南开大学
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

密码学 (1091) 第四次实验

公钥密码算法 RSA

苗发生

年级：2020 级

专业：信息安全 & 法学双学位

指导教师：古力

2022 年 12 月 2 日

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) RSA 算法

1. RSA 算法介绍

RSA 算法是目前理论和实际应用中最为成熟的和完善的公钥密码体制。RSA 用来解决对称密码的密钥分发问题。还可以用来进行数字签名来保证信息的否定与抵赖，利用数字签名较容易发现攻击者对信息的非法篡改以保证信息的完整性。RSA 的安全性依赖于大整数的因子分解的困难性，为了满足信息安全强度的需求，密钥的位数都比较多 (1024 位甚至更高)，导致幂模运算的运算量极大，成为提高 RSA 算法加解密速度的瓶颈

2. RSA 算法存在的问题

1) 产生密钥非常麻烦：受制于当前素数产生技术的限制，很难做到一次一密

2) 安全性：RSA 的安全性依赖于大数的因子分解，但并没有从理论上证明破译 RSA 的难度与大数分解难度等价，而且密码学界多数人士倾向于因子分解不是 NPC 问题。目前，人们已能分解 140 多个十进制位的大素数，这就要求使用更长的密钥。

3) 速度慢：RSA 的分组长度太大，为保证安全性，n 至少也要 512 位以上，使运算代价很高，尤其是速度很慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化

(四) 实验原理

序列密码和分组密码算法都要求通信双方通过交换密钥实现使用同一个密钥，这在密钥的管理、发布和安全性方面存在很多问题，而公钥密码算法解决了这个问题。

公钥密码算法是指一个加密系统的加密密钥和解密密钥是不同的，或者说不能用其中一个推导出另一个。在公钥密码算法的两个密钥中，一个是用于加密的密钥，它是可以公开的，称为公钥；另一个是用于解密的密钥，是保密的，称为私钥。公钥密码算法解决了对称密码体制中密钥管理的难题，并提供了对信息发送人的身份进行验证的手段，是现代密码学最重要的发明。

RSA 密码体制是目前为止最成功的公钥密码算法，它是在 1977 年由 Rivest、Shamir 和 Adleman 提出的第一个比较完善的公钥密码算法。它的安全性是建立在“大数分解和素性检测”这个数论难题的基础上，即将两个大素数相乘在计算上容易实现，而将该乘积分解为两个大素数因子的计算量相当大。虽然它的安全性还未能得到理论证明，但经过 40 多年的密码分析和攻击，迄今仍然被实践证明是安全的。

RSA 算法描述如下：

1) 公钥:

选择两个不同的大素数 p 和 q，n 是二者的乘积，即 n = pq，使

$$\psi(n) = (p-1)(q-1)$$

$\psi(n)$ 为欧拉函数，随机选取正整数 e，使其满足（e，$\psi(n)$）=1, 即 e 和 $\psi(n)$ 互素，则将（n，e）作为公钥

2) 私钥:

求出正数 d，使其满足 $e \times d \equiv 1 mod \psi(n)$, 则将 (n,d) 作为私钥

3) 加密算法

对于明文 m，由 $c \equiv m^e mod n$, 得到密文 c

4) 解密算法

对于密文 c，由 $m \equiv c^d mod n$, 得到明文 m

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。

RSA 整体工作流程如下:

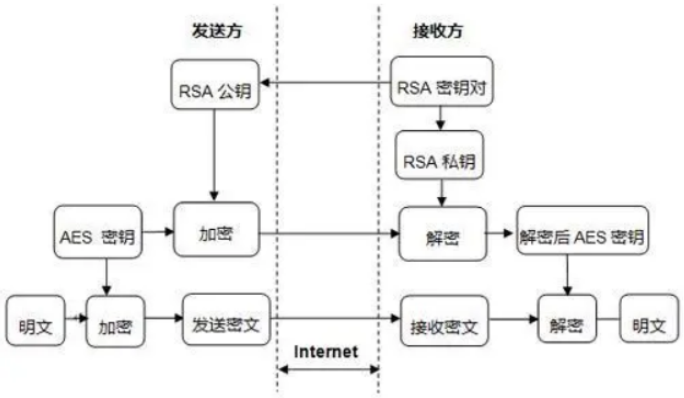


图 1: RSA 原理示意图

（五） 实验内容和步骤

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

（六） 实验报告和要求

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、 实验具体流程

（一） 手工模拟 RSA 加解密过程

给定参数 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

1. RSA 加密

由于 p 和 q 互为素数 $n = p \times q = 33, \psi(n) = (p-1)(q-1) = 20$, 取 e=3，满足

$$1 < e < \psi(n),\ gcd(\psi(n), e) = 1$$

确定满足 d·e=1mod 20 且小于 20 的 d，由于 7×3=20+1，故 d=7，因此公钥为 {3,33}, 私钥为 {7，33}

现进行加密:

$$c \equiv 2^3 = 8 \equiv mod33$$

故加密得密文为 8

2. RSA 解密

由上面计算可知，公钥为 {3,33}, 私钥为 {7，33}

现进行解密:

$$m \equiv 8^7 \equiv 2mod33$$

故解密得明文为 2

经检验，解密成功!

（二） 生成大素数

生成大素数的流程图如下图所示:

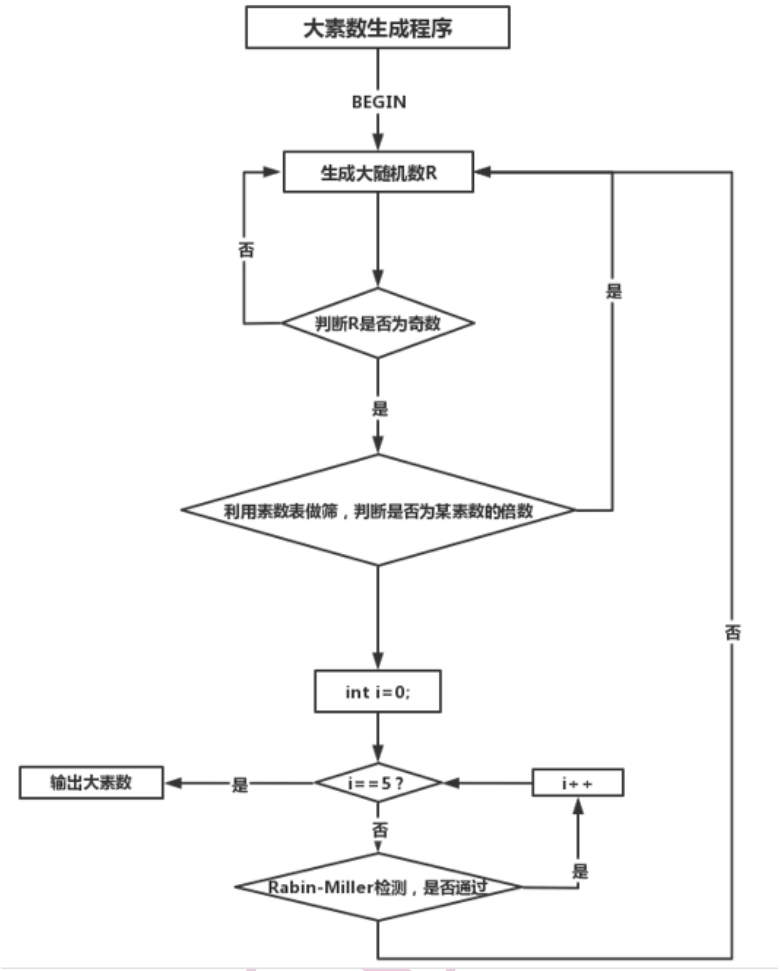


图 2: 生成大素数流程图

主要包括线性同余生成大随机奇数以及 Miller-Rabin 无论测试，接下来对其进行详细介绍:

1. 随机数原理

要生成 512 比特的素数，首先要生产随机数，此部分我们采用线性同余算法计算随机数。线性同余算法的定义如下:

$$X_{n+1} = (aX_n + c)modm$$

其中:

X 是随机序列

$m, 0 < m$, 模

$a, 0 < a < m$，乘子

$c, 0\ c < m$, 增量，也叫做偏移量

$X_0, 0\ X_0\ m$，开始值，通常叫作种子 (seed)

a,c,m 的取值是产生高质量随机数的关键，为使得随机数序列的周期尽可能大，m 应该尽可能大，普遍原则是选 m 接近等于计算机能表示的最大整数，所以这里选择 2^{31}。

评价线性同余算法的性能有以下 3 个标准:

1) 这个函数应该是一个完整周期的产生函数。也就是说，这个函数应该在重复之前产生出 0 到 m 之间的所有数

2) 产生的序列应该看起来是随机的

3) 这个函数应该用 32bit 算术高效实现

实现线性同余算法的核心 C++ 代码如下：

线性同余

```
void Prime::Linearcongruence()
{
    unsigned long long num[16];
    srand((unsigned)time(NULL));
    num[0] = rand() % (INT_MAX - 1) + 1;
    for (int i = 1; i < 16; i++)
    {
        num[i] = (num[i - 1] * LINEAR_A) % LINEAR_M;//线性同余算法
    }
    //让最高位和最低位都变成1
    num[15] |= (1 << 31) & 0x0000000011111111;
    num[0] |= 1;
    for (int i = 0; i < 16; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            number.num[i * 8 + j] = num[i] % 16;
            num[i] /= 16;
        }
    }
    return;
}
```


2. 素数检测原理

两个基本定理：

1) 费马小定理：当 p 为质数，有 $a^p-1 \equiv 1(modp)$，反过来不一定成立，也就是说，如果 a, p 互质，且 $a^p-1 \equiv 1(modp)$，不能推出 p 是质数，比如 Carmichael 数

2) 二次探测：如果 p 是素数，0<x<p，则方程 $x^2 \equiv 1(modp)$ 的解为 x=1 或 x=p−1.

Miller-Rabin 测试的逻辑如下：

```
WITNESS(a,n)
1. 将 n-1 表示为二进制形式 b_k b_{k-1} ··· b_0;
2. d←1
    for i=k downto 0 do{
        x←d;
        d←(d×d)mod n;
        if d=1 and(x≠1)and(x≠n-1) then return FALSE;
        if b_i=1 then d←(d×a)mod n}
    if d≠1 then return FALSE;
    return TRUE.
```

图 3: Miller-Rabin 测试伪代码

算法有两个输入，n 是待检验的数，a 是小于 n 的整数。如果算法的返回值为 FALSE，则 n 肯定不是素数，如果返回值为 TRUE，则 n 有可能是素数。for 循环结束后，有 $d = a^n - 1 mod n$，由 Fermat 定理知，若 n 为素数，则 d 为 1。因此若 d 1，则 n 不为素数，所以返回 FALSE。因为 n − 1≡ − 1 mod n，所以（x ≠1 ）并且（x ≠ n − 1 ）意指 $x^2 = 1$（mod n）有不在 (-1，1) 中的根，因此 n 不为素数，返回 FALSE。对 s 个不同的 a，重复使用这个算法，只要出现 false 就肯定不是素数，加 2 继续计算。如果每次都是 true，则这个数是素数的概率至少为 $1\text{-}2^{-s}$。当 s=5 时，这个概率已经达到了 96.875%，已经很大了，所以取 5。

