

**课 程 实 验 报 告**

**课程名称： 逆向工程分析技术**

**专业班级： 信息安全 1703 班**

**学 号： U201714513**

**姓 名： 徐立威**

**指导教师： 鲁宏伟**

**报告日期： 4.11**

**网络空间安全学院**

目录

[1. 实验准备 1](#_Toc29121)

[2. 实验内容 2](#_Toc3176)

[3. 实验目标 2](#_Toc2420)

[4. 分析方法和过程 3](#_Toc17144)

[4.1. 研究一——发现算法 3](#_Toc32098)

[4.2. 研究二——密钥发现 7](#_Toc28281)

[4.3. 研究三——确认加解密过程 9](#_Toc31395)

[4.4. 研究四——decrypt函数 10](#_Toc13958)

[4.4.1. 分析一——参数调用 10](#_Toc9644)

[4.4.2. 分析二——密钥生成 11](#_Toc25595)

[4.4.3. 分析三——创建密钥的类 12](#_Toc26936)

[4.4.4. 分析四——生成真正使用的密钥 12](#_Toc11955)

[4.4.5. 分析五——解密过程 13](#_Toc23961)

[4.4.6. 分析六——总结 15](#_Toc5650)

[4.5. 研究五——encrypt函数 16](#_Toc5016)

[4.6. 研究六——加解密文件存储 18](#_Toc742)

[4.7. 研究七——随机数生成函数 22](#_Toc10056)

[4.8. 跟踪加解密函数调用图 23](#_Toc9586)

[5. 实验结论 24](#_Toc25206)

[6. 实验心得 26](#_Toc22678)

# 

# 实验准备

准备工具：

1. 需要安装Jeb3、Android Killer、Java虚拟机等软件
2. 由于本实验需要蓝牙连接，普通的虚拟机是无法进行动态调试的，所以需要一部Android手机和USB线

Android基本文件介绍：

主要流程：Java文件在编译成class文件，然后经过Android平台的dx 工具转换为Dex文件后，同Native code（JNI）和资源一 起打包成apk，apk安装到手机后解压出Dex文件。Dalvik会通过dexopt工具将Dex优化，成为Odex文件。Odex文件的效率比Dex高，但其中大部分代码仍然需要 每次执行时编译； 而ART则会将Dex通过dex2oat工具编译得到一个ELF文件 ，它是一个可执行的文件。用图像表示就如图1.1。

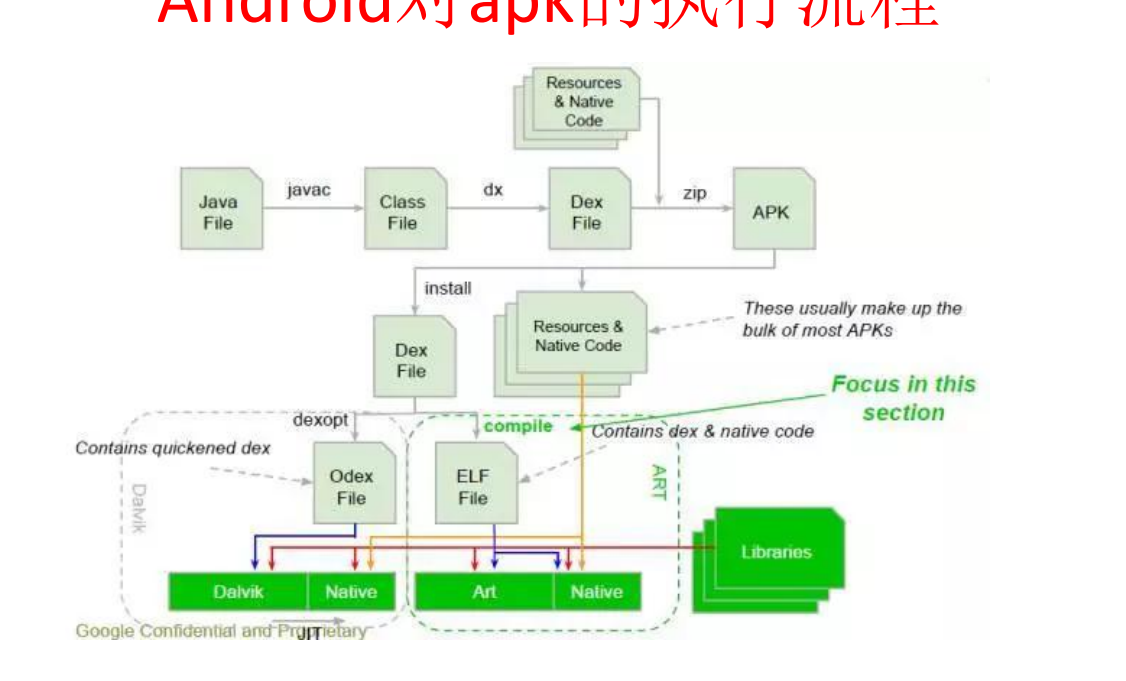


图1.1

Dex文件：dex是Android平台上(Dalvik虚拟机)的可执行文件, 相当于Windows平台中的exe文件, 每个Apk安装包中都有dex文件, 里面包含了该app的所有源码, 通过反编译工具可以获取到相应的java源码。

Class文件： Class文件是一组以8位字节为基础单位的二进制流，各个数据项目严格按照顺序紧凑地排列在Class文件之中，中间没有添加任何分隔符。根据Java虚拟机规范的规定，Class文件结构采用一种类似于C语言结构体的伪结构来存储数据，这种伪结构只有两种数据类型：无符号数和表。

