**湖 南 科 技 大 学**

**毕 业 设 计（ 论 文 ）**

|  |  |
| --- | --- |
| **题目** |  |
| **作者** |  |
| **学院** |  |
| **专业** |  |
| **学号** |  |
| **指导教师** |  |

二〇〇 年 月 日

**摘 要**

本文主要论述了一种基于Android平台的文件加密系统及App加固程序的设计与实现。

该系统采用Java技术，以Android Studio为开发工具，AES为加密算法。设计主要实现了对Android平台各种类型文件的加密与解密，使其以密文形式存储在文件系统中，防止隐私文件被窃取。并且设计实现对Android安装文件Apk文件的加壳，防止App源代码被恶意开发者利用，进而破坏文件加密系统的安全性。

本论文主要按照软件工程的基本原理，阐述了对系统的可行性分析和需求分析，叙述了系统的总体设计和详细设计，详细论述了系统的实现过程，描述了系统进行的各项测试并顺利通过。

关键字：Android; 文件加密; App加固

**ABSTRACT**

This paper discusses the design and implementation of a platform based on the Android system and file encryption App reinforcement program.

The system uses Java technology to Android Studio as a development tool, AES encryption algorithm. Designed primarily to achieve the types of Android platform file encryption and decryption, it is stored in cipher text in the file system, the file is to prevent privacy theft. Design and realization of the installation files for Android Apk file packers, the source code to prevent malicious App developers to take advantage, and thus undermine the security of file encryption system.

In this thesis, according to the basic principles of software engineering, it describes the system requirements analysis and feasibility analysis, describes the overall design and detailed design of the system, discussed in detail the system implementation process, describes the test system and the smooth by.

**Keywords:** Android; file encryption; App reinforcement

**目录**

[第一章 绪论 - 2 -](#_Toc451066833)

[1.1 选题背景 - 2 -](#_Toc451066834)

[1.1.1 手机市场分析 - 2 -](#_Toc451066835)

[1.1.2 Android用户数据安全 - 2 -](#_Toc451066836)

[1.1.3 Android应用盗版情况 - 3 -](#_Toc451066837)

[1.2 选题意义 - 3 -](#_Toc451066838)

[1.3 论文组织结构 - 3 -](#_Toc451066839)

[第二章 相关基础知识 - 5 -](#_Toc451066840)

[2.1 开发环境 - 5 -](#_Toc451066841)

[2.2 AES原理 - 5 -](#_Toc451066842)

[2.3 Apk加壳技术 - 7 -](#_Toc451066843)

[2.3.1 dex、odex文件结构 - 8 -](#_Toc451066844)

[2.3.2 Android动态加载技术 - 10 -](#_Toc451066845)

[第三章 文件加密及APP加固需求分析 - 13 -](#_Toc451066846)

[3.1 文件加密 - 13 -](#_Toc451066847)

[3.2 App加固 - 13 -](#_Toc451066848)

[第四章 文件加密及APP加固系统设计 - 14 -](#_Toc451066849)

[4.1文件加密 - 14 -](#_Toc451066850)

[4.1.1 加密算法的选择 - 14 -](#_Toc451066851)

[4.1.2 密钥管理模块 - 15 -](#_Toc451066852)

[4.1.3 文件管理模块 - 15 -](#_Toc451066853)

[4.2 App加固 - 16 -](#_Toc451066854)

[第五章 文件加密及APP加固详细设计 - 17 -](#_Toc451066855)

[5.1 文件加密系统详细设计 - 17 -](#_Toc451066856)

[5.1.1 界面设计 - 17 -](#_Toc451066857)

[5.1.2 密钥管理模块 - 17 -](#_Toc451066858)

[5.1.3 文件管理模块 - 18 -](#_Toc451066859)

[5.2 App加壳详细设计 - 18 -](#_Toc451066860)

[5.2.1 加壳程序设计 - 18 -](#_Toc451066861)

[5.2.2 脱壳程序设计 - 19 -](#_Toc451066862)

[第六章 测试 - 20 -](#_Toc451066863)

[6.1 文件加密测试 - 20 -](#_Toc451066864)

[6.2 App加壳测试 - 20 -](#_Toc451066865)

[第七章 总结与展望 - 22 -](#_Toc451066866)

[第八章 致谢 - 23 -](#_Toc451066867)

[第九章 参考文献 - 24 -](#_Toc451066868)

# 绪论

## 选题背景

### 手机市场分析

Android自2012年2.2Froyo系统发布以来，一直占据着中国智能手机操作系统市场的大半壁江山，始终是唯一能与其相提并论的只有Apple公司的IOS系统，而根据国外市场分析公司Kantar的数据显示，2015年第一季度和2016年第一季度 Android系统市场占有率稳步提升3.4%，达到了76.4%，如图1.1所示。

从未来发展方向上看，Android系统由于其开放性，有很广阔的发展空间，比如车载系统、智能家居系统等等。

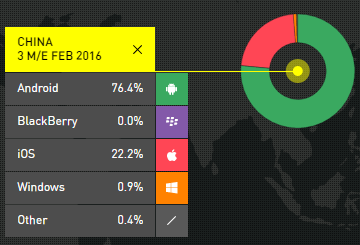
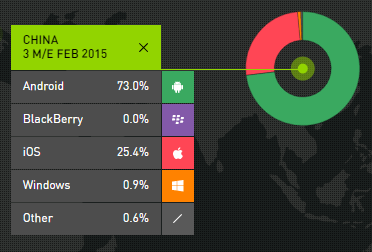


图1.1 2015及2016智能手机系统市场占有率

### Android用户数据安全

Android的安全问题一直饱受诟病，随着Android设备应用越来越广泛，安全问题也更应该引起重视。安全设计其实是贯穿了Android系统架构的各个层面的，覆盖系统内核、虚拟机、应用程序框架层以及应用层各个环节，力求在开放的同时，也恰当保护用户的数据、应用程序和设备的安全。Android安全模型主要提供以下几种安全机制[1]：

