****

**密码学实验**

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全学院 |
| **专 业** | 信息安全 |
| **班 级** | 17273612 |
| **学 号** | 17271229 |
| **学生姓名** | 杨尽能 |
| **教师姓名** | 吕秋云 |
| **完成日期** | 2019.6.10 |
| **成 绩** |  |

**实验一 SM3密码杂凑算法**

* 1. **实验目的**

掌握SM3密码杂凑算法原理，并利用Visual C++实现。

* 1. **实验原理和内容概述**

利用SM3密码杂凑算法，对输入的明文进行哈希摘要。扩展要求实现对文件进行哈希摘要。

* 1. **实验环境**

Windows10，Visual Studio2017，C++。

* 1. **实验主要步骤及实验结果记录（含截图，关键源码，文字解释等）**

1. **算法核心思想：**

（1）总的来说，SM3算法核心就是对长度为l(l < 2^64 ) 比特的消息m，经过填充和迭代压缩，生成杂凑值，杂凑值长度为256比特。

（2）首先因为长度最多可以到2^64比特，这个数已经很大了，所以基本上明文可以随便输。

（3）然后来考虑分组。因为SM3一次是对512比特进行处理，所以要先对输入的明文进行分组，每组512比特。

（4）因为明文不一定刚好是512的倍数，所以需要对余下的比特串后进行填充。填充方法是先添加一个“1”，在添加k个0满足l+1+k=448mod512。最后在添加一个64位比特串，该比特串是长度l的二进制表示。经过填充后，消息m1变成比特长度为512的倍数。

（5） 就可以对m1进行分组处理，这里要注意的一点是消息处理过程中字的存储方式一定为大端（big-endian）格式，若不是，就需要对其进行翻转操作使其变为大端；之后对每一组进行迭代、扩展、压缩的操作，最后便得到了杂凑值。

1. **核心代码及注释：**

//反转四字节整型字节序

unsigned int \*ReverseWord(unsigned int \*word)

{

unsigned char \*byte, temp;

byte = (unsigned char \*)word;

temp = byte[0];

byte[0] = byte[3];

byte[3] = temp;

temp = byte[1];

byte[1] = byte[2];

byte[2] = temp;

return word;

}

//初始化函数

void SM3Init(SM3Context \*context)

{

context->Hash[0] = 0x7380166F;

context->Hash[1] = 0x4914B2B9;

context->Hash[2] = 0x172442D7;

context->Hash[3] = 0xDA8A0600;

context->Hash[4] = 0xA96F30BC;

context->Hash[5] = 0x163138AA;

context->Hash[6] = 0xE38DEE4D;

context->Hash[7] = 0xB0FB0E4E;

}

//处理消息块

void SM3Process\_Yanjinneng(SM3Context \*context)

{

int i;

unsigned int W[68];

unsigned int W1[64];

unsigned int A, B, C, D, E, F, G, H, SS1, SS2, TT1, TT2;

/\* 消息扩展 \*/

for (i = 0; i < 16; i++)

{

W[i] = \*(unsigned int \*)(context->messageBlock + i \* 4);//将消息分组划分为16个字

if (IsLittleEndian())

ReverseWord(W + i);

}

for (i = 16; i < 68; i++)

{

W[i] = P1(W[i - 16] ^ W[i - 9] ^ LeftRotate(W[i - 3], 15))

^ LeftRotate(W[i - 13], 7)

^ W[i - 6];

}

for (i = 0; i < 64; i++)

{

W1[i] = W[i] ^ W[i + 4];

}

/\* 消息压缩 \*/

A = context->Hash[0];

B = context->Hash[1];

C = context->Hash[2];

D = context->Hash[3];

E = context->Hash[4];

F = context->Hash[5];

G = context->Hash[6];

H = context->Hash[7];

for (i = 0; i < 64; i++)

{

SS1 = LeftRotate((LeftRotate(A, 12) + E + LeftRotate(T(i), i)), 7);

SS2 = SS1 ^ LeftRotate(A, 12);

TT1 = FF(A, B, C, i) + D + SS2 + W1[i];

TT2 = GG(E, F, G, i) + H + SS1 + W[i];

D = C;

C = LeftRotate(B, 9);

B = A;

A = TT1;

H = G;

G = LeftRotate(F, 19);

F = E;

E = P0(TT2);

}

context->Hash[0] ^= A;

context->Hash[1] ^= B;

context->Hash[2] ^= C;

context->Hash[3] ^= D;

context->Hash[4] ^= E;

context->Hash[5] ^= F;

context->Hash[6] ^= G;

context->Hash[7] ^= H;

}

//sm3算法主函数

unsigned char \*SM3Calc\_Yangjinneng(const unsigned char \*message,

unsigned int messageLen, unsigned char digest[SM3\_HASH\_SIZE]) {

SM3Context context;

unsigned int i, remainder, bitLen;

