创新创业实践实验报告一

姓名：王树森

学院：网络空间安全

专业：密码科学与技术

学号：202200460004

2025.7.17

一、SM4 的实现

1. **私有静态常量定义**

SM4 算法依赖预定义的常量，需在类外初始化这些静态成员：

（1）Sbox（S 盒）：非线性字节替换表，256 个字节的固定置换，用于实现 tau 变换。

```
// S盒定义（完整国标S盒）
const uint8_t SM4::Sbox[256] = {
    0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0xfa, 0x3e, 0x5a, 0x58, 0x80, 0x8c, 0x93, 0x8f,
    0xd7, 0xfb, 0x27, 0x0e, 0x64, 0x9e, 0xea, 0x5f, 0x38, 0x84, 0x7e, 0x4c, 0xe2, 0xcf, 0x44, 0x09,
    0xdd, 0x26, 0x97, 0x56, 0xf4, 0xeea, 0x65, 0x7a, 0xae, 0x08, 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6,
    0xb4, 0xc2, 0x92, 0xd3, 0xac, 0x0d, 0x43, 0x0c, 0xf0, 0x2c, 0x1e, 0x81, 0x33, 0x89, 0x60, 0x45,
    0x9b, 0x00, 0x86, 0x19, 0xe3, 0x66, 0x0a, 0x13, 0xfc, 0x56, 0x95, 0x17, 0x1a, 0x05, 0x9a, 0x47,
    0x8d, 0x7b, 0x31, 0xb2, 0x16, 0xc9, 0x51, 0x69, 0x83, 0x9f, 0xd4, 0xcf, 0x0f, 0xb6, 0xc1, 0x1d,
    0x2a, 0xcb, 0x73, 0x5b, 0xa0, 0x8b, 0x72, 0x0e, 0x55, 0x2f, 0xc4, 0x91, 0xaf, 0xc7, 0x71, 0x1d,
    0x24, 0x75, 0xb8, 0xe5, 0x54, 0x79, 0x3c, 0x8e, 0x4e, 0x7f, 0x3e, 0x0b, 0x4b, 0x70, 0x56, 0x9e,
    0x34, 0x1a, 0x04, 0xc3, 0x5d, 0x65, 0xd0, 0xe0, 0x22, 0x9f, 0xac, 0x74, 0x0f, 0x02, 0x18, 0xbe,
    0x1b, 0x6b, 0x3a, 0x96, 0x48, 0x07, 0x06, 0x5e, 0x7a, 0xbc, 0x72, 0x1f, 0xc8, 0x50, 0x57, 0x67,
    0x5c, 0xf1, 0x01, 0x7f, 0x23, 0x7d, 0x8b, 0x37, 0x94, 0x91, 0xf2, 0x10, 0x03, 0xd8, 0x9a, 0xe6,
    0x42, 0x2d, 0xc5, 0xdf, 0x69, 0x21, 0x87, 0x9b, 0x76, 0x04, 0x90, 0x09, 0x7c, 0x0a, 0xf3, 0x71,
    0x92, 0x14, 0x6c, 0x4e, 0x08, 0x2e, 0xaa, 0x16, 0xd2, 0x0c, 0x46, 0xcb, 0x29, 0x03, 0xdb, 0x58,
    0x33, 0x88, 0x6e, 0x1e, 0xcf, 0x1f, 0xad, 0xd7, 0x80, 0xc7, 0x1d, 0xa7, 0x41, 0x52, 0x3b, 0x27,
    0xf5, 0x3e, 0x84, 0x24, 0x7e, 0x54, 0x81, 0x26, 0x66, 0x36, 0x83, 0x0f, 0x02, 0x48, 0x68, 0x06,
    0xba, 0x50, 0x55, 0x45, 0x6a, 0x5f, 0x8a, 0x11, 0xd9, 0x29, 0x0b, 0x5b, 0x63, 0xca, 0x1c, 0x73
};
```

