软件学报 ISSN 1000-9825 CODEN RUXUEW
Journal of Software,

E-mail: jos@iscasac..cn
http://www.jos.org.cn
Tel: +86-10-62562563

标准模型下可证明安全的 APK 二次开发授权机制

李道丰, 陈海强, 梁家荣, 赵搏文

(广西大学 计算机与电子信息学院, 广西 南宁 530004)
通讯作者:李道丰, E-mail: ldf_0123@163.com

摘 要: 为了解决 Android APK 文件有效性和版权问题, 需要签名才能发布. 然而当第三方申请对原生 APK 文件进行二次开发和修改授权时, 如何指定第三方的开发和修改权限以及确定 APK 文件二次开发完成后出现的版权问题仍是有待解决的问题. 为此, 文中提出一种细粒度的在标准模型下可证明安全的 APK 授权机制（APK-SAN）. APK-SAN 授权机制主要采用基于身份的可净化签名技术的特有属性, 允许原生 APK 文件的开发者授权给第三方（指定修改者）对 APK 文件的许可区域或位置进行二次开发或修改, 且修改后生成的新 APK 文件的签名仍然有效.分析结果表明, 所提的 APK-SAN 授权机制无需证书存储和管理、提供细粒度修改授权功能, 可维护原开发者和修改者双方的合法权益, 减少了开发者的通信开销和计算开销.

关键词: 可净化签名; APK 签名机制; 二次开发; 基于身份的签名; 标准模型
中图法分类号: TP309.7 **文献标识码**: A DOI:

Provable Secure Authorization Mechanism for the APK Redevelopment in the Standard Model

LI Dao-Feng, CHEN Hai-Qiang, LIANG Jia-Rrong, ZHAO Bo-Wen

(Guangxi University, Nanning 530004, China)
Corresponding author: LI Dao-Feng, E-mail: ldf_0123@163.com

Abstract: Signatures are used to address the validation and authorization issues of the APK file before publishing. When the modifier apply for the right to redevelop APK files how to authorize and ascertain the authorization issues are very important problems which have not been solved. In this work, a new APK authorization mechanism (APK-SAN) is proposed using the sanitizable signature scheme. APK-SAN authorization mechanism utilizes unique properties of sanitizable signature technology that allows original developer to authorize specified modifier to redevelop the designated part of source code of the APK file without interaction between developer and modifier. Moreover, APK-SAN authorization mechanism does not require to storage and management of Certification. Our scheme reduces communication overhead and computational overheads of the original developer. The signature of new APK files after redevelopment is still valid. This maintains the copyright of original developer and modifier.

Key words: Sanitizable Signature; Provable Secure; Redevelopment; ID-Based Signature; Standard Model

---

基金资助: 国家自然科学基金(61662004);广西自然科学基金(2016GXNSFAA380215)
收稿时间: 2017-03-30; 修改时间: 2017-09-26; 采用时间: 2018-11-13; jos 在线出版时间: 2019-05-22
CNKI 网络优先出版: 2019-05-22 15:26:19, http://kns.cnki.net/kcms/detail/11.2560.TP.20190522.1525.011.html

1 引言

由于 Android 的开放性和可扩展性, Android 智能手机得到广泛应用, 受到大量用户、终端厂商和开发者的支持. 然而 Android 智能手机存在不少安全问题, 如用户敏感信息泄露、会话窃听和劫持、恶意吸话费和流量、恶意广告推送等, 对移动互联网的发展和用户的信息安全带来了极大的威胁. 可知, Android 安全问题是移动互联网安全研究中亟待解决的研究课题之一[1-14].

为保护 Android APK 合法性、版权和权限问题, Android 系统采用部分安全机制, 主要包括签名机制[8,15,16,17]、沙盒机制[13,18]和权限管理机制[7,10,12,14]. 其中, 签名机制要求 APK 开发者在发布 APK 应用程序之前先进行数字签名, 要求开发者使用自签名的数字证书对编译打包的 APK 文件进行签名, 确保 APK 文件的完整性、真实性和不可否认性, 同时也确保了 APK 文件发布机构的唯一性, 维护开发者的版权；沙盒机制要求所有的应用程序都是运行在 Dalvik 虚拟机中, 与 POSIX 用户安全机制一起实现自动管理, 确保 APK 的运行安全；权限管理机制主要是当应用程序在安装、运行等阶段时对应用程序以及 Android 系统的保护.Android 应用程序的授权权限分为 normal、dangerous、signature 和 signatureOrSystem 等 4 个级别,由开发者在 AndroidManifest 文件中定义和声明. 但 Android 应用程序仅有粗粒度权限, 很多应用程序的使用会泄露用户的敏感信息[9,11].Paul Pearce 等[10]针对广告程序会用户敏感隐私信息问题, 给出一种 Android 广告权限分离机制 AdDroid, 用于推送广告时保护用户敏感隐私信息. Jinseong Jeon 等[5]提出一种 fine-grained 安全策略用于解决应用框架层存在的粗粒度授权机制问题. 同时, 为了解决 Android 中 APK 存在的越权问题, A Dimitriadis 等[19]给出越权监测机制.

鉴于 Android 系统的普及性和开源特点, 越来越多的 app 需要进行二次开发使用, 例如支付宝 app、健康状态监测 app、天气预报 app 等, 在不同的机构、地区或人群的使用中需要对原生的 app 进行二次开发定制. 当第三方对原生的 app 进行二次开发后, 可能出现在二次开发时第三方在修改后的 app 嵌入恶意代码、修改后的 app 的版权归属等问题. 这是与第三方交互处理后衍生出来的安全问题, 在实际应用研究中无法避开. 赵搏文等[22]提出在授权第三方进行二次开发时利用传递签名传递信任关系, 确保二次开发后的 app 的版权归属问题. 该方法在授权时允许第三方具有自由开发和修改的权限, 即第三方能随意修改原生 app. 在实际应用需求中, 为防止二次开发中嵌入恶意代码以及明确版权归属问题, 在授权给第三方进行二次开发时, 需限定 APK 文件中哪一部分能修改, 哪一部分不允许修改. 为此, 本文提出标准模型下可证明安全的 Android APK 授权机制, 简称 APK-SAN 授权机制. 该授权机制采用基于身份的可净化签名技术[20,21]的特有属性,允许原生 APK 文件的开发者授权给第三方（指定修改者）对 APK 文件的许可区域或位置进行二次开发或修改, 且修改后生成的新 APK 文件的签名仍然有效, 同时能解决修改后的 APK 文件的版权归属问题, 能清晰界定开发者和修改者的权益责任. 若修改者在修改过程中嵌入恶意代码, 可通过授权证据明确责任, 从而一定程度上限制修改过程中在 app 嵌入恶意代码行为. 该机制在授权给第三方后无需继续进行双方交互, 而且无需用户证书, 避免证书管理和存储开销, 能提供细粒度授权, 执行效率高．APK-SAN 授权机制具有特点如下：

(1) 提供细粒度授权．当第三方提出对原生 APK 二次开发申请时, 源开发者结合自身与第三方的实际

情况和需要, 设置允许第三方修改的文件和位置的权限, 并给与授权. 设置权限时采用基于身份的可净化签名方案, 若第三方遵循权限授权规定, 在指定的文件或位置进行二次开发, 则修改后的 APK 合法有效; 若未在指定文件或位置修改, 则修改后的 APK 的签名验证不通过, 可检测越权二次开发行为. 因此具有细粒度授权功能.

(2) 可证明安全．APK-SAN 授权机制主要采用基于身份的可净化签名技术来实现权限授权, 侧重解决二次开发过程中所面临的安全问题和版权问题, 本文主要考虑签名不可伪造安全模型和不可改变安全模型, 并在标准模型下给出安全证明.

