



# 现代密码学课程设计说明书

学号：1952650

姓名：陈子翔

## 目 录

课程设计题目描述.....	1
1 程序设计说明.....	1
1.1 编程语言选择及环境.....	1
1.2 输入方式.....	1
1.3 具体流程.....	2
1.3.1 Alice 生成 RSA 密钥 .....	2
1.3.2 Bob 加密文件 m 并发给 Alice .....	2
1.3.3 Alice 解密恢复文件 m.....	3
1.4 头/源文件划分 .....	3
2 模块实现原理.....	4
2.1 DES 实现原理 .....	4
2.1.1 DES 算法结构图 .....	4
2.1.2 DES 密钥编排算法 .....	5
2.1.3 DES 加解密流程 .....	5
2.2 ANSI 随机数生成原理 .....	5
2.2.1 实现步骤.....	5
2.2.2 代码截图.....	6
2.3 Miller Rabin 素性检测实现原理 .....	7
2.4 RSA 实现原理.....	7
2.4.1 RSA 数学原理.....	7
2.4.2 扩展欧几里得算法.....	8
2.4.3 RSA 加解密.....	8
2.5 AES 实现原理 .....	8
2.5.1 AES 算法结构图 .....	8
2.5.2 加密步骤.....	9
2.5.3 解密步骤.....	9
2.6 CBC 实现原理.....	10
2.6.1 CBC 模式加密.....	10
2.6.2 CBC 模式解密.....	10
3 程序使用说明.....	10
3.1 输入输出格式.....	10
4 执行实例.....	11
4.1 用户友好界面.....	11
4.2 运行结果截图.....	12
5 程序源代码.....	14

## 课程设计题目描述

Alice 生成 RSA 密钥:

1. Alice 生成一对 RSA 公钥 $(n, b)$ 和私钥 $(p, q, a)$ , 其中随机素数 $p, q$ 都为 512 比特长; 或都为 1024 比特长; 两种长度都需支持。  
要求: 基于开源代码 NTL 实现, NTL 为密码界流行的高精度代数数论库。  
注意: 为保证通过素性检测的数是素数的概率足够大, 需测试足够的次数。
2. Alice 把公钥传给 Bob。不需要实现网络传输, 下同。

Bob 加密文件  $m$  并发给 Alice:

1. 输入文件  $m$ ,  $m$  为任意长。如何把文件转成所需的格式, 请自行决定。CBC 模式中, 如果  $m$  的长度不是分组的倍数时, 需要填充, 如何填充请自行决定, 但解密时需要把填充去掉。
2. 生成 128 比特的随机数  $k$  作为临时的会话密钥。
3. 用 Alice 的公钥加密  $k$  得到  $c_1 \leftarrow k^b \bmod n$ 。
4. 用会话密钥  $k$  加密文件  $m$  得到  $c_2$ , 其中加密算法为 AES, 使用 CBC 模式。  
要求: AES 的列混合运算及其逆运算使用课件中提供的快速算法。
5. 把 $(c_1, c_2)$ 发给 Alice。

Alice 解密恢复文件  $m$ :

1. 用 Alice 的 RSA 私钥解密 $c_1$ 得到  $k$ 。
2. 用  $k$  解密 $c_2$ 得到  $m$ 。
3. 输出  $m$ 。

## 1 程序设计说明

### 1.1 编程语言选择及环境

编程语言: C++

环境: 处理器: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz(8 CPUs), ~2.0GHz

内存: 8192MB RAM

编程 IDE: Visual Studio 2019

### 1.2 输入方式

程序中用户交互界面输出到控制台供用户选择使用, 公钥、密文等信息的传输通过文件实现。Alice 生成 RSA 密钥, 并将公钥输出到文件中发送给 Bob, Bob 通过收到的 RSA 公钥加密明文  $m$ , 将加密后的 $(c_1, c_2)$ 输出到文件发给 Alice。Alice 再从文件读入信息并对明文  $m$  进行解密。

## 1.3 具体流程

### 1.3.1 Alice 生成 RSA 密钥

#### 1. 生成 512/1024 比特随机大素数 $p, q$

```
void Prime_Generation(ZZ& p, ZZ& q, int length, ULL Seed, ULL Key1, ULL Key2)
```

① 读入随机产生的 64 位的密钥种子 seed, 密钥 key1 和 key2。

② ANSI 算法产生指定位数的随机数

```
ZZ ANSI(ULL s, int m, ULL k1, ULL k2)
```

    | 使用 DES 加密得到 64 比特的随机数

```
ULL DES_Encrypt(ULL Plain_Text, ULL Secret_Key)
```

```
ULL DES_Decrypt(ULL Encryp_Info, ULL Secret_Key)
```

    || 拼接 m 个产生的 64 位随机数。

③ 应用 Miller Robin 算法对产生的随机数进行素性检测, 不通过则重新调用 ANSI 算法, 通过则返回大素数。

```
bool MillerRabin(const ZZ& n, long t)
```

#### 2. 通过产生的大素数 $p, q$ 产生 RSA 密钥

① 产生公钥( $n, b$ )

    | 生成  $n$ :  $n = pq, \phi(n) = (p - 1)(q - 1)$ 。

    || 生成  $b$ : 得到与  $\phi(n)$  互素的随机数  $b$ 。

```
ZZ b_Generation(ZZ phi_n)
```

② 生成私钥( $p, q, a$ ) (即得出解密指数  $a$ ) , 运用扩展欧几里得算法

```
//生成解密指数 a:  $\phi_n \cdot x + b \cdot y = 1$ , b: 加密指数, y: 解密指数 a
```

```
void Extend_Euclidean(ZZ a, ZZ b, ZZ& x, ZZ& y)
```

#### 3. 将公钥( $n, b$ )输出到文件中, 传送给 Bob。

### 1.3.2 Bob 加密文件 $m$ 并发给 Alice

① Bob 从文件中读入 Alice 产生的 RSA 公钥( $n, b$ )与明文  $m$ 。

② 生成 128 比特的随机数  $k$  作为临时会话密钥。

```
ZZ ANSI(ULL s, int m, ULL k1, ULL k2)
```

③ Bob 利用 RSA 将临时会话密钥  $k$  加密得到密钥密文  $c_1$ 。

```
ZZ RSA_Encrypt(ZZ x, ZZ b, ZZ n)
```

④ 利用会话密钥  $k$  利用 AES 算法加密文件  $m$  得到  $c_2$ , 使用 CBC 模式。

    | AES 加密

```
void Encrypt(ZZ& ciphertext, const ZZ& plaintext)
```

    || CBC 加密

```
void CBC_Encrypt(AES& aes, ZZ& IV, ifstream& infile, ofstream& fout)
```

⑤ Bob 将加密得到的密文( $c_1, c_2$ )发送给 Alice。

### 1.3.3 Alice 解密恢复文件 m

- ① Alice 从文件中读入 Bob 传来的密文( $c_1, c_2$ )。
- ② Alice 使用 RSA 私钥解密密文 $c_1$ 得到会话密钥 k。

```
ZZ RSA_Decrypt(ZZ y, ZZ a, ZZ n)
```

- ③ Alice 利用得到的会话密钥 k 解密 $c_2$ 得到明文 m。

└ CBC 解密

```
void CBC_Decrypt(ZZ& c1, ZZ& IV, ifstream& infile, ofstream& fout, ZZ k_de)
```

└ AES 解密

```
void Decrypt(ZZ& plaintext, const ZZ& ciphertext)
```

- ④ Alice 将明文 m 输出。

## 1.4 头/源文件划分

头文件:

1.Crypto\_Curri\_Design.h 包含的头文件以及函数

```
#include "AES.h"
#include "DES.h"

void CBC_Encrypt(AES& aes, ZZ& IV, ifstream& fin, ofstream& fout);
void CBC_Decrypt(ZZ& c1, ZZ& IV, ifstream& fin, ofstream& fout, ZZ k_de);
//生成加密指数b
ZZ b_Generation(ZZ phi_n);
//生成解密指数a: phi_n*x+b*y=1, b:加密指数, y:解密指数a
void Extend_Euclidean(ZZ a, ZZ b, ZZ& x, ZZ& y);
//RSA加密明文x, 得到密文y
ZZ RSA_Encrypt(ZZ x, ZZ b, ZZ n);
//RSA解密密文y, 得到明文x
ZZ RSA_Decrypt(ZZ y, ZZ a, ZZ n);
//素性检测算法
bool MillerRabin(const ZZ& n, long t);
//生成随机素数
void Prime_Generation(ZZ& p, ZZ& q, int length, ULL Seed, ULL Key1, ULL Key2);
//ANSI随机数生成算法
ZZ ANSI(ULL s, int m, ULL k1, ULL k2);
```

2.DES.h

包含 DES 类的定义即各置换矩阵、类内 DES 加解密算法函数定义。

3.AES.h

包含 AES 类的定义即各置换矩阵、类内 AES 加解密算法函数定义。

源文件:

## 1. Crypto\_Curri\_Design.cpp

主函数所在文件，总体操纵程序运行，控制用户输入输出、各函数调用。

## 2. DES.cpp

包含 DES 类内各函数的定义。

## 3. AES.cpp

包含 AES 类内各函数的定义。

## 4. Prime\_Generation.cpp

包括 Miller Robin 素性检测算法和生成指定位数随机素数的算法。

## 5. RSA.cpp

包括 RSA 私钥生成、加解密的函数定义，包括欧几里得算法、扩展欧几里得算法。

## 6. CBC.cpp

包括 CBC 方式加解密过程。

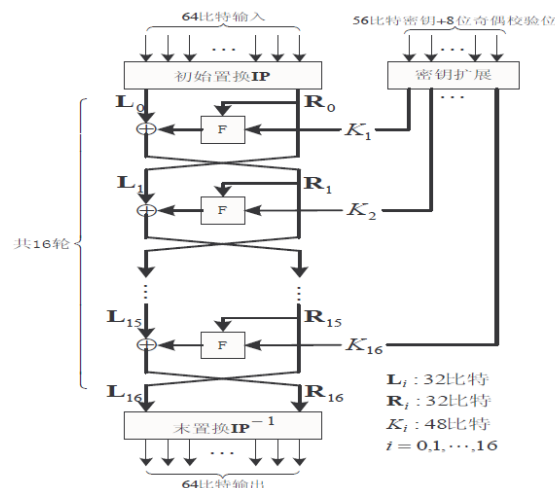
# 2 模块实现原理

## 2.1 DES 实现原理

### 2.1.1 DES 算法结构图

DES 为对称加密，采用 Feistel 轮函数结构。

DES算法结构图：



```
private:
    //密钥编排规则
    const int Key_Schedule[16] = { ... }
    //S盒
    const int S_BOX[32][16] = { ... }
    //初始IP置换
    const int IP_First[64] = { ... }
    //末置换IP-1
    const int IP_Last[64] = { ... }
    //F函数中的扩展置换E
    const int E[48] = { ... }
    //F函数中的置换P
    const int P[32] = { ... }
    //密钥扩展中的置换PC-1
    const int Key_PC1[56] = { ... }
    //密钥扩展中的压缩置换PC-2
    const int Compress_PC2[48] = { ... }
    ULL Secret_Key;
    ULL Encryp_Info;
    ULL Decrypt_Info;
    ULL Plain_Text;
```

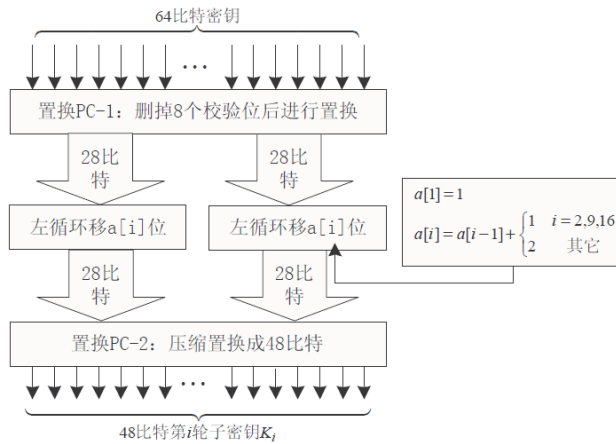
### 2.1.2 DES 密钥编排算法

### 2.1.3 DES 加解密流程

1. 利用 IP\_First 将输入的 64 比特数据进行初始置换。
2. 将置换所得的比特串评分为 Left 和 Right 两部分，进入以下 16 轮循环：
  - ① 利用扩展置换矩阵 E 进行一次扩展置换。
  - ② 置换后的比特串根据密钥编排规律和密钥扩展中的置换 PC\_1 矩阵进行异或运算。
  - ③ 异或得到的比特串通过 S 盒矩阵 S\_BOX 运算。
  - ④ 将比特串通过 F 函数中的置换 P 进行置换。
  - ⑤ 比特串左右异或，left=right，right=运算所得到的比特串。

解密过程与加密过程相同，所应用到的置换矩阵均为加密矩阵的逆，应用扩展密钥的顺序为加密的逆序。

密钥编排算法：



## 2.2 ANSI 随机数生成原理

### 2.2.1 实现步骤

输入：随机（秘密的）64 比特种子  $s$ ，整数  $m$ ，两个 DES 密钥  $k_1, k_2$ 。

输出： $m$  个 64 位伪随机比特串  $x_1, x_2, \dots, x_m$ 。

1. 计算中间值  $I = DES_{k_1}(DES_{k_2}^{-1}(DES_{k_1}(D)))$ ，其中  $D$  表示当前日期/时间的精确表示，DES 表示 DES 加密， $DES^{-1}$  表示 DES 解密。

2. For  $i=1$  to  $m$ :

$$x_i \leftarrow DES_{k_1}(DES_{k_2}^{-1}(DES_{k_1}(I \wedge s)))$$

$$s \leftarrow DES_{k_1}(DES_{k_2}^{-1}(DES_{k_1}(x_i \wedge I)))$$

思路为每次通过 DES 算法生成一个 64 位整数，总共生成  $m$  次，共生成  $m \times 64$  位数据，使用 ZZ 类型数据存储，每次将已有数据左移 64 位并添加本次产生的数据，最后返回结果。

## 2.2.2 代码截图

```
//ANSI随机数生成算法
ZZ ANSI(ULL s, int m, ULL k1, ULL k2)
{
    ZZ Result;
    Result = to_ZZ(0);
    ULL TempI, Xi;
    ULL D = GetSysTimeMicros();
    ZZ TempX;
    TempI = DES_Encrypt(DES_Decrypt(DES_Encrypt(D, k1), k2), k1);
    for (int i = 0; i < m; i++)
    {
        Xi = DES_Encrypt(DES_Decrypt(DES_Encrypt(TempI ^ s, k1), k2), k1);
        s = DES_Encrypt(DES_Decrypt(DES_Encrypt(Xi ^ TempI, k1), k2), k1);
        Uint64_To_ZZ(TempX, Xi);
        Result <<= 64;
        Result = Result + TempX;
    }
    return Result;
}

//获取系统的当前时间，单位微秒(us)
ULL GetSysTimeMicros()
{
    #ifdef _WIN32
        // 从1601年1月1日0:0:0:000到1970年1月1日0:0:0:000的时间(单位100ns)
        #define EPOCHFILETIME (1164447360000000000ULL)
        FILETIME ft;
        LARGE_INTEGER li;
        ULL tt = 0;
        GetSystemTimeAsFileTime(&ft);
        li.LowPart = ft.dwLowDateTime;
        li.HighPart = ft.dwHighDateTime;
        // 从1970年1月1日0:0:0:000到现在的微秒数(UTC时间)
        tt = (li.QuadPart - EPOCHFILETIME) / 10;
        return tt;
    #else
        timeval tv;
        gettimeofday(&tv, 0);
        return (int64_t)tv.tv_sec * 1000000 + (int64_t)tv.tv_usec;
    #endif // _WIN32
    return 0;
}
```



## 2.3 Miller Rabin 素性检测实现原理

由于要求通过素性检测的数是素数的概率足够大，因此需要测试足够多的次数。如产生 512 比特的素数，通过测试的数不是素数的概率接近  $1/2^{512}$ 。因此还需要一个多次测试的循环操作确保概率。

### 算法5.7 Miller-Rabin( $n$ )

```

把  $n-1$  写成  $n-1 = 2^k m$ ，其中  $m$  是一个奇数
随机选取整数  $a$ ，使得  $1 \leq a \leq n-1$ 
 $b \leftarrow a^m \bmod n$  (从  $a^m$  开始检查)
if  $b \equiv 1 \pmod{n}$  (这时形为  $(1, 1, \dots, 1)$ )
    then return ("n is prime")
for  $i \leftarrow 0$  to  $k-1$ 
    do {
        if  $b \equiv -1 \pmod{n}$  (这时形为  $(***-1, 1, \dots, 1)$ )
            then return ("n is prime")
        else  $b \leftarrow b^2 \bmod n$ 
    }
return ("n is composite")

```

## 2.4 RSA 实现原理

### 2.4.1 RSA 数学原理

首先通过 ANSI 算法生成若干个指定位数的大数，同时进行素性检测，直至生成两个指定位数的素数  $p, q$ 。再利用欧几里得算法得到一个与  $\phi(n)$  互素的数  $b$ ，则得到公钥  $n, b$ ；再通过扩展欧几里得算法得到私钥  $a$ ；再通过平方乘算法进行加解密。

### RSA 密码体制

		备注
公开密钥( $n, b$ )	$n = pq$	$\phi(n)$
加密指数 $b$	$b$ 与 $(p-1)(q-1)$ 互素	$= (p-1)(q-1)$
私钥( $a, p, q$ )	$p, q$ 是素数	$ab \equiv 1 \pmod{\phi(n)}$
解密指数 $a$	$a = b^{-1} \bmod (p-1)(q-1)$	要求 $b, \phi(n)$ 互素
加密 $e_K(x)$	$y = x^b \bmod n$	明文 $x \in \mathbb{Z}_n$
解密 $d_K(y)$	$x = y^a \bmod n$	密文 $y \in \mathbb{Z}_n$

RSA 的实现算法：

### 算法5.4 RSA 参数生成算法

- ① 生成两个随机大素数  $p, q$ 。（使用素性检测算法）
- ②  $n \leftarrow pq$ ，且  $\phi(n) \leftarrow (p-1)(q-1)$
- ③ 选择一个随机数  $b(1 < b < \phi(n))$ ，使得  $\gcd(b, \phi(n)) = 1$ （使用 Euclidean 算法判断）
- ④  $a \leftarrow b^{-1} \bmod \phi(n)$ （使用扩展 Euclidean 算法计算）
- ⑤ 公钥为  $(n, b)$ ，私钥为  $(p, q, a)$

