SM3散列方式

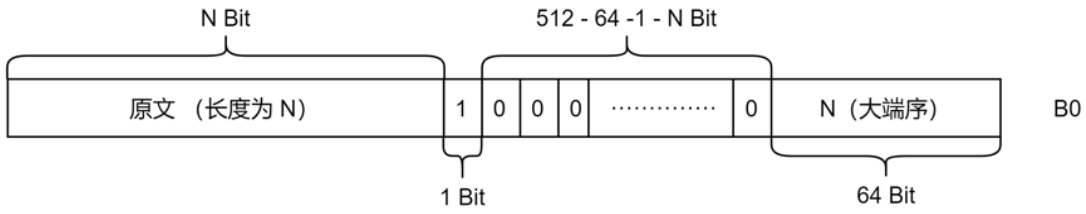
第一步

将需要散列的 16进制 字符串补充到 长度mod512 = 0 ， 参考代码： Sm3_Data 结构 的 构造函数

补充方式

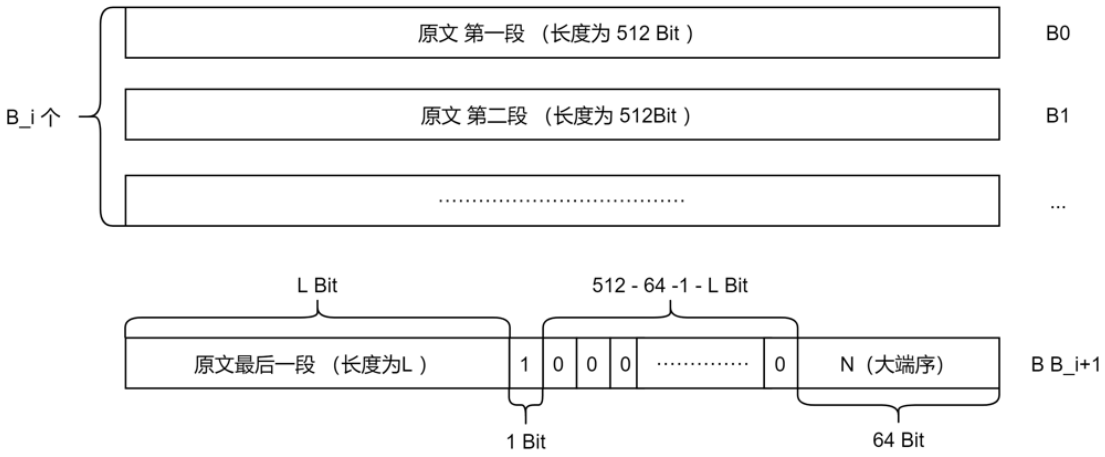
设 需要散列的 文本为"M"，M的长度为N Bit , B_i = N/512 , L = N%512。

N <= 512-64-1 Bit：

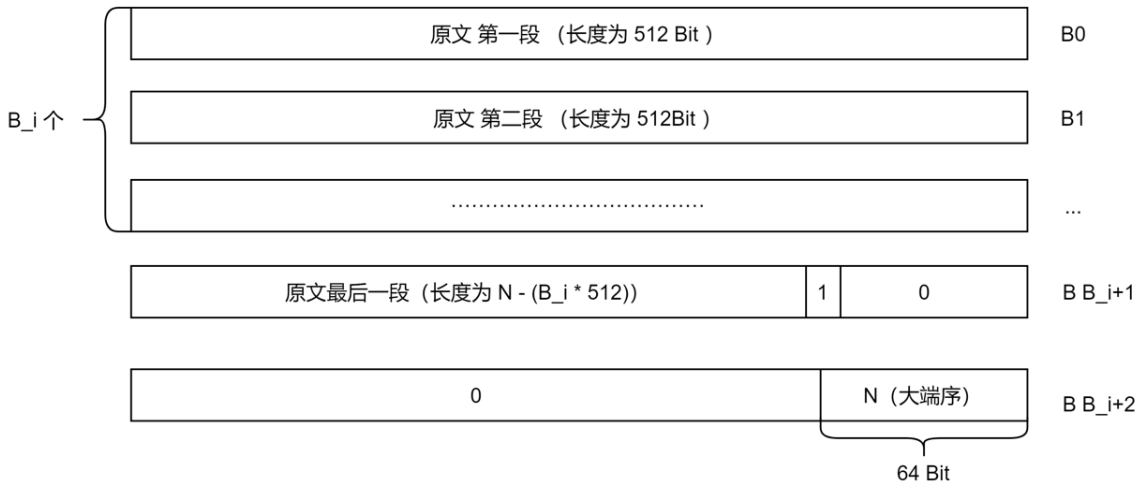


N > i * 512 Bit (i∈N* && i > 1):

