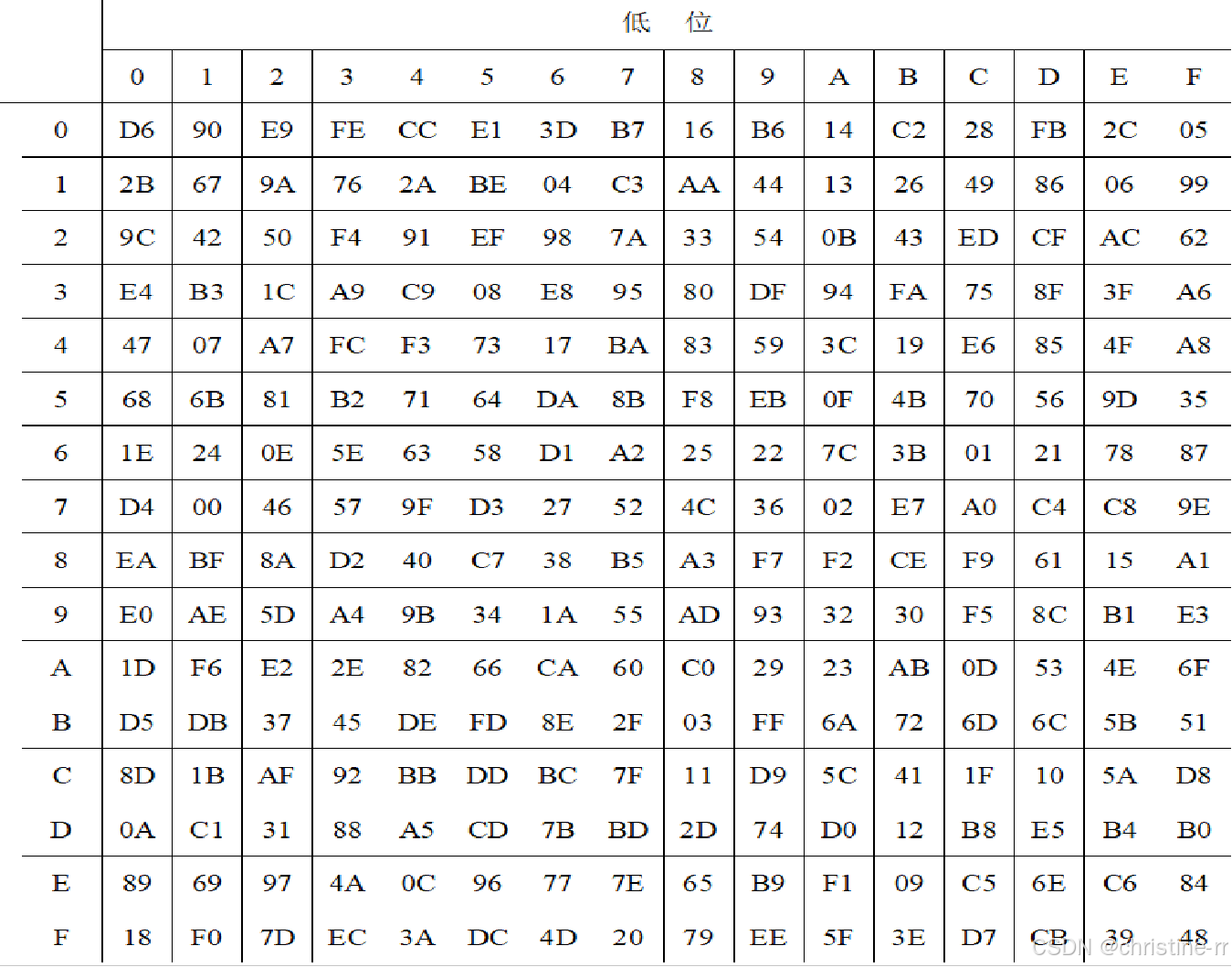
Project1 SM4算法的实现和优化

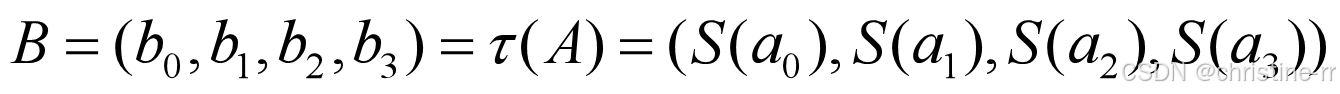
密码22.1 张靖陶 202200460011

1. SM4算法结构
2. S盒：



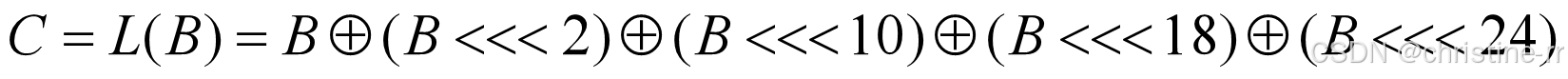
1. 非线性变换τ

由4个S盒并置构成。设输入为A=（）（4个32位的字），输出为B=（）（4个32位的字），则：

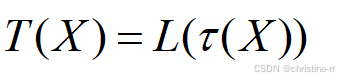


1. 线性变换部件L

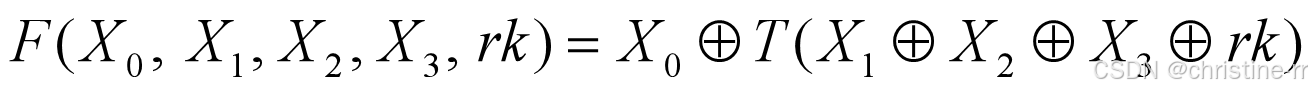
线性变换部件L是以字为处理单位的线性变换，其输入输出都是32位的字。设L的输入为B，输出为C：