Miller-Rabin 测试的核心 C++ 代码实现如下：

Miller-Rabin 测试

```
void Prime::MillerRabin()//米勒罗宾算法检测，这里只检测5次
{
    int a[5] = { 2, 3, 13, 17, 19 };
    int flag = 0;
    BigInt n1;
    BigInt v1, v2;
    v1.set(1);
    v2.set(2);
    BigInt v0;
    v0.set(0);
    int high;
    while (!flag)
    {
        //将number-1存入n1
        flag = 1;
        n1 = sub(number, v1);
        //得到num中的最高位
        for (high = 127; high >= 0; high--)
        {
            if (number.num[high])
                break;
        }
        BigInt temp, d;
```

```
24
25          for (int k = 0; k < 5; k++)
26          {
27              temp.set(a[k]);
28              d.set(1);//d=1
29              for (int i = high * 4 + 3; i >= 0; i--)
30              {
31                  BigInt x;
32                  x.copy(d);//x=d
33                  d = mod(mul(d, d), number);//计算d的平方mod n
34                  if (compare(d, v1) == 0 && compare(x, v1) && compare(x, n1))
                        //如果d=0,x 1 ,,x n-1,并且返回失败
35                  {
36                      flag = 0;
37                      break;
38                  }
39                  if (n1.getbit(i)) d = mod(mul(d, temp), number);
40              }
41              if (compare(d, v1)) flag = 0;
42              if (!flag) break;
43          }
44          //如果检测出不是素数, 则加2继续检测
45          if (!flag)
46          {
47              number = add(number, v2);
48              Check();
49          }
50      }
51  }
```

