Lab 4 实验报告

实验名称：

公钥密码算法RSA

实验内容：

1. 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

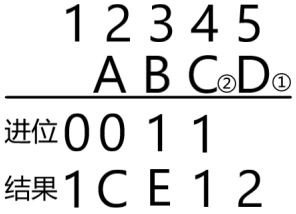
```
(1) n=p×q=33;  ϕ(n)=(p-1)×(q-1)=20
(2) 选取e=3; 1<e<ϕ(n)且gcd(e,ϕ(n))=1
(3) d×e≡1mod20 → d≡7mod20
(4)以{3,33}为公开钥，以{7,33}为秘密钥
```

2、编写一个程序，用于生成**512**比特的素数

（1）在本次实验中，如何存储长度为64bit以上大整数是一个重难点，因为C++中最大的数据结构long也只能存储64bit的数据。所以定义一个类 BigInt ，使用私有成员变量string类型存储十六进制类型的大整数字符串，并对加减乘除等运算进行函数重载。

存储数据的私有成员变量`string num`。

加法符号重载的编程实现：将最低位对其，每位对应相加并加上进位值。




```
BigInt BigInt::operator + (const BigInt& b) const
{
    return BigInt(Addition(num, b.getnum()));
```

```
}

string Addition(string a, string b)
{
    int la = a.length();
    int lb = b.length();
    int l = max(la, lb);//加法结果的长度初始设为加数a和b的最大值
    int carry = 0;//进位的值
    string result(l, 0);//初始化结果字符串
    for (int i = 0;i < l;i++)
    {
        //计算a和b的第i位的16进制值,从最后一个字节，最低位开始计算
        int temp_a=0, temp_b=0;
//GetCalculateNum()函数会将十六进制的字符转换成进行运算的数字，如A会转换10；B
会转换成11；SetCalculateNum()则会将数字转换成十六进制的字符
        if (i < la)
            temp_a = GetCalculateNum(a[la - 1 - i]);
        if (i < lb)
            temp_b = GetCalculateNum(b[lb - 1 - i]);
        //第i位的十六进制值为a和b的第i位十六进制值相加,在加上进位
      result[l-1-i] =SetCalculateNum((temp_a+temp_b+carry));
        carry = (temp_a + temp_b) / 16;//判断是否进位
    }
    if (carry > 0)
        result = "1" + result;
    return result;
}
```

减法重载和加法相似，都是对每个十六进制的对应位计算。但是在本次实验中，只实现了大于零的大整数的存储，没有实现负数的存储，所以减法时，要求被减数大于减数，否则结果为-1。

乘法符号重载的编程实现：

在大整数乘法中，为了加快运算速度，采用快速乘法算法。快速乘法原理为：利用乘法分配率来将a＊b转化为多个式子相加的形式求解（注意：选取a,b中较小的数将其转换成二进制）。

例如：`20×14 = 20×0b1110 = 20×(2^3)×1 + 20×(2^2)×1+20×(2^1)×1+20×(2^0)×0 = 160+80+40=280。`

```
BigInt BigInt::operator * (const BigInt& b) const
```

```
{return BigInt(Mult(num, b.getnum()));}

string Mult(string num, string s)
{
    if (num == "0" || s == "0")//如果乘数是零，直接返回零
        return "0";
    Bit small("");
    string temp_num;
    //将两个乘数中较小值转换成二进制字符串形式，存储在Bit类中
    if (num.length() > s.length()) {
        small.setnum_bit(s);
        temp_num = num;}
    else {
        small.setnum_bit(num);
        temp_num = s;}

    string result = "";
    for (int i = small.size() - 1;i >= 0;--i)
    {//如果对应第i位的二进制字符串是1,则进行计算，将结果加上temp_num
        if (small.at(i))
            result = Addition(result, temp_num);
        //每遍历一个二进制位，temp_num变成原来的二倍
        temp_num = Addition(temp_num, temp_num);
    }
    return result;
}
```

此外，逻辑运算符号`>,<,==`等的重载，通过字符串长度和大小进行比较就可以实现。

（2）随机数生成原理：

本实验需要对大整数BigInt生成随机数，因为BigInt使用string存储其中的十六进制数据，所以可以对string中的每个十六进制字符进行随机数生成，组合成需要长度的随机数。

