Project1 SM4 软件实现和优化

学院 网络空间安全

专业 网络空间安全

学号 202200460149

班级姓名 网安 22.1 班张弛

2025 年 8 月 7 日

目录

1 实验任务 1

2 SM4 软件实现 1

2.1 明文处理 1

2.2 轮操作 1

2.3 密钥扩展算法 2

2.4 详细代码 3

2.5 实现结果 9

3 SM4 算法优化 9

3.1 优化方法 1：查表优化 9

3.2 优化方法 2：SIMD 指令集优化 26

3.3 优化方法 3：AES-NI 优化 27

3.4 时间对比 28

4 SM4-GCM 工作模式优化 30

1　实验任务

Project 1: 做 SM4 的软件实现和优化

2　SM4 软件实现

首先是 SM4 的过程实现，我们根据下图流程来进行实现：

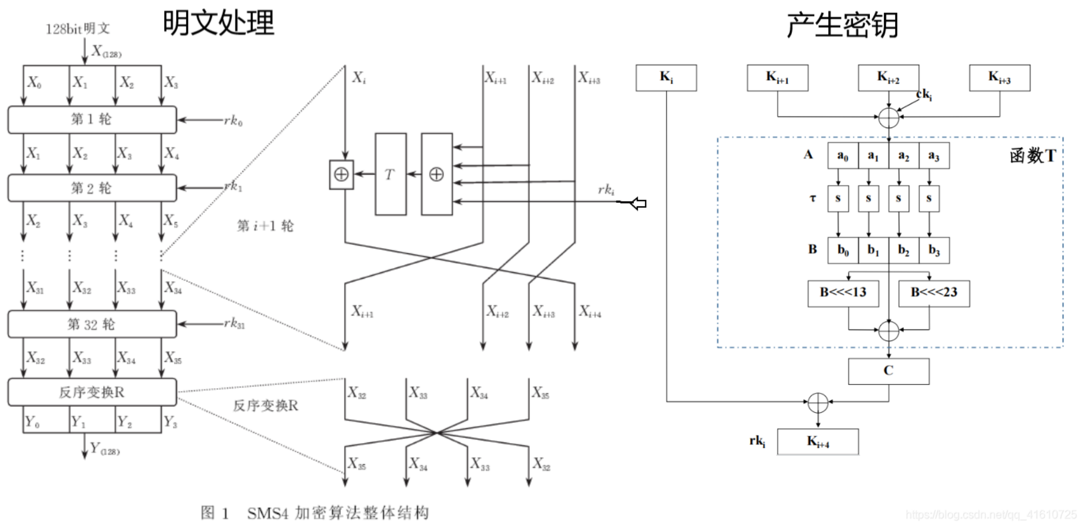

明文处理
128bit明文
$X_{(128)}$
X_0 X_1 X_2 X_3
第 1 轮
rk_0
第 2 轮
rk_1
第 i+1 轮
第 32 轮
rk_{31}
反序变换R
Y_0 Y_1 Y_2 Y_3
$Y_{(128)}$
X_i X_{i+1} X_{i+2} X_{i+3}
T
rk_i
X_{i+1} X_{i+2} X_{i+3} X_{i+4}
反序变换R
X_{32} X_{33} X_{34} X_{35}
X_{35} X_{34} X_{33} X_{32}
产生密钥
K_i K_{i+1} K_{i+2} K_{i+3}
ck_i
函数T
A a_0 a_1 a_2 a_3
τ s s s s
B b_0 b_1 b_2 b_3
B<<<13
B<<<23
C
rk_i K_{i+4}
图 1　SMS4 加密算法整体结构

图 1 SM4 过程

2.1　明文处理

明文处理大致分解为 3 步：

1）将 128bit 的明文分成 4 个 32bit 的字 X1,X2,X3,X4。

2）将上述得到的字进行 32 轮的轮操作。

3）最后将进行过 32 轮操作的 4 个字进行反序变换后组成 128bit 的密文。

2.2　轮操作

将明文拆分后的 4 个字的后 3 个字与该轮的子密钥进行异或处理，之后再经过一个函数 T（将得到的 32bit 的 A 分成 4 部分，每部分 8bit 分别过 s 盒，得到 B 的 4 个部分，分别左移 2 位，10 位，18 位及 24 位，将这四个部分进行异或处理）得到 32bit 的 C，之后再将明文拆分后的第一个字与 C 进行异或。

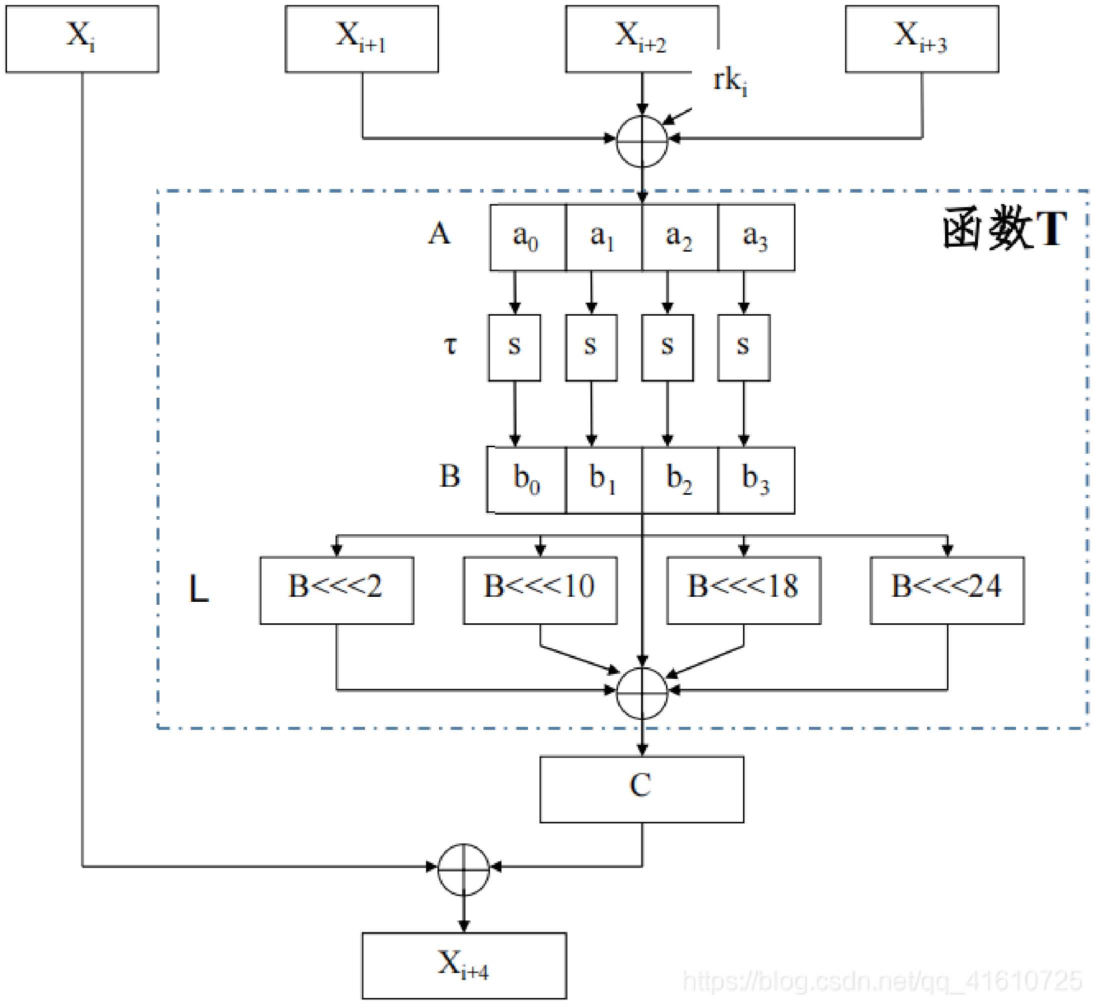


图 2 轮操作

2.3 密钥扩展算法

记加密密钥为 MK，长度为 128 比特，将其分为四项，其中每一项都为为 32 位的字，表示为 MK0、MK1.MK2.MK3。

系统参数为 FK。长度为 128 比特，将其分为四项，其中每一项都为 32 位的字。表示为 FK0,FK1,FK2,FK3.

固定参数为 CK，用于密钥扩展算法。其中每一项都为 32 位的字。表示为 CK0 到 CK31 共 32 项。

轮密钥，其中每一项都为 32 位的字。轮密钥由加密密钥通过密钥扩展算法生成。记为 rk0 到 rk31.

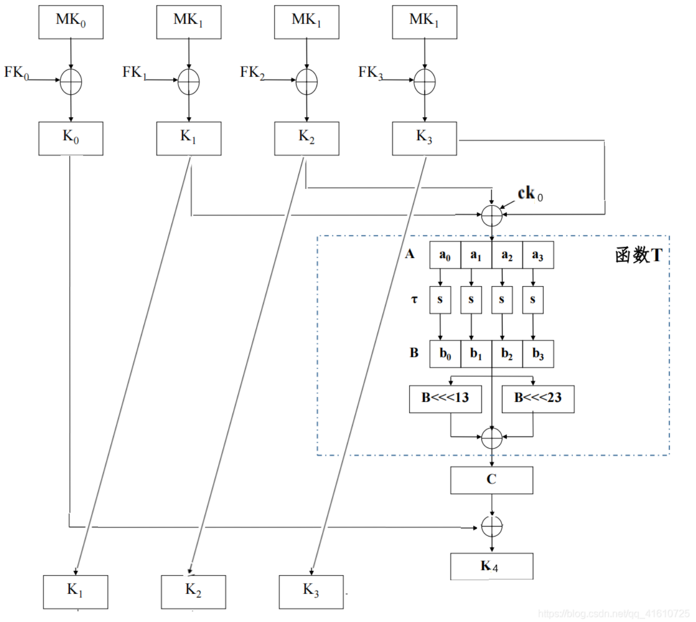


图 3 初始密钥拓展

首先密钥与系统参数的各部分异或，接着利用如下公式不断获取轮密钥：

$rk_i = K_{i+4} = K_i \oplus T(K_{i+1} \oplus K_{i+2} \oplus K_{i+3} \oplus CK_i)$

