现代密码学课程设计说明书

1953484-孙卓阳-现代密码学课程设计

1.总体程序设计说明

（1）采用工程的思想，以main为顶层模块，头文件连接实现对多个具体实现的调用

main内只包含变量定义，输入输出重定向，输出提示与函数调用

（2）输入数据以文件输入

输出的输入提示与用户友好界面输出至控制台，公钥和密文的传递通过文件输出与读入实现

（3）具体执行流程为：

```
1.Alice生成RSA密钥过程：
    1.生成随机大素数p,q （512 bits/1024 bits)
        ZZ generate_prime(p, q, bit_len, seed, key1, key2)
        1.读入生成随机数的密钥种子seed，密钥1key1，密钥2key2；
        2.ansi_9.17算法生成随机数（seed，bit_len/64, key1，key2）：
            ZZ generate_random_num(s, num, k1, k2);
            1.根据要求生成的位数 bit_len多次调用DES生成64bits数：
                1.应用DES加密得64 bits随机数：
                    DES_cipher(m, key)
                    DES_decipher(code, key);
                2.拼接生成的随机数；
        3.应用Miller-rabbin算法进行素性检测，若不通过则返回2.生成随机数
            prime_test(n, t);
    2.生成n=pq,phi(n)=(p-1)(q-1)
    3.生成加密指数b
        void generate_b(ZZ& b, ZZ f_n);
    4.生成解密指数a
        ZZ exp_euclidean(ZZ b, ZZ n);
    5.输出公钥(n,b)到文件public_key.txt
2.Bob加密明文过程：
    1.读入公钥(n,b)和明文m
    2.生成128bits随机数k作为临时会话密钥
        ZZ generate_random_num(s, num, k1, k2);
    3.应用RSA加密k得c1
        ZZ RSA_cipher(ZZ x, ZZ b, ZZ n);
    4.应用AES加密m得c2，密钥为k
        ZZ AES_cipher(ZZ m, ZZ k);
    5.输出(c1,c2)到文件cipher.txt
3.Alice接收（c1,c2）并解密得明文：
    1.读入(c1,c2)
    2.使用私钥a解密c1得AES加密密钥k
        ZZ RSA_decipher(ZZ y, ZZ a, ZZ n);
    3.使用AES密钥k解密c2得明文
        ZZ AES_decipher(ZZ cipher, ZZ k);
```

（4）头文件$course_design.h$内包含的函数如下：

```
//tools
void conver_zz(ZZ a, int len, int& b);
```

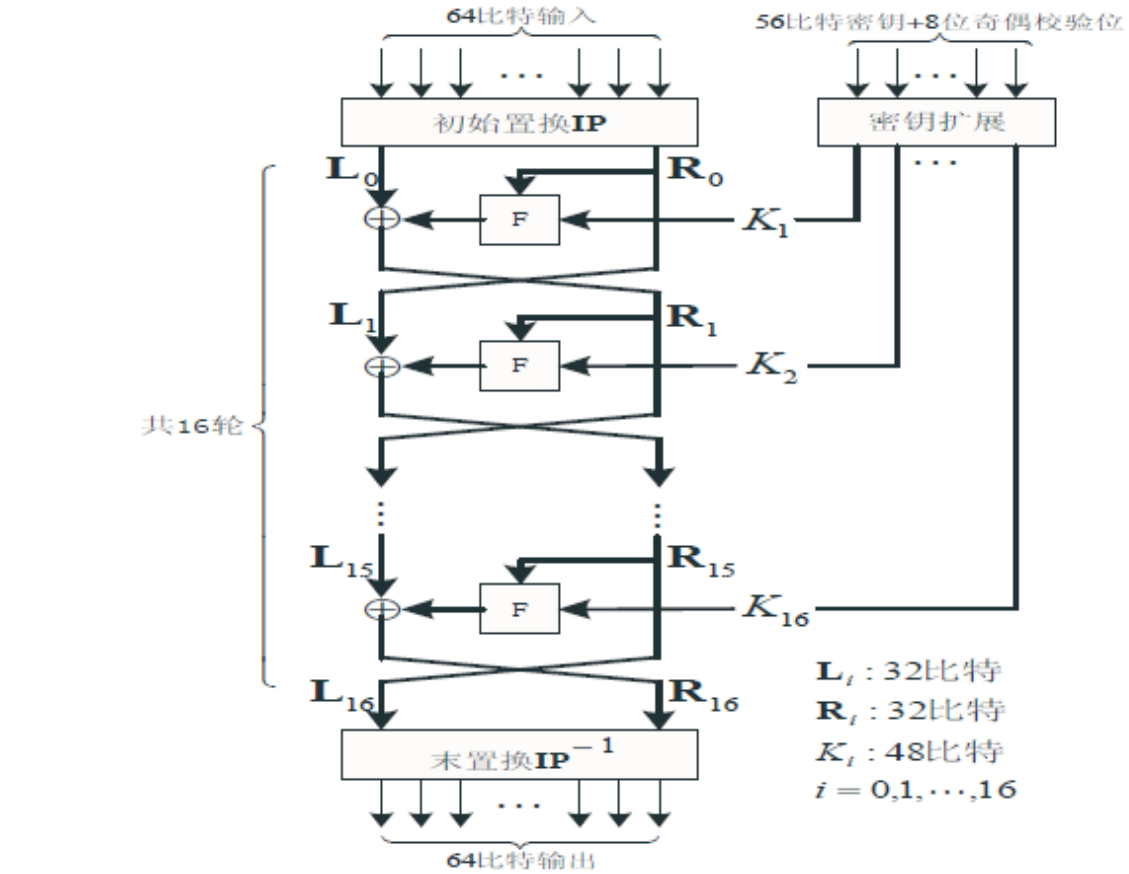
```
void generate_prime(ZZ& p, ZZ& q, int bit_len, unsigned long long seed, unsigned
long long key1, unsigned long long key2);
void generate_b(ZZ& b, ZZ f_n);
void input_path(char* in_path, char* ciphertext_path);
ZZ exp_euclidean(ZZ b, ZZ n);
//RSA
ZZ RSA_cipher(ZZ x, ZZ b, ZZ n);
ZZ RSA_decipher(ZZ y, ZZ a, ZZ n);
//素性检测
long prime_test(const ZZ& n, long t);
//随机数产生器
ZZ generate_random_num(unsigned long long s, int num, unsigned long long k1,
unsigned long long k2);
unsigned long long DES_cipher(unsigned long long m, unsigned long long key);
unsigned long long DES_decipher(unsigned long long code, unsigned long long
key);
//AES
ZZ AES_cipher(ZZ m, ZZ k);
ZZ AES_decipher(ZZ cipher, ZZ k);
```

2.具体实现说明

（1）DES

1.代码结构根据下图的算法结构图设计

DES算法结构图：



DES为对称加密，采用$Feistel$轮函数结构

2.加密过程：

先将输入的64bits进行初始置换，置换表如下：

```
//初始IP置换
const int IP[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
};
```

然后将置换得的比特串平分为left和right，下面进入16次运算循环

每一次循环：

1）先进行一次扩展置换，置换表如下：

```
//扩展置换
const int ep[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
};
```

