RSA 加密算法研究与实现

陈伟剑 (Chen Weijian)

2022 年 5 月 8 日

摘　要

RSA 密码算法是目前在信息安全领域广为使用的公钥加密方法。该算法涉及欧几里得、欧拉函数、乘法逆元等基本数论知识，易于实现。基于 Java 的 BigInterger 类型，设计并实现了 RSA 公钥密码系统中的密钥生成、加密算法、解密算法。其中，通过将扩展欧几里得算法转换成递推形式降低了算法的空间复杂度。

关键字：RSA　Java　BigInteger

Abstract

RSA cryptosystem is the most widely used public key cryptosystem in the field of information security. This algorithm involves Euclid, Euler function, multiplicative inverse and other basic number theory knowledge, which is easy to implement. Based on Java language and its BigInteger class , the key generation , encryption , decryption and digital signature of RSA public key cryptosystem are designed and implementd. Especially, the extended Euclidean algorithm is transformed into a recursive form to reduce the space complexity of the algorithm.

keywords: RSA　Java　BigInteger

目录

1 概念 **1**

2 前置数学知识 **1**

2.1 欧几里得算法 1

2.2 欧拉函数 2

2.3 裴蜀定理 2

2.4 乘法逆元 2

2.5 费马小定理 2

3 RSA 算法过程 **2**

3.1 算法概述 2

3.2 辅助算法实现 3

3.2.1 扩展欧几里得算法与乘法逆元 3

3.2.1.1 一般扩展欧几里得算法推导 3

3.2.1.2 递推形式扩展欧几里得算法推导 4

3.2.1.3 乘法逆元计算 6

3.2.2 快速模幂运算 6

3.2.3 大素数判定 7

3.2.3.1 费马素性测试 7

3.2.3.2 Miller-Rabin 算法测试 7

4 总结 **8**

1 概念

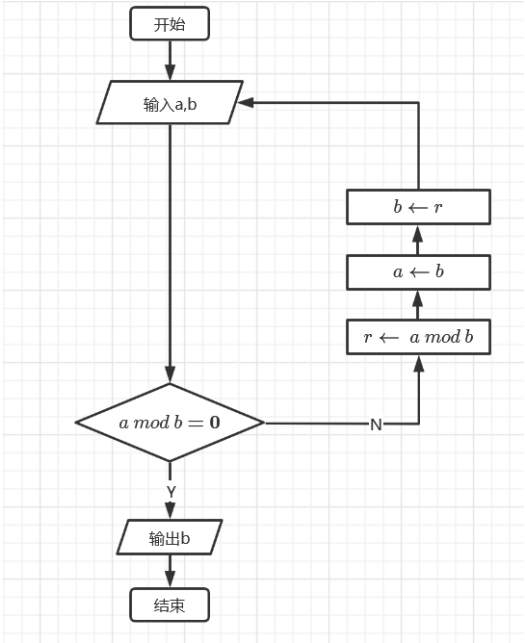
现代密码学体系加密算法一般分为对称加密算法和非对称加密算法（或公钥加密算法）两种。所谓对称加密就是通信双方加密和解密消息时使用相同的密钥，对称密码算法的优点是加解密运算速度较快，适用于处理大批量消息处理，缺点是很难解决密钥分发的安全性和数字签名等问题。在非对称加密算法中，发送方和接收方使用不同的密钥，发送方使用接收放生成的公钥对明文进行加密生成密文，接收方通过自己生成的私钥对密文进行解密获取明文，非对称密码算法的优点是不需要额外开辟安全信道保证传输的安全，缺点是加密速度较慢，故非对称密码算法更适用于处理密钥分发和数字签名。一般实践中，两种加密算法会一起使用，形成混合密码算法，这种算法使用对称加密算法处理传输内容，使用非对称加密算法对密钥进行加密。

RSA 密码体系 (又成 RSA 算法) 是目前比较流行的密码加密算法,该算法在 1977 年由 Riverst、Shamir、Adleman3 位专家提出并于 1978 年首次发表。RSA 算法整体上易于实现，但是当算法中相关的参数值很大时，需要考虑溢出问题，为此可以使用 Java 中的 BigInteger 类型。不过鉴于 Java 中使用 BigInteger 类型会形成大量的对象方法调用从而导致代码可读性较低，故本文只在一处较为简单的辅助算法中给出一般实现和 BigInteger 实现的两种代码。

