南開大學
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

密码学实验报告

公钥密码算法 RSA

于文明

年级：2020 级

专业：信息安全

指导教师：古力

2022 年 12 月 25 日

摘要

关键字：Parallel

目录

一、 实验内容

(一) 实验目的

通过实际编程了解公钥密码算法 RSA 的加密和解密过程，加深对公钥密码算法的了解和使用。

(二) 实验环境

运行 Windows 操作系统的 PC 机，具有 VC 等语言编译环境

(三) 实验内容

1、为了加深对 RSA 算法的了解，根据已知参数：p = 3，q = 11，m = 2，手工计算公钥和私钥，并对明文 m 进行加密，然后对密文进行解密。

2、编写一个程序，用于生成 512 比特的素数。

3. 利用 2 中程序生成的素数，构建一个 n 的长度为 1024 比特的 RSA 算法，利用该算法实现对明文的加密和解密。

4、在附件中还给出了一个可以进行 RSA 加密和解密的对话框程序 RSATool，运行这个程序加密一段文字，了解 RSA 算法原理。

(四) 实验要求

1、对实验步骤 2，写出生成素数的原理，包括随机数的生成原理和素性检测的内容，并给出程序框图;

2、对实验步骤 3，要求分别实现加密和解密两个功能，并分别给出程序框图。

二、 程序流程图

(一) 大素数生成

大素数生成主要包括以下四个过程：生成大随机数，奇偶性检测，素数表筛，Rabin-Miller 检测, 具体原理见 RSA 算法介绍 (大素数生成) 部分, 总体流程图如下:

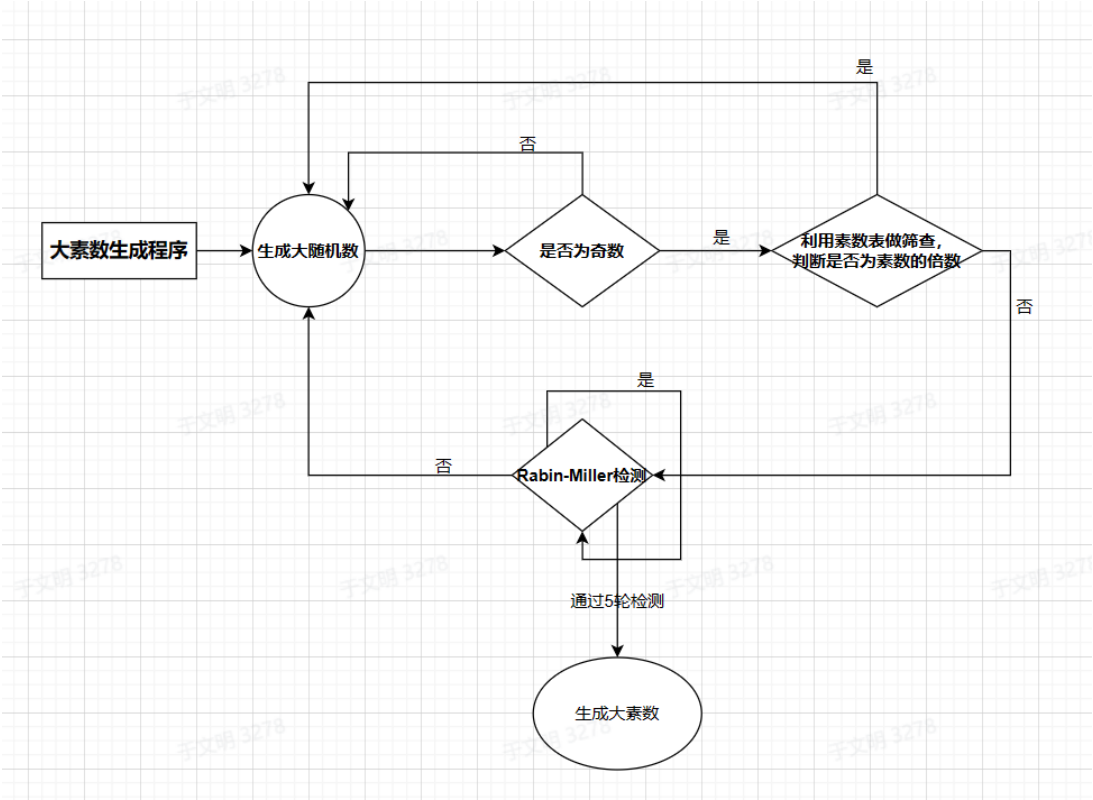

图 1: 大素数生成流程图

1. Rabin-Miller 检测流程图

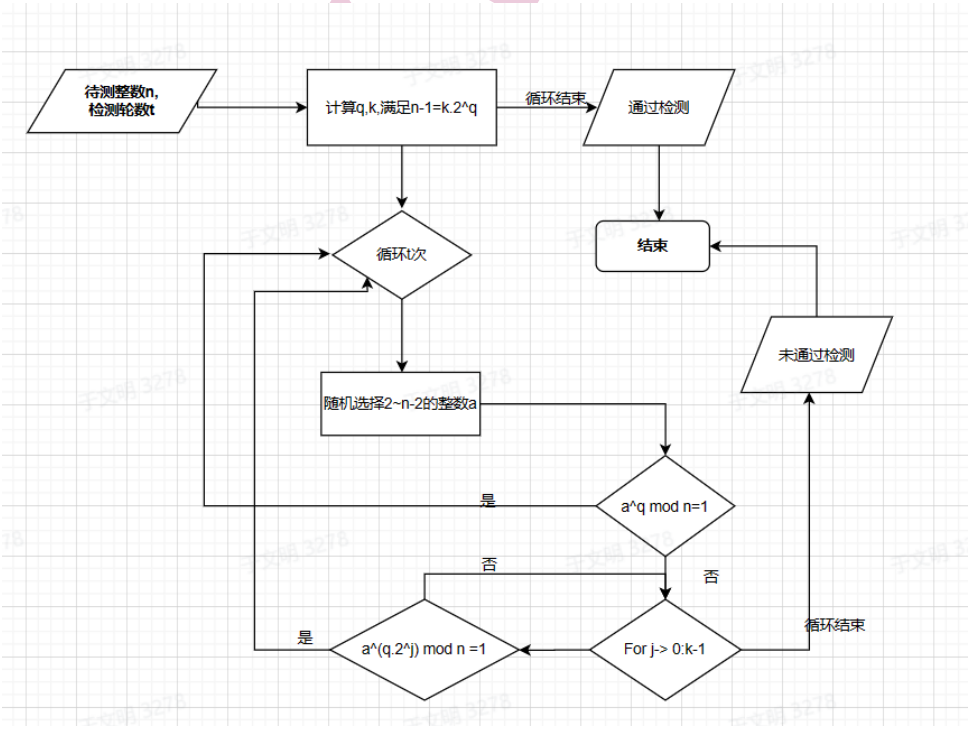

图 2: Rabin-Miller 检测流程图

(二) RSA 加解密

1. RSA 加密

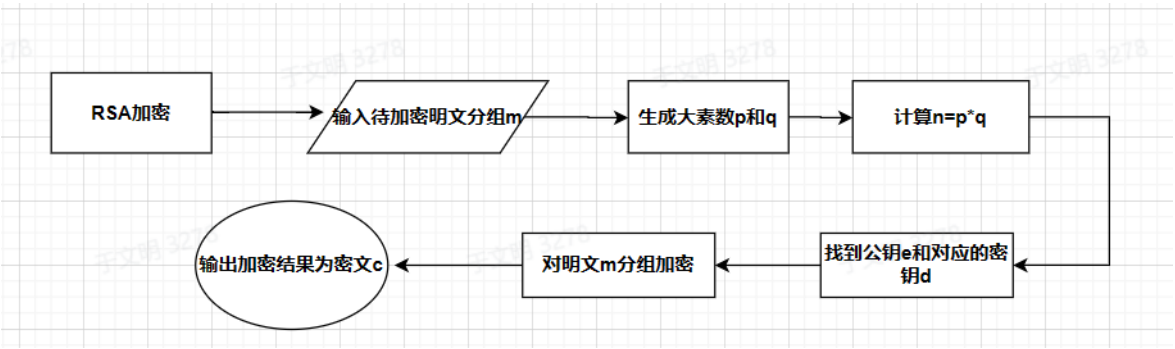

图 3: RSA 加密流程图

2. RSA 解密

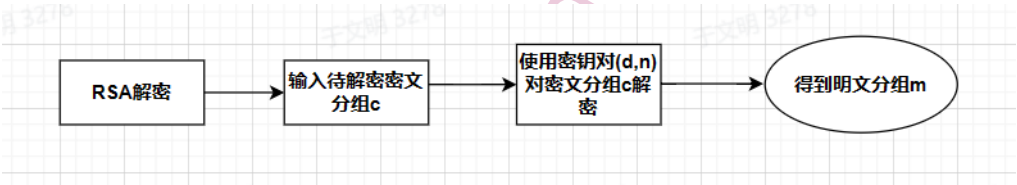

图 4: RSA 解密流程图

三、 RSA 算法介绍

(一) 大素数生成

1. 生成大随机数

因为库函数 rand() 最大只能产生 0X7FFF 的数, 为了能产生 32 位的随机数, 需要 3 次 rand() 操作, 即 rand()«17 +rand()«2+rand(), 同时根据需要生成的大整数的 size, 在外层添加循环 For i->1:size: data[i]=rand()«17 +rand()«2+rand();

2. 素数表筛查

定义 IsPrime() 函数判断是否是素数,然后从小到大生成 1000 个素数填充进素数表 prime[1000]。使用 Random() 生成一个大随机数，进行奇偶性检测，判断是否为奇数。使用 prime[1000] 进行筛查，使之不能被其中的任一素数整除。伪代码

itob

```
IsPrime();
genPrime();
while(!flag):
    n.Random();
    while(!n.isOdd())
        n.Random();
```

```
