合肥工業大學

信息安全实验报告

实 验 加 密 解 密

学 生 丁 瑞

学 2016217676

专 业 物 联 网 16-01 班

2019 年 12 月 30 日

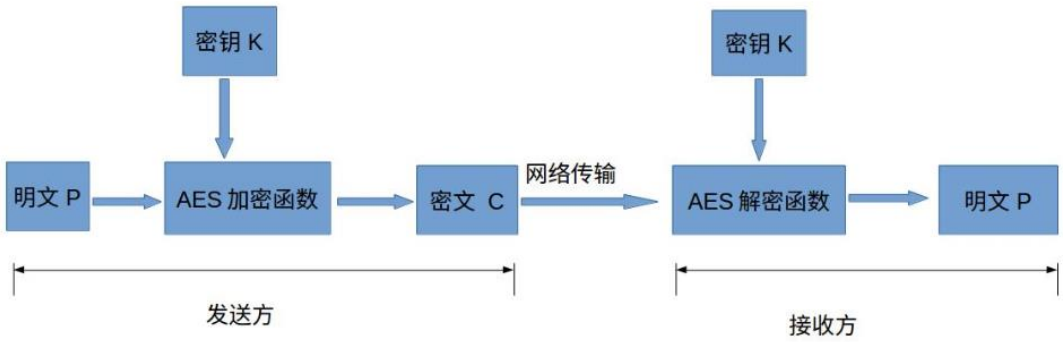
一、 实验目的

加深对加密算法原理的了解，基础算法的实现和应用

二、 实验原理

2.1 AES 加密算法

高级加密标准(AES,Advanced Encryption Standard)为最常见的对称加密算法(微信小程序加密传输就是用这个加密算法的)。对称加密算法也就是加密和解密用相同的密钥，具体的加密流程如下图：



2.2 RSA 加密算法

RSA 公开密钥密码体制。所谓的公开密钥密码体制就是使用不同的加密密钥与解密密钥，是一种“由已知加密密钥推导出解密密钥在计算上是不可行的”密码体制。

在公开密钥密码体制中，加密密钥（即公开密钥）PK 是公开信息，而解密密钥（即秘密密钥）SK 是需要保密的。加密算法 E 和解密算法 D 也都是公开的。

虽然解密密钥 SK 是由公开密钥 PK 决定的，由于无法计算出大数 n 的欧拉函数 phi(N)，所以不能根据 PK 计算出 SK。

正是基于这种理论，1978 年出现了著名的 RSA 算法，它通常是先生成一对 RSA 密钥，其中之一是保密密钥，由用户保存；另一个为公开密钥，可对外公开，甚至可在网络服务器中注册。为提高保密强度，RSA 密钥至少为 500 位长，一般推荐使用 1024 位。这就使加密的计算量很大。为减少计算量，在传送信息时，常采用传统加密方法与公开密钥加密方法相结合的方式，即信息采用改进的 DES 或 IDEA 密钥加密，然后使用 RSA 密钥加密对话密钥和信息摘要。对方收到信息后，用不同的密钥解密并可核对信息摘要。

2.3 SHA256 加密算法

SHA256 是一个哈希函数。

哈希函数，又称散列算法，是一种从任何一种数据中创建小的数字"指纹"的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（或哈希值）的指纹。散列值通常用一个短的随机字母和数字组成的字符串来代表。

对于任意长度的消息，SHA256 都会产生一个 256bit 长的哈希值，称作消息摘要。这个摘要相当于是个长度为 32 个字节的数组，通常用一个长度为 64 的十六进制字符串来表示。