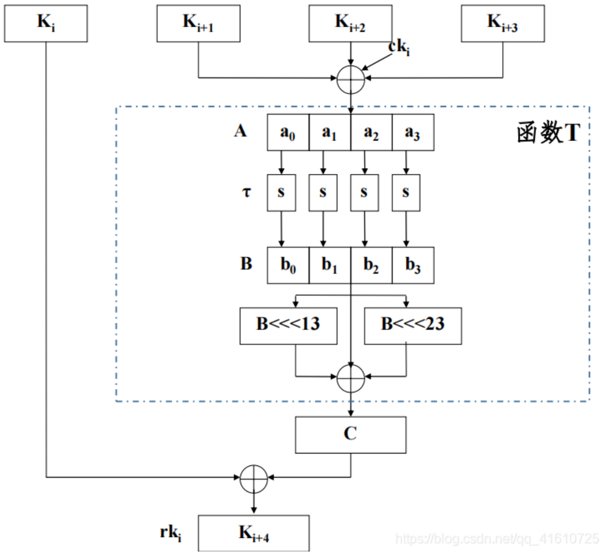


图 4 后续密钥拓展操作

2.4 详细代码

在上述知识基础上完成了代码上的加解密实现，还测试了 cbc 模式和 ecb 模式：

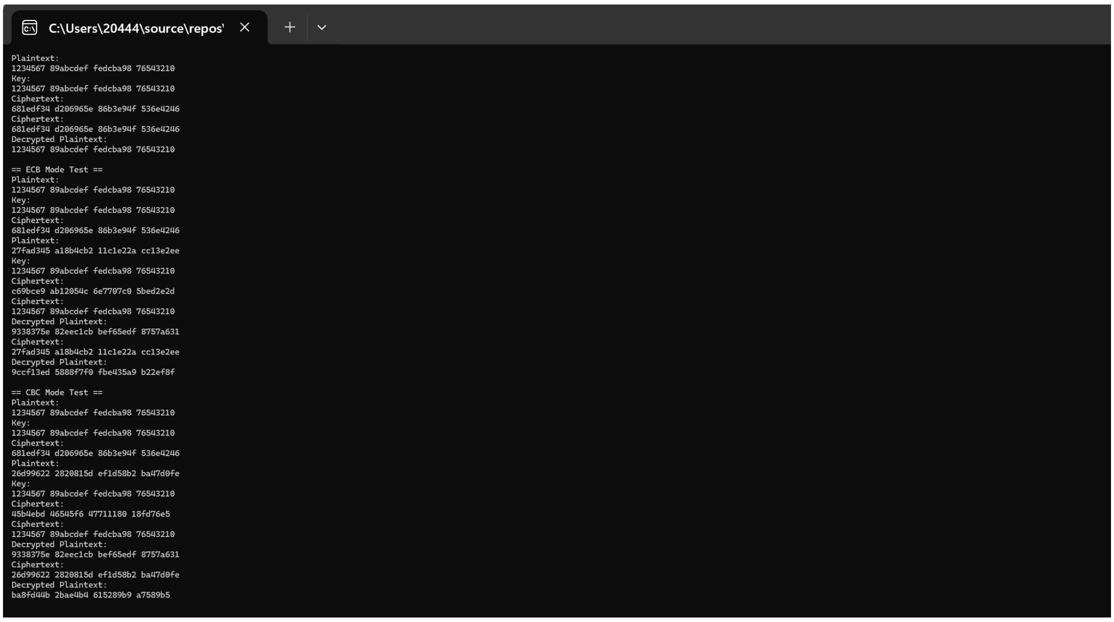
```
#include<iostream>
```

```
using namespace std;

//Round = 32轮数

//S盒
static const unsigned long SboxTable[16][16] = {
    {0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2, 0x28, 0xfb, 0x2c, 0x05},
    {0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99},
    {0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43, 0xed, 0xcf, 0xac, 0x62},
    {0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa, 0x75, 0x8f, 0x3f, 0xa6},
    {0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19, 0xe6, 0x85, 0x4f, 0xa8},
    {0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b, 0x70, 0x56, 0x9d, 0x35},
    {0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b, 0x01, 0x21, 0x78, 0x87},
    {0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7, 0xa0, 0xc4, 0xc8, 0x9e},
    {0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce, 0xf9, 0x61, 0x15, 0xa1},
    {0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30, 0xf5, 0x8c, 0xb1, 0xe3},
    {0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab, 0x0d, 0x53, 0x4e, 0x6f},
    {0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72, 0x6d, 0x6c, 0x5b, 0x51},
    {0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41, 0x1f, 0x10, 0x5a, 0xd8},
    {0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12, 0xb8, 0xe5, 0xb4, 0xb0},
    {0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09, 0xc5, 0x6e, 0xc6, 0x84},
    {0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e, 0xd7, 0xcb, 0x39, 0x48}
};

unsigned long sm4Sbox(unsigned long in) {
    return SboxTable[(in >> 4) & 0x0F][in & 0x0F];
}
```

```

//线性变换L
unsigned long L(unsigned long x) {
    return x ^ (x << 2 | x >> (32 - 2)) ^ (x << 10 | x >> (32 - 10)) ^ (x << 18 | x >> (32 - 18)) ^ (x << 24 | x >> (32 - 24));
}

//非线性T变换
unsigned long T(unsigned long x) {
    unsigned long b = 0;
    for (int i = 0; i < 4; i++) {
        b = (b << 8) | sm4Sbox((x >> ((3 - i) * 8)) & 0xFF);
    }
    return L(b);
}

//系统参数
static const unsigned long FK[4] = { 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc };

//固定参数 CK
static const unsigned long CK[32] = {
    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
};

//密钥扩展
void key_expansion(unsigned long MK[4], unsigned long rk[32]) {
    unsigned long K[36];
    for (int i = 0; i < 4; i++)
        K[i] = MK[i] ^ FK[i];

    for (int i = 0; i < 32; i++) {
        unsigned long tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];
        unsigned long b = 0;
        for (int j = 0; j < 4; j++)
            b = (b << 8) | sm4Sbox((tmp >> ((3 - j) * 8)) & 0xFF);
        unsigned long L = b ^ (b << 13 | b >> (32 - 13)) ^ (b << 23 | b >> (32 - 23));
        rk[i] = K[i] ^ L;
        K[i + 4] = rk[i];
    }
}

//轮操作
unsigned long round_operate(int i, unsigned long* X, unsigned long* rk) {
    return X[i] ^ T(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i]);
}

//加密函数
void sm4_enc(unsigned long MK[4], unsigned long X[4]) {
    cout << hex;
    cout << "Plaintext:" << endl;
    cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;

    cout << hex;
    cout << "Key:" << endl;
    cout << MK[0] << " " << MK[1] << " " << MK[2] << " " << MK[3] << endl;

    unsigned long rk[32];
    key_expansion(MK, rk);

    for (int i = 0; i < 32; i++) {
        unsigned long tmp = round_operate(i, X, rk);
        X[4 + i] = tmp;
    }

    cout << hex;
    cout << "Ciphertext:" << endl;
    cout << X[35] << " " << X[34] << " " << X[33] << " " << X[32] << endl;
}

//解密函数
void sm4_dec(unsigned long MK[4], unsigned long X[4]) {
    cout << hex;
    cout << "Ciphertext:" << endl;
    cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;

    unsigned long rk[32];
    key_expansion(MK, rk);

    //反转轮密钥
    for (int i = 0; i < 16; i++) swap(rk[i], rk[31 - i]);

    unsigned long tmpX[36] = { 0 };
    for (int i = 0; i < 4; i++) tmpX[i] = X[i];

    for (int i = 0; i < 32; i++) {
        tmpX[i + 4] = round_operate(i, tmpX, rk);
    }

    cout << "Decrypted Plaintext:" << endl;
    cout << tmpX[35] << " " << tmpX[34] << " " << tmpX[33] << " " << tmpX[32] << endl;
}

void copy_block(unsigned long* dst, unsigned long* src) {
    for (int i = 0; i < 4; i++) dst[i] = src[i];
}

void sm4_ecb_enc(unsigned long MK[4], unsigned long* data, int blocks) {
    for (int i = 0; i < blocks; i++) {
        sm4_enc(MK, &data[i * 4]);
    }
}

void sm4_ecb_dec(unsigned long MK[4], unsigned long* data, int blocks) {
    for (int i = 0; i < blocks; i++) {
        sm4_dec(MK, &data[i * 4]);
    }
}

void xor_block(unsigned long* dst, unsigned long* src) {
    for (int i = 0; i < 4; i++) dst[i] ^= src[i];
}

void sm4_cbc_enc(unsigned long MK[4], unsigned long* data, int blocks, unsigned long IV[4]) {
    unsigned long last_block[4];
    copy_block(last_block, IV);

    for (int i = 0; i < blocks; i++) {
        xor_block(&data[i * 4], last_block);
        sm4_enc(MK, &data[i * 4]);
        copy_block(last_block, &data[i * 4]);
    }
}

void sm4_cbc_dec(unsigned long MK[4], unsigned long* data, int blocks, unsigned long IV[4]) {
    unsigned long last_block[4];
    copy_block(last_block, IV);

    for (int i = 0; i < blocks; i++) {
        unsigned long tmp[4];
        copy_block(tmp, &data[i * 4]);

        sm4_dec(MK, &data[i * 4]);
        xor_block(&data[i * 4], last_block);

        copy_block(last_block, tmp);
    }
}

int main() {
    unsigned long MK[4] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210 };//加密密钥
    unsigned long X[36] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210 };//明文
    unsigned long C[36] = { 0x681edf34, 0xd206965e, 0x86b3e94f, 0x536e4246 };//密文
    sm4_enc(MK, X);
    sm4_dec(MK, C);

    cout << "\n== ECB Mode Test ==" << endl;
    unsigned long ecb_data[8] = {
        0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,
        0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff
    };
    sm4_ecb_enc(MK, ecb_data, 2);
    sm4_ecb_dec(MK, ecb_data, 2);

    cout << "\n== CBC Mode Test ==" << endl;
    unsigned long cbc_data[8] = {
        0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,
        0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff
    };
    unsigned long IV[4] = { 0x00000000, 0x00000000, 0x00000000, 0x00000000
        };
    sm4_cbc_enc(MK, cbc_data, 2, IV);
    sm4_cbc_dec(MK, cbc_data, 2, IV);

    return 0;
}
```

2.5 实现结果

运行结果如下图所示：



```
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210

== ECB Mode Test ==
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Plaintext:
27fad345 a18b4cb2 11c1e22a cc13e2ee
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
c69bce9 ab12054c 6e7707c0 5bed2e2d
Ciphertext:
1234567 89abcdef fedcba98 76543210
Decrypted Plaintext:
9338375e 82eec1cb bef65edf 8757a631
Ciphertext:
27fad345 a18b4cb2 11c1e22a cc13e2ee
Decrypted Plaintext:
9ccf13ed 5888f7f0 fbe435a9 b22ef8f

== CBC Mode Test ==
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Plaintext:
26d99622 2820815d ef1d58b2 ba47d0fe
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
45b4ebd 46545f6 47711180 18fd76e5
Ciphertext:
1234567 89abcdef fedcba98 76543210
Decrypted Plaintext:
9338375e 82eec1cb bef65edf 8757a631
Ciphertext:
26d99622 2820815d ef1d58b2 ba47d0fe
Decrypted Plaintext:
ba8fd44b 2bae4b4 615289b9 a7589b5
```

