



山东大学  
SHANDONG UNIVERSITY

## 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 7 月 13 日

# 目录

<b>1</b>	<b>实验任务</b>	<b>1</b>
<b>2</b>	<b>SM4 软件实现</b>	<b>1</b>
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
<b>3</b>	<b>SM4 算法优化</b>	<b>9</b>
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 . . . . .	27
3.3	时间对比 . . . . .	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：

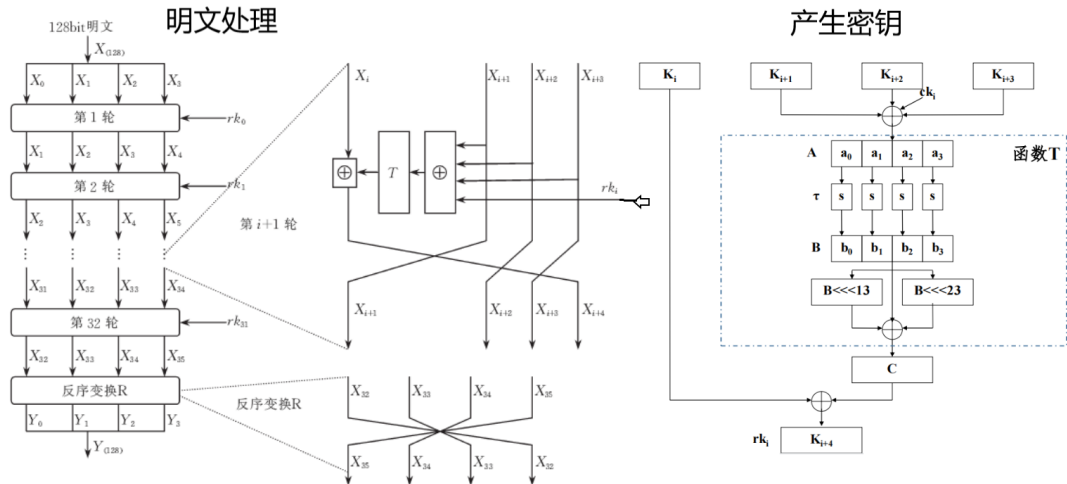


图 1 SMS4 加密算法整体结构

[https://blog.csdn.net/qg\\_41610725](https://blog.csdn.net/qg_41610725)

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  $X_1, X_2, X_3, X_4$ 。
- 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。

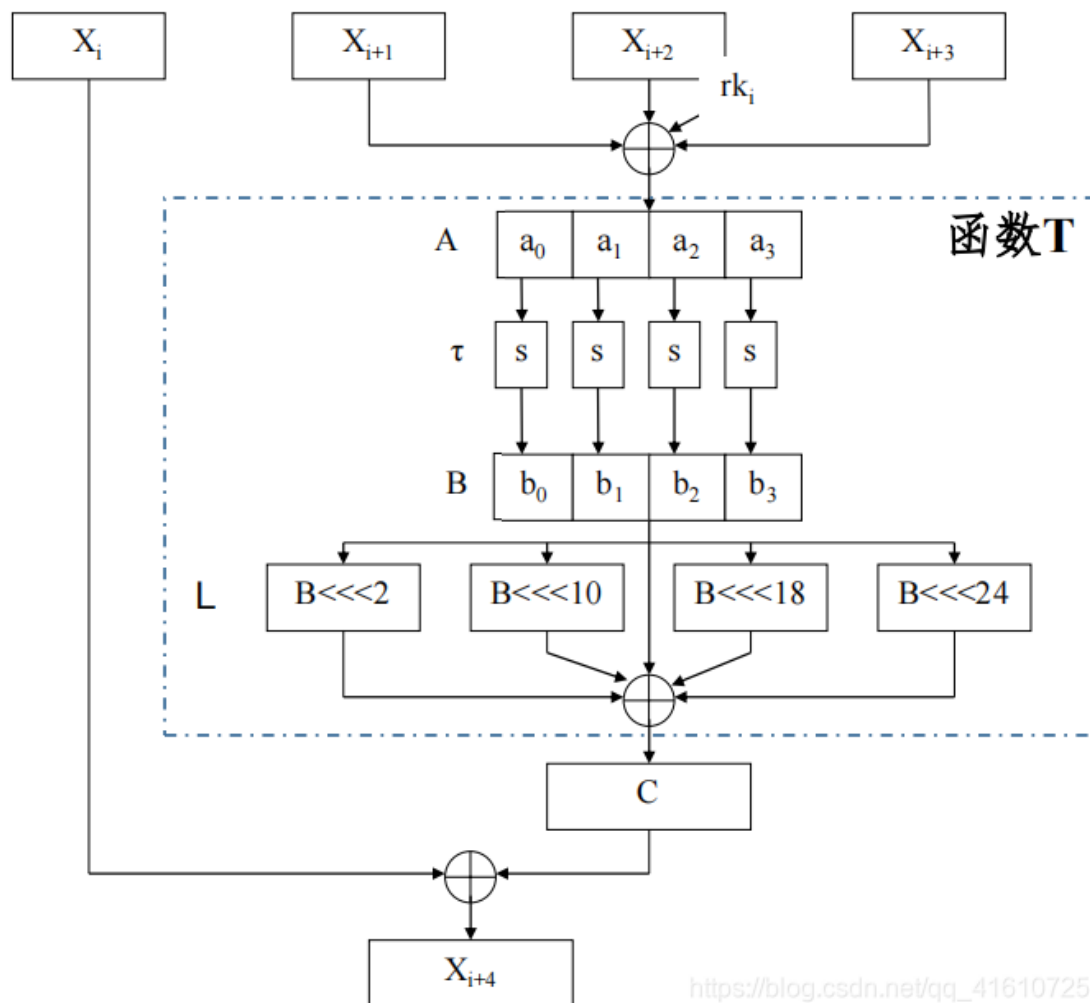


图 2 轮操作

## 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 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  $rk31$ 。

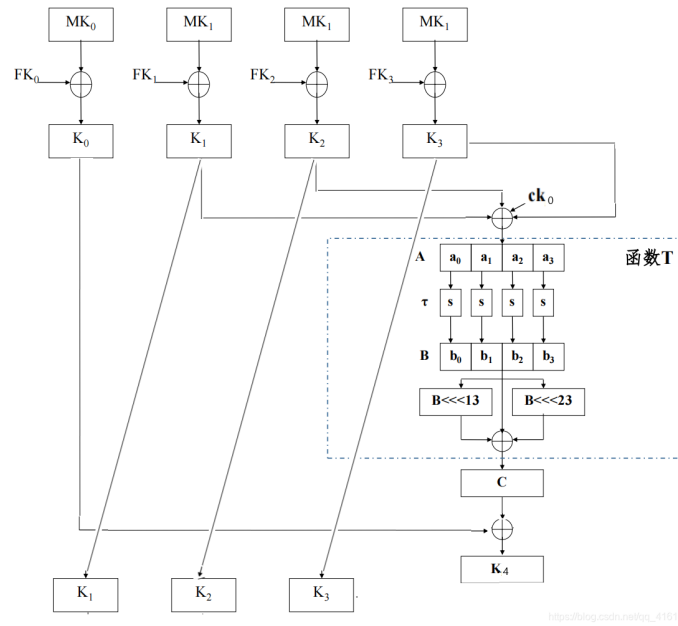


图 3 初始密钥拓展

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：

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

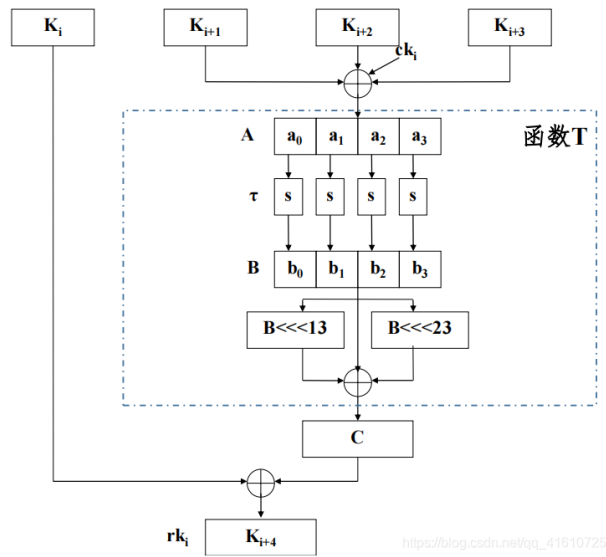


图 4 后续密钥拓展操作

## 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：

```
1 #include<iostream>
```

```

2 using namespace std;
3
4 //Round = 32 轮数
5
6 //S盒
7 static const unsigned long SboxTable[16][16] = {
8     {0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2,
9       0x28, 0xfb, 0x2c, 0x05},
10    {0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26,
11      0x49, 0x86, 0x06, 0x99},
12    {0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43,
13      0xed, 0xcf, 0xac, 0x62},
14    {0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa,
15      0x75, 0x8f, 0x3f, 0xa6},
16    {0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19,
17      0xe6, 0x85, 0x4f, 0xa8},
18    {0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b,
19      0x70, 0x56, 0x9d, 0x35},
20    {0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b,
21      0x01, 0x21, 0x78, 0x87},
22    {0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7,
23      0xa0, 0xc4, 0xc8, 0x9e},
24    {0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce,
25      0xf9, 0x61, 0x15, 0xa1},
26    {0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30,
27      0xf5, 0x8c, 0xb1, 0xe3},
28    {0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab,
29      0x0d, 0x53, 0x4e, 0x6f},
30    {0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72,
31      0x6d, 0x6c, 0x5b, 0x51},
32    {0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41,
33      0x1f, 0x10, 0x5a, 0xd8},
34    {0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12,
35      0xb8, 0xe5, 0xb4, 0xb0},
36    {0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09,
37      0xc5, 0x6e, 0xc6, 0x84},
38    {0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e,
39      0xd7, 0xcb, 0x39, 0x48}
40 };
41
42 unsigned long sm4Sbox(unsigned long in) {
43     return SboxTable[(in >> 4) & 0x0F][in & 0x0F];
44 }