三、 实验步骤

3.1 AES 对文本加解密

AES 加密部分代码

```java
public static byte[] encrypt(byte[] data, byte[] key) {
    if (key.length != 16) {
        throw new RuntimeException("Invalid AES key length (must be 16 byte
    }
    try {
        SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
        byte[] enCodeFormat = secretKey.getEncoded();
        SecretKeySpec seckey = new SecretKeySpec(enCodeFormat, "AES");
        // 创建密码器
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        byte[] iv = new byte[cipher.getBlockSize()];

        SecureRandom secureRandom = new SecureRandom();
        secureRandom.setSeed(key);
        secureRandom.nextBytes(iv);
        IvParameterSpec niv = new IvParameterSpec(iv);
        cipher.init(Cipher.ENCRYPT_MODE, seckey, niv);// 初始化
        print_bytes(iv);
        byte[] result = cipher.doFinal(data);
        return result; // 加密
    } catch (Exception e) {
        e.printStackTrace();
```

AES 解密部分代码

```java
public static byte[] decrypt(byte[] data, byte[] key) {
    if (key.length != 16) {
        throw new RuntimeException("Invalid AES key length (must be 16 byt
    }
    try {
        SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
        byte[] enCodeFormat = secretKey.getEncoded();
        SecretKeySpec seckey = new SecretKeySpec(enCodeFormat, "AES");
        // 创建密码器
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        byte[] iv = new byte[cipher.getBlockSize()];

        SecureRandom secureRandom = new SecureRandom();
        secureRandom.setSeed(key);
        secureRandom.nextBytes(iv);
        IvParameterSpec niv = new IvParameterSpec(iv);
        cipher.init(Cipher.DECRYPT_MODE, seckey, niv);// 初始化
        print_bytes(iv);
        byte[] result = cipher.doFinal(data);
        return result; // 解密
    } catch (Exception e) {
```

AES 测试部分代码

```java
public static void main(String[] args) throws UnsupportedEncodingException
    // TODO Auto-generated method stub
    String string = "guosujietest12341234";
    String key = "1234123412341234";
    byte[] result = encrypt(string.getBytes("UTF-8"), key.getBytes("UTF-8")
    System.out.println("加密:"+bytes_String16(result));
    byte[] dresult = decrypt(result, key.getBytes("UTF-8"));
    System.out.println("解密:");
    for (int i = 0; i < dresult.length; i++) {
        System.out.print((char) dresult[i]);
    }
}
```

加密解密结果

```
加密:7ACD0D145B2E5FA3EC99DCDDB20D22CD5499E2A3B8F7C8F9E9D4EBDCA003AA99
解密:
testtesttesttest
```

1. 样例代码中的密钥字符串为硬编码的 **16** 字符的字符串，即 **128** 位，请对密钥字符串进行替换并进行测试。如果其小于 **16** 字符会使加解密程序出错，尝试使用一种方法解决对小于 **16** 字符的密钥字符串进行处理,使其能应用在加解密程序中。对实验结果进行截图。

答:

AES 算法加密时会将明文拆分成 128bit 的数据块，分别进行加密，也就是如果明文长度非 128bit 的整数倍则必出现不满 128bit 的数据块，填充就是在此时起作用的，填充方式如下。

NoPadding:不做任何填充，但是要求明文必须是 16 字节的整数倍。

PKCS5Padding（默认):如果明文块少于 16 个字节（128bit），在明文块末尾补足相应数量的字符，且每个字节的值等于缺少的字符数。

ISO10126Padding：如果明文块少于 16 个字节（128bit），在明文块末尾补足相应数量的字节，最后一个字符值等于缺少的字符数，其他字符填充随机数。

```
加密:08E45AB0362BB83F34613B8F11CA5E65
解密:
test
```

2. 尝试更改 **AES** 加解密工具中的 **iv**，当加密时的 **iv** 与解密时的 **iv** 不一致时会发生什么？对实验结果进行截图。

CBC 的解密则也是从左往右看，但是加密时 IV 在解密时候，只会用于对第

一个块进行解密，其他块的解密则是使用上一块的加密二进制作为 IV 进行解密操作。

加密时，用 iv 和 key 去加密第一个块，然后用第一个块的加密数据作为下一个块的 iv，依次迭代。解密时，用 n-1 个块的加密数据作为 iv 和 key 去解密第 n 个块（n>1），只有第一个块才会用加密时的 iv 去解密第一个块。按照这样的逻辑来看，那么如果解密时，使用了 iv 错误，出问题的数据应该只有第一个块。