    for i -> 0:sizeof(prime)/sizeof(int)
        if n//prime[i]:
            flag=false;
            break;
```

3. Rabin-Miller 检测

Miller Rabin 算法的依据是费马小定理:$a^{p-1} \equiv 1(modP)$
假设需要判断的数是 p
我们把 p−1 分解为 $2^k * t$
当 p 是素数，有 $a^{(2^k * t)} \equiv 1(modp)$
然后随机选择一个数 a，计算出 $a^t(modp)$
让其不断的自乘，同时结合二次探测定理进行判断
如果我们自乘后的数 (mod p)=1，但是之前的数 (mod p)$\neq \pm 1$
那么这个数就是合数 (违背了二次探测定理)
这样乘 k 次，最后得到的数就是 a^{p-1}
那么如果最后计算出的数不为 1，这个数也是合数 (费马小定理)

(二) RSA 加解密

1. 公钥

选择两个不同的大素数 p 和 q，n 是二者的乘积，即$\varphi(n)$ pq e e n e

2. 私钥

求出正数 d，使其满足 e*d=1 mod $\varphi(n)$ n d

3. 加密算法

对于明文 m，由 c≡ $m^e modn$， c

4. 解密算法

对于密文 c, 由 m≡ $c^d modn$， m

(三) 攻击原理

如果攻击者获得了 n、e 和密文 c，为了破解密文必须计算出私钥 d，为此需要先分解 n。当 n 的长度为 1024 比特时，在目前还是安全的，但从因式分解技术的发展来看，1024 比特并不能保证长期的安全性。为了保证安全性，要求在一般的商业应用中使用 1024 比特的长度，在更高级别的使用场合，要求使用 2048 比特长度。