```

```

29
30 //线性变换L
31 unsigned long L(unsigned long x) {
32     return x ^ (x << 2 | x >> (32 - 2)) ^ (x << 10 | x >> (32 - 10)) ^ (x <<
        18 | x >> (32 - 18)) ^ (x << 24 | x >> (32 - 24));
33 }
34
35 //非线性T变换
36 unsigned long T(unsigned long x) {
37     unsigned long b = 0;
38     for (int i = 0; i < 4; i++) {
39         b = (b << 8) | sm4Sbox((x >> ((3 - i) * 8)) & 0xFF);
40     }
41     return L(b);
42 }
43
44 //系统参数
45 static const unsigned long FK[4] = { 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0
    xb27022dc };
46
47 //固定参数 CK
48 static const unsigned long CK[32] = {
49     0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
50     0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
51     0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
52     0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
53     0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
54     0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
55     0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
56     0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
57 };
58
59 //密钥扩展
60 void key_expansion(unsigned long MK[4], unsigned long rk[32]) {
61     unsigned long K[36];
62     for (int i = 0; i < 4; i++)
63         K[i] = MK[i] ^ FK[i];
64
65     for (int i = 0; i < 32; i++) {
66         unsigned long tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];
67         unsigned long b = 0;
68         for (int j = 0; j < 4; j++)
69             b = (b << 8) | sm4Sbox((tmp >> ((3 - j) * 8)) & 0xFF);

```

```

70         unsigned long L = b ^ (b << 13 | b >> (32 - 13)) ^ (b << 23 | b >>
              (32 - 23));
71         rk[i] = K[i] ^ L;
72         K[i + 4] = rk[i];
73     }
74 }
75
76 //轮操作
77 unsigned long round_operate(int i, unsigned long* X, unsigned long* rk) {
78     return X[i] ^ T(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i]);
79 }
80
81 //加密函数
82 void sm4_enc(unsigned long MK[4], unsigned long X[4]) {
83     cout << hex;
84     cout << "Plaintext:" << endl;
85     cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;
86
87     cout << hex;
88     cout << "Key:" << endl;
89     cout << MK[0] << " " << MK[1] << " " << MK[2] << " " << MK[3] << endl;
90
91     unsigned long rk[32];
92     key_expansion(MK, rk);
93
94     for (int i = 0; i < 32; i++) {
95         unsigned long tmp = round_operate(i, X, rk);
96         X[4 + i] = tmp;
97     }
98
99     cout << hex;
100    cout << "Ciphertext:" << endl;
101    cout << X[35] << " " << X[34] << " " << X[33] << " " << X[32] << endl;
102 }
103
104 //解密函数
105 void sm4_dec(unsigned long MK[4], unsigned long X[4]) {
106     cout << hex;
107     cout << "Ciphertext:" << endl;
108     cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;
109
110     unsigned long rk[32];
111     key_expansion(MK, rk);

```



```

112
113     //反转轮密钥
114     for (int i = 0; i < 16; i++) swap(rk[i], rk[31 - i]);
115
116     unsigned long tmpX[36] = { 0 };
117     for (int i = 0; i < 4; i++) tmpX[i] = X[i];
118
119     for (int i = 0; i < 32; i++) {
120         tmpX[i + 4] = round_operate(i, tmpX, rk);
121     }
122
123     cout << "Decrypted Plaintext:" << endl;
124     cout << tmpX[35] << " " << tmpX[34] << " " << tmpX[33] << " " << tmpX
        [32] << endl;
125 }
126
127 void copy_block(unsigned long* dst, unsigned long* src) {
128     for (int i = 0; i < 4; i++) dst[i] = src[i];
129 }
130
131 void sm4_ecb_enc(unsigned long MK[4], unsigned long* data, int blocks) {
132     for (int i = 0; i < blocks; i++) {
133         sm4_enc(MK, &data[i * 4]);
134     }
135 }
136
137 void sm4_ecb_dec(unsigned long MK[4], unsigned long* data, int blocks) {
138     for (int i = 0; i < blocks; i++) {
139         sm4_dec(MK, &data[i * 4]);
140     }
141 }
142
143 void xor_block(unsigned long* dst, unsigned long* src) {
144     for (int i = 0; i < 4; i++) dst[i] ^= src[i];
145 }
146
147 void sm4_cbc_enc(unsigned long MK[4], unsigned long* data, int blocks,
    unsigned long IV[4]) {
148     unsigned long last_block[4];
149     copy_block(last_block, IV);
150
151     for (int i = 0; i < blocks; i++) {
152         xor_block(&data[i * 4], last_block);

```

```

153     sm4_enc(MK, &data[i * 4]);
154     copy_block(last_block, &data[i * 4]);
155 }
156 }
157
158 void sm4_cbc_dec(unsigned long MK[4], unsigned long* data, int blocks,
159     unsigned long IV[4]) {
160     unsigned long last_block[4];
161     copy_block(last_block, IV);
162
163     for (int i = 0; i < blocks; i++) {
164         unsigned long tmp[4];
165         copy_block(tmp, &data[i * 4]);
166
167         sm4_dec(MK, &data[i * 4]);
168         xor_block(&data[i * 4], last_block);
169
170         copy_block(last_block, tmp);
171     }
172 }
173
174 int main() {
175     unsigned long MK[4] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210
176         };//加 密密 钥
177     unsigned long X[36] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210
178         };//明 文
179     unsigned long C[36] = { 0x681edf34, 0xd206965e, 0x86b3e94f, 0x536e4246
180         };//密 文
181     sm4_enc(MK, X);
182     sm4_dec(MK, C);
183
184     cout << "\n== ECB Mode Test ==" << endl;
185     unsigned long ecb_data[8] = {
186         0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,
187         0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff
188     };
189     sm4_ecb_enc(MK, ecb_data, 2);
190     sm4_ecb_dec(MK, ecb_data, 2);
191
192     cout << "\n== CBC Mode Test ==" << endl;
193     unsigned long cbc_data[8] = {
194         0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,
195         0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff

```

```

192     };
193     unsigned long IV[4] = { 0x00000000, 0x00000000, 0x00000000, 0x00000000
    };
194     sm4_cbc_enc(MK, cbc_data, 2, IV);
195     sm4_cbc_dec(MK, cbc_data, 2, IV);
196
197     return 0;
198 }

```

## 2.5 实现结果

运行结果如下图所示：

```

C:\Users\20444\source\repos' x + v
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf3a d206965e 86b3e9df 536e4246
Ciphertext:
681edf3a d206965e 86b3e9df 536e4246
Decrypted Plaintext:
1234567 89abcdef fedcba98 76543210
== ECB Mode Test ==
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf3a d206965e 86b3e9df 536e4246
Plaintext:
77fad305 a18b4cb2 11c1e22a cc13e2ee
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
c590ce9 ab12054c 6c7707c0 5bed2c2d
Ciphertext:
1234567 89abcdef fedcba98 76543210
Decrypted Plaintext:
9338375e 82eeclcb bef65edf 8757a631
Ciphertext:
77fad305 a18b4cb2 11c1e22a cc13e2ee
Decrypted Plaintext:
9ccf13ed 580877f0 f0e435a9 b22ef6f
== CBC Mode Test ==
Plaintext:
1234567 89abcdef fedcba98 76543210
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
681edf3a d206965e 86b3e9df 536e4246
Plaintext:
26d99622 2820815d efd58b2 ba47d0fe
Key:
1234567 89abcdef fedcba98 76543210
Ciphertext:
4b0bde 40505f6 47711189 18f476e5
Ciphertext:
1234567 89abcdef fedcba98 76543210
Decrypted Plaintext:
9338375e 82eeclcb bef65edf 8757a631
Ciphertext:
26d99622 2820815d efd58b2 ba47d0fe
Decrypted Plaintext:
ba8fd0b 2bae0b4 e1528b99 a7689b5

```

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  $x_0, x_1, x_2, x_3 \rightarrow S(x_0), S(x_1), S(x_2), S(x_3)$ ，其中  $x_i$  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  $\rightarrow$  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**  $i = 0 \rightarrow 31$
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。

1	T0:				
2	8ed55b5b	d0924242	4deaa7a7	06fdfbfb	fccf3333
		65e28787	c93df4f4	6bb5dede	
3	4e165858	6eb4dada	44145050	cac10b0b	8828a0a0
		17f8efef	9c2cb0b0	11051414	
4	872bacac	fb669d9d	f2986a6a	ae77d9d9	822aa8a8
		46bcfafa	14041010	cfc00f0f	
5	02a8aaaa	54451111	5f134c4c	be269898	6d482525
		9e841a1a	1e061818	fd9b6666	
6	ec9e7272	4a430909	10514141	24f7d3d3	d5934646
		53ecbfbf	f89a6262	927be9e9	