1. 进程沙箱隔离机制
2. 应用程序签名机制
3. 权限声明机制
4. 访问控制机制
5. 进程通信机制
6. 内存管理机制

而对于用户数据的的安全性，Android主要是由访问控制机制来保护的，这很明显是远远不够的，针对Meizu MX3进行测试，Meizu MX3自带的文件浏览器中包含有锁定区，访问需要密码，但其具体位置在/data/media/0/.@meizu\_protbox目录下，利用具有root权限的文件浏览器直接访问这个目录，就可以绕过密码查看到锁定区中的所有文件，可以看到，这里只有访问控制，而文件本身是明文存储在文件系统中的，由于root权限很可能由恶意软件通过系统漏洞取得，所以这种做法很不安全。

### Android应用盗版情况

盗版软件一直是影响移动互联网健康发展的毒瘤，不但对用户隐私、资费等造成威胁，更严重影响了正版APP的品牌形象和推广效果，侵害了正版APP开发者的权益。而由于国内Android应用市场众多，各应用市场的审核机制和管理规范良莠不齐，致使很多用户无意中下载了被植入恶意程序的盗版应用。用户一旦安装这类应用，轻则会被垃圾信息和广告骚扰，重则会导致隐私信息泄露、恶意扣费、流量损失等危害。

据360公司2015年Android手机应用盗版情况调研报告显示，2015年Android手机应用盗版情况非常猖獗，平均每款正版APP对应92.7个盗版。其中，在软件类盗版APP中，系统工具类APP的平均盗版数量最多，平均每款正版APP对应324.2个盗版[2]。

游戏类盗版APP中，模拟辅助类APP的平均盗版数量最多，平均每款正版游戏对应盗版228.6个。

软件类盗版APP中，便捷生活类盗版APP感染的设备数量最多，为237,878个；系统安全类盗版APP感染的设备数量为138,037；影音视听类盗版APP感染的设备数量为116,856个。

游戏类盗版APP中，经营策略类盗版APP感染的设备数量最多，为146,103个；休闲益智类盗版APP感染的设备数量为137,003；跑酷竞速类盗版APP感染的设备数量为83,650个。

## 选题意义

Android文件加密系统将在Android自有的访问控制基础之上，将需要加密存储的文件通过加密算法生成密文存储在硬盘中，进一步加强用户数据的安全性。并且通过APP加固手段保护源码，提高破解本程序的技术门槛，并且可以沿用到其他APP中，保护正版APP不被反编译。

## 论文组织结构

全文由七部分组成：

第一章阐述当前在Android设备占领绝大部分市场的情况下，系统对用户数据安全保障的情况以及Android应用加固保护源码的必要性。

第二章介绍了Android开发、APP运行的一些基础知识以及开发Android APP加固程序的一些关键技术。

第三章进行了对开发Android文件加密系统及APP加固程序的可行性分析，并且对该程序进行详尽的需求分析。

第四章根据第三章的需求分析，对系统从功能和架构上进行设计，为第五章的详细设计指明道路。

第五章对系统功能的实现从各个模块的代码编写角度进行详细的介绍。

第六章对系统进行全方位的测试，保证系统的可靠性及健壮性。

第七章总结了系统开发过程中遇到的问题，并且对系统的下一步发展方向进行了展望。

# 相关基础知识

本章涉及到系统中主要用到的一些关键技术的介绍，包括文件加密过程中加密算法的原理。

## 2.1 开发环境

IDE: Android Studio。本系统使用Google公司提供的官方IDE Android Studio进行开发，Android Studio是打造高品质，高性能Android应用程序的最快方法。作为谷歌官方的IDE，Android Studio包含建立一个APP的所有工具，包括代码编辑器，代码分析工具，内存分析工具，模拟器等等。

打包工具：Gradle。Gradle是一个基于Apache Ant和Apache Maven概念的项目自动化建构工具。它使用一种基于Groovy的特定领域语言来声明项目设置，而不是传统的XML。当前其支持的语言限于Java、Groovy和Scala。

开发语言：Java。面向对象的Java语言一直有着不可比拟的优势，跟动态语言比起来，Java的执行效率以及安全性都无可比拟，并且结构规范，方便调试。而跟其他的静态语言相比，Java丰富的类库为它带来了跟多的优势。

版本控制工具：git。作为一款版本控制工具，与CVS、Subversion一类的集中式版本控制工具不同，它采用了分布式版本库的作法，不需要服务器端软件，就可以运作版本控制，使得源代码的发布和交流极其方便。git的速度很快，这对于诸如Linux内核这样的大项目来说自然很重要。git最为出色的是它的合并追踪（merge tracing）能力。

测试设备：运行在Windows10系统上的Nexus 6 虚拟机，API level 22，以及Meizu MX3真机。

## 2.2 AES原理

AES加密过程是在一个4×4的[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82" \o "字节)矩阵上运作，这个矩阵又称为“体（state）”，其初值就是一个明文区块（矩阵中一个元素大小就是明文区块中的一个Byte）。（Rijndael加密法因支持更大的区块，其矩阵行数可视情况增加）加密时，各轮AES加密循环（除最后一轮外）均包含4个步骤[3]：

1. AddRoundKey—矩阵中的每一个字节都与该次[回合密钥](https://zh.wikipedia.org/w/index.php?title=%E5%9B%9E%E5%90%88%E9%87%91%E9%91%B0&action=edit&redlink=1" \o "回合密钥（页面不存在）)（round key）做[XOR运算](https://zh.wikipedia.org/wiki/XOR" \o "XOR)；每个子密钥由密钥生成方案产生。
2. SubBytes—通过一个非线性的替换函数，用[查找表](https://zh.wikipedia.org/wiki/%E6%9F%A5%E6%89%BE%E8%A1%A8" \o "查找表)的方式把每个字节替换成对应的字节。
3. ShiftRows—将矩阵中的每个横列进行循环式移位。
4. MixColumns—为了充分混合矩阵中各个直行的操作。这个步骤使用线性转换来混合每内联的四个字节。最后一个加密循环中省略MixColumns步骤，而以另一个AddRoundKey替换。

表2.2 AES密钥长度与轮数的关系

|  |  |  |  |
| --- | --- | --- | --- |
| 密钥长度 | 128 | 192 | 256 |
| 圈数 | 10 | 12 | 14 |