2）置换后的比特流与根据密钥编排规律与密钥置换生成的密钥进行异或运算

密钥编排规律与密码置换如下：

```
//密码编排规则
const int enckey_order[16] = {
    1, 1, 2, 2 ,2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};
```

```
//密码置换
const int key_IP[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
};
```

3）异或得的比特串进行s盒运算（s盒数据过多，此处省略，详见附录）

4）进行置换P

```
//S盒运算后的那个置换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,
};
```

5）执行完以上步骤后的比特串与left进行异或，left = right，right = 运算得的比特串

3.解密过程：

同加密过程，但是应用扩展密钥的顺序为加密的逆序，其余过程相同

（2）$generate\_random\_num$ 生成随机数

根据任务书要求使用$ansi_9.17$算法实现

伪代码如下：

```
bits[i] = DES_cipher((DES_decipher((DES_cipher(l ^ s, k1)), k2)), k1);
s = DES_cipher(DES_decipher(DES_cipher(bits[i] ^ l, k1), k2), k1);
```

D为当前系统时间，k1为密钥1，k2为密钥2

存在的问题：要求生成512bits或1024bits的随机数，必须得用ZZ变量存储

但是des实现时用的变量类型为unsigned long long，所以需要转换一下

首先将64bits的整数按10进制分解为字符数组存储，然后按顺序赋值给输出的ZZ变量

（3）$prime_test\ Miller_Rabin$素性检测

伪代码如下：

```
def Miller-Rabin(n):
    b = pow(a,m) % n
    if b %n == 1 % n:
        return True
    for i in range(0, k):
        if b % n == -1 % n:
            return True
        else:
            b = b*b % n
    return False
```

上述代码为单次素性检测，根据要求，512bits的随机数通过素性检测的数不是素数的概率约为$\frac{1}{2^{512}}$

所以还需要一个循环控制函数，伪码如下：

```
def prime_test(const ZZ& n, long t):
    if n <= 1:
        return False;

    for i in range(0, t):
        x = RandomBnd(n)// random number between 0 and n-1

        if (Miller-Rabin(n, x))
            return False

    return True
```

（4）AES

1.单次AES加密过程：

根据如下算法流程图设计：

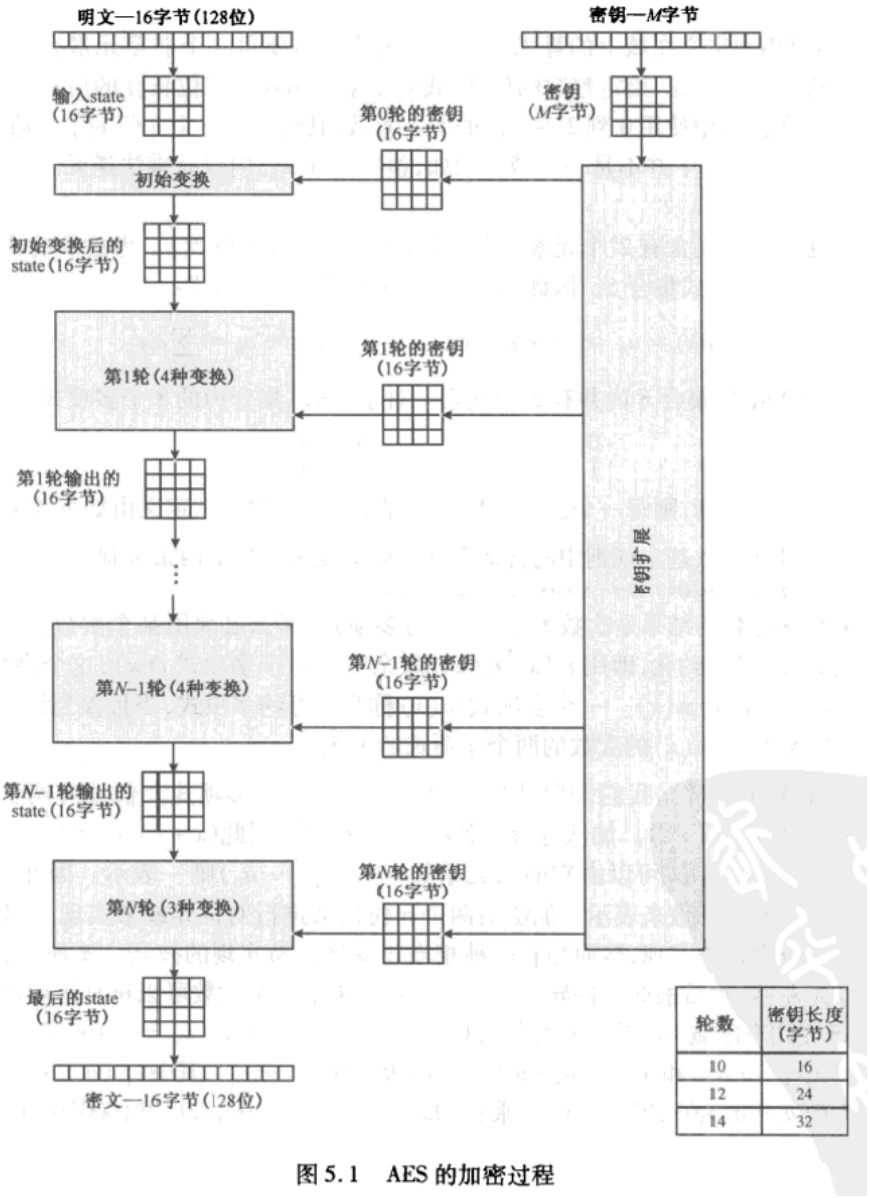


图 5.1　AES 的加密过程

设置64bits密钥转为4x4的16进制数数组，再进行密钥扩展得4x44的扩展密钥
进入加密：
首先将明文与扩展密钥的第一轮进行异或
然后进入循环，循环9次
　　单次循环：
　　　　s盒置换运算
　　　　行移位运算
　　　　列混合运算
　　　　扩展密钥异或运算
s盒置换运算
行移位运算
扩展密钥异或运算