7	ff33cccc	04555151	270b2c2c	4f420d0d	59eeb7b7
		f3cc3f3f	1caeb2b2	ea638989	
8	74e79393	7fb1cece	6c1c7070	0daba6a6	edca2727
		28082020	48eba3a3	c1975656	
9	80820202	a3dc7f7f	c4965252	12f9ebeb	a174d5d5
		b38d3e3e	c33ffcfc	3ea49a9a	
10	5b461d1d	1b071c1c	3ba59e9e	0cfff3f3	3ff0cfcf
		bf72cdcd	4b175c5c	52b8eaea	
11	8f810e0e	3d586565	cc3cf0f0	7d196464	7ee59b9b
		91871616	734e3d3d	08aaa2a2	
12	c869a1a1	c76aadad	85830606	7ab0caca	b570c5c5
		f4659191	b2d96b6b	a7892e2e	
13	18fbe3e3	47e8afaf	330f3c3c	674a2d2d	b071c1c1
		0e575959	e99f7676	e135d4d4	
14	661e7878	b4249090	360e3838	265f7979	ef628d8d
		38596161	95d24747	2aa08a8a	
15	b1259494	aa228888	8c7df1f1	d73becec	05010404
		a5218484	9879e1e1	9b851e1e	
16	84d75353	00000000	5e471919	0b565d5d	e39d7e7e
		9fd04f4f	bb279c9c	1a534949	
17	7c4d3131	ee36d8d8	0a020808	7be49f9f	20a28282
		d4c71313	e8cb2323	e69c7a7a	
18	42e9abab	43bdfefe	a2882a2a	9ad14b4b	40410101
		dbc41f1f	d838e0e0	61b7d6d6	
19	2fa18e8e	2bf4dfdf	3af1cbcb	f6cd3b3b	1dfae7e7
		e5608585	41155454	25a38686	
20	60e38383	16acbaba	295c7575	34a69292	f7996e6e
		e434d0d0	721a6868	01545555	
21	19afb6b6	df914e4e	fa32c8c8	f030c0c0	21f6d7d7
		bc8e3232	75b3c6c6	6fe08f8f	
22	691d7474	2ef5dbdb	6ae18b8b	962eb8b8	8a800a0a
		fe679999	e2c92b2b	e0618181	
23	c0c30303	8d29a4a4	af238c8c	07a9aeae	390d3434
		1f524d4d	764f3939	d36ebdbd	
24	81d65757	b7d86f6f	eb37dcdc	51441515	a6dd7b7b
		09fef7f7	b68c3a3a	932fbcbc	
25	0f030c0c	03fcffff	c26ba9a9	ba73c9c9	d96cb5b5
		dc6db1b1	375a6d6d	15504545	
26	b98f3636	771b6c6c	13adbebe	da904a4a	57b9eeee
		a9de7777	4cbef2f2	837efdfd	
27	55114444	bdda6767	2c5d7171	45400505	631f7c7c
		50104040	325b6969	b8db6363	
28	220a2828	c5c20707	f531c4c4	a88a2222	31a79696

		f9ce3737	977aeded	49bff6f6	
29	992db4b4	a475d1d1	90d34343	5a124848	58bae2e2
		71e69797	64b6d2d2	70b2c2c2	
30	ad8b2626	cd68a5a5	cb955e5e	624b2929	3c0c3030
		ce945a5a	ab76dddd	867ff9f9	
31	f1649595	5dbbe6e6	35f2c7c7	2d092424	d1c61717
		d66fb9b9	dec51b1b	94861212	
32	78186060	30f3c3c3	897cf5f5	5cefb3b3	d23ae8e8
		acdf7373	794c3535	a0208080	
33	9d78e5e5	56edbbbb	235e7d7d	c63ef8f8	8bd45f5f
		e7c82f2f	dd39e4e4	68492121	
34	T1:				
35	5b8ed55b	42d09242	a74deaa7	fb06fdfb	33fccf33
		8765e287	f4c93df4	de6bb5de	
36	584e1658	da6eb4da	50441450	0bcac10b	a08828a0
		ef17f8ef	b09c2cb0	14110514	
37	ac872bac	9dfb669d	6af2986a	d9ae77d9	a8822aa8
		fa46bcfa	10140410	0fcfc00f	
38	aa02a8aa	11544511	4c5f134c	98be2698	256d4825
		1a9e841a	181e0618	66fd9b66	
39	72ec9e72	094a4309	41105141	d324f7d3	46d59346
		bf53ecbf	62f89a62	e9927be9	
40	ccff33cc	51045551	2c270b2c	0d4f420d	b759eeb7
		3ff3cc3f	b21caeb2	89ea6389	
41	9374e793	ce7fb1ce	706c1c70	a60daba6	27edca27
		20280820	a348eba3	56c19756	
42	02808202	7fa3dc7f	52c49652	eb12f9eb	d5a174d5
		3eb38d3e	fcc33ffc	9a3ea49a	
43	1d5b461d	1c1b071c	9e3ba59e	f30cfff3	cf3ff0cf
		cdbf72cd	5c4b175c	ea52b8ea	
44	0e8f810e	653d5865	f0cc3cf0	647d1964	9b7ee59b
		16918716	3d734e3d	a208aaa2	
45	a1c869a1	adc76aad	06858306	ca7ab0ca	c5b570c5
		91f46591	6bb2d96b	2ea7892e	
46	e318fbe3	af47e8af	3c330f3c	2d674a2d	c1b071c1
		590e5759	76e99f76	d4e135d4	
47	78661e78	90b42490	38360e38	79265f79	8def628d
		61385961	4795d247	8a2aa08a	
48	94b12594	88aa2288	f18c7df1	ecd73bec	04050104
		84a52184	e19879e1	1e9b851e	
49	5384d753	00000000	195e4719	5d0b565d	7ee39d7e
		4f9fd04f	9cbb279c	491a5349	
50	317c4d31	d8ee36d8	080a0208	9f7be49f	8220a282

		13d4c713	23e8cb23	7ae69c7a	
51	ab42e9ab	fe43bdfe	2aa2882a	4b9ad14b	01404101
		1fdbbc41f	e0d838e0	d661b7d6	
52	8e2fa18e	df2bf4df	cb3af1cb	3bf6cd3b	e71dfae7
		85e56085	54411554	8625a386	
53	8360e383	ba16acba	75295c75	9234a692	6ef7996e
		d0e434d0	68721a68	55015455	
54	b619afb6	4edf914e	c8fa32c8	c0f030c0	d721f6d7
		32bc8e32	c675b3c6	8f6fe08f	
55	74691d74	db2ef5db	8b6ae18b	b8962eb8	0a8a800a
		99fe6799	2be2c92b	81e06181	
56	03c0c303	a48d29a4	8caf238c	ae07a9ae	34390d34
		4d1f524d	39764f39	bdd36ebd	
57	5781d657	6fb7d86f	dceb37dc	15514415	7ba6dd7b
		f709fef7	3ab68c3a	bc932fbc	
58	0c0f030c	ff03fcff	a9c26ba9	c9ba73c9	b5d96cb5
		b1dc6db1	6d375a6d	45155045	
59	36b98f36	6c771b6c	be13adbe	4ada904a	ee57b9ee
		77a9de77	f24cbef2	fd837efd	
60	44551144	67bdda67	712c5d71	05454005	7c631f7c
		40501040	69325b69	63b8db63	
61	28220a28	07c5c207	c4f531c4	22a88a22	9631a796
		37f9ce37	ed977aed	f649bff6	
62	b4992db4	d1a475d1	4390d343	485a1248	e258bae2
		9771e697	d264b6d2	c270b2c2	
63	26ad8b26	a5cd68a5	5ecb955e	29624b29	303c0c30
		5ace945a	ddab76dd	f9867ff9	
64	95f16495	e65dbbe6	c735f2c7	242d0924	17d1c617
		b9d66fb9	1bdec51b	12948612	
65	60781860	c330f3c3	f5897cf5	b35cefb3	e8d23ae8
		73acdf73	35794c35	80a02080	
66	e59d78e5	bb56edbb	7d235e7d	f8c63ef8	5f8bd45f
		2fe7c82f	e4dd39e4	21684921	
67	T2:				
68	5b5b8ed5	4242d092	a7a74dea	fbfb06fd	3333fccf
		878765e2	f4f4c93d	dede6bb5	
69	58584e16	dada6eb4	50504414	0b0bcac1	a0a08828
		efef17f8	b0b09c2c	14141105	
70	acac872b	9d9dfb66	6a6af298	d9d9ae77	a8a8822a
		fafa46bc	10101404	0f0fcfc0	
71	aaaa02a8	11115445	4c4c5f13	9898be26	25256d48
		1a1a9e84	18181e06	6666fd9b	
72	7272ec9e	09094a43	41411051	d3d324f7	4646d593