3. 测试

本次编写的生成随机素数的程序可以用来一直生成素数，当生成一个素数后，若输入 1 则会继续生成大素数，输入 0 则退出程序，测试截图如下：

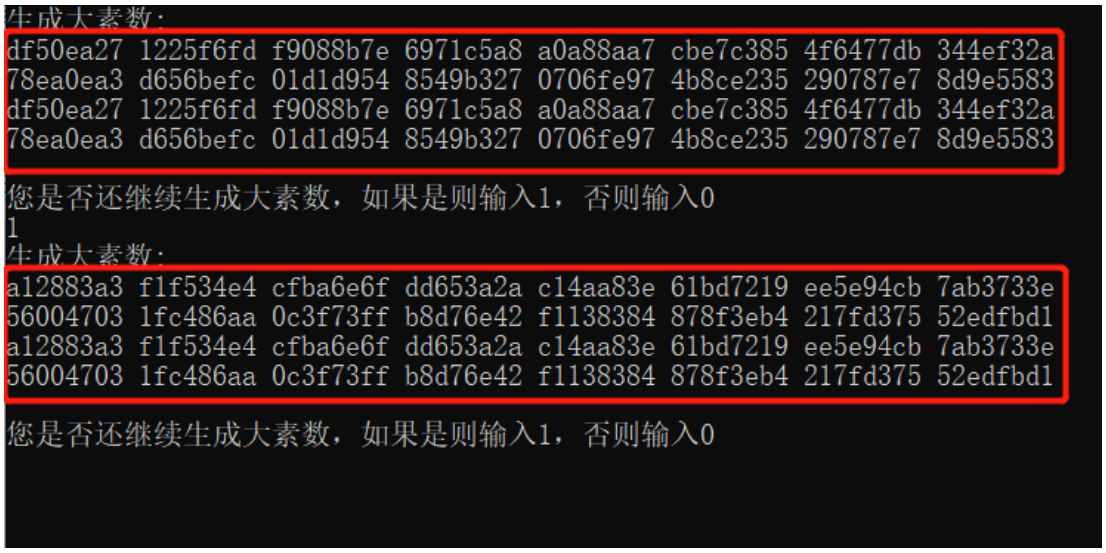


图 4: 生成大素数测试

(三) RSA 加密

1. RSA 加密整体流程

在封装好大整数类型的运算和存储后，在前面生成大素数代码的基础上进行 RSA 的加密就变得相对容易。加密算法：

对于明文 m，由 $c \equiv m^e mod n$, 得到密文 c

RSA 加密的具体流程图如下：

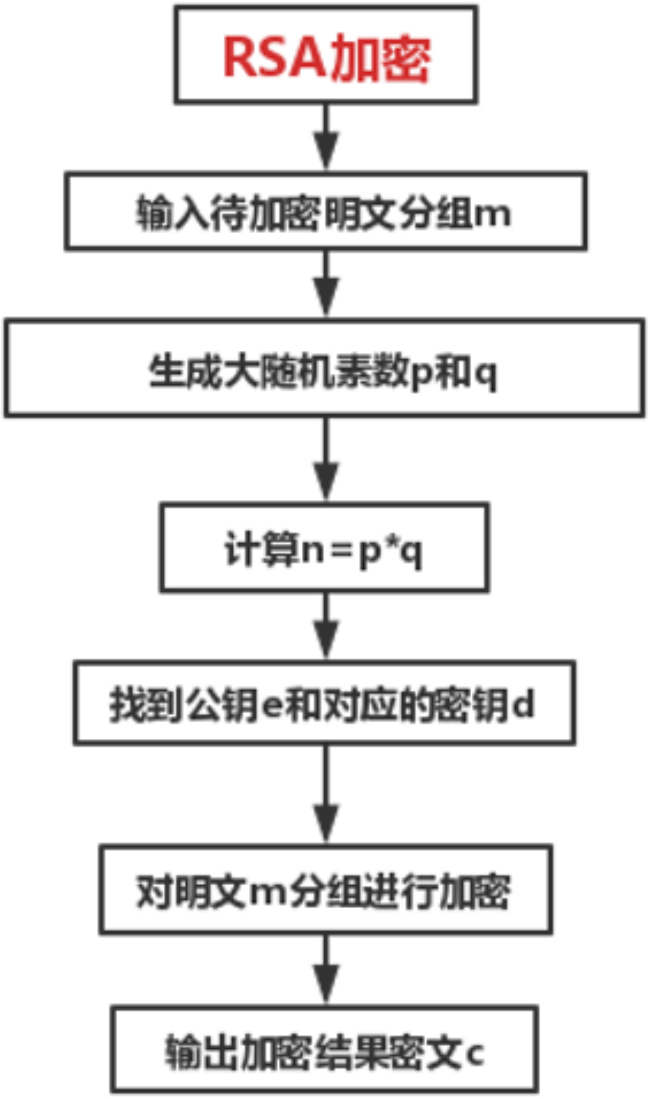


图 5: RSA 加密工作流程

2. RSA 加密关键代码

加密过程中，比较重要的步骤有生成明文 m 以及生成公钥 e
生成明文的具体代码如下：

自动生成加密明文

```
void BigInt::Input()
{
        for (int i = 0; i < (MAXSIZE / 4); i++)
                element[i] = (rand() << 17) + (rand() << 2) + rand() % 4;
};
```

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

生成公钥的具体代码如下：

自动生成公钥

```
BigInt Gene(BigInt t)
{
        BigInt temp;
        while (true)
        {
                //产生与t互质的e
                e.Random();
                while (!(Gcd(e, t) == 1))
                {
                        e.Random();
                }
                temp = ExtendedGcd(e, t, d, y);
                temp = (e * d) % t;
                if (temp == 1)
                        break;

                //否则重新生成e
        }
        return e;
}
```