四、 核心代码

(一) 辅助函数

辅助函数包括欧几里得算法用来求公约数和扩展欧几里得算法求乘法逆元

1. 欧几里得算法 gcd

欧几里得算法使用辗转相除法

gcd

```
BigInt gcd(const BigInt& m, const BigInt& n)
{
        if (n == 0)
                return m;
        else
                return gcd(n, m % n);
}
```

2. 扩展欧几里得算法 extend$_g cd$

extend$_g cd$

```
BigInt Extended_gcd(const BigInt& a, const BigInt& b, BigInt& x, BigInt& y)
{
        BigInt t, d;
        //如果一个操作数为零则无法进行除法运算
        if (b == 0)
        {
                x = 1;
                y = 0;
                return a;
        }
        d = Extended_gcd(b, a % b, x, y);
        t = x;
        x = y;
        y = t - ((a / b) * y);
        return d;
}
```


(二) 大整数 BigInt

1. 大整数结构体

大整数的成员有 data[64] 用来存大整数, bool pn 标识正负号, int len 表示数据长度。Clear() 用来大数清零，GetPN 判断大数的正负，GetLength() 返回大数的长度，Random() 用来产生大随机数，display() 和 Output() 用来输出 16 进制大整数，IsOdd() 判断奇偶性。此外还重载了一系列运算符

BigInt

```
class BigInt {
        friend BigInt operator+ (const BigInt&, const BigInt&);
        friend BigInt operator- (const BigInt&, const BigInt&);
        friend BigInt operator- (const BigInt&, const int&);
```

```
        friend BigInt operator* (const BigInt&, const BigInt&);
        friend BigInt operator* (const BigInt&, const unsigned int&);
        friend BigInt operator% (const BigInt&, const BigInt&);
        friend BigInt operator/ (const BigInt&, const BigInt&);
        friend bool operator< (const BigInt&, const BigInt&);
        friend bool operator> (const BigInt&, const BigInt&);
        friend bool operator<= (const BigInt&, const int&);
        friend bool operator== (const BigInt&, const BigInt&);
        friend bool operator== (const BigInt&, const int&);
        friend ostream& operator<< (ostream&, const BigInt&);
        friend BigInt Power(const BigInt&, const BigInt&, const BigInt&);//计
            算幂
        friend void pTabScr(BigInt& n);

public:
        BigInt();
        BigInt(const int&);
        BigInt(const BigInt&);

        void operator= (const BigInt&);
        void operator= (const int& a) { Clear(); data[0] = a; }
        void operator>> (const int&);

        inline int GetLength() const;    //返回大数的长度
        bool TestSign() { return pn; }  //判断大数的正负
        void Clear();  //大数清0
        //void Random(); //随机产生一个大数
        void Random(bool small);  //随机产生一个稍小的大数
        void display() const;
        void Output(ostream& out) const;
        bool IsOdd() const { return (data[0] & 1); }  //判断大数奇偶性

public:
        unsigned int data[MAXSIZE];
        bool pn;
        int len;
};
```

2. 重载运算符

这里以比较复杂的运算符重载作为示例

(1) 大数乘法，采用竖式乘法的算法

大数乘法

```
BigInt operator* (const BigInt& a, const BigInt& b)
{
        //last存放竖式上一行的积,temp存放当前行的积
        BigInt result, last, tmp;
```

```
        //sum存放当前行带进位的积
        unsigned __int64 sum;
        //存放进位
        unsigned int c;

        //进行竖式乘
        for (int i = 0; i < b.GetLength(); i++)
        {
                c = 0;
                //B的每一位与A相乘
                for (int j = 0; j < a.GetLength() + 1; j++)
                {
                        sum = ((unsigned __int64)a.data[j]) * (b.data[i]) + c
                            ;
                        if ((i + j) < MAXSIZE)
                                tmp.data[i + j] = (unsigned int)sum;
                        c = (sum >> 32);
                }
                result = (tmp + last);
                last = result;
                tmp.Clear();
        }

        //判断积的符号
        if (a.pn == b.pn)
                result.pn = true;
        else
                result.pn = false;
        result.len = result.GetLength();
        return result;
}
```