通过上面的分析就能知道，加密的时候，iv 会影响所有数据的加密结果，而解密时，iv 只会影响第一个加密块的解密结果，其他块的解密可以直接通过分隔加密数据获取正确是 N-1 块的 IV。

Iv 不一致时 只对解密第一块有影响，如下图所示：

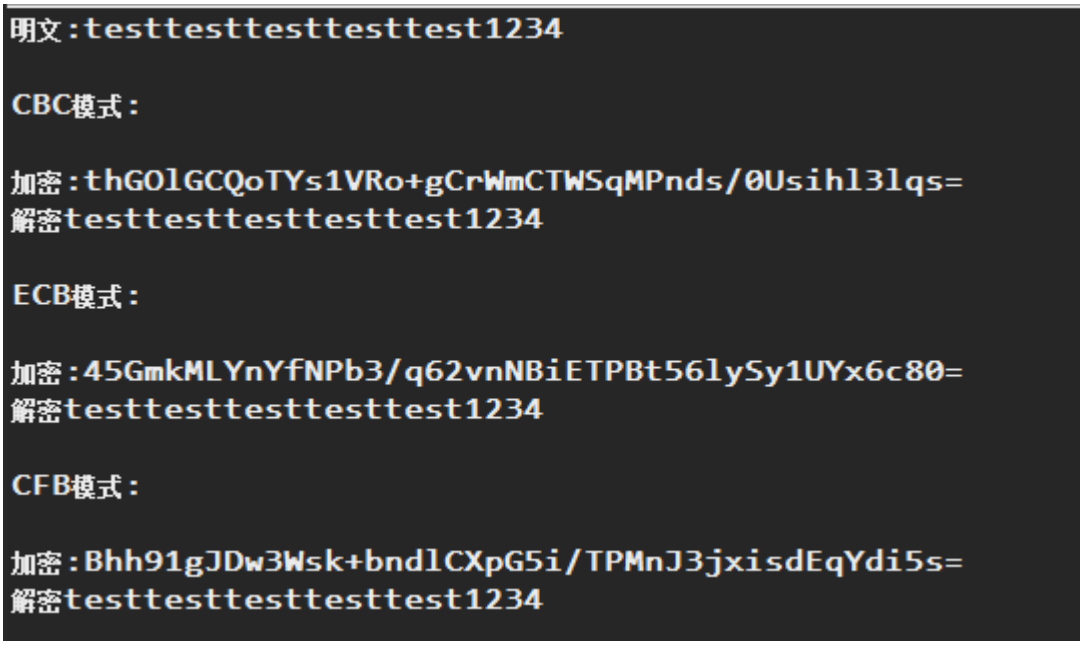
```
 iv
618885-7310-10290-33-8851-75-65105-122-6768
加密:7ACD0D145B2E5FA3EC99DCDDB20D22CDAA9352ADA77BEAD185DDA009E0E55FE9

 iv
618885-7310-10290-33-8851-75-65105-122-6768
解密:
testtesttesttesttest1234
```

```
 iv
618885-7310-10290-33-8851-75-65105-122-6768
加密:7ACD0D145B2E5FA3EC99DCDDB20D22CDAA9352ADA77BEAD185DDA009E0E55FE9

 iv
618885-7310-10290-33-8851-75-65105-122-6768
解密:
testtesttesttesttest1234
```

3. 使用几个其他的 **AES** 加解密模式**(**如 **ECB**、**CFB** 模式**)**进行操作，对实验结果进行截图。



ECB 模式（电子密码本模式：Electronic codebook）

- ECB 是最简单的块密码加密模式，加密前根据加密块大小（如 AES 为 128 位）分成若干块，之后将每块使用相同的密钥单独加密，解密同理。
- ECB 模式由于每块数据的加密是独立的因此加密和解密都可以并行计算。