2 前置数学知识

2.1 欧几里得算法

欧几里得算法又称辗转相除法，用于计算两个非负整数 a，b 的最大公约数。算法流程如下:



相关计算公式 $gcd(a,b) = gcd(b, a \bmod b)$。

朴素欧几里得算法相关代码

```
public static int gcd(int a,int b){
    while(b!=0){
        int t=a%b;
        a=b;
```

```
            b=t;
        }
        return a;
    }
```

2.2 欧拉函数

定义：一组数 $x_1, x_2, ..., x_n$ 称为模 m 的缩系，如果对 $\forall j, s \in \mathbb{Z}, 1 \leqslant j \leqslant s, gcd(x_j, r) = 1$，并定义 $\phi(m) = s$，即 $1, 2, ..., m$ 中与 m 互质的数的个数，$\phi(m)$ 称为欧拉函数。

显然 $\phi(1) = 1$，而对于 $m > 1$，$\phi(m)$ 就是 $1, 2, ..., m-1$ 中与 m 互质的数的个数。特别的，对于 $p \in \{$ 素数 $\}$，有 $\phi(p) = p - 1$。

2.3 裴蜀定理

定理：已知整数 a, b 和 $d = gcd(a, b)$。若 a, b 是整数，那么对任意整数 x, y，$ax + by$ 都一定是 d 的倍数，特别的，一定存在整数 x, y，使 $ax + by = d = gcd(a, b)$ 成立。

推论：a, b 互质的充分必要条件是存在整数 x, y，使 $ax + by = 1$。

2.4 乘法逆元

背景：已知整数 a, b。现要根据 $ax \equiv 1 \ mod \ b$ 计算 x。

对于这个同余式，可以转换为 $ax + by = 1, (y \in \mathbb{Z})$，这个形式恰好符合裴蜀定理的形式，于是 $gcd(a, b) = 1$。这表明 a, b 互质是逆元存在的必要条件。同理可以证明：a, b 互质是 a 在模 b 下存在逆元的充分条件。

2.5 费马小定理

定理[1]：已知 p 为素数，a 为任意自然数，则 $a^p \equiv a \ (mod \ p)$。进一步，若 $gcd(a, p) = 1$，则 $p|(a^{p-1} - 1)$，即 $a^{p-1} \equiv 1 \ (mod \ p)$。

引理 (二次探测定理) 如果 p 是一个素数，且 $0 < x < p$，则方程 $x^2 \equiv 1 \ (mod \ p)$ 的解为 $x = 1 \ or \ p - 1$。

3 RSA 算法过程

3.1 算法概述

整个加密过程需要用到的参数如下

表 1: RSA 算法相关符号

符号	说明
p、q	两个随机素数
n	p 和 q 的乘积
l	n 的欧拉数
e	$(1,l)$ 区间内和 l 互质的随机整数
d	e 在模 l 下的乘法逆元

接收方生产私钥和公钥的基本过程如下:

1、随机生成两个大素数 p□q

2、令 $n = p * q$

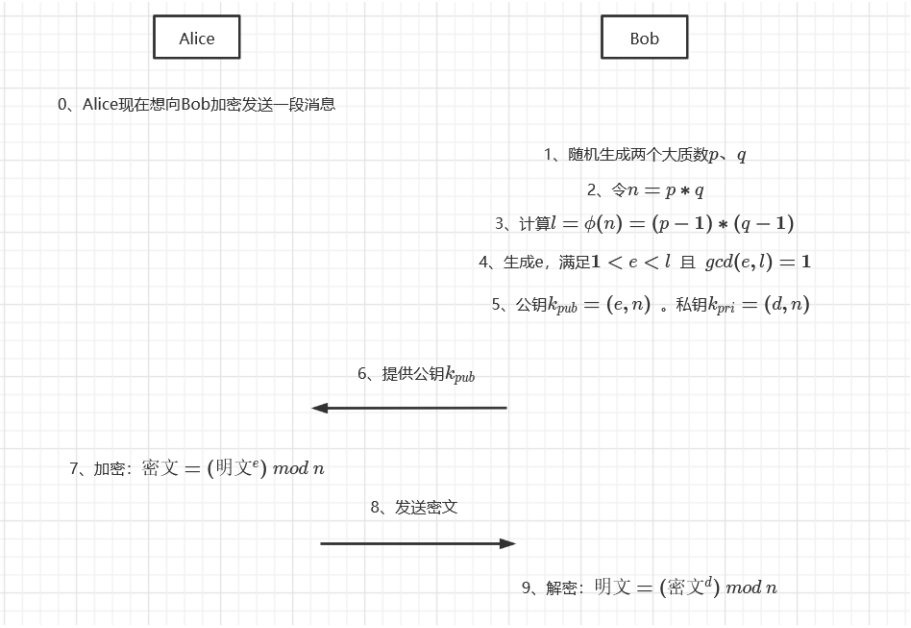
3、令 $l = \phi(n) = (p-1) * (q-1)$

4、随机生成一个数 e，满足 $1 < e < l$ 且 e 和 l 互质

5、由乘法逆元的原理，通过 $(d * e) mod l = 1$ 计算得到 d

6、(e, n) 作为公钥，(d, n) 作为私钥

整体流程[2] 如图:



3.2 辅助算法实现

3.2.1 扩展欧几里得算法与乘法逆元

3.2.1.1 递推形式扩展欧几里得算法推导

欧几里得算法的另一个用途就是求解 $(a, b) = ax + by$ 中的 x, y。在欧几里得算法中，递归部分的核心等式为 $gcd(a, b) = gcd(b, a \ mod \ b) = gcd(b, a - b\lfloor \frac{a}{b} \rfloor)$ ，由此式可以得出和 x, y 相关的递归关系。根据裴蜀定理，可以找到四个整数 x, y, x', y', 使得

$$\begin{cases} ax + by = gcd(a, b) \\ bx' + (a - b\lfloor \frac{a}{b} \rfloor)y' = gcd(b, a - b\lfloor \frac{a}{b} \rfloor) \end{cases}$$

由于等式右侧相等，所以有 $ax+by=bx'+(a-b\lfloor\frac{a}{b}\rfloor)y'$，整理得到 $a(x-y')+b(y-(x'-\lfloor\frac{a}{b}\rfloor y'))=0$，因为 a,b 均不为 0，所以有

$$\begin{cases} x-y'=0 \\ y-(x'-\lfloor\frac{a}{b}\rfloor y')=0 \end{cases}$$

即

$$\begin{cases} x=y' \\ y=x'-\lfloor\frac{a}{b}\rfloor y' \end{cases}$$

因此，要求解 x,y，只需要递归求解更小规模的 x',y' 即可。计算到最后的边界条件 $b=0$ 时，有 $x=1,y=0$。

在数据足够大时，采用递归方式的空间复杂度会急剧增加。现考虑将扩展欧几里得算法转化为递推形式以降低空间复杂度。

对于先前已知的递归式

$$\begin{cases} x=y' \\ y=x'-\lfloor\frac{a}{b}\rfloor y' \end{cases}$$

可以转换为矩阵形式

$$\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ 1 & -d_1 \end{pmatrix}\begin{pmatrix} x' \\ y' \end{pmatrix},(d_1=\lfloor\frac{a}{b}\rfloor)$$

x',y' 同样可以展开成上述形式，由此一直递推到 $x=1,y=0$，形成下式

$$\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ 1 & -d_1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & -d_2 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & -d_3 \end{pmatrix}\cdots\begin{pmatrix} 0 & 1 \\ 1 & -d_n \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

每次迭代时，都可以从左向右计算一个矩阵，从而达到了迭代的目的。

假设在某次迭代之前计算的矩阵之积为 $\begin{pmatrix} m_1 & m_2 \\ n_1 & n_2 \end{pmatrix}$(迭代开始之前是一个单位阵)，下一个待右乘的矩阵为 $\begin{pmatrix} 0 & 1 \\ 1 & -d_t \end{pmatrix}$，二者之积可以写成

$$\begin{pmatrix} m_1 & m_2 \\ n_1 & n_2 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & -d_t \end{pmatrix}=\begin{pmatrix} m_2 & m_1-d_tm_2 \\ n_2 & n_1-d_tn_2 \end{pmatrix}$$

所有矩阵计算完成之后,得到一个最终矩阵 $\begin{pmatrix} M_1 & M_2 \\ N_1 & N_2 \end{pmatrix}$,这个矩阵满足 $\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} M_1 & M_2 \\ N_1 & N_2 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}$，展开可得 $x=M_1,y=N_1$。

综上，递推的算法步骤如下:

1、给矩阵赋初值 $\begin{pmatrix} x & m \\ y & n \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