对每一个字符生成的时候，使用C++库函数rand()进行生成。先随机生成一个 0 至 RAND_MAX范围内的正整数，用它模16获得一个小于16的随机数，然后在十六进制字符表中查表寻找对应字符。

rand()函数实际是一个伪随机数生成器，它利用的是线性同余算法，由以下迭代公式获得到随机数数列：

$$X_n = (aX_{n-1} + c) mod(m)$$

生成随机大的奇数实现代码：

```
//生成一个大的奇数，len是奇数的bit长度
BigInt RSA::CreateOddNum(int len)//512bit的最高位应该固定为1
{
    srand(time(0));
    len = len / 4 + ((len % 4 > 0) ? 1 : 0);//因为string中使用十六进制表示，所以位数应该是4的倍数，不是将它近似为4的倍数
    char hex_table[] = {
'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F' };
    char hex_table_odd[] = { '1','3','5','7','9','B','D','F' };
    string str = "";
    //最高位固定为1
    str.push_back(hex_table[rand() % 8 + 8]);
    for (int i = 1;i < len-1;i++)
        str.push_back(hex_table[rand() % 16]);
    //最后一位确保它为奇数
    str.push_back(hex_table_odd[rand() % 8]);
    return BigInt(str);
}
```

（3）素数生成原理：首先随机选取一个大的奇数，然后用素性检验算法检验这一奇数是否为素数；如果不是，将这个大的奇数加2（或减2），重复这一过程，直到找到素数为止。

实验中使用Miller Robin这一素性检测算法，算法基于费马小定理，二次探测定理进行检测，是一种概率素数测试法。

算法原理：若n为大于2的素数，则有 n-1 = (2^k)*q，k>0，q为奇数，且 1<a<(n-1)
则满足以下二者之一的条件:

1. a^q mod n = 1;存在j在[1, k]；
2. 区间内的正整数, a^(2^(j-1))*q mod n = n-1；

若满足二者条件之一未必为素数；若二者都不满足，必为合数。

在实际编程过程中，为了提高检测素数正确性的概率，进行二十轮检测，每一轮随机生成一个a，计算a^q mod == 1和 a ^(2^(j-1))*q mod n == n-1是否成立。

