**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 - 1 -](#_Toc450870854)

[1.1 选题背景 - 1 -](#_Toc450870855)

[1.1.1 手机市场分析 - 1 -](#_Toc450870856)

[1.1.2 Android用户数据安全 - 1 -](#_Toc450870857)

[1.1.3 Android应用盗版情况 - 2 -](#_Toc450870858)

[1.2 选题意义 - 2 -](#_Toc450870859)

[1.3 论文组织结构 - 2 -](#_Toc450870860)

[第二章 相关基础知识 - 4 -](#_Toc450870861)

[2.1 开发环境 - 4 -](#_Toc450870862)

[2.2 加密相关 - 4 -](#_Toc450870863)

[2.2.1 加密算法的选择 - 4 -](#_Toc450870864)

[2.2.2 AES原理 - 5 -](#_Toc450870865)

[2.3 Apk加壳技术 - 7 -](#_Toc450870866)

[2.3.1 dex、odex文件结构 - 8 -](#_Toc450870867)

[2.3.2 ClassLoader - 10 -](#_Toc450870868)

[第三章 文件加密及APP加固需求分析 - 12 -](#_Toc450870869)

[3.1 文件加密 - 12 -](#_Toc450870870)

[3.2 App加固 - 12 -](#_Toc450870871)

[第四章 文件加密及APP加固系统设计 - 13 -](#_Toc450870872)

[4.1 密钥存储方案 - 13 -](#_Toc450870873)

[第五章 文件加密及APP加固详细设计 - 13 -](#_Toc450870874)

[第六章 测试 - 13 -](#_Toc450870875)

[第七章 总结与展望 - 13 -](#_Toc450870876)

[第八章 致谢 - 13 -](#_Toc450870877)

[第九章 参考文献 - 13 -](#_Toc450870878)

# 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Q1和2016年Q1 Android系统市场占有率稳步提升3.4%，达到了76.4%。

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。

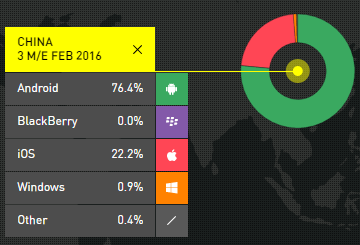
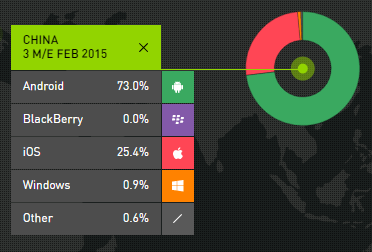


图1.1 2015及2016智能手机系统市场占有率

### 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. 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。

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。

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 加密相关

### 2.2.1 加密算法的选择

加密算法根据密钥类型不同一般分为两类：对称加密算法和非对称加密算法。首先要决定使用对称还是非对称加密算法。

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。

### 2.2.2 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)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：

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)（round key）做[XOR运算](https://zh.wikipedia.org/wiki/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)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 |

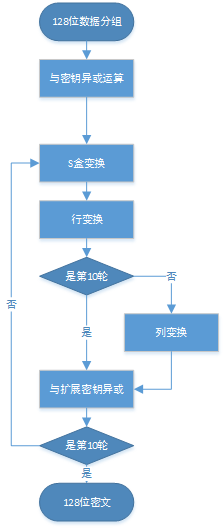


图 2.1 AES算法流程图

各个步骤的实现方式：

1. S盒变换（SubBytes）

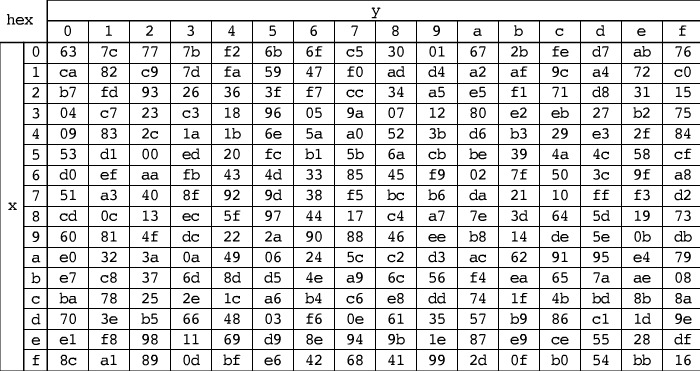


图2.2 AES中的S盒

矩阵中的各字节通过一个8位的[S-box](https://zh.wikipedia.org/wiki/S-box)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)（28）上的乘法[反元素](https://zh.wikipedia.org/wiki/%E5%8F%8D%E5%85%83%E7%B4%A0)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)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)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[1]：

