Sm4 算法描述

SM4 是一种分组密码算法，其分组长度为 128 位（即 16 字节，4 字），密钥长度也为 128 位（即 16 字节，4 字）。其加解密过程采用了 32 轮迭代机制（与 DES、AES 类似），每一轮需要一个轮密钥。

轮密钥：

首先需要让原始密钥的每个字与系统参数 异或，得到 4 个新的字，这四个字即为 k0,k1,k2,k3

```
k[0] = key[0] ^ fk[0];
k[1]= key[1] ^ fk[1];
k[2]= key[2] ^ fk[2];
k[3]= key[3] ^ fk[3];
```

需要对这四个字进行 32 轮迭代，生成 32 个轮密钥。

```
unsigned int get_key(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int ck )
{
    unsigned int rk;
    unsigned int tmp = b ^ c ^ d ^ ck;
    uint8_t mid0[32] = { 0 }, mid1[32] = {0};
    uint8_t bytes[4];
    ui_to_bytes(tmp, bytes);
    uint8_t _bytes[4];
    _bytes[0] = sbox[bytes[0]];
    _bytes[1] = sbox[bytes[1]];
    _bytes[2] = sbox[bytes[2]];
    _bytes[3] = sbox[bytes[3]];
    uint8_t n[32] = { 0 }, _n[8] = { 0 }, __n[8] = { 0 }, ___n[8]{ 0 }, ____n[8] = { 0 };
    byte_to_bits(_bytes[0], _n);
    byte_to_bits(_bytes[1], __n);
    byte_to_bits(_bytes[2], ___n);
    byte_to_bits(_bytes[3], ____n);
    for (int i = 0; i < 8; i++) {
        n[i] = _n[i];
        n[8 + i] = __n[i];
        n[16 + i] = ___n[i];
        n[24 + i] = ____n[i];
    }

    left_move(n, 13,mid0);
    left_move(n, 23,mid1);
    unsigned int lre = bits_to_ui(n) ^ bits_to_ui(mid0) ^ bits_to_ui(mid1);
    rk = a ^ lre;
    return rk;
}
```

先对后三个输入进行异或，得到中间值 tmp，再将 tmp 按字节输入 s 盒中，进行非线性变换，得到 n，再分别对 n 循环左移 13，23，将得到的值与 n 异或，即可得到本轮的轮密钥 rk。

第一轮的轮密钥 rk0 的输入为（k0,k1,k2,k3）。rk0 即为 k4 再得到下一轮轮密钥 rk1 时（k1,k2,k3,k4）。rk1 即为 k5。这样以此类推，不断迭代，即可得到所有的轮密钥。

加密过程：

```
unsigned int sm4_F(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int rk)
{
    unsigned int result;
    unsigned int tmp = b ^ c ^ d ^ rk;
    uint8_t mid0[32], mid1[32],mid2[32],mid3[32];
    uint8_t bytes[4];
    ui_to_bytes(tmp, bytes);
    uint8_t _bytes[4];
    _bytes[0] = sbox[bytes[0]];
    _bytes[1] = sbox[bytes[1]];
    _bytes[2] = sbox[bytes[2]];
    _bytes[3] = sbox[bytes[3]];
    uint8_t n[32] = { 0 }, _n[8] = { 0 }, __n[8] = { 0 }, ___n[8]{ 0 }, ____n[8] = {0};
    byte_to_bits(_bytes[0], _n);
    byte_to_bits(_bytes[1], __n);
    byte_to_bits(_bytes[2], ___n);
    byte_to_bits(_bytes[3], ____n);
    for (int i = 0; i < 8; i++) {
        n[i] = _n[i];
        n[8 + i] = __n[i];
        n[16 + i] = ___n[i];
        n[24 + i] = ____n[i];
    }

    left_move(n, 2 ,mid0);
    left_move(n, 10,mid1);
    left_move(n, 18,mid2);
    left_move(n, 24,mid3);
    unsigned int lre = bits_to_ui(n) ^ bits_to_ui(mid0) ^ bits_to_ui(mid1) ^ bits_to_ui(mid2) ^ bits_to_ui(mid3);
    result = a ^ lre;
    return result;
}
```