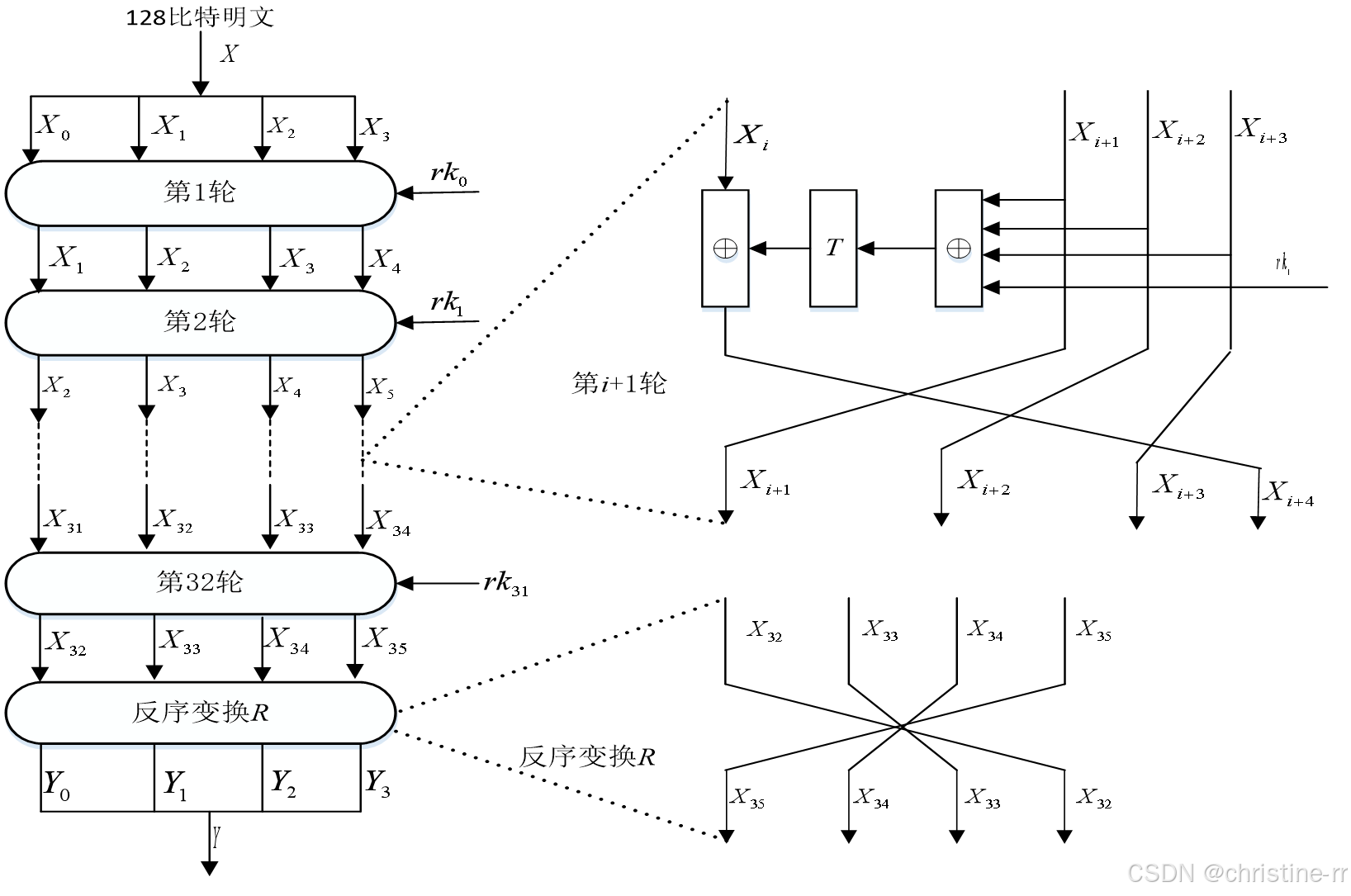


4、合成变换T



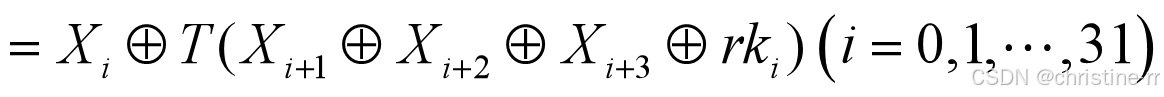
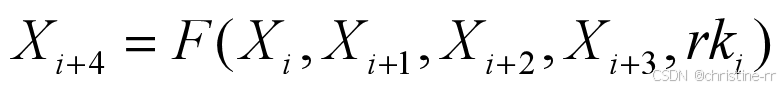
5、轮函数



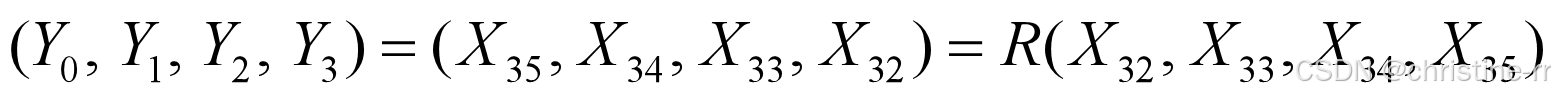


1. 加密算法

加密算法采用32轮迭代结构，每轮使用一个轮密钥。



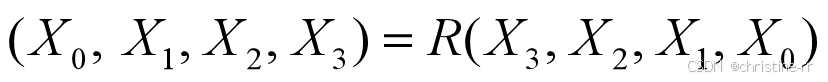
加密算法之后还需要一个反序处理：



7、解密算法

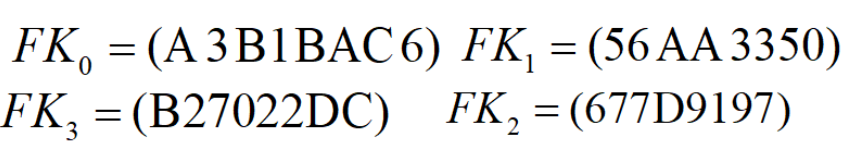
解密算法与加密算法相同，只是轮密钥的使用顺序相反，解密轮密钥是 加密轮密钥的逆序。

