信息安全数学基础
实验指导书

教师：韩琦教授

实验地点：格物楼 213

实验时间：2023.11.11

一、背景介绍

RSA 算法是一种公开密钥算法，由 Ron Rivest、Adi Shamir 和 Leonard Adleman 于 1977 年共同提出。该算法旨在解决利用公开信道传输、分发私钥的难题，还可以用于保护数据信息的完整性，能够抵抗目前已知的绝大多数密码攻击，已被国际标准化组织推荐为公钥数据加密标准。

本次实验要求同学们利用 Java 或者其它编程语言对经典的 RSA 算法和 RSA-OAEP 算法进行实践。

本次实验的目的包括：

1. 理解和掌握 RSA 算法和 RSA-OAEP 算法的原理和流程：通过实验，学习 RSA 算法和 RSA-OAEP 算法的基本原理和流程，了解公钥加密、私钥解密的过程，并理解其安全性和适用性。

2. 了解加密算法的性能和效率：实验中可以对比 RSA 算法和 RSA-OAEP 算法在不同输入大小和密钥长度下的加密和解密性能，对比它们的效率和时间复杂度，评估其在实际应用中的可行性。

3. 实践加密和解密过程：通过实验，实际进行 RSA 算法和 RSA-OAEP 算法的加密和解密操作，加深对于加密和解密过程的理解，并掌握编写相关代码的技巧和经验。

4. 评估算法的安全性：通过实验，了解 RSA 算法和 RSA-OAEP 算法的安全性，对比它们的安全性级别，例如密钥长度、抵御常见攻击方式等，评估其在实际应用中的安全性。

5. 探索实际应用场景：通过实验，尝试将 RSA 算法和 RSA-OAEP 算法应用于实际场景，例如加密通信、数字签名等，评估它们的适用性和效果，并思考如何结合其他技术改进和扩展算法的应用。

二、实验环境

- 操作系统：Windows7、Windows10、Windows11
- 编程语言：Java
- 代码编辑器：Eclipse

Ps：同学可以选择自己常用的代码编辑器。

三、实验内容

3.1 RSA 算法

RSA 算法包括密钥生成算法、加密算法和解密算法。RSA 算法中的一对密钥分为加密密钥（即公钥）PK 和解密密钥（即私钥）SK，公钥 PK 是公开信息，私钥 SK 需要保密。加密时使用公钥 PK，对明文进行加密，得到密文，解密时使用私钥 SK 对密文进行解密，得到明文。

3.1.1 密钥生成算法

（1）任选两个不同的大素数p和q，计算$n = pq$，$\varphi(n) = (p-1)(q-1)$。

（2）任选一个大整数e，满足$gcd\big(e, \varphi(n)\big) = 1$，$e$为公钥 PK。

（3）作为私钥的 SK 的d，应满足$de \bmod \varphi(n) = 1$，即$de = k\varphi(n) + 1$，其中，$k \geq 1$且是任意一个整数。

