杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

《密码学实验》

实验报告

实验 5：RSA 密码算法编程实验

姓　　名：易涛

学　　号：16031330

专　　业：信息安全

实验时间：2018.06.04

指导老师：吕秋云

一．实验目的

实验环境：Windows10 Visual Studio 2017

实验目的：

掌握 RSA 密码加解密原理，并利用 Visual C++编程实现

加密原理：

RSA密钥生成

- 秘密选取两个相同比特长度的素数p和q
- 计算$N=p\cdot q$
- 计算$\varphi(N)=(p-1)(q-1)$，其中$\varphi(N)$是N的欧拉函数
- 选取正整数e，满足$1<e<\varphi(N)$，且$(e,\varphi(N))=1$
- 计算正整数d，满足$ed\equiv 1(\mathrm{mod}\ \varphi(N))$

加密

- 明文$m\in Z_N^*$

- 密文$c=m^e(mod\ N)$

解密

- 密文$c\in Z_N^*$

- 解密后明文$m'=c^d(\mathrm{mod}\ N)$

二．实验内容及实现过程步骤

1、因为 RSA 的安全性基于数论和计算复杂性理论中的大数分解问题，所以首先需要定义足够大的大整数数据结构。采用 256 进制进行运算

```
typedef struct Bigint {
    unsigned char num[SIZE];
};

typedef struct Bigint2 {    //运算结果数组长度最多为两倍
    unsigned char num[2 * SIZE];
};
```

2、基于大整数类型，依次实现大整数的四则运算、模运算、幂运算等函数

```
Bigint Add(Bigint a, Bigint b)
{
    //修改
    Bigint c;
    unsigned short temp;            //加法和可能会有进位，8位不够用，故定义short 16位
    //unsigned long long temp;
    unsigned char carry = 0;        //初始进位为0;
    //unsigned int carry = 0;
    for (int i = 0; i < SIZE; i++)
    {
        temp = a.num[i] + b.num[i] + carry;
        c.num[i] = temp & 0x00ff;   //低8位
        //c.num[i] = temp & 0xffffffff;
        carry = (temp >> 8) & 0xff; //高于8位的全是进位，进位范围8位足够，不会溢出
        //carry = (temp >> 32) & 0xffffffff;
    }
    return c;
}
```

```
Bigint Sub(Bigint a, Bigint b)
{
    //修改
    if (Compare(a, b) == -1) {
        return  a;              //  a < b时返回a
    }
    Bigint c;
    short temp; //结果用short 16位有符号数表示，要处理借位，使用有符号运算，用补码表示负数再处理
    //long long temp;

    unsigned char carry = 0;//借位
    //unsigned int carry = 0;
    for (int i = 0; i < SIZE ; i++)             //错误，SIZE写成SIZE-1，造成死循环
    {
        temp = a.num[i] - b.num[i] - carry; //减去前面低位的借位,如果是负数的话，temp为有符号数，用补码表示，低八位相当于加上了256
        c.num[i] = temp & 0x00ff;               //结果等于低8位
        //c.num[i] = temp & 0xffffffff;
        carry = (temp >> 15) & 0x01;            //最高位为符号位，temp如果是负数，则最高位1，产生借位，借位最多借一位
        //carry = (temp >> 63) & 0x1;
    }
    return c;
}
```

```
Bigint2 Mul(Bigint a, Bigint b)
{
    //修改
    Bigint2 c = { 0 };//初始化
    unsigned short temp;    //临时积
    //unsigned long long temp;
    unsigned char carry;
    //unsigned int carry;
    for (int i = 0; i < SIZE; i++)
    {
        carry = 0;
        for (int j = 0; j < SIZE; j++)
        {
            temp = a.num[i] * b.num[j] + c.num[i + j] + carry;      //模拟乘法列式运算，对照计算过程编写
            c.num[i + j] = temp & 0x00ff;               //a.num[i]*b.num[j]结果放在c.num[i+j]
            //c.num[i + j] = temp & 0xffffffff;
            carry = (temp >> 8) & 0xff;         //乘法进位最大为255,因此用unsigned char型
                                                //a,b都需经初始化置0再参与运算
            //carry = (temp >> 32) & 0xffffffff;
        }
    }
    c.num[SIZE * 2 - 1] = carry;//若是积太大，最高进位循环内未处理，则此处处理
    return c;
}
```

```
Bigint Div(Bigint a, Bigint b)
{
    Bigint B = { 0 };
    Bigint c = { 0 };
    int len = getLength(a) - getLength(b);
    while (len >= 0)
    {
        B = MoveLeft(b, len);    //临时除数，从高位开始除
        while (Compare(a, B) >= 0)
        {
            a = Sub(a, B);          //更新被除数
            c.num[len]++;           //通过减法得到对应位数的商
        }
        len--;                      //除完一次，len减1
    }
    return c;
}
```

```
Bigint Mod(Bigint a, Bigint b)
{
    if (Compare(a, b) < 0)
        return a;        //a<b返回a
    else
    {
        Bigint B = { 0 };
        int len = getLength(a) - getLength(b);
        while (len >= 0)
        {
            B = MoveLeft(b, len);        //提高效率，减一个相对大的b的倍数
            while (Compare(a, B) >= 0)
                a = Sub(a, B);
            len--;
        }
        return a;

    }
}
```

```
bool Inverse(Bigint e, Bigint N, Bigint &d)          //e模N的逆，存在d中
{
    Bigint r1 = { 0 };
    Bigint r2 = { 0 };
    Copy(r1, e);
    Copy(r2, N);
    Bigint s1 = { 1 };
    Bigint s2 = { 0 };
    Bigint s = { 0 };
    Bigint r = { 0 };
    Bigint q = { 0 };

    while (1)
    {
        if (getLength(r1) == 0)
            return 0;
        if (getLength(r1) == 1 && r1.num[0] == 1)
        {
            Copy(d, s1);
            return 1;
        }
        q = Div(r1, r2);
        s = Sub2Mod(s1, MulMod(q, s2, N), N);
        r = Sub(r1, Narrow(Mul(q, r2)));
        Copy(r1, r2);
        Copy(s1, s2);
        Copy(s2, s);
        Copy(r2, r);
    }
}
```

