《信息安全综合实践》实验报告

实验名称： android 加解密功能实现

姓名：费扬 学号：519021910917 邮箱：2019dfff@sjtu.edu.cn 实验时长：>>180 分钟

一、实验目的

1. 理解密码技术的应用；
2. 掌握基本的 android 平台加解密功能的实现方法；
3. 掌握针对加解密功能相关实现的基本安全分析方法；
4. 探索针对加解密功能相关实现的基本安全防范措施。

二、实验内容（60 分）

序	内容	功能实现	涉及密码算法
	环境安装	运行 Test APP	--
a)	基于 library 的实现	■加/解密 □签名/验签	AES
b)	基于 Keystore 的实现	□加/解密 ■签名/验签	Hash
c)	基于自定义白盒的实现	■加/解密 □签名/验签	AES

要求：

1. a)-c)中可在加/解密和签名/验签两类任务中选一类实现即可，在上表中标注并说明所使用的密码算法；
2. 总体上加/解密和签名/验签两类任务均需实现；
3. 对 a)-c)的功能实现各给出至多 6 个截图；
4. 对 c)的功能实现还需要给出完整设计实现的思路。

A) 环境安装：

根据教程安装 Android Studio 和 Java 8，以及 Git 等。

运行 test 项目，测试前先进行 Gradle Sync 进程，创建虚拟设备或使用有线 USB 传输到自己的手机（Harmony OS），打开开发者选项，允许 USB 调试。

实现项目部署运行，在手机上可以正确运行。

主页面 UI 未做修改，原功能测试结果如下图。

```
I/System.out: 验签结果:true私钥:MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAN0Io1vx+vuitKxA3z5Jj/vXaWoLL2sUhz
I/System.out: 公钥:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDdCKNb8fr7orSsQN8+SY/712lqCy9rFIc+C6PLScódoXAEpDx6BdAN5
I/System.out: 加密后内容:z0Cyo7ifHU9bYncXywRXvPAkGm8fBmGDcUu/hedgLOnBdemnhcVvFW5AcmHpauuvPóqhSurh+NjQwpypqFZRtnNR
I/System.out: 解密后内容:testdata123456
D/MainActivity: ECB模式
D/MainActivity: 密文:[-47, 14, 47, 109, 0, -27, 17, -110, 97, 126, 47, -67, 20, 122, 103, -63]
D/MainActivity: 校验:true
D/MainActivity: CBC模式
D/MainActivity: 密文:[-93, 81, 51, 122, -6, -119, 113, 2, -35, 14, 122, -48, 73, -70, -58, 34]
D/MainActivity: 校验:true
```

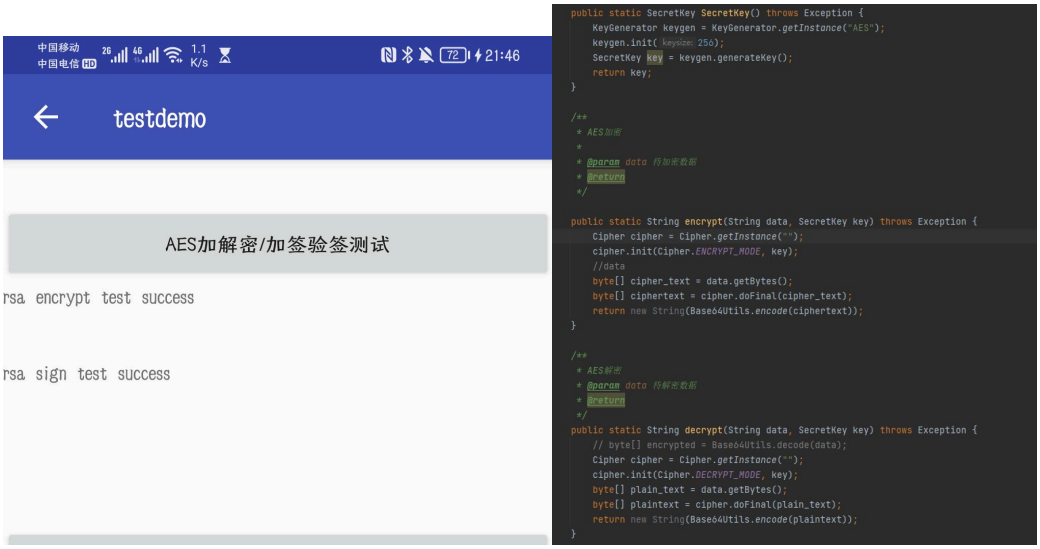
B) 基于 Lib 库实现：

本实验基于 java 库函数实现，主要库依赖如下：

- Javax.crypto.Cipher
- Security.Signature
- Security.KeyPairGenerator
- Security.PublicKey
- Security.PrivateKey