（2）固定参数 FK 和 CK：

```
const uint32_t SM4::FK[4] = { 0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc };
```

```
// 固定参数CK
const uint32_t SM4::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
};
```

2. 字节与 32 位字转换函数

用于处理 32 位整数与 4 个字节的拆分 / 组合:

（1）get_uint8：从 32 位整数中提取第 i 个字节（i=0 为最低位字节，i=3 为最高位字节）

```
// 从32位整数中提取第i个字节（0-3）
uint8_t SM4::get_uint8(uint32_t x, int i) {
    return (x >> (8 * (3 - i))) & 0xff;
}
```

（2）put_uint32：将 4 个字节组合为 32 位整数（按 b0 为最低位，b3 为最高位）

```
// 将4个字节合并为32位整数
uint32_t SM4::put_uint32(uint8_t b0, uint8_t b1, uint8_t b2, uint8_t b3) {
    return (static_cast<uint32_t>(b0) << 24) |
        (static_cast<uint32_t>(b1) << 16) |
        (static_cast<uint32_t>(b2) << 8) |
        static_cast<uint32_t>(b3);
}
```

3. 核心变换函数实现

（1）tau 变换（字节替换）：对 32 位输入的每个字节应用 S 盒置换:

```
// 非线性变换tau（密钥扩展用）
uint32_t SM4::tau(uint32_t x) {
    uint8_t bytes[4];
    for (int i = 0; i < 4; ++i) {
        bytes[i] = Sbox[get_uint8(x, i)];
    }
    return put_uint32(bytes[0], bytes[1], bytes[2], bytes[3]);
}
```

（2）L 变换（线性变换，用于轮函数）：对 tau 的输出进行线性移位异或:

```
// 线性变换L（加密用）
uint32_t SM4::L(uint32_t x) {
    return x ^ ((x << 2) | (x >> 30)) ^ ((x << 10) | (x >> 22)) ^
        ((x << 18) | (x >> 14)) ^ ((x << 24) | (x >> 8));
}
```

（3）L' 变换（线性变换，用于密钥扩展）：与 L 类似但移位参数不同：

```
// 线性变换L'（密钥扩展用）
uint32_t SM4::L_prime(uint32_t x) {
    return x ^ ((x << 13) | (x >> 19)) ^ ((x << 23) | (x >> 9));
}
```

（4）轮函数 F：加密轮迭代的核心计算，输入 4 个 32 位字和轮密钥，输出下一轮的字：

```
// 轮函数F
uint32_t SM4::F(uint32_t x0, uint32_t x1, uint32_t x2, uint32_t x3, uint32_t rk) {
    uint32_t t = x0 ^ x1 ^ x2 ^ x3 ^ rk;
    t = tau(t);  // 非线性变换
    return L(t); // 线性变换
}
```

4. 构造函数（密钥扩展）

（1）输入密钥合法性校验

构造函数接收 std::vector<uint8_t>& key 作为参数，首先验证密钥的有效性：

检查密钥长度是否为 16 字节（128 位，SM4 算法规定的密钥长度）。若长度不符（如小于 16 字节或大于 16 字节），通常会抛出异常或通过断言（assert）终止，确保后续操作基于合法密钥。

```
SM4::SM4(const std::vector<uint8_t>& key) {
    if (key.size() != 16) {
        throw std::invalid_argument("SM4 key must be 16 bytes");
    }
```

（2）密钥格式转换和初始密钥与固定参数异或

将 16 字节密钥转换为 4 个 32 位无符号整数（uint32_t），作为密钥扩展的初始输入。SM4 密钥扩展需使用固定参数 FK[4]（头文件中声明的静态常量），对步骤 2 得到的 K0~K3 进行初始混淆：

```
uint32_t k[36];
for (int i = 0; i < 4; ++i) {
    k[i] = put_uint32(key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]) ^ FK[i];
}
```

