

```
#include <stdlib.h>
```

```
#include <svdpi.h>
```

```
#include <stdio.h>
```

```
//using namespace std;
```

```
static const unsigned char Alogtable[] =
```

```
{
```

```
    1, 3, 5, 15, 17, 51, 85, 255, 26, 46, 114, 150, 161, 248, 19, 53,  
    95, 225, 56, 72, 216, 115, 149, 164, 247, 2, 6, 10, 30, 34, 102, 170,  
    229, 52, 92, 228, 55, 89, 235, 38, 106, 190, 217, 112, 144, 171, 230, 49,  
    83, 245, 4, 12, 20, 60, 68, 204, 79, 209, 104, 184, 211, 110, 178, 205,  
    76, 212, 103, 169, 224, 59, 77, 215, 98, 166, 241, 8, 24, 40, 120, 136,  
    131, 158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206, 73, 219, 118, 154,  
    181, 196, 87, 249, 16, 48, 80, 240, 11, 29, 39, 105, 187, 214, 97, 163,  
    254, 25, 43, 125, 135, 146, 173, 236, 47, 113, 147, 174, 233, 32, 96, 160,  
    251, 22, 58, 78, 210, 109, 183, 194, 93, 231, 50, 86, 250, 21, 63, 65,  
    195, 94, 226, 61, 71, 201, 64, 192, 91, 237, 44, 116, 156, 191, 218, 117,  
    159, 186, 213, 100, 172, 239, 42, 126, 130, 157, 188, 223, 122, 142, 137, 128,  
    155, 182, 193, 88, 232, 35, 101, 175, 234, 37, 111, 177, 200, 67, 197, 84,  
    252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203, 70, 202,  
    69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14,  
    18, 54, 90, 238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23,  
    57, 75, 221, 124, 132, 151, 162, 253, 28, 36, 108, 180, 199, 82, 246, 1,
```

```
};
```

```
static const unsigned char Logtable[] =
```

```
{
```

```
    0, 0, 25, 1, 50, 2, 26, 198, 75, 199, 27, 104, 51, 238, 223, 3,
```

```
100, 4, 224, 14, 52, 141, 129, 239, 76, 113, 8, 200, 248, 105, 28, 193,  
125, 194, 29, 181, 249, 185, 39, 106, 77, 228, 166, 114, 154, 201, 9, 120,  
101, 47, 138, 5, 33, 15, 225, 36, 18, 240, 130, 69, 53, 147, 218, 142,  
150, 143, 219, 189, 54, 208, 206, 148, 19, 92, 210, 241, 64, 70, 131, 56,  
102, 221, 253, 48, 191, 6, 139, 98, 179, 37, 226, 152, 34, 136, 145, 16,  
126, 110, 72, 195, 163, 182, 30, 66, 58, 107, 40, 84, 250, 133, 61, 186,  
43, 121, 10, 21, 155, 159, 94, 202, 78, 212, 172, 229, 243, 115, 167, 87,  
175, 88, 168, 80, 244, 234, 214, 116, 79, 174, 233, 213, 231, 230, 173, 232,  
44, 215, 117, 122, 235, 22, 11, 245, 89, 203, 95, 176, 156, 169, 81, 160,  
127, 12, 246, 111, 23, 196, 73, 236, 216, 67, 31, 45, 164, 118, 123, 183,  
204, 187, 62, 90, 251, 96, 177, 134, 59, 82, 161, 108, 170, 85, 41, 157,  
151, 178, 135, 144, 97, 190, 220, 252, 188, 149, 207, 205, 55, 63, 91, 209,  
83, 57, 132, 60, 65, 162, 109, 71, 20, 42, 158, 93, 86, 242, 211, 171,  
68, 17, 146, 217, 35, 32, 46, 137, 180, 124, 184, 38, 119, 153, 227, 165,  
103, 74, 237, 222, 197, 49, 254, 24, 13, 99, 140, 128, 192, 247, 112, 7,  
};
```

```
static const unsigned char Sen[] =
```

```
{  
    99, 124, 119, 123, 242, 107, 111, 197,  
    48, 1, 103, 43, 254, 215, 171, 118,  
    202, 130, 201, 125, 250, 89, 71, 240,  
    173, 212, 162, 175, 156, 164, 114, 192,  
    183, 253, 147, 38, 54, 63, 247, 204,  
    52, 165, 229, 241, 113, 216, 49, 21,  
    4, 199, 35, 195, 24, 150, 5, 154,  
    7, 18, 128, 226, 235, 39, 178, 117,  
    9, 131, 44, 26, 27, 110, 90, 160,  
    82, 59, 214, 179, 41, 227, 47, 132,
```

```
83, 209, 0, 237, 32, 252, 177, 91,  
106, 203, 190, 57, 74, 76, 88, 207,  
208, 239, 170, 251, 67, 77, 51, 133,  
69, 249, 2, 127, 80, 60, 159, 168,  
81, 163, 64, 143, 146, 157, 56, 245,  
188, 182, 218, 33, 16, 255, 243, 210,  
205, 12, 19, 236, 95, 151, 68, 23,  
196, 167, 126, 61, 100, 93, 25, 115,  
96, 129, 79, 220, 34, 42, 144, 136,  
70, 238, 184, 20, 222, 94, 11, 219,  
224, 50, 58, 10, 73, 6, 36, 92,  
194, 211, 172, 98, 145, 149, 228, 121,  
231, 200, 55, 109, 141, 213, 78, 169,  
108, 86, 244, 234, 101, 122, 174, 8,  
186, 120, 37, 46, 28, 166, 180, 198,  
232, 221, 116, 31, 75, 189, 139, 138,  
112, 62, 181, 102, 72, 3, 246, 14,  
97, 53, 87, 185, 134, 193, 29, 158,  
225, 248, 152, 17, 105, 217, 142, 148,  
155, 30, 135, 233, 206, 85, 40, 223,  
140, 161, 137, 13, 191, 230, 66, 104,  
65, 153, 45, 15, 176, 84, 187, 22  
};
```

```
static const unsigned char Sde[] =
```

```
{  
82, 9, 106, 213, 48, 54, 165, 56, 191, 64, 163, 158, 129, 243, 215, 251,  
124, 227, 57, 130, 155, 47, 255, 135, 52, 142, 67, 68, 196, 222, 233, 203,  
84, 123, 148, 50, 166, 194, 35, 61, 238, 76, 149, 11, 66, 250, 195, 78,
```

```

8, 46, 161, 102, 40, 217, 36, 178, 118, 91, 162, 73, 109, 139, 209, 37,
114, 248, 246, 100, 134, 104, 152, 22, 212, 164, 92, 204, 93, 101, 182, 146,
108, 112, 72, 80, 253, 237, 185, 218, 94, 21, 70, 87, 167, 141, 157, 132,
144, 216, 171, 0, 140, 188, 211, 10, 247, 228, 88, 5, 184, 179, 69, 6,
208, 44, 30, 143, 202, 63, 15, 2, 193, 175, 189, 3, 1, 19, 138, 107,
58, 145, 17, 65, 79, 103, 220, 234, 151, 242, 207, 206, 240, 180, 230, 115,
150, 172, 116, 34, 231, 173, 53, 133, 226, 249, 55, 232, 28, 117, 223, 110,
71, 241, 26, 113, 29, 41, 197, 137, 111, 183, 98, 14, 170, 24, 190, 27,
252, 86, 62, 75, 198, 210, 121, 32, 154, 219, 192, 254, 120, 205, 90, 244,
31, 221, 168, 51, 136, 7, 199, 49, 177, 18, 16, 89, 39, 128, 236, 95,
96, 81, 127, 169, 25, 181, 74, 13, 45, 229, 122, 159, 147, 201, 156, 239,
160, 224, 59, 77, 174, 42, 245, 176, 200, 235, 187, 60, 131, 83, 153, 97,
23, 43, 4, 126, 186, 119, 214, 38, 225, 105, 20, 99, 85, 33, 12, 125,
};