图 5 运行结果

3 SM4 算法优化

3.1 优化方法 1：查表优化

由于查表会消耗较多时间，我们考虑优化查表：

S 盒操作为 $x_0, x_1, x_2, x_3 \to S(x_0), S(x_1), S(x_2), S(x_3)$，其中 xi 为 8bit

为了提升效率，可将 S 盒与后续的循环移位变换 L 合并，即

$$L(S(x_0), S(x_1), S(x_2), S(x_3)) = L(S(x_0) \ll 24) \oplus L(S(x_1) \ll 16) \oplus L(S(x_2) \ll 8) \oplus L(S(x_3))$$

可定义 4 个 8bit → 32bit 查找表 T_i

$$T_0(x) = L(S(x) \ll 24)$$

$$T_1(x) = L(S(x) \ll 16)$$

$$T_2(x) = L(S(x) \ll 8)$$

$$T_3(x) = L(S(x))$$

节省后续的循环移位操作，大致操作如下：
1. 通过移位取出 x_0, x_1, x_2, x_3 2. 返回 $T_0(x_0) \oplus T_1(x_1) \oplus T_2(x_2) \oplus T_3(x_3)$
流程大致如下：输入明文 P，轮密钥 K_r，输出密文 C

```
1. $X_0, X_1, X_2, X_3 \leftarrow P$
2. for $i = 0 \to 31$
3.     $K_i \leftarrow$ 轮密钥
4.     $Tmp \leftarrow X_1 \oplus X_2 \oplus X_3 \oplus K_i$
5.     $Tmp \leftarrow T(Tmp) \oplus X_0$
6.     $X_0, X_1, X_2, X_3 \leftarrow X_1, X_2, X_3, Tmp$
7. $C \leftarrow X_3, X_2, X_1, X_0$
```