2.单次AES解密过程：

总体过程与加密过程相似，运算的执行顺序有所不同，使用的扩展密钥的顺序为加密时的逆序

整体上为加密过程的逆过程

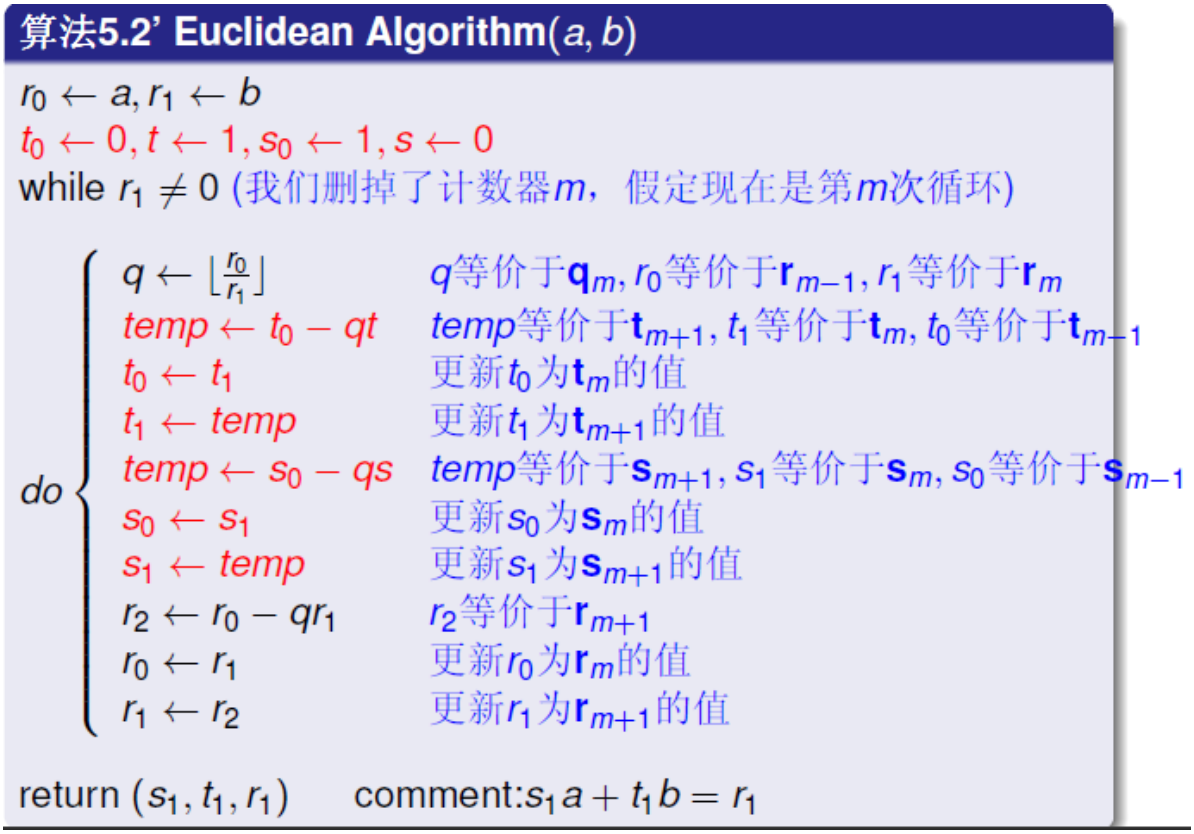
伪码：

```
设置64bits密钥转为4x4的16进制数数组，再进行密钥扩展得4x44的扩展密钥
进入加密：
首先将明文与扩展密钥的第一轮进行异或
然后进入循环，循环9次
    单次循环：
        行移位运算
        逆s盒置换运算
        扩展密钥异或运算
        列混合运算
行移位运算
s盒置换运算
扩展密钥异或运算
```

（5）$exp_euclidean$扩展欧几里得算法

计算模逆元，用于生成私钥的解密指数a

根据课件的伪码设计：



3.输入输出格式说明：

（1）以txt文件作为输入输出的载体；

（2）文件名以下形式均可以：
a.txt：不带路径形式；
..\data\b.dat：相对路径形式；
C:\Windows\System32\c.dat：绝对相对路径形式。