(2) 大数除法，采用试商除法, 采用二分查找法优化

大数除法

```
BigInt operator/ (const BigInt& a, const BigInt& b)
{
        //mul为当前试商,low,high为二分查找试商时所用的标志
        unsigned int mul, low, high;
        //sub为除数与当前试商的积,subsequent为除数与下一试商的积
        //dividend存放临时被除数
        BigInt dividend, quotient, sub, subsequent;
        int lengtha = a.GetLength(), lengthb = b.GetLength();

        //如果被除数小于除数,直接返回0
        if (a < b)
        {
                if (a.pn == b.pn)
                        quotient.pn = true;
```

```
15              else
16                      quotient.pn = false;
17              return quotient;
18      }
19
20      //把被除数按除数的长度从高位截位
21      int i;
22      for (i = 0; i < lengthb; i++)
23              dividend.data[i] = a.data[lengtha - lengthb + i];
24
25      for (i = lengtha - lengthb; i >= 0; i--)
26      {
27              //如果被除数小于除数,再往后补位
28              if (dividend < b)
29              {
30                      for (int j = lengthb; j > 0; j--)
31                              dividend.data[j] = dividend.data[j - 1];
32                      dividend.data[0] = a.data[i - 1];
33                      continue;
34              }
35
36              low = 0;
37              high = 0xffffffff;
38
39              //二分查找法查找试商
40              while (low < high)
41              {
42                      mul = (((unsigned __int64)high) + low) / 2;
43                      sub = (b * mul);
44                      subsequent = (b * (mul + 1));
45
46                      if (((sub < dividend) && (subsequent > dividend)) ||
                            (sub == dividend))
47                              break;
48                      if (subsequent == dividend)
49                      {
50                              mul++;
51                              sub = subsequent;
52                              break;
53                      }
54                      if ((sub < dividend) && (subsequent < dividend))
55                      {
56                              low = mul;
57                              continue;
58                      }
59                      if ((sub > dividend) && (subsequent > dividend))
60                      {
61                              high = mul;
```

```
                                continue;
                        }

                }

                //试商结果保存到商中去
                quotient.data[i] = mul;
                //临时被除数变为被除数与试商积的差
                dividend = dividend - sub;

                //临时被除数往后补位
                if ((i - 1) >= 0)
                {
                        for (int j = lengthb; j > 0; j--)
                                dividend.data[j] = dividend.data[j - 1];
                        dividend.data[0] = a.data[i - 1];
                }
        }

        //判断商的符号
        if (a.pn == b.pn)
                quotient.pn = true;
        else
                quotient.pn = false;
        quotient.len = quotient.GetLength();
        return quotient;
}
```

3. 快速幂

快速幂运算——计算 n 的 p 次幂模 m, 利用 Montgomery 算法

快速幂

```
BigInt Power(const BigInt& n, const BigInt& p, const BigInt& m)
{
        BigInt temp = p;
        BigInt base = n % m;
        BigInt result(1);

        //检测指数p的二进制形式的每一位
        while (!(temp <= 1))
        {
                //如果该位为1, 则表示该位需要参与模运算
                if (temp.IsOdd())
                {
                        result = (result * base) % m;
                }
                base = (base * base) % m;
```

```
                temp >> 1;
        }
        return (base * result) % m;
}
```