# **实验内容**

分析Android APK名称：

[TRAVEL\_AND\_LOCAL][769][100000][1.0.3]FindElfI.apk

具体信息如图2.1。

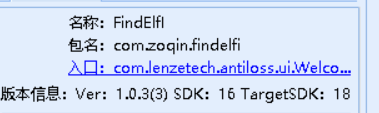


图2.1

# **实验目标**

1. 分析给定的Android APK中在加密函数调用中出现的常值字符串、数组是否为密钥，如为密钥，确认被加密的数据内容是用以本地数据还是网络通信数据，即加密结果是存储在本地还是通过网络传输到服务器。

（2）分析给定APK中是否调用了随机数生成函数。如果有，生成的结果是否与密钥生成过程相关，如有，记录随机数种子的产生方式，并跟踪基于该随机数生成的密钥是否用以网络传输数据的加密。

# **分析方法和过程**

## 研究一——发现算法

分析方法：信息反馈

分析工具：android killer、JD

分析过程：

1. 首先，我利用Android killer搜索有关security、des、rsa等关键字，分别如图4.1.1、图4.1.2、图4.1.3。

基本可以确定存放加密算法的几个文件，分别是AESUtil.smali、Base64.smali和Deviceid.smali当中。

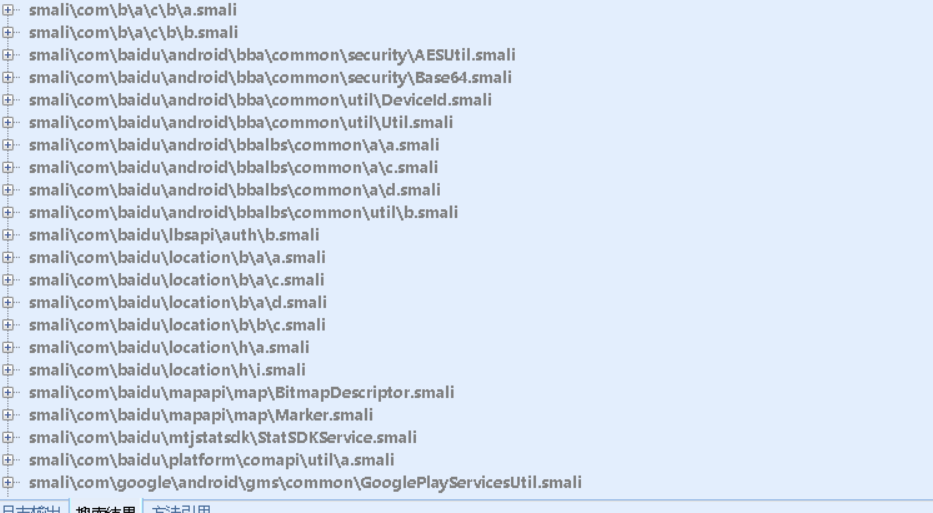


图4.1.1

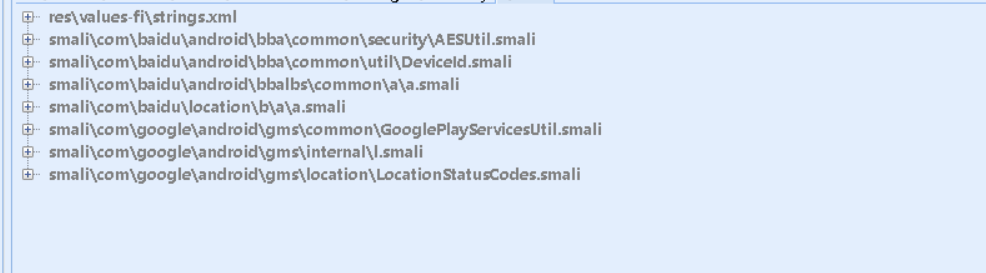


图4.1.2

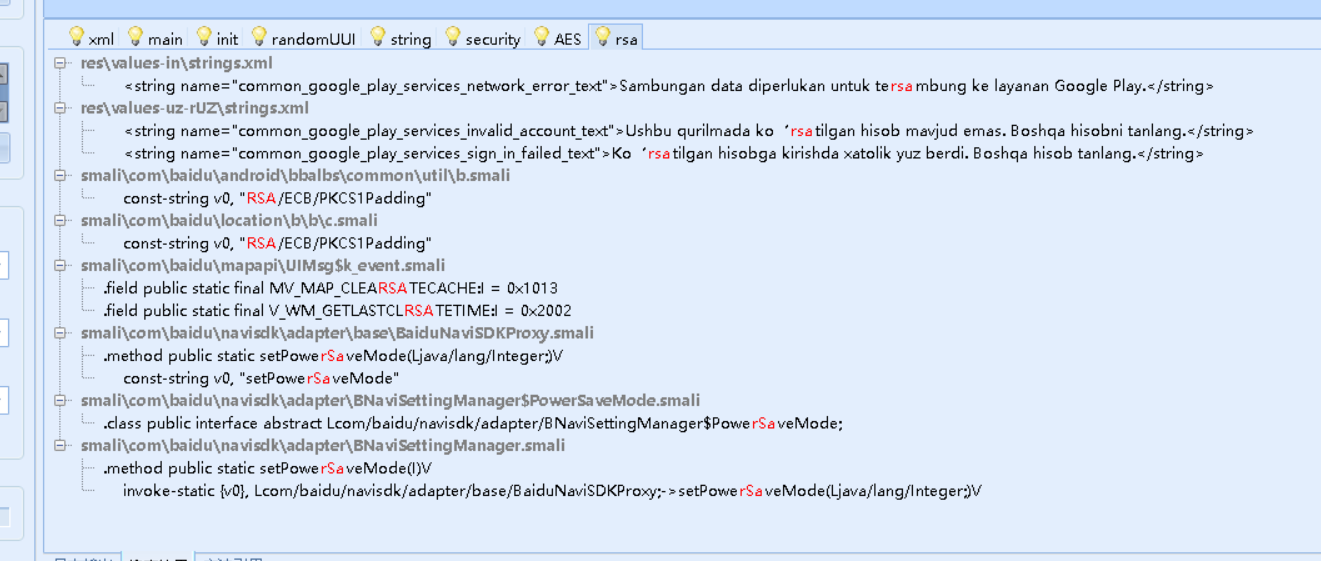


图4.1.3

1. 从上面的三幅图中，我可以简单的看出来，该apk使用的是AES加密算法。利用JD进行下一步的分析，点开security文件夹，里面有两个文件，如图4.1.4。感觉这个apk并没有试图掩饰自己的加密算法，文件名太明显了。