//初始化

SM3Init(&context);

//对前面的消息分组进行处理

for (i = 0; i < messageLen / 64; i++)//分组

{

memcpy(context.messageBlock, message + i \* 64, 64);//把每一个分组内容分别传入消息块中

SM3Process\_Yanjinneng(&context);

}

//填充消息分组并处理

bitLen = messageLen \* 8;

if (IsLittleEndian())

ReverseWord(&bitLen);

remainder = messageLen % 64;//余数

memcpy(context.messageBlock, message + i \* 64, remainder);//余数存到下一个消息块

context.messageBlock[remainder] = 0x80;//后补1

if (remainder < 56)//如果不足448比特

{

memset(context.messageBlock + remainder + 1, 0, 64 - remainder - 1 - 8 + 4);

memcpy(context.messageBlock + 64 - 4, &bitLen, 4);

SM3Process\_Yanjinneng(&context);

}

else

{

memset(context.messageBlock + remainder + 1, 0, 64 - remainder - 1);

SM3Process\_Yanjinneng(&context);

memset(context.messageBlock, 0, 64 - 4);

memcpy(context.messageBlock + 64 - 4, &bitLen, 4);

SM3Process\_Yanjinneng(&context);

}

/\* 返回结果 \*/

if (IsLittleEndian())

for (i = 0; i < 8; i++)

ReverseWord(context.Hash + i);

memcpy(digest, context.Hash, SM3\_HASH\_SIZE);

return digest;

}

1. **扩展要求实现：**

扩展要求对文件进行SM3摘要。其实主要代码已经实现了对输入的明文进行SM3摘要的操作了，因此要实现对文件的摘要其实就是调用资源管理器，增加一个选择文件的按钮，把文件内容输入就行了。

代码如下：

void CSM3YangjinnengDlg::OnBnClickedButton3()

{

// TODO: 在此添加控件通知处理程序代码

CString path1,m,temp;

GetDlgItemText(IDC\_EDIT4, path1);

CFile file;

file.Open(path1, CFile::modeRead, NULL);

DWORD len = file.GetLength();

unsigned char\* buf = (unsigned char\*)malloc(len\*sizeof(unsigned char)), buf1[32] = { 0 };

buf[len] = 0;

file.Read(buf, len);

file.Close();

SM3Calc\_Yangjinneng(buf, len, buf1);

for (int i = 0; i < 32; i++) {

if ((i + 1) % 4 == 0)

temp.Format("%02x ", buf1[i]);

else

temp.Format("%02x", buf1[i]);

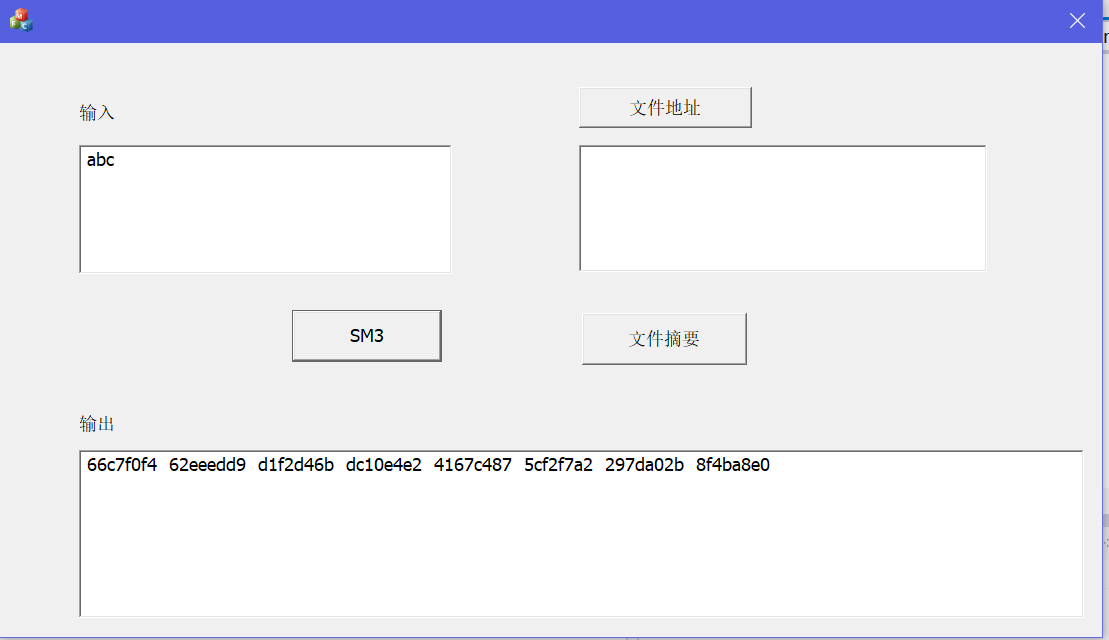
m += temp;