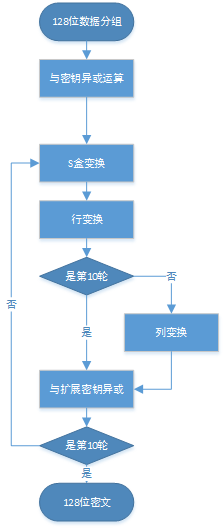


图 2.1 AES算法流程图

各个步骤的实现方式：

1. S盒变换（SubBytes）

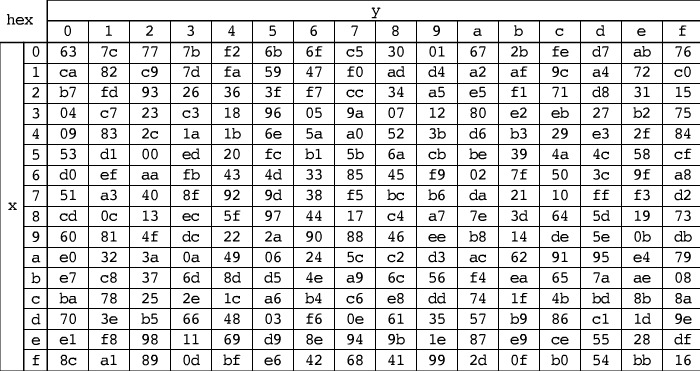


图2.2 AES中的S盒

矩阵中的各字节通过一个8位的[S-box](https://zh.wikipedia.org/wiki/S-box" \o "S-box)进行转换。这个步骤提供了[加密法](https://zh.wikipedia.org/w/index.php?title=%E5%8A%A0%E5%AF%86%E6%B3%95&action=edit&redlink=1)非线性的变换能力。[S-box](https://zh.wikipedia.org/wiki/S-box)与[GF](https://zh.wikipedia.org/wiki/%E6%9C%89%E9%99%90%E5%9F%9F" \o "有限域)（28）上的乘法[反元素](https://zh.wikipedia.org/wiki/%E5%8F%8D%E5%85%83%E7%B4%A0" \o "反元素)有关，已知具有良好的[非线性](https://zh.wikipedia.org/wiki/%E9%9D%9E%E7%B7%9A%E6%80%A7)特性。为了避免简单代数性质的攻击，S-box结合了乘法反元素及一个可逆的[仿射变换](https://zh.wikipedia.org/wiki/%E4%BB%BF%E5%B0%84%E5%8F%98%E6%8D%A2" \o "仿射变换)矩阵建构而成。此外在建构S-box时，刻意避开了[固定点](https://zh.wikipedia.org/w/index.php?title=%E5%9B%BA%E5%AE%9A%E9%BB%9E&action=edit&redlink=1)与[反固定点](https://zh.wikipedia.org/w/index.php?title=%E5%8F%8D%E5%9B%BA%E5%AE%9A%E9%BB%9E&action=edit&redlink=1)，即以S-box替换字节的结果会相当于错排的结果。AES算法中的S盒如图2.2所示

例如一个字节为0x19，经过S盒变换查找n(1,9) = 0xd4,所以就替换为0xd4。

1. 行变换 （ShiftRows）

ShiftRows描述矩阵的行操作。在此步骤中，每一行都向左循环位移某[偏移量](https://zh.wikipedia.org/w/index.php?title=%E5%81%8F%E7%A7%BB%E9%87%8F&action=edit&redlink=1)。在AES中（区块大小128位），第一行维持不变，第二行里的每个字节都向左循环移动一格。同理，第三行及第四行向左循环位移的偏移量就分别是2和3。经过ShiftRows之后，矩阵中每一竖列，都是由输入矩阵中的每个不同列中的元素组成。

1. 列变换 （MixColumns）

在MixColumns步骤，每一列的四个字节通过[线性变换](https://zh.wikipedia.org/wiki/%E7%BA%BF%E6%80%A7%E5%8F%98%E6%8D%A2" \o "线性变换)互相结合。每一列的四个元素分别当作1 , x , x^2 , x^3的系数，合并即为GF（28）中的一个多项式，接着将此多项式和一个固定的多项式c (x) = 3x^3 + x^2 + x + 2在modulo x^4+1下相乘。此步骤亦可视为[Rijndael有限域](https://zh.wikipedia.org/w/index.php?title=Rijndael%E6%9C%89%E9%99%90%E5%9F%9F&action=edit&redlink=1)之下的矩阵乘法。MixColumns函数接受4个字节的输入，输出4个字节，每一个输入的字节都会对输出的四个字节造成影响。因此ShiftRows和MixColumns两步骤为这个密码系统提供了[扩散性](https://zh.wikipedia.org/w/index.php?title=%E6%93%B4%E6%95%A3%E6%80%A7&action=edit&redlink=1)。

1. 扩展密钥（AddRoundKey）

AES算法利用外部输入密钥K(密钥串的字数为Nk),通过密钥的扩展程序得到共计4(Nr+1)字的扩展密钥。它涉及如下三个模块:

(1)位置变换(rotword)——把一个4字节的序列[A,B,C,D]变化成[B,C,D,A]；

(2)S盒变换(subword)——对一个4字节进行S盒代替；

(3)变换Rcon[i]——Rcon[i]表示32位比特字[xi-1,00,00,00]。这里的x是（02），如 Rcon[1]=[01000000]；Rcon[2]=[02000000]；Rcon[3]=[04000000]……

扩展密钥的生成：扩展密钥的前Nk个字就是外部密钥K；以后的字W[[ｉ]]等于它前一个字W[[i-1]]与前第Nk个字W[[i-Nk]]的“异或”,即W[[ｉ]]=W[[i-1]]W[[ｉ- Nk]]。但是若ｉ为Nk的倍数,则W[ｉ]=W[i-Nk]Subword(Rotword(W[[ｉ-1]]))Rcon[i/Nk]。

## 2.3 Apk加壳技术

Dalvik虚拟机是Android设备平台的核心组成部分之一，可以说就是Google，将Java虚拟机为了d适合移动设备内存小、CPU速度慢而进行的优化。其指令集基于寄存器架构，执行其特有的文件格式——dex字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能。它的核心内容是实现库（libdvm.so），大体由C语言实现。依赖于Linux内核的一部分功能——线程机制、内存管理机制，能高效使用内存，并在低速CPU上表现出的高性能。每一个Android应用在底层都会对应一个独立的Dalvik虚拟机实例，其代码在虚拟机的解释下得以执行。