综上所述,我们可以首先生成4个T表,后续使用公式:$T(x) = T0[x0]^T1[x1]^T2[x2]^T3[x3]$ 进行查表，通过上述思路生成四个 T 表如下所示。

```
T0:
8ed55b5b        d0924242        4deaa7a7        06fdfbfb        fccf3333
          65e28787        c93df4f4        6bb5dede
4e165858        6eb4dada        44145050        cac10b0b        8828a0a0
          17f8efef        9c2cb0b0        11051414
872bacac        fb669d9d        f2986a6a        ae77d9d9        822aa8a8
          46bcfafa        14041010        cfc00f0f
02a8aaaa        54451111        5f134c4c        be269898        6d482525
          9e841a1a        1e061818        fd9b6666
ec9e7272        4a430909        10514141        24f7d3d3        d5934646
          53ecbfbf        f89a6262        927be9e9
```

```
ff33cccc          04555151          270b2c2c          4f420d0d          59eeb7b7
            f3cc3f3f          1caeb2b2          ea638989
74e79393          7fb1cece          6c1c7070          0daba6a6          edca2727
            28082020          48eba3a3          c1975656
80820202          a3dc7f7f          c4965252          12f9ebeb          a174d5d5
            b38d3e3e          c33ffcfc          3ea49a9a
5b461d1d          1b071c1c          3ba59e9e          0cfff3f3          3ff0cfcf
            bf72cdcd          4b175c5c          52b8eaea
8f810e0e          3d586565          cc3cf0f0          7d196464          7ee59b9b
            91871616          734e3d3d          08aaa2a2
c869a1a1          c76aadad          85830606          7ab0caca          b570c5c5
            f4659191          b2d96b6b          a7892e2e
18fbe3e3          47e8afaf          330f3c3c          674a2d2d          b071c1c1
            0e575959          e99f7676          e135d4d4
661e7878          b4249090          360e3838          265f7979          ef628d8d
            38596161          95d24747          2aa08a8a
b1259494          aa228888          8c7df1f1          d73becec          05010404
            a5218484          9879e1e1          9b851e1e
84d75353          00000000          5e471919          0b565d5d          e39d7e7e
            9fd04f4f          bb279c9c          1a534949
7c4d3131          ee36d8d8          0a020808          7be49f9f          20a28282
            d4c71313          e8cb2323          e69c7a7a
42e9abab          43bdfefe          a2882a2a          9ad14b4b          40410101
            dbc41f1f          d838e0e0          61b7d6d6
2fa18e8e          2bf4dfdf          3af1cbcb          f6cd3b3b          1dfae7e7
            e5608585          41155454          25a38686
60e38383          16acbaba          295c7575          34a69292          f7996e6e
            e434d0d0          721a6868          01545555
19afb6b6          df914e4e          fa32c8c8          f030c0c0          21f6d7d7
            bc8e3232          75b3c6c6          6fe08f8f
691d7474          2ef5dbdb          6ae18b8b          962eb8b8          8a800a0a
            fe679999          e2c92b2b          e0618181
c0c30303          8d29a4a4          af238c8c          07a9aeae          390d3434
            1f524d4d          764f3939          d36ebdbd
81d65757          b7d86f6f          eb37dcdc          51441515          a6dd7b7b
            09fef7f7          b68c3a3a          932fbcbc
0f030c0c          03fcffff          c26ba9a9          ba73c9c9          d96cb5b5
            dc6db1b1          375a6d6d          15504545
b98f3636          771b6c6c          13adbebe          da904a4a          57b9eeee
            a9de7777          4cbef2f2          837efdfd
55114444          bdda6767          2c5d7171          45400505          631f7c7c
            50104040          325b6969          b8db6363
220a2828          c5c20707          f531c4c4          a88a2222          31a79696
```

```
           f9ce3737       977aeded        49bff6f6
992db4b4        a475d1d1        90d34343        5a124848        58bae2e2
           71e69797       64b6d2d2        70b2c2c2
ad8b2626        cd68a5a5        cb955e5e        624b2929        3c0c3030
           ce945a5a       ab76dddd        867ff9f9
f1649595        5dbbe6e6        35f2c7c7        2d092424        d1c61717
           d66fb9b9       dec51b1b        94861212
78186060        30f3c3c3        897cf5f5        5cefb3b3        d23ae8e8
           acdf7373       794c3535        a0208080
9d78e5e5        56edbbbb        235e7d7d        c63ef8f8        8bd45f5f
           e7c82f2f       dd39e4e4        68492121
T1:
5b8ed55b        42d09242        a74deaa7        fb06fdfb        33fccf33
           8765e287       f4c93df4        de6bb5de
584e1658        da6eb4da        50441450        0bcac10b        a08828a0
           ef17f8ef       b09c2cb0        14110514
ac872bac        9dfb669d        6af2986a        d9ae77d9        a8822aa8
           fa46bcfa       10140410        0fcfc00f
aa02a8aa        11544511        4c5f134c        98be2698        256d4825
           1a9e841a       181e0618        66fd9b66
72ec9e72        094a4309        41105141        d324f7d3        46d59346
           bf53ecbf       62f89a62        e9927be9
ccff33cc        51045551        2c270b2c        0d4f420d        b759eeb7
           3ff3cc3f       b21caeb2        89ea6389
9374e793        ce7fb1ce        706c1c70        a60daba6        27edca27
           20280820       a348eba3        56c19756
02808202        7fa3dc7f        52c49652        eb12f9eb        d5a174d5
           3eb38d3e       fcc33ffc        9a3ea49a
1d5b461d        1c1b071c        9e3ba59e        f30cfff3        cf3ff0cf
           cdbf72cd       5c4b175c        ea52b8ea
0e8f810e        653d5865        f0cc3cf0        647d1964        9b7ee59b
           16918716       3d734e3d        a208aaa2
a1c869a1        adc76aad        06858306        ca7ab0ca        c5b570c5
           91f46591       6bb2d96b        2ea7892e
e318fbe3        af47e8af        3c330f3c        2d674a2d        c1b071c1
           590e5759       76e99f76        d4e135d4
78661e78        90b42490        38360e38        79265f79        8def628d
           61385961       4795d247        8a2aa08a
94b12594        88aa2288        f18c7df1        ecd73bec        04050104
           84a52184       e19879e1        1e9b851e
5384d753        00000000        195e4719        5d0b565d        7ee39d7e
           4f9fd04f       9cbb279c        491a5349
317c4d31        d8ee36d8        080a0208        9f7be49f        8220a282
```

```
          13d4c713        23e8cb23        7ae69c7a
ab42e9ab        fe43bdfe       2aa2882a        4b9ad14b        01404101
          1fdbc41f        e0d838e0        d661b7d6
8e2fa18e        df2bf4df       cb3af1cb        3bf6cd3b        e71dfae7
          85e56085        54411554        8625a386
8360e383        ba16acba       75295c75        9234a692        6ef7996e
          d0e434d0        68721a68        55015455
b619afb6        4edf914e       c8fa32c8        c0f030c0        d721f6d7
          32bc8e32        c675b3c6        8f6fe08f
74691d74        db2ef5db       8b6ae18b        b8962eb8        0a8a800a
          99fe6799        2be2c92b        81e06181
03c0c303        a48d29a4       8caf238c        ae07a9ae        34390d34
          4d1f524d        39764f39        bdd36ebd
5781d657        6fb7d86f       dceb37dc        15514415        7ba6dd7b
          f709fef7        3ab68c3a        bc932fbc
0c0f030c        ff03fcff       a9c26ba9        c9ba73c9        b5d96cb5
          b1dc6db1        6d375a6d        45155045
36b98f36        6c771b6c       be13adbe        4ada904a        ee57b9ee
          77a9de77        f24cbef2        fd837efd
44551144        67bdda67       712c5d71        05454005        7c631f7c
          40501040        69325b69        63b8db63
28220a28        07c5c207       c4f531c4        22a88a22        9631a796
          37f9ce37        ed977aed        f649bff6
b4992db4        d1a475d1       4390d343        485a1248        e258bae2
          9771e697        d264b6d2        c270b2c2
26ad8b26        a5cd68a5       5ecb955e        29624b29        303c0c30
          5ace945a        ddab76dd        f9867ff9
95f16495        e65dbbe6       c735f2c7        242d0924        17d1c617
          b9d66fb9        1bdec51b        12948612
60781860        c330f3c3       f5897cf5        b35cefb3        e8d23ae8
          73acdf73        35794c35        80a02080
e59d78e5        bb56edbb       7d235e7d        f8c63ef8        5f8bd45f
          2fe7c82f        e4dd39e4        21684921
T2:
5b5b8ed5        4242d092       a7a74dea        fbfb06fd        3333fccf
          878765e2        f4f4c93d        dede6bb5
58584e16        dada6eb4       50504414        0b0bcac1        a0a08828
          efef17f8        b0b09c2c        14141105
acac872b        9d9dfb66       6a6af298        d9d9ae77        a8a8822a
          fafa46bc        10101404        0f0fcfc0
aaaa02a8        11115445       4c4c5f13        9898be26        25256d48
          1a1a9e84        18181e06        6666fd9b
7272ec9e        09094a43       41411051        d3d324f7        4646d593
```

```
          bfbf53ec     6262f89a     e9e9927b
ccccff33     51510455     2c2c270b     0d0d4f42     b7b759ee
          3f3ff3cc     b2b21cae     8989ea63
939374e7     cece7fb1     70706c1c     a6a60dab     2727edca
          20202808     a3a348eb     5656c197
02028082     7f7fa3dc     5252c496     ebeb12f9     d5d5a174
          3e3eb38d     fcfcc33f     9a9a3ea4
1d1d5b46     1c1c1b07     9e9e3ba5     f3f30cff     cfcf3ff0
          cdcdbf72     5c5c4b17     eaea52b8
0e0e8f81     65653d58     f0f0cc3c     64647d19     9b9b7ee5
          16169187     3d3d734e     a2a208aa
a1a1c869     adadc76a     06068583     caca7ab0     c5c5b570
          9191f465     6b6bb2d9     2e2ea789
e3e318fb     afaf47e8     3c3c330f     2d2d674a     c1c1b071
          59590e57     7676e99f     d4d4e135
7878661e     9090b424     3838360e     7979265f     8d8def62
          61613859     474795d2     8a8a2aa0
9494b125     8888aa22     f1f18c7d     ececd73b     04040501
          8484a521     e1e19879     1e1e9b85
535384d7     00000000     19195e47     5d5d0b56     7e7ee39d
          4f4f9fd0     9c9cbb27     49491a53
31317c4d     d8d8ee36     08080a02     9f9f7be4     828220a2
          1313d4c7     2323e8cb     7a7ae69c
abab42e9     fefe43bd     2a2aa288     4b4b9ad1     01014041
          1f1fdbc4     e0e0d838     d6d661b7
8e8e2fa1     dfdf2bf4     cbcb3af1     3b3bf6cd     e7e71dfa
          8585e560     54544115     868625a3
838360e3     baba16ac     7575295c     929234a6     6e6ef799
          d0d0e434     6868721a     55550154
b6b619af     4e4edf91     c8c8fa32     c0c0f030     d7d721f6
          3232bc8e     c6c675b3     8f8f6fe0
7474691d     dbdb2ef5     8b8b6ae1     b8b8962e     0a0a8a80
          9999fe67     2b2be2c9     8181e061
0303c0c3     a4a48d29     8c8caf23     aeae07a9     3434390d
          4d4d1f52     3939764f     bdbdd36e
575781d6     6f6fb7d8     dcdceb37     15155144     7b7ba6dd
          f7f709fe     3a3ab68c     bcbc932f
0c0c0f03     ffff03fc     a9a9c26b     c9c9ba73     b5b5d96c
          b1b1dc6d     6d6d375a     45451550
3636b98f     6c6c771b     bebe13ad     4a4ada90     eeee57b9
          7777a9de     f2f24cbe     fdfd837e
44445511     6767bdda     71712c5d     05054540     7c7c631f
          40405010     6969325b     6363b8db
```

```
2828220a 0707c5c2 c4c4f531 2222a88a 969631a7 3737f9ce eded977a f6f649bf
b4b4992d d1d1a475 434390d3 48485a12 e2e258ba 979771e6 d2d264b6 c2c270b2
2626ad8b a5a5cd68 5e5ecb95 2929624b 30303c0c 5a5ace94 ddddab76 f9f9867f
9595f164 e6e65dbb c7c735f2 24242d09 1717d1c6 b9b9d66f 1b1bdec5 12129486
60607818 c3c330f3 f5f5897c b3b35cef e8e8d23a 7373acdf 3535794c 8080a020
e5e59d78 bbbb56ed 7d7d235e f8f8c63e 5f5f8bd4 2f2fe7c8 e4e4dd39 21216849
T3:
d55b5b8e 924242d0 eaa7a74d fdfbfb06 cf3333fc e2878765 3df4f4c9 b5dede6b
1658584e b4dada6e 14505044 c10b0bca 28a0a088 f8efef17 2cb0b09c 05141411
2bacac87 669d9dfb 986a6af2 77d9d9ae 2aa8a882 bcfafa46 04101014 c00f0fcf
a8aaaa02 45111154 134c4c5f 269898be 4825256d 841a1a9e 0618181e 9b6666fd
9e7272ec 4309094a 51414110 f7d3d324 934646d5 ecbfbf53 9a6262f8 7be9e992
33ccccff 55515104 0b2c2c27 420d0d4f eeb7b759 cc3f3ff3 aeb2b21c 638989ea
e7939374 b1cece7f 1c70706c aba6a60d ca2727ed 08202028 eba3a348 975656c1
82020280 dc7f7fa3 965252c4 f9ebeb12 74d5d5a1 8d3e3eb3 3ffcfcc3 a49a9a3e
461d1d5b 071c1c1b a59e9e3b fff3f30c f0cfcf3f 72cdcdbf 175c5c4b b8eaea52
810e0e8f 5865653d 3cf0f0cc 1964647d e59b9b7e 87161691 4e3d3d73 aaa2a208
69a1a1c8 6aadadc7 83060685 b0caca7a 70c5c5b5 659191f4 d96b6bb2 892e2ea7
fbe3e318 e8afaf47 0f3c3c33 4a2d2d67 71c1c1b0 5759590e 9f7676e9 35d4d4e1
1e787866 249090b4 0e383836 5f797926 628d8def 59616138 d2474795 a08a8a2a
259494b1 228888aa 7df1f18c 3bececd7 01040405 218484a5 79e1e198 851e1e9b
d7535384 00000000 4719195e 565d5d0b 9d7e7ee3 d04f4f9f 279c9cbb 5349491a
```

```
4d31317c    36d8d8ee    0208080a    e49f9f7b    a2828220    c71313d4    cb2323e8    9c7a7ae6
e9abab42    bdfefe43    882a2aa2    d14b4b9a    41010140    c41f1fdb    38e0e0d8    b7d6d661
a18e8e2f    f4dfdf2b    f1cbcb3a    cd3b3bf6    fae7e71d    608585e5    15545441    a3868625
e3838360    acbaba16    5c757529    a6929234    996e6ef7    34d0d0e4    1a686872    54555501
afb6b619    914e4edf    32c8c8fa    30c0c0f0    f6d7d721    8e3232bc    b3c6c675    e08f8f6f
1d747469    f5dbdb2e    e18b8b6a    2eb8b896    800a0a8a    679999fe    c92b2be2    618181e0
c30303c0    29a4a48d    238c8caf    a9aeae07    0d343439    524d4d1f    4f393976    6ebdbdd3
d6575781    d86f6fb7    37dcdceb    44151551    dd7b7ba6    fef7f709    8c3a3ab6    2fbcbc93
030c0c0f    fcffff03    6ba9a9c2    73c9c9ba    6cb5b5d9    6db1b1dc    5a6d6d37    50454515
8f3636b9    1b6c6c77    adbebe13    904a4ada    b9eeee57    de7777a9    bef2f24c    7efdfd83
11444455    da6767bd    5d71712c    40050545    1f7c7c63    10404050    5b696932    db6363b8
0a282822    c20707c5    31c4c4f5    8a2222a8    a7969631    ce3737f9    7aeded97    bff6f649
2db4b499    75d1d1a4    d3434390    1248485a    bae2e258    e6979771    b6d2d264    b2c2c270
8b2626ad    68a5a5cd    955e5ecb    4b292962    0c30303c    945a5ace    76ddddab    7ff9f986
649595f1    bbe6e65d    f2c7c735    0924242d    c61717d1    6fb9b9d6    c51b1bde    86121294
18606078    f3c3c330    7cf5f589    efb3b35c    3ae8e8d2    df7373ac    4c353579    208080a0
78e5e59d    edbbbb56    5e7d7d23    3ef8f8c6    d45f5f8b    c82f2fe7    39e4e4dd    49212168
```

