DiffHellman

程序

```cpp
#include<cstdio>
#include<vector>
using namespace std;

struct Point
{
    int x, y;
    Point() {}
    Point(int X, int Y)
    {
        x = X;
        y = Y;
    }
};

int Inv[10001];
int k;

int Mod(int a, int p)
{
    return (a % p >= 0 ? a % p : (a % p) + p);
}

void Inverse(int p)
{
    for (int i = 1; i < p; i++)
        for (int j = 1; j < p; j++)
            if ((i * j) % p == 1)
                Inv[i] = j;
}

Point Plus(Point p1, Point p2, int p, int a)
{
    int Lambda;
    Point TmpPoint;
    if (p1.x == p2.x && p1.y == p2.y)
        Lambda = Mod((3 * p1.x * p1.x + a) * Inv[Mod(2 * p1.y, p)], p);
    else
        Lambda = Mod((p2.y - p1.y) * Inv[Mod(p2.x - p1.x, p)], p);
    TmpPoint.x = Mod(Lambda * Lambda - p1.x - p2.x, p);
    TmpPoint.y = Mod(Lambda * (p1.x - TmpPoint.x) - p1.y, p);
    //printf("lambda=%d,x=%d,y=%d\n",Lambda,TmpPoint.x,TmpPoint.y);
    return TmpPoint;
}

Point Minus(Point p1, Point p2, int p, int a)
{
    Point Minus_p2(p2.x, Mod(p - p2.y, p));
```

```
    return Plus(p1, Minus_p2, p, a);
}

Point Multiple(int k, Point p1, int p, int a)
{
    Point TmpPoint(p1.x, p1.y);
    for (int i = 1; i < k; i++)
    {
        TmpPoint = Plus(TmpPoint, p1, p, a);
        //printf("(%d,%d)\n",TmpPoint.x,TmpPoint.y);
    }
    return TmpPoint;
}

Point Choose_G(int p, int a, int b)
{
    printf("\nChoose a generator frome below (input in the format of x y):\n");
    vector<Point> G_list;
    for (int i = 0; i < p; i++)
        for (int j = 0; j < p; j++)
            if (((i * i * i + a * i + b) % p) == ((j * j) % p))
                G_list.push_back(Point(i, j));
    for (int i = 0; i < G_list.size(); i++)
        printf("(%d,%d) ", G_list[i].x, G_list[i].y);
    int X, Y;
    printf("\nYou choose ");
    scanf("%d%d", &X, &Y);
    bool flag;
    do
    {
        flag = false;
        for (int i = 0; i < G_list.size(); i++)
        {
            if (G_list[i].x == X && G_list[i].y == Y)
            {
                flag = true;
            }
        }
        if (!flag)
        {
            printf("\nWrong! You should choose among the list above!");
            printf("\nYou choose ");
            scanf("%d%d", &X, &Y);
        }
    } while (!flag);
    //printf("Success1\n");
    return Point(X, Y);
}

void Diff(int &p, int &a, int &b)
{
    printf("\n-----------------------------------------------------------------\n");
    printf("ENCODE BEGIN:");
    printf("\n-----------------------------------------------------------------\n");
    printf("\nInput parameters of ECC(y^2=x^3+ax+b mod p):\n");
    printf("a=");
    scanf("%d", &a);
    printf("b=");
```

```
107         scanf("%d", &b);
108         printf("p=");
109         scanf("%d", &p);
110 
111         printf("\n");
112         Inverse(p);
113 
114         Point G = Choose_G(p, a, b);
115 
116         printf("\nYou have choosen ECC E_%d(%d,%d) and generator (%d,%d)\n", p, a,
    b, G.x, G.y);
117 
118         int n_A,n_B;
119 
120         printf("\nInput Alice's private key n_A: ");
121         scanf("%d", &n_A);
122 
123         Point P_A = Multiple(n_A, G, p, a);
124         printf("\nAlice's public key P_A is (%d,%d)\n", P_A.x, P_A.y);
125 
126         printf("\nInput Bob's private key n_B: ");
127         scanf("%d", &n_B);
128 
129         Point P_B = Multiple(n_B, G, p, a);
130         printf("\nBob's public key P_B is (%d,%d)\n", P_B.x, P_B.y);
131 
132         Point K_A=Multiple(n_A,P_B,p,a);
133         printf("\nAlice get the key K=(%d,%d)",K_A.x,K_A.y);
134 
135         Point K_B=Multiple(n_B,P_A,p,a);
136         printf("\nBob get the key K=(%d,%d)",K_B.x,K_B.y);
137 
138         printf("\n\n");
139     }
140 
141     int main()
142     {
143         int p,a,b;
144         Diff(p,a,b);
145     }
```