（3）生成 32 个轮密钥

通过 32 次迭代生成轮密钥 rk[0]~rk[31]，每次迭代依赖前 3 个中间密钥字、轮常量 CK[i]以及变换函数 tau 和 L_prime。

迭代公式（对 i=0 到 31）：

k[i+4] = k[i] ^ L_prime(tau(k[i+1] ^ k[i+2] ^ k[i+3] ^ CK[i]));

rk[i] = k[i+4];

```
for (int i = 0; i < 32; ++i) {
    k[i + 4] = k[i] ^ L_prime(tau(k[i + 1] ^ k[i + 2] ^ k[i + 3] ^ CK[i]));
    rk[i] = k[i + 4];
}
```

5. 加密函数（encrypt）

（1）输入合法性校验

检查 plaintext 的长度是否为 16 字节（SM4 的分组长度固定为 128 位，即 16 字节），若长度不符可能抛出异常或返回错误（具体实现依赖内部逻辑）。确保输出 ciphertext 缓冲区已分配至少 16 字节空间（或在函数内初始化，保证能存储密文结果）。

```
void SM4::encrypt(const std::vector<uint8_t>& plaintext, std::vector<uint8_t>& ciphertext) {
    if (plaintext.size() != 16 || ciphertext.size() != 16) {
        throw std::invalid_argument("SM4 plaintext/ciphertext must be 16 bytes");
    }
```

（2）明文数据预处理

将 16 字节明文转换为 4 个 32 位无符号整数（uint32_t），作为加密迭代的初始状态。调用 put_uint32 函数（头文件中声明的静态工具函数）完成转换：

明文的前 4 字节（plaintext[0]~plaintext[3]）→ 第一个 32 位字 X0；

接下来 4 字节（plaintext[4]~plaintext[7]）→ 第二个 32 位字 X1；

再接下来 4 字节（plaintext[8]~plaintext[11]）→ 第三个 32 位字 X2；

最后 4 字节（plaintext[12]~plaintext[15]）→ 第四个 32 位字 X3。

```
uint32_t x[4];
for (int i = 0; i < 4; ++i) {
    x[i] = put_uint32(plaintext[4 * i], plaintext[4 * i + 1], plaintext[4 * i + 2], plaintext[4 * i + 3]);
}
```

（3）32 轮迭代运算

基于初始状态 X0~X3，使用 32 个轮密钥（rk[0]~rk[31]，由构造函数通过密钥扩展生成）进行 32 轮迭代，每轮生成一个新的 32 位字，迭代公式为：

Xi+4 = F(Xi, Xi+1, Xi+2, Xi+3, rk[i])

轮函数 F 是单轮迭代的核心，接收 4 个 32 位字和 1 个轮密钥，输出 1 个 32 位字，具体步骤如下：

密钥混入与异或：计算中间值 B = Xi XOR Xi+1 XOR Xi+2 XOR rk[i]；

tau 变换（字节替换）：调用 tau(B)对 B 进行非线性置换：

- tau 函数通过 get_uint8 将 32 位字 B 拆分为 4 个 8 位字节（b0, b1,

b2, b3)；

- 每个字节通过 Sbox（头文件中定义的 substitution box）置换；
- 再通过 put_uint32 将置换后的字节重组为 32 位字 C，即 C = tau(B)。

L 变换（线性扩散）：调用 L(C)对 C 进行线性扩散：

根据 SM4 标准，L 函数定义为：

L(C) = C XOR (C <<< 2) XOR (C <<< 10) XOR (C <<< 18) XOR (C <<< 24)

输出本轮结果：计算 Xi+4 = L(C) XOR Xi+3，即本轮迭代的输出为线性扩散结果与第 4 个输入字的异或。

```
for (int i = 0; i < 32; ++i) {
    uint32_t x4 = F(x[0], x[1], x[2], x[3], rk[i]);
    x[0] = x[1];
    x[1] = x[2];
    x[2] = x[3];
    x[3] = x4;
}
```