2 预备知识

2.1 Diffie-Hellman问题

设阶均为 q 的加法群 G_1 和乘法群 G_2, g 为 G_1 的生成元, 存在映射 e: $G_1\times G_1\to G_2$, 具有以下性质：

(1) 双线性：对于$\forall a, b \in Z_q$, $P, Q \in G_1$, 有 $e(P^a, Q^b) = e(P, Q)^{ab}$;

(2) 非退化性：存在 $P, Q \in G_1$, 使得 $e(P, Q) \neq 1$;

(3) 可计算性：$\forall P, Q \in G_1$, 存在计算 $e(P, Q)$的有效算法.

定义 1. 计算 Diffie-Hellman(Computational Diffie-Hellman, CDH)问题：已知有限加法群 G_1, 给定元素 g, g^a, $g^b \in G_1$, 其中 $a,b \in \mathbb{Z}_q^*$ 且 a, b 未知, 求 g^{ab}.

定义 2. (ε, t)-CDH 假定：若不存在任何概率多项式算法能在时间 t 内,以至少概率为 ε 解决群 G_1 上的 CDH 问题,则称群 G_1 上的(ε, t)-CDH 假定成立.

显然, (ε, t)-CDH 假定成立下, CDH 问题是一个难题[23].

2.2 Android原生签名机制

未签名的 APK 文件通常由 res 文件夹、lib 文件夹、resources.arsc 文件、classes.dex 文件以及 AndroidManifest.xml 文件组成, 如图 1 所示. 这些文件（夹）在二次开发时有可能需要修改, 也有可能不需要修改, 授权时需要明确规定.

各文件（夹）存放的资源如下：

(1) res 文件夹：存放资源文件,如图片、布局文件等;

(2) lib 文件夹：存放.so 动态链接库, .so 动态链接库是由 JNI 层的 native 代码编译链接生成的可执行文件, 由 Linux 内核执行. 若工程源码中没有 native 代码, 则 APK 文件中没有此文件夹;

(3) resources.arsc 文件：编译后的二进制资源文件;

(4) classes.dex 文件：Dalvik 可执行文件, 由 Dalvik 虚拟机执行. classes.dex 文件的生成首先 Android 编译器编译工程源代码, 生成相应的 class 文件, 再由 dx 工具转换所有的 class 文件, 生成 classes.dex 可执行文件;

(5) AndroidManifest.xml 文件：全局配置文件.

res 文件夹
lib 文件夹
resources.arsc 文件

classes.dex 文件
AndroidManifest.xml 文件

图 1 未签名 APK 文件格式

Figure 1. The structure of APK file which unsigned

Android 系统规定所有发布的 APK 文件须事先由开发者对其进行签名, 才能安装. 对编译打包后的 APK 文件进行签名旨于确保 APK 文件的完整性、真实性和不可否认性, 同时确保开发者的权益.

APK 文件签名机制流程如下：

(1) 生成 MANIFEST.MF 文件. 遍历 APK 文件中的所有文件, 利用 SHA-1 哈希算法对图 1 的文件逐个生成散列值, 并对每一项散列值进行 Base64 编码, 保存在 MANIFEST.MF 文件中.

(2) 生成 CERT.SF 文件. 使用 SHA-1 哈希算法计算 MANIFEST.MF 文件及其内容的散列值并进行 Base64 编码, 生成 CERT.SF 文件.

(3) 生成 CERT.RSA 文件. 利用 RSA 签名算法和开发者的私钥对 CERT.SF 文件签名, 并建立 CERT.RSA 文件, 用于保存签名和开发者自签名的公钥证书.

(4) 打包所有生成的签名文件. 将新生成的 MANIFEST.MF 文件、CERT.SF 文件及 CERT.RSA 文件打包到 APK 文件的“/META-INF”目录下.

2.3 基于身份的可净化签名方案

可净化签名方案[20]是签名者允许指定的修改者在无需相互交互的情况下修改文档的指定部分内容, 且修改后的文件的签名仍然合法有效. 本文主要采用基于身份的可净化签名方案, 该方案主要由签名者(signer)、可净化者(sanitizer)和验证者(verifier) 等三个参与方共同完成. 方案实现如下：

假定用户身份 $ID=(ID_1,ID_2,\ldots,ID_n)\in\{0,1\}^{\lambda}$,签名消息 $M=(M_1,M_2,\ldots,M_n)\in\{0,1\}^{\lambda}$,签名者完成如下操作：

(1) 系统参数建立(Setup). 给定一个安全参数 k,私钥生成器(PKG)生成系统参数 *params* 和主密钥 *msk*, 其中 *params* 公开,*msk* 保密.

(2) 用户密钥生成(Extract). 给定一个用户及其身份 *ID*,PKG 利用主密钥 *msk* 计算该 *ID* 的用户私有密钥 d_{ID},并将 d_{ID} 通过安全信道秘密发送给用户.

(3) 签名(Sign). 输入系统参数 *params*,签名者身份 *ID*, 消息 M 以及签名者的私钥 d_{ID}, 签名者输出签名 σ 以及秘密信息 ψ.

(4) 净化(Sanitize). 输入系统参数 *params*,净化者身份 ID', 秘密信息 ψ 以及消息 M 的签名 σ. 净化者输出新的消息 M'及其签名 σ'.

(5) 验证(Verify). 输入系统参数 *params*,签名者身份 ID_i, 以及消息/签名(M'_i,σ'_i)以及(M_i,σ_i), 利用签名验证算法进行验证, 输出有效*“True”*或无效*“False”*.

2.4 安全模型

结合可净化签名的安全模型, 以及 APK-SAN 授权机制提供的安全机制, 我们给出不可伪造性和不可改变性两个安全属性的安全模型.

不可伪造性(Unforgeability)：方案中由签名者(signer)和净化者(sanitizer)所产生的签名不可伪造.

不可改变性(Immutability)：sanitizer 只能修改文档中由签名者 signer 指定的部分.

在攻击过程中, 对于不可伪造性和不可改变性,挑战者 C 和攻击者 A 的攻击交互如下：

(1) 系统参数建立；

(2) 私钥请求. 攻击者 A 指定一个身份为 ID 的 APK 开发者, 挑战者 C 运行密钥生成算法得到身份为 ID 的 APK 开发者或修改者的私钥 d_{ID}, 并将 d_{ID} 发送给攻击者 A；

(3) 签名请求. 攻击者 A 指定一个身份为 ID 的 APK 开发者或修改者以及 APK 文件 M 以及签名者的私钥 d_{ID}, 挑战者 C 运行密钥抽取算法和签名算法得到(ID, M)的签名 σ 并发给攻击者 A.

(4) 签名伪造和修改. 攻击者 A 能成功修改关于一个身份为 ID' 的 APK 开发者或修改者的签名 σ', 使得：

① 攻击者 A 未进行过关于身份为 ID' 的 APK 开发者或修改者的私钥提取请求；

② 攻击者 A 未对已开发的 APK 进行过签名请求（不可伪造性）；

③攻击者 A 能对 APK 中未指定（允许修改）的部分进行签名且其签名 σ' 有效（不可改变性）.

若攻击者 A 成功伪造或修改 APK 签名,则其可以解决 CDH 问题[23]的实例.

3 APK-SAN 授权机制

为了便于机制正常运行, 本文假设：通常情况下, 若有修改者提出对原生 APK 文件二次开发申请, 开发者均可授权修改者对其已发布的 APK 文件的指定文件进行修改. 同时, 本文所提到的“二次开发”均与“修改”同义. 用户代表修改者角色. APK-SAN 授权机制中所提到的细粒度授权定义如下：

定义 1. 细粒度授权是指开发者具备授予修改者对图 1 中指定的若干个 APK 文件进行二次开发或修改的权限.