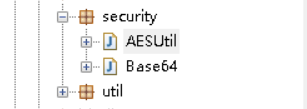


图4.1.4

1. 先尝试着对AESUtil进行分析，其源码如图下4.1.5所示。

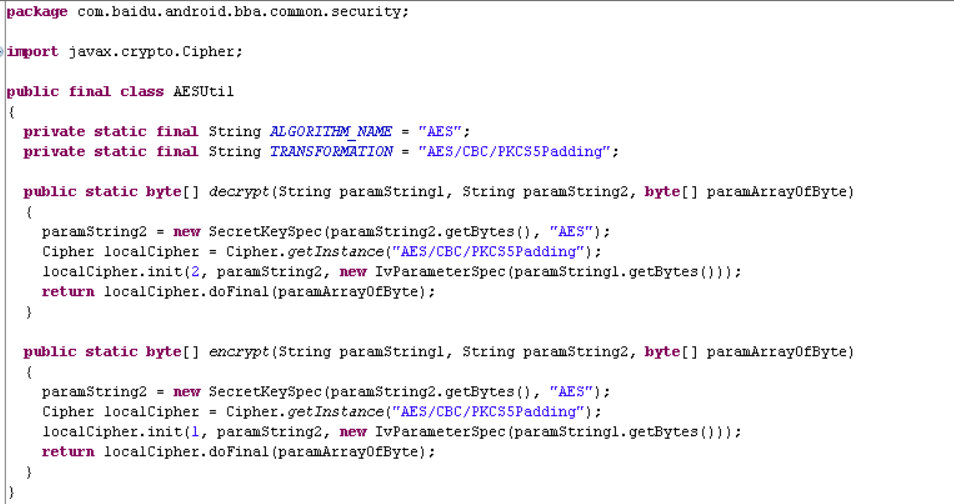


图4.1.5

从源码中，可以提取一些重要的信息。例如：

private static final String ALGORITHM\_NAME = "AES";

private static final String TRANSFORMATION = "AES/CBC/PKCS5Padding";

略微查了一下ALGORITHM\_NAME是算法名称，因此完全可以确定，该APK的加密算法就是AES算法。

然后AES/CBC/PKCS5Padding也查了一下，是JAVA的一种专门进行加解密算法。于是我查阅相关的知识，看到一个博客是就这种加密算法的讨论，从中我初步了解了该算法的实现过程。

博客地址：<https://blog.csdn.net/u013938484/article/details/80415143>

根据博客中的介绍，主要是包括两个函数，加解密函数，分别是

Encrypt(String sSrc, String sKey)和Decrypt(String sSrc, String sKey) 。

恰好反编译的apk的加解密算法正如所述，如下所示：

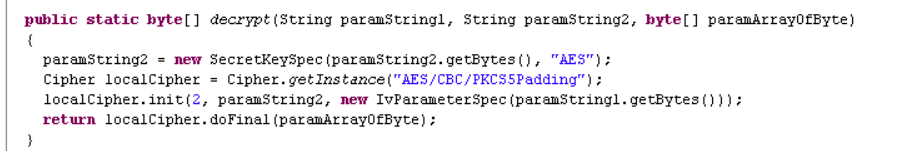


图4.1.6——解密算法

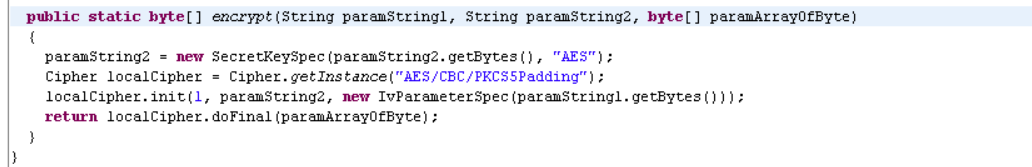


图4.1.7——加密算法

paramString2 = new SecretKeySpec(paramString2.getBytes(), "AES");这句话根据我所查的资料，函数中两个参数，第一个为私钥字节数组， 第二个为加密方式 AES或者DES。