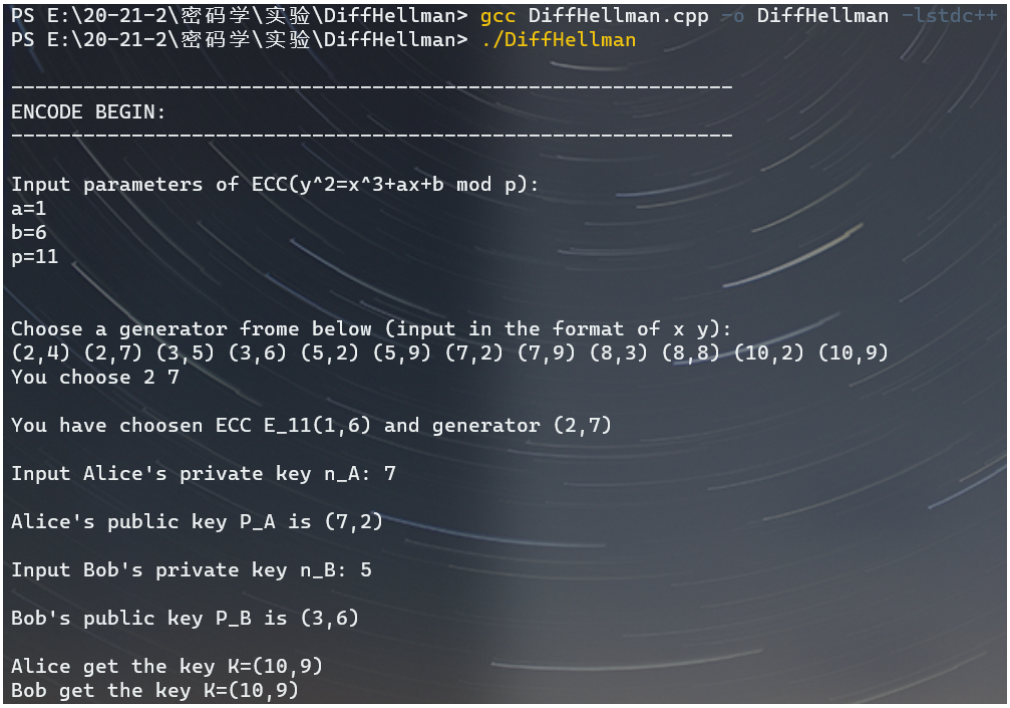
程序解释

算法描述：

- 取素数 $p \approx 2180$ 和参数 a、b，则得椭圆曲线上的点及无穷远点构成Abel群 $E_p(a,b)$
- 取 $E_p(a,b)$ 的某个生成元 $G=(x_1,y_1)$，G 的阶，即满足 $nG=O$ 的最小正整数 n很大。$E_p(a,b)$ 和 G 作为公开参数
- Alice 和 Bob 之间的密钥交换如下进行
    - Alice 随机选取保密的整数 $n_A<n$，计算 $P_A=n_AG$ 并发给 Bob
    - Bob 随机选取秘密的 n_B 并计算 $P_B=n_BG$ 发给 Alice
    - Alice 和 Bob 分别由 $K=n_AP_B$ 和 $K=n_BP_A$ 生成共享的密钥
      $K=n_AP_B=n_A(n_BG)=n_B(n_AG)=n_BP_A$