}

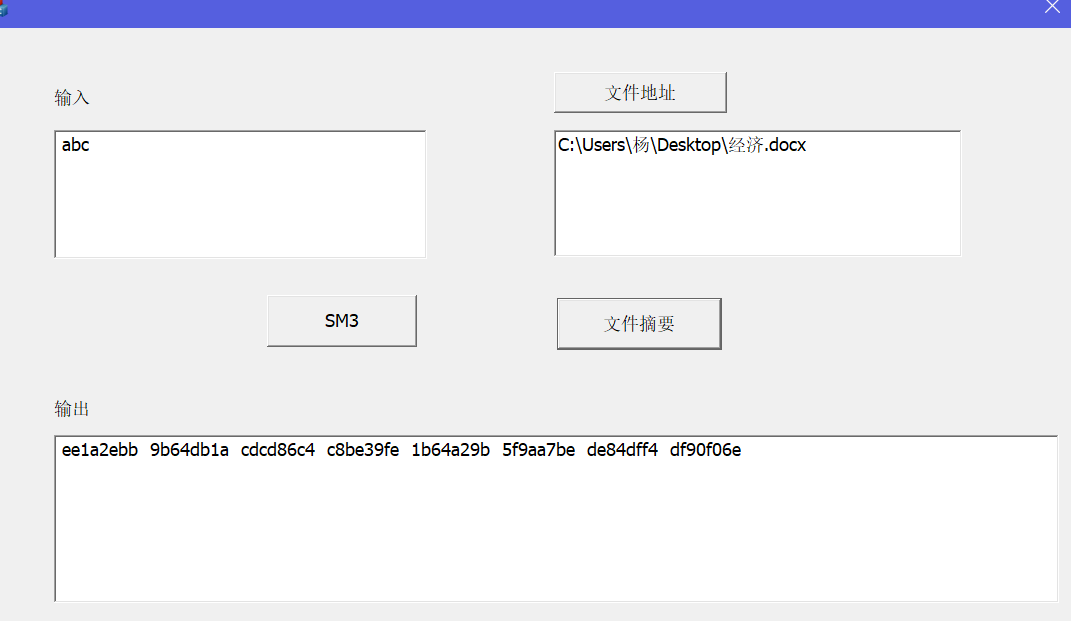
SetDlgItemText(IDC\_EDIT2, m);