要实现APP加固程序，必须要对Dalvik虚拟机有一定的了解，包括其可执行文件dex文件的结构，以及Android APP动态加载的原理。

### 2.3.1 dex、odex文件结构

apk的加固技术和pc exe的加固原理一样，就是在程序的外面再包裹上另外一段代码，保护里面的代码不被非法修改或反编译，在程序运行的时候优先取得程序的控制权做一些我们自己想做的工作。要进行App加固很重要的一环就是修改Dex文件，而修改Dex文件之前必须对Dex文件有全面的了解。

Android 平台中没有直接使用 Class文件格式，因为早期的 Anrdroid 手机内存，存储都比较小，而 Class 文件显 然有很多可以优化的地方，比如每个 Class 文件都有一个常量池，里边存储了一些字符串。一串内容完全相同的字符串很有可能在不同的 Class 文件的常量池中存在，这就是一个可以优化的地方。当然，Dex 文件结构和 Class文件 结构差异的地方还很多，但是从携带的信息上来看，Dex 和 Class文件是一致的。所以，Dex 文件只不过是class文件一个变种罢了。下图所示为dex文件结构的基本组成[4]：

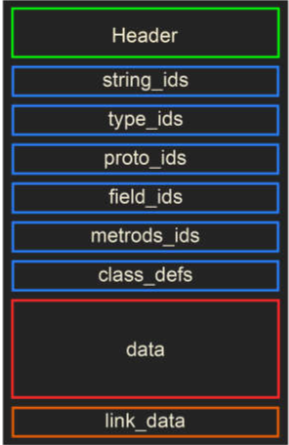


图2.3 class文件结构

整体结构比较简单，由七个结构体组成，Header,文件头，指定了dex文件的一些属性，并记录了其他7部分结构在dex文件中的物理偏移，使用十六进制编辑器查看一个简单的dex文件头。

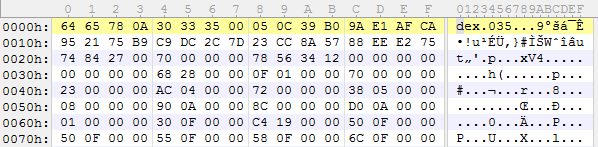


图2.4 一个简单dex文件的header

下图是C语言表示的header数据结构

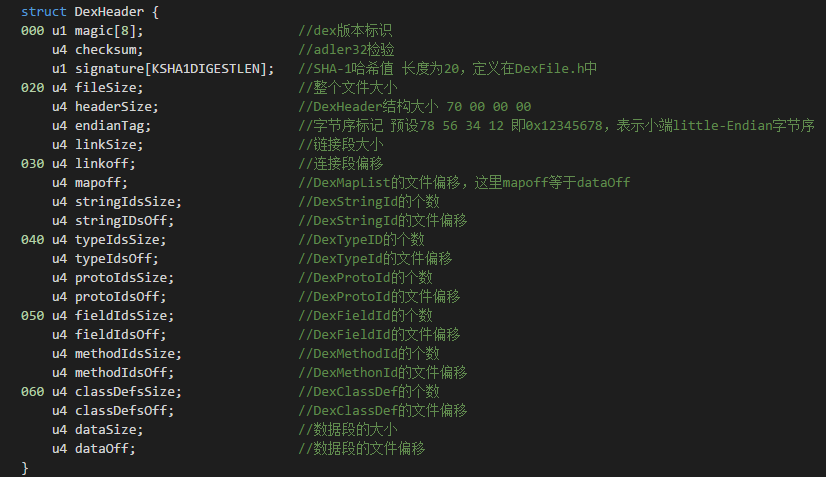


图2.5 header的数据结构

magic字段标识了一个有效的dex文件，目前它的固定值为”dex.035.”，checksum段是dex文件的校验和，通过它可以判断dex文件是否被损坏或篡改。signature字段用来识别最佳化之前的dex文件。fileSize字段记录了整个Dex文件的大小。headerSize字段记录了DexHeader本身占用的字节数，目前值固定为0x70，endian字段指定了dex文件运行环境的CPU字节序。预设小端字节序。linkSize和linkOff指定了链接段的大小和文件偏移，一般情况下都为0。。mapOff指定了DexMapList结构的文件偏移，其余字段分别表示DexStringID，DexTypeID，DexProtoID，DexFieldID，DexMethodID，DexClassDef以及数据段的大小及文件偏移[1]。

odex文件就是optimizedDex的缩写，即优化过的dex文件，odex文件有两种存在方式，一种是从Apk文件中提取出来，与APK文件存放在同一目录切后缀为odex的文件，这一类通常是Android ROM的系统程序，另一种是Dalvik cache的缓存文件，这一类后缀还是dex，存放在cache/dalvik-cache目录下，保存形式为Apk路径@Apk名@classes.dex，

由于Android程序的Apk文件为Zip压缩包格式，Dalvik虚拟机每次加载他们时需要从Apk中读取classes.dex文件，这样会耗费很多CPU时间，而采用odex方式优化的dex文件已经包含了加载dex必须的依赖库文件列表，Dalvik虚拟机只需检测并加载所需的依赖库即可执行相应的dex文件，这大大缩短了读取dex文件所需的时间。

odex文件的写入和读取没有像dex文件那样定义了全系列的数据结构，Dalvik虚拟机将dex文件映射到内存中后是DexFile格式，不过DexFile结构描述的是加载进内存的数据结构，还有一些数据是不会加载进内存的。可以将odex文件结构定义整理如图2.6，DexOptHeader就是odex的头部，Dependence为odex的依赖库列表，ChunkDexClassLookup,ChunkRegisterMapPool,ChunkEnd是整合后的数据结构。