（4）公开整数n和e，秘密保存d。

3.1.2 加密算法

将明文m（$m < n$，是一个整数）加密成密文c，加密算法为

$$c = E(m) = m^e \bmod n$$

3.1.3 解密算法

将密文c解密为明文m，解密算法为

$$m = D(c) = c^d \bmod n$$

3.1.4 算法实现

函数间的调用关系如图 3-1 所示。

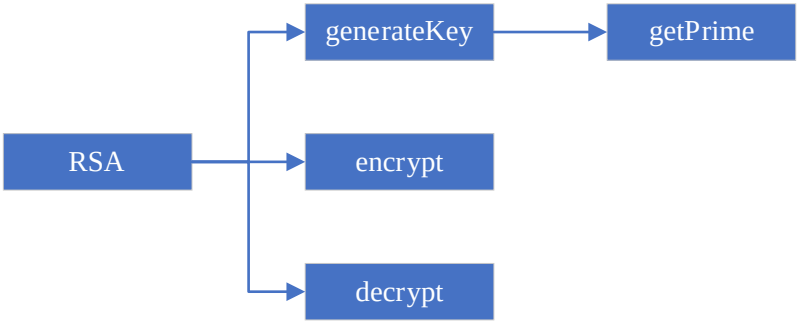

图 3-1 函数之间的调用关系

算法的具体实现如下。同学们将要逐一实现下面代码中的方法，最终实现 RSA 算法，并且将代码的运行进行演示。

```
1   package RSA;
2
3   import java.math.BigInteger;
4   import java.util.Random;
5   import java.util.Scanner;
6
7   public abstract class RSA {
8    /**
9     * 用于生成生成指定长度的大素数
10    * @param bitCount:素数的比特数
11    * @return 素数 init
12    */
13      public BigInteger getPrime(int bitCount) {
14       /**
15        * 请完成这个方法：
16        * 这个方法要求按照 bitCount 生成一个指定长度的大素数；
17        * 使用你在第一次实验中实现的素性检测算法和厄拉多塞筛选算法。
18        */
19      }
20
21      /**
22       * 调用 generateKey 函数来生成加密解密所需要的公、私密钥对
23       * @param bits：安全参数
24       * @return 公钥（e，n）和私钥（d，n）
25       */
26      public BigInteger[] generateKey(int bits){
27       /**
28        * 请完成这个方法：
29        * 这个方法通过 getPrime 方法生成两个大素数，并且根据密钥生成算法
    生成公私钥
30        */
31      }
32
```

```java
    /**
     * 加密
     * @param plaintext:明文
     * @param e：公钥
     * @param n：公钥
     * @return ciphertext：密文
     */
    public BigInteger encrypt(BigInteger plaintext,BigInteger e,BigInteger n) {
        /**
         * 请完成这个方法：
         * 这是加密算法；
         * 使用在第一次实验中完成的快速幂取模算法。
         */
    }

    /**
     * 解密
     * @param ciphertext：密文
     * @param d：私钥
     * @param n：私钥
     * @return plaintext:明文
     */
    public BigInteger decrypt(BigInteger ciphertext,BigInteger d,BigInteger n) {
        /**
         * 请完成这个算法：
         * 这是解密算法；
         * 使用在第一次实验中完成的快速幂取模算法。
         */
    }


    public static void main(String[] args) {
        RSA rsa = new RSA() {};
        Scanner scanner = new Scanner(System.in);

        System.out.print("请输入安全参数 bits: ");
        int bits = scanner.nextInt();

        System.out.print("请输入要加密的明文: ");
        BigInteger plaintext = scanner.nextBigInteger();

        BigInteger[] keys = rsa.generateKey(bits);
```

```
75            BigInteger e = keys[0];
76            BigInteger d = keys[1];
77            BigInteger n = keys[2];
78
79            System.out.println("公钥 (e, n): (" + e + ", " + n + ")");
80            System.out.println("私钥 (d, n): (" + d + ", " + n + ")");
81
82            BigInteger ciphertext = rsa.encrypt(plaintext, e, n);
83            System.out.println("加密后的密文: " + ciphertext);
84
85            BigInteger decryptedText = rsa.decrypt(ciphertext, d, n);
86            System.out.println("解密后的明文: " + decryptedText);
87
88            scanner.close();
89        }
90    }
```

3.2 RSA-OAEP 算法

RSA 算法的加、解密过程固定，因此在实际使用时，需要通过对消息添加 OAEP（最优非对称加密填充）的方式来增加算法的随机性，以此来保证消息的安全，这种算法称为 RSA-OAEP 算法。RSA-OAEP 算法包括三个子算法，分别为填充算法、加密算法和解密算法。

