

# 同济大学计算机系

## 现代密码学课程设计实验报告



学 号

姓 名

专 业

授课老师

# 目录

一、	分组密码与高级加密标准	1
1.	实现 AES 算法	1
1.1.	算法中的变量	1
1.2.	算法流程	1
1.3.	模块实现	2
1.4.	有限域的相关操作	5
1.5.	工程文件结构	9
1.6.	代码测试	9
2.	实现 CBC 模式下的 AES 文件加密和解密	10
2.1.	CBC 实现原理	10
2.2.	算法流程	10
2.3.	模块实现	13
4.1.	文件结构	16
4.2.	代码测试	16
二、	安全哈希 SHA-1	19
1.	用到的数据结构	19
1.1.	std::bitset	19
1.2.	std::vector	19
1.3.	string	19
2.	实现 SHA-1 算法	20
2.1.	算法流程	20
2.2.	模块实现	21
2.3.	文件结构	27
2.4.	代码测试	27
三、	RSA 加密和解密	30
1.	NTL 库的配置和使用	30
1.1.	NTL 简介	30
1.2.	NTL 的安装和配置	30
2.	实现 RSA 算法	30
2.1.	RSA 加密解密算法	30

2.2. 模块实现 .....	30
2.3. 文件结构 .....	35
2.4. 测试 .....	35
<b>四、 签名方案 .....</b>	<b>36</b>
1. RSA 签名方案 .....	36
2. 算法实现 .....	36
2.1. 类封装 .....	36
2.2. 签名算法和验证算法 .....	38
3. 文件结构 .....	38
4. 测试 .....	38
<b>五、 证书方案 .....</b>	<b>40</b>
1. 数字证书的原理 .....	40
2. 数字证书的实现 .....	40
2.1. 类设计 .....	40
2.2. 算法实现 .....	41
3. 文件结构 .....	45
4. 测试 .....	45
<b>六、 文件加密系统 .....</b>	<b>47</b>
1. 文件传输的安全性分析 .....	47
2. 文件加密系统的结构和流程图 .....	49
3. 加密系统的实现 .....	49
3.1. 类型转换模块 .....	49
3.2. 加密模块 .....	50
4. 文件结构 .....	53
5. 测试 .....	54
<b>七、 心得体会 .....</b>	<b>59</b>

# 一、 分组密码与高级加密标准

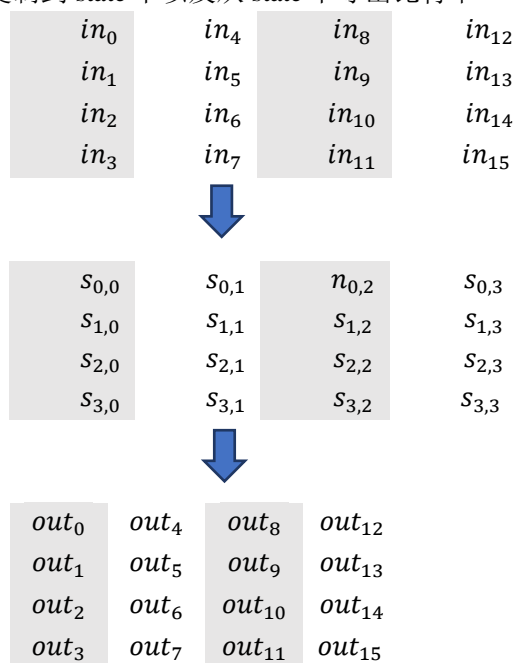
## 1. 实现 AES 算法

### 1.1. 算法中的变量

#### 1.1.1. state

state 是算法中执行操作的二维数组。该数组有四行，Nb 列

由于后续操作都是在 state 数组上执行的，因此 AES 算法前后分别需要将比特串复制 to state 中以及从 state 中导出比特串



#### 1.1.2. Key

Key 是一个 16bytes 的字符串。最初的密钥是 128 位的种子密钥，由种子密钥生成共  $Nr+1$  轮子密钥，每一轮子密钥也是 16bytes，用于和 state 中的每个分量进行异或运算。

### 1.2. 算法流程

- 1) 初始化 state 为 x，对 state 进行 AddRoundKey 操作
- 2) 对前  $Nr-1$  轮的每一轮：  
对 state 依次进行 SubBytes，ShiftRows，MixColumns，AddRoundKey 操作
- 3) 第  $Nr$  轮：依次进行 SubBytes，ShiftRows，AddRoundKey 操作
- 4) 将 state 还原成密文

步骤如下：

## 1.3. 模块实现

## 1.3.1. AddRoundKey

实现方法：逐位异或即可

```

1.  void AddRoundKey(State&s, Key k)
2.  {
3.      for (int row = 0; row < 4; row++)
4.      {
5.          for (int col = 0; col < 4; col++)
6.          {
7.              int idx = row * 4 + col;
8.              s[row][col] = s[row][col] ^ k[idx];
9.          }
10.     }
11. }
```

## 1.3.2. SubBytes

实现方法：调用 S 盒

```

1.  void SubBytes(State &s)
2.  {
3.      for (int row = 0; row < 4; row++)
4.      {
5.          for (int col = 0; col < 4; col++)
6.          {
7.              s[row][col] = sbox(s[row][col]);
8.          }
9.      }
```

AES 的 S 盒可以计算出准确的代数式，计算方法为：

- 1) 将输入 S 盒的一字节数转换成有限域的元素
- 2) 求出对应逆元
- 3) 将逆元转换回一字节数
- 4) 把一字节数看成矢量和变换矩阵相乘

```

1.  char sbox(char state)
2.  {
3.      Field f = BinaryToField(state);
4.      f = FieldInv(f);
5.      char s = FieldToBinary(f);
6.      int matrix[8][8] = { 1,1,1,1,1,0,0,0,
7.                          0,1,1,1,1,1,0,0,
8.                          0,0,1,1,1,1,1,0,
9.                          0,0,0,1,1,1,1,1,
10.                         1,0,0,0,1,1,1,1,
```

```

11.             1,1,0,0,0,1,1,1,
12.             1,1,1,0,0,0,1,1,
13.             1,1,1,1,0,0,0,1 };
14.     int y[8] = { 0,0,0,0,0,0,0,0 };
15.     int c[8] = { 0,1,1,0,0,0,1,1 };
16.     for (int i = 0; i < 8; i++)
17.     {
18.         int x[8];
19.         for (int j = 0; j < 8; j++)
20.         {
21.             x[j] = (s >> (7-j)) & 1;
22.             y[i] = y[i] + x[j] * matrix[i][j];
23.         }
24.     }
25.     for (int i = 0; i < 8; i++)
26.         y[i] = (y[i] + c[i])%2;
27.
28.     char res = y[0];
29.     for (int i = 1; i < 8; i++)
30.     {
31.         res = (res << 1) + y[i];
32.     }
33.     return res;
34. }
    
```

### 1.3.3. ShiftRows

ShiftRows 达到的效果如下：States 从第一个矩阵转到第二个矩阵

$s_{0,0}$	$s_{0,1}$	$n_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$



$s_{0,0}$	$s_{0,1}$	$n_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

实现方法：直接移位操作即可

```

1.     void ShiftRows(State &s)
2.     {
3.         char temp[4][4];
4.         for (int row = 0; row < 4; row++)
5.             for (int col = 0; col < 4; col++)
6.                 temp[row][col] = s[row][col];
7.         for (int row = 0; row < 4; row++)
    
```

```

8.         for (int col = 0; col < 4; col++)
9.             s[row][col]=temp[row][(col+row)%4];
10.    }
    
```

#### 1.3.4. MixColumns

MicColumns 对 states 中四列中的每一列进行操作。State 的每一列都被一个新列替代，这个新列由原列乘上域  $\mathbb{F}_{2^8}$  中的元素组成的矩阵而来。

实现方法：直接乘以变换矩阵，变化矩阵为：

具体地，算法为：

```

MixColumns(c)
For i=1 to 3
    do  $t_i \leftarrow \text{BinaryToField}(s_{i,c})$ 
 $u_0 \leftarrow \text{FieldMult}(x, t_0) \oplus \text{FieldMult}(x+1, t_1) \oplus t_2 \oplus t_3$ 
 $u_1 \leftarrow \text{FieldMult}(x, t_1) \oplus \text{FieldMult}(x+1, t_2) \oplus t_3 \oplus t_0$ 
 $u_2 \leftarrow \text{FieldMult}(x, t_2) \oplus \text{FieldMult}(x+1, t_3) \oplus t_0 \oplus t_1$ 
 $u_3 \leftarrow \text{FieldMult}(x, t_3) \oplus \text{FieldMult}(x+1, t_0) \oplus t_1 \oplus t_2$ 
For i=1 to 3
    do  $s_{i,c} \leftarrow \text{FieldToBinary}$ 
    
```

对于 State 的某一列  $(t_0, t_1, t_2, t_3)^T$ ，经过 MixColumns 变换后将变为  $(s_0, s_1, s_2, s_3)^T$

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = F \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

$F$  为域  $\mathbb{F}_{2^8}$  上的元素，乘法定义为域上的乘法

$$F = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

```

1. void MixColumns(State &s)
2. {
3.     char matrix0[4][4] = {
4.         0x2,0x3,0x1,0x1,
5.         0x1,0x2,0x3,0x1,
6.         0x1,0x1,0x2,0x3,
7.         0x3,0x1,0x1,0x2
8.     };
9.     Field matrix[4][4];
10.    for (int i = 0; i < 4; i++)
11.        for (int j = 0; j < 4; j++)
12.            matrix[i][j] = BinaryToField(matrix0[i][j]);
13.    Field t[4];
14.    Field u[4];
15.    for (int col = 0; col < 4; col++)
    
```

```

16.      {
17.          for (int i = 0; i < 4; i++)
18.              t[i] = BinaryToField(s[i][col]);
19.
20.          for (int row = 0; row < 4; row++)
21.              {
22.                  u[row] = FieldInit();
23.                  for (int i = 0; i < 4; i++)
24.                      u[row] = FieldAdd(u[row], FieldMul(matrix[row][i], t[i]
25.  ));
26.              }
27.          for (int i = 0; i < 4; i++)
28.              s[i][col] = FieldToBinary(u[i]);
29.      }

```

#### 1.4. 有限域的相关操作

##### 1.4.1. 该部分涉及到的数据类型 Field

该类型是一个含有 8 个元素的 `vector<int>` 数组

AES 采用的本原多项式对应的 Field 为常量

```
const Field base = { 1,1,0,1,1,0,0,0 };
```

##### 1.4.2. 实现方法

AES 是在有限域  $\mathbb{F}_{2^8} = \frac{\mathbb{Z}_2[x]}{x^8+x^4+x^3+x+1}$  上进行的操作

为了实现有限域上的求逆，依次实现对  $\mathbb{F}_{2^8}$  上元素的乘积，带余除法以及最终用扩展的欧几里得算法实现求逆元的操作。

##### 乘法的实现

有限域的乘法可以通过逐步相加实现，即

$$(x^2 + 1) \cdot g(x) = x^2 \cdot g(x) + g(x)$$

设

$$g(x) = \sum_{k=0}^7 a_k x^k$$

则  $x$  乘  $g(x)$  等于：

$$g(x) \cdot x = a_7 x^8 + \sum_{k=1}^7 a_{k-1} x^k$$

当  $a_7 = 1$  时，在  $\mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$  的意义下

$$g(x) \cdot x = \sum_{k=1}^7 a_{k-1} x^k + x^4 + x^3 + x + 1$$

上述式子中  $a_k \in \mathbb{Z}_2$



因此，可以得到有限域上的乘法运算

```
1. Field FieldShift(Field f, int n, bool isBase)
2. {
3.     Field res = f;
4.     Field temp;
5.     if (n == 0)
6.         return res;
7.     for (int i = 1; i <= n; i++)
8.     {
9.         temp = res;
10.        for (int j = 0; j < 7; j++)
11.            res[j + 1] = temp[j];
12.        res[0] = 0;
13.        if(temp[7]==1&&(i!=n||!isBase))
14.            res = FieldAdd(res, base);
15.    }
16.    return res;
17. }
```

```
1. Field FieldMul(Field f1, Field f2)
2. {
3.     Field res = FieldInit();
4.     Field temp;
5.     for (int i = 0; i < 8; i++)
6.     {
7.         if (f2[i])
8.         {
9.             temp = FieldShift(f1, i);
10.            res = FieldAdd(res, temp);
11.        }
12.    }
13.    return res;
14. }
```

除法的实现

域上的除法运算除了多项式的每一项系数是在 $\mathbb{Z}_2$ 上，其余和普通多项式除法没有区别。由于是在 $\mathbb{Z}_2$ 上，设

$$f(x) = \sum_{k=0}^7 a_k x^k, g(x) = \sum_{k=0}^7 b_k x^k$$

则在 $\mathbb{Z}_2$ 上，有

$$f(x) - g(x) \equiv \sum_{k=0}^7 (a_k - b_k) x^k \equiv \sum_{k=0}^7 (a_k + b_k) x^k \equiv f(x) + g(x)$$

故作带余除法运算 $f(x) = g(x) \cdot q(x) + r(x)$ 时，可以逐步加 $q(x)$ ，直到 $\deg(r) < \deg(g)$ 为止

因此可以得到有限域上的除法运算

```

1.  Field* FieldDiv(Field f1, Field f2, bool isBase)
2.  {
3.      Field *arr;
4.      arr = new Field[2];
5.      arr[0] = FieldInit();
6.      arr[1] = FieldInit();
7.      int r_deg, divisor_deg;
8.      int shift;
9.      Field divisor = f2;
10.     Field q = FieldInit();
11.     Field r = f1;
12.     r_deg = isBase ? 8 : deg(r);
13.     divisor_deg = deg(divisor);
14.     int round = 0;
15.     while (r_deg >= divisor_deg)
16.     {
17.         round++;
18.         shift = r_deg - divisor_deg;
19.         q[shift] = 1;
20.         if(round==1)
21.             r = FieldAdd(r, FieldShift(divisor, shift, isBase));
22.         else
23.             r = FieldAdd(r, FieldShift(divisor, shift));
24.         r_deg = deg(r);
25.         divisor_deg = deg(divisor);
26.     }
27.     arr[0] = q;

```

```

28.     arr[1] = r;
29.     return arr;
30. }

```

### 求逆元

利用扩展的欧几里得算法即可实现求逆元的操作

扩展欧几里得算法不再赘述，根据课件即可实现

扩展欧几里得算法

$a_0 \leftarrow a, b_0 \leftarrow b$

$t_0 \leftarrow 0, t \leftarrow t, s_0 \leftarrow 1, s \leftarrow 0$

While  $r > 0$

Do

$temp \leftarrow t_0 - qt, t_0 \leftarrow t, t \leftarrow temp$

$temp \leftarrow s_0 - qs, s_0 \leftarrow s, s \leftarrow temp$

$a_0 \leftarrow b_0, b_0 \leftarrow r, q \leftarrow \left\lfloor \frac{a_0}{b_0} \right\rfloor, r \leftarrow a_0 - qb_0$

$r \leftarrow b_0$

Return  $(r, s, t)$

代码如下：

```

1.  Field FieldInv(Field f)
2.  {
3.
4.      if(isZero(f))
5.          return f;
6.      if(isOne(f))
7.          return f;
8.
9.      Field *arr;
10.     //Extended Euclidean algorithm
11.     Field s0 = FieldInit(), s1 = FieldInit(), s2 = FieldInit();
12.     Field t0 = FieldInit(), t1 = FieldInit(), t2 = FieldInit();
13.     Field r0 = base;
14.     s0[0] = 1;
15.     Field r1 = f;
16.     t1[0] = 1;
17.     Field q = FieldInit();
18.     Field r2 = base;
19.     //1. r0=r1*q+r2
20.     //2. r0<=r1
21.     //3. r1<=r2
22.     bool isBase = true;
23.     while (!isOne(r2))