}

1. **运行截图：**



对文件进行摘要：



* 1. **实验分析总结及心得**

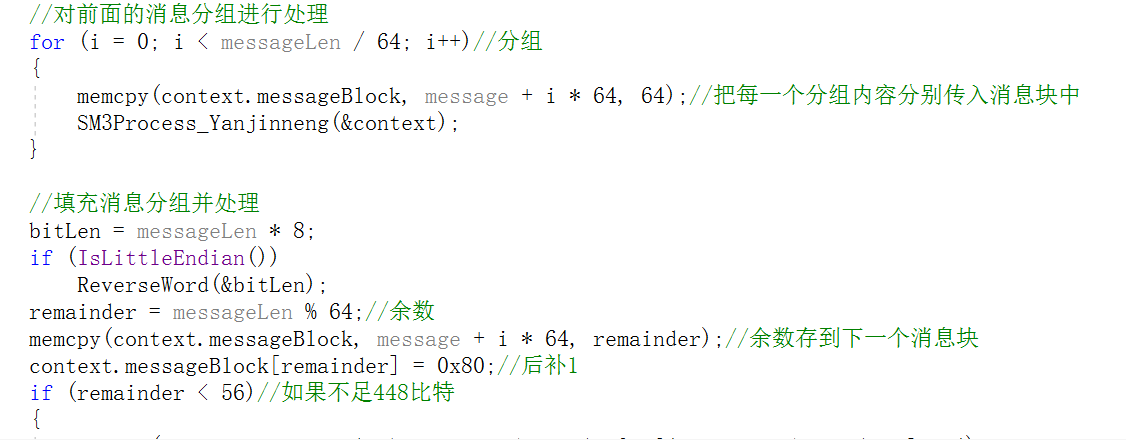
1. **遇到的问题和解决方法：**

**（1）问题：**

**在处理消息块的时候，不确定是先填充消息，然后进行分组处理；还是应该先对前面的消息进行分组处理，然后在填充剩下的。**

**解决方法：**

**查阅资料以后，我决定采用的是先对前面的消息按512比特分组处理，之后再对剩下的进行填充，然后处理。**

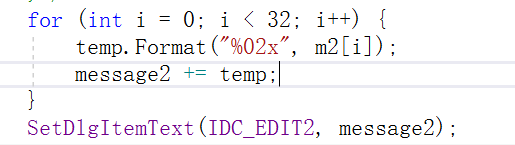


**（2）问题：**

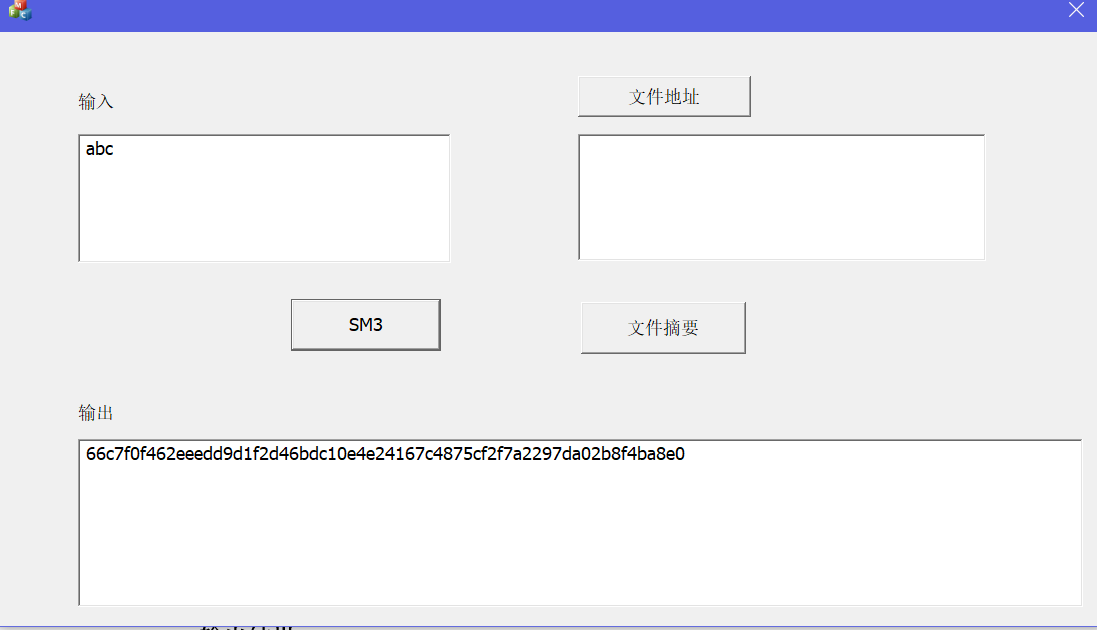
**在对明文进行SM3处理过后，输出的是一串乱码，而不是我们想要的256比特杂凑值。**

**解决方法：**

**最终输出的256比特杂凑值其实是以16进制表示的，所以我们在最后输出的时候也应该用16进制的方式输出。即把字符串用“%02x”的方式输出。**

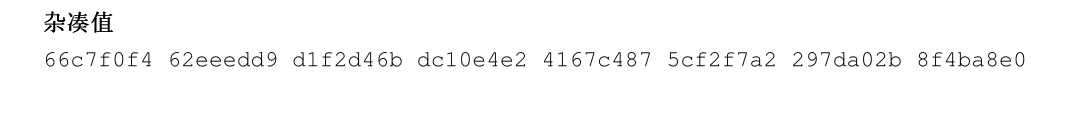


**输出结果：**



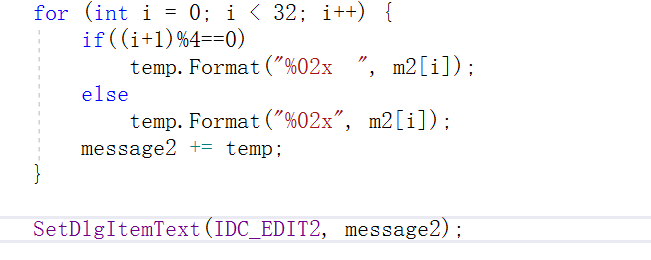
**（3）问题：**

**上一个问题解决之后又来了新问题：我该怎么把最后的输出按照实例给的那样每4个字节中间间隔一个空格。**



**解决方法：**

**用if语句进行筛选，当轮到第i个字节满足（i+1）%4=0时，要在输出的字节后加上一个空格。**



**输出结果：**

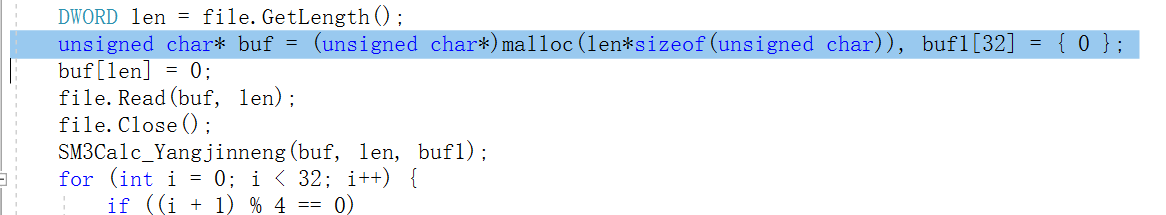


**（4）问题：**

**在对文件进行摘要时，对于怎么得到文件内容作为明文，我的想法是用CFile打开一个文件后，用GetLength得到文件内容长度len，然后把文件的len个字节存到一个数组里，之后对数组进行摘要操作。但问题是我该怎么在不知道文件长度的情况下定义一个足够大的数组？小了不行，大了又会让速度变慢。**