3.2.1 填充算法

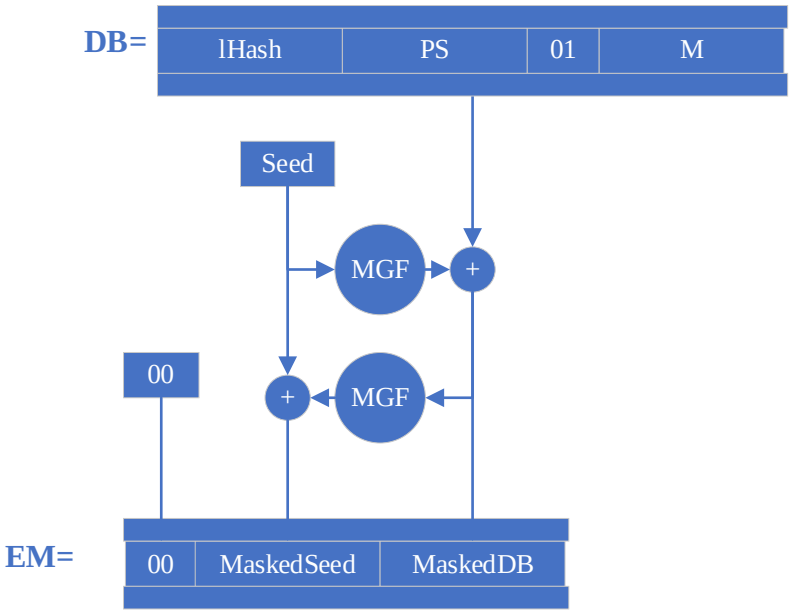

图 3-2 填充算法示意图

（1）选择参数集P作为 Hash 函数H的输入，输出用 0 填充，以获得期望的长度，放入整体数据块DB内。

（2）随机选择一个种子（$seed$）作为掩码生成函数MGF的输入，输出 Hash 值和DB进行按位异或运算，产生掩码DB（$maskedDB$）。该$maskedDB$反过来又作为MGF的输入产生一个$Hash$值，该$Hash$值和种子进行异或运算，产生掩码种子（$maskedSeed$）。将$maskedSeed$和$maskedDB$连接起来，构成编码后的消息 EM。填充算法示意图如 3-2 所示。

3.2.2 填充算法

将填充好的消息作为明文，利用 RSA 算法进行加密，并发送给接收者。

3.2.3 解密算法

接收者收到密文，利用 RSA 算法对其进行解密，得到填充的明文，之后执行以下操作。

1. 将得到的明文M'拆分为$Y||maskedSeed||maskedDB$，其中Y为解密得到的消息的第一个字节，$maskedSeed$的长度为$hLen$，其余部分为$maskedDB$。
2. 计算$seedmask = MGF(maskedDB, hLen)$，计算$seed = maskedSeed \oplus seedmask$。
3. 计算DB的掩码$DBmask = MGF(seed, Lengthof(M') - hLen - 1)$。
4. 计算$DB = maskedDB \oplus DBmask$。
5. 验证DB是否符合如下形式：$DB = lHash||PS||0x01||M$，其中PS中的每一字节均为$0x00$。当以上条件均符合且明文的第一个字节（Y）也为$0x00$，将M取出，作为明文消息，OAEP 验证成功，解密成功。