（3）所有手工输入从控制台中键盘输入，其余为文件输入；

输出以控制台输出为主，文件输出为辅（作为密钥公钥的传输模拟）。

（4）首先输入含有：生成随机数的64bits的种子，密钥key1，密钥key2的txt文件名；

然后输入存储公钥的文件名（存储生成的公钥，Bob加密时从此文件读取公钥）；

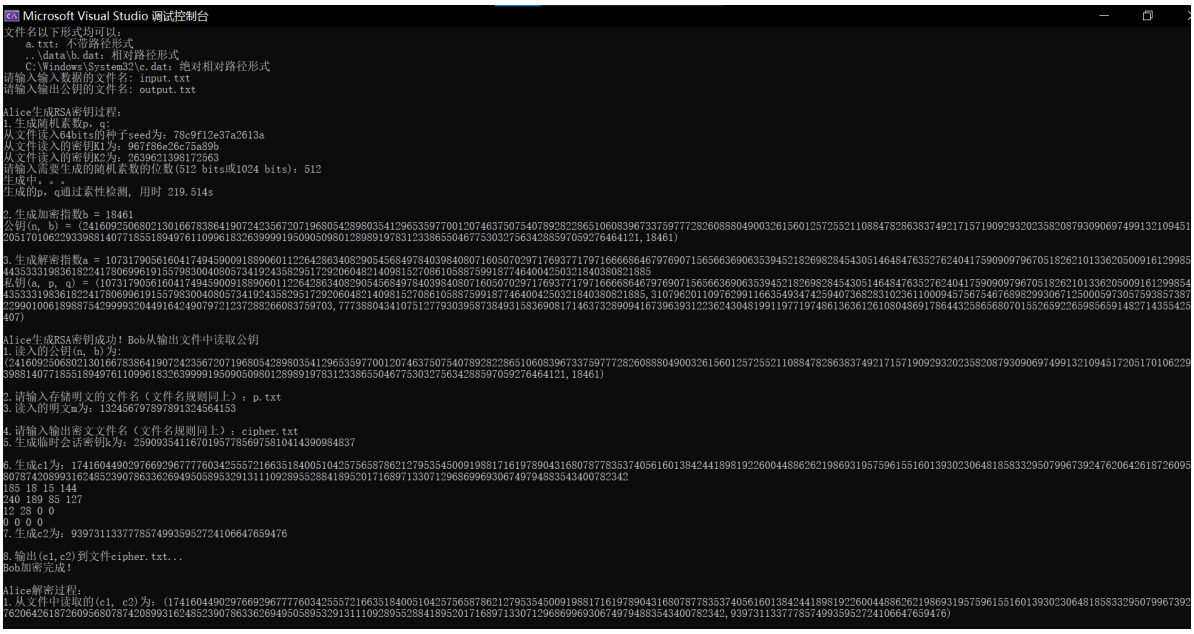
Alice生成RSA密钥成功后，Bob加密时：

先从存储公钥的文件读取公钥（此步不需要输入文件名）；

再输入存储明文的文件名（明文限定为数字），读取明文m；

然后输入存储：加密临时密钥k和明文得到的(c1,c2)的文件名（第三步Alice解密时从此文件读取（c1，c2））；

4.执行实例：



5.附录（源码）

(1)main.cpp

```
//顶层模块，  全流程
int main()
{
    // 变量定义
    int bit_len = 0;
    char in_path[100] = { 0 }, pub_key_path[100] = { 0 };
    ZZ p, q, n, b, a, f_n;
    ull seed, key1, key2;
    //文件处理
    input_path(in_path, pub_key_path);
    ifstream cinfile(in_path, ios::in | ios::binary);
    ofstream out(pub_key_path, ios::out | ios::binary);
    if (!cinfile.is_open())
    {
        cerr << "打开文件失败，请检查路径是否正确，程序退出。" << endl;
        return -1;
    }
    //ZZ temp_k, temp_m, ek, dk;
    //生成随机素数
    cinfile >> seed;
    cinfile >> key1;
    cinfile >> key2;
////////////////////////////////////////////////////////////////////////////////
/////////
    cerr << endl;
    cerr << "Alice生成RSA密钥过程：" << endl;
```

```
    cerr << "1.生成随机素数p，q:" << endl;
    //生成随机素数
    //cerr可以在用管道运算符重定向到文件时使得提示文字仍可显示在cmd窗口
    cerr << "从文件读入64bits的种子seed为：";
    cinfile >> seed;
    cerr << hex << seed << endl;
    cerr << "从文件读入的密钥K1为：";
    cinfile >> seed;
    cerr << hex << key1 << endl;
    cerr << "从文件读入的密钥K2为：";
    cinfile >> key2;
    cerr << hex << key2 << endl;
    cerr << "请输入需要生成的随机素数的位数(512 bits或1024 bits)：";
    cin >> bit_len;
    //生成随机素数
    generate_prime(p, q, bit_len, seed, key1, key2);
    n = p * q;
    f_n = (p - 1) * (q - 1);
    //生成加密指数b
    generate_b(b, f_n);
    cerr << "2.生成加密指数b = " << b << endl;
    cerr << "公钥(n, b) = (" << n << ',' << b << ')' << endl;
    out << n << ' ' << b << endl;
    cerr << endl;
    //生成解密指数a
    a = exp_euclidean(b, f_n);
    cerr << "3.生成解密指数a = " << a << endl;
    cerr << "私钥(a, p, q) = ";
    cerr << "(" << a << ',' << p << ',' << q << ')' << endl;;
    cerr << endl;
////////////////////////////////////////////////////////////////////////////////
/////////
    //Bob加密过程
    ZZ m, pub_b, pub_n, k, c1, c2;
    char cipher_path[100] = { 0 }, plaintext_path[100] = { 0 };
    cerr << "Alice生成RSA密钥成功！Bob从输出文件中读取公钥" << endl;

    //输入输出重定向
    // 读取公钥
    ifstream pkey_in(pub_key_path, ios::in | ios::binary);
    if (!pkey_in.is_open())
    {
        cerr << "打开公钥文件失败，程序退出" << endl;
        return -1;
    }
    pkey_in >> pub_n >> pub_b;
    cerr << "1.读入的公钥(n, b)为:" << endl;
    cerr << '(' << pub_n << ',' << pub_b << ')' << endl << endl;
    //读取明文
    cerr << "2.请输入存储明文的文件名（文件名规则同上）：";
    cin >> plaintext_path;
    ifstream plaintext_in(plaintext_path, ios::in | ios::binary);
    if (!plaintext_in.is_open())
    {
        cerr << "打开明文文件失败，程序退出" << endl;
        return -1;
    }
    cerr << "3.读入的明文m为：";
```

```
    plaintext_in >> m;
    cerr << m << endl << endl;
    //密文输出
    cerr << "4.请输入输出密文文件名（文件名规则同上）：";
    cin >> cipher_path;
    ofstream cipher_out(cipher_path, ios::in | ios::binary);

    //生成临时会话密钥
    k = generate_random_num(seed, 2, key2, key1);
    cerr << "5.生成临时会话密钥k为：" << k << endl << endl;
    //生成c1
    c1 = RSA_cipher(k, pub_b, pub_n);
    cerr << "6.生成c1为：" << c1 << endl;
    //生成c2
    c2 = AES_cipher(m, k);
    cerr << "7.生成c2为：" << c2 << endl << endl;
    cerr << "8.输出(c1,c2)到文件" << cipher_path << "..." << endl;
    cipher_out << c1 << ' ' << c2 << endl;
    cerr << "Bob加密完成！" << endl << endl;

////////////////////////////////////////////////////////////////////////////////
////////
    //Alice解密过程
    cerr << "Alice解密过程：" << endl;
    ZZ Bob_c1, Bob_c2, decipher_k, decipher_m;
    ifstream cipher_in(cipher_path, ios::in | ios::binary);
    if (!cipher_in.is_open())
    {
        cerr << "打开密文文件失败，程序退出" << endl;
        return -1;
    }

    cerr << "1.从文件中读取的(c1, c2)为：";
    cipher_in >> Bob_c1 >> Bob_c2;
    cerr << '(' << Bob_c1 << ',' << Bob_c2 << ')' << endl;
    decipher_k = RSA_decipher(Bob_c1, a, n);

    cerr << endl;
    cerr << "2.解密c1得密钥k："<< endl;
    cerr << decipher_k << endl;
    cerr << "  源密钥k：" << endl;
    cerr << k << endl << endl;

    decipher_m = AES_decipher(Bob_c2, decipher_k);
    cerr << "3.解密出的文件为: ";
    cerr << decipher_m << endl;
    cerr << "  源文件为:       ";
    cerr << m << endl;
    return 0;
}
```

(2)tools.cpp

```
//输入路径
void input_path(char *in_path, char *ciphertext_path)
{
    cerr << "文件名以下形式均可以：" << endl;
```

```
    cerr << "    a.txt：不带路径形式" << endl;
    cerr << "    ..\\data\\b.dat：相对路径形式" << endl;
    cerr << "    C:\\Windows\\System32\\c.dat：绝对相对路径形式" << endl;

    cerr << "请输入输入数据的文件名: ";
    cin >> in_path;
    cerr << "请输入输出公钥的文件名: ";
    cin >> ciphertext_path;

    return;
}
//生成随机素数
void generate_prime(ZZ &p, ZZ &q, int bit_len, unsigned long long seed, unsigned
long long key1, unsigned long long key2)
{
    cerr << "生成中。。。" << endl;
    double time = clock();
    while (1)
    {
        p = generate_random_num(seed, bit_len / 64, key1, key2);
        if (p % 2 == 0)
            p += 1;
        if (prime_test(p, bit_len))
            break;
    }
    while (1)
    {
        q = generate_random_num(seed, bit_len / 64, key1, key2);
        if (q % 2 == 0)
            q += 1;
        if (prime_test(q, bit_len))
        {
            time = (clock() - time) / 1000;
            cerr << "生成的p，q通过素性检测，用时 " << time << 's';
            cerr << endl << endl;
            break;
        }
    }
}
//生成加密指数b
void generate_b(ZZ &b, ZZ f_n)
{
    srand(time(0));
    while (1)
    {
        b = rand();
        if (GCD(b, f_n) == 1)
            break;
    }
}
//扩展Euclidean算法
ZZ ex_gcd(ZZ a, ZZ b, ZZ& x, ZZ& y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
```

```
    }
    ZZ r = ex_gcd(b, a % b, x, y);
    ZZ t = x;
    x = y;
    y = t - a / b * y;
    return r;
}
ZZ exp_euclidean(ZZ b, ZZ n)
{
    ZZ d, x, y, err;
    err = -1;
    d = ex_gcd(b, n, x, y);
    if (d == 1)
        return (x % n + n) % n;
    else
        return err;
}
//convert ZZ to int
void conver_zz(ZZ a, int len, int &b)
{
    b = 0;
    for (int i = 0; i < len; i++)
        if (bit(a, i) == 1)
        {
            b += pow(2, i);
        }
}
```