- ECB 模式最大的缺点是相同的明文块会被加密成相同的密文块，这种方法在某些环境下不能提供严格的数据保密性。

1.2.2 CBC 模式（密码分组链接：Cipher-block chaining）

- CBC 模式对于每个待加密的密码块在加密前会先与前一个密码块的密文异或然后再用加密器加密。
- 第一个明文块与一个叫初始化向量（偏移量 iv）的数据块异或。

- CBC 模式相比 ECB 有更高的保密性，但由于对每个数据块的加密依赖与前一个数据块的加密所以加密无法并行。
- 与 ECB 一样在加密前需要对数据进行 **填充**，不是很适合对流数据进行加密。

1.2.3 CFB 模式（密文反馈：Cipher feedback）

- 与 ECB 和 CBC 模式只能够加密块数据不同，CFB 能够将块密文（Block Cipher）转换为流密文（Stream Cipher）。

- CFB 模式非常适合对流数据进行加密，解密可以并行计算。

3.2 AES 对图片加解密

对于图片的加密，有两种实现方式。众所周知，一张图片由若干个像素点组成，一个像素点具有 RGB 值，因此一种可行的思路为读取图片的像素点信息，对其进行加密。第二种方法为之间通过 IO 操作读取文件的字节流进行加密。

1.针对像素值加解密

（1） 读取图片文件

```
File file = new File("test.png");
BufferedImage bi = null;
try {
bi = ImageIO.read(file);
} catch (IOException e) {
e.printStackTrace();
```

（2） 获得图片 width 与 height，扫描获得像素值，分解成为 r、g、b 三个值。

```
int width = bi.getWidth();
int height = bi.getHeight();
int minX = bi.getMinX();
int minY = bi.getMinY();
for(int y = minY; y < height; y++) {
for(int x = minX; x < width; x++) {
int pixel = bi.getRGB(x, y);
int r = (pixel & 0xff0000) >> 16;
int g = (pixel & 0xff00) >> 8;
int b= (pixel & 0xff); }
```

（3） 将获得的 r、g、b 三个值分别存入不同的字节数组中，利用之间的 AES 加密工具进行加密。

（4） 加密完成后获得的新的字节数组写入新的图片对象，并存储。

```
BufferedImage resultimage=new BufferedImage(width, height,bi.getType(
int count=0;
for(int y = minY; y < height; y++) {
for(int x = minX; x < width; x++) {
int red=r_enresult[count];
int green=g_enresult[count];
int blue=b_enresult[count];
int nrgb=255<<24|red<<16|green<<8|blue;
resultimage.setRGB(x, y, nrgb);
count++;
}
//System.out.println();
}
File files =new File("D://test.jpg");
try {
ImageIO.write(resultimage,"jpg", files);
} catch (IOException e) {
```

2. 直接通过 IO 读取文件到字节数组中，并加密。

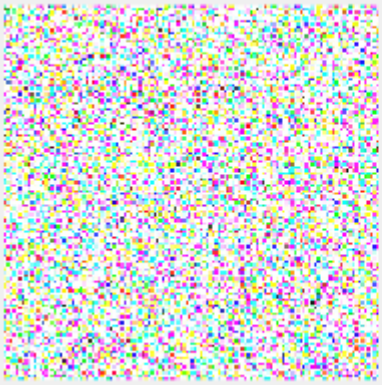
读取文件字节数组的函数为

```
byte[] getFileBytes(String file) {
    try {
        File f = new File(file);
        int length = (int) f.length();
        byte[] data = new byte[length];
        new FileInputStream(f).read(data);
```