```

```

23.      {
24.          arr = FieldDiv(r0, r1, isBase);
25.          q = arr[0];
26.          r2 = arr[1];
27.          s2 = FieldAdd(s0, FieldMul(q, s1));
28.          t2 = FieldAdd(t0, FieldMul(q, t1));
29.          isBase = false;
30.          r0 = r1;
31.          r1 = r2;
32.          s0 = s1;
33.          t0 = t1;
34.          s1 = s2;
35.          t1 = t2;
36.      }
37.
38.      return t2;
39.  }

```

### 1.5. 工程文件结构

文件名	功能
F256.h	声明有限域的头文件
AES.h	声明 AES 中子步骤的头文件
const.h	声明涉及到的常量的头文件
F256.cpp	定义有限域相关操作
AES.cpp	定义子步骤中涉及到的操作
main.cpp	测试主程序

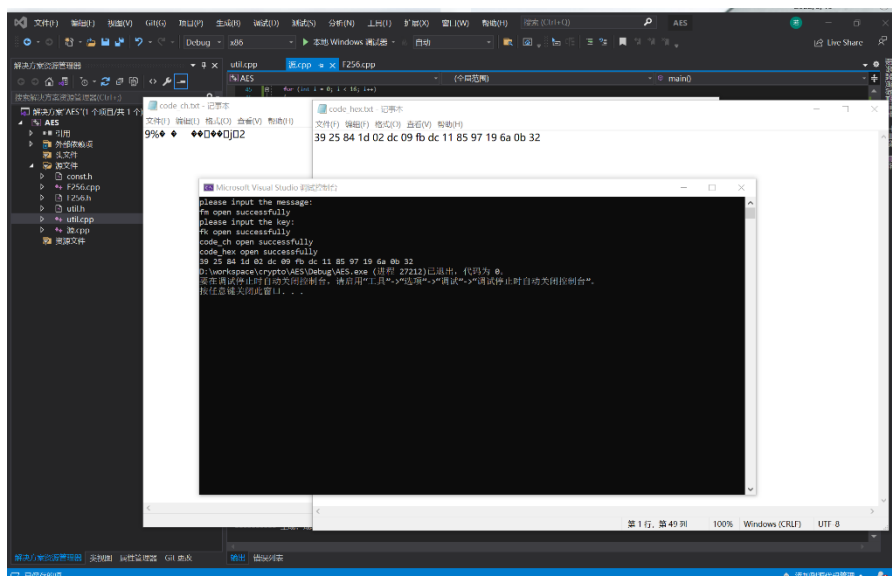
### 1.6. 代码测试

测试数据:

message: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

cipher key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

将明文和密钥分别写入文件 'fm.txt' 和 'fk.txt' 最终将加密结果输出到文件 'code\_ch.txt' 和 'code\_hex.txt' 中, 输出结果如下:



根据手册中的结果比对，结果正确

Input = 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Cipher Key = 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

output

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

## 2. 实现 CBC 模式下的 AES 文件加密和解密

### 2.1. CBC 实现原理

定义  $y_0 = IV$ ,  $IV$  为初始化向量, CBC 构造  $y_i$  的方式是:

$$y_i = e_K(y_{i-1} \oplus x_i)$$

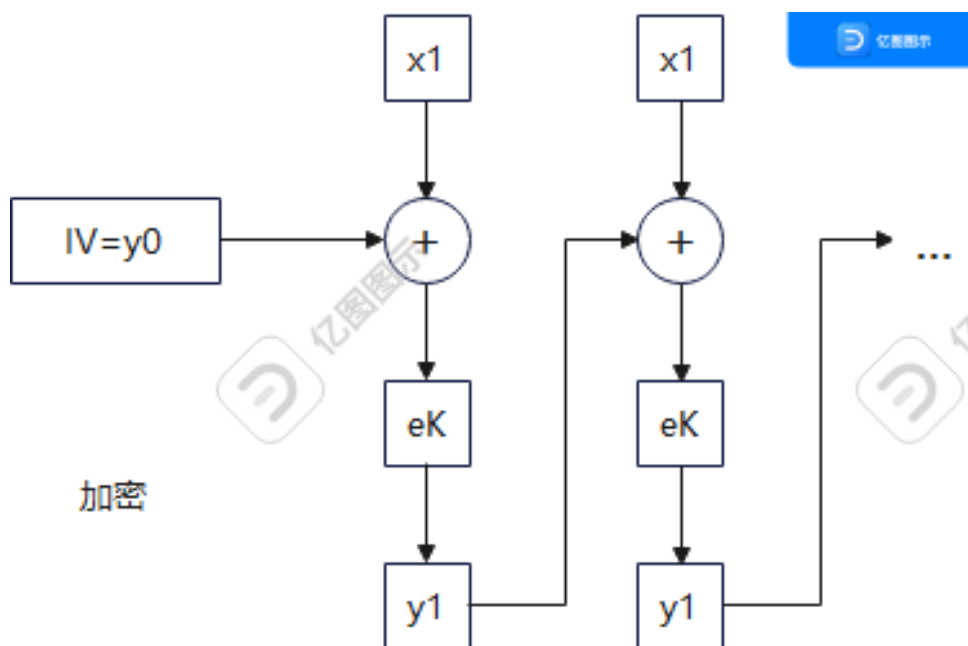
其中这里的  $e_K$  是 AES 的加密函数

### 2.2. 算法流程

#### 2.2.1. 加密流程

- 1) 生成  $IV$
- 2) 对明文进行分组
- 3) 计算  $y_{i-1} \oplus x_i$
- 4) 用 AES 加密

CBC 模式加密描述如下:



$IV$ 采用两种方式生成，其一是直接用 0 向量，其二是用伪随机数生成。  
本次实验我才用的分组方式是：

如果明文长度  $n$  是分组大小的倍数，则直接划分为  $\frac{n}{16}$  组，然后再末尾加上一个

0x0

如果不是，则计算需要多少位凑成 16 的倍数，然后在明文后面添加  $ext = -n - 1 \pmod{16}$  个 0，然后再一次补上  $ext$ , 0x1

其中最后 1 位，表示是否补零。

代码实现为：

```
1.  vector<char> CBCEncryption(vector<char> message, Key seed)
2.  {
3.      vector<char> res;
4.      res = message;
5.      int len = res.size();
6.      char is_ext = 0; //是否填充
7.      if (len % 16 != 0)
8.      {
9.          is_ext = 1;
10.         int ext = 16 - len % 16 - 1; //需要填充的 0 的数量
11.         for (int i = 1; i <= ext; i++)
12.             res.push_back(0);
13.         res.push_back((char)ext); //最后一位填充的是填充 0 的数量
14.     }
15.     cout << "是否使用随机的初始向量?(1/0)" << endl;
16.     bool flag;
17.     cin >> flag;
18.     if (flag)
```

```

19.         IV = randStr();
20.     else
21.         IV = vector<char>(16, 0);
22.
23.     vector<char> y_pre = IV;
24.     vector<char> y_cur(16);
25.     int n = res.size() / 16;
26.     vector<char> x_cur(16);
27.     for (int i = 1; i <= n; i++)
28.     {
29.         for (int j = 0; j < 16; j++)
30.             x_cur[j] = res[(i - 1) * 16 + j];
31.         x_cur = XOR(x_cur, y_pre);
32.         y_cur = AES(x_cur, seed);
33.         for (int j = 0; j < 16; j++)
34.         {
35.             res[(i - 1) * 16 + j] = y_cur[j];
36.         }
37.     }
38.     res.push_back(is_ext);
39.     return res;
40. }

```

### 2.2.2. 解密流程

利用加密的IV文件，反向进行操作即可。

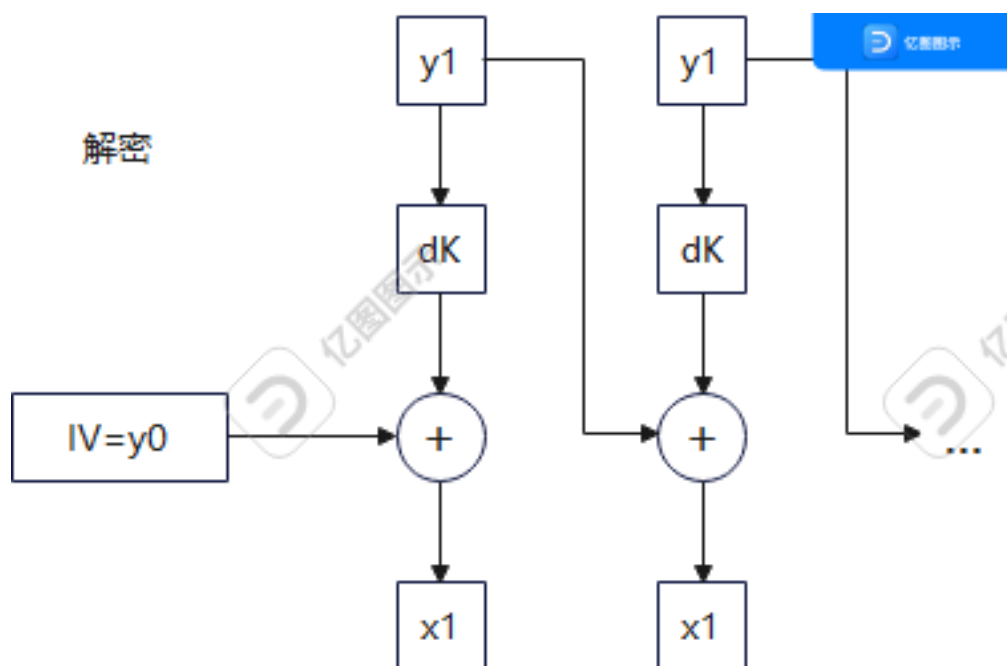
$$y_0 = IV, x_i = y_{i-1} \oplus d_k(y_i)$$

其中 $d_k$ 是 AES 的解密函数

AES 的解密方法如下：

- 1) 依次进行 InvAddRoundKey, InvShiftRows, InvSubBytes
- 2) 对 Nr-1~1 轮解密，依次进行:  
InvAddRoundKey, InvMixColumns, InvShiftRows, InvSubBytes
- 3) 进行 InvAddRoundKey

CBC 模式解密描述如下：



### 2.3. 模块实现

对于 `InvAddRoundKey`，由于

$$a = b \oplus c \Rightarrow b = a \oplus c$$

因此，`InvAddRoundKey`=`AddRoundKey`

### 3. 对于 `InvShiftRows`，只要反向移位即可

即将 `State` 从第一个矩阵变换到第二个

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,0}$



$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

```

1. void InvShiftRows(State& s)
2. {
3.     char temp[4][4];
4.     for (int row = 0; row < 4; row++)
5.         for (int col = 0; col < 4; col++)
6.             temp[row][col] = s[row][col];
7.     for (int row = 0; row < 4; row++)
8.         for (int col = 0; col < 4; col++)
9.             s[row][col] = temp[row][(col - row + 4) % 4];
10. }
    
```



## 4. 对于 InvMixColumns:

对于 State 的某一行  $(t_0, t_1, t_2, t_3)^T$ , 经过 MixColumns 变换后将变为  $(s_0, s_1, s_2, s_3)^T$

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = F \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

$F$  为域  $\mathbb{F}_{2^8}$  上的元素, 乘法定义为域上的乘法

$$F = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

因此要实现 InvMixColumns, 只需要计算出  $F$  在域  $\mathbb{F}_{2^8}$  上的逆  $F^{-1}$  即可

$$F^{-1} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix}$$

则

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = F^{-1} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

代码实现中, 只有变换矩阵不同, 改变 MixColumns 中的变换矩阵即可

```
1. void InvMixColumns(State& s)
2. {
3.     char matrix0[4][4] = {
4.         0x0e, 0x0b, 0x0d, 0x09,
5.         0x09, 0x0e, 0x0b, 0x0d,
6.         0x0d, 0x09, 0x0e, 0x0b,
7.         0x0b, 0x0d, 0x09, 0x0e
8.     };
9.     Field matrix[4][4];
10.    for (int i = 0; i < 4; i++)
11.        for (int j = 0; j < 4; j++)
12.            matrix[i][j] = BinaryToField(matrix0[i][j]);
13.    Field t[4];
14.    Field u[4];
15.    for (int col = 0; col < 4; col++)
16.    {
17.        for (int i = 0; i < 4; i++)
18.            t[i] = BinaryToField(s[i][col]);
19.
20.        for (int row = 0; row < 4; row++)
21.        {
22.            u[row] = FieldInit();
```

```

23.         for (int i = 0; i < 4; i++)
24.             u[row] = FieldAdd(u[row], FieldMul(matrix[row][i], t[i]
));
25.     }
26.     for (int i = 0; i < 4; i++)
27.         s[i][col] = FieldToBinary(u[i]);
28.     }
29. }

```

对于 InvSubBytes:

$$Res = A \cdot m^{-1} \oplus Const$$

A 是矩阵,  $m^{-1}$  是有限域逆元素对应的向量, 则

$$m^{-1} = A^{-1} \cdot (Res \oplus Const)$$

可以求出 A 在  $\mathbb{Z}_2$  下的逆矩阵, 然后修改 SubBytes 中的运算顺序即可

```

1.  char sboxInv(char state)
2.  {
3.      int matrix[8][8] = { 0,1,0,1,0,0,1,0,
4.                           0,0,1,0,1,0,0,1,
5.                           1,0,0,1,0,1,0,0,
6.                           0,1,0,0,1,0,1,0,
7.                           0,0,1,0,0,1,0,1,
8.                           1,0,0,1,0,0,1,0,
9.                           0,1,0,0,1,0,0,1,
10.                          1,0,1,0,0,1,0,0 };
11.
12.      int y[8];
13.      int c[8] = { 0,1,1,0,0,0,1,1 };
14.
15.      for (int j = 0; j < 8; j++)
16.      {
17.          y[j] = (state >> (7 - j)) & 1;
18.      }
19.
20.      for (int i = 0; i < 8; i++)
21.          y[i] = (y[i] + c[i]) % 2;
22.
23.      int x[8] = { 0,0,0,0,0,0,0,0 };
24.
25.      for (int i = 0; i < 8; i++)
26.      {
27.          for (int j = 0; j < 8; j++)
28.          {
29.              x[i] = (x[i] + y[j] * matrix[i][j])%2;
30.          }

```

```

31.     }
32.
33.
34.     char res = x[0];
35.     for (int i = 1; i < 8; i++)
36.     {
37.         res = (res << 1) + x[i];
38.     }
39.     Field f = BinaryToField(res);
40.     f = FieldInv(f);
41.     res = FieldToBinary(f);
42.     return res;
43. }
44.
45. void InvSubBytes(State& s)
46. {
47.     for (int row = 0; row < 4; row++)
48.     {
49.         for (int col = 0; col < 4; col++)
50.             s[row][col] = sboxInv(s[row][col]);
51.     }
52. }

```

#### 4.1. 文件结构

文件名	功能
F256.h	声明有限域的头文件
AES.h	声明 AES 中子步骤的头文件
const.h	声明涉及到的常量的头文件
CBC.h	定义 CBC 加解密的接口
F256.cpp	定义有限域相关操作
AES.cpp	定义子步骤中涉及到的操作
main.cpp	测试主程序
CBC.cpp	CBC 模式下利用 AES 加解密的操作

