SM3的实现与优化

消息填充

SM3的消息扩展步骤是以512位的数据分组作为输入的。因此需要在一开始就把数据长度填充至512位的倍数。具体步骤为：

1、先填充一个“1”，后面加上k个“0”。其中k是满足(l+1+k) mod 512 = 448的最小正整数。

2、追加64位的数据长度（bit为单位，大端序存放。）

```
#填充
num = 64 - (len(data) + 1 & 0x3f)
data += b'\x80' + (len(data) << 3).to_bytes(num if num >= 8 else num + 64, 'big')
V = V0
B = array('L',data)
B.byteswap()
```

迭代压缩

将填充后的消息m'按512b进行分组：$m' = B^0 B^1 \dots B^{n-1}$,其中$n = (l + k + 65)/512$.对m'以如下方式迭代：

```
FOR i=0 TO(n-1)
    V(i+1)=CF(V(i),B(i));
ENDFOR
```

其中，CF是压缩函数，V(0)为256b初始值IV，B(i)为填充后的消息分组，迭代压缩的结果为V(n)。

```
#迭代压缩
for i in range(0, len(B), 16):
    V = CF(V, B[i:i+16])
V = array('L', V)
V.byteswap()
return V.tobytes()
```

压缩函数

令A,B,C,D,E,F,G,H为字寄存器，SS1,SS2,TT1,TT2为中间变量，压缩函数$V^{(i+1)} = CF(V^{(i)}, B^{(i)}), 0 \le i \le n-1$.计算过程描述如下：

```
ABCDEFGH←V(i)
FOR j=0 TO 63
    SS1←((A<<<12)+E+(T_j<<<(jmod32)))<<<7
    SS2←SS1⊕(A<<<12)
    TT1←FF_j(A,B,C)+D+SS2+W_j'
```

```
    TT2←GG_j(E,F,G)+H+SS1+W_j
    D←C
    C←B<<<9
    B←A
    A←TT1
    H←G
    G←F<<<19
    F←E
    E←P0(TT2)
ENDFOR
V(i+1)←ABCDEFGH⊕V(n)
```

杂凑值

$$ABCDEFGH \leftarrow V^{(n)}$$

```
#CF为压缩函数
def CF(V, B):
    #将消息分组B按以下方法扩展生成132个消息字W0,W1,...W63
    W = array('L', B)
    for j in range(16, 68):
        X = W[j-16] ^ W[j-9] ^ (W[j-3] << 15 | W[j-3] >> 17) & 0xffffffff
        W.append((X ^ (X << 15 | X >> 17) ^ (X << 23 | X >> 9) ^ (W[j-13] << 7 | W[j-13] >> 25) ^ W[j-6]) & 0xffffffff)
    W_ = array('L', (W[j] ^ W[j+4] for j in range(64)))
    #A-H为字寄存器
    A, B, C, D, E, F, G, H = V
    for j in range(64):
        A_r112 = A << 12 | A >> 20
        tmp = (A_r112 + E + Tj_r1[j]) & 0xffffffff
        SS1 = (tmp << 7 | tmp >> 25)
        SS2 = SS1 ^ A_r112
        if j & 0x30:
            FF, GG = A & B | A & C | B & C, E & F | ~E & G
        else:
            FF, GG = A ^ B ^ C, E ^ F ^ G
        TT1, TT2 = (FF + D + SS2 + W_[j]) & 0xffffffff, (GG + H + SS1 + W[j]) & 0xffffffff
        C = (B << 9 | B >> 23) & 0xffffffff
        D = C
        G = (F << 19 | F >> 13) & 0xffffffff
        H = G
        A = TT1
        B = A
        E = (TT2 ^ (TT2 << 9 | TT2 >> 23) ^ (TT2 << 17 | TT2 >> 15)) & 0xffffffff
        F = E
    return A ^ V[0], B ^ V[1], C ^ V[2], D ^ V[3], E ^ V[4], F ^ V[5], G ^ V[6], H ^ V[7]
```

优化：

1.用array数组作为中间值，可以减少bytes、list、int之间的类型转换从而达到减小运行时间的效果。

2.调整循环移位运算，pysmx为“一次取模、一次乘法、一次加法”，调整后为“两次移位、一次按位或”，由于位运算比乘除法快，所以运行时间可以减小。

3.由于代码中有大量循环，可以进行并行运算充分利用资源。

运行结果：

```
短消息长度：50B  长消息长度：1000B  测试次数：100
                        短消息Hash      长消息Hash
gmss1-SM3               37.9            663.2
pysmx-SM3               23.0            350.1
youhua-SM3              18.0            279.3
优化后总耗时为pysmx的79.7%、gmss1的42.4%
>>>
```

参考：https://blog.csdn.net/qq_43339242/article/details/123709822