3、解决大整数的问题之后，还需随机生成大素数，采用多次 Miller-Rabin 素性检测算法实现。

```
Bigint BigRand(Bigint n)              //生成0-n之间的数
{
    //修改
    Bigint res = { 0 };
    for (int i = 0; i < SIZE; i++)
        res.num[i] = rand() % 256;
    //res.num[i] = rand() % 4294967296;
    res = Mod(res, n);
    return res;
}
Bigint BigRand(int bytes)             //生成位数为8*bytes的随机数
{
    //修改
    Bigint res = { 0 };
    for (int i = 0; i < bytes - 1; i++)
        res.num[i] = rand() % 256;
        //res.num[i] = rand() % 4294967296;
    res.num[bytes - 1] = 128 + rand() % 128;          //最高位置1
    //res.num[bytes - 1] = 2147483648 + rand() % 2147483648;
    return res;
}
```

```
Bigint BigRandOdd(int bytes)          //生成随机奇数
{
    //修改
    Bigint res = { 0 };
    for (int i = 0; i < bytes - 1; i++)
        res.num[i] = rand() % 256;

    res.num[bytes - 1] = 128 + rand() % 128;
    //res.num[bytes - 1] = 2147483648 + rand() % 2147483648;
    if (!(res.num[0] & 0x01))             //最低位为0,为二的倍数，加一处理
        res.num[0] = res.num[0] + 1;
    return res;
}
```

```
bool MillerRabinOnce(Bigint &n)
{
    Bigint b, m, v, temp;
    Bigint j = { 0 };
    Bigint one = { 1 };
    Bigint two = { 2 };
    Bigint three = { 3 };
    m = Sub(n, one);
    while (!(m.num[0] & 0x01))
    {
        j = Add(j, one);                    //计算m,j,使得n-1 = 2^j m
        BitMoveRight(m);                    //n-1除二j+1，至m为奇数为止
    }
    b = Add(two, BigRand(Sub(n, three)));//选取2-n-1的随机数
    v = PowMod(b, m, n);                    //计算v = b^m mod n
    if (Compare(v, one) == 0)
        return 1;
    Bigint i = { 1 };
    temp = Sub(n, one);
    while (Compare(v, temp) < 0)
    {
        if (Compare(i, j) == 0)
            return 0;
        v = MulMod(v, v, n);
        i = Add(i, one);
    }
    return 1;

}
```

```
bool MillerRabin(Bigint &n, int loop)
{
    for (int i = 0; i < loop; i++)
    {
        if (!MillerRabinOnce(n))
            return 0;
    }
    return 1;
}
```

```
Bigint GenPrime(int bytes)
{
    Bigint res = BigRandOdd(bytes);
    int loop = 20;
    while (!MillerRabin(res, loop))
    {
        res = BigRandOdd(bytes);
    }
    return res;
}
```

三．实验小结

1> 实验中遇到的错误：

代码编写完成之后，运行程序，发现随机生成大素数时，出现死循环，结果运算不出来，于是进行断点调试，逐函数运行，一步步筛查死循环位置。发现在进行除法时，更新被除数的时候，被除数总是大于除数，造成死循环。

```
Bigint2 MoveLeft(Bigint2 a, int len){ ... }
Bigint Div(Bigint a, Bigint b)
{
    Bigint B = { 0 };
    Bigint c = { 0 };
    int len = getLength(a) - getLength(b);
    while (len >= 0)
    {
        B = MoveLeft(b, len);     //临时除数，从高位开始除
        while (Compare(a, B) >= 0)
        {
            a = Sub(a, B);            //更新被除数
            c.num[len]++;             //通过减法得到对应位数的商
        }
        len--;                        //除完一次，len减1
    }
    return c;
}
```

进一步观察，发现进行减法更新被除数时出错。在进行减法时，Bigint 型里依位进行减法，但在代码编写时，将循环条件写错，造成最高位没有进行减法操作，导致，Div 判断 while 循环的时候，被除数总是大于除数。

```
Bigint Sub(Bigint a, Bigint b)
{
    //修改
    if (Compare(a, b) == -1) {
        return  a;              //  a < b时返回a
    }
    Bigint c;
    short temp; //结果用short 16位有符号数表示，要处理借位，使用有符号运算，用补码表示负数再处理
    //long long temp;

    unsigned char carry = 0;//借位
    //unsigned int carry = 0;
    for (int i = 0; i < SIZE ; i++)             //错误，SIZE写成SIZE-1，造成死循环
    {
        temp = a.num[i] - b.num[i] - carry; //减去前面低位的借位,如果是负数的话，temp为有符号数，用补码表示，低八位相当
        c.num[i] = temp & 0x00ff;               //结果等于低8位
        //c.num[i] = temp & 0xffffffff;
        carry = (temp >> 15) & 0x01;            //最高位为符号位，temp如果是负数，则最高位1，产生借位，借位最多借一位
        //carry = (temp >> 63) & 0x1;
    }
    return c;
}
```

2> 实验总结

本次实验为 RSA 密码实验，需要自己定义大整数类型，还需自己编写相应运算函数，较为复杂，但也收获丰富，深入了解了如何利用计算机模拟数学运算，并且实现了 Miller-Rabin 多次素性检测算法。