该项目中比 AES 项目多了 CBC 的头文件和 cpp 文件，CBC 文件中主要是利用 AES 的加解密算法，还包括了明文密文的填充和去除填充，以及明文密文的分组加解密

#### 4.2. 代码测试

测试数据为如下

明文为



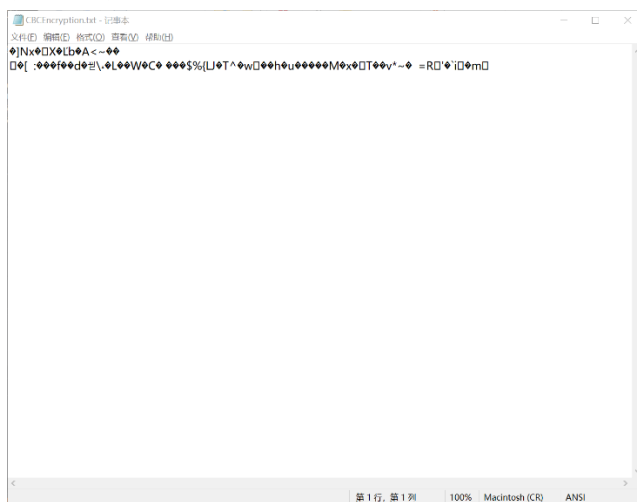
如图所示，依次输入明文路径，密钥路径，密文路径进行加密，会生成密文文件和 IV 文件，选择随机初始向量输入 1，输入 0 采用的是零向量。

```

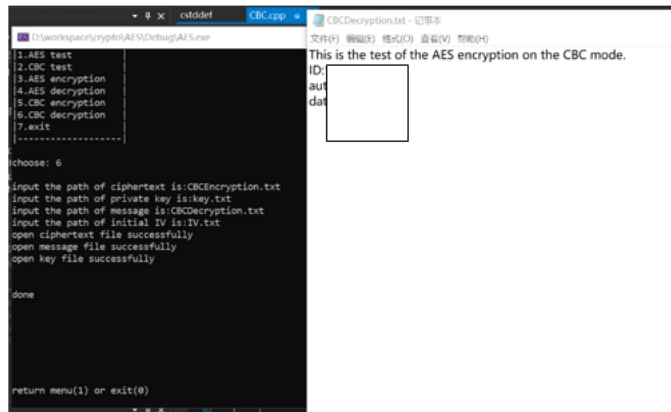
D:\workspace\crypto\AES\Debug\AES.exe
function
1.AES test
2.CBC test
3.AES encryption
4.AES decryption
5.CBC encryption
6.CBC decryption
7.exit
-----
choose: 5

input the path of message is:CBCTestMessage.txt
input the path of private key is:key.txt
input the path of ciphertext is:CBCEncryption.txt
open message file successfully
open ciphertext file successfully
open key file successfully
是否使用随机的初始向量?(1/0)
1
0 f7
1 5d
2 4e
3 78
4 d9
5 14
6 58
7 9a
8 c4
    
```

明文文件如下图



然后再次运行程序，对密文进行解密，依次输入密文路径，密钥路径，明文路径和 IV 文件路径。最终解密并写入明文文件中，可以发现，明文成功还原



## 二、安全哈希 SHA-1

### 1. 用到的数据结构

#### 1.1. `std::bitset`

`bitset` 模板类由若干个位（bit）组成，它提供一些成员函数，使程序员不必通过位运算就能很方便地访问、修改其中的任意一位。

##### 1.1.1. `std::bitset` 的 api

函数接口	功能
<code>bitset&lt;size_t&gt; (string)</code>	从右向左读取 01 串，并写入 <code>bitset</code> 中的构造函数
<code>bitset&lt;size_t&gt;(int num)</code>	将数的二进制形式从右向左写入 <code>bitset</code> 中的构造函数
<code>bitset&lt;size_t&gt;&amp; set(int pos)</code>	将 <code>pos</code> 位置的 bit 设置为 1 的成员函数
<code>bitset&lt;size_t&gt;&amp; operator &lt;&lt;(int n)</code>	将 <code>bitset</code> 逻辑左移 <code>n</code> 位的成员函数
<code>bitset&lt;size_t&gt;&amp; operator &gt;&gt;(int n)</code>	将 <code>bitset</code> 逻辑右移 <code>n</code> 位的成员函数
<code>bitset&lt;size_t&gt;&amp; operator  (bitset&lt;size_t&gt; bit)</code>	进行按位或运算的成员函数
<code>bitset&lt;size_t&gt;&amp; operator &amp;(bitset&lt;size_t&gt; bit)</code>	进行按位与运算的成员函数
<code>bitset&lt;size_t&gt;&amp; operator ^(bitset&lt;size_t&gt; bit)</code>	进行按位异或运算的成员函数
<code>string to_string():</code>	将 <code>bitset</code> 转换成 <code>string</code> 的成员函数

#### 1.2. `std::vector`

由于扩充后的比特串的 512 比特长度分组数量未知，因此采用 `vector` 来进行动态储存。

#### 1.3. `string`

主要用到了字符串的拼接操作，即针对 `string` 对象重载的 `+` 运算符

## 2. 实现 SHA-1 算法

### 2.1. 算法流程

#### 2.1.1. 迭代哈希结构

$Compress: \{0,1\}^{m+t} \rightarrow \{0,1\}^m, t \geq 1$  是一个压缩函数。迭代哈希函数是基于压缩函数构造出来

$$h: \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^t$$

构造的过程有三部：

#### 1) 预处理

给定输入比特串  $x, |x| \geq m + t + 1$ ，用公开算法构造串  $y, |y| \equiv 0 \pmod{t}$ ，并记为

$$y = y_1 \parallel y_2 \parallel \cdots \parallel y_r, \forall 1 \leq i \leq r, |y_i| = t$$

通常预处理采用  $y = x \parallel pad(x)$ ,  $pad(x)$  是填充函数

#### 2) 处理

设  $IV$  是一个长度  $m$  的公开初始值比特串，则计算

$$z_0 \leftarrow IV$$

$$z_1 \leftarrow compress(z_0 \parallel y_1)$$

$$z_2 \leftarrow compress(z_1 \parallel y_2)$$

...

$$z_r \leftarrow compress(z_{r-1} \parallel y_r)$$

#### 3) 输出变换

设  $g: \{0,1\}^m \rightarrow \{0,1\}^l$  是公开函数。定义  $h(x) = g(z_r)$ 。输出变换这一步是可选的，如不需要输出变换，则  $h(x) = z_r$

#### 2.1.2. SHA-1

SHA-1 采用的压缩函数为 SHA-1-PAD（要求  $|x| \leq 2^{64} - 1$ ），算法为：

SHA-1-PAD( $x$ )

$d \leftarrow 447 - |x| \pmod{512}$

$l \leftarrow |x|$  的二进制表示

$y \leftarrow x \parallel 1 \parallel 0^d \parallel l$

压缩函数较为复杂，涉及到了运算模块：循环左移  $s$  位的运算  $ROTL^s$ ，以及函数  $f_t$

```

SHA-1(x)
 $y \leftarrow \text{SHA-1-PAD}(x)$ 
令  $y = M_1 \parallel M_2 \parallel \dots \parallel M_n$ ，其中  $|M_i| = 512$ 
 $H_0 \leftarrow 67452301$ 
 $H_1 \leftarrow \text{EFC DAB89}$ 
 $H_2 \leftarrow 98\text{BADCFE}$ 
 $H_3 \leftarrow 10325476$ 
 $H_4 \leftarrow \text{C3D2E1F0}$ 
for  $i = 1$  to  $n$ 
    令  $M_i = W_0 \parallel W_1 \parallel \dots \parallel W_{15}$ ，其中每个  $W_i$  是一个字
    for  $t = 16$  to  $69$ 
         $W_t \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ 
         $A \leftarrow H_0$ 
         $B \leftarrow H_1$ 
         $C \leftarrow H_2$ 
         $D \leftarrow H_3$ 
         $E \leftarrow H_4$ 
    for  $t = 0$  to  $79$ 
         $temp \leftarrow ROTL^5(A) + f_t(B, C, D) + E + W_t + K_t$ 
         $E \leftarrow D$ 
         $D \leftarrow C$ 
         $C \leftarrow ROTL^{30}(B)$ 
         $B \leftarrow A$ 
         $A \leftarrow temp$ 
     $H_0 \leftarrow H_0 + A$ 
     $H_1 \leftarrow H_1 + B$ 
     $H_2 \leftarrow H_2 + C$ 
     $H_3 \leftarrow H_3 + D$ 
     $H_4 \leftarrow H_4 + E$ 

return  $H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$ 

```

很容易看出，该算法涉及到的加法的对象是字，因此该部分的加法事实上是  $\text{mod } 2^{32}$  意义下的加法

## 2.2. 模块实现

### 2.2.1. SHA-1-PAD

由于不可能一次性存储长度为  $2^{64} - 1$  比特的数据，因此采用分组填充的方式。

从 SHA-1-PAD 的算法描述中，可以看出这个 SHA-1 的填充方案是针对输入比特串  $x$  长度不能被 512 整除的部分。对该部分，SHA-1-PAD 做的是先填充一个 1，再填充 0 至被 512 整除。但是可能填充的 0 的数量可能较多，即一个 512 的输入串，可能



会填充出 2 个 512 长的分组 $M_i$ 。具体地， $l < 447$ 填充出一组，否则两组。因此，可以用代码实现分组填充的功能：

```

1.  bitset<1024> SHA1_PAD(string s)
2.  {
3.      int l = s.length();
4.      int d = mod(447 - l, 512);
5.      bitset<1024> temp(s);
6.      temp = temp << (1 + d + 64);
7.      bitset<1024> one;
8.      one.set(d + 64);
9.      bitset<1024> l_bit(l);
10.     bitset<1024> res = temp | one | l_bit;
11.     return res;
12. }
```

### 2.2.2. readfile

由于需要分组读取，因此需要一个专门的按一定长度读取文件的函数。SHA-1 算法中的分组长度为 512 比特，因此每次需要从文件中读入至多 512 比特的字符串。读取字符数量不够 512 的情况，一定出现在文件的最后一个分组，出现在文件末尾，这种情况可以用文件流的 eof 标识判断。

由于 eof 只有在本次读取操作失败后才被设置，因此，如果先判断，再读取，如果上一次读取到是文件最后一个字符，此次读取的 eof 判断会通过，会再执行一次读操作。所以，eof 的正确使用应该是先读取，再判断 eof，再处理数据。代码实现如下：

```

1.  string readfile(ifstream &f)
2.  {
3.      //读 512bit, 即 64 字节
4.      string res;
5.      char cur;
6.      int count = 0;
7.      while (1)
8.      {
9.          cur = f.get();
10.         if (f.eof())
11.             break;
12.         bitset<8> bit(cur);
13.         res += bit.to_string();
14.         count++;
15.         if (count >= 64)
16.             break;
17.     }
18.     return res;
19. }
```

### 2.2.3. ROTL

循环左移 $s$ 位可以看成是逻辑左移  $s$  位得到，再将被移出去的 $s$ 位串拼接到新串上，即

$$ROTL^s(X) = (X \ll s) \parallel (X \gg (len - s))$$

对于本次实验，由于循环左移的操作只以字为对象，故取 $len = 32$ 。

#### 2.2.4. $f_t$ 和 $K_t$

$f_t$ 的定义如下：

$$f_t(B, C, D) \begin{cases} (B \wedge C) \vee ((-B) \wedge D) & , 0 \leq t \leq 19 \\ B \oplus C \oplus D & , 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & , 40 \leq t \leq 59 \\ B \oplus C \oplus D & , 60 \leq t \leq 79 \end{cases}$$

$K_t$ 的定义如下：

$$K_t = \begin{cases} 5A827999, 0 \leq t \leq 19 \\ 6ED9EBA1, 20 \leq t \leq 39 \\ 8F1BBCDC, 40 \leq t \leq 59 \\ CA62C1D6, 60 \leq t \leq 79 \end{cases}$$

按照上述定义很容易实现 $f_t, K_t$ 的代码。

#### 2.2.5. $\text{mod } 2^{32}$ 意义的加法

事实上 $\text{mod } 2^{32}$ 意义的加法，就是 32 位的二进制数无符号数相加。对于一位的无符号数  $a$  和  $b$  相加，我们将产生的和记为  $s$ ，产生的进位记为  $\text{carry}$ ，可以得到如下的真值表：

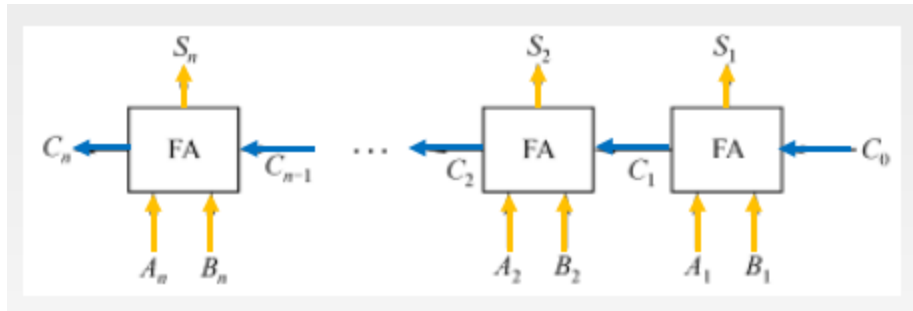
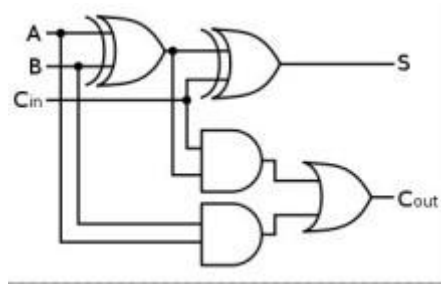
$a$	$b$	$\text{carry}_{n-1}$	$s$	$\text{carry}_n$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

可以得到这些变量之间的组合逻辑为：

$$s = a \oplus b \oplus \text{carry}_{n-1}$$

$$\text{carry}_n = (a \wedge b) \vee (a \wedge \text{carry}_{n-1}) \vee (b \wedge \text{carry}_{n-1})$$

利用上述组合逻辑可以得到一位的全加器。将若干个一位的全加器相互串联，可以得到串联的多位全加器，如下图。本次实验中，将 32 个全加器串联即可。



因此，很容易将组合逻辑转化为 c++ 代码：

```

1.  WORD operator +(WORD left, WORD right)
2.  {
3.      WORD res = WORD();
4.      int carry = 0;
5.      for (int i = 0; i < WORD_LEN; i++)
6.      {
7.          res[i] = left[i] ^ right[i] ^ carry;
8.          carry = left[i] & right[i] | left[i] & carry | right[i] & carr
y;
9.      }
10.     return res;
11. }

```

#### 2.2.6. 分组模块

在 SHA-1 算法中，对每个 512 比特长度的分组  $M_i$ ，要将其再次划分为 16 个字。这里的划分其实就是一个 bitset 向量索引的映射。该映射为

$$f(i, j) = (15 - i) \times \text{WORD\_LEN} + j$$

代码实现为：

```

1.  vector<WORD> divide(bitset<512> M)
2.  {
3.      vector<WORD> W(16);
4.      for (int i = 0; i < 16; i++)
5.          for (int j = 0; j < WORD_LEN; j++)
6.              W[i][j] = M[(15-i) * WORD_LEN + j];
7.      return W;
8.  }