本节结合基于身份的可净化签名方案, 给出一种细粒度的 APK 二次开发授权机制 APK-SAN. APK 二次开发申请授权流程如图 2 所示, 具体的流程步骤如下：

(1) 发布应用. 开发者发布已签名的 APK 文件到应用商城；

(2) 用户体验. 用户从应用市场下载 APK 文件进行体验, 在体验过程中可能根据自己实际需要提出二次开发请求. 如发现 APK 存在 bug 影响用户体验, 需要对其进行修复；或者用户需要在原生 APK 上增加相应的业务功能完成个性化定制；

(3) 授权请求. 用户向开发者发送对原生 APK 进行二次开发授权请求；

(4) 授权. 开发者对用户进行授权, 指定修改者可以修改哪些文件, 不可以修改哪些文件. 此步骤中采用基于身份的可净化签名方案来实现, 具体实现详见 3.2 小节的内容；

(5) 发布修改应用. 用户首先验证开发者的授权合法性与有效性, 具体实现详见 3.3 小节的内容; 然后根据开发者给予的修改授权, 结合自身需求对原生 APK 相应文件进行修改; 再次对修改后的新 APK 文件打包并签名; 最后将其重新发布到应用市场, 具体实现详见 3.4 小节的内容.

其中,上述步骤（1）、（4）、（5）的工作均按照 Android 原生签名和签名验证步骤完成。限于篇幅,我们仅在第 3 节中介绍核心的工作。

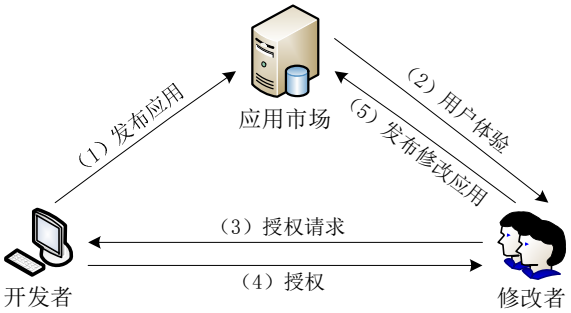

图 2 APK 文件二次开发申请授权流程
Figure 2. The process of applicant to revise for the APK file

其中, 第（4）、（5）步的操作就是利用我们提出的 APK-SAN 授权机制来实现. APK-SAN 授权机制由密钥生成、APK 文件二次开发授权、二次开发过程和修改合法性验证等四部分组成, 具体实现如下：

3.1 密钥生成

系统参数生成. 给定安全参数 k, PKG 选取阶为 q 的两个群 G_1 和 G_2, 一个生成元 g, 以及可允许的双线性配对 e: $G_1 \times G_1 \to G_2$, 其中 q 为大于 1024bit 的素数. 而后随机选取整数 $\alpha \in Z_q^*$, 计算 $g_1 = g^{\alpha}$ 并选取 $g_2 \in G_1$. PKG 选择两个元素 $u', m' \in G_1$ 以及两个含有 n 个元素的向量 $\boldsymbol{U} = (u_i), \boldsymbol{V} = (v_i)$, 其中向量的元素均随机取自于群 G_1. 从而得到系统参数为 $params$=(G_1, G_2, e, q, g, g_1, g_2, u', m', $\boldsymbol{U}$, $\boldsymbol{V}$), 以及系统主密钥为 g_2^{α}. 这里要求 PKG 是绝对安全可信机构,具有高安全的 Android 官方应用市场的管理机构,如华为、小米或 Google Play 等。

用户密钥：每个开发者将自己的身份信息 ID_i= $Hash(Com_i \| IDE_i \| t_i)$做为自己的公钥, 将 ID_i 表示为 $ID_i = \{ID_i^1, ID_i^2, \cdots, ID_i^n\}$, 并发给 PKG, 其中 Com_i 为该开发者所在的公司名称数据形式化表示, IDE_i 为该开发者的唯一标识号; t_i 为开发者公钥的有效期. 其私钥 d_{ID_i} 生成如下：PKG 随机选取一个整数 r_{ID_i}, 并按公式(1)计算：

$$d_{ID_i} = (d_{ID_i}^{(1)}, d_{ID_i}^{(2)}) = (g_2^{\alpha}(u' \prod_{j=1}^{n} u_j^{ID_i^j})^{r_{ID_i}}, g^{r_{ID_i}}) \quad (1)$$

其中 j=1, 2, … , n. 此步中包括 APK 文件修改者的公钥和私钥的生成. 限于篇幅, 我们不再一一给出.

3.2 生成并发布APK文件

授权的实现就是对指定修改者和指定可修改文件进行签名. 在发布 APK 文件之前, 开发者结合自身实际情况, 指定允许修改的文件（夹）和程序源代码的授权操作.

（1）生成 APK 文件. 当开发者完成 Android 应用程序开发后, 将 Java 文件、资源 xml 文件、Class 文件和 Dex 文件打包生成 APK 文件. 进而对 APK 文件中的 res 文件夹、lib 文件夹、resources.arsc 文件、classes.dex 文件和 AndroidManifest.xml 文件依次逐个进行哈希运算和 base64 编码, 生成 Manifest.MF 和 CERT.SF 文件. 具体如下：

Manifest.MF 文件　　CERT.SF 文件

$H_1 = Base64(\ Hash(\ d_{\text{des}} \parallel res \parallel ID_{\text{des}}\ ))$
$H_2 = Base64(\ Hash(\ d_{\text{des}} \parallel lib \parallel ID_{\text{des}}\ ))$
$H_3 = Base64(\ Hash(\ d_{\text{des}} \parallel resources.arsc \parallel ID_{\text{des}}\ ))$
$H_4 = Base64(\ Hash(\ d_{\text{des}} \parallel \text{classes.dex} \parallel ID_{\text{des}}\ ))$
$H_5 = Base64(\ Hash(\ d_{\text{des}} \parallel AndroidManifest.xml \parallel ID_{\text{des}}\ ))$

$h_1 = Base64(\ Hash(\ d_{\text{des}} \parallel H_1 \parallel ID_{\text{des}}\ ))$
$h_2 = Base64(\ Hash(\ d_{\text{des}} \parallel H_2 \parallel ID_{\text{des}}\ ))$
$h_3 = Base64(\ Hash(\ d_{\text{des}} \parallel H_3 \parallel ID_{\text{des}}\ ))$
$h_4 = Base64(\ Hash(\ d_{\text{des}} \parallel H_4 \parallel ID_{\text{des}}\ ))$
$h_5 = Base64(\ Hash(\ d_{\text{des}} \parallel H_5 \parallel ID_{\text{des}}\ ))$

（2）APK 文件签名. 开发者随机选取两个整数 $\bar{r}_{ID_{des}}, r'_{ID_{des}} \in Z_q^*$，按如下等式(2)进行签名，生成 CERT.SAN 文件:

$$\begin{aligned}
&\sigma_1 = d_{ID_{des}}^{(1)} (v' \prod_{i=1}^{n} v_i^{h_i})^{r'_{ID_{des}}} = g_2^{\alpha} (u' \prod_{i=1}^{n} u_i^{ID_{des}^i})^{\bar{r}_{ID_{des}}} (v' \prod_{i=1}^{n} v_i^{h_i})^{r'_{ID_{des}}} \\
&\sigma_2 = d_{ID_{des}}^{(2)} = g^{\bar{r}_{ID_{des}}},\ \sigma_3 = g^{r'_{ID_{des}}} \\
&\sigma = (\sigma_1, \sigma_2, \sigma_3)
\end{aligned} \tag{2}$$

（3）开发者将 Manifest.MF、CERT.SF 和 CERT.SAN 文件打包生成 APK_SAN.SIG 文件，保存到 APK 文件的“/META-INF”目录下，并在应用市场上发布 APK 文件.