**解决方法：**

**用动态分配数组内存的方法，文件有多大，数组就有多大。**



**这样子既能保证文件被顺利读取，又能不减慢速度。**

1. **总结经验：**

（1）SM3适用于商用密码应用中的数字签名和验证、消息认证码的生成与验证以及随机数的生成，可满足多种密码应用的安全需求。

（2）不过是对称加密算法还是摘要算法，都需要在对明文进行分组的情况下进行操作。因为算法操作很复杂，若处理的数据长度太大，算法运行速度就会很慢，而且固定对特定比特串进行操作也更加方便。所以它们都需要固定一个长度值来分组。

（3）有时我们想要的输出并不是乱码，这时候就需要给定输出的字符类型，如十六进制、二进制、八进制等。

（4）当不知道一个数组应该有多长时，可以使其长度非常大，也可以用动态分配内存的方式定义数组。

**实验二 RSA密码算法**

1. **实验目的**

掌握RSA密码的加解密原理，并利用Visual C++实现。

1. **实验原理和内容概述**

利用RSA密码算法，随机生成密钥，对输入的明文进行加密和解密。扩展要求实现对文件进行DES、SM3后进行RSA加解密。

1. **实验环境**

Windows10，Visual Studio2017，C++。

1. **实验主要步骤及实验结果记录（含截图，关键源码，文字解释等）**
2. **算法核心思想：**

（1）RSA是一个非对称密码算法，涉及到几百位甚至上千位的大整数运算，大大超过了编程软件中的整型数据范围，所以需要用类或者结构体实现大整数的运算，包括模幂、模加等。因此在编写RSA算法之前，我们要先设计一个大整数用来运算。在这里我用的是无符号字符数组表示大整数，用结构体实现，为Bigint。

（2）因为无符号字符的数值范围是0~255，正好是一个字节，所以这里的大整数是用256进制表示的。例如数组a表示的数为num[0]\*1+a.num[1]\*256+……+a.num[SIZE-1]\*256^(SIZE-1)。其中SIZE表示数组长度，考虑到后面明文的长度关系，SIZE要尽量选择大的，但又不能太大，否则算法会很慢。这里选择SIZE为64。

（3）因为大整数乘法可能需要的数组长度是原数组的两倍，所以需要定义另一个Bigint2，长度为2\*SIZE。

（4）对于大整数运算来说，不管是加法还是减法，都是从低位到高位逐字节相加减。

（5）在实现加减乘除之后，就可以在此基础上实现求模，模逆和模幂等运算。

（6）实现了大整数的运算后，还需要生成大素数p。这里主要是利用上面的代码生成大整数后，用Miller-Rabin素性检测算法判断生成的是否是素数。因为单次判断有错误概率，所以要进行20次判断。

（7）至此，已经能够生成RSA所需的随机密钥了。之后再进行加密和解密即可。

1. **核心代码及注释：**

void PrintNum(unsigned char \*out, int &len, Bigint a) //打印输出并储存数组

{

int i = 0;

Bigint b = { 0 };

Bigint c = { 10 };

while (Compare(a, b) == 1)

{

out[i] = Mod(a, c).num[0];

a = Div(a, c);

i++;

}

len = i;

unsigned char temp = 0;

for (int k = 0; k < len / 2; k++)

{

temp = out[k];

out[k] = out[len - 1 - k];

out[len - 1 - k] = temp;

}

for (i = 0; i < len; i++)

{

out[i] = out[i] + '0';

}

}

Bigint ByteMoveLeft(Bigint a, int loop) { //左移loop个字节(每左移一位乘以256)

for (int i = Length(a) - 1; i >= 0; i--)

{

if (i + loop >= SIZE)//如果超过最大长度

continue;

a.num[i + loop] = a.num[i];

}

for (int i = loop - 1; i >= 0; i--)//移位后补0

a.num[i] = 0;

return a;

}

void BitMoveRight(Bigint &a) { //右移一个比特(每右移一位除以2)

Bigint two = { 2 };

a = Div(a, two);

}

Bigint BigRand(Bigint n) //生成[0,n）内的随机数

{

Bigint res = { 0 };

for (int i = 0; i < SIZE; i++)