3. RSA 加密测试

下面为测试 RSA 加密的截图：

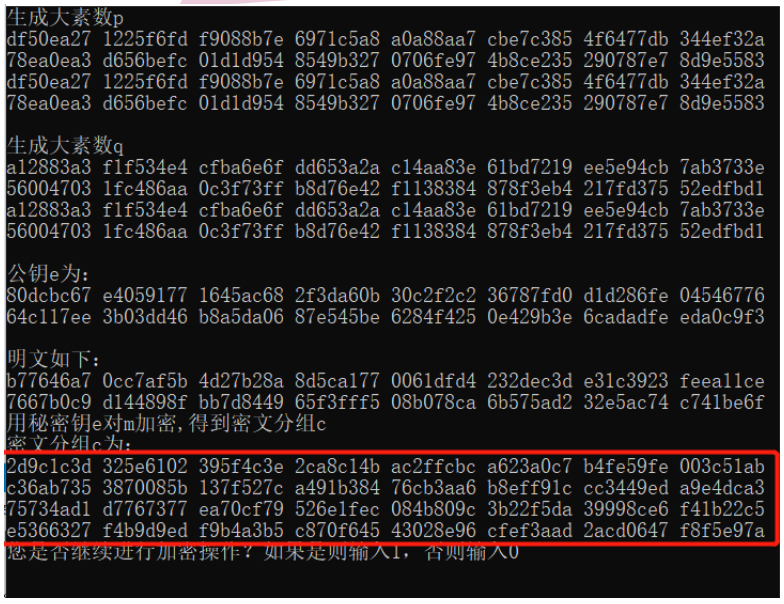


图 6: RSA 加密测试

加密结束后，根据提示，输入 1 可以继续进行加密，输入 0 则退出加密；解密过程类似，本

次实验所实现的加密和解密均可以一次运行多次计算

（四） RSA 解密

1. RSA 解密整体流程

在封装好大整数类型的运算和存储后，在前面生成大素数代码的基础上进行 RSA 的解密就变得相对容易。解密算法

对于密文 c，由 $m \equiv c^d mod n$, 得到明文 m

RSA 解密的具体流程如下：

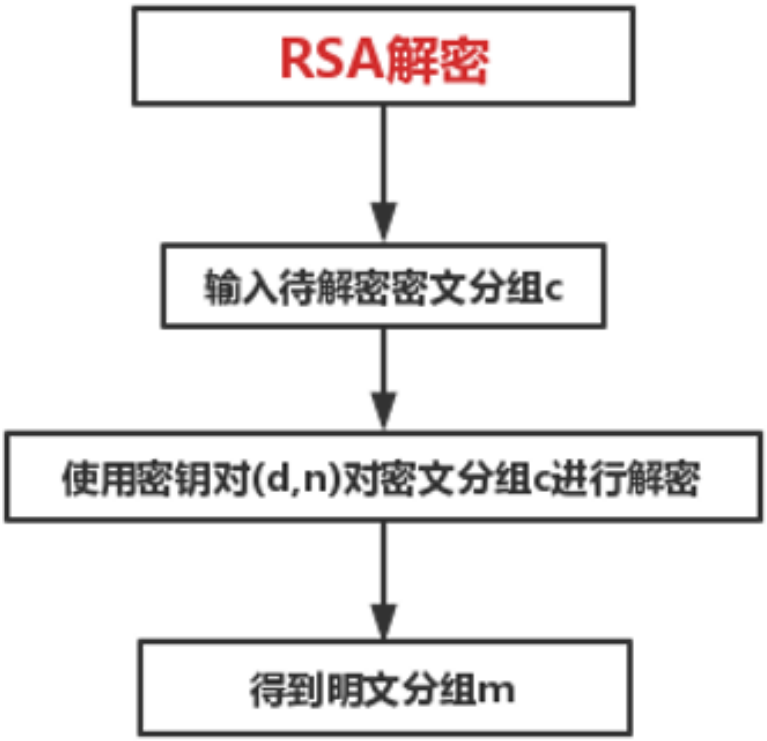


图 7: RSA 解密工作流程

2. RSA 关键解密代码

在 RSA 解密过程中，关键的代码有大整数乘法以及大整数得取 mod 运算，现将关键代码进行展示：

大整数取模运算

大整数取模运算

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

```
6
7           //检测指数p的二进制形式的每一位
8           while (!(temp <= 1))
9           {
10                  if (temp.element[0] & 1)
11                  {
12                          result = (result * base) % m;
13                  }
14                  base = (base * base) % m;
15                  temp >> 1;
16          }
17          return (base * result) % m;
18  }
```