3.3 二次开发授权

授权的实现就是对指定修改者和指定可修改文件进行签名. 当开发者发布 APK 文件后，若收到用户对 APK 文件源代码进行二次开发的授权请求，则开发者结合自身实际情况给予的授权操作，具体步骤如下:

（1）开发者收到修改者的二次开发授权请求时，根据修改者提出的修改需求，分析需要修改的文件和 $i \in K = \{1,2,\cdots,n\}$ 程序源代码. 结合自身实际情况，指定允许修改的文件（夹）和程序源代码.假定允许修改的文件夹集为 $h' = \{h'_i\}$，其中 h'_i 为文件夹所对应的哈希编码，且 $h' \subseteq \{h_i\}$，;

（2）令 $K' = \{i \in K \mid h_i = h'_i\}$、$K'' = \{i \in K \mid h_i \neq h'_i\}$，其中 K' 为允许修改文件夹所对应的哈希编码的指标集，K'' 为不允许修改的文件夹所对应的哈希编码的指标集，满足 $K' \bigcup K'' = K, K' \bigcap K'' = \phi$. 按照如下公式（3）计算，完成实现授权：

$$\begin{aligned}
&h''_i = Base\ 64(Hash\ (ID_{rev} \parallel h_i \parallel d_{des})), i \in K' \\
&S = \prod_{i \in K'} v_i^{h''_i} \\
&\sigma_1^{aut} = \sigma_1 \\
&\sigma_2^{aut} = \sigma_2 (v' \prod_{i \in K'} v_i^{h''_i})^{r'_{ID_{des}}}, i \in K' \\
&\sigma_3^{aut} = \sigma_3 (v' \prod_{i \in K''} v_i^{h_i})^{r'_{ID_{des}}}, i \in K''
\end{aligned} \tag{3}$$

其中 S 为秘密授权信息，σ_2^{aut} σ_2^{aut} 为指定身份为 ID_{rev} 的修改者对指标 $i \in K'$ 的文件夹进行允许修改授权，σ_3^{aut} σ'_3 限定了不可修改的文件夹.

（3）开发者将已进行授权签名的 APK 文件、对应的工程源码及授权信息 h'、$\sigma^{aut} = (\sigma_1^{aut}, \sigma_2^{aut}, \sigma_3^{aut})$ 发送给修改者.

3.4 二次开发过程

修改者在二次开发过程中首先验证授权合法性和有效性，然后依据授权信息进行相应修改，最后对修改后的 APK 文件进行签名. 当修改者收到已授权的 APK 文件和对应的工程源码后，从 APK 文件“/META-INF”目录下的 APK_AUTH.Sig 文件中取出秘密授权信息和签名信息 (P, σ)，进行如下操作：

（1）验证签名 σ 和 σ^{aut} 的有效性. 若签名正确, 则进行第（2）步操作, 否则终止操作.

（2）结合授权信息 h', 对允许修改的文件及其相应源码进行修改, 得到新 APK 文件, 将新 APK 文件按图 1 的格式重新整合并分成 n 份,表示为 $\{h_1^{rev}, h_2^{rev}, \cdots, h_n^{rev}\}$.

$$h_i^{rev} = h_i, \quad (i \in K'')$$

$$h_i^{rev} = Base64(Hash(d_{rev} \| H_i^{rev} \| ID_{rev})), \quad (i \in K')$$

其中, $H_i^{rev} = Base64(Hash(d_{rev} \| file(i) \| ID_{rev}))$, $file(i)$为第 i 个修改的文件（夹）.

（3）在完成二次开发后, 修改者选取随机数 $r'_{rev}, \bar{r}_{rev} \in Z_q^*$,按如下公式（4）对修改后的文件进行签名计算：

$$\bar{\sigma}_1 = \sigma_1^{aut} * (u' \prod_{i \in K'} u_i^{ID_{rev}^i})^{\bar{r}_{ID_{rev}}} * (v' \prod_{i \in K'} v_i^{h_i^{rev}})^{r'_{ID_{rev}}}$$

$$\bar{\sigma}_2 = \sigma_2^{aut} * g^{r'_{ID_{rev}}}, \ \bar{\sigma}_3 = \sigma_3^{aut} * g^{\bar{r}_{ID_{rev}}} \qquad (4)$$

$$\bar{\sigma} = (\bar{\sigma}_1, \bar{\sigma}_2, \bar{\sigma}_3)$$

（4）在相应的“/META-INF”目录下生成签名文件 APK_SAN.SIG,将签名信息 $(\sigma, \bar{\sigma})$ 保存在该文件中, 并重新打包生成新的 APK 文件.

3.5 合法性验证

检验签名信息 $(\sigma, \bar{\sigma})$ 以及 σ^{aut} 是否合法,此处仅给出验证 σ_1 和 σ_2^{aut} 的有效性,验证等式

$e(\sigma_1, g) = e(g_1, g_2) e(\sigma_2, u' \prod_{i=1}^{n} u_i^{ID_{des}^i}) e(\sigma_3, v' \prod_{i=1}^{n} v_i^{h_i})$ 、 $e(\sigma_2^{aut}, g) = e(\sigma_2, g) e(\sigma_3, v' \prod_{i \in K''} v_i^{h_i''})$

等是否成立。具体如下：

$$\begin{aligned} e(\sigma_1, g) &= e(g_2^{\alpha} * (u' \prod_{i=1}^{n} u_i^{ID_{des}^i})^{\bar{r}_{ID_{des}}} * (v' \prod_{i=1}^{n} v_i^{h_i})^{r'_{ID_{des}}}, g) \\ &= e(g^{\alpha}, g_2) e(g, (u' \prod_{i=1}^{n} u_i^{ID_{des}^i})^{\bar{r}_{ID_{des}}}) e(g, (v' \prod_{i=1}^{n} v_i^{h_i})^{r'_{ID_{des}}}) \\ &= e(g^{\alpha}, g_2) e(g^{\bar{r}_{ID_{des}}}, u' \prod_{i=1}^{n} u_i^{ID_{des}^i}) e(g^{r'_{ID_{des}}}, v' \prod_{i=1}^{n} v_i^{h_i}) \\ &= e(g_1, g_2) e(\sigma_2, u' \prod_{i=1}^{n} u_i^{ID_{des}^i}) e(\sigma_3, v' \prod_{i=1}^{n} v_i^{h_i}) \end{aligned}$$

$$\begin{aligned} e(\sigma_2^{aut}, g) &= e(\sigma_2 * (v' \prod_{i \in K''} v_i^{h_i''})^{r'_{ID_{des}}}, g) \\ &= e(\sigma_2 * (v' \prod_{i \in K''} v_i^{h_i''})^{r'_{ID_{des}}}, g) \\ &= e(\sigma_2, g) e(g^{r'_{ID_{des}}}, v' \prod_{i \in K''} v_i^{h_i''}) \\ &= e(\sigma_2, g) e(\sigma_3, v' \prod_{i \in K''} v_i^{h_i''}) \end{aligned}$$

$\bar{\sigma}$ 有效性的验证方法与 σ 的验证相同, σ_3^{aut} 的有效性验证与 σ_2^{aut} 相同. 限于篇幅, 本文不再详细给出.

可知, 若 Android APK 开发者和修改者只要遵循所提出的授权机制, 产生的签名均有效. 从而授权是合法的, 同时可净化者仅能修改了 APK 中指定位置的部分.

4 方案安全性和性能分析

4.1方案安全性分析

定义 **3.** 称攻击者 A 具有 $(t, q_e, q_s, \varepsilon)$ 攻击能力：在 APK-SAN 授权机制中, 若 A 进行 q_e 次私钥抽取请求和 q_s 次的签名请求, 在运行时间为 t 内能以不可忽略的优势 ε 成功伪造 APK 原签名或修改者的签名.

定义 **4.**若 APK-SAN 授权机制能抵御上述定义的攻击, 则称 APK-SAN 授权机制为 $(t, q_e, q_s, \varepsilon)$ 安全.