### RSA 的加密和解密

- ① 加密： $y \leftarrow x^b \bmod n$ 。（使用平方-乘算法）
- ② 解密： $x \leftarrow y^a \bmod n$ 。（使用平方-乘算法）

## 2.4.2 扩展欧几里得算法

此算法作用为计算  $b$  的逆元，得到解密指数  $a$ 。

具体执行过程采用解此方程： $\text{phi\_n} \cdot x + b \cdot y = 1$ ，得到  $y$  即解密指数  $a$ 。

一般的：

- 对任意整数  $a, b$ ，存在整数  $s, t$  满足  $sa + tb = \gcd(a, b)$

证明思路：由Euclidean算法可见：

$$\begin{aligned}\gcd(a, b) &= r_m = r_{m-2} - q_{m-1}r_{m-1} \\ &= \text{往后依次代入} \dots = sa + tb\end{aligned}$$

- 如果  $\gcd(a, b) = 1$ ，那么有  $sa + tb = 1$ ，即

$$t \equiv b^{-1} \pmod{a}$$

$$\text{设 } r_j = s_j a + t_j b.$$

下面推导  $s_j, t_j$  的递归关系(注意：颜色相同的部分是等价的)：

$$\begin{aligned}r_j &= r_{j-2} - q_{j-1}r_{j-1} \text{ (欧几里德算法)} \\ &= (s_{j-2}a + t_{j-2}b) - q_{j-1}(s_{j-1}a + t_{j-1}b) \text{ (代入 } r_{j-2}, r_{j-1} \text{ 的表达式)} \\ &= (s_{j-2} - q_{j-1}s_{j-1})a + (t_{j-2} - q_{j-1}t_{j-1})b \\ &= s_j a + t_j b\end{aligned}$$

由此得到  $s_j, t_j$  的递归关系（此为课本定理5.1的结论。注意，课本中  $t_j$  的公式有印刷错误）：

$$\begin{aligned}s_j &= s_{j-2} - q_{j-1}s_{j-1} \\ t_j &= t_{j-2} - q_{j-1}t_{j-1}\end{aligned}$$

## 2.4.3 RSA 加解密

使用平方乘算法将明文  $x$  加密，得到密文  $y$ 。

//RSA加密明文 $x$ ，得到密文 $y$

ZZ RSA\_Encrypt(ZZ  $x$ , ZZ  $b$ , ZZ  $n$ )

```
{
    ZZ y;
    y = PowerMod(x, b, n);
    return y;
}
```

使用平方乘算法将密文  $y$  解密，得到明文  $x$ 。

//RSA解密密文 $y$ ，得到明文 $x$

ZZ RSA\_Decrypt(ZZ  $y$ , ZZ  $a$ , ZZ  $n$ )

```
{
    ZZ x;
    x = PowerMod(y, a, n);
    return x;
}
```

## 2.5 AES 实现原理

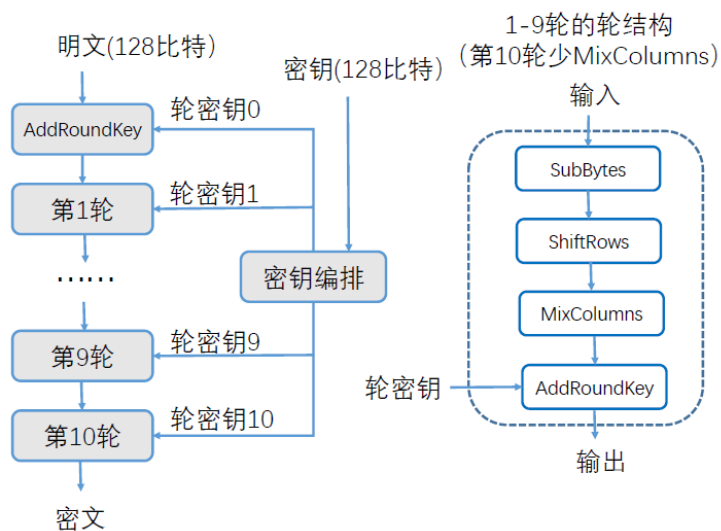
### 2.5.1 AES 算法结构图

四个运算：S 盒运算：SubByte

行移位：ShiftRow

列混合：MixColumn

## 轮密钥异或：AddRoundKey



### 2.5.2 加密步骤

准备步骤：设置 64 比特密钥转为  $4 \times 4$  的 16 进制数组，再进行密钥扩展得到  $4 \times 44$  的扩展密钥。

首先将明文与扩展密钥的第一轮进行异或

进入以下循环，循环 9 次：

- 1.SubBytes：逐字节替换，使用内置矩阵 S\_Box 进行替换操作。
- 2.ShiftRows：平移行运算
- 3.MixColumns：列混合运算
- 4.AddRoundKey：和 roundkey（轮密）做 XOR（位或）操作

最后再进行密钥扩展（Key Expansion）

至此，一个 block 已经加密完成，对于输入的数据划分为不同的 block，同时使用 CBC 方式进行分组加密。

### 2.5.3 解密步骤

解密操作总体上与加密过程相似。首先利用 CBC 解密算法得到分块后的密文串。

- 1.密钥编排算法不变，但轮密钥顺序与加密函数轮密钥相反。
- 2.所有操作逆序进行。
- 3.所有操作均为加密函数的逆操作。

## 2.6 CBC 实现原理

### 2.6.1 CBC 模式加密

实现过程为利用 aes 加密密钥  $k_{en}$ , 生成 128 位随机数  $IV$ , 从文件中读入明文, 每 16 比特 (64 位) 加密一次, 若读到明文结尾不足 16 比特则进行明文填充并加密。将加密后的密文输出到文件中。

初始向量:  $IV$

加密:

$$\begin{aligned}y_0 &= IV \\y_1 &= e_K(y_0 \oplus x_1) \\y_2 &= e_K(y_1 \oplus x_2) \\&\dots\end{aligned}$$

### 2.6.2 CBC 模式解密

解密过程即加密过程的逆过程, 但解密时需要将明文加密时填充的部分去除得到原始明文。

解密:

$$\begin{aligned}y_0 &= IV \\x_1 &= d_K(y_1) \oplus y_0 \\x_2 &= d_K(y_2) \oplus y_1 \\&\dots\end{aligned}$$







## 3 程序使用说明

### 3.1 输入输出格式

- ① 所有明文、密文、密钥、种子均采用文件输入输出方式。
- ② 用户需在控制台正确输入各步骤所对应的文件名
- ③ 用户需在控制台输入所得到的素数大小 (512bits/1024bits)
- ④ 文件名输入顺序: 依照程序友好提示, 顺序依次为:
  - I 储存有种子、密钥 1、密钥 2 的文件名
  - II Alice 将 RSA 公钥输出的文件名
  - III Bob 读入 Alice 产生的 RSA 公钥文件名
  - IV Bob 读入密钥 1、密钥 2 的文件名

- V Bob 将密文 c1 输出的文件名
- VI Bob 读入的存有明文 m 的文件名
- VII Bob 将明文 m 加密输出 c2 的文件名
- VIII Alice 读入密文 c1 的文件名
- IX Alice 读入密文 c2 的文件名
- X Alice 解密密文 c2 得到明文 m 输出的文件名

文件说明：

 Ciphertext_c1.txt	2021/6/29 10:22	文本文档	1 KB
 Ciphertext_c2.txt	2021/6/29 10:22	文本文档	1 KB
 Decryption.txt	2021/6/29 10:23	文本文档	1 KB
 INPUT_SEED_TWOKEY.txt	2021/6/28 21:13	文本文档	1 KB
 Plaintext.txt	2021/6/29 10:35	文本文档	1 KB
 RSA_Public_Key.txt	2021/6/29 10:21	文本文档	1 KB

INPUT\_SEED\_TWOKEY.txt：存放初始随机种子和两个密钥

RSA\_Public\_Key.txt：存放 Alice 得到的 RSA 公钥(n,b)

Ciphertext\_c1.txt：存放 Bob 通过 RSA 公钥加密临时会话密钥得到的密文 c1

Plaintext.txt：存放 Bob 需要加密的明文串 m

Ciphertext\_c2.txt：存放 Bob 通过 AES(CBC)加密得到的明文串 m 对应的密文 c2

Decryption.txt：存放 Alice 将密文串 c2 解密得到的明文串 m

## 4 执行实例

### 4.1 用户友好界面

目录：

 C:\Users\C.X\Desktop\Crypto\_Curri\_Design\Release\Crypto\_Curri\_Design.exe

```
《现代密码学课程设计》：1952650 陈子翔
*****
*                               *
*           项目模块           *
*       1 Alice生成RSA密钥       *
*       2 Bob加密文件m并发给Alice *
*       3 Alice解密恢复文件m     *
*                               *
*****
```

分模块执行提示：

```
*****
*           Alice生成RSA密钥模块：开始           *
*                               *
*****
```