```

```

void decrypt_aes(const int num,unsigned char *block, unsigned char *key);

```

```

void encrypt_aes(const int num,unsigned char *block, unsigned char *key);

```

```

extern "C" void AES_GOLDEN_MODEL(const int num, const int decrypt_i, const svBit* key_i, const svBit*
data_i, svBit* data_o) {

```

```

    unsigned char aes_key[16], aes_data[16];

```

```

    for(int i=0;i<16;i++)    aes_key[15-i] = key_i[i];

```

```

    for(int j=0;j<16;j++)    aes_data[15-j] = data_i[j];

```

```

    if (decrypt_i == 0)

```

```

        encrypt_aes(num, aes_data, aes_key);

```

```

else

    decrypt_aes(num, aes_data, aes_key);

for(int k=0;k<16;k++)    data_o[k] = aes_data[15-k];

};

```

```

void switchblock(unsigned char *block)
{
    int i;
    unsigned char *aux;
    aux = (unsigned char *)malloc(16 * sizeof(char));

    *(aux) = *(block);
    *(aux + 1) = *(block + 4);
    *(aux + 2) = *(block + 8);
    *(aux + 3) = *(block + 12);
    *(aux + 4) = *(block + 1);
    *(aux + 5) = *(block + 5);
    *(aux + 6) = *(block + 9);
    *(aux + 7) = *(block + 13);
    *(aux + 8) = *(block + 2);
    *(aux + 9) = *(block + 6);
    *(aux + 10) = *(block + 10);
    *(aux + 11) = *(block + 14);
    *(aux + 12) = *(block + 3);
    *(aux + 13) = *(block + 7);
    *(aux + 14) = *(block + 11);
    *(aux + 15) = *(block + 15);
}