将上述四个表放入代码中直接查表。

完整代码如下所示：

```
#include<iostream>
#include<chrono>
#include <iomanip>

using namespace std;
```

```

// S盒
static const unsigned long SboxTable[16][16] = {
    {0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2,
         0x28, 0xfb, 0x2c, 0x05},
    {0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26,
         0x49, 0x86, 0x06, 0x99},
    {0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43,
         0xed, 0xcf, 0xac, 0x62},
    {0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa,
         0x75, 0x8f, 0x3f, 0xa6},
    {0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19,
         0xe6, 0x85, 0x4f, 0xa8},
    {0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b,
         0x70, 0x56, 0x9d, 0x35},
    {0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b,
         0x01, 0x21, 0x78, 0x87},
    {0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7,
         0xa0, 0xc4, 0xc8, 0x9e},
    {0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce,
         0xf9, 0x61, 0x15, 0xa1},
    {0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30,
         0xf5, 0x8c, 0xb1, 0xe3},
    {0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab,
         0x0d, 0x53, 0x4e, 0x6f},
    {0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72,
         0x6d, 0x6c, 0x5b, 0x51},
    {0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41,
         0x1f, 0x10, 0x5a, 0xd8},
    {0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12,
         0xb8, 0xe5, 0xb4, 0xb0},
    {0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09,
         0xc5, 0x6e, 0xc6, 0x84},
    {0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e,
         0xd7, 0xcb, 0x39, 0x48}
};

unsigned long sm4Sbox(unsigned long in) {
    return SboxTable[(in >> 4) & 0x0F][in & 0x0F];
}

unsigned long L(unsigned long x) {
    return x ^ (x << 2 | x >> (32 - 2)) ^ (x << 10 | x >> (32 - 10)) ^ (x <<
        18 | x >> (32 - 18)) ^ (x << 24 | x >> (32 - 24));
}

unsigned long T(unsigned long x) {
    unsigned long b = 0;
    for (int i = 0; i < 4; i++) {
        b = (b << 8) | sm4Sbox((x >> ((3 - i) * 8)) & 0xFF);
    }
    return L(b);
}

// 优化 T 可算表
//unsigned long T0[256], T1[256], T2[256], T3[256];
unsigned long T0[256] = {
    0x8ED55B5B, 0xD0924242, 0x4DEAA7A7, 0x06FDFBFB, 0xFCCF3333, 0x65E28787,
        0xC93DF4F4, 0x6BB5DEDE, 0x4E165858, 0x6EB4DADA, 0x44145050, 0xCAC10B0B,
        0x8828A0A0, 0x17F8EFEF, 0x9C2CB0B0, 0x11051414, 0x872BACAC, 0xFB669D9D,
        0xF2986A6A, 0xAE77D9D9, 0x822AA8A8, 0x46BCFAFA, 0x14041010, 0xCFC00F0F,
        0x02A8AAAA, 0x54451111, 0x5F134C4C, 0xBE269898, 0x6D482525, 0x9E841A1A,
        0x1E061818, 0xFD9B6666, 0xEC9E7272, 0x4A430909, 0x10514141, 0x24F7D3D3,
        0xD5934646, 0x53ECBFBF, 0xF89A6262, 0x927BE9E9, 0xFF33CCCC, 0x04555151,
        0x270B2C2C, 0x4F420D0D, 0x59EEB7B7, 0xF3CC3F3F, 0x1CAEB2B2, 0xEA638989,
        0x74E79393, 0x7FB1CECE, 0x6C1C7070, 0x0DABA6A6, 0xEDCA2727, 0x28082020,
        0x48EBA3A3, 0xC1975656, 0x80820202, 0xA3DC7F7F, 0xC4965252, 0x12F9EBEB,
        0xA174D5D5, 0xB38D3E3E, 0xC33FFCFC, 0x3EA49A9A, 0x5B461D1D, 0x1B071C1C,
        0x3BA59E9E, 0x0CFFF3F3, 0x3FF0CFCF, 0xBF72CDCD, 0x4B175C5C, 0x52B8EAEA,
        0x8F810E0E, 0x3D586565, 0xCC3CF0F0, 0x7D196464, 0x7EE59B9B, 0x91871616,
        0x734E3D3D, 0x08AAA2A2, 0xC869A1A1, 0xC76AADAD, 0x85830606, 0x7AB0CACA,
        0xB570C5C5, 0xF4659191, 0xB2D96B6B, 0xA7892E2E, 0x18FBE3E3, 0x47E8AFAF,
        0x330F3C3C, 0x674A2D2D, 0xB071C1C1, 0x0E575959, 0xE99F7676, 0
            xE135D4D4,
        0x661E7878, 0xB4249090, 0x360E3838, 0x265F7979, 0xEF628D8D, 0
            x38596161,
        0x95D24747, 0x2AA08A8A, 0xB1259494, 0xAA228888, 0x8C7DF1F1, 0
            xD73BECEC,
        0x05010404, 0xA5218484, 0x9879E1E1, 0x9B851E1E, 0x84D75353, 0
            x00000000,
        0x5E471919, 0x0B565D5D, 0xE39D7E7E, 0x9FD04F4F, 0xBB279C9C, 0
            x1A534949,
        0x7C4D3131, 0xEE36D8D8, 0x0A020808, 0x7BE49F9F, 0x20A28282, 0
            xD4C71313,
        0xE8CB2323, 0xE69C7A7A, 0x42E9ABAB, 0x43BDFEFE, 0xA2882A2A, 0
            x9AD14B4B,
        0x40410101, 0xDBC41F1F, 0xD838E0E0, 0x61B7D6D6, 0x2FA18E8E, 0
            x2BF4DFDF,
        0x3AF1CBCB, 0xF6CD3B3B, 0x1DFAE7E7, 0xE5608585, 0x41155454, 0
            x25A38686,
        0x60E38383, 0x16ACBABA, 0x295C7575, 0x34A69292, 0xF7996E6E, 0
            xE434D0D0,
        0x721A6868, 0x01545555, 0x19AFB6B6, 0xDF914E4E, 0xFA32C8C8, 0
            xF030C0C0,
        0x21F6D7D7, 0xBC8E3232, 0x75B3C6C6, 0x6FE08F8F, 0x691D7474, 0
            x2EF5DBDB,
        0x6AE18B8B, 0x962EB8B8, 0x8A800A0A, 0xFE679999, 0xE2C92B2B, 0
            xE0618181,
        0xC0C30303, 0x8D29A4A4, 0xAF238C8C, 0x07A9AEAE, 0x390D3434, 0
            x1F524D4D,
        0x764F3939, 0xD36EBDBD, 0x81D65757, 0xB7D86F6F, 0xEB37DCDC, 0
            x51441515,
        0xA6DD7B7B, 0x09FEF7F7, 0xB68C3A3A, 0x932FBCBC, 0x0F030C0C, 0
            x03FCFFFF,
        0xC26BA9A9, 0xBA73C9C9, 0xD96CB5B5, 0xDC6DB1B1, 0x375A6D6D, 0
            x15504545,
        0xB98F3636, 0x771B6C6C, 0x13ADBEBE, 0xDA904A4A, 0x57B9EEEE, 0
            xA9DE7777,
        0x4CBEF2F2, 0x837EFDFD, 0x55114444, 0xBDDA6767, 0x2C5D7171, 0
            x45400505,
        0x631F7C7C, 0x50104040, 0x325B6969, 0xB8DB6363, 0x220A2828, 0
            xC5C20707,
        0xF531C4C4, 0xA88A2222, 0x31A79696, 0xF9CE3737, 0x977AEDED, 0
            x49BFF6F6,
        0x992DB4B4, 0xA475D1D1, 0x90D34343, 0x5A124848, 0x58BAE2E2, 0
```

```
            x71E69797,
        0x64B6D2D2, 0x70B2C2C2, 0xAD8B2626, 0xCD68A5A5, 0xCB955E5E, 0
            x624B2929,
        0x3C0C3030, 0xCE945A5A, 0xAB76DDDD, 0x867FF9F9, 0xF1649595, 0
            x5DBBE6E6,
        0x35F2C7C7, 0x2D092424, 0xD1C61717, 0xD66FB9B9, 0xDEC51B1B, 0
            x94861212,
        0x78186060, 0x30F3C3C3, 0x897CF5F5, 0x5CEFB3B3, 0xD23AE8E8, 0
            xACDF7373,
        0x794C3535, 0xA0208080, 0x9D78E5E5, 0x56EDBBBB, 0x235E7D7D, 0
            xC63EF8F8,
        0x8BD45F5F, 0xE7C82F2F, 0xDD39E4E4, 0x68492121
};

unsigned long T1[256] = {
    0x5B8ED55B, 0x42D09242, 0xA74DEAA7, 0xFB06FDFB, 0x33FCCF33, 0x8765E287,
    0xF4C93DF4, 0xDE6BB5DE, 0x584E1658, 0xDA6EB4DA, 0x50441450, 0x0BCAC10B,
    0xA08828A0, 0xEF17F8EF, 0xB09C2CB0, 0x14110514, 0xAC872BAC, 0x9DFB669D,
    0x6AF2986A, 0xD9AE77D9, 0xA8822AA8, 0xFA46BCFA, 0x10140410, 0x0FCFC00F,
    0xAA02A8AA, 0x11544511, 0x4C5F134C, 0x98BE2698, 0x256D4825, 0x1A9E841A,
    0x181E0618, 0x66FD9B66, 0x72EC9E72, 0x094A4309, 0x41105141, 0xD324F7D3,
    0x46D59346, 0xBF53ECBF, 0x62F89A62, 0xE9927BE9, 0xCCFF33CC, 0x51045551,
    0x2C270B2C, 0x0D4F420D, 0xB759EEB7, 0x3FF3CC3F, 0xB21CAEB2, 0x89EA6389,
    0x9374E793, 0xCE7FB1CE, 0x706C1C70, 0xA60DABA6, 0x27EDCA27, 0x20280820,
    0xA348EBA3, 0x56C19756, 0x02808202, 0x7FA3DC7F, 0x52C49652, 0xEB12F9EB,
    0xD5A174D5, 0x3EB38D3E, 0xFCC33FFC, 0x9A3EA49A, 0x1D5B461D, 0x1C1B071C,
    0x9E3BA59E, 0xF30CFFF3, 0xCF3FF0CF, 0xCDBF72CD, 0x5C4B175C, 0xEA52B8EA,
    0x0E8F810E, 0x653D5865, 0xF0CC3CF0, 0x647D1964, 0x9B7EE59B, 0x16918716,
    0x3D734E3D, 0xA208AAA2, 0xA1C869A1, 0xADC76AAD, 0x06858306, 0xCA7AB0CA,
    0xC5B570C5, 0x91F46591, 0x6BB2D96B, 0x2EA7892E, 0xE318FBE3, 0xAF47E8AF,
    0x3C330F3C, 0x2D674A2D, 0xC1B071C1, 0x590E5759, 0x76E99F76, 0xD4E135D4,
    0x78661E78, 0x90B42490, 0x38360E38, 0x79265F79, 0x8DEF628D, 0x61385961,
    0x4795D247, 0x8A2AA08A, 0x94B12594, 0x88AA2288, 0xF18C7DF1, 0xECD73BEC,
    0x04050104, 0x84A52184, 0xE19879E1, 0x1E9B851E, 0x5384D753, 0x00000000,
    0x195E4719, 0x5D0B565D, 0x7EE39D7E, 0x4F9FD04F, 0x9CBB279C, 0x491A5349,
    0x317C4D31, 0xD8EE36D8, 0x080A0208, 0x9F7BE49F, 0x8220A282, 0x13D4C713,
    0x23E8CB23, 0x7AE69C7A, 0xAB42E9AB, 0xFE43BDFE, 0x2AA2882A, 0x4B9AD14B,
    0x01404101, 0x1FDBC41F, 0xE0D838E0, 0xD661B7D6, 0x8E2FA18E, 0xDF2BF4DF,
    0xCB3AF1CB, 0x3BF6CD3B, 0xE71DFAE7, 0x85E56085, 0x54411554, 0x8625A386,
    0x8360E383, 0xBA16ACBA, 0x75295C75, 0x9234A692, 0x6EF7996E, 0xD0E434D0,
    0x68721A68, 0x55015455, 0xB619AFB6, 0x4EDF914E, 0xC8FA32C8, 0xC0F030C0,
    0xD721F6D7, 0x32BC8E32, 0xC675B3C6, 0x8F6FE08F, 0x74691D74, 0xDB2EF5DB,
    0x8B6AE18B, 0xB8962EB8, 0x0A8A800A, 0x99FE6799, 0x2BE2C92B, 0x81E06181,
    0x03C0C303, 0xA48D29A4, 0x8CAF238C, 0xAE07A9AE, 0x34390D34, 0x4D1F524D,
    0x39764F39, 0xBDD36EBD, 0x5781D657, 0x6FB7D86F, 0xDCEB37DC, 0x15514415,
    0x7BA6DD7B, 0xF709FEF7, 0x3AB68C3A, 0xBC932FBC, 0x0C0F030C, 0xFF03FCFF,
    0xA9C26BA9, 0xC9BA73C9, 0xB5D96CB5, 0xB1DC6DB1, 0x6D375A6D, 0x45155045,
    0x36B98F36, 0x6C771B6C, 0xBE13ADBE, 0x4ADA904A, 0xEE57B9EE, 0x77A9DE77,
    0xF24CBEF2, 0xFD837EFD, 0x44551144, 0x67BDDA67, 0x712C5D71, 0x05454005,
    0x7C631F7C, 0x40501040, 0x69325B69, 0x63B8DB63, 0x28220A28, 0x07C5C207,
    0xC4F531C4, 0x22A88A22, 0x9631A796, 0x37F9CE37, 0xED977AED, 0xF649BFF6,
    0xB4992DB4, 0xD1A475D1, 0x4390D343, 0x485A1248, 0xE258BAE2, 0x9771E697,
    0xD264B6D2, 0xC270B2C2, 0x26AD8B26, 0xA5CD68A5, 0x5ECB955E, 0x29624B29,
    0x303C0C30, 0x5ACE945A, 0xDDAB76DD, 0xF9867FF9, 0x95F16495, 0xE65DBBE6,
    0xC735F2C7, 0x242D0924, 0x17D1C617, 0xB9D66FB9, 0x1BDEC51B, 0x12948612,
    0x60781860, 0xC330F3C3, 0xF5897CF5, 0xB35CEFB3, 0xE8D23AE8, 0x73ACDF73,
    0x35794C35, 0x80A02080, 0xE59D78E5, 0xBB56EDBB, 0x7D235E7D, 0xF8C63EF8,
    0x5F8BD45F, 0x2FE7C82F, 0xE4DD39E4, 0x21684921 };

unsigned long T2[256] = {
    0x5B5B8ED5, 0x4242D092, 0xA7A74DEA, 0xFBFB06FD, 0x3333FCCF, 0x878765E2,
    0xF4F4C93D, 0xDEDE6BB5, 0x58584E16, 0xDADA6EB4, 0x50504414, 0x0B0BCAC1,
    0xA0A08828, 0xEFEF17F8, 0xB0B09C2C, 0x14141105, 0xACAC872B, 0x9D9DFB66,
    0x6A6AF298, 0xD9D9AE77, 0xA8A8822A, 0xFAFA46BC, 0x10101404, 0x0F0FCFC0,
    0xAAAA02A8, 0x11115445, 0x4C4C5F13, 0x9898BE26, 0x25256D48, 0x1A1A9E84,