```

#### 2.2.7. SHA-1

首先，对于分组的扩展操作，我们首先需要将输入串中已经有 512 长度的直接输出，再对剩下的余数部分调用 SHA-1-PAD 操作。要注意的是，余数部分小于 447 时只会扩展出一组，否则扩展出两组。

除此以外，只需要按照 SHA-1 算法将上述模块拼接在一起就行了。一个值得注意的地方是，最后要将  $H_0, H_1, H_2, H_3, H_4$  拼接在一起输出，由于他们之前进行的运算是以字为对象的运算，而拼接字的操作又比较麻烦，因此我采用了先将他们转换为字符串，再利用字符串的拼接操作的做法。代码实现如下：

```

1.  bitset<160> SHA1(ifstream &f)
2.  {
3.      vector<bitset<512>> M;
4.      string s;
5.      bitset<1024> x;
6.      s = readfile(f);
7.      int l = s.length();
8.      int L = 1;
9.      //x= SHA1_PAD(s);
10.     int n = 0;
11.     while (l / 512)
12.     {
13.         bitset<512> x_bit(s);
14.         M.push_back(x_bit);
15.         n++;
16.         s = readfile(f);
17.         l = s.length();
18.         L = L + 1;
19.     }
20.     if (l <= 447)
21.     {
22.         x = SHA1_PAD(s,L);
23.         M.push_back(low(x));
24.         n++;
25.     }
26.     else if(l<512)
27.     {
28.         x = SHA1_PAD(s,L);
29.         M.push_back(high(x));
30.         M.push_back(low(x));
31.         n++;
32.     }
33.
34.     WORD k[80];
35.     WORD W[80];
36.     WORD A, B, C, D, E;

```

```

37.     for (int t = 0; t < 80; t++)
38.         k[t] = K(t);
39.     WORD H[5] = {
40.         0x67452301,
41.         0xEFCDA89,
42.         0x98BADCFE,
43.         0x10325476,
44.         0xC3D2E1F0
45.     };
46.     for (int i = 0; i < n; i++)
47.     {
48.         vector<WORD> W(80);
49.         vector<WORD> W1 = divide(M[i]);
50.         for (int i = 0; i < 16; i++)
51.             W[i] = W1[i];
52.         for (int t = 16; t < 80; t++)
53.             W[t] = ROTL(W[t - 3] ^ W[t - 8] ^ W[t - 14] ^ W[t -
16], 1);
54.         A = H[0];
55.         B = H[1];
56.         C = H[2];
57.         D = H[3];
58.         E = H[4];
59.         for (int t = 0; t < 80; t++)
60.         {
61.             WORD temp = ROTL(A, 5) + ft(B, C, D, t) + E + W[t] + k[t];
62.
63.             E = D;
64.             D = C;
65.             C = ROTL(B, 30);
66.             B = A;
67.             A = temp;
68.         }
69.         H[0] = H[0] + A;
70.         H[1] = H[1] + B;
71.         H[2] = H[2] + C;
72.         H[3] = H[3] + D;
73.         H[4] = H[4] + E;
74.     }
75.     string h[5];
76.     for (int i = 0; i < 5; i++)
77.         h[i] = H[i].to_string();
78.     return bitset<160>(h[0] + h[1] + h[2] + h[3] + h[4]);

```

### 2.3. 文件结构

文件名	功能
SHA.h	声明 SHA 算法的接口
util.h	声明通用的一些库和数据类型，以及调试的宏定义
SHA.cpp	定义 SHA 算法相关操作
main.cpp	测试主程序

### 2.4. 代码测试

先对 ppt 中的数据进行测试，首先将 main 中的宏定义 MY\_TEST 注释掉

```
//#define MY_TEST
#ifndef MY_TEST
#define PPT_TEST
string default_src = "message2.txt";
string default_dst = "MAC2.txt";
#else
string default_src = "message.txt";
string default_dst = "MAC.txt";
#endif // !MY_TEST
```

然后在测试中设置明文路径和消息摘要路径为默认值，如下图

```
D:\workspace\crypto\SHA-1\Debug\SHA-1.exe
1 for SHA-1 algorithm
0 for exit
choose: 1

input the path of message is:m
input the path of mac is:mac
open message file successfully
open MAC file successfully

return menu(1) or exit(0)
```

此时输出的文件为：

SHA-1	""	da39a3ee5e6b4b0d3255bfef95601890afd80709
	MAC2.txt - 记事本	
	文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
	da39a3ee5e6b4b0d3255bfef95601890afd80709	

同理进行 PPT 中的另外几组测试，分别为：

	"a"	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8
	MAC2.txt - 记事本	
	文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8	
	"abc"	a9993e364706816aba3e25717850c26c9cd0d89d
	MAC2.txt - 记事本	
	文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
	a9993e364706816aba3e25717850c26c9cd0d89d	
	"abcdefghijklmnopqrstuvwxyz"	32d10c7b8cf96570ca04ce37f2a19d84240d3a89
	MAC2.txt - 记事本	
	文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
	32d10c7b8cf96570ca04ce37f2a19d84240d3a89	

由于 PPT 中的测试文件长度较短，我自己拟定了一些较长的测试文件，并与开源的 SHA-1 算法进行比对，链接为：[在线加密解密 \(oschina.net\)](http://oschina.net)。为了测试分组的效果，文件大小至少要  $> 512bit = 64byte$ 。测试自己的文件时，需要将 main 中的 MY\_TEST 取消注释，如下图

```
#define MY_TEST
#ifndef MY_TEST
#define PPT_TEST
string default_src = "message2.txt";
string default_dst = "MAC2.txt";
#else
string default_src = "message.txt";
string default_dst = "MAC.txt";
#endif // !MY_TEST
```

我自己的测试数据为如下：

message.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
This is the test of the SHA-1 algorithm.
ID: [ ]
aut [ ]
dat [ ]

这是一个 79 字节的文件，他能得到多个 512 长度的分组。将我的代码得到的输出和开源网站得到的进行对比，结果如下：



可以看到测试通过。



## 三、RSA 加密和解密

### 1. NTL 库的配置和使用

#### 1.1. NTL 简介

NTL 是一种高性能的可移植 C++ 库，提供用于处理带符号的任意长度整数以及整数和有限域上的矢量，矩阵和多项式的数据结构和算法。

#### 1.2. NTL 的安装和配置

本次实验中采用的 NTL 版本为：WinNTL-11\_5\_1

将 NTL 安装好后，在 vs2019 中创建一个静态库新项目。将 NTL 源码包中 src 目录的文件全部添加到工程中。

然后在工程上右键-属性-c++配置-常规，在附加包含目录中添加 NTL 源码包中 include 目录的路径，修改 SDL 检查为否。在 c++配置中选择预编译头，设置为不使用预编译头。然后正常编译即可，编译完成后会生成 .lib 文件。

使用该静态库时，需要和上述步骤一样设置附加包含目录，修改 SDL 检查。然后选择链接器-常规，在附加库目录中将 lib 的文件加入其中。最后选择链接器-输入，将生成的 lib 文件名加入到附加依赖项中。

参考教程为：[\(271 条消息\) NTL 库编译和使用配置\\_zfy1996 的博客-CSDN 博客\\_ntl 库编译](#)

官方使用文档为：[A Tour of NTL: Examples: Big Integers \(libntl.org\)](#)

大整数库文档：[ntl/ZZ.txt at main · libntl/ntl \(github.com\)](#)

### 2. 实现 RSA 算法

#### 2.1. RSA 加密解密算法

RSA 密码体制

设  $n = pq$ ，其中  $p, q$  是素数

$$ab \equiv 1 \pmod{\varphi(n)}$$

则加密函数

$$e_K(x) = x^b \pmod{n}$$

解密函数

$$d_K(y) = y^a \pmod{n}$$

公钥为  $(n, b)$ ，私钥为  $(p, q, a)$ 。

因此对于 RSA 的参数，首先要实现大素数  $p$  和  $q$  的生成，然后计算  $\varphi(n) = (p-1)(q-1)$ ，再在  $1 \sim \varphi(n)$  中选取一个随机数  $b$ ，使之存在  $\pmod{\varphi(n)}$  的逆元  $a$ ，则公钥为  $(n, b)$ ，私钥为  $(p, q, a)$ 。

#### 2.2. 模块实现

##### 2.2.1. 生成大素数

大素数的生成方式是，在一定范围内随机生成大整数，然后对其进行两次素性检验。

第一次用小素数去做试除，这样能保证生成的大整数没有小的素因子。第二步则

Miller-Rabin 素性检测原理是，对于同余方程

$$x^2 \equiv 1 \pmod{n}$$

一定有  $x \equiv \pm 1 \pmod{n}$ 。

若  $n$  是素数，则该方程只有这两个平凡根。若  $n$  不算素数，则由中国剩余定理，知对于

$$n = \prod_{i=1}^l p_i^{e_i}, 2 \nmid p_i$$

$x^2 \equiv a \pmod{n}$  有  $N_1 \cdot N_2 \cdots N_l$  个解  $\Leftrightarrow x^2 \equiv a \pmod{p_i^{e_i}}$  有  $N_i$  个解

下证：若有  $\left(\frac{a}{p_i}\right) = 1$ ，则  $N_i = 2$

若  $\left(\frac{a}{p_i}\right) = 1$ ，则  $x^2 \equiv a \pmod{p_i}$  有两根，设为  $x_0$ ，且  $x_0^2 = p_i k + a$ ， $p_i \nmid a$

考虑  $x = x_0 + pt$ ，则

$$x^2 = x_0^2 + p^2 t^2 + 2ptx_0 \equiv a + p_i k + 2p_i t x_0 = a + p_i (2tx_0 + k) \pmod{p_i^2}$$

又因为  $(2x_0, p_i) = 1$ ，故  $2tx_0 + k$  遍历  $\pmod{p_i}$  的完系，故存在  $t$ ，使  $p_i \mid 2tx_0 +$

$k$

此时有， $x^2 \equiv a \pmod{p_i^2}$

同理可以验证，对于  $x^2 \equiv a \pmod{p_i^{\alpha}}$  的根  $x_0$ ，

可以找到唯一的  $t$ ，使  $x_0 + pt$  是  $x^2 \equiv a \pmod{p_i^{\alpha+1}}$  的根

故  $N_i \geq 2$

而对于  $\forall x, x^2 \equiv a \pmod{p_i^{e_i}} \Rightarrow x^2 \equiv a \pmod{p_i} \Rightarrow N_i \leq 2$

综上  $N_i = 2$

进行若干轮 Miller-Rabin 素性检测。由于一次 Miller-Rabin 素性检测是一个错误概率至多为  $1/4$  的 Monte Carlo 算法，故进行多轮以后可以将检测错误的概率保持在一个较低水平。

Miller-Rabin 算法中，取  $a = 1$ ，故只要找到不等于  $\pm 1$  的  $\pmod{n}$  平凡二次剩余，就可以说  $n$  不是素数。

Miller-Rabin(x)

令  $n - 1 = 2^k m, 2 \nmid m$

随机选取  $a, 1 \leq a \leq n - 1$

$b \leftarrow a^m \pmod{n}$

If  $b \equiv 1 \pmod{n}$

Return “n is prime”

For  $i=0$  to  $k-1$

If  $b \equiv -1 \pmod{n}$

Return “n is prime”

Else  $b \leftarrow b^2 \pmod{n}$

Return “n is composite”

代码实现为:

```

1. bool Miller_Rabin(const ZZ& n, const ZZ& a)
2. {
3.     ZZ m;
4.     m = n - 1;
5.     long k = MakeOdd(m);
6.     //std::cout << "m:" << m << ",k:" << k << std::endl;
7.
8.     ZZ b;
9.     PowerMod(b, a, m, n);
10.    if (b % n == 1)
11.        return IS_PRIME;
12.    for (long i = 0; i < k; i++)
13.    {
14.        if (b % n == -1)
15.            return IS_PRIME;
16.        else
17.            b = (b * b) % n;
18.    }
19.    return NOT_PRIME;
20.}
21.bool easy_prime_test(const ZZ& n)
22.{
23.    PrimeSeq seq;
24.    long p = seq.next();
25.    while(p<2000)
26.    {
27.        if (n % p == 0)
28.        {
29.            if (n == p)
30.                return IS_PRIME;
31.            else
32.                return NOT_PRIME;
33.        }
34.        p = seq.next();
35.    }
36.    return IS_PRIME;
37.}
38.ZZ myGenPrime(int l,ZZ p=ZZ(0))
39.{
40.    ZZ n;
41.    ZZ a;
42.    bool Miller_Rabin_test;
43.    RandomLen(n, l-1);

```

```

44.     n = 2 * n - 1;
45.     while (1)
46.     {
47.         n = n + 2;
48.         if (easy_prime_test(n) == NOT_PRIME)
49.             continue; //没通过素数检测, 重新生成
50.         Miller_Rabin_test = IS_PRIME;
51.         for (int i = 1; i <= PRIME_TEST_ROUND; i++)
52.         {
53.             RandomBnd(a, n);
54.             if (Miller_Rabin(n, a) == NOT_PRIME)
55.             {
56.                 Miller_Rabin_test = NOT_PRIME;
57.                 break; //没通过素数检测, 重新生成
58.             }
59.         }
60.         if (Miller_Rabin_test == IS_PRIME)
61.         {
62.             if (n != p) //通过了素数检测, 判断是否是已经生成的素数
63.                 break;
64.         }
65.     }
66.     return n;
67.}

```

除了自己写素数检验的方法以外, NTL 库中带有生成一定位数素数的函数, 即

```

void GenPrime(ZZ& n, long l, long err = 80);
ZZ GenPrime_ZZ(long l, long err = 80);
long GenPrime_long(long l, long err = 80);

// GenPrime generates a random prime n of length l so that the
// probability that the resulting n is composite is bounded by 2^(-err).

