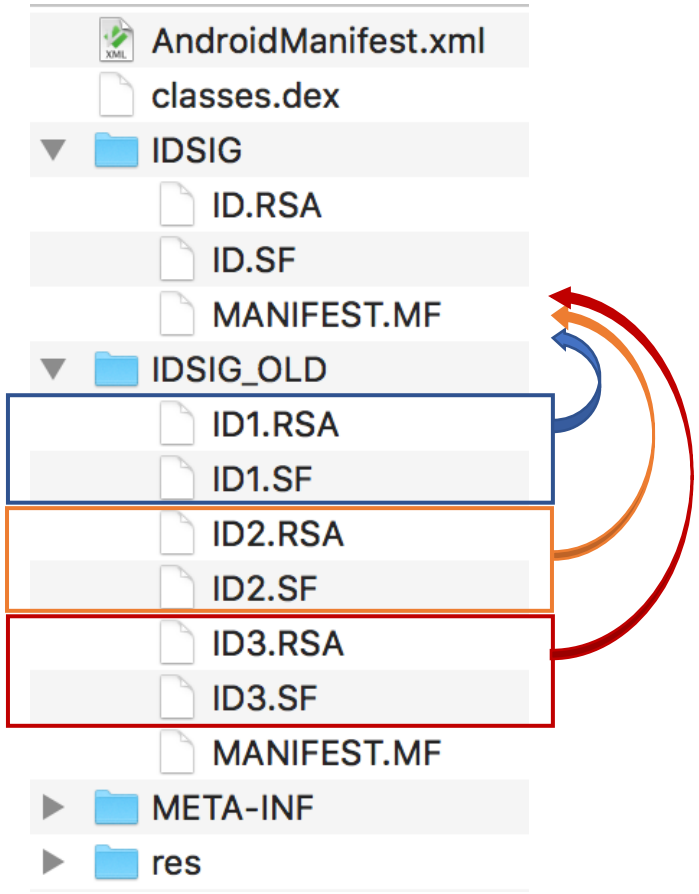
新签名机制 -- IDSIG

- IDSIG_OLD(可选) → 旧IDSIG证书对APK的签名
- IDSIG → 新IDSIG证书对IDSIG_OLD和APK签名
- META-INF → 正常APK签名机制签名（对IDSIG_OLD/IDSIG/APK签名）

- 兼容既有App升级校验机制
  - META-INF安装证书一致
- 为设备厂商提供了额外校验层
  - 新旧App的IDSIG证书匹配
  - 如果新App有IDSIG_OLD目录，则新App可替换用IDSIG证书或者IDSIG_OLD证书做IDSIG签名的旧App

灵活 -- 支持证书"换新"

支持证书跨代升级



安装证书一致的前提下
可从ID/ID1/ID2/ID3签名的
旧App更新到ID签名的App

新机制优势

- 兼容
  - 支持既有Android验签机制
- 灵活
  - 支持证书"换新”
  - 支持跨代升级
- 安全
  - 过期、吊销检查
- 动态
  - 可动态更新验证信息
- 生态联防
  - 不产生新的中心依赖，避免生态独裁

安全 -- 参考CRL，OCSP

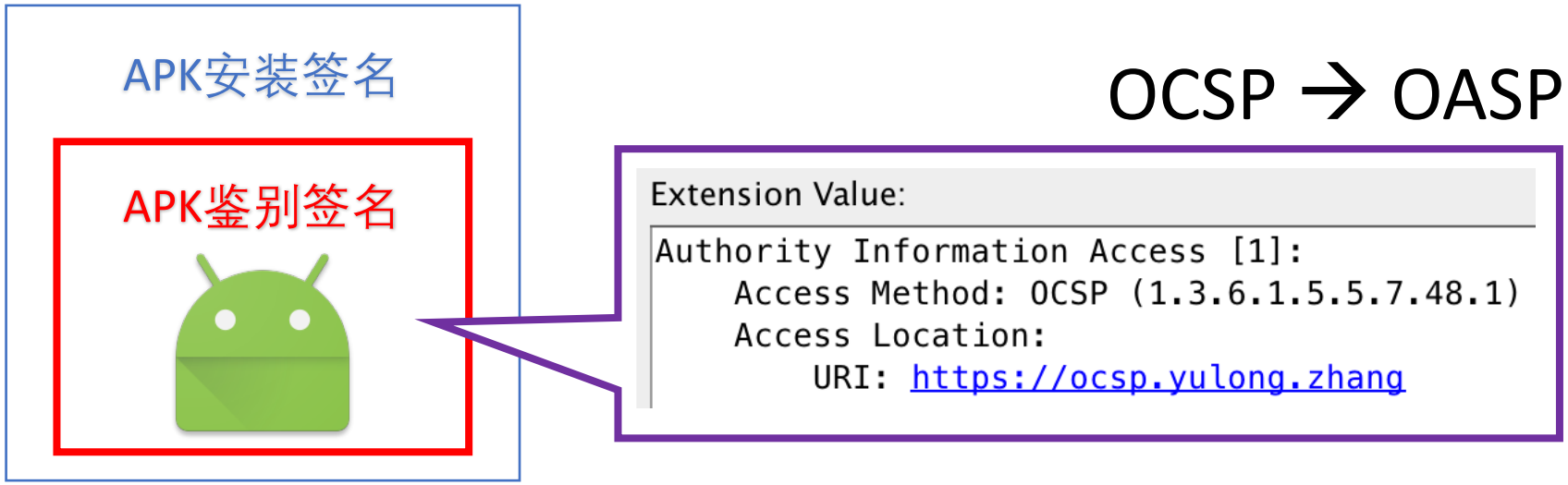
CRL

```
Extension Value:
Authority Information Access [1]:
    Access Method: CRLDistributionPoints (2.5.29.31)
    Access Location:
        URI: https://crl.yulong.zhang/my.crl
```