2、每次迭代时，计算 $d = \lfloor \frac{a}{b} \rfloor$

3、更新矩阵

$$\begin{pmatrix} x & m \\ y & n \end{pmatrix} \leftarrow \begin{pmatrix} x & m \\ y & n \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -d \end{pmatrix} = \begin{pmatrix} m & x - dm \\ n & y - dn \end{pmatrix}$$

4、进行一般欧几里得算法的迭代

$$\begin{pmatrix} a \\ b \end{pmatrix} \leftarrow \begin{pmatrix} b \\ amodb \end{pmatrix}$$

5、$b = 0$ 时，计算结束，此时已经求出需要的 x, y，并且最大公约数存如变量 a 中。

递推代码如下

```java
//java无法传入指针，这里array[0]代表x，array[1]代表y
public static long exgcd(long a,long b,long[] array){
    long m=0;
    long n=1;
    while(b>0){
        long d=a/b;
        long t;

        t=m; m=array[0]-d*t; array[0]=t;
        t=n; n=array[1]-d*t; array[1]=t;
        t=a%b; a=b; b=t;
    }
    return a;
}
```

下面为采用 BigInteger 的代码形式 (由于可读性相对低，本文只在此处展示 BigInteger 相关代码)

```java
public static BigInteger exgcd(BigInteger a,BigInteger b,BigInteger[] array){
    BigInteger m=new BigInteger("0");
    BigInteger n=new BigInteger("1");
   while(b.compareTo(zero)>0){
        BigInteger d=a.divide(b);
        BigInteger t;

        t=m; m=array[0].add(d.negate().multiply(t)); array[0]=t;
        t=n; n=array[1].add(d.negate().multiply(t)); array[1]=t;
        t=a.mod(b); a=b; b=t;
```

```
    }
    return a;
}
```

3.2.1.2 乘法逆元计算

当使用扩展欧几里得算法找到一组 x, y 满足 $ax + by = 1$ 时，就可以得出 $ax \equiv 1 \ (mod\ b)$，即 x 是 a 的乘法逆元, 那么对 $\forall x + kb, k \in \mathbb{N}$ 也都是 a 的乘法逆元，这表明，一定有 a 的一个乘法逆元在区间 $[0, b)$ 内。

利用前面完成的扩展欧几里得算法，找到 n 在模 p 下的乘法逆元。代码如下

```
public static long inv(long a,long p){
   long[] array={1,0}; //array[0]代表x，array[1]代表y
   if(exgcd(a,p,array)==1){
      array[0]%=p;
      return array[0]>=0 ? array[0]:array[0]+p;
   }
      else return -1; //-1代表不存在对应的乘法逆元
}
```

3.2.2 快速模幂运算

快速模幂算法基于快速幂多了一些取模运算，整体思想相同。快速幂的核心思想是每一步都将指数减半，相应的底数做平方运算，这样可以达到快速降低指数的目的。

例如：已知 a, b 求 a^b，假设 $b = 11$。

这里可以将表达式的指数写成二进制的形式

$$a^{11} = a^{2^3+2^1+2^0} = a^{(1011)_2}$$

指数减半等价于对指数的二进制形式右移一位，假如当前末位为 0，则直接进行移位运算。若当前末尾不为 0，则提出一个当前底数后，再将指数右移一位。如 $a^{(1011)_2} = a * a^{(1010)_2} = a * (a^2)^{(101)_2}$

相关代码如下

```
public static long qmod(long a,long p,long mod){
    long ans=1;
    a%=mod;
    while(p>0){
        if((p&1)==1){
            //p为奇数，则需要拿出一个a，指数减一成为偶数后，再将指数右移一位
            ans=(ans*a)%mod; //ans在整个计算过程中都会比mod小，不需要额外进行取模操作
        }
        a=(a*a)%mod;
        p>>=1; //p右移一位
    }
```

```
        return ans;
    }
```

3.2.3 大素数判定

3.2.3.1 费马素性测试

该测试基于费马小定理。费马小定理对所有素数都成立，因此，对于一个待测数 p，可以随机挑选 $[2, p-2]$ 中的任一整数 t，若存在 t 不满足 $a^{t-1} \equiv 1\ (mod\ p)$，说明 p 不是素数。测试次数越大，则素数判定的正确性越高。

但是某些合数也满足费马小定理，这样的合数称为卡迈克尔数。给定卡迈克尔数 C，对所有满足 $gcd(a, C) = 1$ 的整数 a，都满足 $a^{C-1} \equiv 1\ (mod\ C)$。$n = 561 = 3 * 11 * 17$ 就是一个卡迈克尔数。