（4）迭代后数据反序处理

32 轮迭代结束后，得到 4 个 32 位字 X32, X33, X34, X35。根据 SM4 算法规定，需将这 4 个字反序为 X35, X34, X33, X32，作为密文的 32 位字表示。算法设计中，加密的最终输出需通过反序抵消迭代过程中的状态累积偏差，确保密文与明文的映射对称性。

```
for (int i = 0; i < 4; ++i) {
    uint32_t val = x[3 - i];
    ciphertext[4 * i] = get_uint8(val, 0);
    ciphertext[4 * i + 1] = get_uint8(val, 1);
    ciphertext[4 * i + 2] = get_uint8(val, 2);
    ciphertext[4 * i + 3] = get_uint8(val, 3);
}
```

6. 解密函数（decrypt）

（1）输入合法性校验

首先验证输入密文（ciphertext）的长度是否为 16 字节（128 位，SM4 的分组长度），输出明文（plaintext）的缓冲区需提前分配 16 字节空间（若未分配，可能在函数内初始化）。

```
void SM4::decrypt(const std::vector<uint8_t>& ciphertext, std::vector<uint8_t>& plaintext)
    if (ciphertext.size() != 16 || plaintext.size() != 16) {
        throw std::invalid_argument("SM4 ciphertext/plaintext must be 16 bytes");
    }
```

（2）密文数据预处理（字节→32 位字转换）

将 16 字节密文转换为 4 个 32 位字（X0, X1, X2, X3），作为解密的初始状态。实现依赖 put_uint32 函数：将密文的 16 字节按顺序拆分为 4 组（每组 4 字节），每组转换为 1 个 32 位无符号整数（uint32_t）。

```
// 密文拆分为4个32位字
uint32_t x[4];
for (int i = 0; i < 4; ++i) {
    x[i] = put_uint32(ciphertext[4 * i], ciphertext[4 * i + 1], ciphertext[4 * i + 2], ciphertext[4 * i + 3]);
}
```

（3）32 轮解密迭代

与加密的轮迭代逻辑相同，但轮密钥使用顺序相反（从 rk[31]到 rk[0]）。每轮迭代公式为：

Xi+4 = F(Xi, Xi+1, Xi+2, Xi+3, rk[31 - i])

对每轮输入的 4 个 32 位字（Xi, Xi+1, Xi+2, Xi+3）和当前轮密钥（rk[31 - i]），执行以下子步骤：

密钥混入与异或：计算 B = Xi XOR Xi+1 XOR Xi+2 XOR rk[31 - i]

tau 变换（字节替换）：对 B 的 4 个字节分别应用 S 盒置换，得到 C = tau(B)

L 变换（线性扩散）：对 C 执行线性扩散，得到 D = L(C)

输出本轮结果：Xi+4 = D XOR Xi+3

```
for (int i = 0; i < 32; ++i) {
    uint32_t x4 = F(x[0], x[1], x[2], x[3], rk[31 - i]);
    x[0] = x[1];
    x[1] = x[2];
    x[2] = x[3];
    x[3] = x4;
}
```

（32） **迭代后数据反序处理加输出**

32 轮迭代结束后，得到 4 个 32 位字 X32, X33, X34, X35。需将其反序为 X35, X34, X33, X32，作为明文的 32 位字表示。加密过程的最后一步是将迭代结果 X32~X35 反序得到密文，因此解密需反向操作以还原明文。

```
for (int i = 0; i < 4; ++i) {
    uint32_t val = x[3 - i];
    plaintext[4 * i] = get_uint8(val, 0);
    plaintext[4 * i + 1] = get_uint8(val, 1);
    plaintext[4 * i + 2] = get_uint8(val, 2);
    plaintext[4 * i + 3] = get_uint8(val, 3);
}
```