OCSP

```
Extension Value:
Authority Information Access [1]:
    Access Method: OCSP (1.3.6.1.5.5.7.48.1)
    Access Location:
        URI: https://ocsp.yulong.zhang
```

Online App Status Protocol



OASP Server -- GET

- 返回OASP server支持的协议版本

OASP Server -- POST

- 上传
  - 安装证书、鉴别证书
  - Apk哈希值
  - 包名
  - 版本号
- 返回（其中一个）
  - App可信
  - App不可信
  - 未知
  - 新的OASP验证地址

Why OASP?

- 在APK证书链上叠加Web (HTTPS)证书链保障
  - ✓有了Root of Trust
  - ✓有了动态APP黑白名单处理的渠道
  - ✓给了设备/安全厂商有机会在流通的所有环节做检查
  - ✓缓解了OCSP安装检查无用的问题
  - ✓APP开发者和安全厂商之间有了更良性的互动
- HTTPS证书链提供了丰富可靠的信息，可供追溯

断网/DOS情形的处理

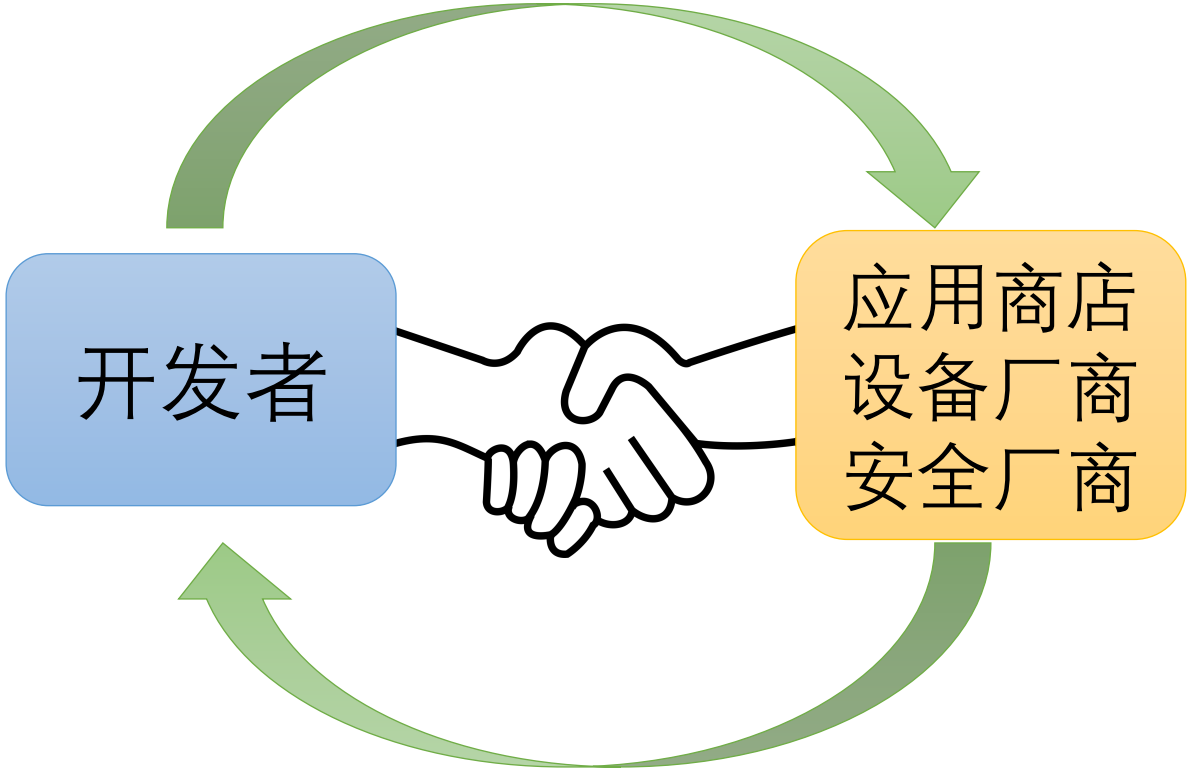
- APK证书验证和时效性没有Web那么强
  - 安装链上、生态的每一环都能做检查