		bfbf53ec	6262f89a	e9e9927b	
73	ccccff33	51510455	2c2c270b	0d0d4f42	b7b759ee
		3f3ff3cc	b2b21cae	8989ea63	
74	939374e7	cece7fb1	70706c1c	a6a60dab	2727edca
		20202808	a3a348eb	5656c197	
75	02028082	7f7fa3dc	5252c496	ebeb12f9	d5d5a174
		3e3eb38d	fcfcc33f	9a9a3ea4	
76	1d1d5b46	1c1c1b07	9e9e3ba5	f3f30cff	cfcf3ff0
		cdcdbf72	5c5c4b17	eaea52b8	
77	0e0e8f81	65653d58	f0f0cc3c	64647d19	9b9b7ee5
		16169187	3d3d734e	a2a208aa	
78	a1a1c869	adadc76a	06068583	caca7ab0	c5c5b570
		9191f465	6b6bb2d9	2e2ea789	
79	e3e318fb	afaf47e8	3c3c330f	2d2d674a	c1c1b071
		59590e57	7676e99f	d4d4e135	
80	7878661e	9090b424	3838360e	7979265f	8d8def62
		61613859	474795d2	8a8a2aa0	
81	9494b125	8888aa22	f1f18c7d	ececd73b	04040501
		8484a521	e1e19879	1e1e9b85	
82	535384d7	00000000	19195e47	5d5d0b56	7e7ee39d
		4f4f9fd0	9c9cbb27	49491a53	
83	31317c4d	d8d8ee36	08080a02	9f9f7be4	828220a2
		1313d4c7	2323e8cb	7a7ae69c	
84	abab42e9	fe4e43bd	2a2aa288	4b4b9ad1	01014041
		1f1fdbcb4	e0e0d838	d6d661b7	
85	8e8e2fa1	dfdf2bf4	cbcb3af1	3b3bf6cd	e7e71dfa
		8585e560	54544115	868625a3	
86	838360e3	baba16ac	7575295c	929234a6	6e6ef799
		d0d0e434	6868721a	55550154	
87	b6b619af	4e4edf91	c8c8fa32	c0c0f030	d7d721f6
		3232bc8e	c6c675b3	8f8f6fe0	
88	7474691d	dbdb2ef5	8b8b6ae1	b8b8962e	0a0a8a80
		9999fe67	2b2be2c9	8181e061	
89	0303c0c3	a4a48d29	8c8caf23	aeae07a9	3434390d
		4d4d1f52	3939764f	bdbdd36e	
90	575781d6	6f6fb7d8	dcdceb37	15155144	7b7ba6dd
		f7f709fe	3a3ab68c	bc932f	
91	0c0c0f03	ffff03fc	a9a9c26b	c9c9ba73	b5b5d96c
		b1b1dc6d	6d6d375a	45451550	
92	3636b98f	6c6c771b	bebe13ad	4a4ada90	eeee57b9
		7777a9de	f2f24cbe	fdfd837e	
93	44445511	6767bdda	71712c5d	05054540	7c7c631f
		40405010	6969325b	6363b8db	



94	2828220a	0707c5c2	c4c4f531	2222a88a	969631a7
		3737f9ce	eded977a	f6f649bf	
95	b4b4992d	d1d1a475	434390d3	48485a12	e2e258ba
		979771e6	d2d264b6	c2c270b2	
96	2626ad8b	a5a5cd68	5e5ecb95	2929624b	30303c0c
		5a5ace94	dddab76	f9f9867f	
97	9595f164	e6e65dbb	c7c735f2	24242d09	1717d1c6
		b9b9d66f	1b1bdec5	12129486	
98	60607818	c3c330f3	f5f5897c	b3b35cef	e8e8d23a
		7373acdf	3535794c	8080a020	
99	e5e59d78	bbbb56ed	7d7d235e	f8f8c63e	5f5f8bd4
		2f2fe7c8	e4e4dd39	21216849	
100	T3:				
101	d55b5b8e	924242d0	ea7a74d	fdfbfb06	cf3333fc
		e2878765	3df4f4c9	b5dede6b	
102	1658584e	b4dada6e	14505044	c10b0bca	28a0a088
		f8efef17	2cb0b09c	05141411	
103	2bacac87	669d9dfb	986a6af2	77d9d9ae	2aa8a882
		bcfafa46	04101014	c00f0fcf	
104	a8aaaa02	45111154	134c4c5f	269898be	4825256d
		841a1a9e	0618181e	9b6666fd	
105	9e7272ec	4309094a	51414110	f7d3d324	934646d5
		ecbfbf53	9a6262f8	7be9e992	
106	33ccccff	55515104	0b2c2c27	420d0d4f	eeb7b759
		cc3f3ff3	aeb2b21c	638989ea	
107	e7939374	b1cece7f	1c70706c	aba6a60d	ca2727ed
		08202028	eba3a348	975656c1	
108	82020280	dc7f7fa3	965252c4	f9ebeb12	74d5d5a1
		8d3e3eb3	3ffcfcc3	a49a9a3e	
109	461d1d5b	071c1c1b	a59e9e3b	fff3f30c	f0cfcf3f
		72cdcdbf	175c5c4b	b8eaea52	
110	810e0e8f	5865653d	3cf0f0cc	1964647d	e59b9b7e
		87161691	4e3d3d73	aaa2a208	
111	69a1a1c8	6aadadc7	83060685	b0caca7a	70c5c5b5
		659191f4	d96b6bb2	892e2ea7	
112	fbe3e318	e8afaf47	0f3c3c33	4a2d2d67	71c1c1b0
		5759590e	9f7676e9	35d4d4e1	
113	1e787866	249090b4	0e383836	5f797926	628d8def
		59616138	d2474795	a08a8a2a	
114	259494b1	228888aa	7df1f18c	3bececd7	01040405
		218484a5	79e1e198	851e1e9b	
115	d7535384	00000000	4719195e	565d5d0b	9d7e7ee3
		d04f4f9f	279c9cbb	5349491a	

116	4d31317c	36d8d8ee	0208080a	e49f9f7b	a2828220
		c71313d4	cb2323e8	9c7a7ae6	
117	e9abab42	bdfefe43	882a2aa2	d14b4b9a	41010140
		c41f1fdb	38e0e0d8	b7d6d661	
118	a18e8e2f	f4dfdf2b	f1cbcb3a	cd3b3bf6	fae7e71d
		608585e5	15545441	a3868625	
119	e3838360	acbaba16	5c757529	a6929234	996e6ef7
		34d0d0e4	1a686872	54555501	
120	afb6b619	914e4edf	32c8c8fa	30c0c0f0	f6d7d721
		8e3232bc	b3c6c675	e08f8f6f	
121	1d747469	f5dbdb2e	e18b8b6a	2eb8b896	800a0a8a
		679999fe	c92b2be2	618181e0	
122	c30303c0	29a4a48d	238c8caf	a9aeae07	0d343439
		524d4d1f	4f393976	6ebdbdd3	
123	d6575781	d86f6fb7	37dcdceb	44151551	dd7b7ba6
		fef7f709	8c3a3ab6	2fbc93	
124	030c0c0f	fcffff03	6ba9a9c2	73c9c9ba	6cb5b5d9
		6db1b1dc	5a6d6d37	50454515	
125	8f3636b9	1b6c6c77	adbebe13	904a4ada	b9eeee57
		de7777a9	bef2f24c	7efdfd83	
126	11444455	da6767bd	5d71712c	40050545	1f7c7c63
		10404050	5b696932	db6363b8	
127	0a282822	c20707c5	31c4c4f5	8a2222a8	a7969631
		ce3737f9	7aeded97	bff6f649	
128	2db4b499	75d1d1a4	d3434390	1248485a	bae2e258
		e6979771	b6d2d264	b2c2c270	
129	8b2626ad	68a5a5cd	955e5ecb	4b292962	0c30303c
		945a5ace	76ddddab	7ff9f986	
130	649595f1	bbe6e65d	f2c7c735	0924242d	c61717d1
		6fb9b9d6	c51b1bde	86121294	
131	18606078	f3c3c330	7cf5f589	efb3b35c	3ae8e8d2
		df7373ac	4c353579	208080a0	
132	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：

```

1 #include<iostream>
2 #include<chrono>
3 #include <iomanip>
4
5 using namespace std;
```

```

6
7 // S盒
8 static const unsigned long SboxTable[16][16] = {
9     {0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2,
10        0x28, 0xfb, 0x2c, 0x05},
11     {0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26,
12        0x49, 0x86, 0x06, 0x99},
13     {0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43,
14        0xed, 0xcf, 0xac, 0x62},
15     {0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa,
16        0x75, 0x8f, 0x3f, 0xa6},
17     {0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19,
18        0xe6, 0x85, 0x4f, 0xa8},
19     {0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b,
20        0x70, 0x56, 0x9d, 0x35},
21     {0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b,
22        0x01, 0x21, 0x78, 0x87},
23     {0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7,
24        0xa0, 0xc4, 0xc8, 0x9e},
25     {0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce,
26        0xf9, 0x61, 0x15, 0xa1},
27     {0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30,
28        0xf5, 0x8c, 0xb1, 0xe3},
29     {0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab,
30        0x0d, 0x53, 0x4e, 0x6f},
31     {0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72,
32        0x6d, 0x6c, 0x5b, 0x51},
33     {0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41,
34        0x1f, 0x10, 0x5a, 0xd8},
35     {0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12,
36        0xb8, 0xe5, 0xb4, 0xb0},
37     {0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09,
38        0xc5, 0x6e, 0xc6, 0x84},
39     {0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e,
40        0xd7, 0xcb, 0x39, 0x48}
41 };
42
43 unsigned long sm4Sbox(unsigned long in) {
44     return SboxTable[(in >> 4) & 0x0F][in & 0x0F];
45 }
46
47 unsigned long L(unsigned long x) {
48     return x ^ (x << 2 | x >> (32 - 2)) ^ (x << 10 | x >> (32 - 10)) ^ (x <<