简单模拟即可。

运行截图



样例选取了椭圆曲线 $E_{11}(1,6)$，生成元 $g=(2,7)$

Alice 选取私钥为 7，Bob 选取私钥为 5

得到 Alice 公钥为 $(7,2)$，Bob 公钥为 $(3,6)$

两人经过密钥交换后都得到了，密钥 $(10,9)$

AES

程序

```
#include <iomanip>
#include <iostream>
using namespace std;
unsigned char SBox[] = {
    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};
unsigned char InvSBox[256] = {
    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};

unsigned char Key[11][4][4];

void SubBytes(unsigned char Text[][4])
{
```

```
    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Text[r][c] = SBox[Text[r][c]];
}

void ShiftRows(unsigned char Text[][4])
{
    unsigned char t[4];
    for (int r = 1; r < 4; r++)
    {
        for (int c = 0; c < 4; c++)
            t[c] = Text[r][(c + r) % 4];
        for (int c = 0; c < 4; c++)
            Text[r][c] = t[c];
    }
}

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

void MixColumns(unsigned char Text[][4])
{
    unsigned char t[4];
    for (int c = 0; c < 4; c++)
    {
        for (int r = 0; r < 4; r++)
            t[r] = Text[r][c];
        for (int r = 0; r < 4; r++)
            Text[r][c] = Multiply(0x02, t[r]) ^ Multiply(0x03, t[(r + 1) % 4]) ^
Multiply(0x01, t[(r + 2) % 4]) ^ Multiply(0x01, t[(r + 3) % 4]);
    }
}

void AddRoundKey(unsigned char Text[][4], unsigned char Key[][4])
{
    for (int c = 0; c < 4; c++)
        for (int r = 0; r < 4; r++)
            Text[r][c] ^= Key[r][c];
}

void KeyExpansion(unsigned char *key, unsigned char Key[][4][4])
{
    unsigned char rc[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36};
```

```
    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Key[0][r][c] = key[r + c * 4];
    for (int i = 1; i <= 10; i++)
        for (int j = 0; j < 4; j++)
        {
            unsigned char t[4];
            for (int r = 0; r < 4; r++)
                t[r] = j ? Key[i][r][j - 1] : Key[i - 1][r][3];
            if (j == 0)
            {
                unsigned char temp = t[0];
                for (int r = 0; r < 3; r++)
                    t[r] = SBox[t[(r + 1) % 4]];
                t[3] = SBox[temp];
                t[0] ^= rc[i - 1];
            }
            for (int r = 0; r < 4; r++)
                Key[i][r][j] = Key[i - 1][r][j] ^ t[r];
        }
}

void InvSubBytes(unsigned char Text[][4])
{
    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Text[r][c] = InvSBox[Text[r][c]];
}

void InvShiftRows(unsigned char Text[][4])
{
    unsigned char t[4];
    for (int r = 1; r < 4; r++)
    {
        for (int c = 0; c < 4; c++)
            t[c] = Text[r][(c - r + 4) % 4];
        for (int c = 0; c < 4; c++)
            Text[r][c] = t[c];
    }
}

void InvMixColumns(unsigned char Text[][4])
{
    unsigned char t[4];
    for (int c = 0; c < 4; c++)
    {
        for (int r = 0; r < 4; r++)
            t[r] = Text[r][c];
        for (int r = 0; r < 4; r++)
            Text[r][c] = Multiply(0x0e, t[r]) ^ Multiply(0x0b, t[(r + 1) % 4]) ^ Multiply(0x0d, t[(r + 2) % 4]) ^ Multiply(0x09, t[(r + 3) % 4]);
    }
}

unsigned char *Cipher(unsigned char *Input)
{
    unsigned char Text[4][4];