    0x18181E06, 0x6666FD9B, 0x7272EC9E, 0x09094A43, 0x41411051, 0xD3D324F7,
    0x4646D593, 0xBFBF53EC, 0x6262F89A, 0xE9E9927B, 0xCCCCFF33, 0x51510455,
    0x2C2C270B, 0x0D0D4F42, 0xB7B759EE, 0x3F3FF3CC, 0xB2B21CAE, 0x8989EA63,
    0x939374E7, 0xCECE7FB1, 0x70706C1C, 0xA6A60DAB, 0x2727EDCA, 0x20202808,
    0xA3A348EB, 0x5656C197, 0x02028082, 0x7F7FA3DC, 0x5252C496, 0xEBEB12F9,
    0xD5D5A174, 0x3E3EB38D, 0xFCFCC33F, 0x9A9A3EA4, 0x1D1D5B46, 0x1C1C1B07,
    0x9E9E3BA5, 0xF3F30CFF, 0xCFCF3FF0, 0xCDCDBF72, 0x5C5C4B17, 0xEAEA52B8,
    0x0E0E8F81, 0x65653D58, 0xF0F0CC3C, 0x64647D19, 0x9B9B7EE5, 0x16169187,
    0x3D3D734E, 0xA2A208AA, 0xA1A1C869, 0xADADC76A, 0x06068583, 0xCACA7AB0,
    0xC5C5B570, 0x9191F465, 0x6B6BB2D9, 0x2E2EA789, 0xE3E318FB, 0xAFAF47E8,
    0x3C3C330F, 0x2D2D674A, 0xC1C1B071, 0x59590E57, 0x7676E99F, 0xD4D4E135,
    0x7878661E, 0x9090B424, 0x3838360E, 0x7979265F, 0x8D8DEF62, 0x61613859,
    0x474795D2, 0x8A8A2AA0, 0x9494B125, 0x8888AA22, 0xF1F18C7D, 0xECECD73B,
    0x04040501, 0x8484A521, 0xE1E19879, 0x1E1E9B85, 0x535384D7, 0x00000000,
    0x19195E47, 0x5D5D0B56, 0x7E7EE39D, 0x4F4F9FD0, 0x9C9CBB27, 0x49491A53,
    0x31317C4D, 0xD8D8EE36, 0x08080A02, 0x9F9F7BE4, 0x828220A2, 0x1313D4C7,
    0x2323E8CB, 0x7A7AE69C, 0xABAB42E9, 0xFEFE43BD, 0x2A2AA288, 0x4B4B9AD1,
    0x01014041, 0x1F1FDBC4, 0xE0E0D838, 0xD6D661B7, 0x8E8E2FA1, 0xDFDF2BF4,
    0xCBCB3AF1, 0x3B3BF6CD, 0xE7E71DFA, 0x8585E560, 0x54544115, 0x868625A3,
    0x838360E3, 0xBABA16AC, 0x7575295C, 0x929234A6, 0x6E6EF799, 0xD0D0E434,
    0x6868721A, 0x55550154, 0xB6B619AF, 0x4E4EDF91, 0xC8C8FA32, 0xC0C0F030,
    0xD7D721F6, 0x3232BC8E, 0xC6C675B3, 0x8F8F6FE0, 0x7474691D, 0xDBDB2EF5,
    0x8B8B6AE1, 0xB8B8962E, 0x0A0A8A80, 0x9999FE67, 0x2B2BE2C9, 0x8181E061,
    0x0303C0C3, 0xA4A48D29, 0x8C8CAF23, 0xAEAE07A9, 0x3434390D, 0x4D4D1F52,
    0x3939764F, 0xBDBDD36E, 0x575781D6, 0x6F6FB7D8, 0xDCDCEB37, 0x15155144,
    0x7B7BA6DD, 0xF7F709FE, 0x3A3AB68C, 0xBCBC932F, 0x0C0C0F03, 0xFFFF03FC,
    0xA9A9C26B, 0xC9C9BA73, 0xB5B5D96C, 0xB1B1DC6D, 0x6D6D375A, 0x45451550,
    0x3636B98F, 0x6C6C771B, 0xBEBE13AD, 0x4A4ADA90, 0xEEEE57B9, 0x7777A9DE,
    0xF2F24CBE, 0xFDFD837E, 0x44445511, 0x6767BDDA, 0x71712C5D, 0x05054540,
    0x7C7C631F, 0x40405010, 0x6969325B, 0x6363B8DB, 0x2828220A, 0x0707C5C2,
    0xC4C4F531, 0x2222A88A, 0x969631A7, 0x3737F9CE, 0xEDED977A, 0xF6F649BF,
    0xB4B4992D, 0xD1D1A475, 0x434390D3, 0x48485A12, 0xE2E258BA, 0x979771E6,
    0xD2D264B6, 0xC2C270B2, 0x2626AD8B, 0xA5A5CD68, 0x5E5ECB95, 0x2929624B,
    0x30303C0C, 0x5A5ACE94, 0xDDDDAB76, 0xF9F9867F, 0x9595F164, 0xE6E65DBB,
    0xC7C735F2, 0x24242D09, 0x1717D1C6, 0xB9B9D66F, 0x1B1BDEC5, 0x12129486,
    0x60607818, 0xC3C330F3, 0xF5F5897C, 0xB3B35CEF, 0xE8E8D23A, 0x7373ACDF,
    0x3535794C, 0x8080A020, 0xE5E59D78, 0xBBBB56ED, 0x7D7D235E, 0xF8F8C63E,
    0x5F5F8BD4, 0x2F2FE7C8, 0xE4E4DD39, 0x21216849 };

unsigned long T3[256] = {
    0xD55B5B8E, 0x924242D0, 0xEAA7A74D, 0xFDFBFB06, 0xCF3333FC, 0xE2878765,
    0x3DF4F4C9, 0xB5DEDE6B, 0x1658584E, 0xB4DADA6E, 0x14505044, 0xC10B0BCA,
    0x28A0A088, 0xF8EFEF17, 0x2CB0B09C, 0x05141411, 0x2BACAC87, 0x669D9DFB,
    0x986A6AF2, 0x77D9D9AE, 0x2AA8A882, 0xBCFAFA46, 0x04101014, 0xC00F0FCF,
    0xA8AAAA02, 0x45111154, 0x134C4C5F, 0x269898BE, 0x4825256D, 0x841A1A9E,
    0x0618181E, 0x9B6666FD, 0x9E7272EC, 0x4309094A, 0x51414110, 0xF7D3D324,
    0x934646D5, 0xECBFBF53, 0x9A6262F8, 0x7BE9E992, 0x33CCCCFF, 0x55515104,
    0x0B2C2C27, 0x420D0D4F, 0xEEB7B759, 0xCC3F3FF3, 0xAEB2B21C, 0x638989EA,
    0xE7939374, 0xB1CECE7F, 0x1C70706C, 0xABA6A60D, 0xCA2727ED, 0x08202028,
    0xEBA3A348, 0x975656C1, 0x82020280, 0xDC7F7FA3, 0x965252C4, 0xF9EBEB12,
    0x74D5D5A1, 0x8D3E3EB3, 0x3FFCFCC3, 0xA49A9A3E, 0x461D1D5B, 0x071C1C1B,
    0xA59E9E3B, 0xFFF3F30C, 0xF0CFCF3F, 0x72CDCDBF, 0x175C5C4B, 0xB8EAEA52,
    0x810E0E8F, 0x5865653D, 0x3CF0F0CC, 0x1964647D, 0xE59B9B7E, 0x87161691,
    0x4E3D3D73, 0xAAA2A208, 0x69A1A1C8, 0x6AADADC7, 0x83060685, 0xB0CACA7A,
    0x70C5C5B5, 0x659191F4, 0xD96B6BB2, 0x892E2EA7, 0xFBE3E318, 0xE8AFAF47,
    0x0F3C3C33, 0x4A2D2D67, 0x71C1C1B0, 0x5759590E, 0x9F7676E9, 0x35D4D4E1,
    0x1E787866, 0x249090B4, 0x0E383836, 0x5F797926, 0x628D8DEF, 0x59616138,
    0xD2474795, 0xA08A8A2A, 0x259494B1, 0x228888AA, 0x7DF1F18C, 0x3BECECD7,
    0x01040405, 0x218484A5, 0x79E1E198, 0x851E1E9B, 0xD7535384, 0x00000000,
    0x4719195E, 0x565D5D0B, 0x9D7E7EE3, 0xD04F4F9F, 0x279C9CBB, 0x5349491A,
    0x4D31317C, 0x36D8D8EE, 0x0208080A, 0xE49F9F7B, 0xA2828220, 0xC71313D4,
    0xCB2323E8, 0x9C7A7AE6, 0xE9ABAB42, 0xBDFEFE43, 0x882A2AA2, 0xD14B4B9A,
    0x41010140, 0xC41F1FDB, 0x38E0E0D8, 0xB7D6D661, 0xA18E8E2F, 0xF4DFDF2B,
    0xF1CBCB3A, 0xCD3B3BF6, 0xFAE7E71D, 0x608585E5, 0x15545441, 0xA3868625,
    0xE3838360, 0xACBABA16, 0x5C757529, 0xA6929234, 0x996E6EF7, 0x34D0D0E4,
    0x1A686872, 0x54555501, 0xAFB6B619, 0x914E4EDF, 0x32C8C8FA, 0x30C0C0F0,
    0xF6D7D721, 0x8E3232BC, 0xB3C6C675, 0xE08F8F6F, 0x1D747469, 0xF5DBDB2E,
    0xE18B8B6A, 0x2EB8B896, 0x800A0A8A, 0x679999FE, 0xC92B2BE2, 0x618181E0,
    0xC30303C0, 0x29A4A48D, 0x238C8CAF, 0xA9AEAE07, 0x0D343439, 0x524D4D1F,
    0x4F393976, 0x6EBDBDD3, 0xD6575781, 0xD86F6FB7, 0x37DCDCEB, 0x44151551,
    0xDD7B7BA6, 0xFEF7F709, 0x8C3A3AB6, 0x2FBCBC93, 0x030C0C0F, 0xFCFFFF03,
    0x6BA9A9C2, 0x73C9C9BA, 0x6CB5B5D9, 0x6DB1B1DC, 0x5A6D6D37, 0x50454515,
    0x8F3636B9, 0x1B6C6C77, 0xADBEBE13, 0x904A4ADA, 0xB9EEEE57, 0xDE7777A9,
    0xBEF2F24C, 0x7EFDFD83, 0x11444455, 0xDA6767BD, 0x5D71712C, 0x40050545,
    0x1F7C7C63, 0x10404050, 0x5B696932, 0xDB6363B8, 0x0A282822, 0xC20707C5,
    0x31C4C4F5, 0x8A2222A8, 0xA7969631, 0xCE3737F9, 0x7AEDED97, 0xBFF6F649,
    0x2DB4B499, 0x75D1D1A4, 0xD3434390, 0x1248485A, 0xBAE2E258, 0xE6979771,
    0xB6D2D264, 0xB2C2C270, 0x8B2626AD, 0x68A5A5CD, 0x955E5ECB, 0x4B292962,
    0x0C30303C, 0x945A5ACE, 0x76DDDDAB, 0x7FF9F986, 0x649595F1, 0xBBE6E65D,
    0xF2C7C735, 0x0924242D, 0xC61717D1, 0x6FB9B9D6, 0xC51B1BDE, 0x86121294,
    0x18606078, 0xF3C3C330, 0x7CF5F589, 0xEFB3B35C, 0x3AE8E8D2, 0xDF7373AC,
    0x4C353579, 0x208080A0, 0x78E5E59D, 0xEDBBBB56, 0x5E7D7D23, 0x3EF8F8C6,
    0xD45F5F8B, 0xC82F2FE7, 0x39E4E4DD, 0x49212168 };

/*
void init_T_tables() {
    for (int x = 0; x < 256; x++) {
        unsigned char s = sm4Sbox(x);
        T0[x] = L((unsigned long)s << 24);
        T1[x] = L((unsigned long)s << 16);
        T2[x] = L((unsigned long)s << 8);
        T3[x] = L((unsigned long)s);
    }
}
*/
unsigned long T_opt(unsigned long x) {
    return T0[(x >> 24) & 0xFF] ^ T1[(x >> 16) & 0xFF] ^ T2[(x >> 8) & 0xFF]
         ^ T3[x & 0xFF];
}

static const unsigned long FK[4] = { 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0
    xb27022dc };

static const unsigned long CK[32] = {
    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269, 0x70777e85, 0x8c939aa1,
        0xa8afb6bd, 0xc4cbd2d9,
    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249, 0x50575e65, 0x6c737a81,
    0x888f969d, 0xa4abb2b9,
    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229, 0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209, 0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
};

void key_expansion(unsigned long MK[4], unsigned long rk[32]) {
    unsigned long K[36];
    for (int i = 0; i < 4; i++)
        K[i] = MK[i] ^ FK[i];
    for (int i = 0; i < 32; i++) {
        unsigned long tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];
        unsigned long b = 0;
        for (int j = 0; j < 4; j++)
            b = (b << 8) | sm4Sbox((tmp >> ((3 - j) * 8)) & 0xFF);
        unsigned long Lval = b ^ (b << 13 | b >> (32 - 13)) ^ (b << 23 | b >> (32 - 23));
        rk[i] = K[i] ^ Lval;
        K[i + 4] = rk[i];
    }
}

unsigned long round_operate(int i, unsigned long* X, unsigned long* rk) {
    return X[i] ^ T_opt(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i]);
}

void sm4_enc(unsigned long MK[4], unsigned long X[4]) {
    cout << hex;
    cout << "Plaintext:" << endl;
    cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;
    cout << "Key:" << endl;
    cout << MK[0] << " " << MK[1] << " " << MK[2] << " " << MK[3] << endl;

    unsigned long rk[32];
    key_expansion(MK, rk);
    for (int i = 0; i < 32; i++) {
        unsigned long tmp = round_operate(i, X, rk);
        X[4 + i] = tmp;
    }
    cout << "Ciphertext:" << endl;
    cout << X[35] << " " << X[34] << " " << X[33] << " " << X[32] << endl;
}
```

```
void sm4_dec(unsigned long MK[4], unsigned long X[4]) {
    cout << hex;
    cout << "Ciphertext:" << endl;
    cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;

    unsigned long rk[32];
    key_expansion(MK, rk);
    for (int i = 0; i < 16; i++) swap(rk[i], rk[31 - i]);

    unsigned long tmpX[36] = { 0 };
    for (int i = 0; i < 4; i++) tmpX[i] = X[i];
    for (int i = 0; i < 32; i++) tmpX[i + 4] = round_operate(i, tmpX, rk);

    cout << "Decrypted Plaintext:" << endl;
    cout << tmpX[35] << " " << tmpX[34] << " " << tmpX[33] << " " << tmpX[32] << endl;
}

int main() {
    //init_T_tables();
    unsigned long MK[4] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210 };
    unsigned long X[36] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210 };
    unsigned long C[36] = { 0x681edf34, 0xd206965e, 0x86b3e94f, 0x536e4246 };
    auto start = chrono::high_resolution_clock::now();
    sm4_enc(MK, X);
    auto end = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::microseconds>(end - start).count();
    cout << "加密耗时：" << duration << " 微秒" << endl;


    start = chrono::high_resolution_clock::now();
    sm4_dec(MK, C);
    end = chrono::high_resolution_clock::now();
    duration = chrono::duration_cast<chrono::microseconds>(end - start).count();
    cout << "解密耗时：" << duration<< " 微秒" << endl;

    return 0;

}
```