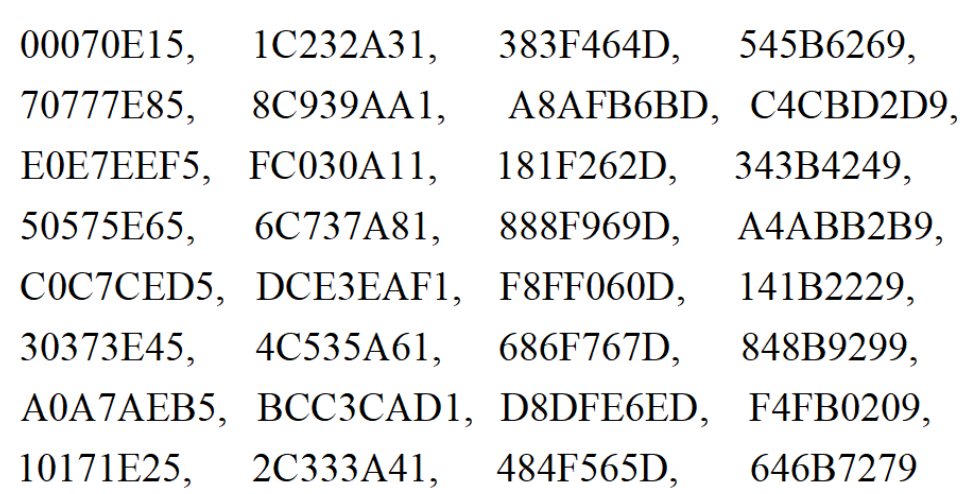
解密算法之后也需要一个反序处理：



8、密钥扩展算法

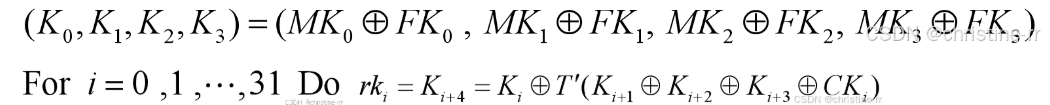
SM4算法加密时输入128位的密钥，采用32轮迭代结构，每一轮使用一个32位的轮密钥，共使用32个轮密钥。使用密钥扩展算法，从加密密钥产生出32个轮密钥。

常数FK：

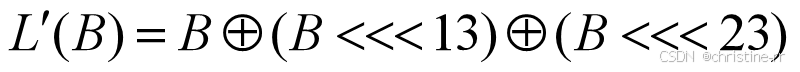


固定参数CK:

输入的加密密钥为MK,输出轮密钥为rk：



其中L与加密算法轮函数中的基本相同，只将其中的线性变化修改为：



1. 代码实现

|  |
| --- |
| #include <stdio.h> |
| #include <stdint.h> |
| #include <string.h> |
| #include <time.h> |
|  |
| #define BLOCK\_SIZE 16 |
| #define ROUND 32 |
|  |
| // 常数 |
| static const uint32\_t FK[4] = { |
| 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc |
| }; |
|  |
| // 固定参数 |
| static const uint32\_t 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 |
| }; |
|  |
| // S盒 |
| static const uint8\_t SBOX[256] = { |
| 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 |
| }; |
|  |
| // 循环左移 |
| static inline uint32\_t rotl32(uint32\_t value, uint8\_t shift) { |
| return (value << shift) | (value >> (32 - shift)); |
| } |
|  |
| // 线性变换 L |
| static uint32\_t linear\_transform(uint32\_t value) { |
| return value ^ rotl32(value, 2) ^ rotl32(value, 10) ^ |
| rotl32(value, 18) ^ rotl32(value, 24); |
| } |
|  |
| // 密钥扩展中的线性变换 L' |
| static uint32\_t key\_linear\_transform(uint32\_t value) { |
| return value ^ rotl32(value, 13) ^ rotl32(value, 23); |
| } |
|  |
| // S盒变换（4字节） |
| static uint32\_t sbox\_transform(uint32\_t value) { |
| uint32\_t result = 0; |
| for (int i = 0; i < 4; i++) { |
| uint8\_t byte = (value >> (24 - i \* 8)) & 0xFF; |
| result |= SBOX[byte] << (24 - i \* 8); |
| } |
| return result; |
| } |
|  |
| // 密钥扩展 |
| void key\_schedule(const uint8\_t\* key, uint32\_t\* round\_keys) { |
| uint32\_t k[4]; |
|  |
| // 将128位密钥转换为4个32位字 |
| for (int i = 0; i < 4; i++) { |
| k[i] = ((uint32\_t)key[4 \* i] << 24) | |
| ((uint32\_t)key[4 \* i + 1] << 16) | |
| ((uint32\_t)key[4 \* i + 2] << 8) | |
| key[4 \* i + 3]; |
| k[i] ^= FK[i]; // 异或系统参数 |
| } |
|  |
| // 生成32轮轮密钥 |
| for (int i = 0; i < ROUND; i++) { |
| uint32\_t tmp = k[1] ^ k[2] ^ k[3] ^ CK[i]; |
| tmp = sbox\_transform(tmp); // S盒变换 |
| tmp = key\_linear\_transform(tmp); // 线性变换L' |
|  |
| round\_keys[i] = k[0] ^ tmp; // 生成轮密钥 |
|  |
| // 更新密钥状态 |
| k[0] = k[1]; |
| k[1] = k[2]; |
| k[2] = k[3]; |
| k[3] = round\_keys[i]; |
| } |
| } |
|  |
| // 加密/解密单块（16字节） |
| void crypt(const uint8\_t\* input, uint8\_t\* output, const uint32\_t\* round\_keys, int decrypt) { |
| uint32\_t x[4]; |
|  |
| // 将128位输入转换为4个32位字 |
| for (int i = 0; i < 4; i++) { |
| x[i] = ((uint32\_t)input[4 \* i] << 24) | |
| ((uint32\_t)input[4 \* i + 1] << 16) | |
| ((uint32\_t)input[4 \* i + 2] << 8) | |
| input[4 \* i + 3]; |