Alice 产生 RSA 密钥模块：结束

```
*****
* Alice生成RSA密钥模块：结束 *
*****
```

Bob 加密明文 m 并发给 Alice 模块：开始

```
*****
* Bob加密文件m并发给Alice模块：开始 *
*****
```

Bob 加密明文 m 并发给 Alice 模块：结束

```
*****
* Bob加密文件m并发给Alice模块：结束 *
*****
```

Alice 解密恢复明文 m 模块：开始

```
*****
* Alice解密恢复明文m模块：开始 *
*****
```

Alice 解密恢复明文 m 模块：结束

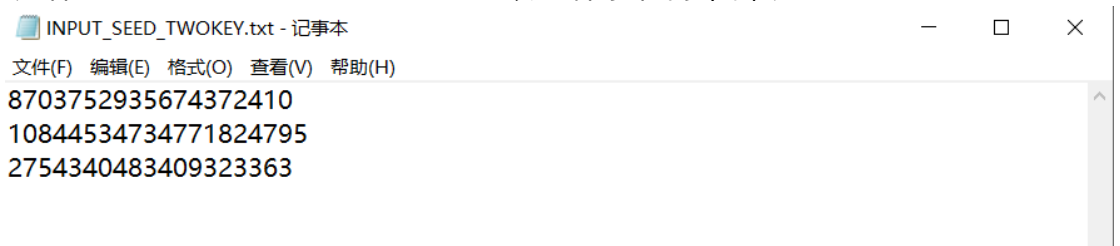
```
*****
* Alice解密恢复明文m模块：结束 *
*****
```

## 4.2 运行结果截图

Alice 生成 RSA 密钥模块：

```
*****
* Alice生成RSA密钥模块：开始 *
*****
Alice从文件读入种子、密钥1、密钥2：
请输入要打开的读入信息的的文件名： INPUT_SEED_TWOKEY.txt
Alice已从文件INPUT_SEED_TWOKEY.txt读入种子、密钥1、密钥2。
请输入要产生素数长度（512bits/1024bits）： 512
开始生成512bits的素数p, q
p = 1045517093243376683571476479219323199100521339797422391291621920882349429256228734269722889006474401884261167471806243750
q = 7894216171438713207869860424693312871639231915849332698676696820813501496211464694662258227642907484394516122813831504514
生成的p, q通过素性检测，用时 1.595s
Alice产生RSA密钥过程：
n = 8253537944997461211953873415710794949365832801514622589108168700948049036143632785822440994886351455757141068496816065833
1939401864958534644123776300638886510540512210011078624723100823539635257602342298749647325722442553251213721819856088565446
b = 9931
RSA公钥为(n, b)为： (8253537944997461211953873415710794949365832801514622589108168700948049036143632785822440994886351455757141
8259152929231411939401864958534644123776300638886510540512210011078624723100823539635257602342298749647325722442553251213721
Alice将RSA公钥输出到文件中传给Bob：
请输入要打开的写入信息的文件名： RSA_Public_Key.txt
已将RSA公钥输出到文件RSA_Public_Key.txt中。
* Alice生成RSA密钥模块：结束 *
*****
```

文件 INPUT\_SEED\_TWOKEY.txt： 读入种子和两个密钥



INPUT\_SEED\_TWOKEY.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

8703752935674372410  
10844534734771824795  
2754340483409323363



文件 RSA\_Public\_Key.txt: 保存 Alice 生成的 RSA 公钥(n,b)。

```
RSA_Public_Key.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
82535379449974612119538734157107949493658328015146225891081687009480490:
9931
```

Bob 加密明文 m 并发送给 Alice 模块:

```
*****
*      Bob加密文件m并发送给Alice模块: 开始      *
*****
Bob从文件中读入RSA公钥(n, b):
请输入要打开的读入信息的的文件名: RSA_Public_Key.txt
Bob已从文件RSA_Public_Key.txt中读入RSA公钥(n, b)。
n_Bob = 825353794499746121195387341571079494936583280151462258910816870094804903614363278582244099488635145575714106849681606
31411939401864958534644123776300638886510540512210011078624723100823539635257602342298749647325722244255325121372181985608856
b_Bob = 9931
Bob从文件读入种子、密钥1、密钥2:
请输入要打开的读入信息的的文件名: INPUT_SEED_TWOKEY.txt
Alice已从文件INPUT_SEED_TWOKEY.txt读入种子、密钥1、密钥2。
Bob生成AES临时会话密钥k_Bob:
Bob已经生成AES临时会话密钥k_Bob = 256461853223350523160680535662127183581
Bob通过RSA公钥加密密钥k_Bob:
Bob已通过RSA公钥加密密钥k_Bob得密文c1_Bob = 520576727025368543698956561673089525225534205543915842286174925525546373851559821
41651850175741131363061232101490488085747542553663824185076815144849837723009751242605848808467346218320972008798990224455239
535203839987
Bob将密文c1_Bob输出到文件中:
请输入要打开的写入信息的的文件名: Ciphertext_c1.txt
Bob已将密文c1_Bob输出到文件Ciphertext_c1.txt中。
Bob从文件Plaintext.txt中读入明文m并对其利用AES (CBC) 模式进行加密:
请输入要打开的读入信息的的文件名: Plaintext.txt
请输入要打开的写入信息的的文件名: Ciphertext_c2.txt
Bob已将密文c2_Bob输出到文件Ciphertext_c2.txt中。
*****
*      Bob加密文件m并发送给Alice模块: 结束      *
*****
```

文件 Ciphertext\_c1.txt: 存放 Bob 通过 RSA 公钥加密的会话密钥 k 的密文 c1

```
Ciphertext_c1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
52057672702536854369895656167308952522553420554391584228617492552554637:
```

文件 Plaintext.txt: 存放 Bob 需要加密的明文串 m

```
Plaintext.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
123456789123456789123456789
```

文件 Ciphertext\_c2.txt: 存放 Bob 通过 AES (CBC) 加密明文得到的密文 c2

```
Ciphertext_c2.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
誤?Z菟蚌孳UW扞灝u  □?囍H□艮^[
```

Alice 解密恢复明文 m 模块:

```
*****
*      Alice解密恢复明文m模块: 开始      *
*****
Alice接收 (c1,c2) 并解密得明文过程: 开始
Alice从文件中读入密文c1 Alice:
请输入要打开的读入信息的的文件名: Ciphertext_c1.txt
Alice已从文件Ciphertext_c1.txt中读入密文c1 Alice。
Alice用RSA私钥a解密c1_Alice得到AES临时会话密钥k_Alice:
Alice已c1_Alice得到AES临时会话密钥k_Alice = 256461853223350523160680535662127183581
Alice使用AES密钥k_Alice解密密文c2_Alice得到明文m_Alice:
请输入要打开的读入信息的的文件名: Ciphertext_c2.txt
请输入要打开的写入信息的文件名: Decryption.txt
*****
*      Alice解密恢复明文m模块: 结束      *
*****
```

文件: Decryption.txt: 存放 Alice 解密密文 c2 得到的明文串 m



经与初始明文串比对, Alice 解密后得到的明文与初始明文一致, 故可以成功解密, 程序正确。

## 5 程序源代码

Crypto\_Curri\_Design.h

```
#pragma once
#include <iostream>
#include <time.h>
#include "AES.h"
#include "DES.h"
#include <fstream>
#include <NTL/ZZ.h>
```

```
using namespace std;
```

```
using namespace NTL;
```



```

typedef unsigned long long ULL;

void CBC_Encrypt(AES& aes, ZZ& IV, ifstream& fin, ofstream& fout);

void CBC_Decrypt(ZZ& IV, ifstream& fin, ofstream& fout, ZZ k_de);

//生成加密指数b
ZZ b_Generation(ZZ phi_n);

//生成解密指数a:  $\phi_n * x + b * y = 1$ , b:加密指数, y:解密指数a
void Extend_Euclidean(ZZ a, ZZ b, ZZ& x, ZZ& y);

//RSA加密明文x, 得到密文y
ZZ RSA_Encrypt(ZZ x, ZZ b, ZZ n);

//RSA解密密文y, 得到明文x
ZZ RSA_Decrypt(ZZ y, ZZ a, ZZ n);

//素性检测算法
bool MillerRabin(const ZZ& n, long t);

//生成随机素数
void Prime_Generation(ZZ& p, ZZ& q, int length, ULL Seed, ULL Key1, ULL Key2);

//ANSI随机数生成算法
ZZ ANSI(ULL s, int m, ULL k1, ULL k2);

```

## DES.h

```

#pragma once
#include <NTL/ZZ.h>

using namespace NTL;

typedef unsigned long long ULL;

class DES {
private:
    //密钥编排规则
    const int Key_Schedule[16] = {
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
    };
    //S盒

```

```

const int S_BOX[32][16] = {
14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7, //S1
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,
15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10, //S2
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,
10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8, //S3
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,
7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15, //S4
13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,
2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9, //S5
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,
12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11, //S6
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,
4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1, //S7
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,
13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7, //S8
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11
};

//初始IP置换
const int IP_First[64] = {
58, 50, 42, 34, 26, 18, 10, 2,
60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6,
64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3,