实现代码：

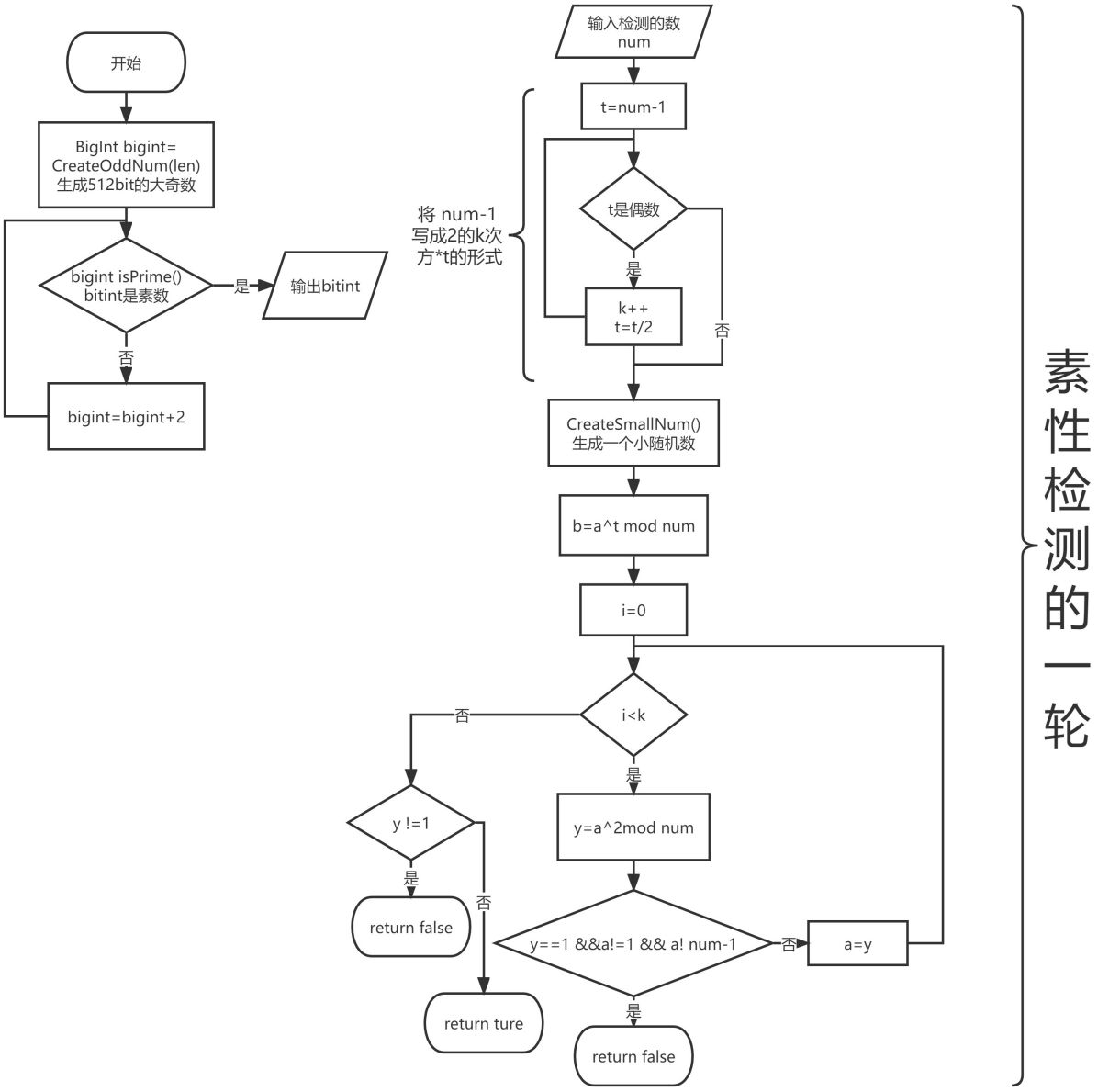
```
bool RSA::isPrime(BigInt num)
{
```

```
    srand(time(0));
    if (num == BigInt("2"))//2是素数
        return true;
    if (num < BigInt("2"))//0，1不是素数
        return false;
    BigInt num_1 = num - BigInt("1");
    Bit b(num_1);
    for (int t = 0;t < 20;t++)
    {
        BigInt a = CreateSmallNum();//生成一个小的随机数a
        BigInt d("1");
        for (int i = num.getbitnum() - 1;i >= 0;--i)
        {
            BigInt x = d;
            d = (d * d) % num;
            if (d == BigInt("1") && x != BigInt("1") && x != (num-
BigInt("1")))
                return false;

            if (b.at(i))
                d = (a * d) % num;
        }
        if (d != BigInt("1"))
            return false;
    }
    return true;
}

BigInt RSA::CreatePrime(int len) {
    BigInt result = CreateOddNum(len);//随机生成大奇数
    int count = 0;
    while (!isPrime(result))//判断大奇数是否为素数
    {//如果大奇数不是素数，素数+2继续遍历
        result = result + BigInt("2");
        count++;
        if (count > 10) {
    //如果这个大奇数基础上加上了10个20仍然没有找到素数，重新生成大奇数
            result = CreateOddNum(len);
            count = 0;
    }}
    return result;
}
```