1. 接着对Base64分析，源码如下。由于代码过长，就只截图一部分，具体的在下面进行分析。

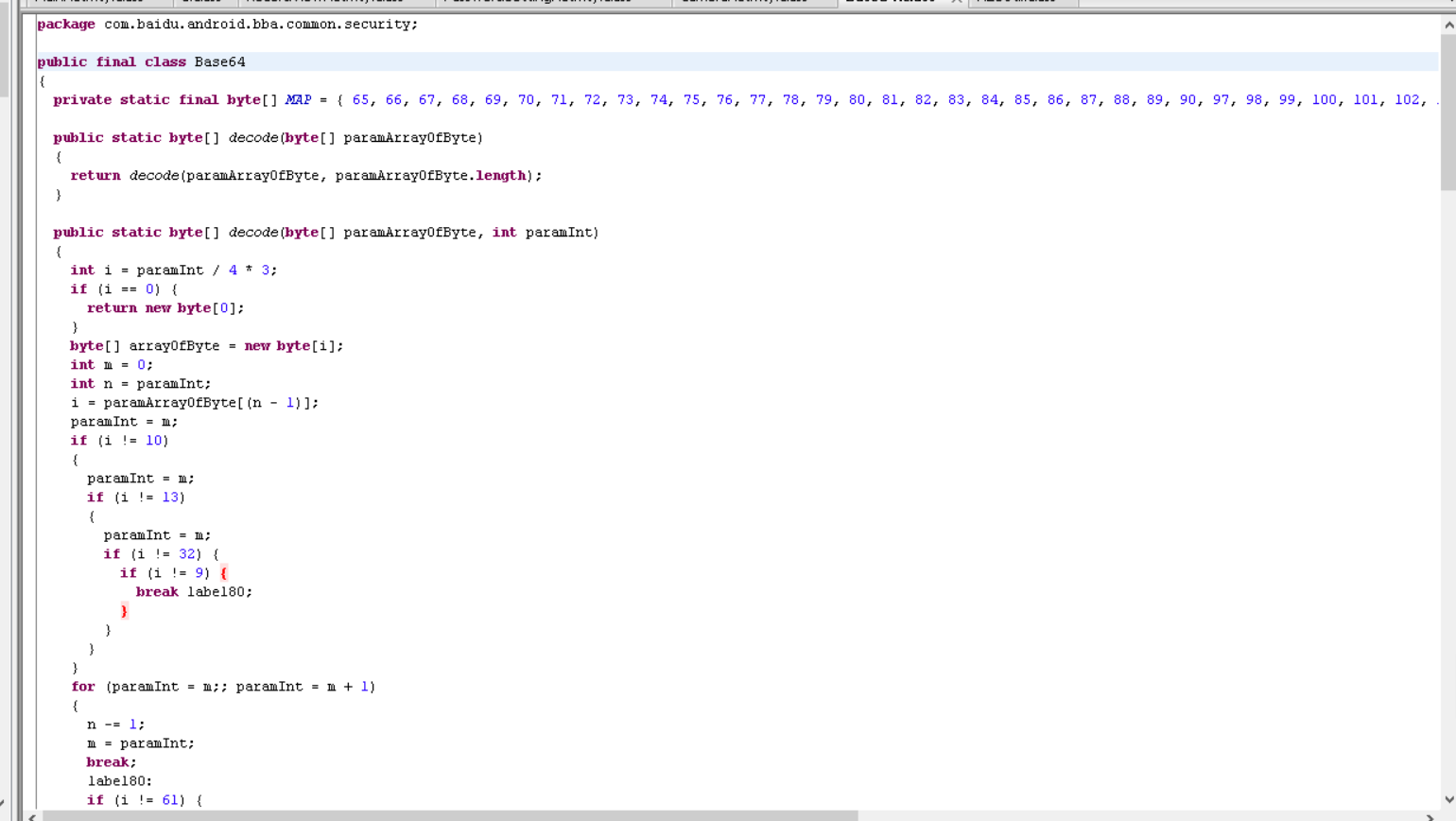


图4.1.8

从百度百科搜索的关于Base64的信息：Base64是网络上最常见的用于传输8Bit字节码的编码方式之一，Base64就是一种基于64个可打印字符来表示二进制数据的方法，这等于再次的加密。

1. 加解密过程并不是现在研究的重点，之后会进行详细的的介绍，目前已知的是加解密函数的位置以及所使用的算法。

## 研究二——密钥发现

分析方法：信息反馈

分析工具：JEB3、android killer、JD

分析过程：

1. 直接搜索aes，根据之前的研究，猜测AES算法是主要算法，如图4.2.1。



图4.2.1

1. 在搜索的文件里，我们发现有一个变量名为AES\_Key,所以我们猜测这就是AES的密钥，如图4.2.2。同样的我也可以在JD中找到，如图4.2.3。

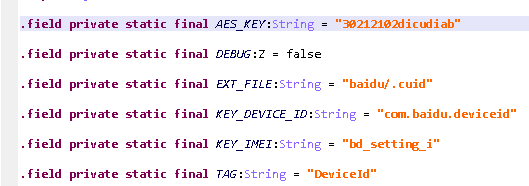


图4.2.2

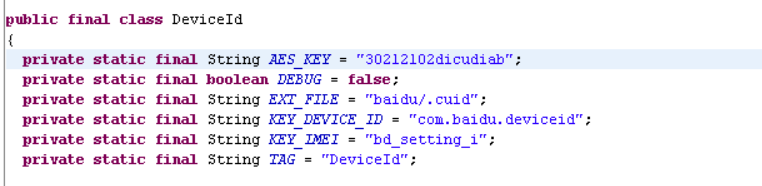


图4.2.3

1. 从图中还可以得到其他的信息，信息如下

private static final String AES\_KEY = "30212102dicudiab";

//猜测为密钥

private static final boolean DEBUG = false;

//这样形式的代码然后用于条件编译，把一些只用于调试用途的代码保留在源码//里但让其不起作用

private static final String EXT\_FILE = "baidu/.cuid";

//从字面上了解是应该是退出保存文件

private static final String KEY\_DEVICE\_ID = "com.baidu.deviceid";

//个人感觉是设备的ID，但是不明白后面这个com.baidu.deviceid是什么，可能局势ID的内容

private static final String KEY\_IMEI = "bd\_setting\_i";

//IMEI是国际移动设备识别码

private static final String TAG = "DeviceId";

//标签，类似ID

在这次的研究中，找到了意思AES算法的密钥。

## 研究三——确认加解密过程

分析方法：函数追踪

分析工具：JEB3、android killer、JD

根据4.1的发现，有两个对密钥进行操作的函数，一个decrypt，一个encrypt函数。我利用android killer扫描decrypt。如图4.3.1，这里显然上面是之前照的关于decrypt函数的声明，而下面那个是唯一的函数的调用。接着我接住JD打开下面的函数调用的位置，如图4.3.2。