定理 1. 如果(ε',t')-CDH 假定成立,上述的 APK-SAN 授权机制是 $(t, q_e, q_s, \varepsilon)$ 安全, 其中双线性映射中所用群的阶满足 $q > 2(q_e + q_s)$.

证明. 假定存在一个攻击者 A 能成功攻击 APK-SAN 授权机制, 也就是攻击者 A 能成功伪造 APK-SAN 授权机制中的授权 σ. 即攻击者 A 进行 q_e 次私钥抽取请求和 q_s 次的签名请求, 在运行时间为 t 内能以 ε 的优势根据 APK-SAN 授权机制的步骤能成功伪造有效的可净化签名. 在这样的假定下, 给定群 G_1 和它的生成元 g, 则存在一个求解算法 B 能在运行时间 t 内以 ε 的优势解决 CDH 问题的一个实例, 即在收到 g^a、 g^b 后, 能用算法 B 求解 g^{ab}, 其中

$$t' = t + O(5q_e + (2n+4)q_s)t_{G_1},\quad \varepsilon' = \frac{\varepsilon}{16q_s(q_e+q_s)(n+1)^2}$$

其中 n 为 APK 文件比特串长度, t_{G_1} 为群 G_1 中指数运算所花的总时间.

在攻击过程中, 算法 B 充当挑战者的角色与攻击者 A 进行交互, 两者共享交互视图资源, 在攻击者 A 能攻击 APK-SAN 授权机制并成功伪造合法的授权（签名）时, 算法 B 也能成功求解攻击场景中涉及到的具体 CDH 问题的一个实例. 具体攻击步骤如下:

首先,算法 B 构造用于求解 CDH 问题实例的部分数据和系统参数. 具体设置如下:

整数 r_u 和 r_m, 其中要求 $0<r_u<q$, $0<r_m<q$;

整数 s_u 和 s_m, 其中要求 $0<s_u<n$, $0<s_m<n(r_u(n+1)<q, r_m(n+1)<q)$;

令 $r_u = 2(q_e + q_s)$, $r_m = 2q_s$.

整数 $x' \in \mathbb{Z}_{r_u}$, $y' \in \mathbb{Z}_{r_m}$, $z' \in \mathbb{Z}_q$, $w' \in \mathbb{Z}_q$;

n 维向量$(x_1, x_2, \ldots, x_n)$, 其中 $x_i \in \mathbb{Z}_{r_u}$;

n 维向量$(y_1, y_2, \ldots, y_n)$, 其中 $y_i \in \mathbb{Z}_{r_m}$;

n 维向量$(z_1, z_2, \ldots, z_n)$, 其中 $z_i \in \mathbb{Z}_q$;

n 维向量$(w_1, w_2, \ldots, w_n)$, 其中 $w_i \in \mathbb{Z}_q$;

同时, 对于网上商城的任意身份为 ID^* 的开发者, 将其身份表示为 $ID^* = \{ID_1^*, ID_2^*, \cdots, ID_n^*\}$, 并构造与身份为 ID^* 的开发者/修改者相关的 APK 文件, 其编码表示为 $h^* = \{h_1^*, h_2^*, \cdots, h_n^*\}$. 利用这些信息构造如下 4 个函数：

$$F(ID^*) = x' + \sum_{i=1}^{n} x_i ID_i^* - r_u s_u;\ J(ID^*) = z' + \sum_{i=1}^{n} z_i ID_i^*;$$

$$K(h^*) = y' + \sum_{i=1}^{n} y_i h_i^* - r_m s_m;\ L(h^*) = w' + \sum_{i=1}^{n} w_i h_i^*;$$

并构造以下系统参数：

$$g_1 = g^a,\ g_2 = g^b;$$

$$u' = g_2^{-r_u s_u + x'} g^{z'},\ u_i = g_2^{x_i} g^{z_i}, 1 \le i \le n;$$

$$u' = g_2^{-r_m s_m + y'} g^{w'},\ v_i = g_2^{y_i} g^{w_i}, 1 \le i \le n;$$

且有

$u' \prod_{i=1}^{n} u_i^{ID_i} = g_2^{F(ID)} g^{J(ID)}$, $v' \prod_{i=1}^{n} v_i^{h_i^*} = g_2^{K(h^*)} g^{L(h^*)}$。

然后将上面的信息发给攻击者 A.

其次, 在攻击 APK-SAN 授权机制过程中, 当攻击者 A 在任何时间内进行询问 AS 私钥抽取 oracle 和 APK 签名 oracle 时, 算法 B 给以如下响应：

(1) 私钥抽取请求. 当攻击者 A 向算法 B 提交关于身份为 ID^* 的开发者的私钥提取请求时, 算法 B 进行如下操作：

① 若 $F(ID^*)=0 \bmod(r_u)$, 则算法 B 终止并输出“失败”;

② 若 $F(ID^*)\neq 0 \bmod(r_u)$, 则算法 B 随机选取一个整数 $r_{ID^*}\in\mathbb{Z}_q^*$ 并按公式(5)计算私钥：

$$d_{ID^*}=(d_{ID^*}^{(1)},d_{ID^*}^{(2)})=(g_1^{-\frac{J(ID^*)}{F(ID^*)}}(g_2^{F(ID^*)}g^{J(ID)})^{r_{ID^*}},g_1^{-\frac{1}{F(ID^*)}}g^{r_{ID^*}}) \quad (5)$$

并将身份为 ID^* 的开发者的私钥 $(d_{ID^*}^{(1)},d_{ID^*}^{(2)})$ 发给攻击者.

由于

$$\begin{aligned}d_{ID^*}^{(1)}&=g_1^{-\frac{J(ID^*)}{F(ID^*)}}(g_2^{F(ID^*)}g^{J(ID^*)})^{r_{ID^*}}\\&=g_2^{a}(g_2^{F(ID^*)}g^{J(ID^*)})^{\frac{-a}{F(ID^*)}}(g_2^{F(ID^*)}g^{J(ID^*)})^{r_{ID^*}}\\&=g_2^{a}(g_2^{F(ID^*)}g^{J(ID^*)})^{r_{ID^*}-\frac{-a}{F(ID^*)}}\\&=g_2^{a}(u'\prod_{i=1}^{n}u_i^{ID_i^*})^{r_{ID^*}}\end{aligned}$$

$$d_{ID^*}^{(2)}=g_1^{-\frac{1}{F(ID^*)}}g^{r_{ID^*}}=g^{-\frac{a}{F(ID^*)}}g^{r_{ID^*}}=g^{r_{ID^*}-\frac{a}{F(ID^*)}}=g^{\vec{r}_{ID^*}}$$

其中 $\vec{r}_{ID^*}=r_{ID^*}-\dfrac{a}{F(ID^*)}$.

因此所构造的私钥 $(d_{ID^*}^{(1)},d_{ID^*}^{(2)})$ 有效.

(2) APK 文件授权（签名）请求. 当攻击者 A 向签名 oracles 提出关于身份为 ID^* 的开发者/修改者生成的 APK 文件签名时, 算法 B 操作如下：

① 若 $F(ID^*)\neq 0 \bmod(r_u)$,算法 B 调用前面给出的私钥提取请求来构造身份为 ID^* 的开发者/修改者的私钥, 再用私钥对 APK 文件进行签名即可;

② 若 $F(ID^*)=0 \bmod(r_u)$ 且 $K(h^*)\neq 0 \bmod(r_m)$, 则算法 B 选取 $r',r''\in\mathbb{Z}_q^*$,并按公式(6)计算得到签名：

$$\begin{aligned}\sigma&=(\sigma_1,\sigma_2,\sigma_3)\\&=\left(\left(u'\prod_{i=1}^{n}u_i^{ID^*}\right)^{r'}\cdot g_1^{-\frac{L(h^*)}{K(h^*)}}\cdot\left(v'\prod_{i=1}^{n}v_i^{h_i^*}\right)^{r''},g^{r'},g_1^{-\frac{1}{K(h^*)}}\cdot g^{r''}\right)\end{aligned} \quad (6)$$

③若 $F(ID^*)=0 \bmod(r_u)$ 且 $K(h^*)=0 \bmod(r_m)$, 则算法 B 终止并输出“失败”.