```

```
    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Text[r][c] = Input[c * 4 + r];

    AddRoundKey(Text, Key[0]);

    for (int i = 1; i <= 10; i++)
    {
        SubBytes(Text);
        ShiftRows(Text);
        if (i != 10)
            MixColumns(Text);
        AddRoundKey(Text, Key[i]);
    }

    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Input[c * 4 + r] = Text[r][c];

    return Input;
}

unsigned char *InvCipher(unsigned char *Input)
{
    unsigned char Text[4][4];

    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Text[r][c] = Input[c * 4 + r];

    AddRoundKey(Text, Key[10]);
    for (int i = 9; i >= 0; i--)
    {
        InvShiftRows(Text);
        InvSubBytes(Text);
        AddRoundKey(Text, Key[i]);
        if (i)
            InvMixColumns(Text);
    }

    for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
            Input[c * 4 + r] = Text[r][c];
    return Input;
}

int main()
{
    unsigned char Data[100001][4][4];
    string str;

    cout << "Input the initial key: ";
    cin >> str;
    if (str.length() != 16)
    {
        cout << "\nWrong! Length of the key should be 16!\n\nInput the initial
key: ";
        cin >> str;
```

```
    }

    for (int r = 0; r< 4; r++)
        for (int c = 0; c < 4; c++)
            Key[0][r][c] = str[r * 4 + c];

    cout << "\nInput your text: ";
    cin >> str;

    cout << "\nYour plain text in HEX is \n";
    int tmp = (str.length() % 16 == 0 ? 0 : 16 - str.length() % 16);
    for (int i = 1; i <= tmp; i++)
        str += ((char)(0));

    tmp = 0;
    for (int k = 1; k <= str.length() / 16; k++)
    {
        for (int r = 0; r < 4; r++)
            for (int c = 0; c < 4; c++)
            {
                Data[k][r][c] = str[tmp];
                tmp++;
            }
        for (int r = 0; r < 4; r++)
        {
            for (int c = 0; c < 4; c++)
            {
                cout << hex << setw(2) << setfill('0') << (int)(Data[k][r][c])
<< ' ';
            }
            cout << endl;
        }
    }
    cout << "\nThe cipher text in HEX is \n";
    for (int k = 1; k <= str.length() / 16; k++)
    {
        Cipher(*Data[k]);
        for (int r = 0; r < 4; r++)
        {
            for (int c = 0; c < 4; c++)
            {
                cout << hex << setw(2) << setfill('0') << (int)(Data[k][r][c])
<< ' ';
            }
            cout << endl;
        }
    }
    cout << "\nThe plain text in HEX is \n";
    for (int k = 1; k <= str.length() / 16; k++)
    {
        InvCipher(*Data[k]);
        for (int r = 0; r < 4; r++)
        {
            for (int c = 0; c < 4; c++)
            {
                cout << hex << setw(2) << setfill('0') << (int)(Data[k][r][c])
<< ' ';
            }
```

```
268                 cout << endl;
269             }
270         }
271     }
```

程序解释

```
1   unsigned char SBox[] = {
2       0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, ...};
3   unsigned char InvSBox[256] = {
4       0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, ...};
```

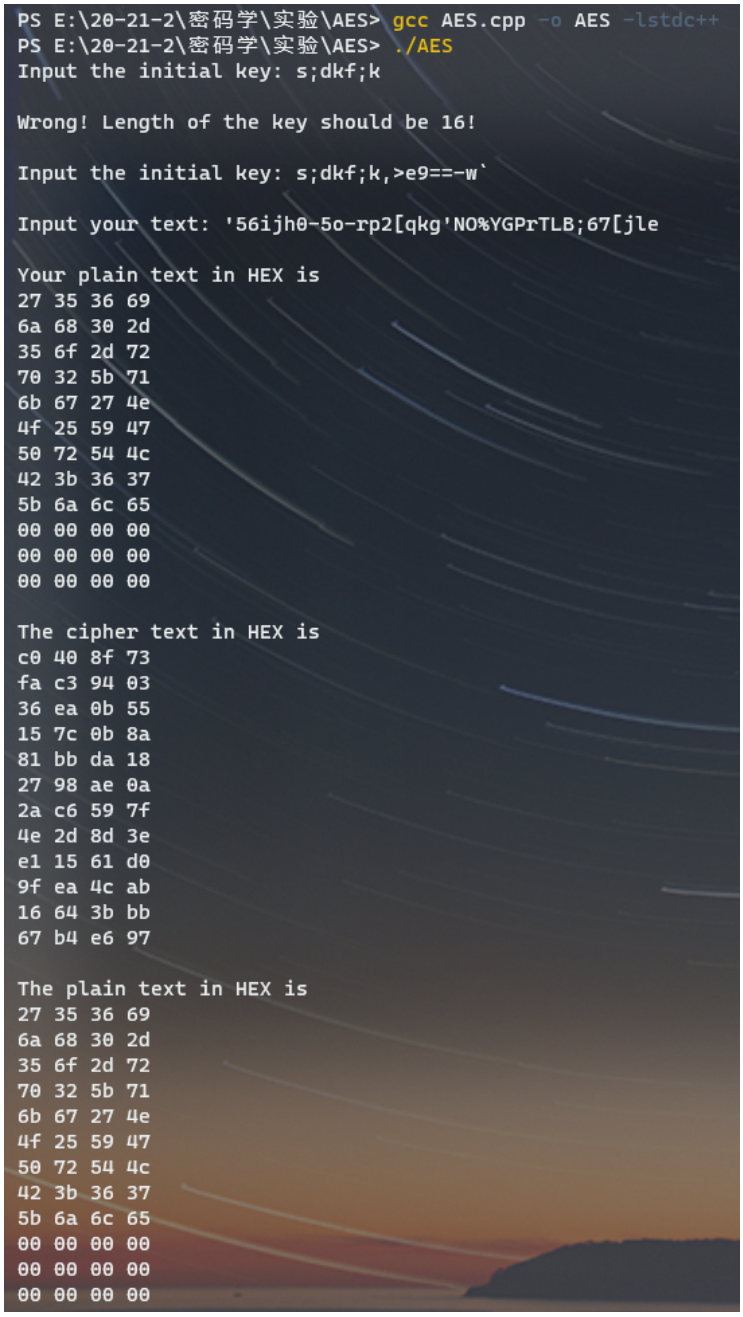
S 盒和逆 S 盒是给定的

```
1   unsigned char Multiply(unsigned char, unsigned char)
```