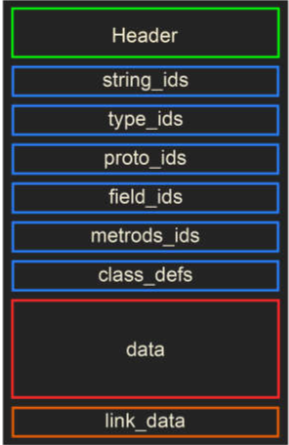


图2.3 class文件结构

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。

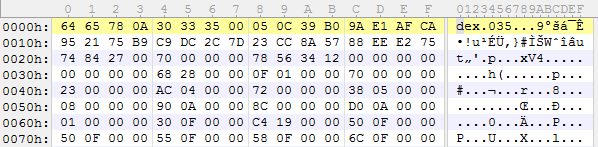


图2.4 一个简单dex文件的header

下图是C语言表示的header数据结构

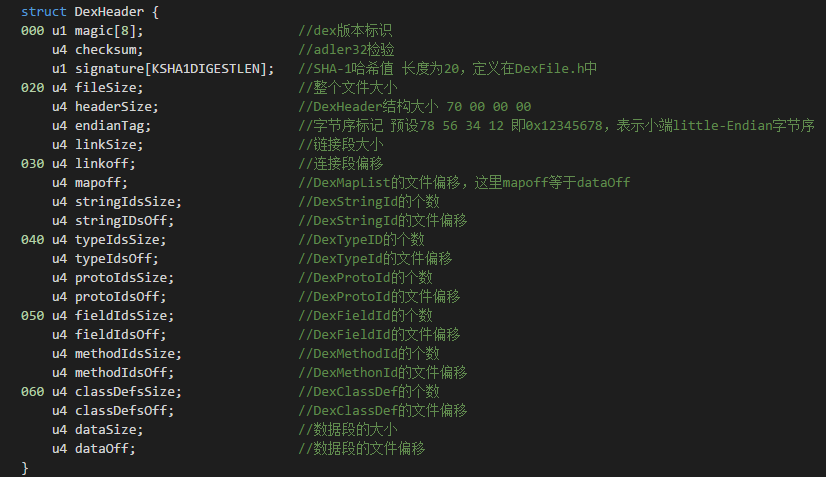


图2.5 header的数据结构

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。

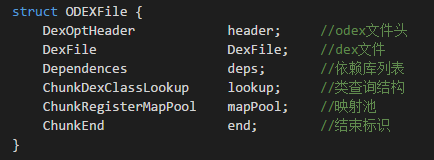


图2.6 odex文件数据结构

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

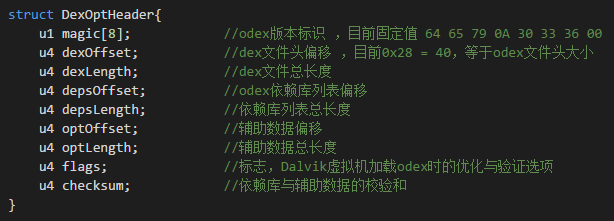


图2.7 odex文件头数据结构

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 ClassLoader

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。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

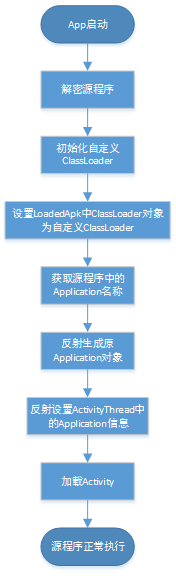


图2.8 加固Apk加载流程

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。

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。

# 文件加密及APP加固需求分析

## 3.1 文件加密

## 3.2 App加固

Android平台以其免费和开源的特性使其占据了移动领域大半壁江山，越来越的人投身到移动应用开发中，但本文恐怕要先给Android应用开发者泼盆冷水，据360发布的《2014年中国手机安全状况报告》显示，2014全年，360互联网安全中心累计监测到Android用户感染恶意程序3.19亿人次，平均每天恶意程序感染量达到了87.5万人次。同时，Android应用被破解和盗版等事件也层出不穷。

很明显，Android平台已经成为恶意程序和破解者攻击的众矢之的，于是越来越多的Android开发者开始意识到应用安全的重要性。

# 文件加密及APP加固系统设计

## 4.1 密钥存储方案

# 文件加密及APP加固详细设计

# 测试

# 总结与展望

# 致谢

# 参考文献

[1] 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.