```

```

61, 53, 45, 37, 29, 21, 13, 5,
63, 55, 47, 39, 31, 23, 15, 7
};

//未置换IP-1
const int IP_Last[64] = {
40, 8, 48, 16, 56, 24, 64, 32,
39, 7, 47, 15, 55, 23, 63, 31,
38, 6, 46, 14, 54, 22, 62, 30,
37, 5, 45, 13, 53, 21, 61, 29,
36, 4, 44, 12, 52, 20, 60, 28,
35, 3, 43, 11, 51, 19, 59, 27,
34, 2, 42, 10, 50, 18, 58, 26,
33, 1, 41, 9, 49, 17, 57, 25
};

//F函数中的扩展置换E
const int E[48] = {
32, 1, 2, 3, 4, 5,
4, 5, 6, 7, 8, 9,
8, 9, 10, 11, 12, 13,
12, 13, 14, 15, 16, 17,
16, 17, 18, 19, 20, 21,
20, 21, 22, 23, 24, 25,
24, 25, 26, 27, 28, 29,
28, 29, 30, 31, 32, 1
};

//F函数中的置换P
const int P[32] = {
16, 7, 20, 21,
29, 12, 28, 17,
1, 15, 23, 26,
5, 18, 31, 10,
2, 8, 24, 14,
32, 27, 3, 9,
19, 13, 30, 6,
22, 11, 4, 25,
};

//密钥扩展中的置换PC-1
const int Key_PC1[56] = {
57, 49, 41, 33, 25, 17, 9,
1, 58, 50, 42, 34, 26, 18,
10, 2, 59, 51, 43, 35, 27,
19, 11, 3, 60, 52, 44, 36,

```

```

//above for Ci, below for Di
63, 55, 47, 39, 31, 23, 15,
7, 62, 54, 46, 38, 30, 22,
14, 6, 61, 53, 45, 37, 29,
21, 13, 5, 28, 20, 12, 4
};
//密钥扩展中的压缩置换PC-2
const int Compress_PC2[48] = {
14, 17, 11, 24, 1, 5,
3, 28, 15, 6, 21, 10,
23, 19, 12, 4, 26, 8,
16, 7, 27, 20, 13, 2,
41, 52, 31, 37, 47, 55,
30, 40, 51, 45, 33, 48,
44, 49, 39, 56, 34, 53,
46, 42, 50, 36, 29, 32
};
ULL Secret_Key;
ULL Encryp_Info;
ULL Decrypt_Info;
ULL Plain_Text;
public:
ULL Input_Key[16];
//构造函数, 初始化密钥, 明文
DES();
void Set_Key(ULL k);
void Set_Plain_Text(ULL p);
void Set_Encryp_Info(ULL c);

ULL Get_Secret_Key();

ULL Get_Encryp_Info();

ULL Get_Decrypt_Info();

ULL Get_Plain_Text();

ULL Bit_Extension(ULL num, const int p[], int Max, int length);

ULL Transform(ULL key, int n);

void Generate_Key();

```

```

    ULL S_Boxes_Transform(ULL num);

    void Encrypt();

    void Decrypt();
};

ULL DES_Encrypt(ULL Plain_Text, ULL Secret_Key);

ULL DES_Decrypt(ULL Encryp_Info, ULL Secret_Key);

```

## AES.h

```

#pragma once
#include <NTL/ZZ.h>

using namespace NTL;

typedef unsigned long long ULL;

class AES {
private:
    const unsigned char S_Box[16][16] = {
        {0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE,
        0xD7, 0xAB, 0x76},
        {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C,
        0xA4, 0x72, 0xC0},
        {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71,
        0xD8, 0x31, 0x15},
        {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB,
        0x27, 0xB2, 0x75},
        {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29,
        0xE3, 0x2F, 0x84},
        {0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A,
        0x4C, 0x58, 0xCF},
        {0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50,
        0x3C, 0x9F, 0xA8},
        {0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10,
        0xFF, 0xF3, 0xD2},
        {0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64,
        0x5D, 0x19, 0x73},
        {0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE,

```

```

0x5E, 0x0B, 0xDB},
    {0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91,
0x95, 0xE4, 0x79},
    {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65,
0x7A, 0xAE, 0x08},
    {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B,
0xBD, 0x8B, 0x8A},
    {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86,
0xC1, 0x1D, 0x9E},
    {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE,
0x55, 0x28, 0xDF},
    {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0,
0x54, 0xBB, 0x16}
};

const unsigned char INV_S_Box[16][16] = {
    {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81,
0xf3, 0xd7, 0xfb},
    {0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4,
0xde, 0xe9, 0xcb},
    {0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42,
0xfa, 0xc3, 0x4e},
    {0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d,
0x8b, 0xd1, 0x25},
    {0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d,
0x65, 0xb6, 0x92},
    {0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7,
0x8d, 0x9d, 0x84},
    {0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8,
0xb3, 0x45, 0x06},
    {0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01,
0x13, 0x8a, 0x6b},
    {0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0,
0xb4, 0xe6, 0x73},
    {0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c,
0x75, 0xdf, 0x6e},
    {0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa,
0x18, 0xbe, 0x1b},
    {0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78,
0xcd, 0x5a, 0xf4},
    {0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27,
0x80, 0xec, 0x5f},
    {0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93,

```

```

0xc9, 0x9c, 0xef},
    {0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83,
0x53, 0x99, 0x61},
    {0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55,
0x21, 0x0c, 0x7d}
};

const unsigned int Rcon[11] = {
    0,
    0x00000001U, // 0x01000000U,
    0x00000002U, // 0x02000000U,
    0x00000004U, // 0x04000000U,
    0x00000008U, // 0x08000000U,
    0x00000010U, // 0x10000000U,
    0x00000020U, // 0x20000000U,
    0x00000040U, // 0x40000000U,
    0x00000080U, // 0x80000000U,
    0x0000001BU, // 0x1B000000U,
    0x00000036U // 0x36000000U
};

unsigned char State[16];
unsigned char Round_Key[11][16]; //轮密钥
unsigned char state[4][4]; //没用
public:
    AES(const ZZ& key);
    void Encrypt(ZZ& ciphertext, const ZZ& plaintext);
    void Decrypt(ZZ& plaintext, const ZZ& ciphertext);
    void tranRowCol(unsigned char* buf);
    void Transpose(unsigned char* State, int len);
    void SubBytes();
    void Inv_SubBytes();
    void ShiftRows();
    void Inv_ShiftRows();
    unsigned char GFfield_2_8_Mult(unsigned char x);
    unsigned char Ffmul(unsigned char a, unsigned char b);
    void MixColumns();
    void Inv_MixColumns();
    void AddRoundKey(const unsigned char* key);
    ULL SubWord(ULL in);
    ULL RotWord(ULL in);
};

```

Crypto\_Curri\_Design.cpp

```

#include "Crypto_Curri_Design.h"
#include <windows.h>

//获取系统的当前时间，单位微秒(us)
ULL GetSysTimeMicros()
{
#ifdef _WIN32
    // 从1601年1月1日0:0:0:000到1970年1月1日0:0:0:000的时间(单位100ns)
#define EPOCHFILETIME (1164447360000000000ULL)
    FILETIME ft;
    LARGE_INTEGER li;
    ULL tt = 0;
    GetSystemTimeAsFileTime(&ft);
    li.LowPart = ft.dwLowDateTime;
    li.HighPart = ft.dwHighDateTime;
    // 从1970年1月1日0:0:0:000到现在的微秒数(UTC时间)
    tt = (li.QuadPart - EPOCHFILETIME) / 10;
    return tt;
#else
    timeval tv;
    gettimeofday(&tv, 0);
    return (int64_t)tv.tv_sec * 1000000 + (int64_t)tv.tv_usec;
#endif // _WIN32
    return 0;
}

//ZZ变量转为ULL变量
ULL ZZ_To_Uint64(ZZ a)
{
    ULL Temp = 0;
    for (int j = 0; j < 64; j++)
        if (bit(a, j) == 1)
            Temp += pow(2, j);
    return Temp;
}

//ULL变量转为ZZ变量
void Uint64_To_ZZ(ZZ& x, ULL n)
{
    const unsigned char* Temp = (const unsigned char*)&n;
    ZZFromBytes(x, Temp, 8);
}

```



```

//ANSI随机数生成算法
ZZ ANSI(ULL s, int m, ULL k1, ULL k2)
{
    ZZ Result;
    Result = to_ZZ(0);
    ULL TempI, Xi;
    ULL D = GetSysTimeMicros();
    ZZ TempX;
    TempI = DES_Encrypt(DES_Decrypt(DES_Encrypt(D, k1), k2), k1);
    for (int i = 0; i < m; i++)
    {
        Xi = DES_Encrypt(DES_Decrypt(DES_Encrypt(TempI ^ s, k1), k2), k1);
        s = DES_Encrypt(DES_Decrypt(DES_Encrypt(Xi ^ TempI, k1), k2), k1);
        Uint64_To_ZZ(TempX, Xi);
        Result <<= 64;
        Result = Result + TempX;
    }
    return Result;
}

void Open_Input_File(ifstream& infile)
{
    char* p = new char[30];
    cout << "请输入要打开的读入信息的文件名: ";
    cin >> p;
    infile.open(p, ios::in);
    if (infile.is_open() == 0)
    {
        cout << "打开文件失败" << endl;
        exit(-1);
    }
}

void Open_Output_File(ofstream& fout)
{
    char* p = new char[30];
    cout << "请输入要打开的写入信息的文件名: ";
    cin >> p;
    fout.open(p, ios::out);
    if (fout.is_open() == 0)
    {