```

```
    for (i = 0; i < 16; i++)  
        *(block + i) = *(aux + i);  
  
}
```

```
void RotWord(unsigned char *a)  
{  
    unsigned char aux;  
    aux = *(a + 0);  
    *(a + 0) = *(a + 1);  
    *(a + 1) = *(a + 2);  
    *(a + 2) = *(a + 3);  
    *(a + 3) = aux;  
}
```

```
void KeySchedule(unsigned char *key, unsigned char *rcon)  
{  
    unsigned char aux1[4];  
    unsigned char aux2[4];  
    unsigned char aux3[4];  
    int i = 0;  
    for (i = 0; i < 4; i++)  
    {  
        aux1[i] = *(key + i * 4 + 3);  
        aux2[i] = *(key + i * 4);  
    }  
    RotWord((unsigned char *)aux1);
```

```
//SubBytes
```

```
    for (i = 0; i < 4; i++)
    {
        aux1[i] = Sen[aux1[i]];
    }
    for (i = 0; i < 4; i++)
    {
        aux3[i] = aux2[i] ^ aux1[i] ^ *(rcon + i);
        *(key + i * 4) = aux3[i];
    }

    for (i = 0; i < 4; i++)
    {
        aux3[i] = *(key + i * 4 + 1) ^ aux3[i];
        *(key + i * 4 + 1) = aux3[i];
    }

    for (i = 0; i < 4; i++)
    {
        aux3[i] = *(key + i * 4 + 2) ^ aux3[i];
        *(key + i * 4 + 2) = aux3[i];
    }

    for (i = 0; i < 4; i++)
    {
        aux3[i] = *(key + i * 4 + 3) ^ aux3[i];
        *(key + i * 4 + 3) = aux3[i];
    }
```

```
}
```

```
unsigned char mul(unsigned char a, unsigned char b)
```

```
{
```

```
    if (a && b)
```

```
        return Alogtable[(Logtable[a] + Logtable[b])%255];
```

```
    else
```

```
        return 0;
```

```
}
```

```
void MixColumn(unsigned char *state)
```

```
{
```

```
    int j = 0;
```

```
    unsigned char aux = 0;
```

```
    unsigned char aux_vector[16];
```

```
    for (j = 0; j < 4; j++)
```

```
    {
```

```
        aux = mul(0x2, *(state + j)) ^ mul(0x3, *(state + j + 4)) ^ *(state + j + 8) ^ *(state + j + 12);
```

```
        aux_vector[j] = aux;
```

```
        aux = *(state + j) ^ mul(0x2, *(state + j + 4)) ^ mul(0x3, *(state + j + 8)) ^ *(state + j + 12);
```

```
        aux_vector[j + 4] = aux;
```

```
        aux = *(state + j) ^ *(state + j + 4) ^ mul(0x2, *(state + j + 8)) ^ mul(0x3, *(state + j + 12));
```

```
        aux_vector[j + 8] = aux;
```

```
        aux = mul(0x3, *(state + j)) ^ *(state + j + 4) ^ *(state + j + 8) ^ mul(0x2, *(state + j + 12));
```

```
        aux_vector[j + 12] = aux;
```

```
    }
```

```
    for (j = 0; j < 16; j++)
```

```
        *(state + j) = aux_vector[j];
```



```
}
```

```
void InvMixColumn(unsigned char *state)
```

```
{
```

```
    int j = 0;
```

```
    unsigned char aux = 0;
```

```
    unsigned char aux_vector[16];
```

```
    for (j = 0; j < 4; j++)
```

```
    {
```

```
        aux = mul(0x0e, *(state + j)) ^ mul(0x0b, *(state + j + 4)) ^ mul(0x0d, *(state + j + 8)) ^  
mul(0x09, *(state + j + 12));
```

```
        aux_vector[j] = aux;
```

```
        aux = mul(0x09, *(state + j)) ^ mul(0x0e, *(state + j + 4)) ^ mul(0x0b, *(state + j + 8)) ^  
mul(0x0d, *(state + j + 12));
```

```
        aux_vector[j + 4] = aux;
```

```
        aux = mul(0x0d, *(state + j)) ^ mul(0x09, *(state + j + 4)) ^ mul(0x0e, *(state + j + 8)) ^  
mul(0x0b, *(state + j + 12));
```

```
        aux_vector[j + 8] = aux;
```

```
        aux = mul(0x0b, *(state + j)) ^ mul(0x0d, *(state + j + 4)) ^ mul(0x09, *(state + j + 8)) ^  
mul(0x0e, *(state + j + 12));
```

```
        aux_vector[j + 12] = aux;
```

```
    }
```

```
    for (j = 0; j < 16; j++)
```

```
        *(state + j) = aux_vector[j];
```

```
}
```

```
void AddRoundKey(unsigned char *state, unsigned char *key)
```

```
{
```

```
    int i = 0;
```

```

    for (i = 0; i < 16; i++)
    {
        *(state + i) ^= *(key + i);
        //printf("M: %u\n", *(key + i));
    }
}