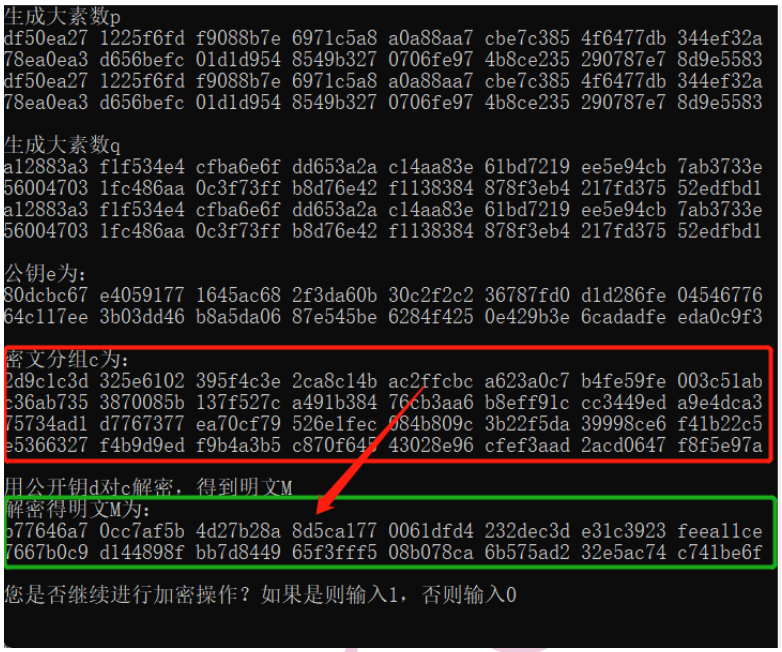
大整数乘法 C++ 代码如下：

大整数乘法

```
1           unsigned int carry;
2
3           //进行竖式乘
4           for (int i = 0; i < right.GetLength(); i++)
5           {
6                   carry = 0;
7                   //B的每一位与A相乘
8                   for (int j = 0; j < left.GetLength() + 1; j++)
9                   {
10                          sum = ((unsigned __int64)left.element[j]) * (right.
                                element[i]) + carry;
11                          if ((i + j) < MAXSIZE)
12                                  temp.element[i + j] = (unsigned int)sum;
13                          carry = (sum >> 32);
14                  }
15                  result = (temp + last);
16                  last = result;
17                  temp.Putzero();
18          }
19          //判断积的符号
20          if (left.sign == right.sign)
21                  result.sign = true;
22          else
23                  result.sign = false;
24
25          return result;
26  }
```