```

直接利用该函数, 能获得一个效率较高的大素数生成方式。

### 2.2.2. 生成密钥

获得了大素数以后, 还需要随机生成一个整数  $b$ ,  $1 < b < \varphi(n)$ ,  $\gcd(\varphi(n), b) = 1$ , 其中  $n = pq$ 。然后求出  $b \bmod \varphi(n)$  的逆元  $a$ 。则公钥为  $(n, b)$ , 私钥为  $(p, q, a)$ 。

代码实现为:

```

1. Key RSA(int key_len)
2. {
3.     ZZ p, q;
4.     int l;
5.     try {
6.         if (key_len != PRIME_LEN1 && key_len != PRIME_LEN2)
7.             throw std::string("key length is not support");

```

```

8.         else
9.             l = key_len;
10.        }
11.        catch (std::string e)
12.        {
13.            std::cout << e << std::endl;
14.        }
15.
16.        GenPrime(p, l);
17.        GenPrime(q, l);
18.        bool flag = Miller_Rabin(p, RandomBnd(p));
19.        //p = myGenPrime(l);
20.        //q = myGenPrime(l, p);
21.
22.        ZZ phi;
23.        mul(phi, p - 1, q - 1);
24.
25.        ZZ a, b, d;
26.        do {
27.            RandomBnd(b, phi);
28.            GCD(d, b, phi);
29.        } while (b <= 1 || d != 1);
30.
31.        InvMod(a, b, phi);
32.        ZZ n;
33.        mul(n, p, q);
34.        Public_key pub = { n, b };
35.        Private_key pri = { p, q, a };
36.        Key res;
37.        res.pub = pub;
38.        res.pri = pri;
39.        return res;
40.}

```

### 2.2.3. 加密模块和解密模块

RSA 的加密方式是，对于明文  $x$ ，利用公钥加密，加密函数为  $e_k(x) = x^b \bmod n$ 。而解密函数为  $d_k(x) = y^a \bmod n$ 。

可以验证加密函数和解密函数的性质。在知道了 RSA 加密后的密文，以及加密函数的陷门后，可以很容易得到  $y^a = (x^b)^a = x^{ab} = x^{k\varphi(n)+1} \equiv x \pmod n$ ，故 RSA 解密函数是合理的。

```

1. ZZ RSA_En(Public_key pub, ZZ x)
2. {
3.     return PowerMod(x, pub.b, pub.n);
4. }

```

```

5. ZZ RSA_De(Private_key pri, Public_key pub, ZZ y)
6. {
7.     return PowerMod(y, pri.a, pub.n);
8. }

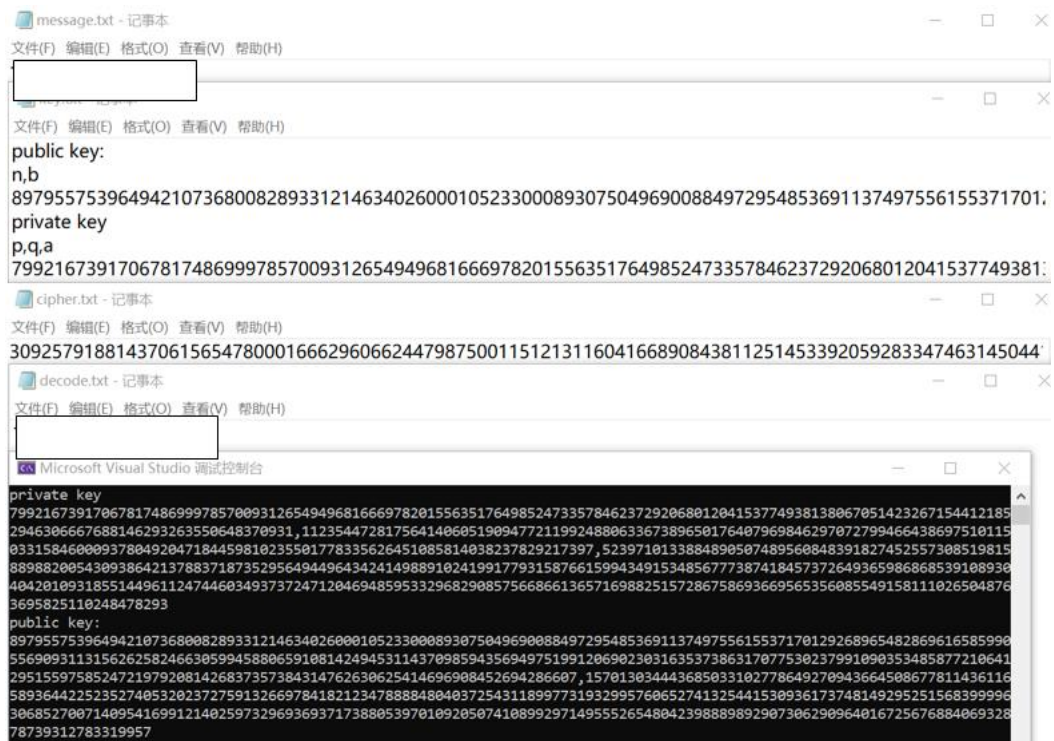
```

### 2.3. 文件结构

文件名	功能
RSA.h	RSA 加密解密的接口
RSA.cpp	RSA 加密解密算法的实现
main.cpp	RSA 的测试程序

### 2.4. 测试

直接在程序中修改加密文件和解密文件路径。会在控制台中输出生成的公钥和私钥，并产生密钥文件。



## 四、签名方案

### 1. RSA 签名方案

一个签名方案是一个满足下列条件的五元组  $(P, A, K, S, V)$

1.  $P$  是由所有可能的消息组成的一个集合
2.  $A$  是由所有可能的签名组成的一个集合
3.  $K$  为密钥空间，它是由所有可能的密钥组成的一个有限集合
4. 对每一个  $k \in K$ ，有一个签名算法  $sig_k \in S$  和一个相应的验证算法  $ver_k \in V$ 。对每一个消息  $x \in P$  和每一个签名  $y \in A$ ，每个  $sig_k: P \rightarrow A$  和  $ver: P \times A \rightarrow \{true, false\}$  都是满足下列条件的函数：

$$ver_k(x, y) = \begin{cases} true, & y = sig_k(x) \\ false, & y \neq sig_k(x) \end{cases}$$

由  $x \in P$  和  $y \in A$  组成的对  $(x, y)$  称为签名消息。

为了保证加密文件的完整性和发报人的信息，需要设计一种类似于手写签名的数字签名。

本次实验实现的是利用 RSA 密码体制的数字签名方案，即 RSA 签名方案：

#### RSA 签名方案

设  $n = pq$ ， $p, q$  是素数。设  $P = A = \mathbb{Z}_n$ ，并定义

$$K = \{(n, p, q, a, b) \mid ab \equiv 1 \pmod{\varphi(n)}\}$$

公钥为  $(n, b)$ ，私钥为  $(p, q, a)$ 。对  $k \in K$ ，定义  $sig_k(x) = x^a \bmod n$  以及  $ver_k(x, y) = true \Leftrightarrow x \equiv y^b \pmod{n}$ ，其中  $x, y \in \mathbb{Z}_n$

### 2. 算法实现

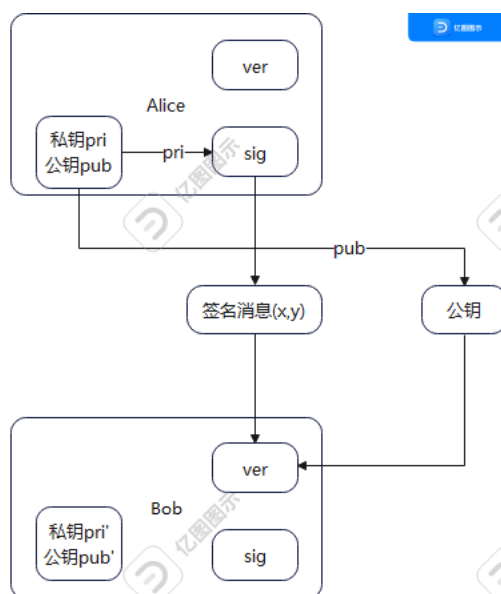
#### 2.1. 类封装

只需要调用 RSA 加密解密算法即可，在这里重新封装一下 RSA 算法。将 RSA 算法改写成成一个类。

先梳理下 RSA 加解密的逻辑。对于用户 Alice，他发报给 Bob，为了说明是自己发报的，他需要附上自己的数字签名。对于采用 RSA 数字签名的情况，他需要一个用于签名的私钥以及一个发送给其他人用于验证他的签名的公钥。因此，由 Alice 自己生成 RSA 加密解密算法的公钥私钥对。而 Bob 收到 Alice 的公钥后，需要保存他，并用于验证签名。此时 Bob 有两个公钥，一个是他自己生成的用于验证自己签名的公钥，另一个是他收到的用于验证 Alice 签名的公钥。因此在 RSA 签名方法的类中，应该用一个变量来保存需要验证签名的对象的公钥，也需要一个函数来获取自己的公钥，一个函数来设置自己用于验证别人签名的公钥。

将 RSA 加密解密算法的公钥私钥对，作为类的私有成员变量，将加密解密方法作为公有成员变量。其中签名算法调用的是自己的私钥，签名验证算法调用的是自己保存的别人的公钥。

整个逻辑的流程图如下：



RSA 密码体制类声明如下：

```

1. class RSA {
2. public:
3.     struct Public_key
4.     {
5.         ZZ n;
6.         ZZ b;
7.     };
8.     Public_key pub; //用于加别人密的公钥
9. public:
10.    RSA();
11.    void setPublicKey(Public_key);
12.    void GenerateKey(int l=PRIME_LEN1);
13.    ZZ encrypt(ZZ);
14.    ZZ decrypt(ZZ);
15.    Public_key GetPublicKey() { return this->key.pub; }
16. private:
17.     struct Private_key
18.     {
19.         ZZ p;
20.         ZZ q;
21.         ZZ a;
22.     };
23.     struct Key
24.     {
25.         Public_key pub;
26.         Private_key pri;
27.     };
28.     Key key;

```



```
29.};
```

而 RSA 签名方案类声明如下：

```
1. class RsaSig{
2. public:
3.     bool ver(ZZ, ZZ);
4.     ZZ sig(ZZ);
5. public:
6.     RSA rsa;
7. };
```

## 2.2. 签名算法和验证算法

对于签名算法，直接调用 RSA 的解密算法即可，而验证算法则需要计算 RSA 的加密算法。即：

$$\begin{aligned} sig_k &\equiv x^a \bmod p \\ ver_k &\equiv x^b \bmod p \end{aligned}$$

代码实现为：

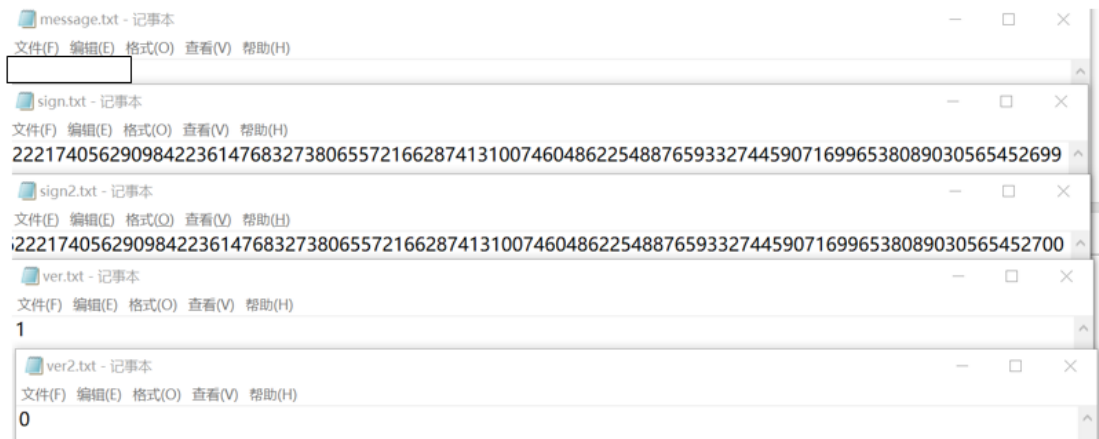
```
1. bool RsaSig::ver(ZZ x, ZZ y)
2. {
3.     return x == this->rsa.encrypt(y);
4. }
5. ZZ RsaSig::sig(ZZ x)
6. {
7.     return this->rsa.decrypt(x);
8. }
```

## 3. 文件结构

文件名	功能
RSA.h	RSA 类的声明
RSA.cpp	RSA 类的实现
RsaSig.h	RSA 签名算法类的声明
RsaSig.cpp	RSA 签名算法类的实现
main.cpp	RSA 的测试程序

## 4. 测试

直接在程序中修改消息文件和输出结果文件路径。用签名方案生成签名后，将生成的签名和修改后的签名分别进行验证，并输出结果到文件。其中，`sign` 是正确的签名，`sign2` 是 `sign+1` 的结果，`ver` 和 `ver2` 分别是他们的验证结果。1 表示验证通过，0 表示验证失败。



## 五、证书方案

### 1. 数字证书的原理

网络用户的证书包含用户的身份信息，公钥以及 TA 对这些消息的签名。证书允许网络用户验证彼此公钥的真实性。一般地，我们假定有一个可信任的权威机构 TA，为网络中的用户签署公钥，其实证书就是权威机构分发的签名。

一个简单的证书方案如下：

向 Alice 颁布证书

1. TA 通过身份证等信息验证 Alice 身份。TA 形成一个传，记为  $ID(Alice)$ ， $ID(Alice)$  中包含 Alice 的身份识别信息。
2. Alice 的秘密签名密钥  $sig_{Alice}$  和相应的公开验证密钥  $ver_{Alice}$  被确定
3. TA 产生对 Alice 身份标识和验证密钥的签名，即：

$$s = sig_{TA}(ID(Alice) \parallel ver_{Alice})$$

把证书

$$Cert(Alice) = (ID(Alice) \parallel ver_{Alice} \parallel s)$$

连同 Alice 的私钥  $sig_{Alice}$  一起传给 Alice

### 2. 数字证书的实现

#### 2.1. 类设计

从数字证书的方案中，可以看到，我们需要一个可信任的权威机构以及参与网络信息传输的用户。

对于一个权威机构，他需要具备给网络中的用户颁布证书的功能，而权威机构 TA 为某个用户颁布证书，其实就是对这个用户的公钥进行签名。因此表示权威机构的类中应该包含之前设计好的 Sig 类。

对于参与信息传输的每个用户，他需要具备向权威机构申请证书的功能，还需要具备验证 TA 颁布证书的功能。申请证书的过程可以通过调用权威机构的函数实现。而验证证书，可以用权威机构的公钥进行验证。

本次小实验中，权威机构的类叫做 TA，设计如下：

```

1. class TA {
2. public:
3.     TA() {};
4.     TA(const string&);
5.     TA(int, const string&);
6.     void operator=(const TA&);
7.     string makeCertificate(Client& const);
8.     string getID() const { return ID; }
9.     Public_key getPubTA() const { return this->sig.rsa.GetPublicKey(); }
10. private:
11.     RsaSig sig;
12.     string ID;
13.     int primeLen;

```

```
14.};
```

可以看到，除了上述分析过的颁布证书功能，这个类中还有获取机构名称的功能 (getID)。这是因为，权威机构不止一个，验证证书时，不仅需要验证证书是权威机构颁布的，还要验证他是我们认定的某个权威机构颁布的。

本次小实验中，用户的类叫做 Client，设计如下：

```
1. class TA;
2. class Client {
3.     friend class TA;
4. public:
5.     Client() {}
6.     Client(const string& id) { ID = id; }
7.     void operator=(const Client&);
8.     string getID() const { return ID; }
9.
10.    void callCertificate(const TA&);
11.    //bool verifyCertificate(Client, const TA&);
12.    bool verifyCertificate(const string&, const TA&);
13.    string getCertificate() const { return Certificate; }
14.    void writeLog(const string&, const string&, bool);
15. public:
16.     RsSig sig;
17. private:
18.     string Certificate;
19.     string ID;
20.};
```

可以看到，除了已经分析过的获取证书，验证证书以外，还有获取用户名称的功能 (getID)，出示用户证书的功能 (getCertificate) 以及写验证日志的功能 (writeLog)。获取用户名的功能在 TA 对某用户的公钥签名时需要用到，出示证书在发报给另一用户时需要用到，而写验证日志是为了记录每一次操作以及便于程序的调试。

## 2.2. 算法实现

### 2.2.1. TA 构造函数

TA 构造函数的目的是，生成 TA 给网络用户的公钥签名的公钥私钥对。因此调用 TA 中的 RsSig 成员，用他内含的 GenerateKey 来生成密钥。

```
1. TA::TA(const string& id)
2. {
3.     primeLen = PRIME_LEN1;
4.     this->sig.rsa.GenerateKey();
5.     this->ID = id;
6. }
7.
8. TA::TA(int len, const string& id)
9. {
10.    try {
11.        if (len == PRIME_LEN1 || len == PRIME_LEN2)
```

```

12.         primeLen = len;
13.     else
14.         throw string("prime length not allowed");
15.     }
16. catch (string e)
17.     {
18.         cerr << e << endl;
19.     }
20. this->sig.rsa.GenerateKey(len);
21. this->ID = id;
22. }