3.2.4 算法实现

函数之间的调用关系如图 3-3 所示。

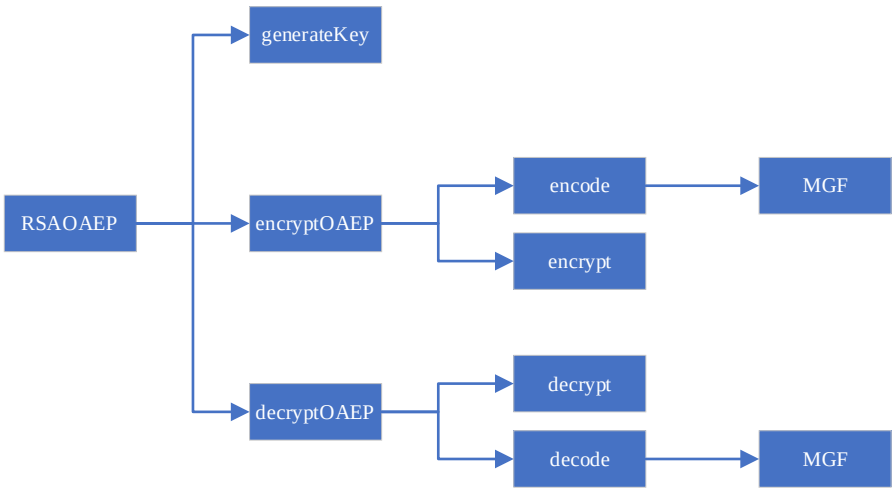

图 3-3 函数调用关系

算法的具体实现如下。同学们将要逐一实现下面代码中的方法，最终实现 RSA-OAEP 算法，并且将代码的运行进行演示。

```
1   package RSA_OAEP;
2
3   import java.math.BigInteger;
4   import java.nio.ByteBuffer;
5   import java.util.Arrays;
6   import java.util.Random;
7   import java.util.Scanner;
8   import java.security.MessageDigest;
9   import java.security.NoSuchAlgorithmException;
10  import java.security.SecureRandom;
11
12
13  public class RSA_OAEP{
14
15   /**
16    * 用于生成生成指定长度的大素数
17    * @param bitCount:素数的比特数
18    * @return 素数 init
19    */
20      public BigInteger getPrime(int bitCount) {
21       /**
22        * 和 RSA 算法中实现的方法一样
23        */
24      }
25
26      /**
27       * 调用 generateKey 函数来生成加密解密所需要的公、私密钥对
28       * @param bits：安全参数
29       * @return 公钥（e，n）和（d，n）
```

```
30          */
31         public BigInteger[] generateKey(int bits){
32          /**
33           * 和在 RSA 算法中实现的方法一样
34           */
35         }
36
37         /**
38          * 生成掩码
39          * @param X:输入消息
40          * @param maskLen：掩码长度
41          * @return T：产生的掩码
42          */
43         public byte[] MGF(byte[] X, int maskLen) throws NoSuchAlgorit
   hmException {
44          /**
45           * 请实现这个方法：
46           * 输入消息 X，利用 Hash 函数，生成长度为 maskLen 的掩码 T
47           */
48         }
49
50         /**
51          * 填充（编码）
52          * @param plainOAEP：明文
53          * @param L：参数标签
54          * @param k：编码长度
55          * @return EM：编码后的消息
56          * @throws NoSuchAlgorithmException
57          */
58         public byte[] encode(byte[] plainOAEP,String L, int k) throws
    NoSuchAlgorithmException {
59          /**
60           * 请完成这个方法：
61           * 通过调用 MGF 方法，按照填充算法的指示来完成编码方法 encode。
62           */
63         }
64
65         /**
66          * OAEP 加密
67          * @param plainOAEP：明文
68          * @param L：参数
69          * @param k：编码长度
70          * @param e：公钥
71          * @param n：公钥
```

```
72          * @return cipherOAEP：加密结果
73          * @throws NoSuchAlgorithmException
74          */
75         public BigInteger encryptOAEP(byte[] plainOAEP,String L,int k
    ,BigInteger e,BigInteger n) throws NoSuchAlgorithmException {
76          /**
77           * 请完成这个方法：
78           * 调用你在 RSA 算法中实现的加密算法以及 encode 方法来对填充后的明
    文进行加密。
79           */
80         }
81
82         /**
83          * 解码
84          * @param EM:编码后的消息
85          * @param L：参数标签
86          * @param k：编码长度
87          * @return 明文：plainOAEP
88          * @throws NoSuchAlgorithmException
89          */
90         public byte[] decode(byte[] EM,String L,int k) throws NoSuchA
    lgorithmException {
91             /**
92              * 请完成这个方法：
93              * 对经过填充（编码）的明文进行解码，得到原始的明文。
94              */
95         }
96
97         /**
98          * 解密
99          * @param cipherOAEP：密文
100         * @param L：标签参数
101         * @param k：编码长度
102         * @param d：私钥
103         * @param n：私钥
104         * @return plainOAEP：明文
105         * @throws NoSuchAlgorithmException
106         */
107        public byte[] decrypt(BigInteger cipherOAEP,String L,int k,Bi
    gInteger d,BigInteger n) throws NoSuchAlgorithmException {
108         /**
109          * 请完成这个方法：
110          * 调用在 RSA 算法中实现的解密算法和 decode 方法对密文解密之后再解
    码得到明文。
```

```
111         */
112     }
113
114     public static void main(String[] args) throws NoSuchAlgorithm
    Exception {
115         RSA_OAEP rsa = new RSA_OAEP();
116         Scanner scanner = new Scanner(System.in);
117
118         // 输入比特数
119         System.out.print("Enter the bit count for RSA key generat
    ion: ");
120         int bitCount = scanner.nextInt();
121
122         // 生成公钥和私钥
123         BigInteger[] keys = rsa.generateKey(bitCount);
124         BigInteger e = keys[0]; // 公钥 e
125         BigInteger d = keys[1]; // 私钥 d
126         BigInteger n = keys[2]; // 公钥和私钥共用 n
127
128         // 输入参数标签
129         System.out.print("Enter the label: ");
130         String label = scanner.next();
131
132         // 输入要加密的明文
133         System.out.print("Enter the plaintext to be encrypted: ")
    ;
134         String message = scanner.next();
135         byte[] plainText = message.getBytes();
136
137
138         System.out.print("Enter the the padded length k: ");
139         int k = scanner.nextInt();
140
141         // 加密
142         BigInteger encrypted = rsa.encryptOAEP(plainText, label,
    k, e, n);
143         System.out.println("Encrypted message: " + encrypted);
144
145         // 解密
146         byte[] decrypted = rsa.decrypt(encrypted, label, k,d,n);
147         System.out.println("Decrypted message: " + new String(dec
    rypted));
148
149         scanner.close();
```

```
150         }
151     }
```

四、实验要求

RSA 算法和 RSA-OAEP 算法的输入可以根据 main 方法的提示进行输入，RSA 算法进行的加密的明文为数字，RSA-OAEP 算法进行加密的明文为字符串。

完成实验后进行现场演示，实验结果要存放在实验报告中，并且和实验源码一起打包发送给助教，提交时**注明姓名和学号**。