原文件中的算法实现使用 RSA 算法，其具体实现函数在 RSA.java 文件中，对其加密算法进行修改，使用对称密钥算法 AES，实现加密解密。具体实现在 AES.java 文件中。

实现效果如图：




```
2022-04-09 21:45:56.791 32574-32574/com.alipay.keymaster I/System.out: 私钥:MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAPBWx1pZpQ2pTBcuVVoFvm1c6HB5Ajsl9Yv5J1tevgz4xXW4RaSi
2022-04-09 21:45:56.791 32574-32574/com.alipay.keymaster I/System.out: 公钥:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDwVsdaWaUNqUwXLlVaBb5tXOhweQI7JfWL+SdbXr4M+MV1uEWktZRL7FspmjJG
2022-04-09 21:45:56.795 32574-32574/com.alipay.keymaster I/System.out: 加密后内容:ByJbdY6zB1GpyL9Dd+NT73VBOhb1dRBCiWkCmUh8/ehG2FF2maO6pQjwDuen7Li8d4gZhvDF3wmKMnR5mQhpfLkCejCL59g
2022-04-09 21:45:56.798 32574-32574/com.alipay.keymaster I/System.out: 解密后内容:shjd090701
```

```
D/goldfish-address-space: allocate: ioctl allocate returned offset 0x3f3ffe000 size 0x2000
D/HostConnection: HostComposition ext ANDROID_EMU_CHECKSUM_HELPER_v1 ANDROID_EMU_native_sync_v2 ANDROID_EMU_native_sync_v3 ANDROID_EMU_native_sync
I/OpenGLRenderer: Davey! duration=1081ms; Flags=1, IntendedVsync=3193947753371, Vsync=3193964420037, OldestInputEvent=9223372036854775807, NewestI
I/Choreographer: Skipped 86 frames!  The application may be doing too much work on its main thread.
I/OpenGLRenderer: Davey! duration=1488ms; Flags=0, IntendedVsync=3193981090845, Vsync=3195414424121, OldestInputEvent=9223372036854775807, NewestI
I/System.out: 私钥:MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAMRhm8Z3fH043dA5jE25YfpVw1i7mX3LwmAfJu3Lx8Q+6OBVgFwhZcSkxD5BH7wHór9EBFpUAXJWLUdg
I/System.out: 公钥:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEYZvGd3x9ON3QOYxNuWHóVcNYu5l9y8JgHybty8fEPutAVYBcIWXEpMQ+QR+8B+q/RARaVAFyVi1HaKnfaAKmOlTP
I/System.out: 加密后内容:uqdV5PRUJgUVJHY3ur+JP/90Yóf1wBbCx/FPZAumUr359Sj5epsc3H80wkFVjLqiz/ERO4Y5wJUe8pHtEPEmlJsAhWqfKvzmhXfepPHBCdDUHZNPsAGswhe7ACv
I/System.out: 解密后内容:shjd090701
```

对代码的解释：

不同于 RSA 的非对称加密会有公钥私钥，AES 只需要使用一个 SecretKey 函数，使用 Keygenerator 产生调用一个 key 作为密钥；

加解密函数模式可以用 AES/GCM/NoPadding，也可以不填。使用 Cipher 函数加载实例，选定加密或解密模式，参数传输格式为 String→byte[]→String，在输出时需要注意 Base64 编码格式转换。

在 MainActivity 中编写 testAES()模块，仿照原文件中的 testRSA()。具体过程为生成密钥，AES 加密，AES 解密，测试对比。注意边界条件的判断即可。同时需要注意将 button 中点击事件修改。

C) 基于硬件库 KeyStore 实现：

本实验基于 KeyStoreUtil.java 和 KeymasterActivity.java 两个文件实现，主要操作是在加签验签的过程中加入 hash 摘要算法(DigestMessage 函数)，将其应用在 sign 和 verify 两个函数中，使用的算法为 Hash-256。

实现效果如图：


```
I/System.out: 验签结果:true私钥:MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAN0Io1vx+vuitKxA3z5Jj/vXaWoLL2sUhz
I/System.out: 公钥:MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDdCKNb8fr7orSsQN8+SY/712lqCy9rFIc+CóPLScódoXAEpDxóBdAN5
I/System.out: 加密后内容:z0Cyo7ifHU9bYncXywRXvPAkGm8fBmGDcUu/hedgLOnBdemnhcVvFW5AcmHpauuvPóqhSurh+NjQwpypqFZRtnNR
I/System.out: 解密后内容:testdata123456
D/MainActivity: ECB模式
D/MainActivity: 密文:[-47, 14, 47, 109, 0, -27, 17, -110, 97, 126, 47, -67, 20, 122, 103, -63]
D/MainActivity: 校验:true
D/MainActivity: CBC模式
D/MainActivity: 密文:[-93, 81, 51, 122, -6, -119, 113, 2, -35, 14, 122, -48, 73, -70, -58, 34]
D/MainActivity: 校验:true
```