请根据上述内容完成两种方式的图片加解密操作，并记录

实验结果，比较二者不同。

加密后



解密后

两者的不同点在于：

方法一以像素点的形式加密，最终的加密结果会成为一张新的图片，而方法二一 IO 流进行加密，最终加密结果会变成一串字符串。

3.3 RSA 加解密操作

RSA 加密算法，是世界上第一个非对称加密算法，也是数论的第一个实际应用。它的算法如下：

1.找两个非常大的质数 p 和 q（通常 p 和 q 都有 155 十进制位或都有 512 十进制位）并计算 n=pq，k=(p-1)(q-1)。

2.将明文编码成整数 M，保证 M 不小于 0 但是小于 n。

3.任取一个整数 e，保证 e 和 k 互质，而且 e 不小于 0 但是小于 k。加密钥匙（称作公钥）是(e, n)。

4.找到一个整数 d，使得 ed 除以 k 的余数是 1（只要 e 和 n 满足上面条件，d 肯定存在）。解密钥匙（称作密钥）是(d, n)。 加密过程： 加密后的编码 C 等于 M 的 e 次方除以 n 所得的余数。 解密过程： 解密后的编码 N 等于 C 的 d 次方除以 n 所得的余数。 只要 e、d 和 n 满足上面给定的条件。M 等于 N。

将提供的 RSA.java、Base64.java、ConvertUtils.java 以及 Digest.java 复制到工程文件夹下，修改相应包名为自己工程的包名。在主工程中引入复制进来的 RSA，其提供了加解密与创建密钥对的函数。首先创建密钥对，获得公钥与私钥，完成加解密操作。

编写两个实体 A 与 B，A 与 B 互相知道对方的公钥并持有自身的私钥，使得 A 与 B 能够安全的与对方交互数据(即发送方使用接收方公钥加密数据，接收方使用自身私钥解密数据)。完成实验并记录实验结果。

定义实体类：

```java
package net_secutriy;
import java.util.Map;

public class Test_A {
    private static Map<String, String> map;

    static {
        try {
            map = RSA.generateKeyPair();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    /**
     * 使用对方公钥进行加密
     * @param data
     * @return
     * @throws Exception
     */
    public static String AEncrypt(String data,String pub) throws Exception {
        return RSA.encrypt(data, pub);
    }

    /**
     * 获取本对象的公钥
     * @return
     * @throws Exception
     */
    public static String APublicKey() throws Exception {
        return map.get("publicKey");
    }

    /**
     * 使用自身私钥进行解密
     * @param en_result
     * @return
     * @throws Exception
     */
    public static String ADecrypt(String en_result) throws Exception {
        String pri = map.get("privateKey");
        return RSA.decrypt(en_result, pri);
    }

}
```

调用示例：

```java
public static void main(String[] args) {
    try {
          Test_A obj_A=new Test_A();

          Test_A obj_B=new Test_A();
          String data_A2B="hello, B";
          String en_result=obj_A.AEncrypt(data_A2B,obj_B.APublicKey() );
          System.out.println("A向B说: "+data_A2B);
          System.out.println("A向B发送密文: "+en_result);
          String de_result=obj_B.ADecrypt(en_result);
          System.out.println("B解密的结果: "+de_result);

          String data_B2A="I Know";
          en_result=obj_B.AEncrypt(data_B2A,obj_A.APublicKey() );
          System.out.println("B向A说: "+data_B2A);
          System.out.println("B向A发送密文: "+en_result);
          de_result=obj_A.ADecrypt(en_result);
          System.out.println("A解密的结果: "+de_result);
```

实现结果：

```
A向B说：hello，B
A向B发送密文：UuPir3bBCmJla+SDiHCSxzPiolUB6f0JV1EhOSU2KjFwt/ZhrhfNaJFt0m2euJ/0eFA19HYVHmU1
B解密的结果：hello，B
B向A说：I Know
B向A发送密文：Vo6i0UCsXB/S9be3a3xOjL7wQoygVWu5vS4atldsSLnOuqpFIhtDnHaaFToxaodgBOsYHF7//9em
A解密的结果：I Know
```