新机制优势

- 兼容
  - 支持既有Android验签机制
- 灵活
  - 支持证书"换新”
  - 支持跨代升级
- 安全
  - 过期、吊销检查
- 动态
  - 可动态更新验证信息
- 生态联防
  - 不产生新的中心依赖，避免生态独裁

生态联防

开发厂商及时提供在线状态信息



远程验证

生态联防 -- 应用商店

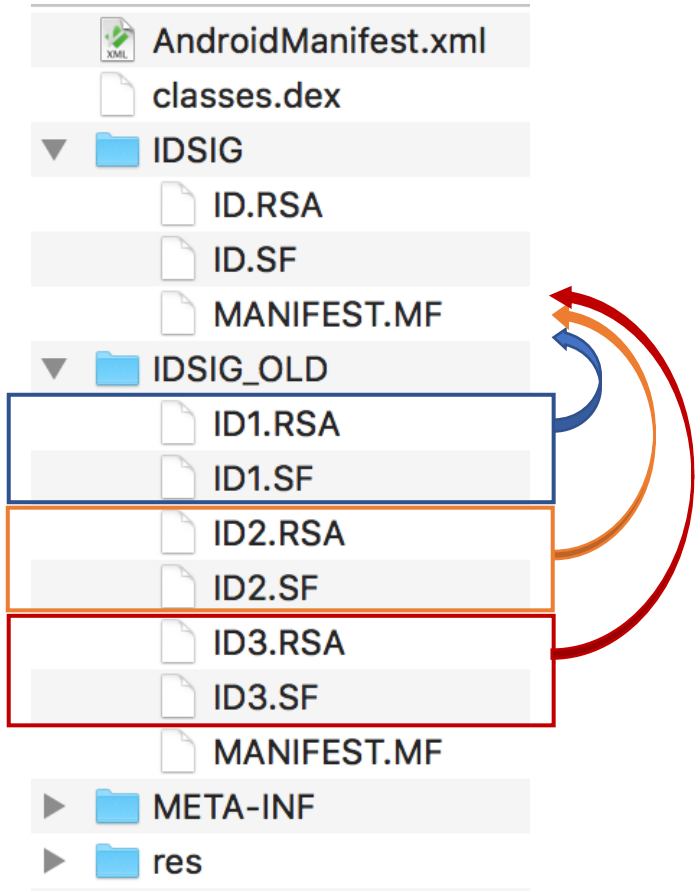
- 开发者可以提交、设置OASP信息
- 应用更新时，应用商店根据OASP信息验证新App
- 应用商店可定期扫描App的OASP状态，下架有问题的App
- 第三方可以从应用商店获取OASP信息

生态联防 -- 安全厂商

- 云端：
  - 扫描监控IDSIG签发信息和OASP HTTPS证书信息，生成相关reputation
- 设备端：
  - App扫描时结合OASP状态及reputation，判断App是否恶意

生态联防 -- 设备厂商

- 在兼容既有App升级校验的基础上，提供了IDSIG这一额外校验层
  - 支持(跨代)证书升级



基于OASP的安全生态

APK安装签名

APK鉴别签名

- 证书遗失：更换鉴别签名
- 安装签名共享：细分鉴别签名
- 产品转卖：更换鉴别签名
- ……

应用商店、设备厂商、安全厂商有了更可靠的依据，不会被单一厂商掐住脖子

基于OASP的安全生态

束缚 V.S. 保护

IDSIG优势

- 兼容
  - 支持既有Android验签机制
- 灵活
  - 支持证书"换新”
  - 支持跨代升级
- 安全
  - 过期、吊销检查
- 动态
  - 可动态更新验证信息
- 生态联防
  - 不产生新的中心依赖，避免生态独裁

参考实现

参考实现

- https://github.com/baidu/OASP
- 欢迎大家一起来合作、改进

Thanks!

百度安全实验室 Baidu X-Lab

2017.06