代码流程图：

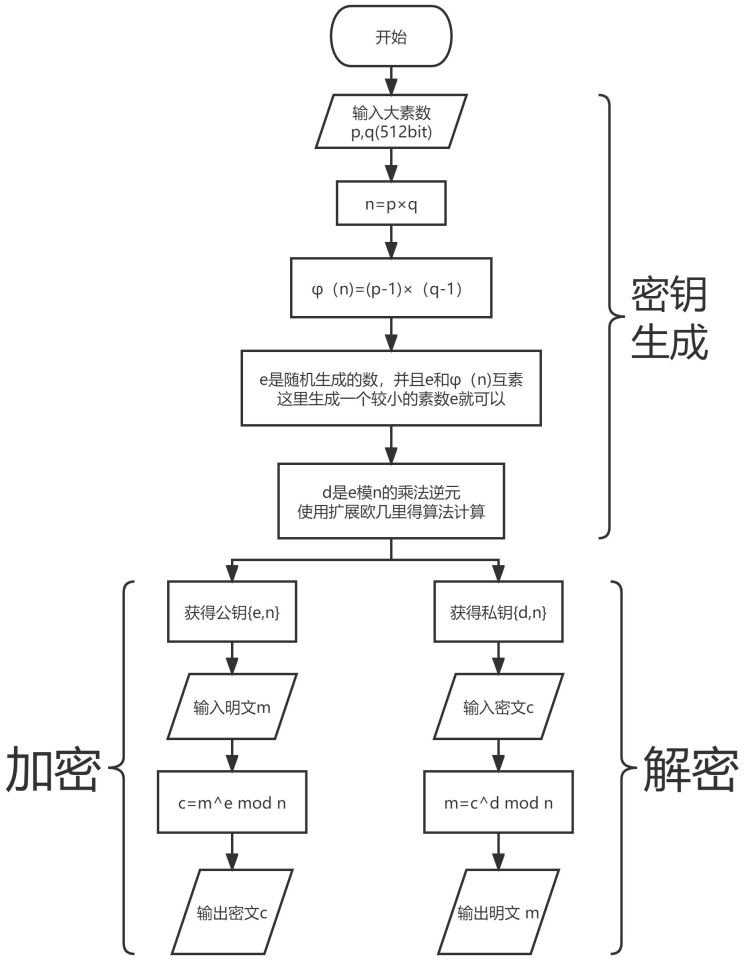


实验结果（获得的512bit的大素数）：

```
9A828710A7078455615CF1D56454038E0E3824B1C3432006AA45FA15AD24D52F1D063C53BB49DA3930ABB2ABD6020897DD85366FD7911988BCE591EE
8766F489
```

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

（1）程序框图



（2）密钥生成：

乘法逆元的生成：

使用扩展欧几里得算法生成乘法逆元。在算法实现过程中，使用递归算法，当递归到某个时候，使a等于0，认为递归结束。（实际上就是将辗转相除法中产生的式子逐个倒回去，得到 ax+by=gcd(a,b)的整数解x和y）。

代码实现：

```
void ExtendEuclid(BigInt a, BigInt b, BigInt& x, BigInt& y, const
BigInt& m) {
    if (a == BigInt("0")) {
        x = BigInt("0"), y = BigInt("1");
        return;
    }
    BigInt c = b / a, d = b % a;
    ExtendEuclid(d, a, y, x, m);
    x = (x + m - (c * y) % m) % m;
}
```

实验结果：

使用大素数p和q：

p=0xF658394908C497875F21EC86A36293FC4D6931CE102D2F6BE09DA0276DA3DBAAF90BF4834906C1821B1515ABFAE8A0608CC9956743B537661C5FA5B7C10684DB；

q=0xEAA1645713B01E0B797E969A26D393CDCBBC1AF6AF8DA45E2BF5DFFD8FFF8B300AC3B93E4A3C73DF98CCD3C2F3CE7673EEE06A3044B1544177682C3AB350123B;

使用程序获得公钥为：

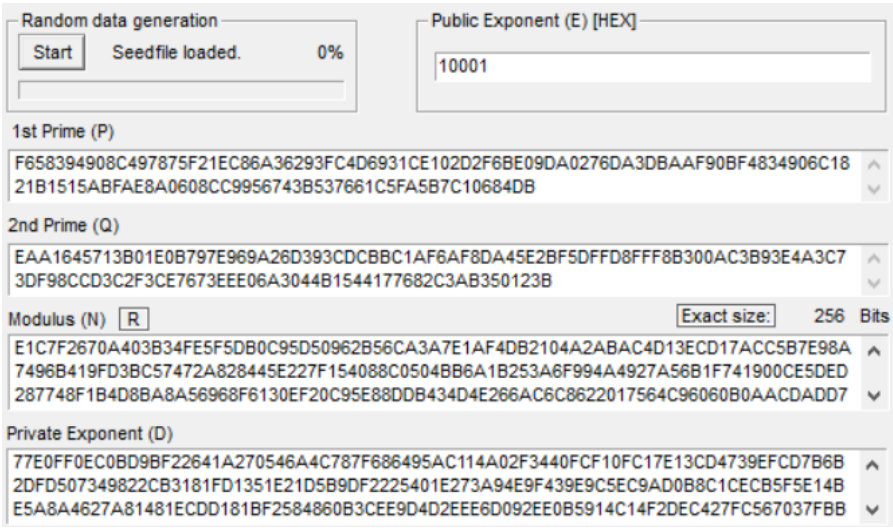
e=0x10001；'n=0xE1C7F2670A403B34FE5F5DB0C95D50962B56CA3A7E1AF4DB2104A2ABAC4D13ECD17ACC5B7E98A7496B419FD3BC57472A828445E227F154088C0504BB6A1B253A6F994A4927A56B1F741900CE5DED287748F1B4D8BA8A56968F6130EF20C95E88DDB434D4E266AC6C8622017564C96060B0AACDADD79ADC1C295DBB7897480479