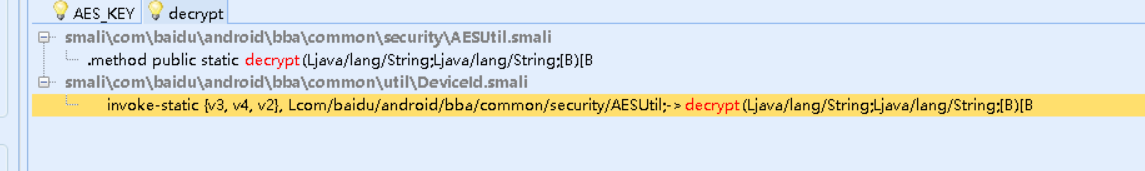


图4.3.1



图4.3.2

根据后面的研究4.4的研究，基本能够确定decrypt，encrypt的功能，前两个参数是密钥，第三个参数是明文、密文。

那个唯一一个调用encrypt函数的代码如下所示：

localObject = new String(AESUtil.decrypt("30212102dicudiab", "30212102dicudiab", Base64.decode(localStringBuilder.toString().getBytes()))).split("=");

从这里我可以直接跳过其他部分，只对加密的信息来源进行分析，其中加密的信息来源是Base64.decode(localStringBuilder.toString().getBytes()))).split("=")，Base64、decode都是简单加密过程，而getBytes是获得二进制串，我们暂时忽略，直接检查localStringBuilder.toString()这个函数。

这部分的具体分析详情请看4.4和4.5。

## 研究四——decrypt函数

Decrypt是安全函数的解密函数，现在根据函数本身以及函数调用关系，来探究一下该函数的具体实现方式以及功能。

下面是函数本体申明，然后就每一行进行详细的分析，另外也把调用函数的方式也显示出来，方便分析。

1 public static byte[] decrypt(String arg4, String arg5, byte[] arg6) {

2 SecretKeySpec v0 = new SecretKeySpec(arg5.getBytes(), "AES");

3 Cipher v1 = Cipher.getInstance("AES/CBC/PKCS5Padding");

4 v1.init(2, ((Key)v0), new IvParameterSpec(arg4.getBytes()));

5 return v1.doFinal(arg6);

6 }

调用：String[] v1\_5 = new String(AESUtil.decrypt("30212102dicudiab", "30212102dicudiab", Base64.decode(v2.toString().getBytes()))).split("=");

### 分析一——参数调用

参数有三个，String arg4, String arg5, byte[] arg6。前两个是一样的，是本地的密钥，这个可以从调用函数可以看出来，均是30212102dicudiab。第三个，暂且看不出来，不过可以推测这是解密后的信息。

### 分析二——密钥生成

其中，paramString2是之前输入的参数，初始值是30212102dicudiab。SecretKeySpec函数是用于生成扩展密钥的函数，所采用的算法是AES。可以看做30212102dicudiab作为原始字节，来生成一组扩展密匙，并放入一个数组v0之中。

下面是关于这个函数的具体调用功能：

SecretKeySpec public SecretKeySpec(byte[] key,String algorithm)

根据给定的字节数组构造一个密钥。

此构造方法不检查给定的字节实际上是否指定了一个指定算法的密钥。例如，如果算法是 DES，此构造方法不检查 key 是否是 8 字节长，并且也不检查是否是全弱或者半弱密钥。为了执行这些检查，应当使用一个特定于算法的密钥规范（在此情况下是：DESKeySpec）。

参数：

key - 密钥的密钥内容。复制该数组的内容来防止后续修改。

algorithm - 与给定的密钥内容相关联的密钥算法的名称。有关标准算法名称的信息请参见 Java Cryptography Architecture Reference Guide 中的附录 A。

抛出：

IllegalArgumentException - 如果 algorithm 为 null 或者 key 为 null 或空。

### 分析三——创建密钥的类

首先，查阅了下getinstance的函数意义，如下所示：

getInstance是一个函数，在java中，可以使用这种方式使用单例模式创建类的实例，所谓单例模式就是一个类有且只有一个实例，不像object ob=new object();的这种方式去实例化后去使用。

其次，查看类似函数的用法，发现这就是个创造一个密钥类的函数，大多情况都是这么操作。

博客地址：https://www.cnblogs.com/hailiang/archive/2012/02/29/2373154.html

### 分析四——生成真正使用的密钥

和分析三的操作是对应的，是把创造的密钥进行初始化，并指定密钥，总结下分析三和分析四的操作。

调用AES/DES加密算法包最精要的就是下面两句话：

Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, key, zeroIv);

CBC是工作模式，DES一共有电子密码本模式（ECB）、加密分组链接模式（CBC）、加密反馈模式（CFB）和输出反馈模式（OFB）四种模式，

PKCS5Padding是填充模式，还有其它的填充模式：

然后，cipher.init（)一共有三个参数：Cipher.ENCRYPT\_MODE, key, zeroIv，zeroIv就是初始化向量。

工作模式、填充模式、初始化向量这三种因素一个都不能少。否则，如果你不指定的话，那么就要程序就要调用默认实现。

结果就是把v0构造成真正使用的密钥，并被后面的函数所利用。

### 分析五——解密过程

查阅了有关dofinal函数的用法和定义，理解以下就是对密钥和密文/明文进行解密/加密操作，具体是什么操作，要看前面cipher的创造。下面是有关dofinal的函数用法。