3.4 SHA256 信息摘要算法

SHA256 是 SHA-2 下细分出的一种算法，产生的输出是一个 256-bit 的报文摘要。能够计算出一段信息的摘要，也即数字签名。利用数字签名可以知道消息是否被更改过，可以认证消息是否是确实来自意定的信源，还可以使信源不能否认曾将发送的消息。

请结合前一章节的 RSA 算法，完成对消息进行数字签名的方法(发送方对消息进行信息摘要计算，使用自身的私钥进行加密生成数字签名，与消息一同发送至接收方。接收方计算消息的信息摘要，使用发送方的公钥解密数字签名并与信息摘要比对，若二者相同则认为消息没有被篡改)。完成实验并记录实验结果。

生成数字签名和信息摘要：

```java
    /**
     * 生成信息摘要
     * @param data
     * @return
     * @throws NoSuchAlgorithmException
     * @throws UnsupportedEncodingException
     * @throws Exception
     */
    public static String Create_Info(String data) throws NoSuchAlgorithmExcept
        MessageDigest messageDigest=MessageDigest.getInstance("SHA-256");
         messageDigest.update(data.getBytes("UTF-8"));
         String info=new String(Base64.encode(messageDigest.digest()));
         return info;
    }

    /**
     * 私钥对信息摘要进行加密 生成数字签名
     * @param data
     * @return
     * @throws Exception
     */
    public static String Create_Signature(String info,String pri) throws Excep
        return RSA.encrypt(info, pri);
    }
```

调用示例：

```java
public class SHA256 {
    public static void main(String[] args) throws Exception {
        SHA_Test_A obj_A = new SHA_Test_A();
        SHA_Test_A obj_B = new SHA_Test_A();
        String data_A2B="hello, B";
        String en_result=obj_A.AEncrypt(data_A2B,obj_B.APublicKey() );//A用B的公钥加
        String info_A=obj_A.Create_Info(data_A2B);//A生成数字签名
        String qianming=obj_A.AEncrypt(info_A,obj_A.APrivateKey() );//A用B的公钥加密
        System.out.println("A向B说："+data_A2B);
        System.out.println("A向B发送密文："+en_result);
        System.out.println("A生成的信息摘要："+info_A);
        System.out.println("A发送的数字签名："+qianming);
        String de_result=obj_B.ADecrypt(en_result);
        System.out.println("B解密的结果："+de_result);
        String en=obj_B.ADecrypt(qianming );
        System.out.println("B解密后内容生成的信息摘要："+en);
        System.out.println("B解密数字签名得到的信息摘要："+en);
    }
```

实验结果：

```
A向B说：hello, B
A向B发送密文：P2FCl5kRpVS8pcxo3rU934ryB25AB/yvnanCHD9MA6PdqMcfdpiy0zxsHWUc8KtLoIoE8+aPnZxzn3xIDgOqEqs+vxHw
A生成的信息摘要：G/bCGb14CpCDWRZYzIr2pxxJYkXaSwLovoa6qVTb3bU=
A发送的数字签名：Q728rLrc4JRYM/d8R1535tIdHB1QrctlHlLFcGE8iMpq1H376gJmeL8n3oLscSrReWeyY7yIsLSbkt12fg09IcdshpJ
B解密的结果：hello, B
B解密后内容生成的信息摘要：G/bCGb14CpCDWRZYzIr2pxxJYkXaSwLovoa6qVTb3bU=
B解密数字签名得到的信息摘要：G/bCGb14CpCDWRZYzIr2pxxJYkXaSwLovoa6qVTb3bU=
```

三、 实验总结

通过本次实验，我对 AES、RSA、SHA256 算法有了基本的了解，结合课上学习的理论知识，真正应用实践，让我对原理更深的理解了，加强了自己的代码能力。