3. RSA 解密测试

依据 RSA 加密所用的大素数、密钥 e 以及产生的密文 c 进行解密，进而测试 RSA 加解密的正确性，测试截图如下：



通过 RSA 加密和解密的对比测试，实验成功！

（五） 使用 RSATool 工具进行测试

大素数 P：EF90AEF1D12CDB3E0EB6389B71213733

大素数 Q：B44D330DB009AC919F32FB635887F323

模 N：A8B9F94D47BF097304BE5EAB8B4F501726572390794FD4B1096EB6D9A2D4F4F9

私钥 D:544F79CFB00639C68B9C44DA1B66AA71D51C9A32C28681F02A3770C7C2B52B31

公钥 E：0x10001

参数设置如下：

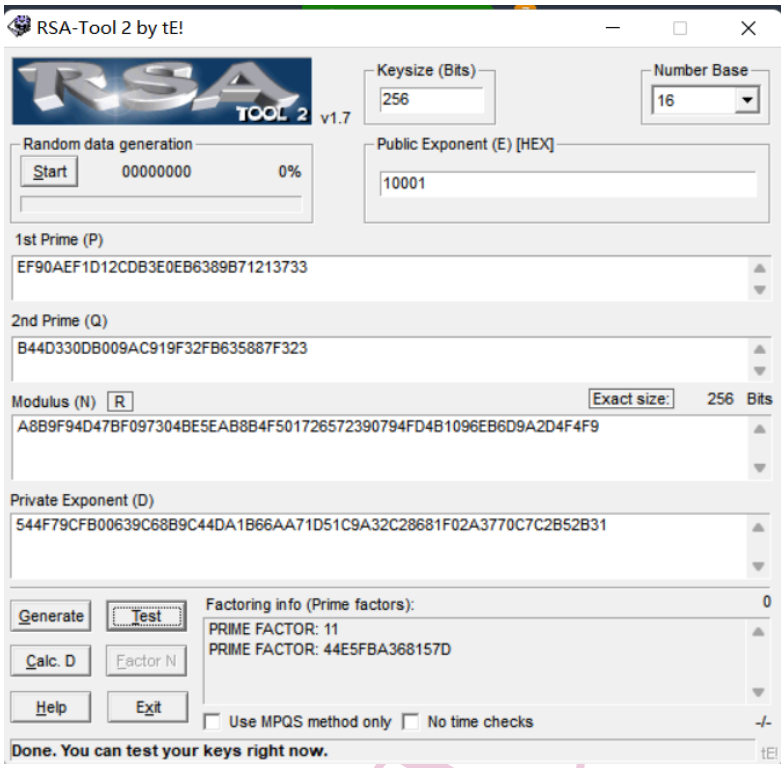


图 8: RSATool 参数设置

1. 利用 RSATool 工具进行加密

明文：I will come to Tsinghua
测试截图如下：

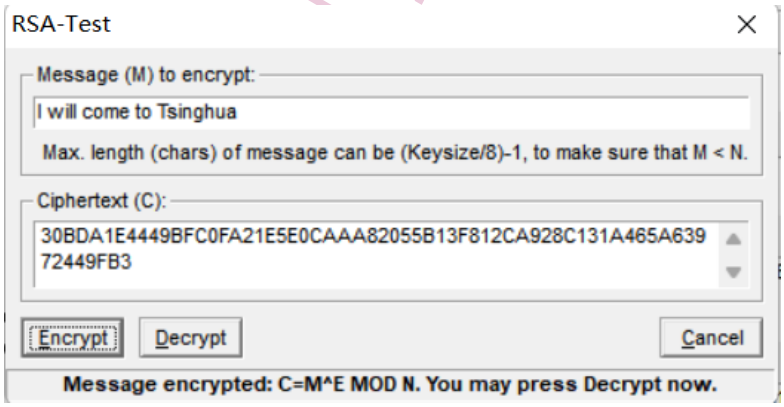


图 9: 利用 RSATool 工具进行加密

得到密文:30BDA1E4449BFC0FA21E5E0CAAA82055B13F812CA928C131A465A63972449FB3

2. 利用 RSATool 工具进行解密

利用刚刚得到的密文进行解密，观察加解密过程和结果
所用密文:30BDA1E4449BFC0FA21E5E0CAAA82055B13F812CA928C131A465A63972449FB3
测试截图：

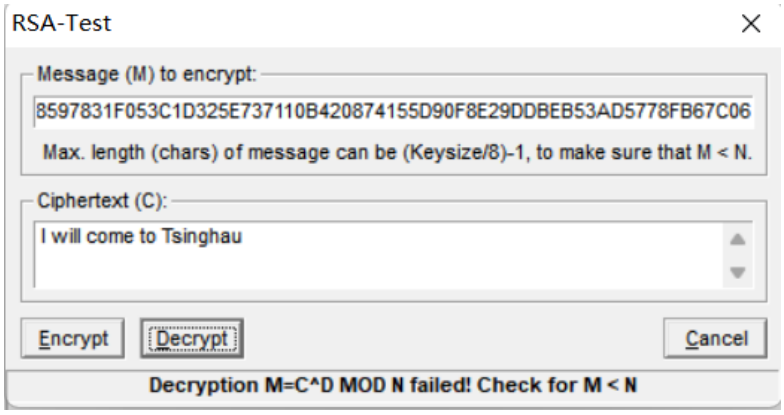


图 10: 利用 RSATool 工具进行解密

得到明文：I will come to Tsinghua.
通过比较分析，RSATool 加密和解密成功!

三、 程序优化

由于 RSA 是基于大整数的，计算相对比较耗时，为了节约时间，本次实现过程中的优化如下:

1) 尽量多的采用位运算来进行相应计算
2) 大数除除法，采用试商除法, 并利用二分查找法优化
3) 利用 Montgomery 算法快速求指数幂
4) 存储过程尽量节省空间，能用 bit 绝不用 byte
5) 提前将小素数生成后存储起来，避免运行程序的时候再生成耗时