```

### 2.2.2. makeCertificate

首先需要为用户生成公钥私钥对，然后将自己的公钥复制给用户，便于用户其他用户证书的验证。然后按照证书算法的格式，进行计算，并输出证书内容到文件。

本次实验中，我规定的证书格式为：

*Client:*

*<client name>*

*b:*

*<b>*

*n:*

*<n>*

*s:*

*<s>*

*TA\_ID:*

*<TA ID>*

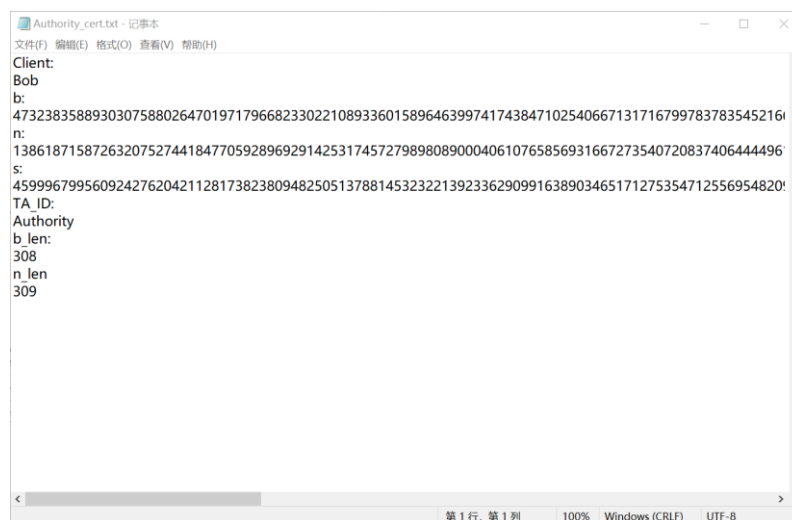
*b\_len:*

*<b length>*

*n\_len:*

*<n length>*

例如下图中的文件就是 TA 生成的一个证书



代码实现为:

```

1. string TA::makeCertificate(Client& user) const
2. {
3.     //为 user 分配 rsa 密钥
4.     int pubKeyLen[2];
5.     user.sig.rsa.GenerateKey(primeLen);
6.     Public_key pub = user.sig.rsa.GetPublicKey(); //user 的加密公钥
7.     pubKeyLen[0] = ZZ2str(pub.b).length();
8.     pubKeyLen[1] = ZZ2str(pub.n).length();
9.
10.
11.    //生成 TA 的签名方案
12.    ZZ sig_obj = strZZ(user.getID() + ZZ2str(pub.b) + ZZ2str(pub.n));
13.    //user.pubTA = this->sig.rsa.GetPublicKey();
14.    Public_key pubTA = this->sig.rsa.GetPublicKey();
15.    ZZ s = this->sig.sig(sig_obj % pubTA.n);
16.
17.
18.    //user.pubTA = this->sig.rsa.GetPublicKey(); //给用户验证签名的公钥
19.    //给 n 和 b 都分配 2*prime_len 的长度来储存
20.
21.    string cert =
22.        "Client:\n"
23.        + user.getID() + "\n"
24.        + "b:\n"
25.        + ZZ2str(pub.b) + "\n"
26.        + "n:\n"
27.        + ZZ2str(pub.n) + "\n"
28.        + "s:\n"
29.        + ZZ2str(s) + "\n"
30.        + "TA_ID:\n"
31.        + this->ID + "\n"
32.        + "b_len:\n"
33.        + int2str(pubKeyLen[0]) + "\n"
34.        + "n_len\n"
35.        + int2str(pubKeyLen[1]) + "\n";
36.
37.    //写证书到文件
38.    string dir = ".\\" + user.getID();
39.    if (_access(dir.c_str(), 0) == -1)
40.        _mkdir(dir.c_str());
41.    string filename = ".\\" + user.getID() + "\\" + this-
>ID + "_cert.txt";
42.    ofstream fcert(filename, ios::out);

```

```

43.     try {
44.         if (!fcert.is_open())
45.             throw string("cert open error");
46.     }
47.     catch (string e)
48.     {
49.         cerr << e << endl;
50.     }
51.     fcert << cert;
52.     fcert.close();
53.
54.     return cert;
55. }

```

### 2.2.3. verifyCertificate

这个函数的实现，主要有以下几个步骤

1. 解析证书文件中的各项数据，即获取证书持有人 ID，公钥 b,n，签名 s
2. 用以上数据，以及 TA 的公钥对签名 s 进行验证，即计算：

$$ver_{TA}(ID \parallel b \parallel n, s)$$

代码实现为：

```

1. bool Client::verifyCertificate(const string& cert, const TA& ta)
2. {
3.     string TName = ta.getID();
4.     stringstream stream(cert);
5.     string ID;
6.     getline(stream, ID);
7.     getline(stream, ID);
8.
9.     string bstr, nstr;
10.    ZZ b, n;
11.    Public_key pub;
12.    getline(stream, bstr);
13.    getline(stream, bstr);
14.    getline(stream, nstr);
15.    getline(stream, nstr);
16.    b = str2ZZ(bstr);
17.    n = str2ZZ(nstr);
18.    pub.b = b;
19.    pub.n = n;
20.
21.    string sstr;
22.    ZZ s;
23.    getline(stream, sstr);

```

```

24.    getline(stream, sstr);
25.    s = str2ZZ(sstr);
26.
27.    ZZ sig_obj = str2ZZ(ID + bstr + nstr);
28.
29.
30.
31.    Public_key pubTA = ta.getPubTA();
32.    this->sig.rsa.setPublicKey(pubTA);
33.
34.
35.    bool is_pass = this->sig.ver(sig_obj % pubTA.n, s % pubTA.n);
36.    if (is_pass)
37.        this->sig.rsa.setPublicKey(pub);
38.
39.    writeLog(ID, TAname, is_pass);
40.
41.    return is_pass;
42.}

```

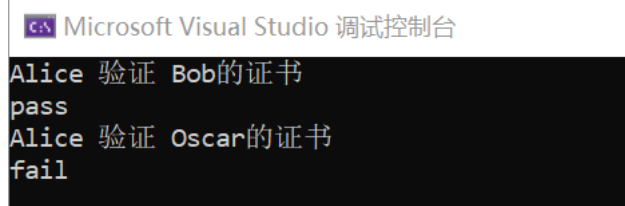
### 3. 文件结构

文件名	功能
RSA.h	RSA 类的声明
RSA.cpp	RSA 类的实现
RsaSig.h	RSA 签名算法类的声明
RsaSig.cpp	RSA 签名算法类的实现
Certificate.h	证书相关类的声明
Certificate.cpp	证书的具体实现
main.cpp	RSA 的测试程序

### 4. 测试

先声明两个 Client 用户，即 Client(“Alice”)和 Client(“Bob”)，然后声明一个可信任的权威机构，即 TA(“Authority”)。为了模拟黑客伪造证书，再声明一个攻击用户，即 Client(“Bob”)，他想仿造 Bob 的证书；声明一个为他伪造证书的机构 TA(“fake”)。

测试如下



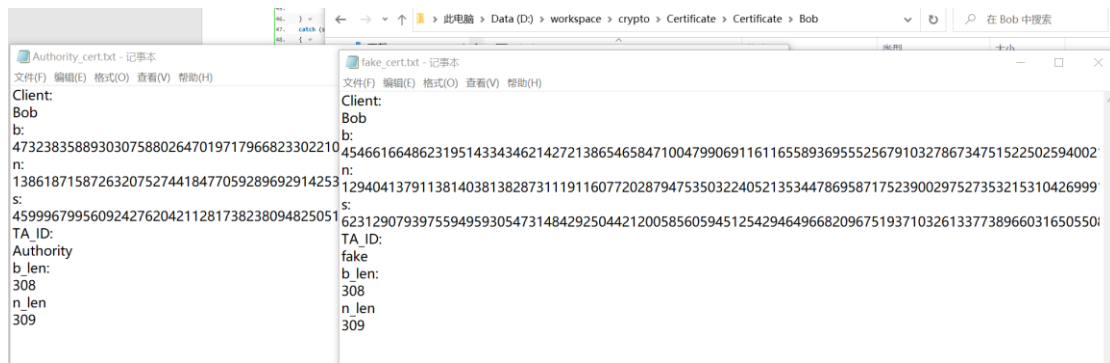
```

Microsoft Visual Studio 调试控制台
Alice 验证 Bob的证书
pass
Alice 验证 Oscar的证书
fail

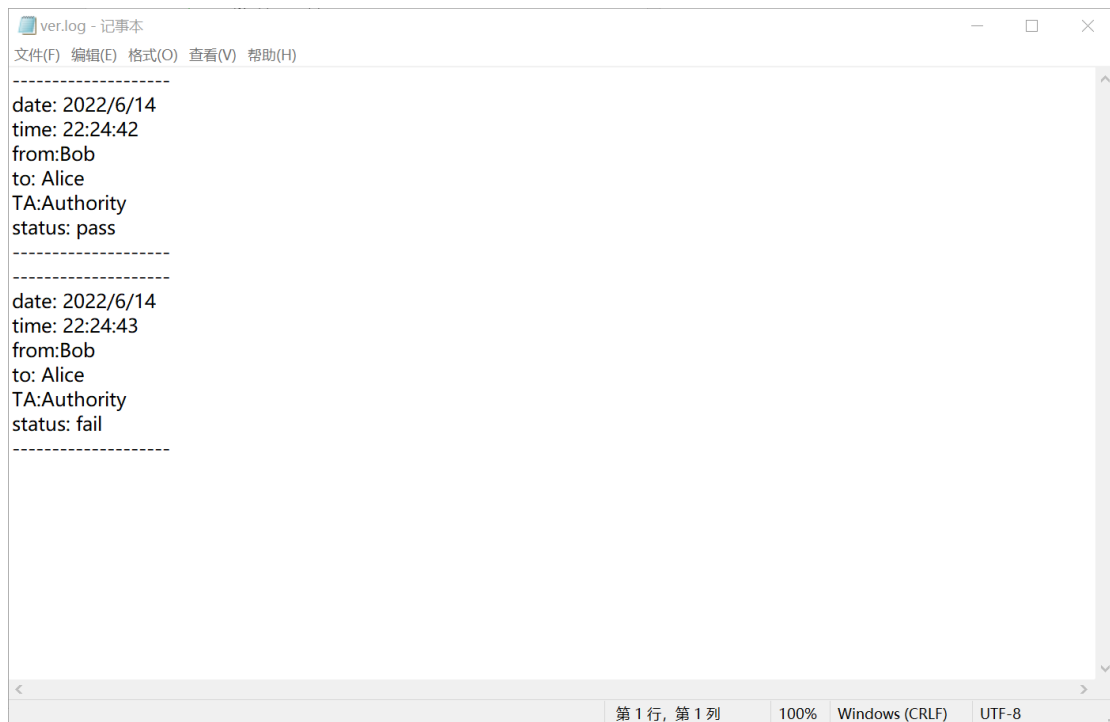
```

观察 Bob 目录下的两个证书文件





可以看到除了颁布证书的机构不同外，他们的公钥私钥对，签名都不同。然后在 Alcic 目录下观察他检验证书的日志文件，结果如下：

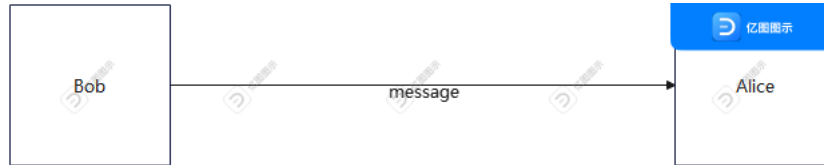


可以看到，第一次正确的证书验证通过，第二次伪造的证书无法通过。

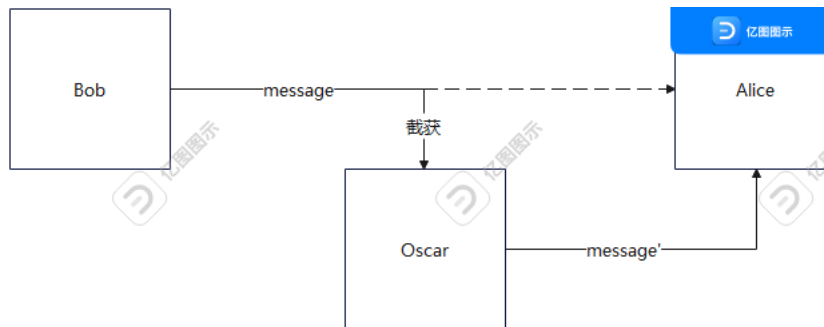
## 六、文件加密系统

### 1. 文件传输的安全性分析

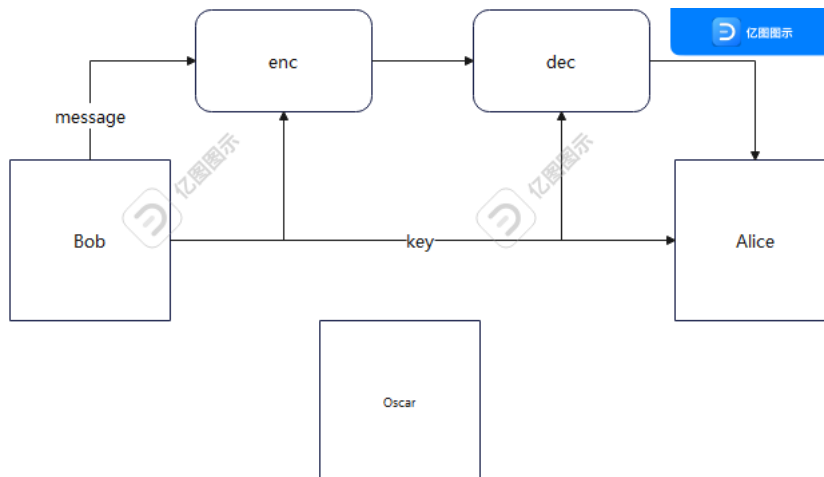
考虑文件传输的模型，由 Bob 给 Alice 发送信息。



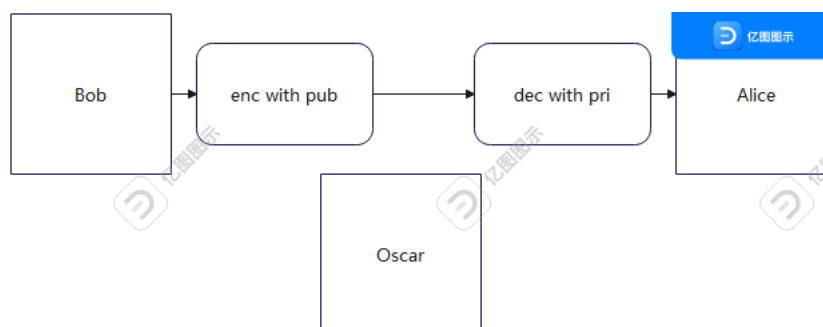
在最简单的文件传输模型中，如果有一个不怀好意的监听者 Oscar，则 Oscar 可以获取他们传输的内容，甚至篡改传输内容。



为了避免这种情况 Alice 和 Bob 可以提前商量好一个统一的密钥，用这个密钥对传输的信息利用对称加密算法进行加密。但是对称加密算法的一个缺点是，通信双方需要提前协商好密钥，且用一个安全的信道传输，如果密钥泄露，则加密的保密性就丧失了。