```

```

        cout << "打开文件失败" << endl;
        exit(-1);
    }
}

void Friendly_Menu(int select)
{
    if (select == 0)
    {
        int i;
        cout << "《现代密码学课程设计》：1952650 陈子翔" << endl;
        for (i = 0; i < 50; i++)
        {
            cout << '*';
        }
        cout << endl;
        cout << "*\t\t 项目模块\t\t\t *\n";
        cout << "*\t\t1 Alice生成RSA密钥\t\t *\n";
        cout << "*\t\t2 Bob加密明文m并发给Alice\t *\n";
        cout << "*\t\t3 Alice解密恢复明文m\t\t *\n";
        for (i = 0; i < 50; i++)
        {
            cout << '*';
        }
        cout << endl << endl;
    }
    if (select == 1)
    {
        int i;
        for (i = 0; i < 50; i++)
        {
            cout << '*';
        }
        cout << endl;
        cout << "*\tAlice生成RSA密钥模块：开始\t\t *\n";
        for (i = 0; i < 50; i++)
        {
            cout << '*';
        }
        cout << endl;
    }
    if (select == 2)

```

```

{
    int i;
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
    cout << endl;
    cout << " *\tAlice生成RSA密钥模块：结束\t\t *\n";
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
    cout << endl << endl;
}
if (select == 3)
{
    int i;
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
    cout << endl;
    cout << " *\tBob加密文件m并发给Alice模块：开始\t *\n";
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
    cout << endl;
}
if (select == 4)
{
    int i;
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
    cout << endl;
    cout << " *\tBob加密文件m并发给Alice模块：结束\t *\n";
    for (i = 0; i < 50; i++)
    {
        cout << '*' ;
    }
}

```

```

        cout << endl << endl;
    }
}

int main()
{
    Friendly_Menu(0);
    Friendly_Menu(1);
    cout << "Alice从文件读入种子、密钥1、密钥2: " << endl;
    ULL seed_Alice(0), key1_Alice(0), key2_Alice(0);
    ZZ p, q;
    ifstream infile;
    Open_Input_File(infile);
    infile >> seed_Alice >> key1_Alice >> key2_Alice;
    cout << "Alice已从文件INPUT_SEED_TWOKEY.txt读入种子、密钥1、密钥2。" << endl;
    infile.close();
    int len;
    cout << "请输入要产生素数长度 (512bits/1024bits): ";
    cin >> len;
    Prime_Generation(p, q, len, seed_Alice, key1_Alice, key2_Alice);
    cout << "Alice产生RSA密钥过程: " << endl;
    ZZ n, phi_n, b;
    n = p * q;
    cout << "n = " << n << endl;
    phi_n = p * q - p - q + 1;
    b = b_Generation(phi_n);
    cout << "b = " << b << endl;
    cout << "RSA公钥为(n,b)为: (" << n << ", " << b << ")" << endl;
    cout << "Alice将RSA公钥输出到文件中传给Bob: " << endl;
    ofstream fout;
    Open_Output_File(fout);
    fout << n << endl << b << endl;
    cout << "已将RSA公钥输出到文件RSA_Public_Key.txt中。" << endl;
    fout.close();
    Friendly_Menu(2);

    Friendly_Menu(3);
    ZZ n_Bob, b_Bob;
    cout << "Bob从文件中读入RSA公钥(n,b): " << endl;
    Open_Input_File(infile);
    infile >> n_Bob >> b_Bob;

```

```

cout << "Bob已从文件RSA_Public_Key.txt中读入RSA公钥(n,b)。" << endl;
infile.close();
cout << "n_Bob = " << n_Bob << endl;
cout << "b_Bob = " << b_Bob << endl;
cout << "Bob从文件读入种子、密钥1、密钥2：" << endl;
Open_Input_File(infile);
ZZ k_Bob;
ULL seed_Bob, Key1_Bob, Key2_Bob;
infile >> seed_Bob >> Key1_Bob >> Key2_Bob;
cout << "Alice已从文件INPUT_SEED_TWOKEY.txt读入种子、密钥1、密钥2。" << endl;
infile.close();
cout << "Bob生成AES临时会话密钥k_Bob：" << endl;
k_Bob = ANSI(seed_Bob, 2, Key1_Bob, Key2_Bob);
cout << "Bob已经生成AES临时会话密钥k_Bob = " << k_Bob << endl;
cout << "Bob通过RSA公钥加密密钥k_Bob：" << endl;
ZZ c1_Bob;
AES aes_Bob(k_Bob);
c1_Bob = RSA_Encrypt(k_Bob, b_Bob, n_Bob);
cout << "Bob已通过RSA公钥加密密钥k_Bob得密文c1_Bob = " << c1_Bob << endl;
cout << "Bob将密文c1_Bob输出到文件中：" << endl;
Open_Output_File(fout);
fout << c1_Bob << endl;
cout << "Bob已将密文c1_Bob输出到文件Ciphertext_c1.txt中。" << endl;
fout.close();
ZZ m, IV;
cout << "Bob从文件Plaintext.txt中读入明文m并对其利用AES（CBC）模式进行加密：" <<
endl;
Open_Input_File(infile);
Open_Output_File(fout);
CBC_Encrypt(aes_Bob, IV, infile, fout);
cout << "Bob已将密文c2_Bob输出到文件Ciphertext_c2.txt中。" << endl;
infile.close();
fout.close();
Friendly_Menu(4);

for (int i = 0; i < 50; i++)
    cout << '*';
cout << endl;
cout << "*\tAlice解密恢复明文m模块：开始\t\t*\n";
for (int i = 0; i < 50; i++)
    cout << '*';

```

```

cout << endl;
cout << "Alice接收 (c1,c2) 并解密得明文过程：开始" << endl;
ZZ k_Alice, c1_Alice;
ZZ a, x;
Extend_Euclidean(phi_n, b, x, a);
cout << "Alice从文件中读入密文c1_Alice：" << endl;
Open_Input_File(infile);
infile >> c1_Alice;
cout << "Alice已从文件Ciphertext_c1.txt中读入密文c1_Alice。" << endl;
infile.close();
cout << "Alice用RSA私钥a解密c1_Alice得到AES临时会话密钥k_Alice：" << endl;
k_Alice = RSA_Decrypt(c1_Alice, a, n);
cout << "Alice已c1_Alice得到AES临时会话密钥k_Alice = " << k_Alice << endl;
cout << "Alice使用AES密钥k_Alice解密密文c2_Alice得到明文m_Alice：" << endl;
Open_Input_File(infile);
Open_Output_File(fout);
CBC_Decrypt(IV, infile, fout, k_Alice);
for (int i = 0; i < 50; i++)
    cout << '*';
cout << endl;
cout << "*\tAlice解密恢复明文m模块：结束\t\t*\n";
for (int i = 0; i < 50; i++)
    cout << '*';
cout << endl << endl;

return 0;
}

```

## DES.cpp

```

#include <NTL/ZZ.h>
#include "DES.h"

using namespace NTL;

DES::DES() {
    Secret_Key = 0;
    Encryp_Info = 0;
    Decrypt_Info = 0;
    Plain_Text = 0;
    for (int i = 0; i < 16; i++)
        Input_Key[i] = 0;
}

```

```

void DES::Set_Key(ULL k)
{
    Secret_Key = k;
}

void DES::Set_Plain_Text(ULL p)
{
    Plain_Text = p;
}

void DES::Set_Encryp_Info(ULL c)
{
    Encryp_Info = c;
}

ULL DES::Get_Secret_Key()
{
    return Secret_Key;
}

ULL DES::Get_Encryp_Info()
{
    return Encryp_Info;
}

ULL DES::Get_Decrypt_Info()
{
    return Decrypt_Info;
}

ULL DES::Get_Plain_Text()
{
    return Plain_Text;
}

ULL DES::Bit_Extension(ULL num, const int p[], int Max, int length)
{
    ULL Result = 0;
    int i;
    for (i = 0; i < length; i++)
    {
        Result <<= 1;
        Result |= (num >> (Max - p[i])) & 1;
    }
}

```

```

    }
    return Result;
}

ULL DES::Transform(ULL key, int n)
{
    ULL Result = 0;
    ULL Left, Right;
    Left = (key & 0xFFFFFFFF00000000LL) >> 28;
    Right = key & 0x00000000FFFFFFFF;
    if (n == 1) {
        Left = ((Left & 0x7FFFFFFF) << 1) | ((Left >> 27) & 1);
        Right = ((Right & 0x7FFFFFFF) << 1) | ((Right >> 27) & 1);
        Result = (Left << 28) | Right;
    }
    else if (n == 2) {
        Left = ((Left & 0x3FFFFFFF) << 2) | ((Left >> 26) & 3);
        Right = ((Right & 0x7FFFFFFF) << 2) | ((Right >> 26) & 3);
        Result = (Left << 28) | Right;
    }
    return Result;
}

void DES::Generate_Key()
{
    ULL generate_key;
    generate_key = Bit_Extension(Secret_Key, Key_PC1, 64, 56);
    for (int i = 0; i < 16; i++)
    {
        generate_key = Transform(generate_key, Key_Schedule[i]);
        Input_Key[i] = Bit_Extension(generate_key, Compress_PC2, 56, 48);
    }
}

ULL DES::S_Boxes_Transform(ULL num)
{
    ULL temp, result = 0;
    for (int i = 0; i < 8; i++)
    {
        temp = (num >> ((7 - i) * 6)) & 0x3f;
        int x = ((temp >> 4) & 0x2) | (temp & 0x1) + i * 4;
        int y = (temp >> 1) & 0xf;
    }
}