对名为 alias1 的明文内容进行加签验签 结果 still match

作为对比，在加签时候加入摘要步骤，而验签时候不加入摘要的步骤，则在 verify 时两个结果不 match：

摘要后加签再验签 结果 not match

```
/**
 * 签名前压缩算法
 * @param data 原始数据
 * */

public byte[] Digestmessage (byte[] data) throws NoSuchAlgorithmException {
    byte[] message = data;
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    byte[] digest = md.digest(message);
    return digest;

}

/**
 * 对数据进行签名
 *
 * @param data
 * @param alias
 * */
```

对代码的解释：

基于硬件库实现的加解密算法和加签验签算法受限，可以参考一些网站（加密 | Android 开发者 | Android Developers）对其中支持的算法进行了解学习。在生成签名时，使用的算法是 SHA256withRSA，这也是由 Signature 支持的。

D) 基于自定义白盒的实现

这项任务的核心在于不调用 java 的库，根据底层原理实现一个算法，进行加密解密/加签验签。在这里我选择的是 AES 加密算法。

没有接触过安卓开发也没有系统的学习过 java 的我在起初遇到了不小的困难，查阅了大量资料，从算法原理到 WBAES 的构造实现，密码学的实践实现资料较少，本项任务的实现花费了大量时间（多于一周）。最终实现了 AES 的两种实现思路，分别在 mytest.java 和 mytest_aes.java 中。

下面介绍实现的思路：

- **Mytest.java：**

Mytest.java 中通过定义 x2time、x3time、x9time 等函数实现了前几轮和最后一轮操作的分割，用硬件电路的思路实现了按位输出，支持 String 和 byte[]输入输出。S 盒和逆 S 盒用于 Encrypt 和 Decrypt 中密钥的生成。

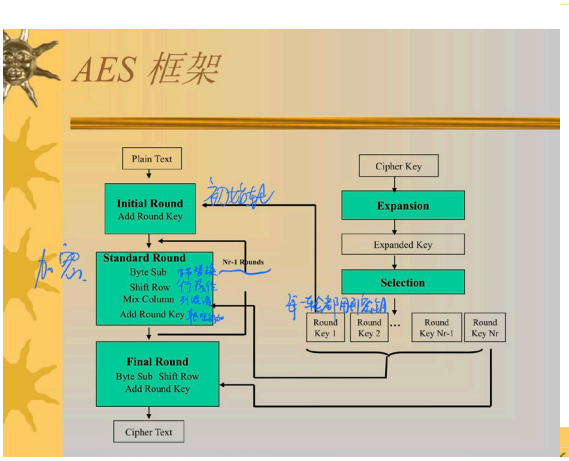
实现中注意有一个 byte[]到 char 的类型转换。

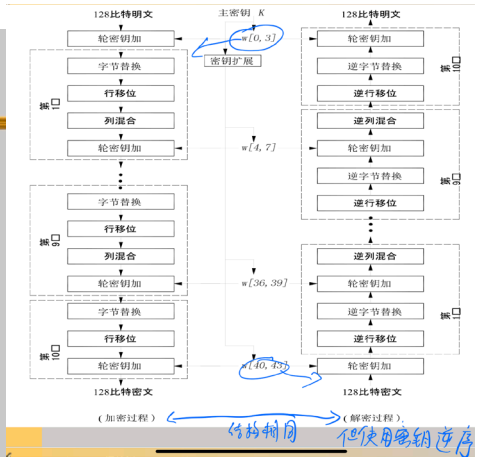
输出结果如下图：

Mytest.java 实现效果

- **Mytest_aes.java**

Mytest_aes.java 中以 AES 的基本原理一步一步实现了 AES 加解密，具体实现中包括函数 ShiftRows、SubBytes、KeyExpansion、FFMul、MixColumns 等，实现了 Encrypt 和 Decrypt，Char[]类型输入输出。S 盒和逆 S 盒用于 Encrypt 和 Decrypt 中密钥的生成。





参考：现代密码学课程 PPT

相应的定义用于解密的函数，包括 InvSubBytes、InvShiftRows、InvKeyExpansions 等。Sbox 和 Invbox 定义用于加解密中的非线性部分。