| } |
|  |
| // 32轮迭代 |
| for (int round = 0; round < ROUND; round++) { |
| // 选择轮密钥（解密时逆序使用） |
| uint32\_t rk = decrypt ? round\_keys[ROUND - 1 - round] : round\_keys[round]; |
|  |
| uint32\_t tmp = x[1] ^ x[2] ^ x[3] ^ rk; |
| tmp = sbox\_transform(tmp); // S盒变换 |
| tmp = linear\_transform(tmp); // 线性变换L |
|  |
| uint32\_t new\_x = x[0] ^ tmp; // 生成新字 |
|  |
| // 更新状态 |
| x[0] = x[1]; |
| x[1] = x[2]; |
| x[2] = x[3]; |
| x[3] = new\_x; |
| } |
|  |
| // 最终反序变换 |
| uint32\_t temp = x[0]; |
| x[0] = x[3]; |
| x[3] = temp; |
|  |
| temp = x[1]; |
| x[1] = x[2]; |
| x[2] = temp; |
|  |
| // 将结果转换为字节输出 |
| for (int i = 0; i < 4; i++) { |
| output[4 \* i] = (x[i] >> 24) & 0xFF; |
| output[4 \* i + 1] = (x[i] >> 16) & 0xFF; |
| output[4 \* i + 2] = (x[i] >> 8) & 0xFF; |
| output[4 \* i + 3] = x[i] & 0xFF; |
| } |
| } |
|  |
| // 打印十六进制数据 |
| void print\_hex(const char\* label, const uint8\_t\* data, size\_t len) { |
| printf("%s: ", label); |
| for (size\_t i = 0; i < len; i++) { |
| printf("%02x", data[i]); |
| } |
| printf("\n"); |
| } |
|  |
| int main() { |
| // 标准测试向量 (GB/T 32907-2016) |
| uint8\_t key[BLOCK\_SIZE] = { |
| 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, |
| 0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10 |
| }; |
|  |
| uint8\_t plain[BLOCK\_SIZE] = { |
| 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, |
| 0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10 |
| }; |
|  |
| uint8\_t expected\_cipher[BLOCK\_SIZE] = { |
| 0x68, 0x1e, 0xdf, 0x34, 0xd2, 0x06, 0x96, 0x5e, |
| 0x86, 0xb3, 0xe9, 0x4f, 0x53, 0x6e, 0x42, 0x46 |
| }; |
|  |
| uint8\_t cipher[BLOCK\_SIZE]; |
| uint8\_t decrypted[BLOCK\_SIZE]; |
| uint32\_t round\_keys[ROUND]; |
|  |
| // 密钥扩展 |
| key\_schedule(key, round\_keys); |
|  |
| // 加密 |
| crypt(plain, cipher, round\_keys, 0); |
| print\_hex("明文", plain, BLOCK\_SIZE); |
| print\_hex("加密结果", cipher, BLOCK\_SIZE); |
| print\_hex("期望密文", expected\_cipher, BLOCK\_SIZE); |
|  |
| // 验证加密结果 |
| if (memcmp(cipher, expected\_cipher, BLOCK\_SIZE) == 0) { |
| printf("加密成功!\n"); |
| } |
| else { |
| printf("加密失败!\n"); |
| } |
|  |
| // 解密 |
| crypt(cipher, decrypted, round\_keys, 1); |
| print\_hex("解密结果", decrypted, BLOCK\_SIZE); |
|  |
| // 验证解密结果 |
| if (memcmp(decrypted, plain, BLOCK\_SIZE) == 0) { |
| printf("解密成功!\n"); |
| } |
| else { |
| printf("解密失败!\n"); |
| } |
|  |
| return 0; |
| } |