摘自：<https://www.169it.com/article/15568277851296016041.html>

Dofinal：

public final int dofinal(bytebuffer input,bytebuffer output)

throws shortbufferexception,illegalblocksizeexception,badpaddingexception

按单部分操作加密或解密数据，或者结束一个多部分操作。数据将被加密或解密（具体取决于此 cipher 的初始化方式）。

处理从 input.position() 开始的所有 input.remaining() 个字节。结果将存储在输出缓冲区中。返回时，输入缓冲区的位置将等于其限制；其限制不变。输出缓冲区的位置将前移 n，其中 n 为此方法返回的值；输出缓冲区的限制不变。

如果 output.remaining() 个字节还不够保存该结果，则抛出 shortbufferexception。这种情况下，使用一个稍大的缓冲区再次调用。使用 getoutputsize 确定输出缓冲区应为多大。

结束时，此方法将此 cipher 对象重置为上一次调用 init 初始化得到的状态。即该对象被重置，并可用于加密或解密（具体取决于调用 init 时指定的操作模式）更多的数据。

注：如果抛出了任何异常，则再次使用此 cipher 对象前需要将其重置。

注：此方法应该是复制安全的，这意味着 input 和 output 缓冲区可引用相同的 byte 数组，并且在将结果复制到输出缓冲区时，不会覆盖任何未处理的输入数据。

参数：

input - 输入 bytebuffer

output - 输出 bytebuffer

返回：

output 中存储的字节数

作用：利用处理过的密钥，对paramArrayOfByte进行加密，并返回加密后的密文。

### 分析六——总结

因此总结下，这个函数就是利用本地的密钥，并借助一系列的函数对密钥进行处理，来生成一个新的密钥，再对第三个参数（明密文）进行加解密处理。

## 研究五——encrypt函数

Encrypt是安全函数的加密函数，根据函数的声明，和decrypt函数一模一样，所以不做无用功，直接可以照搬研究四的内容。

下面是encrypt函数的声明：

public static byte[] encrypt(String arg4, String arg5, byte[] arg6) {

SecretKeySpec v0 = new SecretKeySpec(arg5.getBytes(), "AES");

Cipher v1 = Cipher.getInstance("AES/CBC/PKCS5Padding");

v1.init(1, ((Key)v0), new IvParameterSpec(arg4.getBytes()));

return v1.doFinal(arg6);

}

不过我在搜索encrypt函数的时候还发现了两个encrypt声明，如图4.5.1和4.5.2所示。函数的native和JNI表明，他们是调用了本地的库，所以往上查阅，果然有个 System.loadLibrary("locSDK6a");的调用，如图4.5.3。然后在左侧的lib库中也找到了liblocSDK6a.so，如图4.5.4。

如果要进行下一步分析，可以利用IDA进行静态调试，不过个人感觉这和研究目标并没太大联系，所以暂时放一边。



图4.5.1

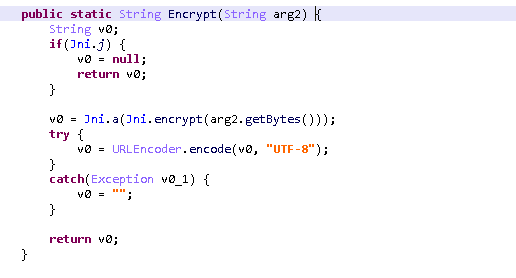


图4.5.2

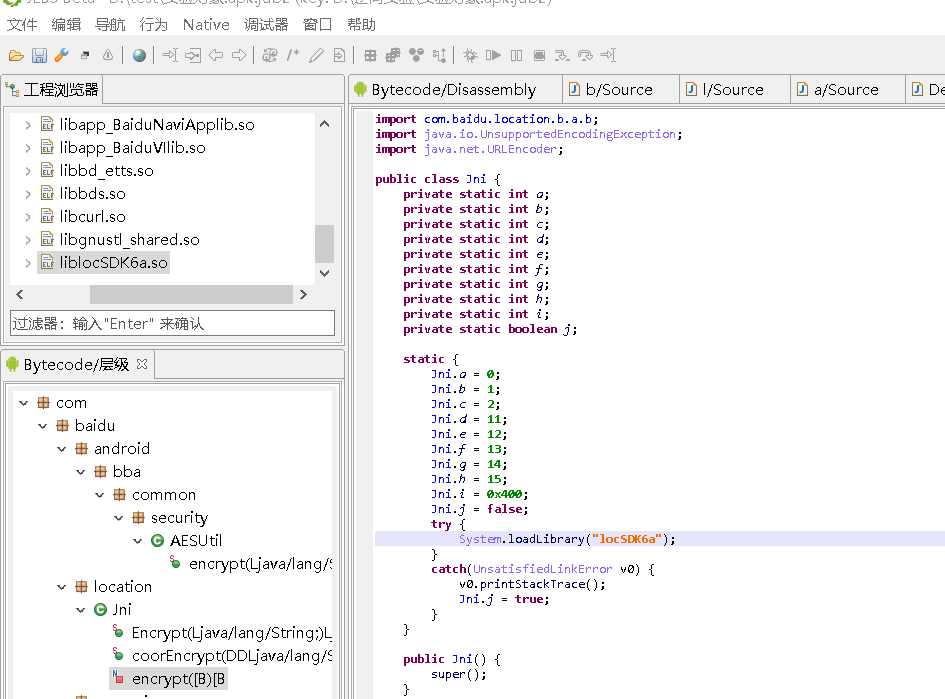


图4.5.3

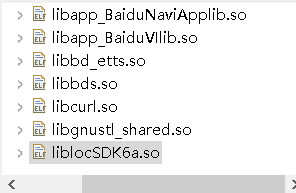


图4.5.4

## 研究六——加解密文件存储

查看调用encrypt、decrypt的函数，我先看的是encrypt函数的调用函数setExternalDeviceId，从中很容易的可以从里面看出文件存储的方式。