{

res.num[i] = rand() % 256; //每个字节都取成[0,255]内随机数

}

res = Mod(res, n); //利用求模函数使其落到[0,n)中

return res;

}

Bigint BigRandOdd(int bytes) //生成字节数为bytes的随机奇数

{

Bigint res = { 0 };

for (int i = 0; i < bytes - 1; i++)

{

res.num[i] = rand() % 256;

}

res.num[bytes - 1] = 128 + rand() % 128; //最高位取1

if (!(res.num[0] & 0x01))

res.num[0] = res.num[0] + 1;//若为偶数，则加1为奇数

return res;

}

bool MillerRabinKnl(Bigint &n) //单次MillerRabin素性检测

{

Bigint b, m, v, temp;

Bigint j = { 0 };

Bigint one = { 1 };

Bigint two = { 2 };

Bigint three = { 3 };

m = Sub(n, one);

//计算m,j,使得n-1=2^j\*m(m为奇数)

while (!(m.num[0] & 0x01)) //若m不为奇数

{

j = Add(j, one); //j=j+1;

BitMoveRight(m); //相当于除2

}

b = Add(two, BigRand(Sub(n, three)));//随机选取b在[2,N-1]

v = PowMod(b, m, n); //计算v=b^mmodn

if (Compare(v, one) == 0) //若v=1,通过测试

return 1;

//若v!=1

Bigint i = { 1 };

temp = Sub(n, one);

while (Compare(v, temp) < 0) //若v<n-1,不通过

{

if (Compare(i, j) == 0) //若i=j，是合数，不通过

return 0;

v = MulMod(v, v, n); //v=v^2modn,i=i+1

i = Add(i, one);

}

return 1; //若v=n-1,通过检测

}

Bigint GenPrime(int bytes) //生成字节数为bytes的大素数

{

Bigint res = BigRandOdd(bytes);

int loop = 20;

while (!MillerRabin(res, loop)) //如果多次检测不通过，重新取一个奇数

{

res = BigRandOdd(bytes);

}

return res;

}

Bigint PowMod(Bigint a, Bigint b, Bigint n) //模幂，计算a^bmodn

{

Bigint c = { 1 };

Bigint temp = { 1 };

while (Length(b) > 0)

{

//当b为偶数时：a^b mod c = ((a^2)^b/2) mod c;当b为奇数时：ab mod c = ((a^2)^b/2× a) mod c

while (!(b.num[0] & 1))//b是偶数

{

BitMoveRight(b);//b除以2

a = MulMod(a, a, n);//a=a^2modn

} //其实就是每次将b分一半，直到b=0

b = Sub(b, temp);

c = MulMod(a, c, n);

}

return c;

}

bool Inverse(Bigint e, Bigint N, Bigint &d) //大整数模逆，求e模N的逆，结果存入d

{

Bigint r1 = { 0 };

Bigint r2 = { 0 };

Copy(r1, e);

Copy(r2, N); //设初始值r1=e,r2=N

Bigint s1 = { 1 }; //设系数初始值s1=1,s2=0

Bigint s2 = { 0 };

Bigint s = { 0 };

Bigint r = { 0 };

while (1)

{

if (Length(r1) == 0) //若r1=0，求模逆失败

return 0;

if (Length(r1) == 1 && r1.num[0] == 1)

{

Copy(d, s1); //若r1=1,求模逆成功，将结果存入d

return 1;

}

q = Div(r1, r2); //商q=r1/r2

s = Sub2Mod(s1, MulMod(q, s2, N), N);//s=s1-q\*s2,为了结果非负，使用模N运算

r = Sub(r1, Narrow(Mul(q, r2))); //r=r1-q\*r2

Copy(r1, r2);

Copy(s1, s2);

Copy(s2, s);

Copy(r2, r);

