

Project1 SM4 的软件实现和优化

一．文件说明

SM4.cpp 基础 sm4 算法实现
SIMD.cpp 使用 SIMD 优化的 sm4 算法
T_table.cpp 使用 T_table 优化的 sm4 算法
sm4_gcm.cpp GCM 工作模式下的 sm4 算法优化

二. sm4 原理

SM4 加密算法是采用分组加密的方式，每一个分组的长度为 128bit，密钥长度也为 128bit。理想化输入方式，输入的明文按照 128bit 进行分组，将每组按照字(32bit)分为 4 个字。故输入明文简化为 $X = (X_0, X_1, X_2, X_3)$ ，其中 X_i 均为 32bit。

加密流程：

输入:4 字明文(X_0, X_1, X_2, X_3)

迭代过程(32 次){

第一次：使用前 4 字明文和第一轮轮密钥计算第 5 个字， $X_4 = F(X_0, X_1, X_2, X_3, rk_0)$;

第二次：使用前 4 字和第二轮轮密钥计算第 6 个字， $X_5 = F(X_1, X_2, X_3, X_4, rk_1)$;

...

第三十二次：使用前 4 字和第三十二轮轮密钥计算第 36 个字， $X_{35} = F(X_{31}, X_{32}, X_{33}, X_{34}, rk_{31})$;

反序变换： $(Y_0, Y_1, Y_2, Y_3) = R(X_{35}, X_{34}, X_{33}, X_{32})$ 。

F 函数：

F 函数称为轮函数， \oplus 代表按位异或，T 函数是一个合成变换，由一个非线性变换和线性变换复合而成。
$$F(X_0, X_1, X_2, X_3, rk) = X_0 \oplus T(X_1 \oplus X_2 \oplus X_3 \oplus rk)$$

T 函数：

T 函数的输入为一个字(32 bit)，假设输入为 $A = (a_0, a_1, a_2, a_3)$ ， a_i 为一个字节(8bit)。T 函数首先进行非线性变换，使用一个 S 盒来实现字节替换，得到输出 $B = (S(a_0), S(a_1), S(a_2), S(a_3))$ ，之后将输出进行一次线性变换使用循环左移变换实现，如下：

$$B = L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24).$$

轮密钥的生成：

假设初始加密密钥为 $MK = (MK_0, MK_1, MK_2, MK_3)$ ；

将初始加密密钥与系统参数异或， $(K_0, K_1, K_2, K_3) = (MK_0 \oplus FK_0, MK_1 \oplus FK_1, MK_2 \oplus FK_2, MK_3 \oplus FK_3)$ ；其中系统参数的值是确定的。

轮密钥生成方式：

迭代过程(32 次){

第一次：使用前 4 字(K_0, K_1, K_2, K_3)计算第 5 个字 K_4 ，并将 K_4 作为第一轮轮密钥 rk_0 。
$$rk_0 = K_4 = K_0 \oplus T'(K_1 \oplus K_2 \oplus K_3 \oplus CK_0)$$

第二次：使用前 4 字(K_1, K_2, K_3, K_4)计算第 6 个字 K_5 ，并将 K_5 作为第二轮轮密钥 rk_1 。
$$rk_1 = K_5 = K_1 \oplus T'(K_2 \oplus K_3 \oplus K_4 \oplus CK_1)$$

...

第三十二次：使用前 4 字(K31, K32, K33, K34)计算第 36 个字 K35, 并将 K35 作为第三十二轮轮密钥 rk31。

T 变换是将合成置换中 T 中的线性变换 L 换成 L', 其余不变。

$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 23)$$

解密过程与加密流程大体相同, 使用的密钥顺序相反, 在这里我们使用 mod 标志位选择密钥使用顺序, 来实现加密和解密的功能。

三.优化思路总结:

1. T-table 优化 SM4 单块加密

- 原理:

将 S-box 与线性变换 L (轮函数) 提前组合成查表 (T-table), 每轮只做 4 次表查 + 异或即可。

- T1_table 用于轮函数加密
- T2_table 用于密钥扩展

- 效果:

- 减少每轮中多次位移和 S-box 查表操作
- 提高 CPU 缓存命中率
- 流水线利用率提升

2. T-table 优化密钥扩展

- 原理:

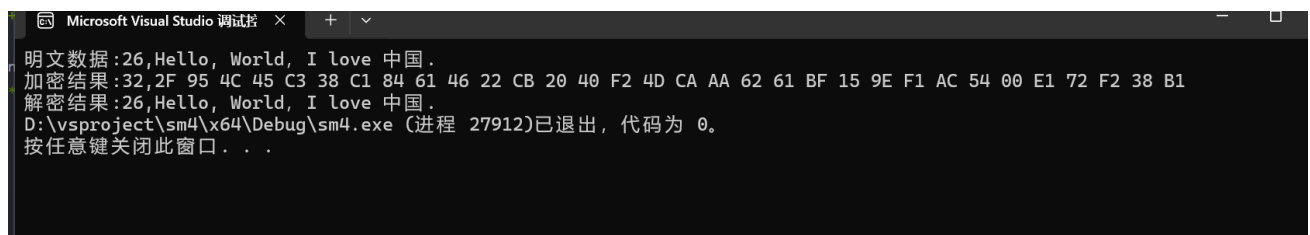
- 将密钥扩展中 S-box + 线性变换组合为 T2_table
- 直接查表生成轮密钥，无需重复循环和位移运算
- 效果：
 - 密钥扩展速度明显提升
 - 对多次加密场景（同密钥不同 IV）特别有效

3.SIMD 优化

- 当前：
 - 每 16 字节 block 调用一次单块 SM4
 - 异或生成密文
- 优化：
 - 批量生成 keystream（SIMD 并行）
 - 使用 _mm_xor_si128 进行 16 字节异或加速

四. 代码运行结果

具体实现时我选择了 CBC 工作模式，顺利实现 sm4 算法基本功能：



```
Microsoft Visual Studio 调试  x + -
明文数据:26,Hello, World, I love 中国.
加密结果:32,2F 95 4C 45 C3 38 C1 84 61 46 22 CB 20 40 F2 4D CA AA 62 61 BF 15 9E F1 AC 54 00 E1 72 F2 38 B1
解密结果:26,Hello, World, I love 中国.
D:\vsproject\sm4\x64\Debug\sm4.exe (进程 27912)已退出，代码为 0。
按任意键关闭此窗口...
```

相同明文长度（160000）下记录加密解密所花时间：

SM4.cpp

```
Microsoft Visual Studio 调试器
加密所用时间 9473微秒
解密所用时间：9140微秒

D:\vsproject\sm4\x64\Debug\sm4.exe (进程 27608)已退出，代码为 0。
按任意键关闭此窗口...
```

SIMD.cpp

```
Microsoft Visual Studio 调试器
加密所用时间 6574微秒
解密所用时间：6616微秒

D:\vsproject\SIMD\simd\x64\Debug\simd.exe (进程 57820)已退出，代码为 0。
按任意键关闭此窗口...
```

T_table.cpp

```
Microsoft Visual Studio 调试器
加密所用时间：2290 微秒
解密所用时间：2035 微秒

D:\vsproject\ttable\ttable\x64\Debug\ttable.exe (进程 46572)已退出，代码为 0。
按任意键关闭此窗口...
```

GCM 工作模式下的 sm4 算法（优化前）

```
Microsoft Visual Studio 调试器 × + ▾
SM4-GCM 加密耗时：80758 微秒
SM4-GCM 解密+验证耗时：81479 微秒
Tag 验证：OK
Plaintext 恢复正确

D:\vsproject\SIMD\gcm\x64\Debug\gcm.exe (进程 50756)已退出，代码为 0。
按任意键关闭此窗口...
```

GCM 工作模式下的 sm4 算法（优化后）

```
Microsoft Visual Studio 调试器 × + ▾
SM4-GCM(T-table) 加密耗时：71045 微秒
SM4-GCM(T-table) 解密+验证耗时：73215 微秒
Tag 验证：OK
Plaintext 恢复正确

D:\vsproject\SIMD\gcm\x64\Debug\gcm.exe (进程 54164)已退出，代码为 0。
按任意键关闭此窗口...
```

五.优化效果分析

优化方法	所用时间	优化效果	优化倍率
无	9364us	无	无
SIMD	6583us	-2781	1.42
T_table	2140us	-7224	4.37