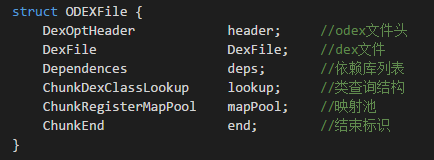


图2.6 odex文件数据结构

odex文件的文件头DexOptHeader在DexFile.h文件中定义如图2.7所示：

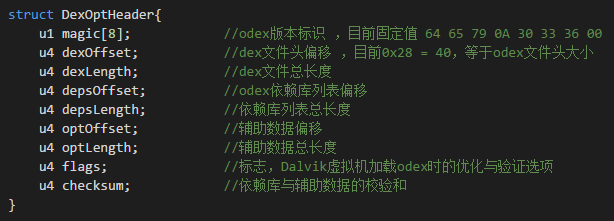


图2.7 odex文件头数据结构

magic字段标识了一个有效的odex文件，目前固定值为”64 65 19 0A 30 33 36 00”，dexOffset字段为dex文件头的偏移，一般等于DexOptHeader的大小0x28，dexLength子字段问dex文件的总大小，depsOffSet字段为依赖库的起始偏移，depsLength字段为依赖库的总长度，flags字段为DexOptFlags的常量值，标识了Dalvik虚拟机加载odex时的优化与验证选项，checksum字段为odex文件的检验和，标识了odex是否合法有效。

### 2.3.2 Android动态加载技术

App加壳之后需要我们在程序运行之前解析出原dex文件并交给系统执行，这里需要Android的动态加载技术，而动态加载技术的核心就是ClassLoader的使用。

Android的Dalvik/ART虚拟机如同标准JAVA的JVM虚拟机一样，在运行程序时首先需要将对应的类加载到内存中，加载流程如图2.8。因此，我们可以利用这一点，在程序运行时手动加载Class，从而达到代码动态加载可执行文件的目的。Android的Dalvik/ART虚拟机虽然与标准Java的JVM虚拟机不一样，ClassLoader具体的加载细节不一样，但是工作机制是类似的，也就是说在Android中同样可以采用类似的动态加载插件的功能。



图2.8 java类加载流程

如果将dex文件加密，类的加载流程就需要增加一步解密过程，而且必须发生在装载之前，如图2.9：



图2.9 加密类加载流程

为了在装载之前解密，必须替换ClassLoader。在Android系统启动的时候会创建一个Boot类型的ClassLoader实例，用于加载一些系统Framework层级需要的类，我们的Android应用里也需要用到一些系统的类，所以APP启动的时候也会把这个Boot类型的ClassLoader传进来。此外，APP也有自己的类，这些类保存在APK的dex文件里面，所以APP启动的时候，也会创建一个自己的ClassLoader实例，用于加载自己dex文件中的类。

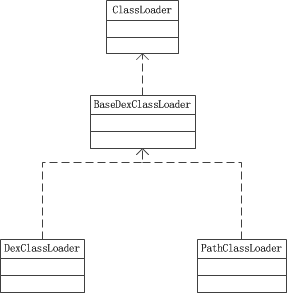


图2.10 Android ClassLoader体系类图

动态加载外部的dex文件的时候，我们也可以使用自己创建的ClassLoader实例来加载dex里面的Class，不过ClassLoader的创建方式有点特殊，创建一个ClassLoader实例的时候，需要使用一个现有的ClassLoader实例作为新创建的实例的Parent。这样一来，一个Android应用，甚至整个Android系统里所有的ClassLoader实例都会被一棵树关联起来，这也是ClassLoader的 双亲代理模型（Parent-Delegation Model）的特点。

使用双亲代理模型主要是为了它的共享作用，一些Framework层级的类一旦被顶层的ClassLoader加载过就缓存在内存里面，以后任何地方用到都不需要重新加载。除此之外还有隔离功能，不同继承路线上的ClassLoader加载的类肯定不是同一个类，这样的限制避免了用户自己的代码冒充核心类库的类访问核心类库包可见成员的情况。这也好理解，一些系统层级的类会在系统初始化的时候被加载，比如java.lang.String，如果在一个应用里面能够简单地用自定义的String类把这个系统的String类给替换掉，那将会有严重的安全问题。

# 文件加密及APP加固需求分析

## 3.1 文件加密

文件加密程序第一次启动之后，应首先提示用户输入密码，并用用户输入再加入一些处理作为密钥来创建保存文件密钥的仓库KeyStore，创建完毕即可进入应用主界面，主界面应该作为一个文件浏览器显示用户外部存储的所有文件，根据文件类型判断用户点击事件的结果，如果是文件夹则打开文件夹，如果是文件则判断是否已加密，未加密弹出加密菜单，已加密弹出解密菜单。最终根据用户点击结果进行相应处理。

程序再次启动则提示用户需输入密码才能进入，根据用户输入的密码解密KeyStore，解密成功则进入应用主界面，解密失败则提示用户重新输入。

根据以上描述，本文件加密系统须具有以下功能：

1. 创建密钥仓库KeyStore；
2. 检索文件；
3. 从KeyStore中增删查密钥；
4. 加密文件；
5. 解密文件；

## 3.2 App加固

App加固时首先要创建一个新的Apk，然后把源程序Apk加密为密文，写入新的APK的dex文件尾部，并在尾部添加密文大小。最终修改新APK checksum，signature，和filesize头信息字段。

根据以上描述，本加固程序须具有以下功能：

1. 加密源程序
2. 修改文件
3. 生成文件摘要
4. 给文件签名

# 文件加密及APP加固系统设计

## 4.1文件加密

文件加密系统从功能上来讲主要包含两个模块，密钥管理以及文件管理，如图4.1，两个模块之间不存在任何交叉功能。文件管理模块负责用户进行的文件检索，文件加解密操作，其中文件加解密需要依赖密钥管理模块存储及检索密钥。而密钥管理模块主要负责用户初始化，后续验证身份以及向文件管理模块提供存储/检索密钥的接口。