(三) 生成大素数

生成大素数主要由四个过程：生成大随机数，奇偶检测，素数表筛，Rabin-Miller 检测，主要代码如下

GeneratePrime

```
BigInt GeneratePrime()
{
        BigInt n;
        int i = 0;

        //无限次循环，不断产生素数，直到i==5时（通过五轮RabinMiller测试）才会跳出while循环
        while (i < 5)
        {
                cout << "产生待测大奇数: " << endl;
                pTabScr(n);
                n.display();

                i = 0;
                //进行五轮RABINMILLER测试，五轮全部通过则素数合格
                for (; i < 5; i++)
                {
                        if (!RabinMiller(n))
                        {
                                cout << "RABINMILLER测试失败" << endl;
                                break;
                        }
                        cout << "RABINMILLER测试通过" << endl;
                }
        }
        return n;
}
```

1. 生成大随机数

产生一个随机大数,if small=false 大数的 LENGTH 为 SIZE 的 1/4;small=true, 大数的 LENGTH 为 SIZE 的 1/8. 由于 RAND() 最大只能产生 0X7FFF 的数, 为了能产生 32 位的随机数, 需要 3 次 RAND() 操作

Random

```
void BigInt::Random(bool small)
```

```
{
        if (small == false)
        {
                for (int i = 0; i < (MAXSIZE / 4); i++)
                        //由于RAND()最大只能产生0X7FFF的数,为了能产生32位的随机数,需要
                        //3次RAND()操作
                        data[i] = (rand() << 17) + (rand() << 2) + rand() % 4;
        }
        else
        {
                for (int i = 0; i < (MAXSIZE / 16); i++)
                        //由于RAND()最大只能产生0X7FFF的数,为了能产生32位的随机数,需要
                        //3次RAND()操作
                        data[i] = (rand() << 17) + (rand() << 2) + rand() % 4;
        }
        len = this->GetLength();
}
```

2. 奇偶判断和素数表筛查

素数表筛查

```
//素数表筛查, 产生一个待测奇数,不能被小于5000的素数整除
void pTabScr(BigInt& n)
{
        int i = 0;
        BigInt divisor;
        const int length = sizeof(prime) / sizeof(int);

        while (i != length)
        {
                n.Random(false);
                while (!n.IsOdd())
                        n.Random(false);

                i = 0;
                for (; i < length; i++)
                {
                        divisor = prime[i];
                        if ((n % divisor) == 0)
                                break;
                }
        }
}
```

3. Rabin-Miller 检测

具体 Rabin-Miller 检测的实验原理请见算法介绍和实验框图

Rabin-Miller

```
bool RabinMiller(const BigInt& n)
{
        BigInt r, a, y;
        unsigned int s, j;
        r = n - 1;
        s = 0;

        while (!r.IsOdd())
        {
                s++;
                r >> 1;
        }

        //随机产生一个小于N-1的检测数a
        a.Random(true);

        //y = a的r次幂模n
        y = Power(a, r, n);

        //检测J=2至J<S轮
        if ((!(y == 1)) && (!(y == (n - 1))))
        {
                j = 1;
                while ((j <= s - 1) && (!(y == (n - 1))))
                {
                        y = (y * y) % n;
                        if (y == 1)
                                return false;
                        j++;
                }
                if (!(y == (n - 1)))
                        return false;
        }
        return true;
}
```

(四) main 函数 (加密和解密)