(3) APK 文件授权（签名）伪造. 倘若算法 B 在求解过程中未终止, 则攻击者 A 能成功输出一个身份为 ID' 的开发者/修改者生成的 APK 文件（其编码为 $h'=\{h_1',h_2',\cdots,h_n'\}$）的签名 $\sigma'=(\sigma_1',\sigma_2',\sigma_3')$, 从而能成功伪造有效的 APK 文件授权（签名）.

最后,算法 B 对 CDH 问题实例进行求解. 若 $F(ID')\neq 0 \bmod(q)$ 且 $K(h')\neq 0 \bmod(q)$, 算法 B 终止操作. 否则, 若 $F(ID')=0 \bmod(q)$ 且 $K(h')=0 \bmod(q)$, 算法 B 可求出 CDH 问题的实例的一个解为

$$g^{ab}=\frac{\sigma_1'}{(\sigma_2')^{J(ID')}(\sigma_3')^{J(h')}}.$$

为了确保算法 B 能以至少ε的概率成功解决 CDH 问题实例, 算法 B 需在同时满足如下三个事件的条件下进行：

①*Evt1*：当攻击者 A 进行私钥提取请求时算法 B 不会终止, 即要求 $F(ID^*)\neq 0 \bmod(r_u)$;

②*Evt2*：攻击者 A 能生成身份为 *ID'*的开发者/修改者生成的 APK 文件授权（签名）$\sigma'=(\sigma_1',\sigma_2',\sigma_3')$；即至少要求 $K(h^*)\neq 0 \bmod(r_m)$;

③*Evt3*：APK 文件授权（签名）伪造成功,同时要求 $F(ID')=0 \bmod(q)$ 且 $K(h')=0 \bmod(q)$, 其中 $1\le i\le n$;

为了便于计算, 定义 $U_i: F(ID^*)\neq 0 \bmod(r_u)$; $U^*: F(ID')=0 \bmod(q)$; $V_j: K(h^*)\neq 0 \bmod(r_m)$;

$V^*: K(h')=0 \bmod(q)$. 则算法 B 成功的概率等价于

$$\begin{aligned}\Pr[B_{succ}] &\ge \Pr[\bigcap_{i=1}^{q_e+q_s} U_i \cap U^* \cap \bigcap_{j=1}^{q_s} V_j \cap V^*]\\ &= \Pr[\bigcap_{i=1}^{q_e+q_s} U_i \cap U^*]\times \Pr[\bigcap_{j=1}^{q_s} V_j \cap V^*] \qquad (7)\\ &= \Pr[\bigcap_{i=1}^{q_e+q_s} U_i]\times \Pr[U^*]\times \Pr[\bigcap_{j=1}^{q_s} V_j]\times \Pr[V^*] \qquad (8)\end{aligned}$$

式(7)、(8)均由 U_i 、U^*、V_j 、V_j 相互独立而得到, 其中 $\Pr[U^*]$ 以及 $\Pr[\bigcap_{i=1}^{q_e+q_s} U_i]$ 分别计算如下：

$$\begin{aligned}\Pr[U^*] &= \Pr[(F(ID')=0 \bmod q)\wedge(F(ID')=0 \bmod r_u)]\\ &= \Pr[F(ID')=0 \bmod r_u]\times \Pr[(F(ID')=0 \bmod q)\,|\,(F(ID')=0 \bmod r_u)]\\ &= \frac{1}{r_u}\times\frac{1}{n+1}\end{aligned}$$

又因为

$$\Pr[\bigcap_{i=1}^{q_e+q_s} U_i]=1-\Pr[\bigcup_{i=1}^{q_e+q_s} \neg U_i]\ge 1-\frac{q_e+q_s}{n}$$

因此, 我们得到

$$\begin{aligned}\Pr[\bigcap_{i=1}^{q_e+q_s} U_i \wedge U^*] &\ge \left(\frac{1}{r_u}\times\frac{1}{n+1}\right)\times\left(1-\frac{q_e+q_s}{r_u}\right)\\ &\ge \frac{1}{4(q_e+q_s)(n+1)}\end{aligned}$$

同理可得

$$\Pr[\bigcap_{i=1}^{q_s} V_j \wedge V^*]\ge \frac{1}{4q_s(n+1)}$$

从而有

$$\Pr[B_{succ}] \geq \frac{1}{16q_s(q_e+q_s)(n+1)^2}.$$

得到ε的值为

$\varepsilon' = \dfrac{\varepsilon}{16q_s(q_e+q_s)(n+1)^2}$. 特别地, 由于 $n=5$, 因此有 $\varepsilon' = \dfrac{\varepsilon}{576(q_e+q_s)}$.

证毕.

在 APK 二次开发、修改过程中, 修改者只能对指定的修改位置或文件进行修改, 同时对二次开发或修改后 APK 文件进行可净化签名. 任何修改者均不能修改授权修改以外的位置或文件. 具体地, 若攻击者 A 能修改开发者授权修改以外的位置或文件, 即修改与 base64 编码满足 $K'' = \{i \in K \mid h_i \neq h_i'\}$ 对应的文件, 并产生有效的可净化签名, 则意味着我们的 APK-SAN 授权机制未能达到存在不可伪造签名安全. 因此得出如下结论：

定理 2. 假定存在有界多项式攻击者 A 能在时间 t 内经过至多 q'_e 次的密钥提取询问和 q'_s 次的签名询问后以优势 ε' 来攻破 APK-SAN 授权机制中 APK 文件授权修改位置或文件的不可改变性, 则存在一个算法 B 能在时间 t'' 内以优势ε'' 来产生一个有效的签名.

证明. APK 文件授权修改位置的不可改变性是指若攻击者 A 故意或非法修改与 base64 编码满足 $K'' = \{i \in K \mid h_i \neq h_i'\}$ 对应的文件, 则可以通过签名有效性验证发现非法修改行为, 并告知修改的 APK 文件无效. 若攻击者 A 能突破 APK 文件授权修改位置的不可改变性,意味着算法 B 可以产生一个有效的签名. 具体分析如下：

(1) 参数建立. 算法 B 与挑战者 C 进行如下交互：

① 算法B向挑战者C提交一个请求, 挑战者C给算法B返回系统的参数如下(G_1, G_2, e, q, g, g_1, g_2, u', v', u_1, … , u_n, v_1, … , v_n)；

② 对于 $i \in K'$,算法 B 选择 $t_i \in \mathbb{Z}_q^*$,并令 $u_i = g^{t_i}$,作为算法 B 的系统参数；

③ 对于 $i \in K'$,算法 B 选择 $s_i \in \mathbb{Z}_q^*$,并令 $v_i = g^{s_i}$,作为算法 B 的系统参数.

(2) 私钥抽取请求. 攻击者 A 向算法 B 提交关于身份为 ID'的修改者的私钥请求, 算法 B 如下响应：

① 算法 B 向挑战者 C 请求身份为 ID'的修改者的私钥, 并获得私钥 $(d_{ID_{i-1}}^{(1)}, d_{ID_{i-1}}^{(2)})$；

② 算法 B 令 $\bar{d}_{ID_i}^{(1)} = d_{ID_i}^{(1)} \cdot (d_{ID_i}^{(2)})^{\prod_{i \in K'} ID_i t_i}$ 且 $\bar{d}_{ID_i}^{(2)} = d_{ID_{i-1}}^{(2)}$, 并将 $(\bar{d}_{ID_i}^{(1)}, \bar{d}_{ID_i}^{(2)})$ 发送给攻击者 A.