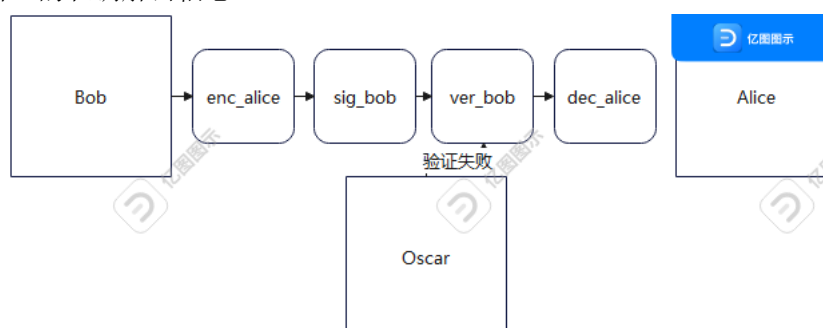


而非对称的加密算法可以避免这个问题，由于私钥只有解密者自己持有，公钥是给所有人的，这种情况下就不需要考虑私钥的安全传输问题。



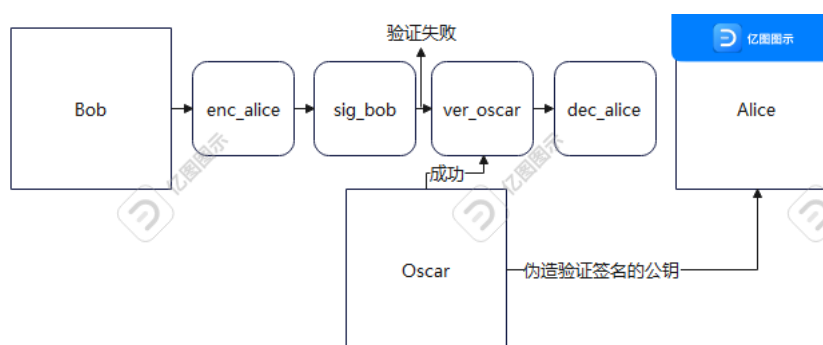
而这种方法缺点是，虽然 Oscar 不能对 Bob 发送的信息进行监听或者修改，但是他也能用 Alice 的公钥给 Alice 发送消息，如此一来 Alice 就无法判断哪个是 Bob 发来的消息了。

类似现实中的手写签名，数字签名是每个网络用户独一无二的标志。Bob 发送信息给 Alice 时，用自己的私钥产生一个签名，附在上面，Alice 收到后先验证 Bob 的签名再用自己的私钥解密信息。



由于数字签名一般采用非对称加密，而非对称加密计算量较大，故需要先用 hash 函数对原来的数据进行压缩，然后再产生签名。即先 hash 再签名。

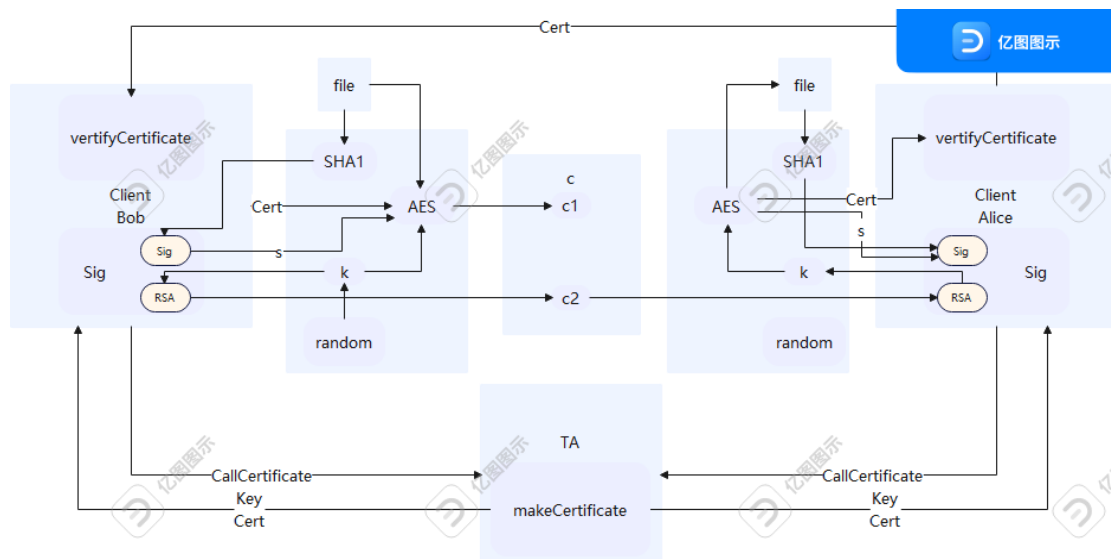
签名体制仍然不够完美，因为签名的公钥如果被 Oscar 截取，并伪造自己的公钥给 Alice，那么 Oscar 就可以随意伪造 Bob 的信息了。



所以，我们最终要做的是保证这个公钥来源的准确性。而这个准确性建立在一个可信任的权威机构 TA 上，他对每个用户的公钥进行签名，确保他们的公钥都是自己的，而不是顶替他人的，然后再将他自己验证签名的公钥发布给所有用户。每个用户收到公钥时，先验证证书，确认公钥来源的准确性。

## 2. 文件加密系统的结构和流程图

由前述的安全性分析知道，我们需要一个加密系统加密文本，需要用户的签名来确保是某人发送的，需要 hash 函数来减小签名的计算量，需要一个颁布证书的权威机构来确保收到的公钥的来源。故可以得到加密系统的结构如下：



利用前面的实验，可以拼装出一个文件加密系统。

## 3. 加密系统的实现

### 3.1. 类型转换模块

由于每个模块的数据类型不同，因此我们需要编写各类型之间的转换。

每个模块需要的自定义数据类型统计如下：

模块名	数据类型
CBC	string
RSA	ZZ
AES	Key

故我们需要如下的类型转换函数

```

1. string ZZ2str(const ZZ&);
2. ZZ str2ZZ(const string&);
3. vector<string> stringSplit(const string&, char);
4. string vec2str(vector<char>);
5. vector<char> str2vec(const string&);
6. ZZ Key2ZZ(const Key&);
7. void ZZ2Key(const ZZ&, Key&);

```

### 3.2. 加密模块

按照文件加密系统的结构图，我们将各加密模块整合到一个类 Sys 中。先研究加密的顺序和过程。

Bob 加密的步骤如下：

1. 输入文件 m
2. 用 SHA-1 算法生成 m 的散列值 SHA-1(m)
3. 用 Bob 的签名私钥对 SHA-1(m) 签名，得到 s
4. 生成随机的 128 比特会话密钥 k
5. 在 CBC 模式下用 AES 会话密钥 k 加密 m||s||Bob 的签名公钥证书，得到 c1
6. 用 Alice 的证书，使用证书中 Alice 的公钥加密 k，得到 c2
7. 向 Alice 发送 c=c1||c2

代码实现为：

```

1. EncInfo Sys::encrypt(const string& dir, const string& cert)
2. {
3.     ifstream m(dir, ios::in);
4.     try {
5.         if (!m.is_open())
6.             throw string("file open error");
7.     }
8.     catch (string e)
9.     {
10.        cerr << e << endl;
11.    }
12.
13.    cout << "生成散列值" << endl;
14.    ShaVal sha1 = SHA1(m); //生成文件 m 散列值
15.
16.    m.close();
17.
18.    cout << "生成" << this->client.getID() << "的签名" << endl;
19.    ZZ s = this->client.sig.sig(ShaVal2ZZ(sha1)); //用发送方 Bob 的私钥对散列值
    签名，得到签名 s
20.
21.    cout << "生成 128 比特的随机 AES 会话密钥 k" << endl;
22.    Key k;
23.    randomKey(k); //生成 aes 的 128bit 随机密钥
24.
25.    cout << "在 CBC 模式下用 AES 会话密钥加密信息" << endl;
26.    CBCAES cbc;
27.    m.open(dir, ios::out);
28.    string mstr = readFile(m);
29.    string sstr = ZZ2str(s);
30.    string certstr = this->client.getCertificate();

```

```

31.     string obj =
32.         mstr + "\n$"
33.         + sstr + "\n"
34.         + certstr;
35.     string c1=cbc.CBCEncryption(obj,k);
36.
37.     cout << "用接收方公钥加密 k" << endl;
38.     ZZ kzz = Key2ZZ(k);
39.     ZZ c2 = this->client.sig.rsa.encrypt(kzz);
40.
41.
42.     EncInfo c;
43.     c.c1 = c1;
44.     c.c2 = c2;
45.
46.
47.     cout << "向接收方发送 c" << endl;
48.     return c;
49.}

```

Alice 解密的步骤如下:

1. 用 Alice 的加密私钥解密 c2 得到 k
2. 用 k 解密 c1, 得到 m||s||Bob 的签名公钥证书
3. 验证 Bob 的签名公钥证书, 如果通过, 继续操作, 否则退出
4. 计算 SHA-1(m)
5. 用 Bob 的签名公钥证书中的公钥验证 s 是否是 SHA-1(m)的正确签名。如果通过, 则是来自 Bob 的信息。

代码实现如下:

```

1. void Sys::decrypt(EncInfo c,const string& dir)
2. {
3.     ofstream m(dir, ios::out);
4.     assert(m.is_open());
5.
6.     cout << "用私钥解密 k" << endl;
7.     ZZ kzz = this->client.sig.rsa.decrypt(c.c2);
8.     Key k;
9.     ZZKey(kzz, k);
10.
11.     cout << "用 k 解密 c1" << endl;
12.     CBCAES cbc;
13.     string buf = cbc.CBCDecryption(c.c1, k);
14.     stringstream stream(buf);
15.

```

```

16.  string mstr, sstr, cert;
17.  //getline(stream, mstr);
18.  readMessage(stream, mstr);
19.  getline(stream, sstr);
20.  readCert(stream, cert);
21.  m << mstr;
22.  m.close();
23.  ifstream mopen(dir, ios::in);
24.
25.  cout << "验证证书" << endl;
26.  assert(this->varify(cert));
27.  ShaVal sha1 = SHA1(mopen);
28.
29.  cout << "验证签名" << endl;
30.  bool is_Bob = this->client.sig.ver(ShaVal2ZZ(sha1), str2ZZ(sstr));
31.
32.
33.  if (is_Bob)
34.      cout << SUCCESS << endl;
35.  else
36.      cout << ERROR << endl;
37.}

```

上述代码中较为复杂的一部分是 CBC 模式下 AES 的加解密。

首先需要定义好一个便于处理的串连接方式，然后找出对应的输出处理手段。我对 CBC 输入文件的格式规定如下：

```

string obj =
    mstr + "\n$"
    + sstr + "\n"
    + certstr;

```

其中“\n\$”是为了便于分割开数据。因为第一段 m 数据是加密的文本内容，中间是可能存在换行符号的，用换行符号后的“\$”来和普通的换行区分。而 sstr 是签名转化出来的字符串，这是一行的数据，可以直接 getline 处理，而 certstr 是证书文件，证书的格式和上一个实验中证书格式相同。

而为了读取一段文本，针对读取 m 和证书，分别写了两个不同的段读取函数，代码实现如下：

```

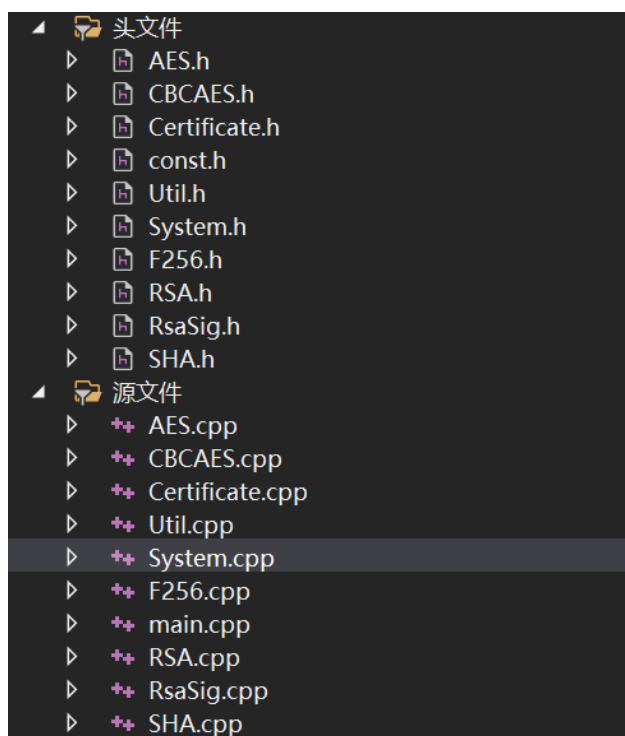
1. void readMessage(stringstream& stream, string& m)
2. {
3.     char ch;
4.     while (1)
5.     {
6.         ch = stream.get();

```

```
7.         if (ch == '\n' && stream.peek() == '$')
8.         {
9.             stream.get();
10.            break;
11.        }
12.        else
13.            m = m + ch;
14.    }
15.}
16.
17.void readCert(stringstream& stream, string& cert)
18.{
19.    string temp;
20.    while (1)
21.    {
22.        getline(stream, temp);
23.        if (stream.eof())
24.            break;
25.        cert = cert + temp + '\n';
26.    }
27.}
```

其中，readMessage 是读取 c1 解密后的 m，readCert 是读取 c1 解密后的证书。

#### 4. 文件结构



其中 Util.h 是类型转换函数的声明，Util.cpp 是类型转换函数的定义。  
System.h 和 System.cpp 包含了用户加密和解密的类和方法。



## 5. 测试

本次实验中文件加密系统的一个前提是，名字就代表了向 TA 申请证书的身份认证，因此不应该重复。

我们假定是用户 Bob 和 Oscar 给用户 Alice 传送数据，他们都向名为 Authority 的 TA 申请了证书。这里测试的是，能得到公钥来源的功能。

然后再随意设置一个证书，对他进行验证，测试证书验证的功能。

测试的顺序可以从代码中看出，代码如下：

```
1. #ifndef DEBUG
2.
3.
4.
5.     bool is_pass;
6.
7.     TA ta("Authority");
8.     Client Alice("Alice");
9.     Client Bob("Bob");
10.
11.     Sys desys(Alice, ta);
12.
13.     string aliceCert = desys.client.getCertificate();
14.     cout << "将 Alice 的证书发送给 Bob" << endl;
15.
16.     Sys ensys(Bob, ta);
17.     cout << "验证证书" << endl;
18.     assert(ensys.verify(aliceCert));
19.
20.
21.     cout << "文件加密" << endl;
22.     EncInfo info = ensys.encrypt(message, aliceCert);
23.     cout << "文件解密" << endl;
24.     desys.decrypt(info, plaintext);
25.
26.     cout << "以下是伪装证书的测试" << endl;
27.     ifstream fake("fakeCert.txt", ios::in);
28.     string fakeCert = readfile0(fake);
29.     cout << "验证证书" << endl;
30.     desys.verify(fakeCert);
31.
32.     cout << "以下是其他用户发送文件给 Alice 的测试" << endl;
33.     Client Oscar("Oscar");
34.     Sys ensys2(Oscar, ta);
35.     cout << "验证证书" << endl;
36.     assert(ensys2.verify(aliceCert));
```