私钥中的d为：

77E0FF0EC0BD9BF22641A270546A4C787F686495AC114A02F3440FCF10FC17E13CD4739EFCD7B6B2DFD507349822CB3181FD1351E21D5B9DF2225401E273A94E9F439E9C5EC9AD0B8C1CECB5F5E14BE5A8A4627A81481ECDD181BF2584860B3CEE9D4D2EEE6D092EE0B5914C14F2DEC427FC567037FBB1E044F044FEA89D0195

```
请选择操作模式：1：明文加密；2：密文解密；3：获取密钥对；4：生成大素数
3
公开钥：
e:10001
n:E1C7F2670A403B34FE5F5DB0C95D50962B56CA3A7E1AF4DB2104A2ABAC4D13ECD17ACC5B7E98A7496B419FD3BC57472A828445E227F154088C0504
BB6A1B253A6F994A4927A56B1F741900CE5DED287748F1B4D8BA8A56968F6130EF20C95E88DDB434D4E266AC6C8622017564C96060B0AACDADD79ADC
1C295DBB7897480479
秘密钥：
d:77E0FF0EC0BD9BF22641A270546A4C787F686495AC114A02F3440FCF10FC17E13CD4739EFCD7B6B2DFD507349822CB3181FD1351E21D5B9DF22254
01E273A94E9F439E9C5EC9AD0B8C1CECB5F5E14BE5A8A4627A81481ECDD181BF2584860B3CEE9D4D2EEE6D092EE0B5914C14F2DEC427FC567037FBB1
E044F044FEA89D0195
n:E1C7F2670A403B34FE5F5DB0C95D50962B56CA3A7E1AF4DB2104A2ABAC4D13ECD17ACC5B7E98A7496B419FD3BC57472A828445E227F154088C0504
BB6A1B253A6F994A4927A56B1F741900CE5DED287748F1B4D8BA8A56968F6130EF20C95E88DDB434D4E266AC6C8622017564C96060B0AACDADD79ADC
1C295DBB7897480479
```

和给出的RSA Tool生成的结果一致，说明程序正确



（3）加密函数

在获取公钥{e，n}之后，加密过程就是对明文m进行以下运算：

$$c = m^e mod(n)$$

实现代码：

```
//指数求模运算，a的b次方模m
BigInt Moden(const BigInt& a,const BigInt& b, const BigInt& m)
{
    if (b == BigInt("0")) return BigInt("1");
    if (b == BigInt("1")) return (a % m);
    BigInt hb = b/BigInt("2");
    BigInt phb = Moden(a, hb, m);
    BigInt res = (phb * phb) % m;
    if (!b.isEven())
        res = (a * res) % m;
    return res;
}

BigInt RSA::Encrypt(BigInt m, BigInt e, BigInt n) {
    return Moden(m, e, n);
}
```

（4）解密函数

在获取私钥{d，n}之后，解密过程就是对密文c进行以下运算：

$$m = c^d mod(n)$$

```
BigInt RSA::Decrypt(BigInt c, BigInt d, BigInt n) {
    return Moden(c,d, n);
}
```