```

```

18 | x >> (32 - 18)) ^ (x << 24 | x >> (32 - 24));
33 }
34
35 unsigned long T(unsigned long x) {
36     unsigned long b = 0;
37     for (int i = 0; i < 4; i++) {
38         b = (b << 8) | sm4Sbox((x >> ((3 - i) * 8)) & 0xFF);
39     }
40     return L(b);
41 }
42
43 // 优化 T 可算表
44 //unsigned long T0[256], T1[256], T2[256], T3[256];
45 unsigned long T0[256] = {
46     0x8ED55B5B, 0xD0924242, 0x4DEAA7A7, 0x06FDFBFB, 0xFCCF3333, 0x65E28787,
47     0xC93DF4F4, 0x6BB5DEDE, 0x4E165858, 0x6EB4DADA, 0x44145050, 0
48     xCAC10B0B,
49     0x8828A0A0, 0x17F8EFEF, 0x9C2CB0B0, 0x11051414, 0x872BACAC, 0
50     xFB669D9D,
51     0xF2986A6A, 0xAE77D9D9, 0x822AA8A8, 0x46BCFAFA, 0x14041010, 0
52     xCFC00F0F,
53     0x02A8AAAA, 0x54451111, 0x5F134C4C, 0xBE269898, 0x6D482525, 0
54     x9E841A1A,
55     0x1E061818, 0xFD9B6666, 0xEC9E7272, 0x4A430909, 0x10514141, 0
56     x24F7D3D3,
57     0xD5934646, 0x53ECBFBF, 0xF89A6262, 0x927BE9E9, 0xFF33CCCC, 0
58     x04555151,
59     0x270B2C2C, 0x4F420D0D, 0x59EEB7B7, 0xF3CC3F3F, 0x1CAEB2B2, 0
60     xEA638989,
    0x74E79393, 0x7FB1CECE, 0x6C1C7070, 0x0DABA6A6, 0xEDCA2727, 0
    x28082020,
    0x48EBA3A3, 0xC1975656, 0x80820202, 0xA3DC7F7F, 0xC4965252, 0
    x12F9EBEB,
    0xA174D5D5, 0xB38D3E3E, 0xC33FFCFC, 0x3EA49A9A, 0x5B461D1D, 0
    x1B071C1C,
    0x3BA59E9E, 0x0CFFF3F3, 0x3FF0CFCF, 0xBF72CDCD, 0x4B175C5C, 0
    x52B8EAEA,
    0x8F810E0E, 0x3D586565, 0xCC3CF0F0, 0x7D196464, 0x7EE59B9B, 0
    x91871616,
    0x734E3D3D, 0x08AAA2A2, 0xC869A1A1, 0xC76AADAD, 0x85830606, 0
    x7AB0CACA,
    0xB570C5C5, 0xF4659191, 0xB2D96B6B, 0xA7892E2E, 0x18FBE3E3, 0
    x47E8AFAF,

```

61 0x330F3C3C, 0x674A2D2D, 0xB071C1C1, 0x0E575959, 0xE99F7676, 0  
x E135D4D4,

62 0x661E7878, 0xB4249090, 0x360E3838, 0x265F7979, 0xEF628D8D, 0  
x 38596161,

63 0x95D24747, 0x2AA08A8A, 0xB1259494, 0xAA228888, 0x8C7DF1F1, 0  
x D73BECEC,

64 0x05010404, 0xA5218484, 0x9879E1E1, 0x9B851E1E, 0x84D75353, 0  
x 00000000,

65 0x5E471919, 0x0B565D5D, 0xE39D7E7E, 0x9FD04F4F, 0xBB279C9C, 0  
x 1A534949,

66 0x7C4D3131, 0xEE36D8D8, 0x0A020808, 0x7BE49F9F, 0x20A28282, 0  
x D4C71313,

67 0xE8CB2323, 0xE69C7A7A, 0x42E9ABAB, 0x43BDFEFE, 0xA2882A2A, 0  
x 9AD14B4B,

68 0x40410101, 0xDBC41F1F, 0xD838E0E0, 0x61B7D6D6, 0x2FA18E8E, 0  
x 2BF4DFDF,

69 0x3AF1CBCB, 0xF6CD3B3B, 0x1DFAE7E7, 0xE5608585, 0x41155454, 0  
x 25A38686,

70 0x60E38383, 0x16ACBABA, 0x295C7575, 0x34A69292, 0xF7996E6E, 0  
x E434D0D0,

71 0x721A6868, 0x01545555, 0x19AFB6B6, 0xDF914E4E, 0xFA32C8C8, 0  
x F030C0C0,

72 0x21F6D7D7, 0xBC8E3232, 0x75B3C6C6, 0x6FE08F8F, 0x691D7474, 0  
x 2EF5DBDB,

73 0x6AE18B8B, 0x962EB8B8, 0x8A800A0A, 0xFE679999, 0xE2C92B2B, 0  
x E0618181,

74 0xC0C30303, 0x8D29A4A4, 0xAF238C8C, 0x07A9AEAE, 0x390D3434, 0  
x 1F524D4D,

75 0x764F3939, 0xD36EBDBD, 0x81D65757, 0xB7D86F6F, 0xEB37DCDC, 0  
x 51441515,

76 0xA6DD7B7B, 0x09FEF7F7, 0xB68C3A3A, 0x932FBCBC, 0x0F030C0C, 0  
x 03FCFFFF,

77 0xC26BA9A9, 0xBA73C9C9, 0xD96CB5B5, 0xDC6DB1B1, 0x375A6D6D, 0  
x 15504545,

78 0xB98F3636, 0x771B6C6C, 0x13ADBEBE, 0xDA904A4A, 0x57B9EEEE, 0  
x A9DE7777,

79 0x4CBEF2F2, 0x837EFD FD, 0x55114444, 0xBD DA6767, 0x2C5D7171, 0  
x 45400505,

80 0x631F7C7C, 0x50104040, 0x325B6969, 0xB8DB6363, 0x220A2828, 0  
x C5C20707,

81 0xF531C4C4, 0xA88A2222, 0x31A79696, 0xF9CE3737, 0x977AEDED, 0  
x 49BFF6F6,

82 0x992DB4B4, 0xA475D1D1, 0x90D34343, 0x5A124848, 0x58BAE2E2, 0

```

83         x71E69797,
            0x64B6D2D2, 0x70B2C2C2, 0xAD8B2626, 0xCD68A5A5, 0xCB955E5E, 0
84         x624B2929,
            0x3C0C3030, 0xCE945A5A, 0xAB76DDDD, 0x867FF9F9, 0xF1649595, 0
85         x5DBBE6E6,
            0x35F2C7C7, 0x2D092424, 0xD1C61717, 0xD66FB9B9, 0xDEC51B1B, 0
86         x94861212,
            0x78186060, 0x30F3C3C3, 0x897CF5F5, 0x5CEFB3B3, 0xD23AE8E8, 0
87         xACDF7373,
            0x794C3535, 0xA0208080, 0x9D78E5E5, 0x56EDBBBB, 0x235E7D7D, 0
88         xC63EF8F8,
            0x8BD45F5F, 0xE7C82F2F, 0xDD39E4E4, 0x68492121
89     };
90
91     unsigned long T1[256] = {
92         0x5B8ED55B, 0x42D09242, 0xA74DEAA7, 0xFB06FDFB, 0x33FCCF33, 0x8765E287,
93         0xF4C93DF4, 0xDE6BB5DE, 0x584E1658, 0xDA6EB4DA, 0x50441450, 0x0BCAC10B,
94         0xA08828A0, 0xEF17F8EF, 0xB09C2CB0, 0x14110514, 0xAC872BAC, 0x9DFB669D,
95         0x6AF2986A, 0xD9AE77D9, 0xA8822AA8, 0xFA46BCFA, 0x10140410, 0x0FCFC00F,
96         0xAA02A8AA, 0x11544511, 0x4C5F134C, 0x98BE2698, 0x256D4825, 0x1A9E841A,
97         0x181E0618, 0x66FD9B66, 0x72EC9E72, 0x094A4309, 0x41105141, 0xD324F7D3,
98         0x46D59346, 0xBF53ECBF, 0x62F89A62, 0xE9927BE9, 0xCCFF33CC, 0x51045551,
99         0x2C270B2C, 0x0D4F420D, 0xB759EEB7, 0x3FF3CC3F, 0xB21CAEB2, 0x89EA6389,
100        0x9374E793, 0xCE7FB1CE, 0x706C1C70, 0xA60DABA6, 0x27EDCA27, 0x20280820,
101        0xA348EBA3, 0x56C19756, 0x02808202, 0x7FA3DC7F, 0x52C49652, 0xEB12F9EB,
102        0xD5A174D5, 0x3EB38D3E, 0xFCC33FFC, 0x9A3EA49A, 0x1D5B461D, 0x1C1B071C,
103        0x9E3BA59E, 0xF30CFFF3, 0xCF3FF0CF, 0xCDBF72CD, 0x5C4B175C, 0xEA52B8EA,
104        0x0E8F810E, 0x653D5865, 0xF0CC3CF0, 0x647D1964, 0x9B7EE59B, 0x16918716,
105        0x3D734E3D, 0xA208AAA2, 0xA1C869A1, 0xADC76AAD, 0x06858306, 0xCA7AB0CA,
106        0xC5B570C5, 0x91F46591, 0x6BB2D96B, 0x2EA7892E, 0xE318FBE3, 0xAF47E8AF,
107        0x3C330F3C, 0x2D674A2D, 0xC1B071C1, 0x590E5759, 0x76E99F76, 0xD4E135D4,
108        0x78661E78, 0x90B42490, 0x38360E38, 0x79265F79, 0x8DEF628D, 0x61385961,
109        0x4795D247, 0x8A2AA08A, 0x94B12594, 0x88AA2288, 0xF18C7DF1, 0xECD73BEC,
110        0x04050104, 0x84A52184, 0xE19879E1, 0x1E9B851E, 0x5384D753, 0x00000000,
111        0x195E4719, 0x5D0B565D, 0x7EE39D7E, 0x4F9FD04F, 0x9CBB279C, 0x491A5349,
112        0x317C4D31, 0xD8EE36D8, 0x080A0208, 0x9F7BE49F, 0x8220A282, 0x13D4C713,
113        0x23E8CB23, 0x7AE69C7A, 0xAB42E9AB, 0xFE43BDFF, 0x2AA2882A, 0x4B9AD14B,
114        0x01404101, 0x1FDBC41F, 0xE0D838E0, 0xD661B7D6, 0x8E2FA18E, 0xDF2BF4DF,
115        0xCB3AF1CB, 0x3BF6CD3B, 0xE71DFAE7, 0x85E56085, 0x54411554, 0x8625A386,
116        0x8360E383, 0xBA16ACBA, 0x75295C75, 0x9234A692, 0x6EF7996E, 0xD0E434D0,
117        0x68721A68, 0x55015455, 0xB619AFB6, 0x4EDF914E, 0xC8FA32C8, 0xC0F030C0,
118        0xD721F6D7, 0x32BC8E32, 0xC675B3C6, 0x8F6FE08F, 0x74691D74, 0xDB2EF5DB,
119        0x8B6AE18B, 0xB8962EB8, 0x0A8A800A, 0x99FE6799, 0x2BE2C92B, 0x81E06181,

