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 10 日

目录

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：

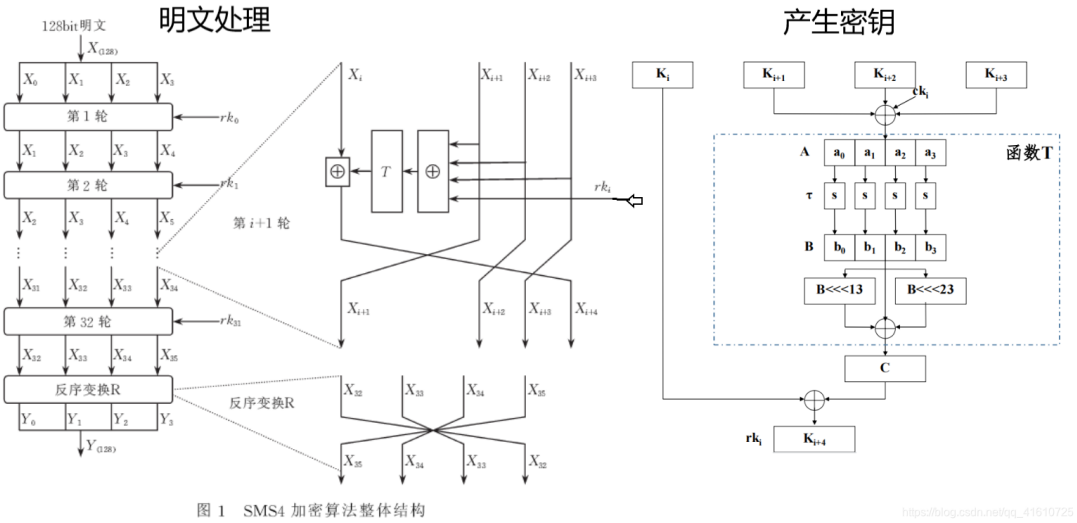

图 1　SMS4 加密算法整体结构

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。

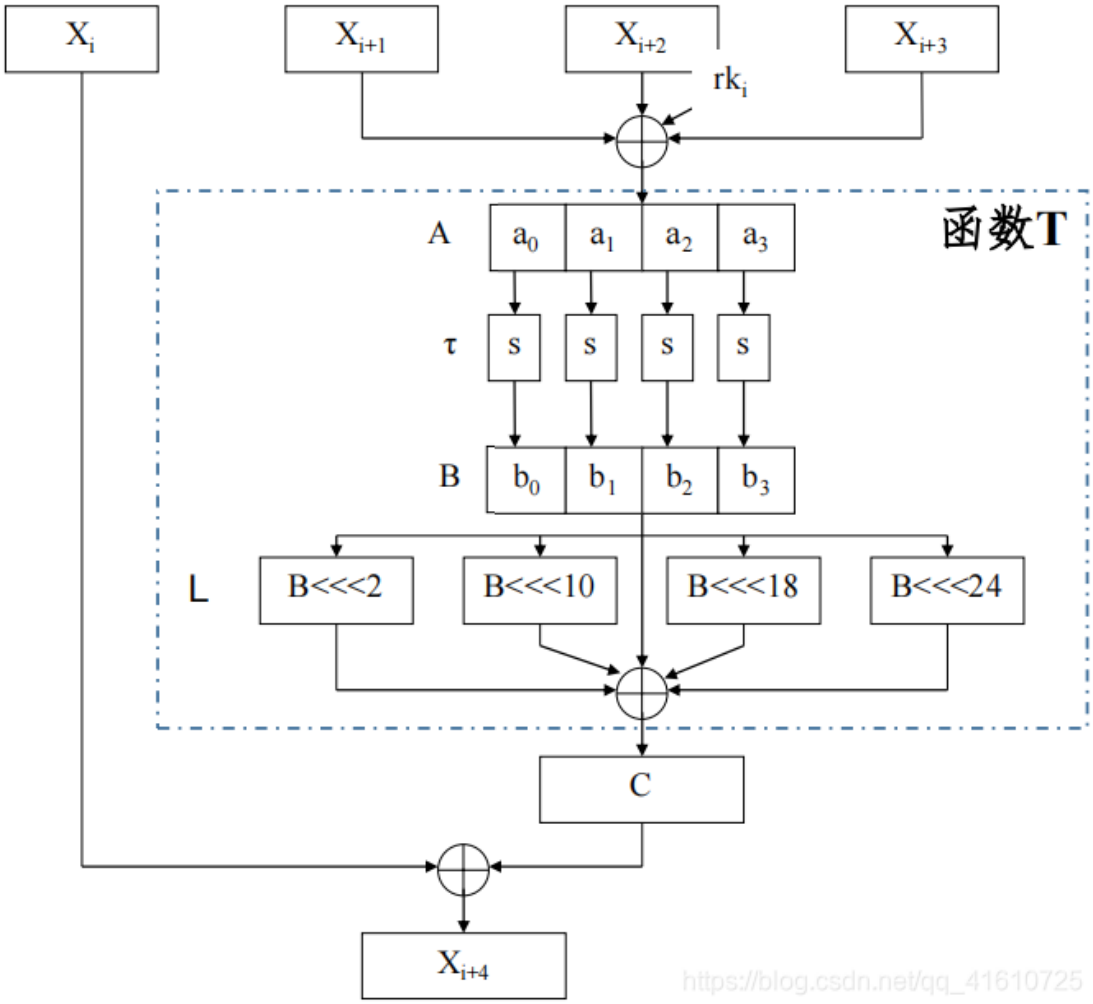

图 2 轮操作

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.

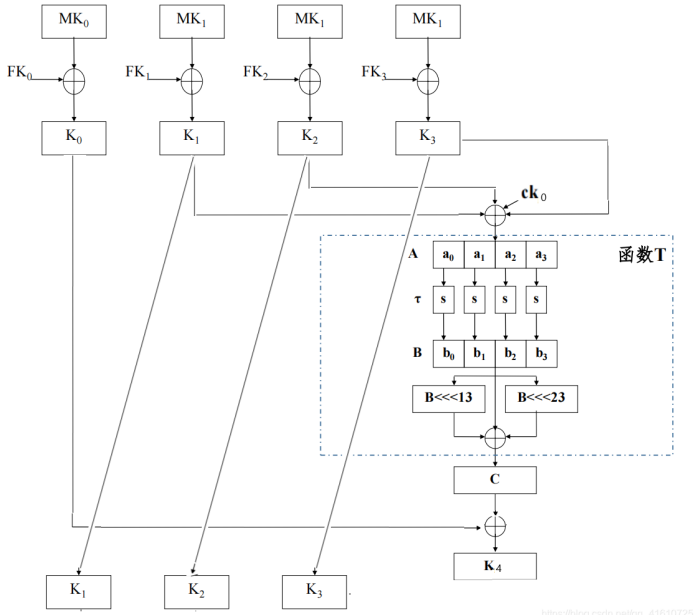

图 3 初始密钥拓展