(3)random_num.cpp

```
//实现算法 ANSI X9.17
ZZ generate_random_num(unsigned long long s, int num, unsigned long long k1,
unsigned long long k2)
{
    unsigned long long l, D;
    unsigned long long * bits = new unsigned long long[num];// [64] = { 0 };
    char** t = new char*[num];
    for (int i = 0; i < num; i++)
        t[i] = new char[64];
    char *temp_zz = new char[num * 64];
    ZZ test = to_ZZ(0);
    memset(bits, 0, sizeof(ZZ) * num);
    memset(temp_zz, 0, sizeof(char) * num * 64);

    D = time(0);
    l = DES_cipher((DES_decipher((DES_cipher(D, k1)), k2)), k1);

    for (int i = 0; i < num; i++)
    {
        bits[i] = DES_cipher((DES_decipher((DES_cipher(l ^ s, k1)), k2)), k1);
        s = DES_cipher(DES_decipher(DES_cipher(bits[i] ^ l, k1), k2), k1);
    }
    for (int i = 0; i < num; i++)
    {
        unsigned long long temp = bits[i];
        int len = 0;
        while (temp != 0)
```

```
        {
            len++;
            temp /= 10;
        }
        for (int j = len - 1; j >= 0; j--)
        {
            t[i][j] = (bits[i] % 10) + '0';
            bits[i] /= 10;
        }
    }
    for (int i = 0; i < num; i++)
    {
        ZZ plus = to_ZZ(t[i]);
        test += plus;

        if(i < num -1)
            test <<= 64;
    }

    return test;
}
```

(4)DES.cpp

```
class DES {
private:
    //密码编排规则
    const int enckey_order[16] = {
        1, 1, 2, 2 ,2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
    };
    //初始IP置换
    const int IP[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
    };
    //IP置换的逆置换
    const int inverse_IP[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
    };//
    //密码置换
    const int key_IP[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
    };
    //扩展置换
    const int ep[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
    };
    //压缩置换
    const int cp[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
    };
    //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
    };
    //S盒运算后的那个置换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,
    };
public:
    unsigned long long key;
    unsigned long long cipher_text;
    unsigned long long decipher_text;
    unsigned long long plain_text;
    unsigned long long encode_key[16];
    //构造函数，初始化密钥，明文
    DES() {
        key = 0;
        cipher_text = 0;
        decipher_text = 0;
        plain_text = 0;
    }
    void set_key(unsigned long long k)
    {
        key = k;
    }
    void set_plaintext(unsigned long long p)
    {
        plain_text = p;
    }
    void set_ciphertext(unsigned long long c)
    {
        cipher_text = c;
    }
    void generate_enckey()
    {
        unsigned long long generate_key;
        generate_key = permutation(key, key_IP, 64, 56);
        for (int i = 0; i < 16; i++)
        {
            generate_key = shift(generate_key, enckey_order[i]);
            encode_key[i] = permutation(generate_key, cp, 56, 48);
        }
```

```
    }
    unsigned long long shift(unsigned long long key, int n)
    {
        unsigned long long tempKey = 0;
        unsigned long long L, R;
        L = (key & 0xFFFFFFF0000000LL) >> 28;
        R = key & 0x0000000FFFFFFF;
        if (n == 1) {
            L = ((L & 0x7FFFFF) << 1) | ((L >> 27) & 1);
            R = ((R & 0x7FFFFF) << 1) | ((R >> 27) & 1);
            tempKey = (L << 28) | R;
        }
        else if (n == 2) {
            L = ((L & 0x3FFFFF) << 2) | ((L >> 26) & 3);
            R = ((R & 0x7FFFFF) << 2) | ((R >> 26) & 3);
            tempKey = (L << 28) | R;
        }
        return tempKey;
    }
    unsigned long long permutation(unsigned long long num, const int p[], int
pmax, int n)
    {
        unsigned long long temp = 0;
        int i;
        for (i = 0; i < n; i++)
        {
            temp <<= 1;
            temp |= (num >> (pmax - p[i])) & 1;
        }
        return temp;
    }
    unsigned long long s_boxes(unsigned long long num)
    {
        unsigned long long temp, res = 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);
            res |= temp;
        }
        return res;
    }
    void encrypt()
    {
        unsigned long long L, R, temp_r, temp;

        temp = permutation(plain_text, IP, 64, 64);
        L = (temp & 0xFFFFFFFF00000000LL) >> 32;
        R = (temp & 0x00000000FFFFFFFFLL);
        temp_r = R;

        for (int i = 0; i < 16; i++)
        {
            temp_r = permutation(R, ep, 32, 48);
            temp_r = temp_r ^ encode_key[i];
```

```
            temp_r = s_boxes(temp_r);
            temp_r = permutation(temp_r, P, 32, 32);
            temp_r ^= L;
            L = R;
            R = temp_r;
        }
        temp = (R << 32) | L;
        temp = permutation(temp, inverse_IP, 64, 64);
        cipher_text = temp;
    }
    void decrypt()
    {
        unsigned long long L, R, temp_r, temp;

        temp = permutation(cipher_text, IP, 64, 64);
        L = (temp & 0xffffffff00000000LL) >> 32;
        R = (temp & 0x00000000ffffffffLL);
        temp_r = R;

        for (int i = 0; i < 16; i++)
        {
            temp_r = permutation(R, ep, 32, 48);
            temp_r = temp_r ^ encode_key[15 - i];
            temp_r = s_boxes(temp_r);
            temp_r = permutation(temp_r, P, 32, 32);
            temp_r ^= L;
            L = R;
            R = temp_r;
        }
        temp = (R << 32) | L;
        temp = permutation(temp, inverse_IP, 64, 64);
        decipher_text = temp;
    }
};

unsigned long long DES_cipher(unsigned long long m, unsigned long long key)
{
    DES des;
    des.set_key(key);
    des.set_plaintext(m);
    des.generate_enckey();
    des.encrypt();

    return des.cipher_text;
}
unsigned long long DES_decipher(unsigned long long code, unsigned long long key)
{
    DES des;
    des.set_ciphertext(code);
    des.set_key(key);
    des.generate_enckey();
    des.decrypt();
    return des.decipher_text;
}
```

(5)prime_test.cpp

```
long prime_test(const ZZ& n, long t)
{
    if (n <= 1)
        return 0;

    ZZ x;
    long i;

    for (i = 0; i < t; i++)
    {
        x = RandomBnd(n); // random number between 0 and n-1

        if (witness(n, x))
            return 0;
    }

    return 1;
}
long witness(const ZZ& n, const ZZ& x)
{
    ZZ m, y, z;
    long j, k;

    if (x == 0)
        return 0;

    k = 1;
    m = n / 2;
    while (m % 2 == 0)
    {
        k++;
        m /= 2;
    }

    z = PowerMod(x, m, n); // z = x^m % n
    if (z == 1) return 0;

    j = 0;
    do {
        y = z;
        z = (y * y) % n;
        j++;
    } while (j < k && z != 1);

    return z != 1 || y != n - 1;
}
```

