说明文档

一、实验目标

1. 实现 SM4 对称加密算法。
2. 优化 SM4 的性能，采用以下技术：
    - T-table 查表加速
    - AES-NI 指令集并行化
    - 最新指令集（如 GFNI、VPROLD）优化
3. 实现 SM4-GCM 模式并尝试优化。

二、基本实现

1. 单块加密

- SM4 使用 128-bit 明文和 128-bit 密钥。
- 明文分为 4 个 32-bit 分组 `X0,X1,X2,X3`。
- 32 轮迭代加密，每轮使用轮密钥 `rk[i]` 结合非线性 S 盒变换和线性变换 L（或 T-table 优化）。
- 输出密文为 16 字节。

核心函数

```
void SM4::encrypt_block(const uint8_t* in, uint8_t* out);
```

- 实现逻辑：
    1. 将字节转换为 32-bit 分组
    2. 进行 32 轮迭代，每轮执行 `X[i] ^= T_table_transform(X[j] ^ X[k] ^ X[l] ^ rk[i])`
    3. 将分组转回字节输出

---

2. 轮密钥扩展

```
void SM4::key_expansion(const std::array<uint32_t,4>& key);
```

- 输入 128-bit 密钥，生成 32 轮轮密钥 `rk[32]`

---

3. T-table 优化

- 将 S 盒 + 线性变换 L 预计算成 256 条 32-bit T-table
- 避免每轮重复计算 S 盒 + L，提高性能

```
void SM4::init_t_table();
uint32_t SM4::T_table_transform(uint32_t x);
```

- 输入 32-bit 数据，直接查表进行非线性 + 线性变换

三、性能优化

1. 并行加密

- 批量处理 4 块 16 字节数据，每块用 256-bit 寄存器存放
- 利用 `_mm256_xor_si256` 执行轮密钥 XOR，实现 SIMD 并行

```
void SM4::encrypt_blocks_avx2(const uint8_t* in, uint8_t* out, int num_blocks);
```

- 循环处理每 4 块，剩余不足 4 块使用单块加密

四、SM4-GCM 实现

1. Counter 模式加密

- 将计数器生成 16 字节块，每块 XOR 明文生成密文
- 批量 4 块使用 AVX2 并行加密，剩余块使用单块加密

```
void SM4_GCM_AVX2::encrypt(const std::vector<uint8_t>& plaintext,
                           std::vector<uint8_t>& ciphertext,
                           std::array<uint8_t,16>& tag);
```

- 优化点：
  - AVX2 并行加密多个计数器
  - 内存连续性利用缓存，提高吞吐量

---

2. GHASH 认证标签计算

- 将密文按 16 字节块进行 Galois 域累加
- 实现：用 `_mm_xor_si128` 做 SIMD XOR，占位 GF(2^128) 乘法

```
void SM4_GCM_AVX2::compute_ghash_avx2(const uint8_t* data, size_t len,
std::array<uint8_t,16>& tag);
```

- 优化点：
  - 批量 4 块 SIMD 并行
  - GFNI 可进一步加速 128-bit GF(2^128) 乘法

五、实验结果

```
Ciphertext: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1e d0 d9 3d c3 82 25 26 27 98 3a d4 11 62 2f 2e 2f 70 31 32 33 34 35 36 37 88
 ef 6b d3 c3 42 3e 3f
Tag: b0 f8 1f e0 a6 00 00 00 00 c5 af d2 b1 7d 00 01
```