}

}

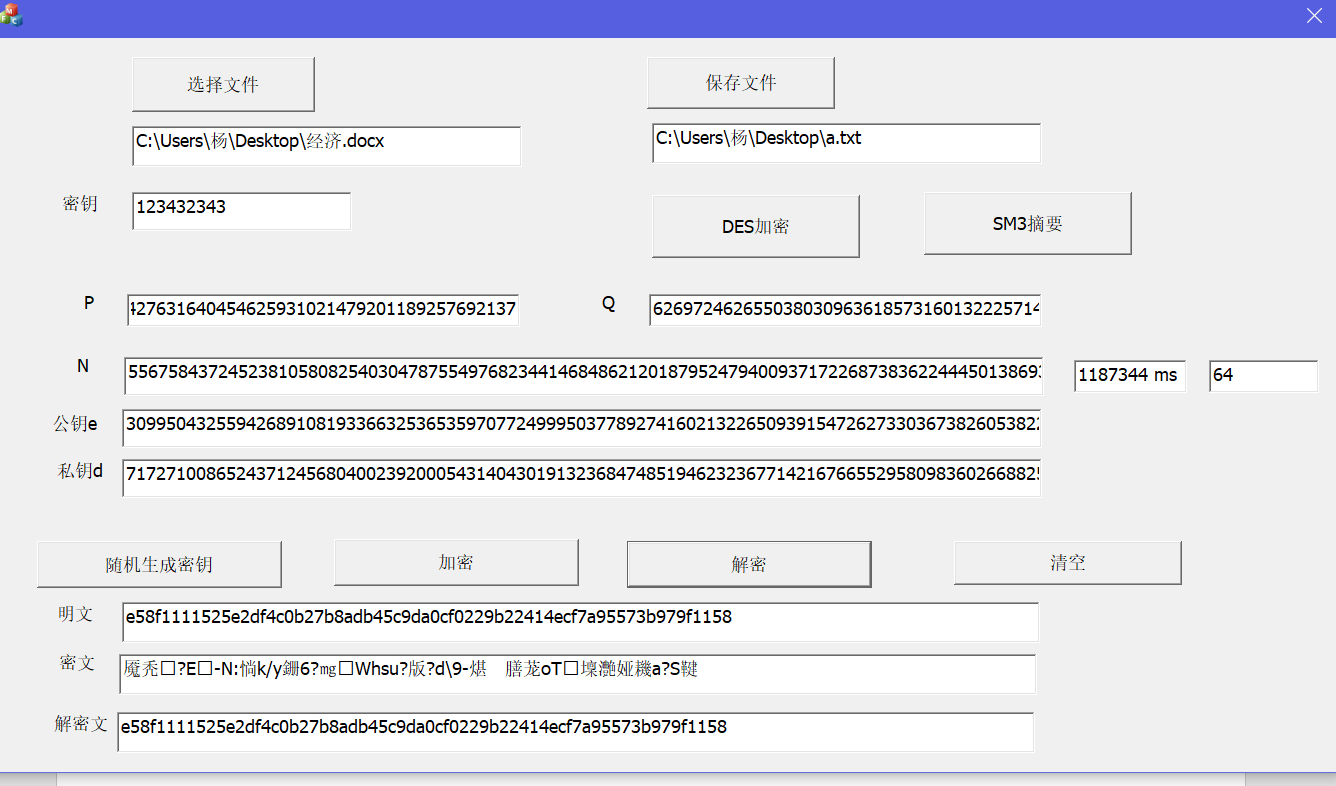
1. **扩展要求实现：**

扩展要求实现对文件进行DES加密、SM3摘要后，在对摘要值进行RSA加密。其实也就是把之前实验做好的DES和SM3的相关代码合并进来即可。我采取的方法是用把它们的代码写入头文件，在引用就可以了。

1. **运行截图：**



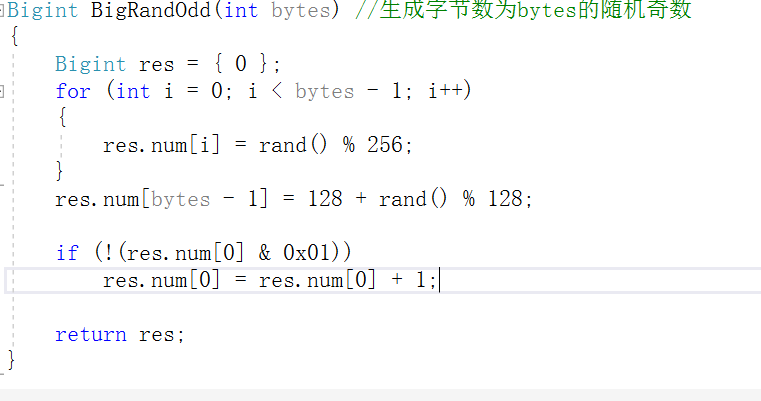
**对文件进行操作：**



1. **实验分析总结及心得**
2. **遇到的问题和解决方法：**

**（1）问题：**

**对生成固定字节的随机奇数函数BigRandOdd的代码理解不清楚，不明白其中**res.num[bytes - 1] = 128 + rand() % 128;**是什么意思。**

**解决方法：**

**这个函数前半部分其实就是生成一个随机数。用res作为临时数组，数组 里从0到bytes-1的位置都是0~255的随机数。之后的**res.num[bytes - 1] = 128 + rand() % 128;**的意思是把这个生成随机数的最高位取成1（128就是10000000），然后加上一个随机数。为什么要保证最高位是1呢？这是因为如果最高位不是1，它的字节数就不再是bytes了，这就与我们的要求不符，所以必须限定最高位是1。之后的代码则是保证随机数为奇数。**if (!(res.num[0] & 0x01) res.num[0] = res.num[0] + 1;**的意思是：如果生成的是偶数，则要把它加1变成奇数。经过这样以后，就能生成一个字节数为bytes的大奇数了。**