(6)RSA.cpp

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

(7)AES.cpp

```
struct word
{
    ZZ wordKey[4];
};
class AES
{
private:
    const int Nb = 4, Nk = 4, Nr = 10;
    ZZ cipher_key[16];
    word plain_text[4];
    word decipher_text[4];
    word Rcon[11];
    word round_key[44];
    const ZZ s_box[16][16] = {
    {to_ZZ(99),  to_ZZ(124), to_ZZ(119), to_ZZ(123), to_ZZ(242), to_ZZ(107),
to_ZZ(111), to_ZZ(197), to_ZZ(48),  to_ZZ(01),  to_ZZ(103), to_ZZ(43),
 to_ZZ(254), to_ZZ(215), to_ZZ(171), to_ZZ(118)},
    {to_ZZ(202), to_ZZ(130), to_ZZ(201), to_ZZ(125), to_ZZ(250), to_ZZ(89),
 to_ZZ(71),  to_ZZ(240), to_ZZ(173), to_ZZ(212), to_ZZ(162), to_ZZ(175),
to_ZZ(156), to_ZZ(164), to_ZZ(114), to_ZZ(192)},
    {to_ZZ(183), to_ZZ(253), to_ZZ(147), to_ZZ(38),  to_ZZ(54),  to_ZZ(63),
 to_ZZ(247), to_ZZ(204), to_ZZ(52),  to_ZZ(165), to_ZZ(229), to_ZZ(241),
to_ZZ(113), to_ZZ(216), to_ZZ(49),  to_ZZ(21)},
    {to_ZZ(4),   to_ZZ(199), to_ZZ(35),  to_ZZ(195), to_ZZ(24),  to_ZZ(150),
to_ZZ(05),  to_ZZ(154), to_ZZ(07),  to_ZZ(18),  to_ZZ(128), to_ZZ(226),
to_ZZ(235), to_ZZ(39),  to_ZZ(178), to_ZZ(117)},
    {to_ZZ(9),   to_ZZ(131), to_ZZ(44),  to_ZZ(26),  to_ZZ(27),  to_ZZ(110),
to_ZZ(90),  to_ZZ(160), to_ZZ(82),  to_ZZ(59),  to_ZZ(214), to_ZZ(179),
to_ZZ(41),  to_ZZ(227), to_ZZ(47),  to_ZZ(132)},
    {to_ZZ(83),  to_ZZ(209), to_ZZ(00),  to_ZZ(237), to_ZZ(32),  to_ZZ(252),
to_ZZ(177), to_ZZ(91),  to_ZZ(106), to_ZZ(203), to_ZZ(190), to_ZZ(57),
 to_ZZ(74),  to_ZZ(76),  to_ZZ(88),  to_ZZ(207)},
    {to_ZZ(208), to_ZZ(239), to_ZZ(170), to_ZZ(251), to_ZZ(67),  to_ZZ(77),
 to_ZZ(51),  to_ZZ(133), to_ZZ(69),  to_ZZ(249), to_ZZ(02),  to_ZZ(127),
to_ZZ(80),  to_ZZ(60),  to_ZZ(159), to_ZZ(168)},
    {to_ZZ(81),  to_ZZ(163), to_ZZ(64),  to_ZZ(143), to_ZZ(146), to_ZZ(157),
to_ZZ(56),  to_ZZ(245), to_ZZ(188), to_ZZ(182), to_ZZ(218), to_ZZ(33),
 to_ZZ(16),  to_ZZ(255), to_ZZ(243), to_ZZ(210)},
    {to_ZZ(205), to_ZZ(12),  to_ZZ(19),  to_ZZ(236), to_ZZ(95),  to_ZZ(151),
to_ZZ(68),  to_ZZ(23),  to_ZZ(196), to_ZZ(167), to_ZZ(126), to_ZZ(61),
 to_ZZ(100), to_ZZ(93),  to_ZZ(25),  to_ZZ(115)},
```

```
    {to_zz(96),  to_zz(129), to_zz(79),  to_zz(220), to_zz(34),  to_zz(42),
 to_zz(144), to_zz(136), to_zz(70),  to_zz(238), to_zz(184), to_zz(20),
 to_zz(222), to_zz(94),  to_zz(11),  to_zz(219)},
    {to_zz(224), to_zz(50),  to_zz(58),  to_zz(10),  to_zz(73),  to_zz(6),
to_zz(36),  to_zz(92),  to_zz(194), to_zz(211), to_zz(172), to_zz(98),
 to_zz(145), to_zz(149), to_zz(228), to_zz(121)},
    {to_zz(231), to_zz(200), to_zz(55),  to_zz(109), to_zz(141), to_zz(213),
to_zz(78),  to_zz(169), to_zz(108), to_zz(86),  to_zz(244), to_zz(234),
to_zz(101), to_zz(122), to_zz(174), to_zz(8)},
    {to_zz(186), to_zz(120), to_zz(37),  to_zz(46),  to_zz(28),  to_zz(166),
to_zz(180), to_zz(198), to_zz(232), to_zz(221), to_zz(116), to_zz(31),
 to_zz(75),  to_zz(189), to_zz(139), to_zz(138)},
    {to_zz(112), to_zz(62),  to_zz(181), to_zz(102), to_zz(72),  to_zz(03),
 to_zz(246), to_zz(14),  to_zz(97),  to_zz(53),  to_zz(87),  to_zz(185),
to_zz(134), to_zz(193), to_zz(29),  to_zz(158)},
    {to_zz(225), to_zz(248), to_zz(152), to_zz(17),  to_zz(105), to_zz(217),
to_zz(142), to_zz(148), to_zz(155), to_zz(30),  to_zz(135), to_zz(233),
to_zz(206), to_zz(85),  to_zz(40),  to_zz(223)},
    {to_zz(140), to_zz(161), to_zz(137), to_zz(13),  to_zz(191), to_zz(230),
to_zz(66),  to_zz(104), to_zz(65),  to_zz(153), to_zz(45),  to_zz(15),
 to_zz(176), to_zz(84),  to_zz(187), to_zz(22)}
    };
    const zz inv_s_box[16][16] = {
    {to_zz(82),  to_zz(9),   to_zz(106), to_zz(213), to_zz(48),  to_zz(54),
 to_zz(165), to_zz(56),  to_zz(191), to_zz(64),  to_zz(163), to_zz(158),
to_zz(129), to_zz(243), to_zz(215), to_zz(251)},
    {to_zz(124), to_zz(227), to_zz(57),  to_zz(130), to_zz(155), to_zz(47),
 to_zz(255), to_zz(135), to_zz(52),  to_zz(142), to_zz(67),  to_zz(68),
 to_zz(196), to_zz(222), to_zz(233), to_zz(203)},
    {to_zz(84),  to_zz(123), to_zz(148), to_zz(50),  to_zz(166), to_zz(194),
to_zz(35),  to_zz(61),  to_zz(238), to_zz(76),  to_zz(149), to_zz(11),
 to_zz(66),  to_zz(250), to_zz(195), to_zz(78)},
    {to_zz(8),   to_zz(46),  to_zz(161), to_zz(102), to_zz(40),  to_zz(217),
to_zz(36),  to_zz(178), to_zz(118), to_zz(91),  to_zz(162), to_zz(73),
 to_zz(109), to_zz(139), to_zz(209), to_zz(37)},
    {to_zz(114), to_zz(248), to_zz(246), to_zz(100), to_zz(134), to_zz(104),
to_zz(152), to_zz(22),  to_zz(212), to_zz(164), to_zz(92),  to_zz(204),
to_zz(93),  to_zz(101), to_zz(182), to_zz(146)},
    {to_zz(108), to_zz(112), to_zz(72),  to_zz(80),  to_zz(253), to_zz(237),
to_zz(185), to_zz(218), to_zz(94),  to_zz(21),  to_zz(70),  to_zz(87),
 to_zz(167), to_zz(141), to_zz(157), to_zz(132)},
    {to_zz(144), to_zz(216), to_zz(171), to_zz(0),   to_zz(140), to_zz(188),
to_zz(211), to_zz(10),  to_zz(247), to_zz(228), to_zz(88),  to_zz(5),
to_zz(184), to_zz(179), to_zz(69),  to_zz(6)},
    {to_zz(208), to_zz(44),  to_zz(30),  to_zz(143), to_zz(202), to_zz(63),
 to_zz(15),  to_zz(2),   to_zz(193), to_zz(175), to_zz(189), to_zz(3),
to_zz(1),   to_zz(19),  to_zz(138), to_zz(107)},
    {to_zz(58),  to_zz(145), to_zz(17),  to_zz(65),  to_zz(79),  to_zz(103),
to_zz(220), to_zz(234), to_zz(151), to_zz(242), to_zz(207), to_zz(206),
to_zz(240), to_zz(180), to_zz(230), to_zz(115)},
    {to_zz(150), to_zz(172), to_zz(116), to_zz(34),  to_zz(231), to_zz(173),
to_zz(53),  to_zz(133), to_zz(226), to_zz(249), to_zz(55),  to_zz(232),
to_zz(28),  to_zz(117), to_zz(223), to_zz(110)},
    {to_zz(71),  to_zz(241), to_zz(26),  to_zz(113), to_zz(29),  to_zz(41),