3.2.3.2 Miller-Rabin 算法测试

费马素性测试的测试范围不够大，实践中更多会采用 Miller-Rabin 算法进行测试[4]。

假设 n 是奇素数，则 $n-1$ 必为偶数，令 $n - 1 = 2^q * m$。

随机选取整数 $a(1 < a < n)$，由费马小定理知 $(a^{2^q * m} = a^{n-1}) \equiv 1\ (mod\ n)$ 。

由二次探测定理知：$a^{2^{q-1} * m} \equiv 1\ (mod\ n)$ 或 $a^{2^{q-1} * m} \equiv n - 1\ (mod\ n)$。

若 $a^{2^{q-1} * m} \equiv 1\ (mod\ n)$ 成立，则再由二次探测定理可知：$a^{2^{q-2} * m} \equiv 1\ (mod\ n)$ 或 $a^{2^{q-2} * m} \equiv n - 1\ (mod\ n)$。

如此反复，直到某一步 $a^m \equiv 1\ (mod\ n)$ 或 $a^m \equiv n - 1\ (mod\ n)$ 。如果 n 是素数，则 $a^m \equiv 1\ (mod\ n)$ ，或存在 $0 \le r \le q - 1$ 使 $a^{2^r * m} \equiv 1\ (mod\ n)$。

算法思路为:

1、给定奇数 n，为了判定 n 是不是素数，首先测试 $a^{2^q * m} \equiv 1(modn)$，是否成立 (这一步需要使用快速模幂算法)，若不成立，则 n 一定是合数；若成立，则继续运行算法。

2、考察下面的 Miller 序列：

$$a^{2^0 m}(modn), a^{2^1 m}(modn), a^{2^2 m}(modn), ...a^{2^{q-1} m}(modn),$$

若 $a^m \equiv 1(modn)$，或存在 $0 \le r \le q - 1$ 使 $a^{2^r m} \equiv 1(modn)$ 成立，则通过测试。

相关代码如下

```
public static boolean millerRabin(long num){
   if(num==2) return true;
   else if(num<2 || (num&1)==0) return false;

   int judgeTime=100; //随机挑100个数进行判定，可以定义判断的次数
    long m=num-1;
    long t=0;
    //计算 num-1=m*(2^t) 的m和t
```

```
        while((m&1)==0) {
            m >>= 1;
            t++;
        }

        for(int i=1;i<=judgeTime;i++){
            //[2,num-1]中的随机数a
            long a=(long)(Math.random()*(num-2)%num+2);
            //计算 a^m % num
            long x=qmod(a,m,num),y=x;

            for(int j=0;j<t;j++){
                y=qmod(x,2,num); //x^2 mod num
                //不满足二次探测定理，也就是y==1了但是x并不等于1或者n-1，那么n就一定不是素数
                if(y==1 && x!=1 && x!=num-1){
                    return false;
                }
                x=y;
            }
            //子循环结束后，y仍不为1，则num不是素数
            if(x!=1) return false;
        }
        return true;
    }
```

4 总结

RSA 是目前使用广泛的一种公钥密码系统，它的安全性基于大整数因子分解难题[5]。目前主流的电子计算机的算力尚不能快速破解经 RSA 加密的密钥，同时，RSA 生成的密钥长度从以前的 1024 位发展到了现在的 2048 位和 3082 位，因此目前 RSA 加密的安全性还是有所保证的。RSA 密码系统可用于消息加密和数字签名，密码强度主要与 RSA 模数 N 的长度有关，目前长度是 2048 位，即 256 字节，非常安全。经过本次 RSA 算法的粗略设计与实现，一定程度上能够提高对数论与编程的认知，这对今后了解更前沿的密码算法大有裨益。

参考文献

[1] 费马小定理 https://zhuanlan.zhihu.com/p/87611586

[2] RSA 算法分析 https://zhuanlan.zhihu.com/p/36347853

[3] 扩展欧几里得算法 https://zhuanlan.zhihu.com/p/58241990

[4] Miller-Rabin 素性测试 https://blog.csdn.net/ECNU_LZJ/article/details/72675595

[5] 王生玉, 汪金苗, 董清风, 等. 基于属性加密技术研究综述 [J]. 信息网络安全,2019,19(9)：76-80