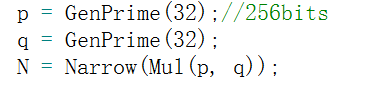
**（2）问题：**

**我想把模N的比特数给改成512比特时，将随机数p和q都改为了32字节，按理来说这样生成的N=p\*q就为64字节。可是还是生成32字节的N。**

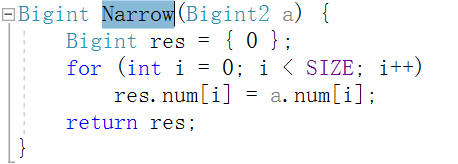


**解决方法：**

**解决这个问题需要仔细看看函数，对于生成N来说，最主要的是Narrow函数。**



**所以进去Narrow函数看一下：**

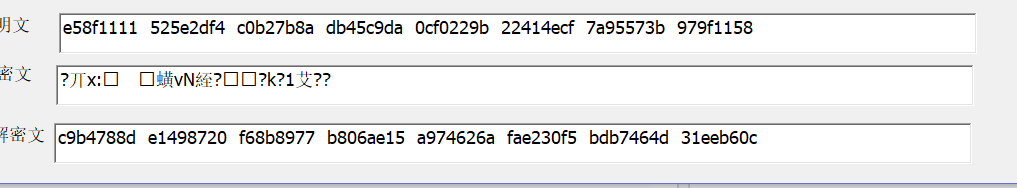


**可以看到最终的结果res的长度是与SIZE有关的。也就是说就算怎么改p,q，只要SIZE一直不变，N就会一直是SIZE长度。所以我们需要把SIZE改成65。这样就能生成64字节的N了。**

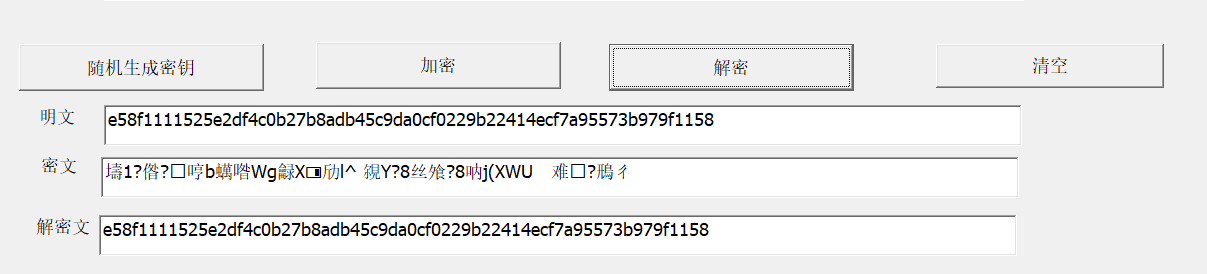


**（3）问题：**

**在将DES和SM3的相关功能合并入RSA后，进行SM3摘要后生成的哈希值加密和解密结果不一样。**



**解决方法：这个问题困扰了我好久好久，结果问题其实很简单，说多了都是泪……因为看函数本身也没有看出来什么问题，就采用调试的方法，一个一个变量地看，最后发现加密后生成的密文长度一直都是64，也就是SIZE的长度，可以看出密文长度和N的长度是一样的。而我之前一直认为大整数只和生成密钥的过程有关，现在想想明文和密文也要用到大整数啊。在经过一阵分析后，我知道了问题原因：因为SIZE为64，代表数组长度最大就为64，而我的SM3摘要后的明文本来应该是64字节，可我用空格把它们分割了，于是空格也占据了几个字节，这样总字节就变成了80，已经超过了64，这当然就造成了数据的丢失。因此解决方法就是把空格给去掉，这样加解密就正常了。**



1. **总结经验：**

（1）RSA是一个非对称密码算法，相比于对称密码算法，非对称密码算法要么原理简单但计算量巨大，要么原理过于抽象，难以理解。因此，要成功编写出非对称密码算法，需要我们对其原理认真研究，弄懂原理以后，才能实现算法。

（2）在处理非对称密码算法这样子需要庞大整数运算的算法时，因为现有的数据范围不够，所以我们需要自己先把大整数类型设计出来，才能用其编写。

（3）编程中碰到问题是很正常的事，这时候不要惊慌，需要认认真真地对函数进行分析，搞懂它真正在干什么，知道它的目的，这样子才能对症下药。若连函数干什么都不知道，那出了问题自然也就不能解决。

（4）调试很有用也很重要。在你觉得函数本身没有问题，实在不知道问题出在哪时，采用调试的方法，一个一个变量地分析，就能找到问题所在。