 to_zz(197), to_zz(137), to_zz(111), to_zz(183), to_zz(98),  to_zz(14),
 to_zz(170), to_zz(24),  to_zz(190), to_zz(27)},
```

```
    {to_ZZ(252), to_ZZ(86),  to_ZZ(62),  to_ZZ(75),  to_ZZ(198), to_ZZ(210),
to_ZZ(121), to_ZZ(32),  to_ZZ(154), to_ZZ(219), to_ZZ(192), to_ZZ(254),
to_ZZ(120), to_ZZ(205), to_ZZ(90),  to_ZZ(244)},
    {to_ZZ(31),  to_ZZ(221), to_ZZ(168), to_ZZ(51),  to_ZZ(136), to_ZZ(7),
to_ZZ(199), to_ZZ(49),  to_ZZ(177), to_ZZ(18),  to_ZZ(16),  to_ZZ(89),
 to_ZZ(39),  to_ZZ(128), to_ZZ(236), to_ZZ(95)},
    {to_ZZ(96),  to_ZZ(81),  to_ZZ(127), to_ZZ(169), to_ZZ(25),  to_ZZ(181),
to_ZZ(74),  to_ZZ(13),  to_ZZ(45),  to_ZZ(229), to_ZZ(122), to_ZZ(159),
to_ZZ(147), to_ZZ(201), to_ZZ(156), to_ZZ(239)},
    {to_ZZ(160), to_ZZ(224), to_ZZ(59),  to_ZZ(77),  to_ZZ(174), to_ZZ(42),
 to_ZZ(245), to_ZZ(176), to_ZZ(200), to_ZZ(235), to_ZZ(187), to_ZZ(60),
 to_ZZ(131), to_ZZ(83),  to_ZZ(153), to_ZZ(97)},
    {to_ZZ(23),  to_ZZ(43),  to_ZZ(4),   to_ZZ(126), to_ZZ(186), to_ZZ(119),
to_ZZ(214), to_ZZ(38),  to_ZZ(225), to_ZZ(105), to_ZZ(20),  to_ZZ(99),
 to_ZZ(85),  to_ZZ(33),  to_ZZ(12),  to_ZZ(125)}
    };
public:
    word cipher_text[4];
    AES() {
        ini_Rcon();
    }
    void ini_Rcon()
    {
        for (int i = 0; i < 10; i++)
            for (int j = 0; j < 4; j++)
                Rcon[i].wordKey[j] = 0;
        Rcon[1].wordKey[0] = 01;
        Rcon[2].wordKey[0] = 02;
        Rcon[3].wordKey[0] = 04;
        Rcon[4].wordKey[0] = 8;
        Rcon[5].wordKey[0] = 16;
        Rcon[6].wordKey[0] = 32;
        Rcon[7].wordKey[0] = 64;
        Rcon[8].wordKey[0] = 128;
        Rcon[9].wordKey[0] = 27;
        Rcon[10].wordKey[0] = 54;
    }
    void set_key(ZZ key)
    {
        for (int i = 0; i < 16; i++)
        {
            ZZ temp;
            int temp_1 = 0;
            for (int j = 0; j < 8; j++)
                if (bit(key, j) == 1)
                    temp_1 += pow(2, j);
            temp = temp_1;
            cipher_key[i] = temp;
            key >>= 8;
        }
        key_expansion(cipher_key, round_key);
        return;
    }
    void set_plaintext(ZZ p_text)
    {
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
```

```
            {
                ZZ temp;
                int temp_1 = 0;
                for (int k = 0; k < 8; k++)
                    if (bit(p_text, k) == 1)
                        temp_1 += pow(2, k);

                temp = temp_1;
                plain_text[i].wordKey[j] = temp;
                p_text >>= 8;
            }
        }
    }
    void set_cipher(ZZ cipher)
    {
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                ZZ temp, v2 = to_ZZ(2);
                int temp_1 = 0;
                for (int k = 0; k < 8; k++)
                    if (bit(cipher, k) == 1)
                        temp += power(v2, k);

                cipher_text[i].wordKey[j] = temp;
                cipher >>= 8;
            }
        }
    }

    word rotWord(word w)
    {
        word temp;
        for (int i = 0; i < 4; i++)
            temp.wordKey[(i + 3) % 4] = w.wordKey[i];
        return temp;
    }
    word subWord(word w)
    {
        ZZ t_l, t_r;
        int L = 0, R = 0;
        for (int i = 0; i < 4; i++)
        {
            t_l = w.wordKey[i] >> 4;
            t_r = w.wordKey[i] & 0x0F;
            conver_zz(t_l, 8, L);
            conver_zz(t_r, 8, R);
            w.wordKey[i] = s_box[L][R];
        }
        return w;
    }
    word wordXor(word w1, word w2)
    {
        word temp;
        for (int i = 0; i < 4; i++)
            temp.wordKey[i] = w1.wordKey[i] ^ w2.wordKey[i];
        return temp;
```

```
}
void key_expansion(ZZ key[], word w[])
{
    int i = 0;
    word temp;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            w[j].wordKey[i] = key[j + 4 * i];

    i = 4;
    while (i < 44)
    {
        temp = w[i - 1];
        if ((i % 4) == 0)
        {
            temp = rotWord(temp);
            temp = subWord(temp);
            temp = wordXor(temp, Rcon[i / 4]);
        }
        w[i] = wordXor(w[i - 4], temp);
        i++;
    }
}

void sub_byte(word w[])
{
    ZZ t_l, t_r;
    int l, r;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            t_l = w[i].wordKey[j] >> 4;
            t_r = w[i].wordKey[j] & 0x0F;
            conver_zz(t_l, 8, l);
            conver_zz(t_r, 8, r);
            w[i].wordKey[j] = s_box[l][r];
        }
    }
}
void sub_shift(word& sub_w)
{
    ZZ temp = sub_w.wordKey[0];
    for (int i = 1; i <= 3; i++)
        sub_w.wordKey[i - 1] = sub_w.wordKey[i];
    sub_w.wordKey[3] = temp;
}
void shift_rows(word w[])
{
    for (int i = 1; i < 4; i++)
        for (int j = 0; j < i; j++)
            sub_shift(w[i]);
}
ZZ field_mul(ZZ x)
{
    ZZ y, t;
    t = x & 0b10000000;
```

```
        if (t != 0)
            y = ((x - 128) << 1) ^ 27;
        else
            y = x << 1;

        return y;
    }
    void mix_columns(word w[])
    {
        for (int i = 0; i < 4; i++)
        {
            ZZ x0, x1, x2, x3;
            x0 = w[0].wordKey[i];
            x1 = w[1].wordKey[i];
            x2 = w[2].wordKey[i];
            x3 = w[3].wordKey[i];

            w[0].wordKey[i] = x1 ^ x2 ^ x3;
            w[1].wordKey[i] = x0 ^ x2 ^ x3;
            w[2].wordKey[i] = x0 ^ x1 ^ x3;
            w[3].wordKey[i] = x0 ^ x1 ^ x2;

            x0 = field_mul(x0);
            x1 = field_mul(x1);
            x2 = field_mul(x2);
            x3 = field_mul(x3);

            w[0].wordKey[i] = w[0].wordKey[i] ^ x0 ^ x1;
