**SM4算法及SM4-GCM工作模式技术文档**

1. SM4算法概述

SM4是中国国家密码管理局发布的商用分组密码算法标准，属于对称加密算法。其主要特点：

分组长度：128位

密钥长度：128位

轮数：32轮

结构：非平衡Feistel网络

设计原则：安全性、高效性、易于实现

2. 数学表示与算法推导

2.1 基本结构

SM4采用32轮非平衡Feistel结构，每轮处理32位数据。加密过程可表示为：

设输入为(X₀, X₁, X₂, X₃) ∈ (GF(2)³²)⁴

For r = 0 to 31:

Xᵣ₊₄ = Xᵣ ⊕ T(Xᵣ₊₁ ⊕ Xᵣ₊₂ ⊕ Xᵣ₊₃ ⊕ RKᵣ)

输出为(X₃₅, X₃₄, X₃₃, X₃₂)

其中：

T(·)为合成置换函数

RKᵣ为轮密钥

2.2 轮函数 T(·)

T(·) = L(τ(·))，由非线性变换τ和线性变换L复合而成

2.2.1 非线性变换 τ

τ: GF(2)³² → GF(2)³²

将32位输入分为4个8位字节：

τ(A) = (Sbox(a₀), Sbox(a₁), Sbox(a₂), Sbox(a₃))

其中 A = a₀‖a₁‖a₂‖a₃，‖表示连接操作

S盒为8×8比特置换，提供非线性特性：

Sbox(x) = L(Affine(x⁻¹)) 在GF(2⁸)上

其中Affine变换为：

y = x ⊕ (x⋙2) ⊕ (x⋙3) ⊕ (x⋙4) ⊕ (x⋙5) ⊕ (x⋙6) ⊕ (x⋙7)

2.2.2 线性变换 L

L: GF(2)³² → GF(2)³²

L(B) = B ⊕ (B⋘2) ⊕ (B⋘10) ⊕ (B⋘18) ⊕ (B⋘24)

2.3 密钥扩展算法

密钥扩展生成32个轮密钥RKᵣ (0 ≤ r ≤ 31)：

设MK = (MK₀, MK₁, MK₂, MK₃) ∈ (GF(2)³²)⁴

Kᵢ = MKᵢ ⊕ FKᵢ, i=0,1,2,3

For r = 0 to 31:

RKᵣ = Kᵣ ⊕ T'(Kᵣ₊₁ ⊕ Kᵣ₊₂ ⊕ Kᵣ₊₃ ⊕ CKᵣ)

Kᵣ₊₄ = RKᵣ

其中：

FK为系统参数：FK₀=0xA3B1BAC6, FK₁=0x56AA3350, FK₂=0x677D9197, FK₃=0xB27022DC

CK为固定参数序列

T'(·)与T(·)类似，但线性变换不同：L'(B) = B ⊕ (B⋘13) ⊕ (B⋘23)

3. 优化实现技术

3.1 T-table优化

将非线性变换τ和线性变换L合并为查表操作：

预计算：

T₀[i] = L(Sbox(i))

T₁[i] = ROTL32(T₀[i], 24)

T₂[i] = ROTL32(T₀[i], 16)

T₃[i] = ROTL32(T₀[i], 8)

轮函数优化：

T(X) = T₀[X>>24] ⊕ T₁[(X>>16)&0xFF] ⊕ T₂[(X>>8)&0xFF] ⊕ T₃[X&0xFF]

优势：将每轮4次S盒查找和线性变换简化为4次查表和3次异或

3.2 AESNI指令优化

利用AES-NI指令加速S盒计算：

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

x = \_mm\_aesenc\_si128(x, \_mm\_setzero\_si128());

return x;

}

实现原理：

AES S盒与SM4 S盒有相似的代数结构，通过仿射变换转换：

SM4\_Sbox(x) = Affine(AES\_Sbox(x'))

3.3 GFNI+AVX512优化

使用GFNI指令实现S盒，AVX512实现并行处理：

\_\_m512i sm4\_gfni\_sbox(\_\_m512i x) {

const \_\_m512i gfni\_const = \_mm512\_set1\_epi8(0x8E);

return \_mm512\_gf2p8affine\_epi64\_epi8(x, gfni\_const, 0);

}

\_\_m512i sm4\_linear\_avx512(\_\_m512i x) {

\_\_m512i t2 = \_mm512\_rol\_epi32(x, 2);

\_\_m512i t10 = \_mm512\_rol\_epi32(x, 10);

\_\_m512i t18 = \_mm512\_rol\_epi32(x, 18);