```

```

120     0x03C0C303, 0xA48D29A4, 0x8CAF238C, 0xAE07A9AE, 0x34390D34, 0x4D1F524D,
121     0x39764F39, 0xBDD36EBD, 0x5781D657, 0x6FB7D86F, 0xDCEB37DC, 0x15514415,
122     0x7BA6DD7B, 0xF709FEF7, 0x3AB68C3A, 0xBC932FBC, 0x0C0F030C, 0xFF03FCFF,
123     0xA9C26BA9, 0xC9BA73C9, 0xB5D96CB5, 0xB1DC6DB1, 0x6D375A6D, 0x45155045,
124     0x36B98F36, 0x6C771B6C, 0xBE13ADBE, 0x4ADA904A, 0xEE57B9EE, 0x77A9DE77,
125     0xF24CBEF2, 0xFD837EFD, 0x44551144, 0x67BDDA67, 0x712C5D71, 0x05454005,
126     0x7C631F7C, 0x40501040, 0x69325B69, 0x63B8DB63, 0x28220A28, 0x07C5C207,
127     0xC4F531C4, 0x22A88A22, 0x9631A796, 0x37F9CE37, 0xED977AED, 0xF649BFF6,
128     0xB4992DB4, 0xD1A475D1, 0x4390D343, 0x485A1248, 0xE258BAE2, 0x9771E697,
129     0xD264B6D2, 0xC270B2C2, 0x26AD8B26, 0xA5CD68A5, 0x5ECB955E, 0x29624B29,
130     0x303C0C30, 0x5ACE945A, 0xDDAB76DD, 0xF9867FF9, 0x95F16495, 0xE65DBBE6,
131     0xC735F2C7, 0x242D0924, 0x17D1C617, 0xB9D66FB9, 0x1BDEC51B, 0x12948612,
132     0x60781860, 0xC330F3C3, 0xF5897CF5, 0xB35CEFB3, 0xE8D23AE8, 0x73ACDF73,
133     0x35794C35, 0x80A02080, 0xE59D78E5, 0xBB56EDBB, 0x7D235E7D, 0xF8C63EF8,
134     0x5F8BD45F, 0x2FE7C82F, 0xE4DD39E4, 0x21684921 };
135
136 unsigned long T2[256] = {
137     0x5B5B8ED5, 0x4242D092, 0xA7A74DEA, 0xFBFB06FD, 0x3333FCCF, 0x878765E2,
138     0xF4F4C93D, 0xDEDE6BB5, 0x58584E16, 0xDADA6EB4, 0x50504414, 0x0B0BCAC1,
139     0xA0A08828, 0xEFEF17F8, 0xB0B09C2C, 0x14141105, 0xACAC872B, 0x9D9DFB66,
140     0x6A6AF298, 0xD9D9AE77, 0xA8A8822A, 0xFAFA46BC, 0x10101404, 0x0F0FCFC0,
141     0xAAAA02A8, 0x11115445, 0x4C4C5F13, 0x9898BE26, 0x25256D48, 0x1A1A9E84,
142     0x18181E06, 0x6666FD9B, 0x7272EC9E, 0x09094A43, 0x41411051, 0xD3D324F7,
143     0x4646D593, 0xBFBF53EC, 0x6262F89A, 0xE9E9927B, 0xCCCCFF33, 0x51510455,
144     0x2C2C270B, 0x0D0D4F42, 0xB7B759EE, 0x3F3FF3CC, 0xB2B21CAE, 0x8989EA63,
145     0x939374E7, 0xCECE7FB1, 0x70706C1C, 0xA6A60DAB, 0x2727EDCA, 0x20202808,
146     0xA3A348EB, 0x5656C197, 0x02028082, 0x7F7FA3DC, 0x5252C496, 0xEBEB12F9,
147     0xD5D5A174, 0x3E3EB38D, 0xFCFC33F, 0x9A9A3EA4, 0x1D1D5B46, 0x1C1C1B07,
148     0x9E9E3BA5, 0xF3F30CFF, 0xCFCF3FF0, 0xCDCDBF72, 0x5C5C4B17, 0xEAEA52B8,
149     0x0E0E8F81, 0x65653D58, 0xF0F0CC3C, 0x64647D19, 0x9B9B7EE5, 0x16169187,
150     0x3D3D734E, 0xA2A208AA, 0xA1A1C869, 0xADADC76A, 0x06068583, 0xCACA7AB0,
151     0xC5C5B570, 0x9191F465, 0x6B6BB2D9, 0x2E2EA789, 0xE3E318FB, 0xAFAF47E8,
152     0x3C3C330F, 0x2D2D674A, 0xC1C1B071, 0x59590E57, 0x7676E99F, 0xD4D4E135,
153     0x7878661E, 0x9090B424, 0x3838360E, 0x7979265F, 0x8D8DEF62, 0x61613859,
154     0x474795D2, 0x8A8A2AA0, 0x9494B125, 0x8888AA22, 0xF1F18C7D, 0xECECD73B,
155     0x04040501, 0x8484A521, 0xE1E19879, 0x1E1E9B85, 0x535384D7, 0x00000000,
156     0x19195E47, 0x5D5D0B56, 0x7E7EE39D, 0x4F4F9FD0, 0x9C9CB27, 0x4949A153,
157     0x31317C4D, 0xD8D8EE36, 0x08080A02, 0x9F9F7BE4, 0x828220A2, 0x1313D4C7,
158     0x2323E8CB, 0x7A7AE69C, 0xABAB42E9, 0xFEFE43BD, 0x2A2AA288, 0x4B4B9AD1,
159     0x01014041, 0x1F1FDBC4, 0xE0E0D838, 0xD6D661B7, 0x8E8E2FA1, 0xDFDF2BF4,
160     0xCBCB3AF1, 0x3B3BF6CD, 0xE7E71DFA, 0x8585E560, 0x54544115, 0x868625A3,
161     0x838360E3, 0xBABA16AC, 0x7575295C, 0x929234A6, 0x6E6EF799, 0xD0D0E434,
162     0x6868721A, 0x55550154, 0xB6B619AF, 0x4E4EDF91, 0xC8C8FA32, 0xC0C0F030,