用来计算 GF 域下的乘法

对应过程	加密函数	解密函数
SubBytes	`void SubBytes(unsigned char)`	`void InvSubBytes(unsigned char)`
ShiftRows	`void ShiftRows(unsigned char)`	`void InvShiftRows(unsigned char)`
MixColumns	`void MixColumns(unsigned char)`	`void InvMixColumns(unsigned char)`
AddRoundKey	`void AddRoundKey(unsigned char, unsigned char)`	无
Encode&Decode	`unsigned char *Cipher(unsigned char *)`	`unsigned char *InvCipher(unsigned char *)`

运行截图



- 密钥为 16 字节（128 位）
- 要加密的数据理论上可以无限长，文本可以为任意类型
- 16 字节一组进行加密，采用 ECB 模式，不足位补 0x00
- 输出采用 16 进制

exgcd

程序

```cpp
#include <cstdio>
using namespace std;

void ex_gcd(long long a, long long b, long long &x, long long &y)
{
    if (b == 0)
    {
        x = 1;
```

```
        y = 0;
        return;
    }
    ex_gcd(b, a % b, x, y);
    long long t = x;
    x = y;
    y = t - a / b * y;
    return;
}

long long gcd(long long a, long long b)
{
    if (a % b == 0)
        return b;
    return gcd(b, a % b);
}

long long Inverse(long long a, long long p)
{
    long long x, y;
    ex_gcd(a, p, x, y);
    x = x + (x < 0 ? p : 0);
    return x;
}

int main()
{
    printf("Input 2 integers: ");
    long long x, y, ans;
    scanf("%lld%lld", &x, &y);
    ans = gcd(x, y);
    printf("GCD is %lld\n\n", ans);

    printf("Input an integer and mod: ");
    scanf("%lld%lld", &x, &y);
    ans = Inverse(x, y);
    printf("Inverse is %lld\n", ans);
}
```

程序解释

```
void ex_gcd(long long a, long long b, long long &x, long long &y)
```

是扩展欧几里得算法，计算结果得到 $ax + by = 1$

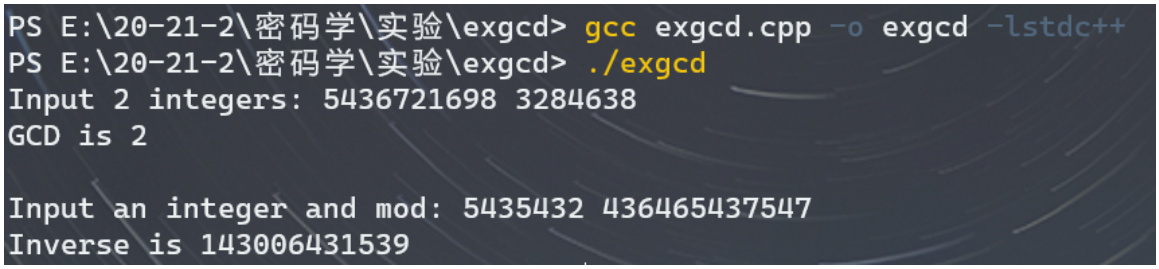
```
long long gcd(long long a, long long b)
```

为求最大公约数

```
long long Inverse(long long a, long long p)
```

为利用扩展欧几里得算法求模运算的逆

运行截图



- 随机选取 5436721698 和 3284638，得到最大公约数为 2
- 随机选取 $a=5435432, p=436465437547$，计算得到 $a^{-1} \equiv 143006431539 \pmod{p}$