Cipher 和 InvCipher 两个函数实现了加密解密的函数封装调用，在 Mainactivity 中定义一个调用示例，连接 button。

```
    System.out.print("明文: ");
    for(int i = 0; i < 16; i++){
        System.out.print(mw[i]);
        // println ("%0X ",mw[i]);
    }

    System.out.print("密钥: ");
    for(int i = 0; i < 16; i++){
        System.out.print(key[i]);
    }
    mytest_aes.Cipher(mw);
    System.out.print("加密后: ");
    //printf("%s",mw);
    for(int i = 0; i < 16; i++){
        System.out.print(mw[i]);
    }

    mytest_aes.InvCipher(mw);
    System.out.print("解密后: ");
    for(int i = 0; i < 16; i++){
        System.out.print(mw[i]);
    }
```

```
//获取测试结果更新UI
boolean ecb_result = Arrays.equals(backup, mw);
TextView ecb_text = findViewById(R.id.txt_sm4_ecb);
String ecb_showtext = "aes test fail";
if (ecb_result) {
    ecb_showtext = "aes test success";
}
ecb_text.setText(ecb_showtext.toCharArray(), start: 0, ecb_showtext.length());
```

输出结果如下图:


```
2022-04-09 22:24:49.771 5783-5783/com.alipay.keymaster V/AudioManager: querySoundEffectsEnabled...
2022-04-09 22:24:49.772 5783-5783/com.alipay.keymaster I/System.out: Plaintext:
2022-04-09 22:24:49.773 5783-5783/com.alipay.keymaster I/System.out:   Hex:74 65 73 74 20 61 65 73 20 68 61 68 61 68 68 61
2022-04-09 22:24:49.773 5783-5783/com.alipay.keymaster I/System.out:   CH:test aes hahahha
2022-04-09 22:24:49.773 5783-5783/com.alipay.keymaster I/System.out: Ciphertext:
2022-04-09 22:24:49.774 5783-5783/com.alipay.keymaster I/System.out:   Hex:a7 65 db 91 74 fa ba 1b 58 a5 c7 4a 30 20 df 8f
```

原计划在实现了 AES 之后，按照 Chow et al 的设计思路与方法构建 WBAES 算法，即白盒 AES，但由于设计复杂，实现设计的文件繁多，背景知识有限未能实现白盒 AES，未来考虑设计 Whitebox-base56。

设计参考: ph4r05/Whitebox-crypto-AES-java: Whitebox AES implementation in Java. Chow scheme. (github.com)

三、分析和思考（40 分）

1. 在要求 a 的实现中，密钥在持久化状态下存储在哪里？在何种情况下它们可能被（恶意地）未授权访问？（10 分）

持久状态： *对象经过 Session 持久化操作，缓存中存在这个对象的数据为持久状态并且数据库中存在这个对象对应的数据为持久状态。*

1、密钥直接明文存在私有目录的 sharedprefs 文件中，程序很容易被 IDA 逆向成 java 代码而文件很容易被查看。总体而言密钥存储在本地或是 java 层中，安全性不高。

2、使用 root 权限机器即可导出查看，本地数据和敏感信息可以被逆向获取。

2. 在要求 b 的实现中，密钥在其使用过程中会出现在内存中的什么位置？它们可能会被恶意访问吗？如果是，在何种情况下发生？（10 分）

存储密钥在密钥库的数据库中，默认情况下此数据库存储在名为 keystore 的文件中。Android 提供的这个 KeyStore 最大的作用就是不需要开发者去维护这个密匙的存储问题，相比起存储在用户的数据空间或者是外部存储器都更加安全。注意的是这个密匙随着用户清除数据或者卸载应用都会被清除掉。

利用 Android KeyStore System，用户可以在容器中存储加密密钥，从而提高从设备中提取密钥的难度。在密钥进入密钥库后，可以将它们用于加密操作，而

密钥材料仍不可导出。此外，它提供了密钥使用的时间和方式限制措施，例如要求进行用户身份验证才能使用密钥，或者限制为只能在某些加密模式中使用。

而在 AndroidKeystore 中，每个 APP 有自己的密钥命名空间，APP1 中的 TestKey 和 APP2 中的 TestKey 是不同的。

在 /data/misc/keystore/user_x/ 目录下有各个 APP 的密钥条目，如 10174_USRPKEY_step_info，其中前面的 10174 是 APP 的 UID，step_info 是密钥的别名(alias)。