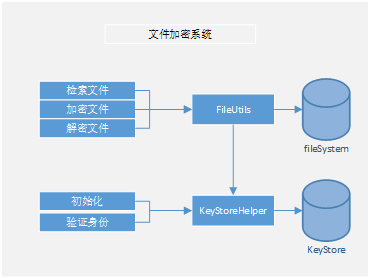


图4.1 文件加密总体架构

### 4.1.1 加密算法的选择

在加密算法的选择中，主要有以下区别：

1. 非对称加密算法的运行速度比对称加密算法要慢很多，当需要加密大量数据时，建议采用对称加密算法。
2. 对称加密算法不能实现签名，因此签名只能非对称算法。
3. 由于对称加密算法的密钥管理是一个复杂的过程，密钥的管理直接决定着他的安全性，因此当数据量很小时，可以考虑采用非对称加密算法。
4. 非对称加密算法基于未解决的数学难题，在破解上几乎不可能。对于对称加密算法密码算法，到了AES虽说从理论来说是不可能破解的，但从计算机的发展角度来看。公钥更具有优越性。
5. 选定了加密算法后，那采用多少位的密钥呢？一般来说，密钥越长，运行的速度就越慢，应该根据的我们实际需要的安全级别来选择，一般来说，RSA建议采用1024位的数字，ECC建议采用160位，AES采用128为即可。

由于我们的文件加密系统是在本地加密并不需要在网络中通信，而且可能会加密视频、音频等比较大的文件，而在加密数据时非对称加密的效率没有对称加密高，所以本系统需要采用对称加密算法。为了保障密钥的安全，再使用非对称加密算法加密密钥。

目前常用的对称加密算法主要有三种：

1. DES（Data Encryption Standard）:

数据加密标准，速度较快，适用于加密大量数据的场合。

1. 3DES (Triple DES):

是基于DES，对一块数据用三个不同的密钥进行三次加密，强度更高。

1. AES（Advanced Encryption Standard）：

高级加密标准，新一代的加密算法标准，速度快，安全级别高。

由于DES是1997年公布实施的，现在来说已经有些落后，3DES和AES的安全性都比其高很多，所以要在2DES和AES中选择一个，两者对比如下：

表2.1 AES与3DES加密算法的比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法名称 | 算法类型 | 密钥长度 | 速度 | 破解时间 | 资源消耗 |
| AES | 对称Block密码 | 128、192、256位 | 高 | 1490000亿年 | 低 |
| 3DES | 对称feistel密码 | 112或168位 | 低 | 46亿年 | 中 |

通过两者的对比，根据文件加密系统的实际情况，选择AES加密算法，在密钥长度的选择中，虽然通常长度越大的密钥破解难度越大，但是加解密效率也越低，而且随着计算机的发展，密钥长度对破解难度的影响并不是很大，所以采用128位密钥即可。

### 4.1.2 密钥管理模块

在加密文件过程中，为了增强安全性，每一个文件的加密密钥都不同，所以加密文件所用的密钥应该由系统自动生成，然后存储在文件中，在java中，存储密钥最方便安全的莫过于KeyStore。加载KeyStore的密码由用户输入而不存储在本地，用户只需记住KeyStore的密码，如果遗忘，不能找回。

密钥存储方案如图4.1：

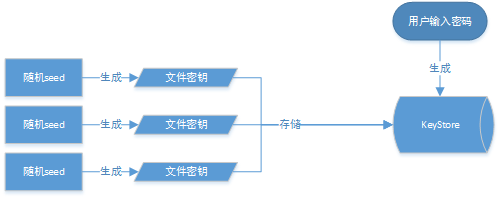


图4.1 密钥存储方案

### 4.1.3 文件管理模块

文件管理模块控制了文件检索，文件加解密三个功能的实现，用户验证身份之后进入主界面，此时主界面的列表请求获取文件列表，显示在界面上，待用户点击，

## 4.2 App加固

App加固主要包含两个流程，加壳流程和脱壳流程。

加壳流程一共包括三步，第一步将源文件加密，第二步将加密后的源文件与壳文件合并，最后修改合并后的dex文件使其合法。加密时由于Apk文件为了效率以及安全性考虑依然采用AES加密算法。



图4.2 Apk加壳流程

脱壳模块流程比较复杂，但是主要就是ClassLoader的运用，在类加载之前将类加载器替换成自定义的加载器，解密带壳程序，然后使用自定义的加载器加载解密出的源文件[3]。

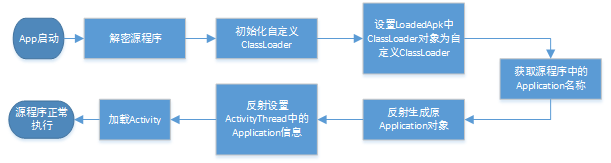


图4.2 Apk脱壳流程

# 文件加密及APP加固详细设计

## 5.1 文件加密系统详细设计

### 5.1.1 界面设计

文件加密系统包含两个界面，一个密码输入界面，一个文件浏览界面；

密码输入界面包含两个部分，输入框使用四个RadioButton代表当前输入密码长度。输入面板使用10个原型按钮表示十个阿拉伯数字以及一个回删按钮用来修改输入。

用户可输入的密码为四位，为了增强安全性，系统自动取得设备ID与密码一起进行简单加密后作为生成或加载KeyStore的密码。第一次启动时提示用户设置密码，输入满四位之后生成KeyStore并进入主界面。后续启动提示用户需输入密码，输入满四位之后开始进行加载，加载成功则跳转至主界面，加载失败则提示用户重新输入。

文件浏览界面也包含两个部分，顶部Toolbar显示当前文件夹路径，下方使用一个RecyclerView显示当前文件夹中的所有文件（包括文件夹及各种类型文件），RecyclerView的每一项显示一个代表文件夹或者普通文件的图标和文件名称。

点击文件夹就会跳转至此文件夹目录并刷新Toolbar显示的路径，长按文件夹则提示是否将此文件夹下所有文件全部加密。点击普通文件则首先判断是否已经加密，未加密则弹出加密菜单，已加密则弹出揭秘菜单，等待用户操作。

### 5.1.2 密钥管理模块

密钥管理模块包含两个类，KeyTools和KeyStoreHelper，并且KeyStoreHelper的许多功能依赖于KeyTools来实现。

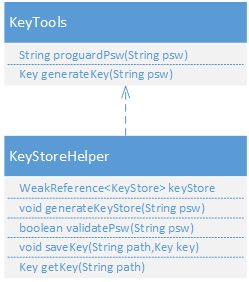


图 5.1 密钥管理类图

KeyTools作为一个工具类，两个方法的功能都和Key的生成有关，用户输入的密码仅仅包含四位，在使用前需要将其与特定字符串混淆来增强破解难度。第二个方法有两种形式，有参数则根据参数生成一个Key并返回，没有参数则生成一个随机的Key并返回。