其中v1主要是用来指定存储位置，其所使用的函数是Environment.getExternalStorageDirectory，这个函数是用来获取手机的外部存储位置。

可参考博客：<https://blog.csdn.net/u011288271/article/details/80539456>

而v2是根据v1指定的存储位置，进行存储写入，这样子的话可以很轻松的看出来，改APK是把加密结果存储到本地当中。

同样的decrypt函数也找到调用它的函数，结构和encrypt函数相似,所以可以得到结论，那就是加解密文件存储的位置手机本地。

下面是代码：

private static void setExternalDeviceId(String arg4, String arg5) {

if(!TextUtils.isEmpty(((CharSequence)arg4))) {

StringBuilder v0 = new StringBuilder();

v0.append(arg4);

v0.append("=");

v0.append(arg5);

File v1 = new File(Environment.getExternalStorageDirectory(), "baidu/.cuid");

try {

new File(v1.getParent()).mkdirs();

FileWriter v2 = new FileWriter(v1, false);

v2.write(Base64.encode(AESUtil.encrypt("30212102dicudiab", "30212102dicudiab", v0.toString().getBytes()), "utf-8"));

v2.flush();

v2.close();

}

catch(Exception v0\_1) {

}

catch(IOException v0\_2) {

}

}

}

}

private static String getExternalDeviceId(String arg5) {

String v0;

if(TextUtils.isEmpty(((CharSequence)arg5))) {

v0 = "";

return v0;

}

v0 = "";

File v1 = new File(Environment.getExternalStorageDirectory(), "baidu/.cuid");

try {

BufferedReader v1\_4 = new BufferedReader(new FileReader(v1));

StringBuilder v2 = new StringBuilder();

while(true) {

String v3 = v1\_4.readLine();

if(v3 == null) {

break;

}

v2.append(v3);

v2.append("\r\n");

}

v1\_4.close();

String[] v1\_5 = new String(AESUtil.decrypt("30212102dicudiab", "30212102dicudiab", Base64.decode(v2.toString().getBytes()))).split("=");

if(v1\_5 == null) {

return v0;

}

if(v1\_5.length != 2) {

return v0;

}

if(!arg5.equals(v1\_5[0])) {

return v0;

}

v0 = v1\_5[1];

label\_22:

}

catch(FileNotFoundException v1\_1) {

goto label\_22;

}

catch(Exception v1\_2) {

}

catch(IOException v1\_3) {

}

return v0;

}

## 研究七——随机数生成函数

一般来说密钥的随机数获取是randomUUID这个函数，我在JEB3中搜索random，发现只有两个地方存在这个字符串。

接着我对照了下加解密函数调用图，发现l并不在其中，我基本可以确定该函数与加解密并无关系，如图4.7.1。

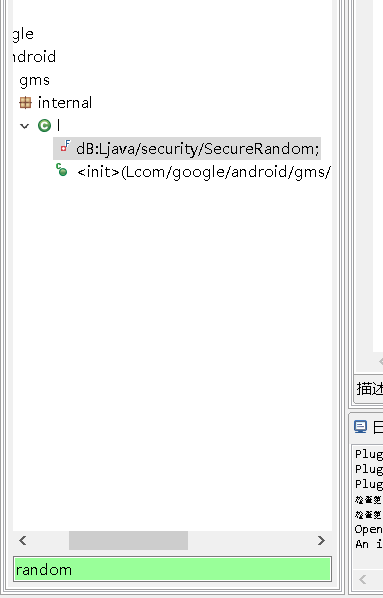


图4.7.1

## 跟踪加解密函数调用图

利用JEB3的交叉引用，对decrypt和encrypt进行跟踪调用，描绘出下面的图片。简单的来说，两个函数都有公共部分，均被getDeviceID这个函数所调用。根据这个函数的名字，感觉是用来获取设备的ID的功能。而据我了解，我所研究的APK是一个利用百度地图来跟踪其他设备的软件，主要就是防丢功能，所以感觉并没什么问题，如图4.8.1。