```



```

        temp = S_BOX[x][y];
        temp = temp << ((7 - i) * 4);
        result |= temp;
    }
    return result;
}

void DES::Encrypt()
{
    ULL L, R, temp_r, temp;

    temp = Bit_Extension(Plain_Text, IP_First, 64, 64);
    L = (temp & 0xFFFFFFFF00000000LL) >> 32;
    R = (temp & 0x00000000FFFFFFFFLL);
    temp_r = R;

    for (int i = 0; i < 16; i++)
    {
        temp_r = Bit_Extension(R, E, 32, 48);
        temp_r = temp_r ^ Input_Key[i];
        temp_r = S_Boxes_Transform(temp_r);
        temp_r = Bit_Extension(temp_r, P, 32, 32);
        temp_r ^= L;
        L = R;
        R = temp_r;
    }
    temp = (R << 32) | L;
    temp = Bit_Extension(temp, IP_Last, 64, 64);
    Encryp_Info = temp;
}

void DES::Decrypt()
{
    ULL L, R, temp_r, temp;

    temp = Bit_Extension(Encryp_Info, IP_First, 64, 64);
    L = (temp & 0xffffffff00000000LL) >> 32;
    R = (temp & 0x00000000ffffffffLL);
    temp_r = R;

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

```

```

        temp_r = Bit_Extension(R, E, 32, 48);
        temp_r = temp_r ^ Input_Key[15 - i];
        temp_r = S_Boxes_Transform(temp_r);
        temp_r = Bit_Extension(temp_r, P, 32, 32);
        temp_r ^ = L;
        L = R;
        R = temp_r;
    }
    temp = (R << 32) | L;
    temp = Bit_Extension(temp, IP_Last, 64, 64);
    Decrypt_Info = temp;
}

```

//DES加密过程

```

ULL DES_Encrypt(ULL Plain_Text, ULL Secret_Key)
{
    DES des;
    des.Set_Plain_Text(Plain_Text);
    des.Set_Key(Secret_Key);
    des.Generate_Key();
    des.Encrypt();
    return des.Get_Encryp_Info();
}

```

//DES解密过程

```

ULL DES_Decrypt(ULL Encryp_Info, ULL Secret_Key)
{
    DES des;
    des.Set_Encryp_Info(Encryp_Info);
    des.Set_Key(Secret_Key);
    des.Generate_Key();
    des.Decrypt();
    return des.Get_Decrypt_Info();
}

```

## AES.cpp

```

#include <NTL/ZZ.h>
#include "AES.h"

using namespace NTL;

AES::AES(const ZZ& key)

```

```

{
    unsigned int temp;

    unsigned int* w = (unsigned int*)Round_Key;
    BytesFromZZ((unsigned char*)w, key, 16);
    Transpose((unsigned char*)w, 16);

    for (int i = 4; i < 44; ++i)
    {
        temp = w[i - 1];
        // 被4整除: 低2位为0
        if (!(i & 0x3))
            temp = SubWord(RotWord(temp)) ^ Rcon[i / 4];

        w[i] = w[i - 4] ^ temp;
    }
}

void AES::Encrypt(ZZ& ciphertext, const ZZ& plaintext)
{
    BytesFromZZ(State, plaintext, 16);
    // 小字序转换成正序
    Transpose(State, 16);
    // 行列转换
    tranRowCol(State);

    AddRoundKey(Round_Key[0]);

    for (int i = 1; i <= 9; i++) {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(Round_Key[i]);
    }

    SubBytes();

    ShiftRows();

    AddRoundKey(Round_Key[10]);

    // 正序转回小字序

```

```

Transpose(State, 16);
// 行列转换
tranRowCol(State);
ZZFromBytes(ciphertext, State, 16);
}

void AES::Decrypt(ZZ& plaintext, const ZZ& ciphertext)
{
    BytesFromZZ(State, ciphertext, 16);
    // 小字序转换成正序
    Transpose(State, 16);
    // 行列转换
    tranRowCol(State);

    AddRoundKey(Round_Key[10]);

    for (int round = 1; round <= 9; ++round) {
        Inv_ShiftRows();
        Inv_SubBytes();
        AddRoundKey(Round_Key[10 - round]);
        Inv_MixColumns();
    }

    Inv_ShiftRows();
    Inv_SubBytes();
    AddRoundKey(Round_Key[0]);

    // 正序转回小字序
    Transpose(State, 16);
    // 行列转换
    tranRowCol(State);
    ZZFromBytes(plaintext, State, 16);
}

void AES::tranRowCol(unsigned char* buf)
{
    unsigned char(*p)[4] = (unsigned char(*)[4])buf;
    unsigned char temp;
    for (int i = 0; i < 4; ++i)
        for (int j = i + 1; j < 4; ++j)
        {
            temp = p[i][j];

```

```

        p[i][j] = p[j][i];
        p[j][i] = temp;
    }
}

void AES::Transpose(unsigned char* State, int len)
{
    int t = len / 2;
    unsigned char temp;
    for (int i = 0; i < t; ++i)
    {
        temp = State[i];
        State[i] = State[len - i - 1];
        State[len - i - 1] = temp;
    }
}

void AES::SubBytes()
{
    for (int i = 0; i < 16; i++)
        State[i] = S_Box[(State[i] & 0xF0U) >> 4][State[i] & 0x0FU];
    //没用
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            state[i][j] = S_Box[i][j];
}

void AES::Inv_SubBytes()
{
    for (int i = 0; i < 16; i++)
        State[i] = INV_S_Box[(State[i] & 0xF0U) >> 4][State[i] & 0x0FU];
    //没用
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            state[i][j] = S_Box[i][j];
}

void AES::ShiftRows()
{
    unsigned char Mid;

    Mid = State[4];

```

```

    for (int i = 4; i < 7; ++i)
        State[i] = State[i + 1];
    State[7] = Mid;

    //交换State[8]和State[10]
    Mid = State[8];
    State[8] = State[10];
    State[10] = Mid;
    //交换State[9]和State[11]
    Mid = State[9];
    State[9] = State[11];
    State[11] = Mid;

    Mid = State[15];
    for (int i = 15; i > 12; --i)
        State[i] = State[i - 1];
    State[12] = Mid;

    //以下没用
    unsigned char t[4];
    int r, c;
    for (r = 1; r < 4; r++)
    {
        for (c = 0; c < 4; c++)
        {
            t[c] = state[r][(c + r) % 4];
        }
        for (c = 0; c < 4; c++)
        {
            state[r][c] = t[c];
        }
    }
}

void AES::Inv_ShiftRows()
{
    unsigned char Mid;

    Mid = State[7];
    for (int i = 7; i > 4; --i)
        State[i] = State[i - 1];
    State[4] = Mid;

```

```

//交换State[8]和State[10]
Mid = State[8];
State[8] = State[10];
State[10] = Mid;
//交换State[9]和State[11]
Mid = State[9];
State[9] = State[11];
State[11] = Mid;

Mid = State[12];
for (int i = 12; i < 15; ++i)
    State[i] = State[i + 1];
State[15] = Mid;

//以下没用
unsigned char t[4];
int r, c;
for (r = 1; r < 4; r++)
{
    for (c = 0; c < 4; c++)
    {
        t[c] = state[r][(c - r + 4) % 4];
    }
    for (c = 0; c < 4; c++)
    {
        state[r][c] = t[c];
    }
}
}

unsigned char AES::GField_2_8_Mult(unsigned char x)
{
    if ((x & '\x80') != 0)
        return (x << 1) ^ '\x1b';
    else
        return x << 1;
}

unsigned char AES::FFmul(unsigned char a, unsigned char b)
{
    unsigned char res = 0;

```

```

    unsigned char x[4];
    int i;
    x[0] = b;
    for (i = 1; i < 4; i++)
    {
        x[i] = x[i - 1] << 1;
        if (x[i - 1] & 0x80)
        {
            x[i] ^= 0x1b;
        }
    }
    for (i = 0; i < 4; i++)
    {
        if ((a >> i) & 0x01)
        {
            res ^= x[i];
        }
    }
    return res;
}

void AES::MixColumns()
{
    unsigned char Inter_Variable[4];
    unsigned char(*TDPointer)[4] = (unsigned char(*)[4])State;

    for (int i = 0; i < 4; i++)
    {
        Inter_Variable[0] = TDPointer[0][i];
        Inter_Variable[1] = TDPointer[1][i];
        Inter_Variable[2] = TDPointer[2][i];
        Inter_Variable[3] = TDPointer[3][i];

        TDPointer[0][i] = Inter_Variable[1] ^ Inter_Variable[2] ^ Inter_Variable[3];
        TDPointer[1][i] = Inter_Variable[0] ^ Inter_Variable[2] ^ Inter_Variable[3];
        TDPointer[2][i] = Inter_Variable[0] ^ Inter_Variable[1] ^ Inter_Variable[3];
        TDPointer[3][i] = Inter_Variable[0] ^ Inter_Variable[1] ^ Inter_Variable[2];

        for (int j = 0; j < 4; j++)
            Inter_Variable[j] = GField_2_8_Mult(Inter_Variable[j]);

        TDPointer[0][i] = TDPointer[0][i] ^ Inter_Variable[0] ^ Inter_Variable[1];
    }
}