KeyStoreHelper是KeyStore的管理者，其它所有对象想要访问KeyStore都需要通过它来实现，并且由于KeyStore作为一个文件，需要保障它的线程安全性，并且每次加载都比较耗时，所以需要使用单例模式来访问它。并且使用缓存机制(使用弱引用来保存)，如果已经加载并且没有被回收，则不需要再次加载，但是如果内存不足，就即使释放避免内存不足。

### 5.1.3 文件管理模块

文件管理模块的功能全部集中在一个FileUtils类中，包括新建文件，删除文件以及加解密等。FileUtils依赖于KeyStoreHelper执行文件的加解密操作，类图如图5.2：

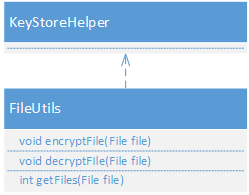


图 5.2 文件管理类图

## 5.2 App加壳详细设计

App加壳程序需要分为两个部分，加壳程序是一个java工程，输入一个Apk文件，输出加密过的Apk，而脱壳程序是融合在Apk中的一部分，指的是App启动之后执行的一段程序。

### 5.2.1 加壳程序设计

加壳部分功能主要依赖两个工具类来完成：EncryTools和FileTools，类图如图5.3所示；

EncryTools的功能主要包含两个部分，产生密钥，加密字节流。功能虽少但却是核心部分，ApkShellTool获取到源文件字节流之后先用EncryTools来产生一个密钥，然后返回加密字节流。

FileTools的功能比较多，首先需要打开文件并获取到字节流，并且将加密过的字节流写入到文件中，还要能够重新签名文件并修复文件摘要。

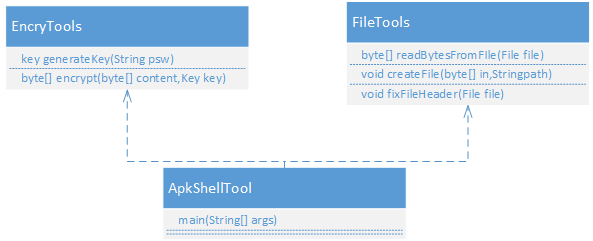


图5.3 App加壳类图

### 5.2.2 脱壳程序设计

脱壳程序需要在App启动之后首先执行，而对于Android来说启动一个应用之后首先执行的就是Application类了，而在Application类中首先回调的是attachBaseContext()方法，因此脱壳程序的入口就在Application中的attachBaseContext()方法中。

脱壳时的三个步骤：

1. 替换回原有Application。

给Application加壳时替换了Apk的Application来达到脱壳的目的，正常运行是需要替换会原有的Application。通过在AndroidMainfest.xml文件中配置原有Applicaiton类信息来达到我们的目的。解壳程序要在运行完毕后通过创建配置的Application对象，并通过反射修改回原Application。

1. 通过DexClassLoader动态加载源Apk代码。

根据双亲代理模型，我们自定义的ClassLoader需要加入到系统ClassLoader的家族体系中去，加载出的类才能够真正被系统识别。

1. 配置Apk资源

代码默认引用的Apk资源文件在最外层的已加密的程序中，因此要添加系统资源加载路径才能实现对脱壳后APK文件资源的加载。

# 测试

## 6.1 文件加密测试

1. 功能测试

主要测试系统的三个功能，身份验证，文件加密，文件解密。

当输入错误密码时如图6.1左一,测试文件加密前文本如图6.1左二，加密后使用文本编辑器打开如图6.1右一.解密后如图6.1右二

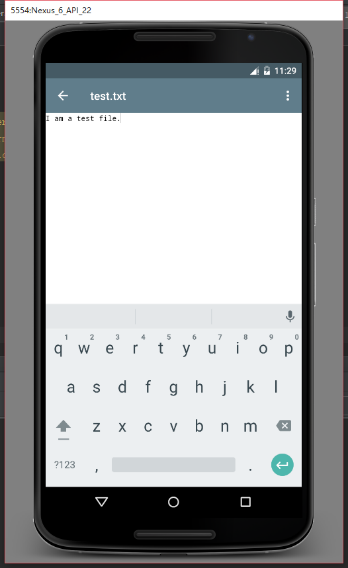
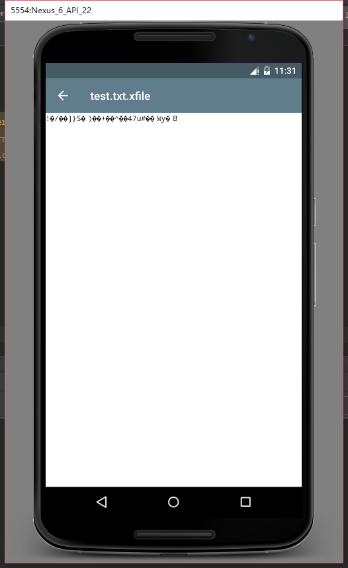
-