另外，我还发现JNI接口这块也有一个encrypt函数，不过应该不是我们的考虑范围，但还是做了个函数调用图，如图4.8.2。

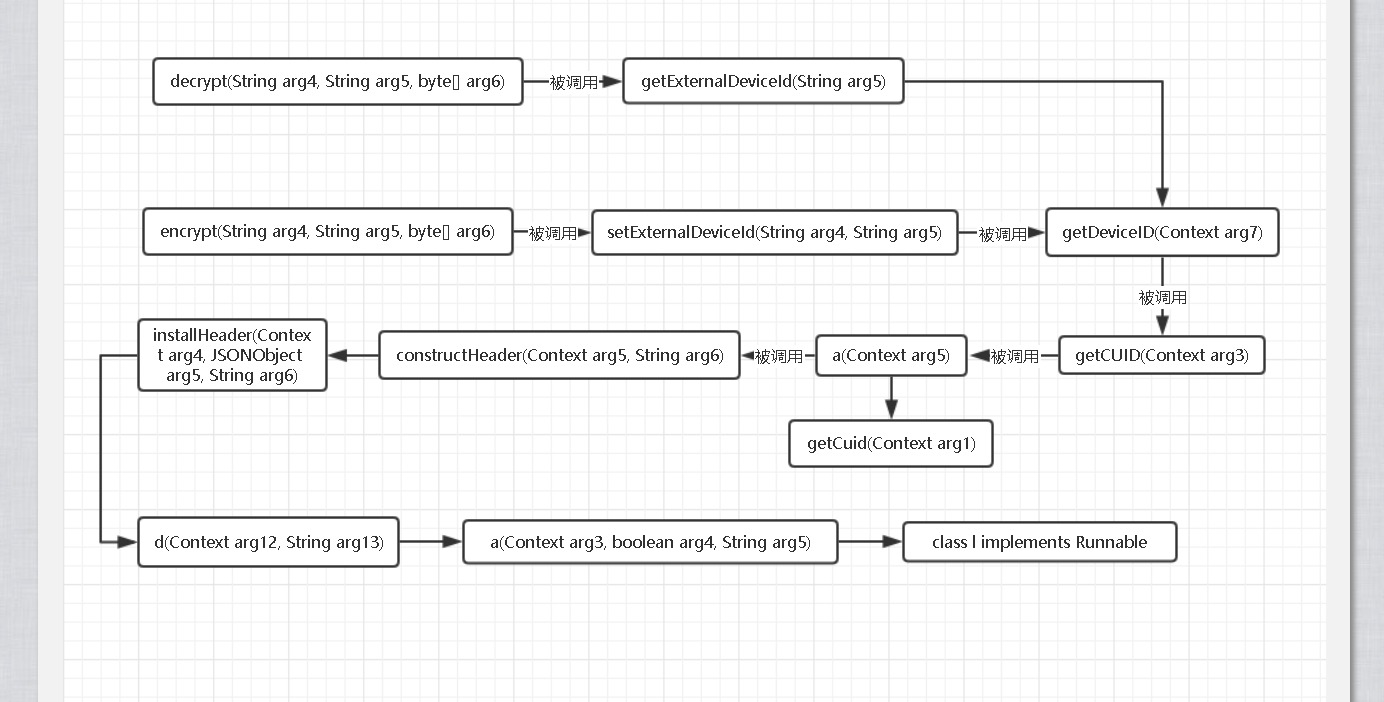


图4.8.1

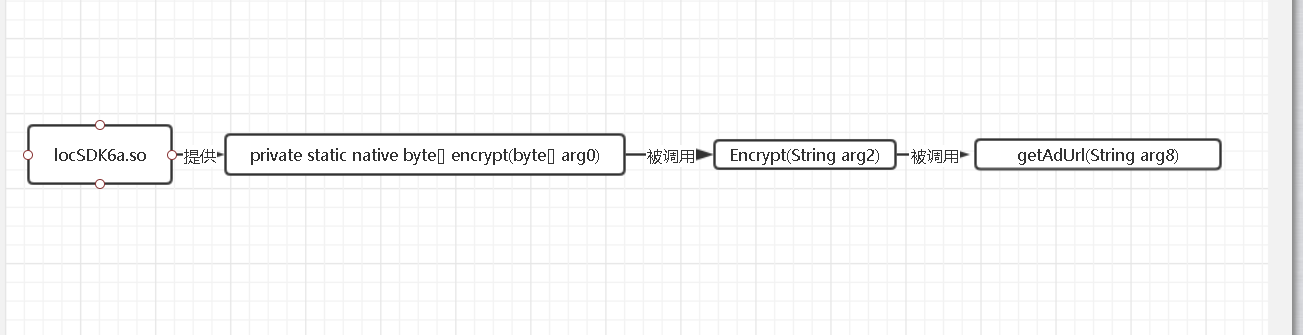


图4.8.2

# 实验结论

首先，先回答下实验目标的两个问题，然后我再总结个人的结论。

1. 分析给定的Android APK中在加密函数调用中出现的常值字符串、数组是否为密钥，如为密钥，确认被加密的数据内容是用以本地数据还是网络通信数据，即加密结果是存储在本地还是通过网络传输到服务器。

经过搜索常值字符串，找到了密钥，其密钥是30212102dicudiab。但实际所采用的的密钥，是根据该本地密钥进行处理得到的，具体分析参照4.4。

在4.6的分析当中，我们发现数据内容都是利用Environment.getExternalStorageDirectory这个函数来获取，这个函数是获取手机存储的外部存储，因此都是本地数据。

1. 分析给定APK中是否调用了随机数生成函数。如果有，生成的结果是否与密钥生成过程相关，如有，记录随机数种子的产生方式，并跟踪基于该随机数生成的密钥是否用以网络传输数据的加密。

该APK调用了随机数生成函数SecureRandom，但是并没有和密钥生成过程相关，根据4.8的跟踪加解密函数调用图可以看出来，这个函数并没有被调用图中的函数所包括，因此无关。

1. 功能结果与逆向结果的判断

该APK是一个防丢软件，通过蓝牙和其他设备进行联系，连接设备。所以个人感觉，研究的加解密过程主要是用于对设备ID的获取，因此是不需要和网络想通，这和我在加解密存储文件的存放位置方面不约而同，也恰恰证实了我的推理。

# 实验心得

本次实验主要是用JEB3、androidkille、JD进行分析，实验过程中出现过很多的问题，也有了一些自己的经验之谈，下面我陈述一下自己实验心得：

1. 安装软件需要注意的地方，软件最好安装在无中文路径的位置，比如说IDA和JEB都会因为这个问题导致无法正常运行。
2. 一般的反汇编的操作键是tab键或者F5
3. 用IDA分析lib库的时候可以先用Androidkiller加载，然后再Androidkiller的project下寻找lib库，当然也可以先在Androidkiller中记下地址，简化自己的寻找。
4. 如果在androidkiller环境下不能用JD打开的话，可以换JEB操作。至于无法打开的原因是因为找不到文件名，如果想利用androidkiller打开，要自己手动的转换文件，当然这样很麻烦，个人不推荐。
5. JEB的交叉使用功能十分的舒服，可以看到所有调用该函数的地址。

同时，个人也不知死活的去参加腾讯安全竞赛的安卓组，初赛被淘汰了，当然也收获颇多，把之前的知识巩固并加以实践，虽然最后在动态调试的过程中因为掌握不牢而失败了，但也熟悉了这类crackme.so出题方式，也可以说是收获不少，我会把比赛的分析也放在文件夹当中。