1. 优化思路

预计算S盒查找表：创建4个32位查找表，每个表包含256个条目。

每个表负责处理32位整数的不同字节位置：

SBOX\_TABLE0：处理最高字节（左移24位）

SBOX\_TABLE1：处理次高字节（左移16位）

SBOX\_TABLE2：处理次低字节（左移8位）

SBOX\_TABLE3：处理最低字节（不移位）

通过4次查表和3次位或操作完成整个32位值的S盒替换，消除循环开销和多次移位操作，提升了S盒处理速度。

1. 优化后部分代码

|  |
| --- |
| // 预计算32位S盒查找表 |
| static uint32\_t SBOX\_TABLE0[256]; |
| static uint32\_t SBOX\_TABLE1[256]; |
| static uint32\_t SBOX\_TABLE2[256]; |
| static uint32\_t SBOX\_TABLE3[256]; |
|  |
| // 初始化S盒查找表 |
| void init\_sbox\_tables() { |
| for (int i = 0; i < 256; i++) { |
| SBOX\_TABLE0[i] = (uint32\_t)SBOX[i] << 24; |
| SBOX\_TABLE1[i] = (uint32\_t)SBOX[i] << 16; |
| SBOX\_TABLE2[i] = (uint32\_t)SBOX[i] << 8; |
| SBOX\_TABLE3[i] = SBOX[i]; |
| } |
| } |
|  |
| // 优化的S盒变换 |
| static uint32\_t sbox\_transform\_new(uint32\_t value) { |
| return SBOX\_TABLE0[(value >> 24) & 0xFF] | |
| SBOX\_TABLE1[(value >> 16) & 0xFF] | |
| SBOX\_TABLE2[(value >> 8) & 0xFF] | |
| SBOX\_TABLE3[value & 0xFF]; |
| } |