3.2 优化方法 2：SIMD 指令集优化

这里面有多个优化点：

优化点 1：循环左移函数内联化 +simd 优化利用 SIMD 加速循环左移：

```
inline __m128i rotl32_simd(__m128i x, int r) {
    return _mm_or_si128(_mm_slli_epi32(x, r), _mm_srli_epi32(x, 32 - r));
}
```

优化点 2：线性变换 L 加速

使用 SIMD 的 _mm_slli_epi32 和 _mm_srli_epi32 实现 4 个 32 位数并行的循环左移，然后 _mm_xor_si128 实现多次异或。

```
// SIMD优化L变换：对4个32位整数同时做L变换
inline __m128i L_simd(__m128i x) {
    __m128i t1 = rotl32_simd(x, 2);
    __m128i t2 = rotl32_simd(x, 10);
    __m128i t3 = rotl32_simd(x, 18);
    __m128i t4 = rotl32_simd(x, 24);
    return _mm_xor_si128(_mm_xor_si128(_mm_xor_si128(x, t1), t2),
        _mm_xor_si128(t3, t4));
}
```

优化点 3：轮函数 T 使用 SIMD 优化的 L

虽然 s 盒查找更大头，但是这里没有继续做优化了，只是用来优化后的 L：

```
unsigned long T(unsigned long x) {
    unsigned long b = 0;
    for (int i = 0; i < 4; i++) {
        b = (b << 8) | sm4Sbox((x >> ((3 - i) * 8)) & 0xFF);
    }
    __m128i val = _mm_set_epi32(0, 0, 0, b);
    __m128i res = L_simd(val);
    return _mm_cvtsi128_si32(res);
}
```

3.3 优化方法 3：AES-NI 优化

由于 AES-NI 中的 _mm_aesenclast_si128 可以用来硬件加速但是由于他本身只适用 AES，所以我们对他进行转换后来加速 SM4：

先通过字节置换（用 _mm_shuffle_epi8）把 SM4 输入字节映射到 AES S 盒输入；

```
    __m128i mask = _mm_set_epi8(3, 6, 9, 12, 15, 2, 5, 8,
        11, 14, 1, 4, 7, 10, 13, 0);
    x = _mm_shuffle_epi8(x, mask);

    x = _mm_xor_si128(MulMatrixToAES(x), _mm_set1_epi8(0x23));
```

用 AES-NI 指令 _mm_aesenclast_si128 执行硬件 AES S 盒；

```
    x = _mm_aesenclast_si128(x, _mm_setzero_si128());
```

再通过另一个矩阵变换（MulMatrixBack）把 AES S 盒的结果“映射回”SM4 的 S 盒输出。

```
    x = _mm_xor_si128(MulMatrixBack(x), _mm_set1_epi8(0x3B));
```

