SM4优化

实验目的

本实验旨在通过不同优化技术提升SM4加密算法的性能，包括：

1. 基础T-Table优化实现
2. SM4-GCM工作模式优化

SM4算法原理

SM4是一种分组密码算法，其分组长度为128位，密钥长度也为128位。其加解密过程采用了32轮迭代机制，每一轮需要一个轮密钥。

1. 加密

SM4采用32轮非线性迭代结构，每轮使用相同的轮函数处理128位数据分组，整体流程可分为：

- 密钥扩展（生成32个轮密钥）
- 初始变换（明文分组处理）
- 32轮迭代运算
- 反序输出（得到密文）

2. 解密（SM4采用Feistel结构，加解密仅轮密钥顺序不同，其他步骤完全一致。）

- 密钥扩展（与加密相同，生成32个轮密钥）
- 轮密钥逆序（`rk[31]→rk[0]`）
- 32轮迭代运算（与加密相同的轮函数）
- 反序输出（得到明文）

3. 密钥扩展算法

- 初始化：将128位密钥分成四个32位字`K[0]`到`K[3]`，并将这四个字分别与`FK`数组中的元素进行异或操作，得到扩展密钥的前四个字。
- 生成后续密钥：从第5个字开始，通过一个循环来计算剩下的28个字。对于每一个新的字`K[i]`（其中`i >= 4`），它的值等于`K[i-4]`与一个变换后的值的异或结果。这个变换包括对`K[i-1]`、`K[i-2]`、`K[i-3]`以及固定值`CK[i-4]`进行特定的函数处理。
- 轮密钥：在完成密钥扩展之后，我们得到了32个32位的轮密钥，这些轮密钥将按顺序用于加密过程中的每一轮。对于解密，轮密钥使用的顺序是反向的。

4. 轮函数

接受四个32位字X0,X1,X2,X3作为输入，并产生一个输出，这个输出将替换X0用于下一轮的计算。

$$F(X0, X1, X2, X3, rk) = X0 \oplus T(X1 \oplus X2 \oplus X3 \oplus rk)$$

合成置换函数T包含两部分：非线性的S盒变换以及线性的混合操作：

- S盒变换：首先，将输入分成四个8位的部分，每个部分通过一个固定的S盒进行替代。
- S盒变换后的结果会经过一个固定线性变换（循环移位）。

实验过程

1. 基础T-Table优化：过预计算S盒和线性变换的组合结果，将实时计算转变为查表操作。

将S盒查找、4次移位、5次异或合并为单次查表、3次异或，减少计算量，但会占用额外的内存开销用来存储T表。

代码块

```
    // T-Table预计算结果
uint32_t T_Table[4][256];

// 初始化T-Table
void InitTTable() {
    for (int i = 0; i < 256; ++i) {
        uint8_t b = SBOX[i];
        uint32_t y = (b << 24) | (b << 16) | (b << 8) | b;
        uint32_t t = y ^ Rotl(y, 2) ^ Rotl(y, 10) ^ Rotl(y, 18) ^ Rotl(y, 24);
        // 填充4个T-Table
        T_Table[0][i] = (t << 24) | (t >> 8);
        T_Table[1][i] = (t << 16) | (t >> 16);
        T_Table[2][i] = (t << 8) | (t >> 24);
        T_Table[3][i] = t;
    }
}
```

2. SM4-GCM工作模式优化：

SM4-GCM工作模式结合了SM4 的加密和GCM 的认证加密功能，实现了在保证数据机密性的同时，还能验证数据的完整性。GCM 模式允许指定一段不需要加密，但需要进行完整性验证的数据，这部分数据被称为附加认证数据(AAD)。SM4-GCM 使用CTR 模式进行加密，同时使用GHASH 函数对明文和 AAD 进行认证，生成一个认证标签(MAC)。最终的输出包括加密后的密文和生成的认证标签。

代码块

```
// GCM相关常量
const size_t BLOCK_SIZE = 16; // SM4块大小（16字节）
const size_t TAG_SIZE = 16;    // GMAC标签长度（通常为16字节）
```

```
4
5   // GCM工具函数
6   void XOR_block(const uint8_t* a, const uint8_t* b, uint8_t* result) {
7       for (size_t i = 0; i < BLOCK_SIZE; ++i) {
8           result[i] = a[i] ^ b[i];
9       }
10  }
11
12  void IncrementCounter(uint8_t* counter) {
13      for (int i = BLOCK_SIZE - 1; i >= 0; --i) {
14          if (++counter[i] != 0) break; // 处理进位
15      }
16  }
17
18  // SM4-CTR加密（核心函数）
19  void SM4_CTR_Crypt(const uint8_t* input, uint8_t* output, size_t length,
20                     const uint8_t* nonce, const uint32_t rk[32]) {
21      uint8_t counter[BLOCK_SIZE];
22      uint8_t keystream[BLOCK_SIZE];
23      memcpy(counter, nonce, BLOCK_SIZE);
24
25      for (size_t i = 0; i < length; i += BLOCK_SIZE) {
26          SM4Crypt(counter, keystream, rk);  // 加密计数器生成密钥流
27          size_t block_size = std::min(BLOCK_SIZE, length - i);
28          for (size_t j = 0; j < block_size; ++j) {
29              output[i + j] = input[i + j] ^ keystream[j]; // 异或加密
30          }
31          IncrementCounter(counter);          // 计数器递增
32      }
33  }
34
35  // GMAC计算（基于GF(2^128)乘法）
36  void GMAC(const uint8_t* data, size_t data_len, const uint8_t* key, uint8_t*
    tag) {
37      // 1. 生成轮密钥
38      uint32_t rk[32];
39      ExpandKey(key, rk); // 将原始密钥扩展为轮密钥
40
41      // 2. 计算哈希键H（加密全0块）
42      uint8_t zero_block[BLOCK_SIZE] = {0};
43      uint8_t H[BLOCK_SIZE];
44      SM4Crypt(zero_block, H, rk); // 使用轮密钥rk加密
45
46      // 3. 初始化tag为0
47      memset(tag, 0, TAG_SIZE);
48
49      // 4. 处理数据块（简化为异或，实际需实现GF乘法）
```

```
    for (size_t i = 0; i < data_len; i += BLOCK_SIZE) {
        uint8_t block[BLOCK_SIZE];
        size_t block_len = std::min(BLOCK_SIZE, data_len - i);
        memcpy(block, data + i, block_len);
        if (block_len < BLOCK_SIZE) {
            memset(block + block_len, 0, BLOCK_SIZE - block_len); // 填充
        }
        XOR_block(tag, block, tag); // 实际需替换为GF(2^128)乘法
    }
}

// SM4-GCM加密
void SM4_GCM_Encrypt(
    const uint8_t* plaintext, size_t plaintext_len,
    const uint8_t* key, const uint8_t* nonce,
    uint8_t* ciphertext, uint8_t* tag
) {
    uint32_t rk[32];
    ExpandKey(key, rk); // 生成轮密钥

    // 1. 加密明文（CTR模式）
    SM4_CTR_Crypt(plaintext, ciphertext, plaintext_len, nonce, rk);

    // 2. 计算GMAC标签（包括附加数据和密文）
    GMAC(ciphertext, plaintext_len, key, tag);
}
```