```

```

163     0xD7D721F6, 0x3232BC8E, 0xC6C675B3, 0x8F8F6FE0, 0x7474691D, 0xDBDB2EF5,
164     0x8B8B6AE1, 0xB8B8962E, 0x0A0A8A80, 0x9999FE67, 0x2B2BE2C9, 0x8181E061,
165     0x0303C0C3, 0xA4A48D29, 0x8C8CAF23, 0xAEAE07A9, 0x3434390D, 0x4D4D1F52,
166     0x3939764F, 0xBDBDD36E, 0x575781D6, 0x6F6FB7D8, 0xDCDCEB37, 0x15155144,
167     0x7B7BA6DD, 0xF7F709FE, 0x3A3AB68C, 0xBCBC932F, 0x0C0C0F03, 0xFFFF03FC,
168     0xA9A9C26B, 0xC9C9BA73, 0xB5B5D96C, 0xB1B1DC6D, 0x6D6D375A, 0x45451550,
169     0x3636B98F, 0x6C6C771B, 0xBEBE13AD, 0x4A4ADA90, 0xEEEE57B9, 0x7777A9DE,
170     0xF2F24CBE, 0xFDFD837E, 0x44445511, 0x6767BDDA, 0x71712C5D, 0x05054540,
171     0x7C7C631F, 0x40405010, 0x6969325B, 0x6363B8DB, 0x2828220A, 0x0707C5C2,
172     0xC4C4F531, 0x2222A88A, 0x969631A7, 0x3737F9CE, 0xEDED977A, 0xF6F649BF,
173     0xB4B4992D, 0xD1D1A475, 0x434390D3, 0x48485A12, 0xE2E258BA, 0x979771E6,
174     0xD2D264B6, 0xC2C270B2, 0x2626AD8B, 0xA5A5CD68, 0x5E5ECB95, 0x2929624B,
175     0x30303C0C, 0x5A5ACE94, 0xDDDDAB76, 0xF9F9867F, 0x9595F164, 0xE6E65DBB,
176     0xC7C735F2, 0x24242D09, 0x1717D1C6, 0xB9B9D66F, 0x1B1BDEC5, 0x12129486,
177     0x60607818, 0xC3C330F3, 0xF5F5897C, 0xB3B35CEF, 0xE8E8D23A, 0x7373ACDF,
178     0x3535794C, 0x8080A020, 0xE5E59D78, 0BBBB56ED, 0x7D7D235E, 0xF8F8C63E,
179     0x5F5F8BD4, 0x2F2FE7C8, 0xE4E4DD39, 0x21216849 };
180
181 unsigned long T3[256] = {
182     0xD5B5B8E, 0x924242D0, 0xEAA7A74D, 0xFDFBFB06, 0xCF3333FC, 0xE2878765,
183     0x3DF4F4C9, 0xB5DEDE6B, 0x1658584E, 0xB4DADA6E, 0x14505044, 0xC10B0BCA,
184     0x28A0A088, 0xF8EFEF17, 0x2CB0B09C, 0x05141411, 0x2BACAC87, 0x669D9DFB,
185     0x986A6AF2, 0x77D9D9AE, 0x2AA8A882, 0xBCFAFA46, 0x04101014, 0xC00F0FCF,
186     0xA8AAAA02, 0x45111154, 0x134C4C5F, 0x269898BE, 0x4825256D, 0x841A1A9E,
187     0x0618181E, 0x9B6666FD, 0x9E7272EC, 0x4309094A, 0x51414110, 0xF7D3D324,
188     0x934646D5, 0xECBFBF53, 0x9A6262F8, 0x7BE9E992, 0x33CCCCFF, 0x55515104,
189     0x0B2C2C27, 0x420D0D4F, 0xEEB7B759, 0xCC3F3FF3, 0xAEB2B21C, 0x638989EA,
190     0xE7939374, 0xB1CECE7F, 0x1C70706C, 0xABA6A60D, 0xCA2727ED, 0x08202028,
191     0xEBA3A348, 0x975656C1, 0x82020280, 0xDC7F7FA3, 0x965252C4, 0xF9EBEB12,
192     0x74D5D5A1, 0x8D3E3EB3, 0x3FFCFCC3, 0xA49A9A3E, 0x461D1D5B, 0x071C1C1B,
193     0xA59E9E3B, 0xFFF3F30C, 0xF0CFCF3F, 0x72CDCDBF, 0x175C5C4B, 0xB8EAEA52,
194     0x810E0E8F, 0x5865653D, 0x3CF0F0CC, 0x1964647D, 0xE59B9B7E, 0x87161691,
195     0x4E3D3D73, 0xAAA2A208, 0x69A1A1C8, 0x6AADADC7, 0x83060685, 0xB0CACA7A,
196     0x70C5C5B5, 0x659191F4, 0xD96B6BB2, 0x892E2EA7, 0xFBE3E318, 0xE8AF4F47,
197     0x0F3C3C33, 0x4A2D2D67, 0x71C1C1B0, 0x5759590E, 0x9F7676E9, 0x35D4D4E1,
198     0x1E787866, 0x249090B4, 0x0E383836, 0x5F797926, 0x628D8DEF, 0x59616138,
199     0xD2474795, 0xA08A8A2A, 0x259494B1, 0x228888AA, 0x7DF1F18C, 0x3BECECD7,
200     0x01040405, 0x218484A5, 0x79E1E198, 0x851E1E9B, 0xD7535384, 0x00000000,
201     0x4719195E, 0x565D5D0B, 0x9D7E7EE3, 0xD04F4F9F, 0x279C9CBB, 0x5349491A,
202     0x4D31317C, 0x36D8D8EE, 0x0208080A, 0xE49F9F7B, 0xA2828220, 0xC71313D4,
203     0xCB2323E8, 0x9C7A7AE6, 0xE9ABAB42, 0xBDFEFE43, 0x882A2AA2, 0xD14B4B9A,
204     0x41010140, 0xC41F1FDB, 0x38E0E0D8, 0xB7D6D661, 0xA18E8E2F, 0xF4DFDF2B,
205     0xF1CBCB3A, 0xCD3B3BF6, 0xFAE7E71D, 0x608585E5, 0x15545441, 0xA3868625,

```



```

206     0xE3838360, 0xACBABA16, 0x5C757529, 0xA6929234, 0x996E6EF7, 0x34D0D0E4,
207     0x1A686872, 0x54555501, 0xAFB6B619, 0x914E4EDF, 0x32C8C8FA, 0x30C0C0F0,
208     0xF6D7D721, 0x8E3232BC, 0xB3C6C675, 0xE08F8F6F, 0x1D747469, 0xF5DBDB2E,
209     0xE18B8B6A, 0x2EB8B896, 0x800A0A8A, 0x679999FE, 0xC92B2BE2, 0x618181E0,
210     0xC30303C0, 0x29A4A48D, 0x238C8CAF, 0xA9AEAE07, 0x0D343439, 0x524D4D1F,
211     0x4F393976, 0x6EBDBDD3, 0xD6575781, 0xD86F6FB7, 0x37DCDCEB, 0x44151551,
212     0xDD7B7BA6, 0xFE7F7F09, 0x8C3A3AB6, 0x2FBCBC93, 0x030C0C0F, 0xFCFFFF03,
213     0x6BA9A9C2, 0x73C9C9BA, 0x6CB5B5D9, 0x6DB1B1DC, 0x5A6D6D37, 0x50454515,
214     0x8F3636B9, 0x1B6C6C77, 0xADBE13, 0x904A4ADA, 0xB9EEEE57, 0xDE7777A9,
215     0xBEF2F24C, 0x7EFD7FD83, 0x11444455, 0xDA6767BD, 0x5D71712C, 0x40050545,
216     0x1F7C7C63, 0x10404050, 0x5B696932, 0xDB6363B8, 0x0A282822, 0xC20707C5,
217     0x31C4C4F5, 0x8A2222A8, 0xA7969631, 0xCE3737F9, 0x7AEDED97, 0xBFF6F649,
218     0x2DB4B499, 0x75D1D1A4, 0xD3434390, 0x1248485A, 0xBAE2E258, 0xE6979771,
219     0xB6D2D264, 0xB2C2C270, 0x8B2626AD, 0x68A5A5CD, 0x955E5ECB, 0x4B292962,
220     0x0C30303C, 0x945A5ACE, 0x76DDDDAB, 0x7FF9F986, 0x649595F1, 0xBBE6E65D,
221     0xF2C7C735, 0x0924242D, 0xC61717D1, 0x6FB9B9D6, 0xC51B1BDE, 0x86121294,
222     0x18606078, 0xF3C3C330, 0x7CF5F589, 0xEFB3B35C, 0x3AE8E8D2, 0xDF7373AC,
223     0x4C353579, 0x208080A0, 0x78E5E59D, 0xEDBBB56, 0x5E7D7D23, 0x3EF8F8C6,
224     0xD45F5F8B, 0xC82F2FE7, 0x39E4E4DD, 0x49212168 };
225
226 /*
227 void init_T_tables() {
228     for (int x = 0; x < 256; x++) {
229         unsigned char s = sm4Sbox(x);
230         T0[x] = L((unsigned long)s << 24);
231         T1[x] = L((unsigned long)s << 16);
232         T2[x] = L((unsigned long)s << 8);
233         T3[x] = L((unsigned long)s);
234     }
235 }
236 */
237 unsigned long T_opt(unsigned long x) {
238     return T0[(x >> 24) & 0xFF] ^ T1[(x >> 16) & 0xFF] ^ T2[(x >> 8) & 0xFF]
        ^ T3[x & 0xFF];
239 }
240
241 static const unsigned long FK[4] = { 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0
    xb27022dc };
242
243 static const unsigned long CK[32] = {
244     0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269, 0x70777e85, 0x8c939aa1,
        0xa8afb6bd, 0xc4cbd2d9,
245     0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249, 0x50575e65, 0x6c737a81,

```

```

    0x888f969d, 0xa4abb2b9,
246    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229, 0x30373e45, 0x4c535a61,
    0x686f767d, 0x848b9299,
247    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209, 0x10171e25, 0x2c333a41,
    0x484f565d, 0x646b7279
248 };
249
250 void key_expansion(unsigned long MK[4], unsigned long rk[32]) {
251     unsigned long K[36];
252     for (int i = 0; i < 4; i++)
253         K[i] = MK[i] ^ FK[i];
254     for (int i = 0; i < 32; i++) {
255         unsigned long tmp = K[i + 1] ^ K[i + 2] ^ K[i + 3] ^ CK[i];
256         unsigned long b = 0;
257         for (int j = 0; j < 4; j++)
258             b = (b << 8) | sm4Sbox((tmp >> ((3 - j) * 8)) & 0xFF);
259         unsigned long Lval = b ^ (b << 13 | b >> (32 - 13)) ^ (b << 23 | b
            >> (32 - 23));
260         rk[i] = K[i] ^ Lval;
261         K[i + 4] = rk[i];
262     }
263 }
264
265 unsigned long round_operate(int i, unsigned long* X, unsigned long* rk) {
266     return X[i] ^ T_opt(X[i + 1] ^ X[i + 2] ^ X[i + 3] ^ rk[i]);
267 }
268
269 void sm4_enc(unsigned long MK[4], unsigned long X[4]) {
270     cout << hex;
271     cout << "Plaintext:" << endl;
272     cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;
273     cout << "Key:" << endl;
274     cout << MK[0] << " " << MK[1] << " " << MK[2] << " " << MK[3] << endl;
275
276     unsigned long rk[32];
277     key_expansion(MK, rk);
278     for (int i = 0; i < 32; i++) {
279         unsigned long tmp = round_operate(i, X, rk);
280         X[4 + i] = tmp;
281     }
282     cout << "Ciphertext:" << endl;
283     cout << X[35] << " " << X[34] << " " << X[33] << " " << X[32] << endl;
284 }