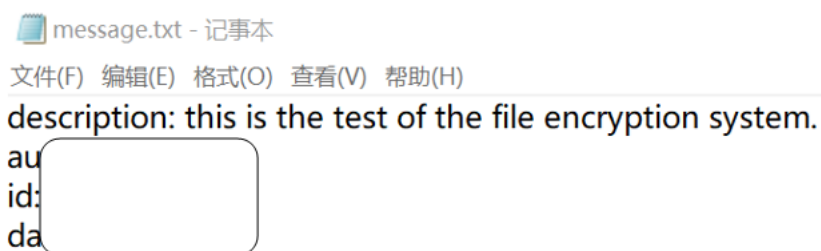
```

37.     cout << "文件加密" << endl;
38.     EncInfo info2 = ensys2.encrypt(message2, aliceCert);
39.     cout << "文件解密" << endl;
40.     desys.decrypt(info2, plaintext2);
41. #endif

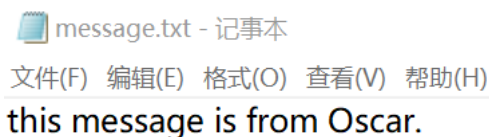
```

故测试的顺序是，先检测 Bob 发送的，再检测成伪造成 Bob 发送的，然后检测 Oscar 发送的。

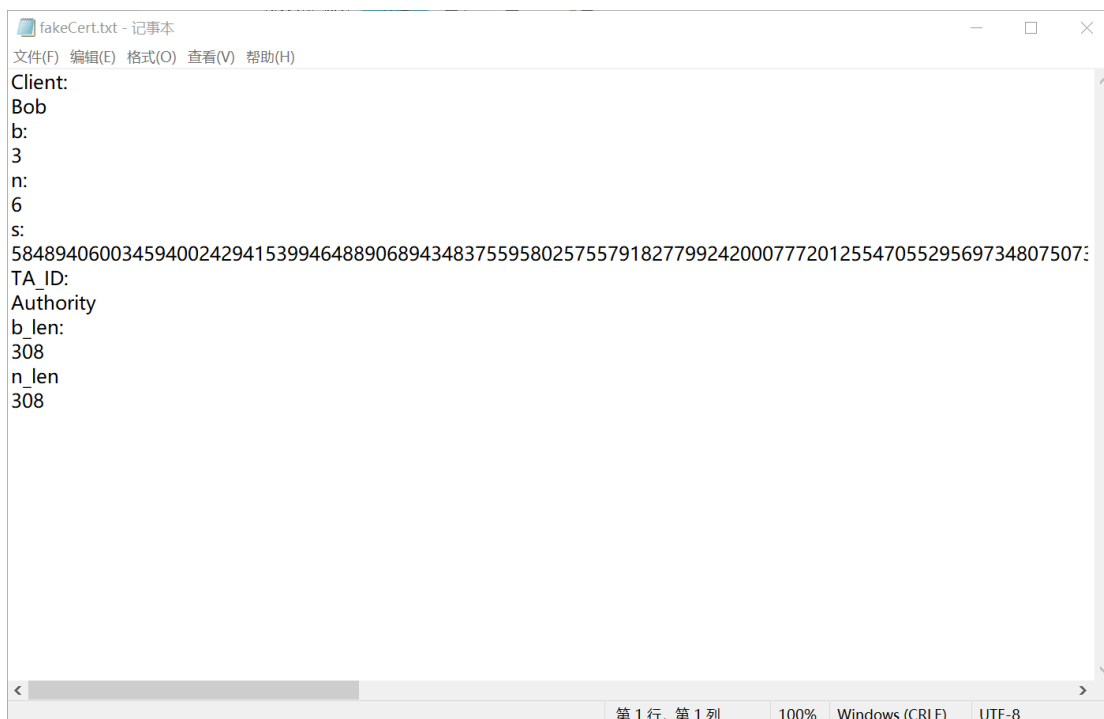
其中 Bob 传送的文件是



Oscar 传送的文件是：



伪造的证书是：



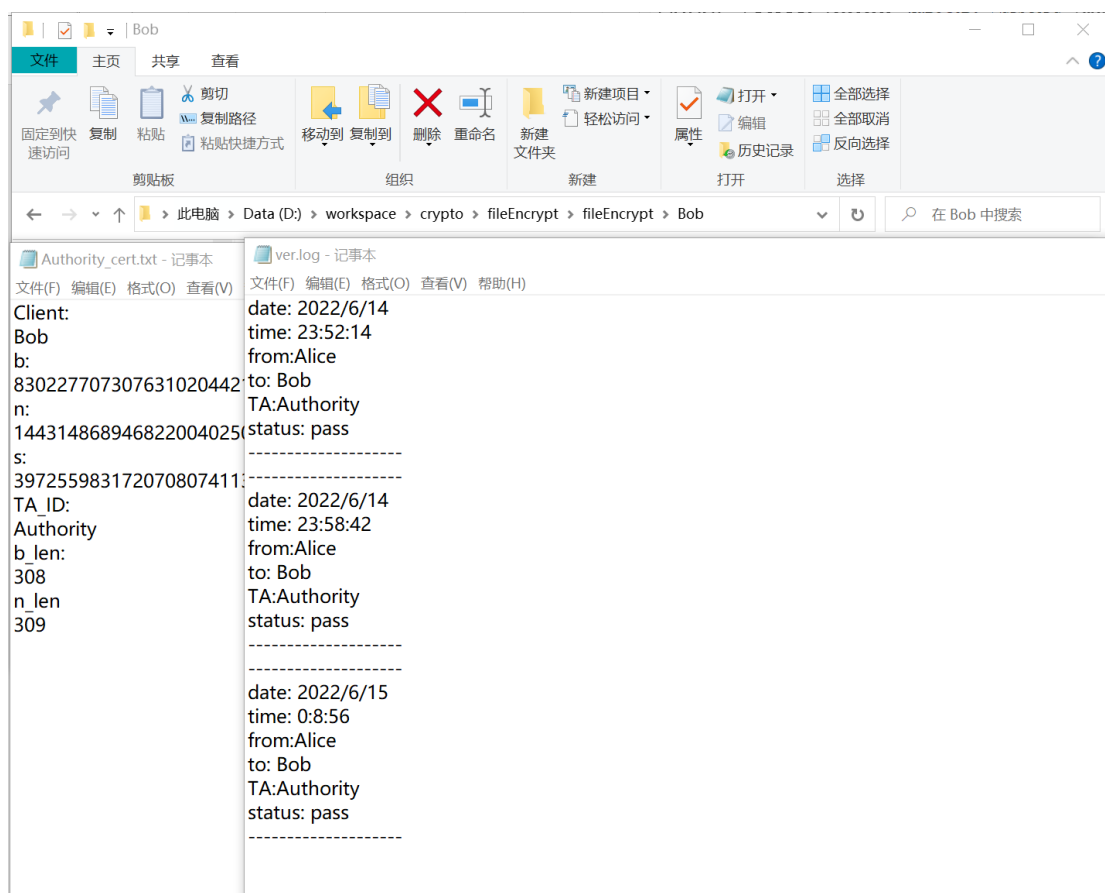
测试结果如下：

对于 Bob 传送给 Alice 的数据，有如下输出

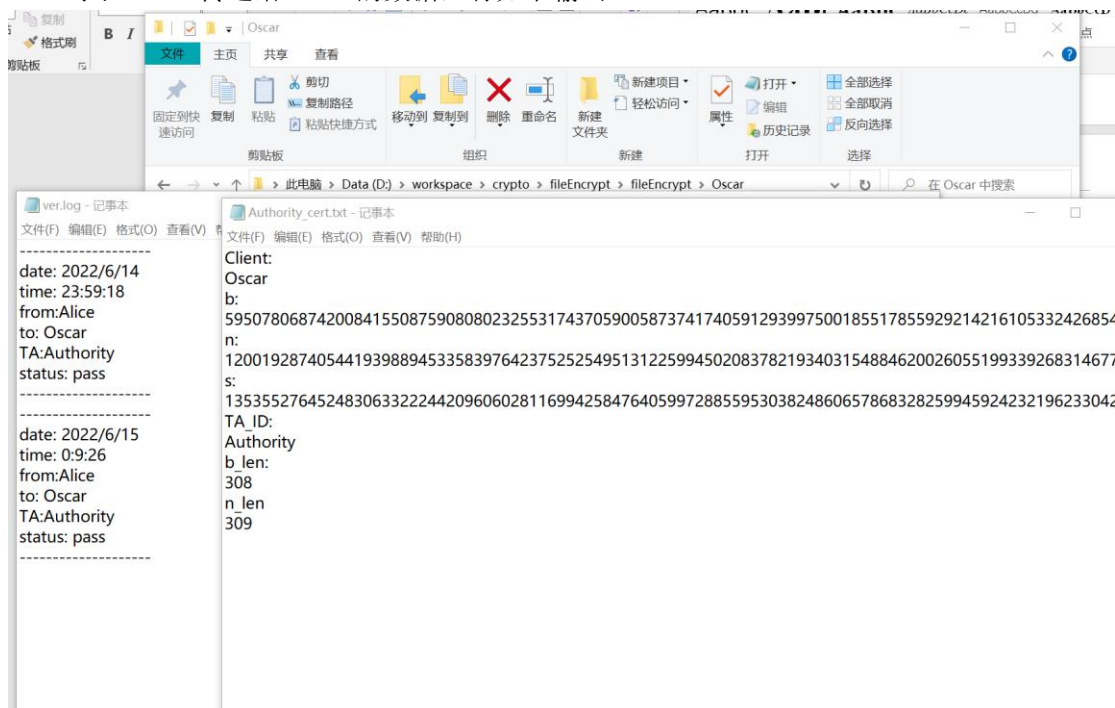
C:\D:\workspace\crypto\fileEncrypt\Debug\fileEncrypt.exe

```
生成Authority为用户Alice生成的证书和密钥
将Alice的证书发送给Bob
生成Authority为用户Bob生成的证书和密钥
验证证书
验证通过
文件加密
生成散列值
生成Bob的签名
生成128比特的随机AES会话密钥k
在CBC模式下用AES会话密钥加密信息
用接收方公钥加密k
向接收方发送c
文件解密
用私钥解密k
用k解密c1
验证证书
验证通过
验证签名
验证通过
```

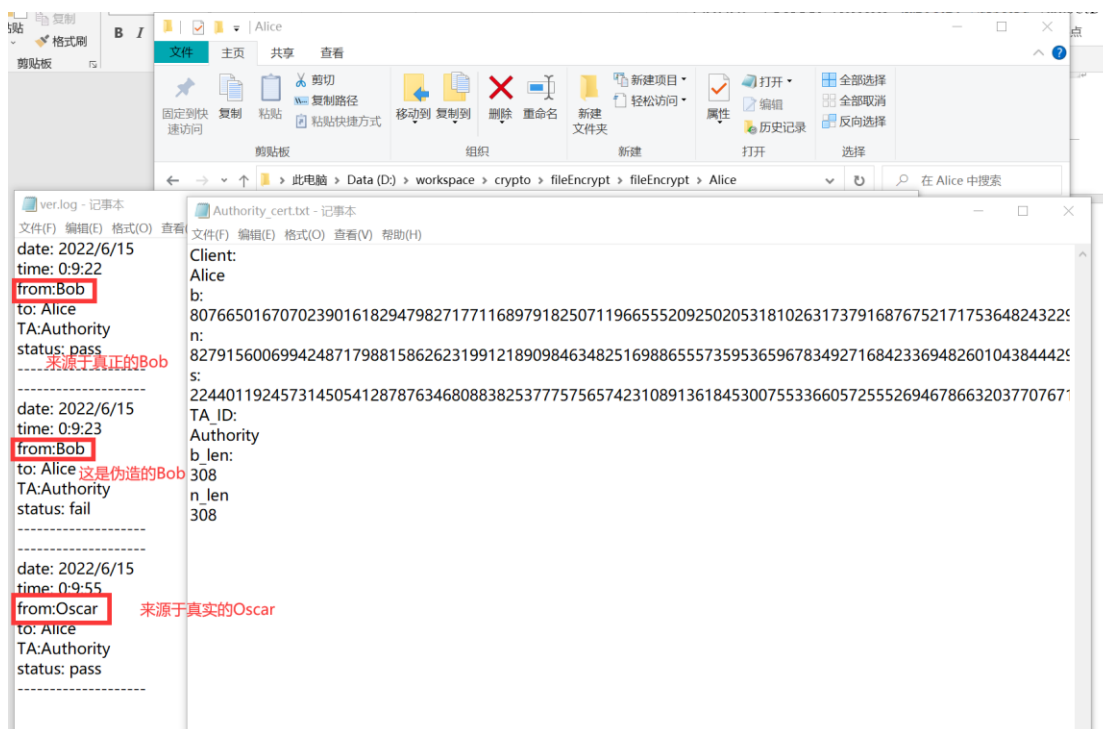
查看 Bob 文件夹，可以看到 Bob 的证书，以及他对 Alice 证书的验证情况。



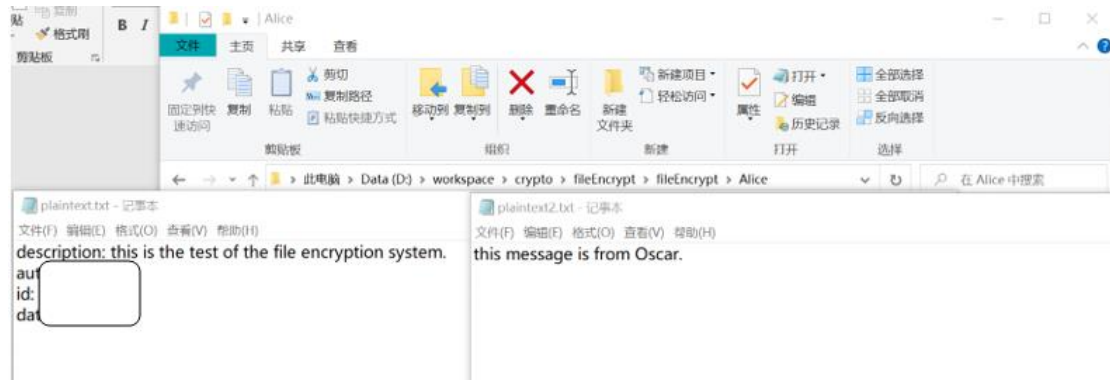
对于 Oscar 传送给 Alice 的数据，有如下输出：



然后查看 Alice 文件夹，查看他的证书，以及他依次检验 Bob 证书，伪造证书，Oscar 证书的情况。



最后查看 Alice 文件中的 plaintext.txt 和 plaintext2.txt，他们分别是 Bob 和 Oscar 加密文件解密后的结果。结果为：



## 七、心得体会

本次课程设计，涉及了 AES 加密，CBC 分组模式，安全哈希算法，RSA 加密，RSA 签名，证书方案，以及最后综合起来的文件加密系统。整个过程不仅是对我学习密码学的一个推动力，也加深了我对密码算法的理解，更让我从宏观的角度学习了各种密码体制，安全体制的优缺点。而且采用 C++ 实现这些密码学实验，加强了我对 C++ 的运用，学习到了很多库，比如 `std` 的一些容器，特别是其中的 `bitset` 容器以及各种流对象 (`stringstream`, `fstream`)，和数论库 `NTL`。作为信息安全专业的学生，除了数学基础能力以外，编程能力的提高也是十分重要，此次课程设计就给了我这个机会。

此外，详细的去理解，实现密码学算法，让我体会到了各种密码体制，安全涉及内含的数学知识，让我知道了数学知识在密码学中的重要性和地位，并对此产生了较为浓厚的兴趣。我私下也去了解了一些密码学的常识，以及密码学在工业界的地位，此次实验也许是引领我进入相关行业的第一盏灯。