(3) 授权(签名)请求. 当攻击者 A 向算法 B 提交关于身份 ID^*的开发者对编码为 $h' = \{h_1', h_2', \cdots, h_n'\}$ 的 APK 文件进行二次开发（修改）时, 算法 B 进行如下操作：

① 算法 B 计算：

$$h_i^* = h_i', \quad (i \in K'')$$
$$h_i^* = Base64(Hash(d^* \parallel H_i^* \parallel ID^*)), \quad (i \in K')$$

并表示为 $h^* = \{h_1^*, h_2^*, \cdots, h_n^*\}$；

② 算法 B 向挑战者 C 询问属于身份 ID^*的部分编码为 $h_i^* (i \in K'')$ 的 APK 的二次开发, 并得到 (σ_1,σ_2,σ_3)；

③ 算法 B 令 $\sigma_1^* = \sigma_1 \cdot \prod_{i \in K''} \sigma_2^{ID_i^* t_i} \cdot \prod_{i \in K''} \sigma_3^{h_i^* s_i}$，$\sigma_2^* = \sigma_2$，$\sigma_3^* = \sigma_3$；

④ 算法 B 将 $(\sigma_1^*, \sigma_2^*, \sigma_3^*) \bigcup (\sigma_3^{h_i^* s_i})$ 发给攻击者 A，其中 $i \in K''$.

(4) 修改伪造. 假定攻击者 A 能输出一个由身份 $I\vec{D}$ 的开发者及其编码为 $\hat{h} = \{\hat{h}_1, \hat{h}_2, \cdots, \hat{h}_n\}$ 的 APK 文件进行有效修改，即产生的新 APK 文件且其签名 $(\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3)$ 合法，则算法 B 在攻击者 A 伪造的签名的基础上也能获得一个有效 APK 文件签名，具体如下：

① 攻击者 A 将编码为 $\hat{h} = \{\hat{h}_1, \hat{h}_2, \cdots, \hat{h}_n\}$ 的 APK 文件修改后的签名 $(\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3)$ 发给算法 B；

② B 构造由身份为 $I\vec{D}$ 的开发者开发的编码为 $\vec{h} = \{\vec{h}_1, \vec{h}_2, \cdots, \vec{h}_n\}$ 的 APK 文件．即设置 $\{I\vec{D}_i\}$ 和 $\{\vec{h}_i\}$，其中 $\vec{h}_i = \hat{h}_i, i \in K''$．则存在某个 $h_i, i \in K'$，算法 B 可如下构造：

$$\vec{\sigma}_1 = \frac{\hat{\sigma}_1}{\prod_{i \in K''} (\hat{\sigma}_2)^{I\hat{D}_i \hat{t}_i} \cdot \prod_{i \in K''} (\hat{\sigma}_3)^{\hat{h}_i \hat{s}_i}}, \vec{\sigma}_2 = \hat{\sigma}_2, \vec{\sigma}_3 = \hat{\sigma}_3.$$

很容易验证，$\vec{\sigma} = (\vec{\sigma}_1, \vec{\sigma}_2, \vec{\sigma}_3)$ 是身份为 $I\vec{D}$ 的 AS_i^* 对编码为 $\vec{h}_i = \hat{h}_i$，$i \in K''$ 的 APK 文件的签名．从定义 1 可知，算法 B 能求解 CDH 问题的一个实例．

显然，算法 B 成功伪造 APK 文件签名的前提条件是攻击者 A 能改变了 APK-SAN 授权机制的 APK 文件修改授权的不可改变性，因此 B 成功的概率 $\varepsilon'' \geq \varepsilon$.

同时，算法 B 运行的时间 t'' 等于攻击者 A 运行的时间 t 再加上它响应 q'_e 次的密钥提取询问和 q'_s 次的签名询问所花的时间．而每次密钥抽取询问需要算法 B 在群 G_1 中进行 $|K''|$ 次的指数运算；每次签名询问需要算法 B 在群 G_1 中进行 $2|K''|$ 次的指数运算．假定群 G_1 中进行指数运算所花的时间为 t_e，则有 $t'' = t + (q'_e + 2 \cdot q'_s) \cdot |K''| \cdot t_e$ ．

证毕.

4.2 性能分析

考虑这样的情景：当开发者 Designer 完成某个 APK 开发并发布后，另一位开发者（应用者）Applicant 申请对此 APK 进行修改，开发者 Designer 授权 Applicant 对此 APK 指定的位置进行改进或为其增加某些功能．根据 Android 应用程序签名机制要求，开发者 Applicant 会进行如下两种可能情况的操作：1）Applicant 完成 APK 修改后，发给 Designer,由 Designer 重新签名并发布；2）Applicant 直接对修改后的 APK 进行签名并发布．本文试图提供一种可以由 Applicant 直接签名并能够验证 Designer 的版权的方案．假定情况 1）为目前 Android 所采用基于 RSA 算法的签名机制的情景,称为方案 1；情况 2）为方案 2[22]以及本文所提方案采用的情景，三种方案的开销及其提供的安全服务比较如表 1 所示．其中，表中 *dec*．表示一次 RSA 公钥算法解密开销，*enc.*表示一次 RSA 加密开销，n 为该 APK 所包含的文件个数，P 代表本文方案中的大素数，t 表示 Applicant 返回给 Designer 传输开销，m 表示 APK 中要修改的文件的个数，显然 $m \leq n$，e 表示指数运算的运算量，E 表示 pair 对运算的运算量；提供细粒度授权表示方案能否提供到只能修改 APK 指定的某位置某部分内容的授权，认证是指能够验证 Designer 的版权以及双方的签名的有效性，需要证书表示方案中是否采用证书机制．

表 1 计算量和性能比较

Table 1 the comparing of the cost and security services

方案	签名开销	验证开销	提供细粒度授权	认证	需要证书
方案 1	*ndec.* + *t*	*nenc.*	否	是	是
方案 2[22]	(*n*+1) *dec.*	(*n*+1) *enc.*	否	是	是
APK-SAN 方案	(2*m*)e	(2*m*)(e+2*E*)	是	是	否

在运算开销方面. 表 1 是三种方案的运算开销对比, 根据 Android APK 签名规则, 对 APK 中的 *n* 个文件的签名开销等同于执行 *n* 次 RSA 加密运算, 即 *n dec.*, 验证 APK 中 *n* 个文件的签名开销等价于执行 *n* 次 RSA 解密运算, 即 *nenc.*, 方案 1 的签名开销还包括 Applicant 和 Designer 之间的传输开销 *t*, 因此方案 1 的签名开销为 *ndec.* + *t*, 验证开销为 *nenc.*. 方案 2 不需要在 Applicant 和 Designer 之间传输 APK, 因此无传输开销, 但生成和验证授权签名时会分别增加 1 次 RSA 解密运算和 1 次 RSA 加密运算. 根据前文对 APK-SAN 方案的描述可知, 对 APK 中 *m* 个修改文件签名需要 2*m* 次指数运算, 即 2*me*, 验证签名时还需增加 4 次 pair 对数运算, 因此签名验证开销为 2*me*+4*me*E.

方案仿真中,我们基于 3.60 GHz Intel Xeon E5 4 核处理器和 64 位 windows 7 操作系统, 并配置 Android 开发环境和反编译工具 APKToolKit V3.0, 借助 JPBC(Java Pairing-Based Cryptography Library)②以及 Java API 库完成仿真实验. 仿真结果可知, 在方案 1 和方案 2 中, 1024 比特的 RSA 的签名所花时间为 50ms, 签名验证所花的时间为 2.5 ms；而有限域 F_{97} 上的配对运算所花时间为 43 ms. 显然, 方案 2 和 APK-SAN 方案所耗时间开销比方案 1 少; 当 $m < n$ 时, APK-SAN 方案时间开销与方案 2 的时间开销大致相等; $m = n$ 时, APK-SAN 方案时间开销多于方案 2 的时间开销.