```

```

285
286 void sm4_dec(unsigned long MK[4], unsigned long X[4]) {
287     cout << hex;
288     cout << "Ciphertext:" << endl;
289     cout << X[0] << " " << X[1] << " " << X[2] << " " << X[3] << endl;
290
291     unsigned long rk[32];
292     key_expansion(MK, rk);
293     for (int i = 0; i < 16; i++) swap(rk[i], rk[31 - i]);
294
295     unsigned long tmpX[36] = { 0 };
296     for (int i = 0; i < 4; i++) tmpX[i] = X[i];
297     for (int i = 0; i < 32; i++) tmpX[i + 4] = round_operate(i, tmpX, rk);
298
299     cout << "Decrypted Plaintext:" << endl;
300     cout << tmpX[35] << " " << tmpX[34] << " " << tmpX[33] << " " << tmpX
        [32] << endl;
301 }
302
303 int main() {
304     //init_T_tables();
305     unsigned long MK[4] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210
        };
306     unsigned long X[36] = { 0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210
        };
307     unsigned long C[36] = { 0x681edf34, 0xd206965e, 0x86b3e94f, 0x536e4246
        };
308     auto start = chrono::high_resolution_clock::now();
309     sm4_enc(MK, X);
310     auto end = chrono::high_resolution_clock::now();
311     auto duration = chrono::duration_cast<chrono::microseconds>(end - start)
        .count();
312     cout << "加密耗时: " << duration << " 微秒" << endl;
313
314
315     start = chrono::high_resolution_clock::now();
316     sm4_dec(MK, C);
317     end = chrono::high_resolution_clock::now();
318     duration = chrono::duration_cast<chrono::microseconds>(end - start).
        count();
319     cout << "解密耗时: " << duration << " 微秒" << endl;
320     /*
321     cout << "T0:" << endl;

```

```

322     for (int i = 0; i <= 31; i++)
323     {
324         cout << setfill('0') << setw(8) << hex << T0[i * 8] << "\t"
325             << setfill('0') << setw(8) << hex << T0[i * 8 + 1] << "\t"
326             << setfill('0') << setw(8) << hex << T0[i * 8 + 2] << "\t"
327             << setfill('0') << setw(8) << hex << T0[i * 8 + 3] << "\t"
328             << setfill('0') << setw(8) << hex << T0[i * 8 + 4] << "\t"
329             << setfill('0') << setw(8) << hex << T0[i * 8 + 5] << "\t"
330             << setfill('0') << setw(8) << hex << T0[i * 8 + 6] << "\t"
331             << setfill('0') << setw(8) << hex << T0[i * 8 + 7] << endl;
332
333     }
334
335     cout << "T1:" << endl;
336     for (int i = 0; i <= 31; i++)
337     {
338         cout << setfill('0') << setw(8) << hex << T1[i * 8] << "\t"
339             << setfill('0') << setw(8) << hex << T1[i * 8 + 1] << "\t"
340             << setfill('0') << setw(8) << hex << T1[i * 8 + 2] << "\t"
341             << setfill('0') << setw(8) << hex << T1[i * 8 + 3] << "\t"
342             << setfill('0') << setw(8) << hex << T1[i * 8 + 4] << "\t"
343             << setfill('0') << setw(8) << hex << T1[i * 8 + 5] << "\t"
344             << setfill('0') << setw(8) << hex << T1[i * 8 + 6] << "\t"
345             << setfill('0') << setw(8) << hex << T1[i * 8 + 7] << endl;
346
347     }
348     cout << "T2:" << endl;
349     for (int i = 0; i <= 31; i++)
350     {
351         cout << setfill('0') << setw(8) << hex << T2[i * 8] << "\t"
352             << setfill('0') << setw(8) << hex << T2[i * 8 + 1] << "\t"
353             << setfill('0') << setw(8) << hex << T2[i * 8 + 2] << "\t"
354             << setfill('0') << setw(8) << hex << T2[i * 8 + 3] << "\t"
355             << setfill('0') << setw(8) << hex << T2[i * 8 + 4] << "\t"
356             << setfill('0') << setw(8) << hex << T2[i * 8 + 5] << "\t"
357             << setfill('0') << setw(8) << hex << T2[i * 8 + 6] << "\t"
358             << setfill('0') << setw(8) << hex << T2[i * 8 + 7] << endl;
359
360     }
361     cout << "T3:" << endl;
362     for (int i = 0; i <= 31; i++)
363     {
364         cout << setfill('0') << setw(8) << hex << T3[i * 8] << "\t"

```

```

365         << setfill('0') << setw(8) << hex << T3[i * 8 + 1] << "\t"
366         << setfill('0') << setw(8) << hex << T3[i * 8 + 2] << "\t"
367         << setfill('0') << setw(8) << hex << T3[i * 8 + 3] << "\t"
368         << setfill('0') << setw(8) << hex << T3[i * 8 + 4] << "\t"
369         << setfill('0') << setw(8) << hex << T3[i * 8 + 5] << "\t"
370         << setfill('0') << setw(8) << hex << T3[i * 8 + 6] << "\t"
371         << setfill('0') << setw(8) << hex << T3[i * 8 + 7] << endl;
372
373     }
374 */
375     return 0;
376
377 }
```

## 3.2 优化方法 2: SIMD 指令集优化

SIMD 指令集优化可以一次性并行处理 4 个 32 位整数,和上述优化的逻辑大致相似。具体代码如下所示:

```

1  #include<iostream>
2  #include<chrono>
3  #include <iomanip>
4  #include <immintrin.h>
5  using namespace std;
6
7  // S 盒
8  static const unsigned char Sbox[256] = {
9      0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2,
10     0x28, 0xfb, 0x2c, 0x05,
11     0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26,
12     0x49, 0x86, 0x06, 0x99,
13     0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43,
14     0xed, 0xcf, 0xac, 0x62,
15     0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa,
16     0x75, 0x8f, 0x3f, 0xa6,
17     0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19,
18     0xe6, 0x85, 0x4f, 0xa8,
19     0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b,
20     0x70, 0x56, 0x9d, 0x35,
21     0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b,
22     0x01, 0x21, 0x78, 0x87,
23     0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7,
```

```

    0xa0, 0xc4, 0xc8, 0x9e,
17  0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce,
    0xf9, 0x61, 0x15, 0xa1,
18  0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30,
    0xf5, 0x8c, 0xb1, 0xe3,
19  0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab,
    0x0d, 0x53, 0x4e, 0x6f,
20  0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72,
    0x6d, 0x6c, 0x5b, 0x51,
21  0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41,
    0x1f, 0x10, 0x5a, 0xd8,
22  0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12,
    0xb8, 0xe5, 0xb4, 0xb0,
23  0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09,
    0xc5, 0x6e, 0xc6, 0x84,
24  0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e,
    0xd7, 0xcb, 0x39, 0x48
25 };
26
27 // SIMD L transform (for 4 32-bit integers in parallel)
28 __m128i L_SIMD(__m128i x) {
29     return _mm_xor_si128(
30         x,
31         _mm_xor_si128(
32             _mm_or_si128(_mm_slli_epi32(x, 2), _mm_srli_epi32(x, 30)),
33             _mm_xor_si128(
34                 _mm_or_si128(_mm_slli_epi32(x, 10), _mm_srli_epi32(x, 22)),
35                 _mm_xor_si128(
36                     _mm_or_si128(_mm_slli_epi32(x, 18), _mm_srli_epi32(x,
37                                     14)),
38                     _mm_or_si128(_mm_slli_epi32(x, 24), _mm_srli_epi32(x, 8)
39                                     )
40                     )
41                 )
42             )
43         );
44 }
45
46 // SIMD T transform for 4 32-bit integers
47 __m128i T_SIMD(__m128i input) {
48     alignas(16) unsigned int buffer[4];
49     _mm_store_si128((__m128i*)buffer, input);
50 }

```

```

49     for (int i = 0; i < 4; ++i) {
50         buffer[i] =
51             (Sbox[(buffer[i] >> 24) & 0xFF] << 24) |
52             (Sbox[(buffer[i] >> 16) & 0xFF] << 16) |
53             (Sbox[(buffer[i] >> 8) & 0xFF] << 8) |
54             (Sbox[buffer[i] & 0xFF]);
55     }
56
57     __m128i b = _mm_load_si128((__m128i*)buffer);
58     return L_SIMD(b);
59 }
60
61 int main() {
62     alignas(16) unsigned int test_data[4] = { 0x01234567, 0x89ABCDEF, 0
        x12345678, 0x9ABCDEF0 };
63     __m128i input = _mm_load_si128((__m128i*)test_data);
64
65     auto start = chrono::high_resolution_clock::now();
66     __m128i output = T_SIMD(input);
67     auto end = chrono::high_resolution_clock::now();
68
69     alignas(16) unsigned int result[4];
70     _mm_store_si128((__m128i*)result, output);
71
72     cout << "SIMD T Transform Results:" << endl;
73     for (int i = 0; i < 4; ++i) {
74         cout << "Result[" << i << "] = 0x" << hex << setw(8) << setfill('0')
            << result[i] << endl;
75     }
76
77     auto duration = chrono::duration_cast<chrono::nanoseconds>(end - start).
        count();
78     cout << "SIMD execution time: " << duration << " ns" << endl;
79
80     return 0;
81 }

```

具体而言, L\_SIMD 函数: 线性变换 L cpp 复制编辑 \_\_m128i L\_SIMD(\_\_m128i x) 函数进行 B 的 4 个左移操作

T\_SIMD 函数是  $T(B) = L(\tau(B))$  的实现, 将 input 中的 4 个 32 位整数保存到 buffer 中, 然后对 buffer[i] 的每个字节应用 S-Box: 每次从 buffer[i] 取出高、中、高中、低字节 (共 4 个字节), 查表替代后再拼接起来。将替代后的结果重新打

包成 \_\_m128i 向量, 然后对其进行 L\_SIMD 线性变换。该优化方法没有实现解密方法, 因为主要优化还是在加密中。

### 3.3 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：

```
SIMD T Transform Results:  
Result[0] = 0x6ef180c9  
Result[1] = 0x43a134f5  
Result[2] = 0xf32f23e8  
Result[3] = 0x8559a0b3  
SIMD execution time: a8c ns
```

图 8 优化 2 代码运行结果

可以看到该优化方法能达到的最快速度更快一点，多次运行结果下两个优化方案的加密速度是差不多的（总体 SIMD 方法更快）。