```

```

void SubBytes(unsigned char *state)
{
    int i = 0;
    for (i = 0; i < 16; i++)
    {
        *(state + i) = Sen[*(state + i)];
    }
}

```

```

void InvSubBytes(unsigned char *state)
{
    int i = 0;
    for (i = 0; i < 16; i++)
    {
        *(state + i) = Sde[*(state + i)];
    }
}

```

```

void ShiftRows(unsigned char *state)
{
    unsigned char AUX[16];

```

```

int i = 0;
for (i = 0; i < 16; i++)
{
    AUX[i] = *(state + i);
}

*(state + 4) = AUX[5];
*(state + 5) = AUX[6];
*(state + 6) = AUX[7];
*(state + 7) = AUX[4];

*(state + 8) = AUX[10];
*(state + 9) = AUX[11];
*(state + 10) = AUX[8];
*(state + 11) = AUX[9];

*(state + 12) = AUX[15];
*(state + 13) = AUX[12];
*(state + 14) = AUX[13];
*(state + 15) = AUX[14];
}

```

```

void InvShiftRows(unsigned char *state)
{
    unsigned char AUX[16];
    int i = 0;
    for (i = 0; i < 16; i++)
    {
        AUX[i] = *(state + i);
    }
}

```

```

    }

    *(state + 4) = AUX[7];
    *(state + 5) = AUX[4];
    *(state + 6) = AUX[5];
    *(state + 7) = AUX[6];

    *(state + 8) = AUX[10];
    *(state + 9) = AUX[11];
    *(state + 10) = AUX[8];
    *(state + 11) = AUX[9];

    *(state + 12) = AUX[13];
    *(state + 13) = AUX[14];
    *(state + 14) = AUX[15];
    *(state + 15) = AUX[12];
}

```

```

unsigned char * KeyExpand(unsigned char *key)
{
    int i = 0;
    int j = 0;

    unsigned char aux[10] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};
    unsigned char rcon[4] = {0x00, 0x00, 0x00, 0x00};

    unsigned char auxmalloc[1000];
    unsigned char *expandedkey;
    expandedkey = (unsigned char *)auxmalloc;

```

```

    for (i = 0; i < 16; i++)
        *(expandedkey + i) = *(key + i);

    for (i = 0; i < 10; i++)
    {
        rcon[0] = aux[i];
        KeySchedule((unsigned char *)key, (unsigned char *)rcon);
        for (j = 0; j < 16; j++)
            *(expandedkey + 16 * (i + 1) + j) = *(key + j);
    }
    return expandedkey;
}

void encrypt_aes(const int num, unsigned char *block, unsigned char *key)
{
    int i = 0;

    switchblock(block);
    switchblock(key);

    unsigned char *expandedkey;
    expandedkey = KeyExpand((unsigned char *)key);

    AddRoundKey((unsigned char *)block, (unsigned char *)expandedkey);

```

```

for (i = 0; i < num; i++)
{
    SubBytes((unsigned char *)block);
    ShiftRows((unsigned char *)block);
    if (i != 9)
        MixColumn((unsigned char *)block);
    AddRoundKey((unsigned char *)block, (unsigned char *)(expandedkey + 16 * (i + 1)));
}

switchblock(block);
}

```

```

void decrypt_aes(const int num, unsigned char *block, unsigned char *key)

```

```

{
    int i = 0;
    unsigned char *block_aux;

    switchblock(block);
    switchblock(key);

    unsigned char *expandedkey;
    expandedkey = KeyExpand((unsigned char *)key);

    AddRoundKey((unsigned char *)block, (unsigned char *)(expandedkey + 16 * 10));

    for (i = 10; i > (10 - num); i--)
    {
        if (i != 10)

```

```
        InvMixColum((unsigned char *)block);
        InvShiftRows((unsigned char *)block);
        InvSubBytes((unsigned char *)block);
        AddRoundKey((unsigned char *)block, (unsigned char *)(expandedkey + 16 * (i - 1)));
    }
    switchblock(block);
}
```