图6.1 文件加密测试结果

## 6.2 App加壳测试

对未进行加壳的Apk使用Apktool反编译

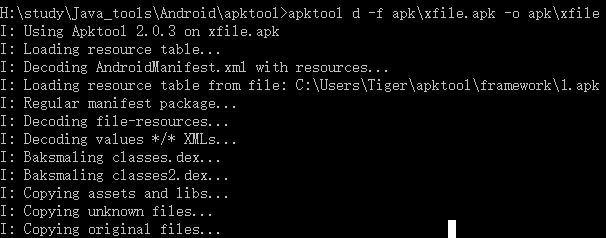


图6. 2 反编译过程

使用Visual Studio查看输出的文件夹，反编译成功，可查看到原文的xml文件如图：

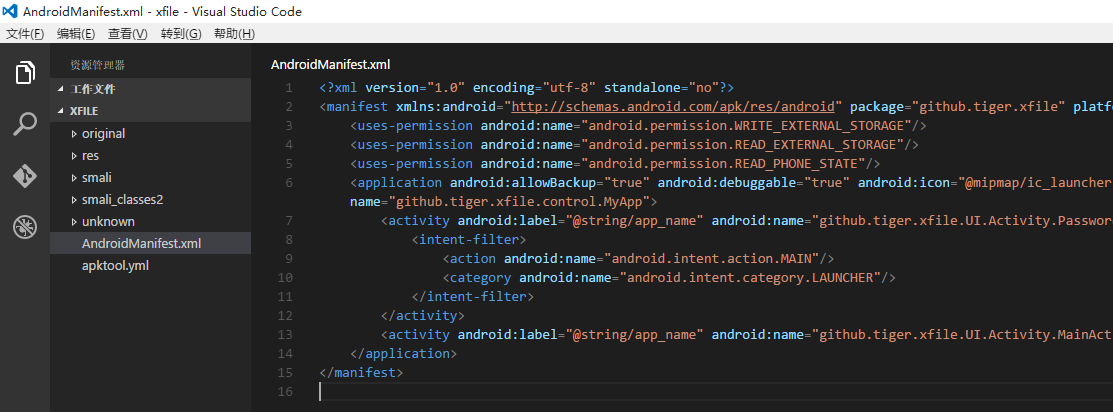


图6.3 查看xml文件结果

解压未加密apk，得到classes.dex，使用dex2jar工具转为jar文件，转换结果如图使用jd-jui个工具查看jar文件可看到源代码如图6.5：

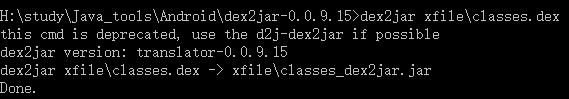


图6.4 dex文件转换结果

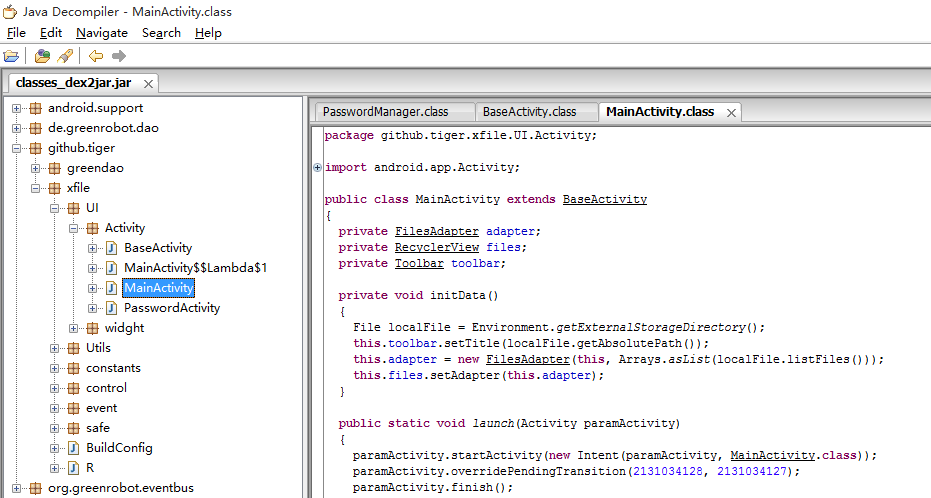


图6.5 查看转换结果

App加壳后，再使用Apktool进行反编译：（待完成）

# 总结与展望

现在只是简单的完成了一个加密文件和加固Apk文件的系统，虽然功能基本满足了需求，但是跟最初的想法还是相差甚远的。

首先拿文件加密系统来讲，目前没有做查看文件的功能，以至于只能将文件解密到硬盘上，再借助于其他的查看文件的软件进行查看，对于文件保护本身来讲这样做是很不安全的。

下一步继续完善这个软件的话，首先第一步是要田间常见文件类型浏览功能，比如文本，图片和视频文件，第二步是查看小文件时不将文件解密到硬盘上，而是直接在内存中读取并显示，减少被恶意软件获取到的风险。对于大文件的解密方法，可以从文件系统本身去找解决办法。第三步是从底层入手，监听用户空间中运行的程序申请对文件的访问，如果访问到已经加密的文件，可以直接提示用户解密。这样最为人性化，也最为安全。

对于App加固来讲，目前实现的功能都是写在java文件里的，被破解的风险还是有，只是提高了破解者的技术门槛而已，下一步打算是写在用C或者C++写的so库中，增加安全性。总之破解与反破解的技术始终是道高一尺魔高一丈，只有不断的进步才能保持不被落下。

# 致谢

寒来暑往，乌飞兔走，四年大学生活转瞬即逝，回忆起大一刚刚进入大学的日子，虽然依然很清晰感觉像过去了不久，但是又深刻的体会到了自身发生的变化，体会到了大学对于人的塑造作用。在这四年期间，我不仅在生活能力，思想素质上得到了很大的提高，更在专业能力上有了质的飞跃。

在本篇论文完成中，我首先要感谢的廖俊国老师耐心而又专业的引导，在本次毕业设计开始时，我对于设计的目标都一直是很迷茫的，经过和廖老师一次一次的商量之后才最终确立了软件设计目标，而后针对项目中的难点，廖老师又提出了很多建设性的建议，本项目能够顺利完成，完全离不开廖老师的指导。

其次，还要感谢在外面工作过程中帮助过我的领导和同事，正是他们的引领让我在工作过程中学到了很多的知识和本领，这些知识在本次项目的完成中也是居功甚伟的。而且对于我技术的成长也会一直有着很大的作用。

同时，我也要感谢在学校时帮助过我的老师和同学们，你们的帮助让我在学习过程中体会到了集体的温暖和友谊的珍贵，体会到了大学生活。

最后，我还要感谢我的父母，感谢他们含辛茹苦把我养他，不辞艰辛的供我读了大学，受到的教育最重要的莫过于这二十多年来莫过于来自父亲母亲无私的爱。你们是我始终值得信赖的后盾。愿父母永远健康快乐！

在论文即将完成之际，再次感谢这一路上帮助过我，指导过我的人，祝他们身体健康，心想事成！

# 参考文献

[1]吴倩 赵晨啸 郭莹.Android安全机制解析与应用实践[M].北京.机械工业出版社.2013.5.

[2] Wikipedia. Advanced Encryption Standard. https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard.

[3] jack\_jia Android App加壳技术方案. <http://blog.csdn.net/androidsecurity/article/details/8809542>

[4] 丰生强.Android软件安全与逆向分析[M].北京.人民邮电出版社.2013.2.