\_\_m512i t24 = \_mm512\_rol\_epi32(x, 24);

return \_mm512\_xor\_si512(x, \_mm512\_xor\_si512(t2,

\_mm512\_xor\_si512(t10, \_mm512\_xor\_si512(t18, t24))));

}

优势：单指令处理16个S盒计算，64字节/周期吞吐量

4. SM4-GCM工作模式

4.1 GCM模式概述

Galois/Counter Mode (GCM) 提供认证加密功能：

加密：CTR模式

认证：GHASH函数

特点：并行计算、高效实现

工作流程：

生成初始计数器J0

加密：Cᵢ = Pᵢ ⊕ Eₖ(J0 + i)

认证：T = GHASH(AAD, C) ⊕ Eₖ(J0)

4.2 GHASH函数

GHASH在GF(2¹²⁸)上定义：

GHASH(H, A, C) =

(A₁·Hⁿ ⊕ A₂·Hⁿ⁻¹ ⊕ ... ⊕ Aₘ·H) ⊕

(C₁·Hᵐ ⊕ C₂·Hᵐ⁻¹ ⊕ ... ⊕ Cₙ·H) ⊕

(len(A)‖len(C))·H

其中：

H = Eₖ(0¹²⁸)

·表示GF(2¹²⁸)乘法

不可约多项式：x¹²⁸ + x⁷ + x² + x + 1

4.3 优化实现

4.3.1 PCLMULQDQ优化

使用PCLMULQDQ指令加速GF(2¹²⁸)乘法：

\_\_m128i ghash\_pclmulqdq(\_\_m128i a, \_\_m128i b) {

\_\_m128i t1 = \_mm\_clmulepi64\_si128(a, b, 0x00);

\_\_m128i t2 = \_mm\_clmulepi64\_si128(a, b, 0x11);

\_\_m128i t3 = \_mm\_clmulepi64\_si128(a, b, 0x01);

\_\_m128i t4 = \_mm\_clmulepi64\_si128(a, b, 0x10);

t3 = \_mm\_xor\_si128(t3, t4);

t4 = \_mm\_slli\_si128(t3, 8);

t3 = \_mm\_srli\_si128(t3, 8);

t1 = \_mm\_xor\_si128(t1, t4);

t2 = \_mm\_xor\_si128(t2, t3);

// 模约简

\_\_m128i r = \_mm\_set\_epi32(0, 0, 0, 0x87);

\_\_m128i m = \_mm\_clmulepi64\_si128(t1, r, 0x00);

t1 = \_mm\_srli\_si128(t1, 8);

t1 = \_mm\_xor\_si128(t1, m);

m = \_mm\_clmulepi64\_si128(t1, r, 0x10);

t1 = \_mm\_srli\_si128(t1, 8);

t2 = \_mm\_xor\_si128(t2, m);

return \_mm\_unpacklo\_epi64(t1, t2);

}

4.3.2 并行CTR加密

使用AVX512实现多块并行加密：

void sm4\_gcm\_enc\_avx512(uint8\_t \*out, const uint8\_t \*in, size\_t len,

\_\_m128i iv, const \_\_m512i \*rk) {

\_\_m128i ctr = iv;

for (size\_t i = 0; i < len; i += 64) {

\_\_m512i blocks[4];

// 加载4个CTR值

for (int j = 0; j < 4; j++) {

blocks[j] = \_mm512\_set1\_epi32(ctr);

ctr = \_mm\_add\_epi64(ctr, \_mm\_set\_epi32(0,0,0,1));

}

sm4\_avx512(blocks, rk); // 并行加密

// 与明文异或

\_mm512\_storeu\_si512(out + i,

\_mm512\_xor\_si512(blocks[0],

\_mm512\_loadu\_si512(in + i)));

}

}

5. 参考实现

// SM4基本加密函数

void sm4\_encrypt(const uint32\_t rk, const uint8\_t in, uint8\_t \*out) {

uint32\_t x[4];

for (int i = 0; i < 4; i++) {

x[i] = LOAD32U(in + i \* 4);

}

for (int i = 0; i < SM4\_NUM\_ROUNDS; i++) {

uint32\_t tmp = x[0] ^ sm4\_t(x[1] ^ x[2] ^ x[3] ^ rk[i]);

x[0] = x[1]; x[1] = x[2]; x[2] = x[3]; x[3] = tmp;

}

STORE32U(out, x[3]); STORE32U(out + 4, x[2]);

STORE32U(out + 8, x[1]); STORE32U(out + 12, x[0]);

}

// SM4-GCM加密

void sm4\_gcm\_encrypt(sm4\_gcm\_ctx \*ctx, const