512-L>64+1:



512-L<64+1:



第二步

将经过第一步的 M 叫为 Mc，将Mc分为 i 个 512 Bit 的B（上图中的 B0,B1,....Bi）

对每一个B 计算 对应的W 与 W'，参考代码：`BOOLEAN Mc_To_WWc(PSm3_Data Data);`

（W0 - W15需要以四字节为单位转换端序）

将消息分组$B^{(i)}$按以下方法扩展生成132个字$W_0, W_1, \cdots, W_{67}, W'_0, W'_1, \cdots, W'_{63}$，用于压缩函数$CF$：

a)将消息分组$B^{(i)}$划分为16个字$W_0, W_1, \cdots, W_{15}$。

b)FOR j=16 TO 67

$$W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$$

ENDFOR

c)FOR j=0 TO 63

$$W'_j = W_j \oplus W_{j+4}$$

ENDFOR

我的代码中，对于一个消息 Mc，对应的W与W'的样式如下：

	W0	W1	W2	W3	.....	W67
B0	nW[0] [0]					
B1		nW[1] [1]				
B2			nW[2] [2]			
...						
Bi-1	nW[i-1] [0]	nW[i-1] [1]	nW[i-1] [2]	nW[i-1] [3]		nW[i-1] [67]

	W'0	W'1	W'2	W'3	.....	W'63
B0	nWc[0] [0]					
B1		nWc[1] [1]				
B2			nWc[2] [2]			
...						

	W'0	W'1	W'2	W'3	.....	W'63
Bi-1	nWc[i-1] [0]	nWc[i-1] [1]	nWc[i-1] [2]	nWc[i-1] [3]		nWc[i-1] [63]

表格中为空白的地方，即为函数Mc_To_WWc（上图算法）计算的得到的。

第三步

对每一个B(B0、B1、....Bi-1)进行计算得到V i+1 ,其中V0为初始向量：参考代码：函数 `BOOLEAN Sm3_Hash(PSm3_Data Data);`

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$V^{i+1} = CF(V^{(i)}, B^{(i)}), 0 \leq i \leq n-1$。计算过程描述如下：

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

FOR j=0 TO 63

$SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$

$SS2 \leftarrow SS1 \oplus (A \lll 12)$

$TT1 \leftarrow FF_j(A,B,C) + D + SS2 + W'_j$

$TT2 \leftarrow GG_j(E,F,G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B \lll 9$

$B \leftarrow A$

$A \leftarrow TT1$

$H \leftarrow G$

$G \leftarrow F \lll 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$

其中，字的存储为大端(big-endian)格式。

例如：

B0对应的67个W 和 B0对应的63个W' 和 V0 三者通过上述算法可以得到 V1。

B1对应的67个W 和 B1对应的63个W' 和 V1 三者通过上述算法可以得到 V2。

.....

Bi-1对应的67个W 和 Bi对应的63个W' 和 Vi 三者通过上述算法可以得到 Vi。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
V[0]	0x7380166f	0x4914b2b9	0x172442d7	0xda8a0600	0xa96f30bc	0x163138aa	0xe38dee4d	0xb0fb0e4e
V[1]		V[1] [1]						
...								
V[i-1]							V[i-1] [6]	
V[i]	V[i] [0]	V[i] [1]	V[i] [2]	V[i] [3]	V[i] [4]	V[i] [5]	V[i] [6]	V[i] [7]

V0为初始化值，如表格所示。

Vi即为最终Hash 结果

对上述算法细节补充：

```
ULONG32 SL(ULONG32 X, int n)//左循环
{
    unsigned __int64 x = X;
    x = x << (n % 32);
    unsigned long l = (unsigned long)(x >> 32);
    return x | l;
}


ULONG32 Tj(int j) {
    if (j >= 0 && j <= 15)
        return 0x79cc4519;
    else
        return 0x7a879d8a;
}

ULONG32 FFj(int j, ULONG32 X, ULONG32 Y, ULONG32 Z) {
    if (j >= 0 && j <= 15)
        return X ^ Y ^ Z;
    else
        return ((X & Y) | (X & Z) | (Y & Z));
}

ULONG32 GGj(int j, ULONG32 X, ULONG32 Y, ULONG32 Z) {
    if (j >= 0 && j <= 15)
        return X ^ Y ^ Z;
    else
        return ((X & Y) | (~X & Z));

}

ULONG32 P0(ULONG32 X) {
    return X ^ SL(X, 9) ^ SL(X, 17);
}

ULONG32 P1(ULONG32 X) {
    return X ^ SL(X, 15) ^ SL(X, 23);
}
```