与得到轮密钥的过程相似，加密过程也需要用到非线性变化 s 盒与线性变化循环左移。对输入的后三个数与轮密钥进行异或得到 tmp，再将 tmp 按字节输入 s 盒中，进行非线性变换，得到 n，再分别对 n 循环左移 2，10，18，24，将得到的值与 n 异或，即可得到本轮的输出。再将本轮的输出作为下一轮的输入之一，这样依次迭代，得到所有的输出。

```
*(_output+0) = x[35];
*(_output+1) = x[34];
*(_output+2) = x[33];
*(_output+3) = x[32];
```

在最后还要进行一次反序变换，将迭代最后得到的四个字，进行反序，得到最终的密文。

1.Sm4 的多线程加速

对于 sm4 有多种加密模式，其中 edc 模式可以进行并行处理，故可以对 edc 模式使用多线程进行加速。

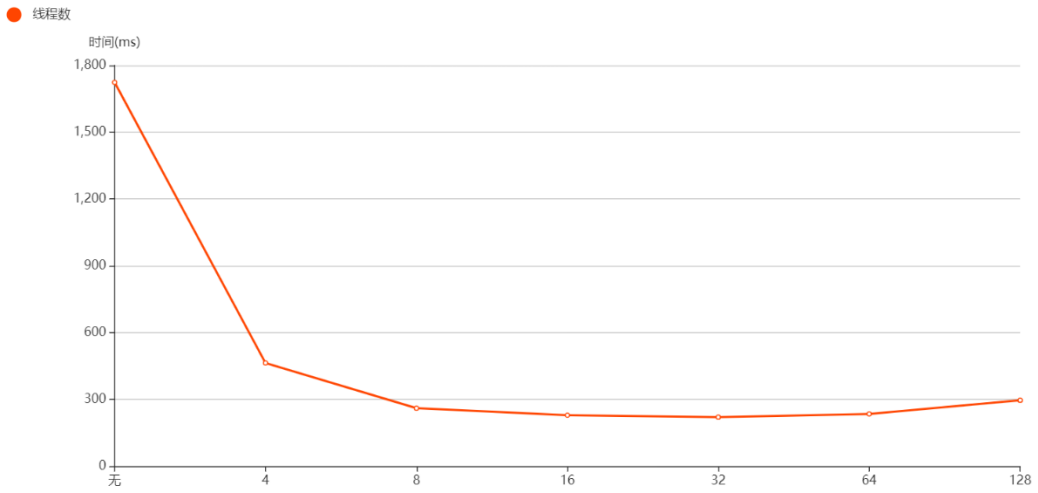
```
void sm4_enc_edc_thread(int thread_start,int thread_end)
{
    unsigned int key[4] = { 0x01234567,0x89abcdef, 0xfedcba98, 0x76543210 };
    for (int i = thread_start; i < thread_end; i = i + 4)
    {
        sm4_enc(input + i, key, output + i);
    }
}
```

```
//多线程
for (int i = 0; i < thread_num; i++)
{
    thread_start = i * offset;
    thread_end = i * offset + offset;
    th[i] = thread(sm4_enc_edc_thread,thread_start, thread_end);
}
for (int i = 0; i < thread_num; i++)
{
    th[i].join();
}
```

此时加密的规模是 2 的 14 次方。

线程数	无	4	8	16	32	64	128
时间(ms)	1723.51	463.348	260.36	228.87	220.091	234.578	295.909

图　　表　　：



从图表中可以看出当线程数小于 8 时，随着线程数的增长，算法的速度迅速提高，而当线程数大于 8 后，算法的速度增长缓慢，其中，线程数为 32 时，加密算法的速度达到最快，可以得出结论，使用多线程对 sm4 进行加密时，32 为最合适的线程数目。

2.sm4 的循环展开加速

在对 sm4 进行加速时，还可以考虑用流水线循环展开的方法进行。我尝试了二阶，三阶，四阶的循环展开，但时间均和不采用循环展开相近。