```



```

        TDPinter[1][i] = TDPinter[1][i] ^ Inter_Variable[1] ^ Inter_Variable[2];
        TDPinter[2][i] = TDPinter[2][i] ^ Inter_Variable[2] ^ Inter_Variable[3];
        TDPinter[3][i] = TDPinter[3][i] ^ Inter_Variable[3] ^ Inter_Variable[0];
    }

    //以下没用
    unsigned char t[4];
    int r, c;
    for (c = 0; c < 4; c++)
    {
        for (r = 0; r < 4; r++)
        {
            t[r] = state[r][c];
        }
        for (r = 0; r < 4; r++)
        {
            state[r][c] = FFmul(0x02, t[r])
                ^ FFmul(0x03, t[(r + 1) % 4])
                ^ FFmul(0x01, t[(r + 2) % 4])
                ^ FFmul(0x01, t[(r + 3) % 4]);
        }
    }
}

void AES::Inv_MixColumns()
{
    unsigned char Inter_Variable[4];
    unsigned char(*TDPinter)[4] = (unsigned char(*)[4])State;

    for (int i = 0; i < 4; ++i)
    {
        //将State中的值拷贝到中间变量Inter_Variable中
        Inter_Variable[0] = TDPinter[0][i];
        Inter_Variable[1] = TDPinter[1][i];
        Inter_Variable[2] = TDPinter[2][i];
        Inter_Variable[3] = TDPinter[3][i];

        TDPinter[0][i] = Inter_Variable[1] ^ Inter_Variable[2] ^ Inter_Variable[3];
        TDPinter[1][i] = Inter_Variable[0] ^ Inter_Variable[2] ^ Inter_Variable[3];
        TDPinter[2][i] = Inter_Variable[0] ^ Inter_Variable[1] ^ Inter_Variable[3];
        TDPinter[3][i] = Inter_Variable[0] ^ Inter_Variable[1] ^ Inter_Variable[2];
    }
}

```

```

for (int j = 0; j < 4; j++)
    Inter_Variable[j] = GField_2_8_Mult(Inter_Variable[j]);

TDPinter[0][i] = TDPinter[0][i] ^ Inter_Variable[0] ^ Inter_Variable[1];
TDPinter[1][i] = TDPinter[1][i] ^ Inter_Variable[1] ^ Inter_Variable[2];
TDPinter[2][i] = TDPinter[2][i] ^ Inter_Variable[2] ^ Inter_Variable[3];
TDPinter[3][i] = TDPinter[3][i] ^ Inter_Variable[3] ^ Inter_Variable[0];

Inter_Variable[0] = GField_2_8_Mult(Inter_Variable[0] ^ Inter_Variable[2]);
Inter_Variable[1] = GField_2_8_Mult(Inter_Variable[1] ^ Inter_Variable[3]);

TDPinter[0][i] ^= Inter_Variable[0];
TDPinter[1][i] ^= Inter_Variable[1];
TDPinter[2][i] ^= Inter_Variable[0];
TDPinter[3][i] ^= Inter_Variable[1];

Inter_Variable[0] = GField_2_8_Mult(Inter_Variable[0] ^ Inter_Variable[1]);

TDPinter[0][i] ^= Inter_Variable[0];
TDPinter[1][i] ^= Inter_Variable[0];
TDPinter[2][i] ^= Inter_Variable[0];
TDPinter[3][i] ^= Inter_Variable[0];
}

//以下没用
unsigned char t[4];
int r, c;
for (c = 0; c < 4; c++)
{
    for (r = 0; r < 4; r++)
    {
        t[r] = state[r][c];
    }
    for (r = 0; r < 4; r++)
    {
        state[r][c] = FFmul(0x0e, t[r])
            ^ FFmul(0x0b, t[(r + 1) % 4])
            ^ FFmul(0x0d, t[(r + 2) % 4])
            ^ FFmul(0x09, t[(r + 3) % 4]);
    }
}
};

```

```

void AES::AddRoundKey(const unsigned char* key)
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            State[i * 4 + j] ^= key[j * 4 + i];
}

```

```

ULL AES::SubWord(ULL in)
{
    unsigned char* p = (unsigned char*)&in;
    p[0] = S_Box[(p[0] & 0xF0U) >> 4][p[0] & 0x0FU];
    p[1] = S_Box[(p[1] & 0xF0U) >> 4][p[1] & 0x0FU];
    p[2] = S_Box[(p[2] & 0xF0U) >> 4][p[2] & 0x0FU];
    p[3] = S_Box[(p[3] & 0xF0U) >> 4][p[3] & 0x0FU];

    return in;
}

```

```

ULL AES::RotWord(ULL in)
{
    unsigned char* Pointer = (unsigned char*)&in;
    unsigned char Mid = Pointer[0];
    Pointer[0] = Pointer[1];
    Pointer[1] = Pointer[2];
    Pointer[2] = Pointer[3];
    Pointer[3] = Mid;
    return in;
}

```

## Prime\_Generation.cpp

```
#include "Crypto_Curri_Design.h"
```

```
//素性检测算法
```

```

bool MillerRabin(const ZZ& n, long t)
{
    ZZ x;
    long i;
    bool flag = true;
    for (i = 0; i < t; i++)
    {
        x = RandomBnd(n);

```

```

    ZZ m, y, z;
    long j, k;
    if (x == 0)
        flag = true;
    else
    {
        k = 1;
        m = n / 2;
        while (m % 2 == 0)
        {
            k++;
            m /= 2;
        }
        z = PowerMod(x, m, n); // z = x^m % n
        if (z == 1)
            flag = true;
        else
        {
            j = 0;
            do {
                y = z;
                z = (y * y) % n;
                j++;
            } while (j < k && z != 1);
            if (z != 1 || y != n - 1)
                flag = false;
        }
    }

    if (flag == false)
        return false;
}

return flag;
}

//生成随机素数
void Prime_Generation(ZZ& p, ZZ& q, int length, ULL Seed, ULL Key1, ULL Key2)
{
    cout << "开始生成" << length << "bits的素数p, q" << endl;
    double Time = clock();
    while (1)
    {

```

```

        p = ANSI(Seed, length / 64, Key1, Key2);
        if (p % 2 == 0)
            p += 1;
        if (MillerRabin(p, length))
            break;
    }
    cout << "p = " << p << endl;
    while (1)
    {
        q = ANSI(Seed, length / 64, Key1, Key2);
        if (q % 2 == 0)
            q += 1;
        if (MillerRabin(q, length))
            break;
    }
    cout << "q = " << q << endl;
    Time = (clock() - Time) / 1000;
    cout << "生成的p, q通过素性检测, 用时 " << Time << 's';
    cout << endl;
}

```

## RSA.cpp

```
#include "Crypto_Curri_Design.h"
```

```
//生成加密指数b
```

```
ZZ b_Generation(ZZ phi_n)
```

```

{
    srand(time(0));
    ZZ b;
    while (1)
    {
        b = rand();
        if (GCD(b, phi_n) == 1)
            break;
    }
    return b;
}

```

```
//生成解密指数a:  $\phi_n \cdot x + b \cdot y = 1$ , b:加密指数, y:解密指数a
```

```
void Extend_Euclidean(ZZ a, ZZ b, ZZ& x, ZZ& y)
```

```

{
    ZZ m = to_ZZ(0), n = to_ZZ(1), t;

```

```

x = 1, y = 0;
while (b != to_ZZ(0))
{
    t = m, m = x - a / b * m, x = t;
    t = n, n = y - a / b * n, y = t;
    t = b, b = a % b, a = t;
}
}

```

//RSA加密明文x, 得到密文y

```

ZZ RSA_Encrypt(ZZ x, ZZ b, ZZ n)
{
    ZZ y;
    y = PowerMod(x, b, n);
    return y;
}

```

//RSA解密密文y, 得到明文x

```

ZZ RSA_Decrypt(ZZ y, ZZ a, ZZ n)
{
    ZZ x;
    x = PowerMod(y, a, n);
    return x;
}

```

## CBC.cpp

```

#include "Crypto_Curri_Design.h"

```

//CBC加密

```

void CBC_Encrypt(AES& aes, ZZ& IV, ifstream& infile, ofstream& fout)
{
    ZZ Cipher_Text, Plain_Text;
    unsigned char str[16];

    IV = RandomBnd(128);
    Cipher_Text = IV;

    while (1)
    {
        if (infile.read((char*)str, 16).gcount() != 16)
            break;
        ZZFromBytes(Plain_Text, str, 16L);
    }
}

```

```

    Plain_Text ^= Cipher_Text;
    aes.Encrypt(Cipher_Text, Plain_Text);
    BytesFromZZ(str, Cipher_Text, 16L);
    fout.write((const char*)str, 16);
}

//不足位填充
int Filling = 16 - (int)infile.gcount();
memset(str + infile.gcount(), Filling, Filling);
ZZFromBytes(Plain_Text, str, 16L);
Plain_Text ^= Cipher_Text;
aes.Encrypt(Cipher_Text, Plain_Text);
BytesFromZZ(str, Cipher_Text, 16L);
fout.write((const char*)str, 16);
}

//CBC解密
void CBC_Decrypt(ZZ& IV, ifstream& infile, ofstream& fout, ZZ k_de)
{
    ZZ Cipher_Text, Plain_Text;
    unsigned char str[16];

    AES Alice(k_de);
    ZZ Past_Cipher_Text(IV);

    while (1)
    {
        infile.read((char*)str, 16);
        ZZFromBytes(Cipher_Text, str, 16L);

        Alice.Decrypt(Plain_Text, Cipher_Text);
        Plain_Text ^= Past_Cipher_Text;

        Past_Cipher_Text = Cipher_Text;
        BytesFromZZ(str, Plain_Text, 16L);
        if (infile.peek() != EOF)
            fout.write((const char*)str, 16);
        else
        {
            //最后一字节
            fout.write((const char*)str, 16 - str[15]);
            break;
        }
    }
}

```

}

}

}