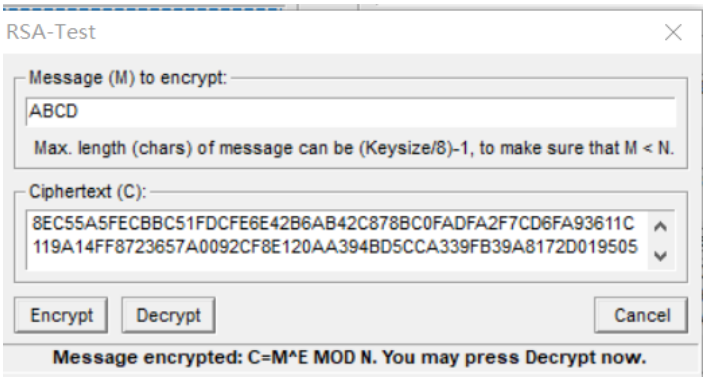
实验结果：

使用上文生成的公钥，加密16进制明文`41424344`。获得的密文为`8EC55A5FECBBC51FDCFE6E42B6AB42C878BC0FADFA2F7CD6FA93611C119A14FF8723657A0092CF8E120AA394BD5CCA339FB39A8172D01950592395BEF6D1D47EE651290334083EBB012B00997EA8812A6BF5D03EAF3B143E2AE6F18A8607CC26E3D9185BE2F51D55DB2D6DE15797EA0E60F182B9284EBC028A081353D681F01C`；

使用密钥解密，可以成功获得`41424344`。

```
请选择操作模式：1：明文加密；2：密文解密；3：获取密钥对
1
输入十六进制明文数据:41424344
输入公钥对:输入e:10001
输入n:E1C7F2670A403B34FE5F5DB0C95D50962B56CA3A7E1AF4DB2104A2ABAC4D13ECD17ACC5B7E98A7496B419FD3BC57472A828445E227F154088
0504BB6A1B253A6F994A4927A56B1F741900CE5DED287748F1B4D8BA8A56968F6130EF20C95E88DDB434D4E266AC6C8622017564C96060B0AACDADD
9ADC1C295DBB7897480479
8EC55A5FECBBC51FDCFE6E42B6AB42C878BC0FADFA2F7CD6FA93611C119A14FF8723657A0092CF8E120AA394BD5CCA339FB39A8172D01950592395BE
F6D1D47EE651290334083EBB012B00997EA8812A6BF5D03EAF3B143E2AE6F18A8607CC26E3D9185BE2F51D55DB2D6DE15797EA0E60F182B9284EBC0
8A081353D681F01C
请选择操作模式：1：明文加密；2：密文解密；3：获取密钥对
2
输入十六进制密文数据:8EC55A5FECBBC51FDCFE6E42B6AB42C878BC0FADFA2F7CD6FA93611C119A14FF8723657A0092CF8E120AA394BD5CCA339FE
39A8172D01950592395BEF6D1D47EE651290334083EBB012B00997EA8812A6BF5D03EAF3B143E2AE6F18A8607CC26E3D9185BE2F51D55DB2D6DE157
7EA0E60F182B9284EBC028A081353D681F01C
输入私钥对:输入d:77E0FF0EC0BD9BF22641A270546A4C787F686495AC114A02F3440FCF10FC17E13CD4739EFCD7B6B2DFD507349822CB3181FD13
1E21D5B9DF2225401E273A94E9F439E9C5EC9AD0B8C1CECB5F5E14BE5A8A4627A81481ECDD181BF2584860B3CEE9D4D2EEE6D092EE0B5914C14F2DE
427FC567037FBB1E044F044FEA89D0195
输入n:E1C7F2670A403B34FE5F5DB0C95D50962B56CA3A7E1AF4DB2104A2ABAC4D13ECD17ACC5B7E98A7496B419FD3BC57472A828445E227F154088
0504BB6A1B253A6F994A4927A56B1F741900CE5DED287748F1B4D8BA8A56968F6130EF20C95E88DDB434D4E266AC6C8622017564C96060B0AACDADD
9ADC1C295DBB7897480479
41424344
```

使用RSA Tool加密明文`ABCD`（对应16进制0x41424344），获得的密文和使用程序加密结果相同，说明程序加密解密正确。



4.实验缺点和不足

本次实验，生成一个大素数和密文解密的过程耗费的时间都比较长，检测一个512bit的数是否为大素数平均需要花费40s，所以可以对指数求模算法进行进一步优化，以加快程序执行时间。