在 root 的手机上，通过文件的复制，更改密钥条目文件的 UID，就可以盗用其他 APP 的密钥条目。

3. 在要求 c 的实现中，密钥存在被其它 API 调用进行数字签名的可能性吗？如果有，如何做到？你有办法能阻止这种调用，或在验证签名时发现这种调用吗？（15 分）

在白盒加密算法中，密钥会被直接调用，从而被数字签名，对其进行攻击破坏。对抗加密不一定非得逆向出算法，可以通过黑盒直接调用的方式。拿到能加密签名后的结果即可。一般通过 Xposed，frida 反射签名函数搭建 RPC 服务亦或者通过 unicorn 模拟器执行。

以前黑盒调用一般由 AndServer+Service 打造 Android 服务器实现 so 文件调用；一般需要 xposed hook 调用函数或者 dlopen 打开关键 so 并调用特定函数调用，且需要使用 Andserver 搭建 server 做加签名的服务。而基于长链接和代码注入的 Android private API 暴露框架 sekiro 已搭建好基础的暴露服务我们仅需几行代码就可轻松实现一个加密服务。

一个例子：zhaoboy9692/flask_frida_rpc: 使用 flask 和 frida 完成抖音、饿了么 app 的算法调用 pojie。 (github.com)

防范：黑盒调用需要解决的问题包括：1、SO 装载 2、内存映射 3、栈分配 4、API 调用 5、返回值读取。

故我们可以避免使用纯算法，增加核心算法的上下文依赖，防止黑盒调用。

4. 据你所知，有哪些方法可以提升密钥存储与使用过程的安全性，它们如何起作用？（5 分）

一些基本操作：

更新：当密钥已经达到其使用期限或者密钥已经被破解时，密码系统需要有密钥更新机制来重新产生新的密钥。

备份：密钥丢失将导致密文数据无法解密，这样便造成了数据的丢失。应依据具体场景，来评估是否需要对密钥提供备份与恢复机制。

销毁：不再使用的密钥应当立即删除。

除此之外：

通过中间密钥 KEK 对 DEK 再次进行加密：由 KEK 来保证 DEK 的机密性。但如此则陷入一个循环，因为 KEK 本身也需要保护。实际应用系统中，往往存在硬件唯一密钥 HUK，用于保护中间密钥 KEK；KEK 再直接参与对 DEK 的保护，或者先保护另外一个 KEK’之后再由 KEK’保护 DEK。硬件根密钥只能在硬件中访问，这样就保证了根密钥的安全性，进而层级的保证 KEK 以及 DEK 的安全性。

用 WBC 保护软件应用程序中的加密密钥，也就是白盒加密：核心思想是把秘钥隐藏起来，加密执行过程中，内存中不会出现秘钥的值. 现在通用的技术是查找表技术，即把秘钥隐藏在查找表中。

另外有两个思考题，针对源代码里的几个点，已回答在代码注释内，分别在 RSA.java 和 MainActivity.java 文件里。

四、实验总结（收获和心得）

本次实验是信息安全综合实践联合蚂蚁安全开展的一次密码学的工业级应用实验，我感觉真的收获颇丰，安卓开发，java，密码学知识应用，rsa，aes，sm4……我从中收获很多。

本次的三个实验分别从不同的设计角度来实例密码学，从最基本的调库，到硬件实现，到原理实现或是白盒加密，能够真正的对比密码学知识在实际行业中效果。此外，我还进一步学习了 java 编程和 Android Studio 开发环境，第一次尝试在手机上部署安卓 app 调试并开发。这种体验能够激发我学习信息安全领域知识的热情，让我在实践中收获进步。

但本次实验也有不少困难，我花费了很久在白盒加解密研究上，从背景知识，阅读 Chow et al 白盒架构论文，构建 AES 底层代码，到最后的代码撰写等。虽然最后的结果是并没有完全实现，但我仍然从中体会到了发现 bug，定位 bug，调试 bug，解决 bug 的短暂快乐（虽然很掉头发）。希望后续能进一步把这个实验做完善。

感谢孟老师和蚂蚁安全的工程师们对本次课程的教学和课外支持！

本次实验的代码仓库：

2020dfff/android_dev: Assignment for IS497, lab3 Keymaster-test. Reference: Alibaba (github.com)

实现中的一些参考：

1.[原创]利用 root 权限盗用 AndroidKeystore 中其他 APP 的密钥-Android 安全-看雪论坛-安全社区|安全招聘|bbs.pediy.com

2.Android 密钥库系统 | Android 开发者 | Android Developers (google.cn)

3. 白盒加密 (whiteboxcrypto.com)

4. ph4r05/Whitebox-crypto-AES-java: Whitebox AES implementation in Java. Chow scheme. (github.com)