            w[1].wordKey[i] = w[1].wordKey[i] ^ x1 ^ x2;
            w[2].wordKey[i] = w[2].wordKey[i] ^ x2 ^ x3;
            w[3].wordKey[i] = w[3].wordKey[i] ^ x3 ^ x0;
        }
    }
    void add_round_key(word w[], int round)
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                w[j].wordKey[i] ^= round_key[i + 4 * round].wordKey[j];
    }
    void encrypt()
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                cipher_text[i].wordKey[j] = plain_text[i].wordKey[j];
        add_round_key(cipher_text, 0);
        for (int i = 1; i < 10; i++)
        {
            sub_byte(cipher_text);
            shift_rows(cipher_text);
            mix_columns(cipher_text);
            add_round_key(cipher_text, i);
        }
        sub_byte(cipher_text);
        shift_rows(cipher_text);
        add_round_key(cipher_text, 10);
    }

    void inv_sub_byte(word w[])
```

```
    {
        ZZ t_l, t_r;
        int l, r;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                t_l = w[i].wordKey[j] >> 4;
                t_r = w[i].wordKey[j] & 0x0F;
                conver_zz(t_l, 8, l);
                conver_zz(t_r, 8, r);
                w[i].wordKey[j] = inv_s_box[l][r];
            }
        }
    }
    void inv_shoft_rows(word w[])
    {
        for (int i = 1; i < 4; i++)
            for (int j = 4; j > i; j--)
                sub_shift(w[i]);
    }
    void inv_mix_columns(word w[])
    {
        for (int i = 0; i < 4; i++)
        {
            ZZ x0, x1, x2, x3;
            x0 = w[0].wordKey[i];
            x1 = w[1].wordKey[i];
            x2 = w[2].wordKey[i];
            x3 = w[3].wordKey[i];

            w[0].wordKey[i] = x1 ^ x2 ^ x3;
            w[1].wordKey[i] = x0 ^ x2 ^ x3;
            w[2].wordKey[i] = x0 ^ x1 ^ x3;
            w[3].wordKey[i] = x0 ^ x1 ^ x2;

            x0 = field_mul(x0);
            x1 = field_mul(x1);
            x2 = field_mul(x2);
            x3 = field_mul(x3);

            w[0].wordKey[i] = w[0].wordKey[i] ^ x0 ^ x1;
            w[1].wordKey[i] = w[1].wordKey[i] ^ x1 ^ x2;
            w[2].wordKey[i] = w[2].wordKey[i] ^ x2 ^ x3;
            w[3].wordKey[i] = w[3].wordKey[i] ^ x3 ^ x0;

            x0 = field_mul(x0 ^ x2);
            x1 = field_mul(x1 ^ x3);

            w[0].wordKey[i] = w[0].wordKey[i] ^ x0;
            w[1].wordKey[i] = w[1].wordKey[i] ^ x1;
            w[2].wordKey[i] = w[2].wordKey[i] ^ x0;
            w[3].wordKey[i] = w[3].wordKey[i] ^ x1;

            x0 = field_mul(x0 ^ x1);

            w[0].wordKey[i] = w[0].wordKey[i] ^ x0;
            w[1].wordKey[i] = w[1].wordKey[i] ^ x0;
```

```
            w[2].wordKey[i] = w[2].wordKey[i] ^ x0;
            w[3].wordKey[i] = w[3].wordKey[i] ^ x0;
        }
    }
    void decrypt()
    {
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                decipher_text[i].wordKey[j] = cipher_text[i].wordKey[j];
        add_round_key(decipher_text, 10);
        for (int i = 9; i > 0; i--)
        {
            inv_shoft_rows(decipher_text);
            inv_sub_byte(decipher_text);
            add_round_key(decipher_text, i);
            inv_mix_columns(decipher_text);
        }
        inv_shoft_rows(decipher_text);
        inv_sub_byte(decipher_text);
        add_round_key(decipher_text, 0);
    }

    ZZ get_cipher(int len)
    {
        ZZ res = to_ZZ(0);
        for (int i = 3; i >= 0; i--)
        {
            for (int j = 3; j >= 0; j--)
            {
                res += cipher_text[i].wordKey[j];
                if (i == 0 && j == 0)
                    break;
                else
                    res <<= 8;
            }
        }
        return res;
    }
    ZZ get_decipher_text(int len)
    {
        ZZ res = to_ZZ(0);
        int start = 0, flag = 0, count = 0, space = 128 - len;
        start = space / 8;
        space %= 8;
        for (int i = 3; i >= 0; i--)
        {
            for (int j = 3; j >= 0; j--)
            {
                count++;
                if (count > start)
                {
                    res += decipher_text[i].wordKey[j];
                    if (count < 16)
                        res <<= 8;
                }
            }
        }
        return res;
```

```
    }
};

ZZ AES_cipher(ZZ m, ZZ k)
{
    ZZ cipher = to_ZZ(0);
    ZZ temp = m;
    ZZ y;
    int bit_len = NumBits(m);
    while (bit_len > 0)
    {
        AES aes;
        ZZ temp_m = to_ZZ(0), v2 = to_ZZ(2);

        for (int i = 0; i < 128; i++)
        {
            if (bit(m, i) == 1)
                temp_m += power(v2, i);
        }
        temp_m ^= y;
        aes.set_plaintext(temp_m);
        aes.set_key(k);
        aes.show_plaintext();
        aes.encrypt();
        y = aes.get_cipher(bit_len);
        cipher += y;
        if (bit_len > 128)
            cipher <<= 128;
        m >>= 128;
        bit_len -= 128;
    }
    return cipher;
}
ZZ AES_decipher(ZZ cipher, ZZ k)
{
    ZZ decipher_text = to_ZZ(0);
    ZZ temp = cipher;
    ZZ y;
    int bit_len = NumBits(cipher);
    while (bit_len > 0)
    {
        AES aes;
        ZZ temp_m = to_ZZ(0), v2 = to_ZZ(2);

        for (int i = 0; i < 128; i++)
        {
            if (bit(cipher, i) == 1)
                temp_m += power(v2, i);
        }
        aes.set_cipher(temp_m);
        aes.set_key(k);

        aes.decrypt();
        y = aes.get_decipher_text(bit_len) ^ y;
        decipher_text += y;

        if (bit_len > 128)
            decipher_text <<= 128;
```

```
        cipher >>= 128;
        bit_len -= 128;
    }
    return decipher_text;
}
```