# Project 1

## SM4软件实现与优化

SM4的基本实现中，有128位分组密码，32轮迭代。轮函数为：

uint32\_t RoundFunction(uint32\_t x0, uint32\_t x1, uint32\_t x2, uint32\_t x3, uint32\_t rk) {

uint32\_t t = x1 ^ x2 ^ x3 ^ rk;

return x0 ^ LinearTransform(SboxTransform(t));

}

其中：

SboxTransform：4个8-bit S盒并行。

LinearTransform：32-bit循环移位异或。

优化策略

### T-table优化

预计算4个1KB表（共4KB）：

T0[i] = LinearTransform((uint32\_t)Sbox[i] << 24);

T1[i] = LinearTransform((uint32\_t)Sbox[i] << 16);

T2[i] = LinearTransform((uint32\_t)Sbox[i] << 8);

T3[i] = LinearTransform((uint32\_t)Sbox[i]);

轮函数简化为：

uint32\_t t = x1 ^ x2 ^ x3 ^ rk;

uint32\_t t\_out = T0[t & 0xFF] ^ T1[(t >> 8) & 0xFF] ^

T2[(t >> 16) & 0xFF] ^ T3[t >> 24];

return x0 ^ t\_out;

1. table优化占用4KB缓存达到减少S盒盒线性变换计算开销效果。

### AES-NI优化S盒

SM4 S盒分解，使用AES-NI指令实现仿射变换：

\_\_m128i sm4\_sbox(\_\_m128i x) {

x = \_mm\_aesenclast\_si128(\_mm\_aesimc\_si128(x), affine\_const);

}

AES-NI优化S盒适用于无GFNI支持的CPU。

### GDNI指令优化

单指令完成S盒

\_\_m512i sbox\_out = \_mm512\_gf2p8affineinv\_epi64\_epi8(

input,

\_mm512\_set1\_epi64(AFFINE\_MATRIX),

AFFINE\_CONST

);

完整轮函数向量化（AVX512）

\_\_m512i t = \_mm512\_xor\_epi32(x1, \_mm512\_xor\_epi32(x2, \_mm512\_xor\_epi32(x3, rk)));

\_\_m512i sbox\_out = \_mm512\_gf2p8affineinv\_epi64\_epi8(t, ...);

\_\_m512i L\_out = sbox\_out;

L\_out = \_mm512\_xor\_epi32(L\_out, \_mm512\_rolv\_epi32(sbox\_out, \_mm512\_set1\_epi32(2)));

L\_out = \_mm512\_xor\_epi32(L\_out, \_mm512\_rolv\_epi32(sbox\_out, \_mm512\_set1\_epi32(10)));

L\_out = \_mm512\_xor\_epi32(L\_out, \_mm512\_rolv\_epi32(sbox\_out, \_mm512\_set1\_epi32(18)));

L\_out = \_mm512\_xor\_epi32(L\_out, \_mm512\_rolv\_epi32(sbox\_out, \_mm512\_set1\_epi32(24)));

\_\_m512i x4 = \_mm512\_xor\_epi32(x0, L\_out);

## b）SM4-GCM优化实现

SM4-GCM结合计数器模式（CTR）加密盒Galois哈希（GHASH）认证，优化关键在于并行处理加密和认证操作。

### 优化思路：

1. CTR模式优化

批处理加密：一次性处理16个计数器块（256字节）

并行密钥流生成：利用AVX2指令集并行加密多个块

向量化计数器管理：使用SIMD指令高效更新计数器

1. GHASH优化

预计算表：预计算H的幂次表（4KB）

向量化GF乘法：使用PCLMULQDQ指令

分组聚合：讲多个块分组并行计算

1. 异步处理

加密和认证并行

多缓冲区处理减少内存延迟

### 关键代码：

1、批量计数器处理

void ctr\_mode(uint8\_t\* out, const uint8\_t\* in, size\_t len, uint8\_t counter[16]) {

constexpr size\_t BLOCK\_BATCH = 16; // 一次处理16个块

uint8\_t counters[BLOCK\_BATCH \* 16];

for (size\_t i = 0; i < len; i += BLOCK\_BATCH \* 16) {

// 准备计数器块

prepare\_counters(counters, counter, current\_batch);

// 生成密钥流

sm4.encrypt\_16blocks(keystream, counters, enc\_rk);

// 异或加密

for (size\_t j = 0; j < bytes\_to\_process; j++) {

out[i + j] = in[i + j] ^ keystream[j];

}

// 更新主计数器

update\_counter(counter, current\_batch);

}

}

1. GHASH分组聚合

\_\_m128i process\_group(const uint8\_t\* data, size\_t block\_count, const \_\_m128i& state) {

\_\_m128i group\_state = \_mm\_setzero\_si128();

// Horner方法计算组内聚合

for (size\_t i = 0; i < block\_count; i++) {

\_\_m128i block = \_mm\_loadu\_si128(...);

group\_state = \_mm\_xor\_si128(group\_state, block);

group\_state = gfmul(group\_state, H);

}

// 组间聚合

\_\_m128i multiplier = H\_powers[block\_count];

\_\_m128i state\_mult = gfmul(state, multiplier);

return \_mm\_xor\_si128(state\_mult, group\_state);

}

1. 高效GF乘法

\_\_m128i gfmul(\_\_m128i a, \_\_m128i b) {

// Karatsuba算法

tmp3 = \_mm\_clmulepi64\_si128(a, b, 0x00);

tmp4 = \_mm\_clmulepi64\_si128(a, b, 0x10);

// ...

// 模约简

tmp7 = \_mm\_clmulepi64\_si128(tmp3, \_mm\_set\_epi32(0, 0, 0, 0x87), 0x01);

// ...

return tmp3;

}