在安全服务方面. 方案 1 和方案 2 虽然能够实现验证 Designer 的版权并验证 Applicant 的版权和签名的合法性, 能实现认证功能, 但是需要证书, X.509 证书的大小约为 600byte[24], 2016 年我国网民使用的 android 手机为 19.3 亿部③, 若均以证书方式管理,证书数量约为 19 亿个,需要存储的证书大小为 1.14TB,则将会增加很大的存储开销和管理开销; 同时未能提供细粒度授权, 从而不能有效实现 Applicant 对 APK 文件修改权限的控制, Applicant 可以修改 APK 文件中的任意内容. 若 Applicant 恶意非法篡改 Designer 的 APK 文件, 添加恶意代码文件, 会引起恶意 APK 文件产生的可能. APK-SAN 方案中 Designer 在给 Applicant 授权时, 能限定 Applicant 修改的位置和子文件, 若 Applicant 违反授权规定, 修改了 APK 文件的其它非授权位置或子文件, 修改后的 APK 文件生成的签名无效, 因此能提供细粒度授权机制, 亦能限制 Applicant 的修改权限, 从而防止在 APK 文件中嵌入恶意代码等行为, 一定程度上保护了 Designer 的权益. 同时 APK-SAN 方案的工作无需用户证书, 可以避免用户证书的存储开销.

5 总结

本文利用可净化签名方案的特有属性设计一个新的标准模型下可证明安全的 APK 授权机制

② JPBC-Java Pairing-Based Cryptography Library.http://gas.dia.unisa.it/projects/jpbc/download.html#.W_O7JPnwrwM

③ 《中国移动互联网发展状况及其安全报告（2017）》, http://news.xinhuanet.com/info/ttgg/2017-05/17/c_136291536.htm

APK-SAN．该机制允许 Android 开发者授权给修改者对 APK 文件源代码的指定部分进行修改, 并对修改完成后编译打包的新 APK 文件签名, 且签名是有效的. APK-SAN 签名机制旨在保证 APK 文件的完整性、真实性和不可否认性的同时, 减少开发者的通信时间和计算开销, 保障开发者和修改者双方的合法权益．因此, 我们的授权机制可以解决 APK 授权后验证 APK 原设计者签名有效性困难的问题, 也能够保护 APK 开发者的版权权益．今后的研究工作将进一步研究如何保护 PKG 的安全及合法性、探讨身份管理机制以及允许修改位置的恶意代码检测机制,以便更好地确保 APK-SAN 授权机制的有效性。

致 谢 审稿人提出的修改意见对提高论文水平有很大的帮助,在此表示感谢!

References:

[1] Qing Sihan. Research progress on android security. Journal of Software. 2016.27(1):45-71.

[2] Enck W, Octeau D, Mcdaniel P, et al. A study of Android application security. Proc Usenix Security Symposium, 2011.

[3] Cheng Yao, Ying Lingyun, Jiao Sibei, Su Purui, Feng Dengguo. Research on user privacy leakage in mobile social messaging applications. Chinese Journal of Computers. 2014.37(1):87-100.

[4] Jae-Kyung Park,Sang-Yong Choi.Studying Security Weaknesses of Android System.International Journal of Security and Its Applications.Vol 9,No.3, 2015.pp:7-12.

[5] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein.Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Raleigh, NC, USA, October 2012,pages 3--14.

[6] J. Kim, Y. Yoon, K. Yi, J. Shin, and S. Center. ScanDal: Static analyzer for detecting privacy leaks in Android applications. in MoST 2012: Mobile Security Technologies 2012, H. Chen, L. Koved, and D. S. Wallach, Eds. Los Alamitos, CA, USA: IEEE, May 2012. [Online]. Available: http://ropas.snu.ac.kr/scandal/.

[7] Md Yasser Karim,Huzefa Kagdi,MassimiJiano Di Penta.Mining Android Apps to Recommend Permissions.2016 IEEE23rd International Conference on Software Analysis,Evolution,and Reengineering.2016,page:427-437.

[8] Mohsen Toorani ,Ali A. Beheshti .A lightweight public key infrastructure for the mobile environment. Communication Systems,19-21 Nov. 2008:162-166.

[9] Pablo Silva,Viceme J.P. Amcrim,Filipe N.Ribeiro,Igcr Muzetti.PrivacyMod:Controlling and Monitoring Abuse of Privacy-Related Data by Android Applications.2015 Brazilian Symposium on Computing Systems Engineering.2015:42-47.

[10] Paul Pearce,Gabriel Nunez,David Wagner. Addroid: Privilege separation for applications and advertisers in Android. In: Proc. of the 7th ACM Asia Conf. on Computer and Communications Security (ASIACCS 2012). 2012, pp:71-72.

[11] Junho Choi,Woon Sung,Chang Choi,Pankoo Kim.Personal information leakage detection method using the inference-based access control model on the Android platform.Pervasive and Mobile Computing. 2015 (24):138-149.

[12] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to Android. In Proceedings of the 17th ACM conference on Computer and communications security. 2010, pp. 73-84.

[13] T. Blasing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak. An Android application sandbox system for suspicious software detection. In Malicious and unwanted software (MALWARE), the 5th international conference on. IEEE, 2010, pp. 55-62.

[14] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission re-delegation: Attacks and defenses. Proceedings of the 20th USENIX Security Symposium, San Francisco, CA. 2011:1-16.

[15] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013, pages 73-84.

[16] S. Shuai, D. Guowei, G. Tao, Y. Tianchang, and S. Chenjie. Modelling analysis and auto-detection of cryptographic misuse in android applications. In Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on, 2014, pages 75-80.

[17] Siqi Ma, David Lo,Teng Li, Robert H.Deng. CDRep:Automatic Repair of Cryptographic Misuses in Android Applications. 11th ACM Asia Conference on Computer and Communications Security (AsiaCCS '16),2016:711-722.

[18] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou. A whitebox approach for automated security testing of Android applications on the cloud. In Automation of Software Test (AST), 2012 7th International Workshop on. IEEE, 2012, pp. 22-28.

[19] Antonios Dimitriadis,Pavlos S. Efraimidis,Vasilios Katos.Malevolent App Pairs: An Android Permission Overpassing Scheme.In: ACM International Conference on Computing Frontiers 2016,16-18 May 2016,Como,Italy,pp:431-436.

[20] Ateniese G,Chou D H ,DeMedeiros B,et al． Sanitizable Signature〃ESORICS 2005.Milan: [s.n.],2005: 159-177.

[21] Brzuska C,Fischlin M,Freudenreich T,et al． Security of sanitizable signatures revisited〃PKC 2009. California:[s.n],2009: 317-336.

[22] Zhao Bowen, Zhang Xiaoping, Li Daofeng, Su Jiebo, He Peicong. An APK authorization scheme based on transitive signature mechanism. Journal of Cryptologic Research. 2016. 3(1): 22–32.

[23] Waters B.Efficient identity-based encryption without random oracles.//Advances in Cryptology-Proceedings of the EUROCRYPT 2005.Aarhus,Denmark,2005:114-127.

[24] Granger R,Page D,Stam M.On small characteristic algebraic Tori in Paring-based cryptography. LMS Journal of Computation and Mathematics,2006,9(13):64-85.

附中文参考文献:

[1] 卿斯汉. Android 安全研究进展.软件学报, 2016, 27(1):45−71.

[3] 程瑶,应凌云,焦四辈,苏璞睿,冯登国.移动社交应用的用户隐私泄漏问题研究. 计算机学报,2014,37(1):87-100.

[22] 赵搏文,张小萍,李道丰,苏杰波,何佩聪.基于可传递签名机制的 APK 授权方案研究.密码学报, 2016,3(1):22-32.