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:

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

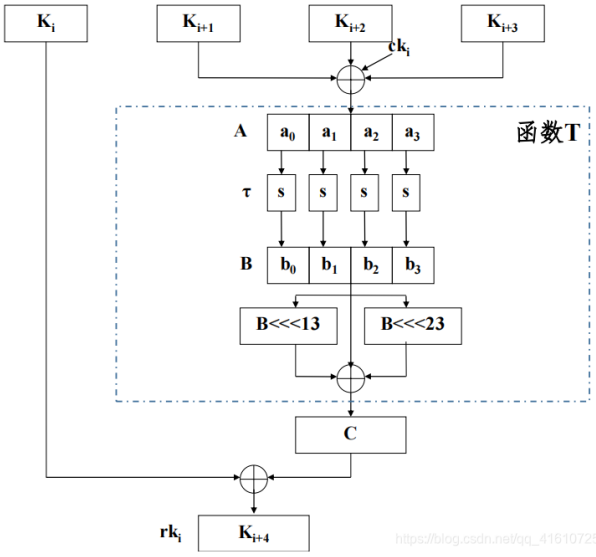

图 4 后续密钥拓展操作

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:

```
1 #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];
}
```

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

3 SM4 算法优化

3.1 优化 1

3.2 优化 2

3.3 优化 3