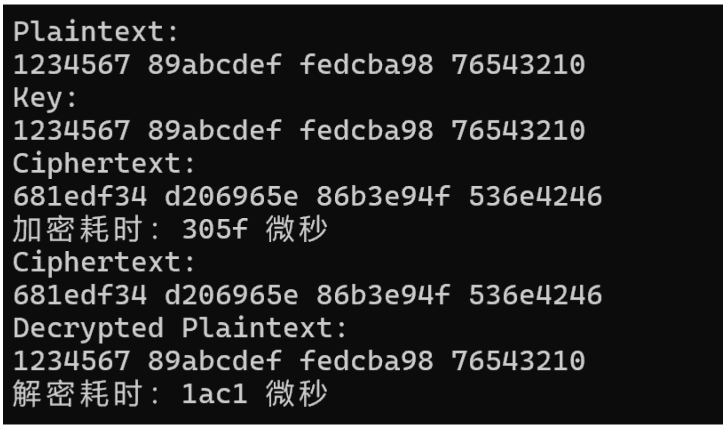
变化相关的函数如下所示：

```
inline __m128i MulMatrix(__m128i x, __m128i higherMask, __m128i lowerMask) {
    // 取x低4位
    __m128i low_nibble = _mm_and_si128(x, _mm_set1_epi32(0x0f0f0f0f));
    // 取x高4位（每16位右移4位后掩码）
    __m128i high_nibble = _mm_and_si128(_mm_srli_epi16(x, 4), _mm_set1_epi32(0x0f0f0f0f));

    // 分别对高低4位用mask做字节置换查表
    __m128i low_part = _mm_shuffle_epi8(lowerMask, low_nibble);
    __m128i high_part = _mm_shuffle_epi8(higherMask, high_nibble);

    // 异或合并返回
    return _mm_xor_si128(low_part, high_part);
}
inline static __m128i MulMatrixToAES(__m128i x) {
    __m128i higherMask = _mm_set_epi8(0x22, 0x58, 0x1a, 0x60, 0x02, 0x78, 0x3a, 0x40, 0x62, 0x18,
        0x5a, 0x20, 0x42, 0x38, 0x7a, 0x00);
    __m128i lowerMask = _mm_set_epi8(0xe2, 0x28, 0x95, 0x5f, 0x69, 0xa3, 0x1e, 0xd4, 0x36, 0xfc,
        0x41, 0x8b, 0xbd, 0x77, 0xca, 0x00);
    return MulMatrix(x, higherMask, lowerMask);
}

inline static __m128i MulMatrixBack(__m128i x) {
    __m128i higherMask = _mm_set_epi8(0x14, 0x07, 0xc6, 0xd5, 0x6c, 0x7f, 0
        xbe, 0xad, 0xb9, 0xaa,
        0x6b, 0x78, 0xc1, 0xd2, 0x13, 0x00);
    __m128i lowerMask = _mm_set_epi8(0xd8, 0xb8, 0xfa, 0x9a, 0xc5, 0xa5, 0
        xe7, 0x87, 0x5f, 0x3f,
        0x7d, 0x1d, 0x42, 0x22, 0x60, 0x00);
    return MulMatrix(x, higherMask, lowerMask);
}
```

在这个指令下主要是优化了 S 盒查表的速度。

3.4 时间对比

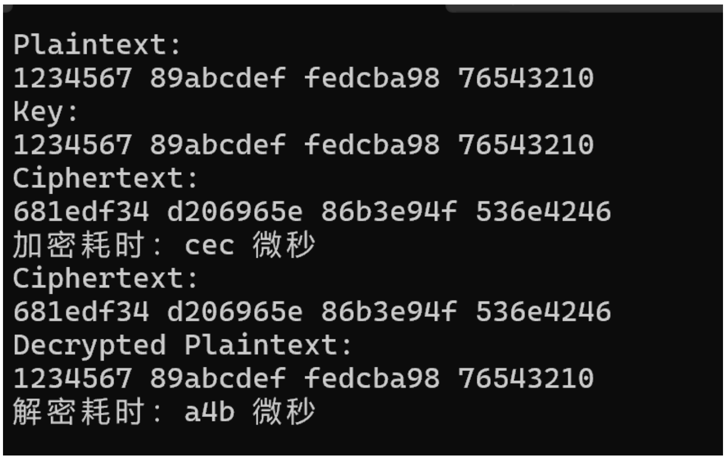
原始代码结果如下所示:



```
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
加密耗时： 305f 微秒
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210
解密耗时： 1ac1 微秒
```

图 6 原始代码运行结果

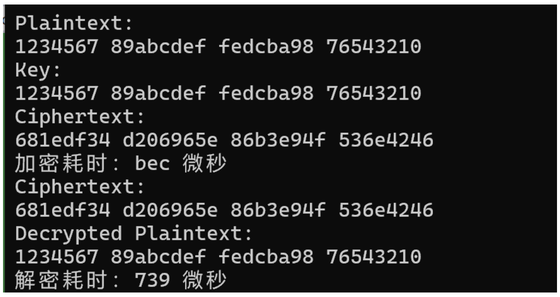
优化后的结果如下所示:



```
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
加密耗时: cec 微秒
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210
解密耗时: a4b 微秒
```

图 7 优化代码运行结果

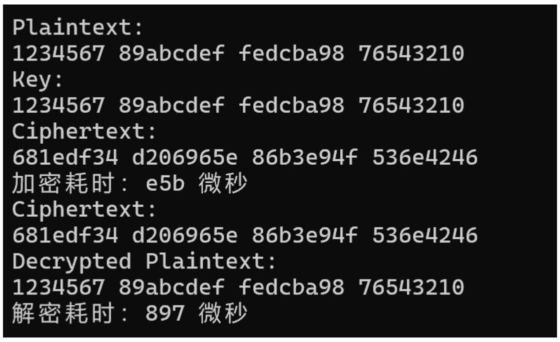
我们可以看到优化方法 1 加密速度提升将近 4 倍，解密速度提升大概 2.5 倍。优化方法 2 的结果差不多，如下图所示：



```
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
加密耗时: bec 微秒
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210
解密耗时: 739 微秒
```

图 8 优化 2 运行结果

优化 3 的结果如下所示：



```
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
加密耗时: e5b 微秒
Ciphertext:
681edf34 d206965e 86b3e94f 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210
解密耗时: 897 微秒
```

图 9 优化 3 运行结果

可以看到优化方法 2 能达到的最快速度更快一点，优化方案 3 的速度不太高，要是和优化 2 结合效率应该会更好（总体 SIMD 方法更快）。

4 SM4-GCM 工作模式优化

GCM 是基于块加密的认证加密模式，SM4 块加密的代码实现在基础实现中已经实现，而 GCM 的核心是：

使用块加密对 IV（初始向量）生成 GHASH 的哈希密钥 H（H = E(K, 0□128)），然后使用计数器模式 (CTR) 对明文加密:

```
// CTR 模式加解密
void increment_counter(uint8_t counter[16]) {
    for (int i = 15; i >= 12; i--) {
        if (++counter[i]) break;
    }
}

void sm4_ctr_crypt(const uint8_t key[16], uint8_t counter[16], const uint8_t* input, uint8_t* output, size_t length) {
    uint8_t block[16];
    size_t blocks = (length + 15) / 16;
    for (size_t i = 0; i < blocks; ++i) {
        sm4_encrypt_block(counter, key, block);
        size_t offset = i * 16;
        size_t block_len = std::min(size_t(16), length - offset);
        for (size_t j = 0; j < block_len; ++j)
            output[offset + j] = input[offset + j] ^ block[j];
        increment_counter(counter);
    }
}
```

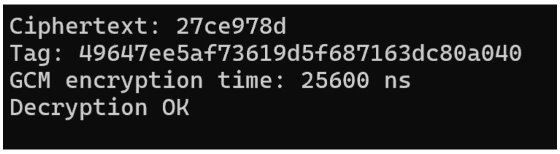
GHASH 用于消息认证码计算，基于 Galois 域的乘法

```
void galois_multiply(uint8_t* X, const uint8_t* Y) {
    uint8_t Z[16] = { 0 };
    uint8_t V[16];
    memcpy(V, Y, 16);

    for (int i = 0; i < 128; ++i) {
        int bit = (X[i / 8] >> (7 - (i % 8))) & 1;
        if (bit)
            for (int j = 0; j < 16; ++j) Z[j] ^= V[j];

        uint8_t lsb = V[15] & 1;
        for (int j = 15; j > 0; --j) V[j] = (V[j] >> 1) | ((V[j - 1] & 1) << 7);
        V[0] >>= 1;
        if (lsb) V[0] ^= 0xe1;
    }
    memcpy(X, Z, 16);
}

void ghash(const uint8_t H[16], const uint8_t* aad, size_t aad_len,
    const uint8_t* ciphertext, size_t ct_len, uint8_t tag[16]) {
    uint8_t Y[16] = { 0 };

    for (size_t i = 0; i < aad_len; i += 16) {
        uint8_t block[16] = { 0 };
        size_t len = std::min(size_t(16), aad_len - i);
        memcpy(block, aad + i, len);
        for (int j = 0; j < 16; ++j) Y[j] ^= block[j];
        galois_multiply(Y, H);
    }

    for (size_t i = 0; i < ct_len; i += 16) {
        uint8_t block[16] = { 0 };
        size_t len = std::min(size_t(16), ct_len - i);
        memcpy(block, ciphertext + i, len);
        for (int j = 0; j < 16; ++j) Y[j] ^= block[j];
        galois_multiply(Y, H);
    }

    uint8_t len_block[16] = { 0 };
    uint64_t aad_bits = aad_len * 8;
    uint64_t ct_bits = ct_len * 8;
    for (int i = 0; i < 8; ++i) len_block[7 - i] = (aad_bits >> (i * 8)) & 0xFF;
    for (int i = 0; i < 8; ++i) len_block[15 - i] = (ct_bits >> (i * 8)) & 0xFF;

    for (int j = 0; j < 16; ++j) Y[j] ^= len_block[j];
    galois_multiply(Y, H);
    memcpy(tag, Y, 16);
}
```

同时使用了内联函数优化:

```
inline unsigned long L(unsigned long x) {
    return x ^ ((x << 2) | (x >> (32 - 2)))
        ^ ((x << 10) | (x >> (32 - 10)))
        ^ ((x << 18) | (x >> (32 - 18)))
        ^ ((x << 24) | (x >> (32 - 24)));
}
```

在 main 函数中，我们定义明文，key，add 附加信息，iv，然后测试时间并验证解密结果：



```
Ciphertext: 27ce978d
Tag: 49647ee5af73619d5f687163dc80a040
GCM encryption time: 25600 ns
Decryption OK
```

图 10 SM4-gcm 模式