根据实验原理部分和加密解密框图，利用上述函数组装起来实现 RSA 的加密和解密

RSAmain

```
int main()
{
```

```
3         genPrime();
4         ofstream outfile("RSA.txt");
5         cout << "开始生成大素数p" << endl;
6
7         //产生大素数
8         BigInt p = GeneratePrime();
9
10        //16进制形式显示
11        p.display();
12        outfile << "大素数p:" << endl;
13        outfile << p;
14        cout << endl;
15
16        cout << "开始生成素数q" << endl;
17
18        //产生大素数
19        BigInt q = GeneratePrime();
20
21        //16进制形式显示
22        q.display();
23        outfile << "大素数q:" << endl;
24        outfile << q;
25        cout << endl;
26        cout << "公钥n = p * q" << endl;
27        BigInt n = p * q;
28
29        cout << "公钥n为:  " << endl;
30        //16进制形式显示
31        n.display();
32        outfile << "公钥n为: " << endl;
33        outfile << n;
34        cout << endl;
35        cout << "公钥e和秘钥d " << endl;
36        BigInt t = (p - 1) * (q - 1);
37
38        //e为公开钥
39        BigInt e;
40
41        //d为秘密钥, 即e模t的乘法逆元
42        BigInt d;
43
44        //y用于参与扩展欧几里得运算, 存储t模e的乘法逆元
45        BigInt y;
46
47        BigInt temp;
48
49        while (1)
50        {
```

```
51                  //产生与t互质的e
52                  e.Random(false);
53                  while (!(gcd(e, t) == 1))
54                  {
55                          e.Random(false);
56                  }
57
58                  //用扩展欧几里德算法试图求出e模t的乘法逆元
59                  temp = Extended_gcd(e, t, d, y);
60
61                  //e*d模t结果为1, 说明d确实是e模t的乘法逆元
62                  temp = (e * d) % t;
63                  if (temp == 1)
64                          break;
65
66                  //否则重新生成e
67          }
68
69
70          cout << "公钥e为:  " << endl;
71          //16进制形式显示
72          e.display();
73          outfile << "公钥e:" << endl;
74          outfile << e;
75          cout << endl;
76
77          cout << "秘钥d为:  " << endl;
78          //16进制形式显示
79          d.display();
80          outfile << "秘钥d:" << endl;
81          outfile << d;
82          cout << endl;
83          cout << "随机生成明文分组m " << endl;
84          BigInt m;
85          m.Random(false);
86          cout << "明文分组m为: " << endl;
87          //16进制形式显示
88          m.display();
89          outfile << "明文分组m为: " << endl;
90          outfile << m;
91          cout << endl;
92          cout << "用秘密钥e对m加密,得到密文分组c " << endl;
93          BigInt c = Power(m, e, n);
94          cout << "密文分组c为: " << endl;
95          //16进制形式显示
96          c.display();
97          outfile << "密文分组c为: " << endl;
98          outfile << c;
```

```
99          cout << endl;
100         cout << "用公开钥d对c解密，得到明文分组m2 " << endl;
101         BigInt m2 = Power(c, d, n);
102         cout << "明文分组m2为： " << endl;
103         //16进制形式显示
104         m2.display();
105         outfile << "明文分组m2为： " << endl;
106         outfile << m2;
107         cout << endl;
108
109         system("pause");
110         return 0;
111 }
```

五、 实验结果

(一) 生成大素数

1. 生成大奇数 (未通过 Rabin-Miller 检测)

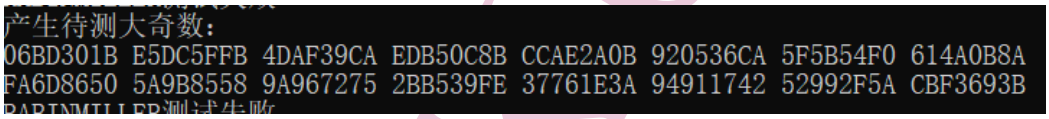

图 5: 生成大奇数

2. 生成大奇数 (通过 5 次 Rabin-Miller 检测)

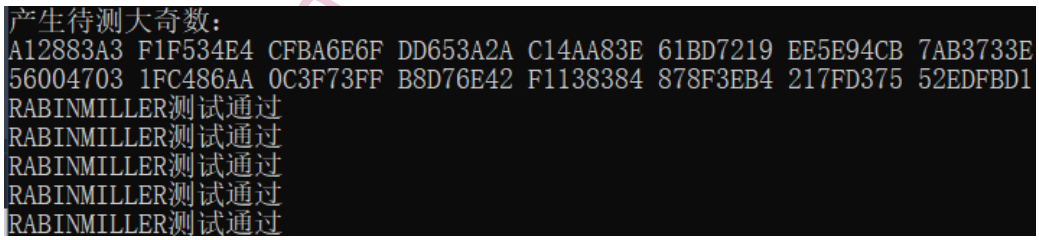

图 6: 生成大奇数 (通过 5 次 Rabin-Miller 检测)

(二) 生成公钥密钥

1. 公钥 n,e 和密钥 d

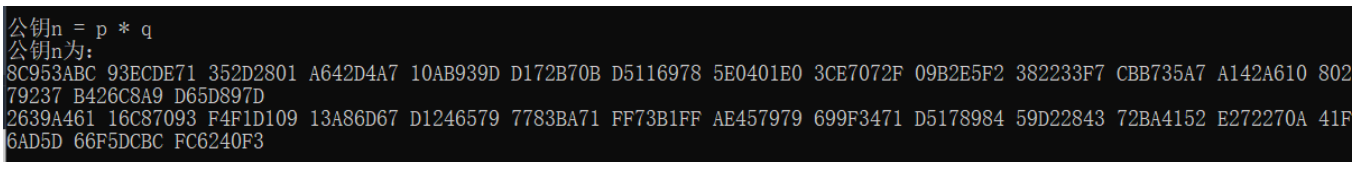

图 7: 公钥 n

```
公钥e为:
80DCBC67 E4059177 1645AC68 2F3DA60B 30C2F2C2 36787FD0 D1D286FE 04546776
64C117EE 3B03DD46 B8A5DA06 87E545BE 6284F425 0E429B3E 6CADADFE EDA0C9F3
```

图 8: 公钥 e

```
秘钥d为:
3B11D55B 63157F7C 2FC48387 87F778FA 421A719B 545736E7 E6770B2D 85961B9B 8B819E34 3573D250 6F861C81 B1B94028 EE830D50 38B
A5DDA 15750122 D1881C42
1293C132 72971492 D8EE5A52 39AE5E72 B98CB3B3 A824F320 99FAC984 9D012A48 199D53ED EFACD22C C8A353E5 32380342 5357D97D 1C9
7A7E8 9E9A4E1E B31CB75B
```

图 9: 公钥 e

(三) 加密解密

1. 生成明文分组 m

```
明文分组m为:
B77646A7 0CC7AF5B 4D27B28A 8D5CA177 0061DFD4 232DEC3D E31C3923 FEEA11CE
7667B0C9 D144898F BB7D8449 65F3FFF5 08B078CA 6B575AD2 32E5AC74 C741BE6F
```

图 10: 明文分组 m

2. 加密

```
用秘密钥e对m加密,得到密文分组c
密文分组c为:
2D9C1C3D 325E6102 395F4C3E 2CA8C14B AC2FFCBC A623A0C7 B4FE59FE 003C51AB C36AB735 3870085B 137F527C A491B384 76CB3AA6
B8EFF91C CC3449ED A9E4DCA3
75734AD1 D7767377 EA70CF79 526E1FEC 084B809C 3B22F5DA 39998CE6 F41B22C5 E5366327 F4B9D9ED F9B4A3B5 C870F645 43028E96
CFEF3AAD 2ACD0647 F8F5E97A
```

图 11: 加密

3. 解密

```
用公开钥d对c解密，得到明文分组m2
明文分组m2为:
B77646A7 0CC7AF5B 4D27B28A 8D5CA177 0061DFD4 232DEC3D E31C3923 FEEA11CE
7667B0C9 D144898F BB7D8449 65F3FFF5 08B078CA 6B575AD2 32E5AC74 C741BE6F
```

图 12: 解密

参考文献 [1]

六、 附录

本次实验相关资源已经上传至 github:https://github.com/Metetor/NKU_Cryptography

参考文献

[1] 吴世忠、宋晓龙、郭涛等译 Paul Garrett 著. *An Introduction to Cryptology*. 机械工业出版社, 2003.