密码学 zk 系列

第 11 课：zkSnark-SHA256 查找表

lynndell 博士

新火科技 密码学专家 lynndell2010@gmail.com

目录

1.SHA256

1.1. 常量与基本运算

基础工具包括：8+64 个初始常量、信息预处理（数据填充）、逻辑运算

轮密钥加，需要添加一些常量。

1.1.1 常量

8 个初值常量如下：（256bit = 8*8*4）

h0 := 0x**6a0**9e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

来源：对自然数中前 8 个质数（2,3,5,7,11,13,17,19）的平方根的小数部分取前 32bit。

例如：$ \sqrt{2} $小数部分约为 0.41421356237309048，而

$$0.41421356237309048 \approx \mathbf{6}*16^{-1}+\mathbf{a}*16^{-2}+\mathbf{0}*16^{-3}+...$$

所以，质数 2 的平方根的小数部分取前 32bit 就对应出了 0x**6a0**9e667。**每个参数都有来源根据，没有后门。**

64 个常量如下：

428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2

来源：这些常量是对自然数中前 64 个质数(2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97…)的立方根的小数部分取前 32bit 而来。

1.1.2 基本运算

6 个逻辑运算：

- 前 4 个计算用于 64 轮循环（**用于数据非线性压缩**）
- 后 2 个计算用于 16 个字扩展为 64 个字（**用于数据非线性扩展**）

1 个字=32 比特

$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$

$\Sigma 0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$

$\Sigma 1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$

$\sigma 0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$

$\sigma 1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$

$\wedge$ 按位“与”

$\neg$ 按位“补”

$\oplus$ 按位“异或”

S^n 循环右移 n 个 bit

R^n 右移 n 个 bit

1.2 数据预处理(pre-processing)

数据填充使整个消息长度和结构满足规定。

信息的预处理分为 2 步：附加填充比特和附加长度。

1.2.1 STEP1：填充比特

填充规则：先补第一个比特为 1，然后都补 0，直到长度满足对 512 取模后余数是 448。剩余 64bit 填充数据长度。SHA256 压缩数据长度 2^64

需要注意的是，信息必须进行填充，也就是说，即使长度已经满足对 512 取模后余数是 448，补位也必须要进行，这时要填充 512 个比特。

因此，填充是至少补一位，最多补 512 位。

例：以信息“abc”为例显示补位的过程。

ASCII 码分别是 97,98,99

二进制编码为：01100001，01100010，01100011

补位第一步，首先补一个”1”： 0110000101100010 01100011 1

补位第二步,补 423 个“0”：01100001 01100010 01100011 10000000 00000000 … 00000000

补位完成后的数据如下（为了简介用 **16 进制**表示）：

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000
在第一步的预处理后，**第二步会再附加上一个 64bit 的数据，用来表示原始报文的长度信息。**而 448+64=512，正好拼成了一个完整的结构。
综上：
01100001 01100010 01100011 10000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

1.2.2 STEP2：附加长度值

448 + 64 = 512

用一个 64 位的数据来表示原始消息的长度。
因此，通过 SHA256 计算的消息长度必须要小于 2^64，当然绝大多数情况这足够大了。
长度信息的编码方式为 64-bit big-endian integer
关于 Big endian 的含义，文末给出了补充
回到刚刚的例子，消息“abc”，3 个字符，占用 24 个 bit
因此，在进行了补长度的操作以后，整个消息就变成下面这样了（十六进制 18 = 十进制 24）
61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018
最终的二进制编码为：
01100001 01100010 01100011 10000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011000

1.3. 计算摘要

SHA256 算法的主体部分，即消息摘要是如何计算的。
首先：将消息分解成 512-bit 大小的块，不足 512 则根据上一节方法填充。

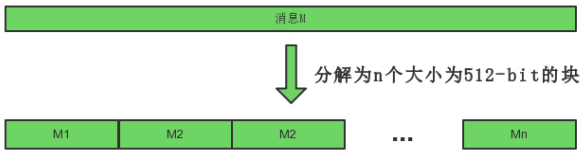


图 1：消息分为 n 个块，|M| = 512 * n

消息 M 可以被分解为 n 个块，整个算法需要完成 n 次迭代，n 次迭代的结果就是最终的哈希值，即 256bit 的数字摘要。

一个 256-bit 的摘要的**初始值 H0**，经过第一个数据块进行运算，得到 H1，即完成了第一次迭代 H1 经过第二个数据块得到 H2，……，依次处理，最后得到 Hn，Hn 即为最终的 256-bit 消息摘要将每次迭代进行的映射

$$H_i := Map(H_{i-1}, M_i)$$

于是迭代可以更形象的展示为：

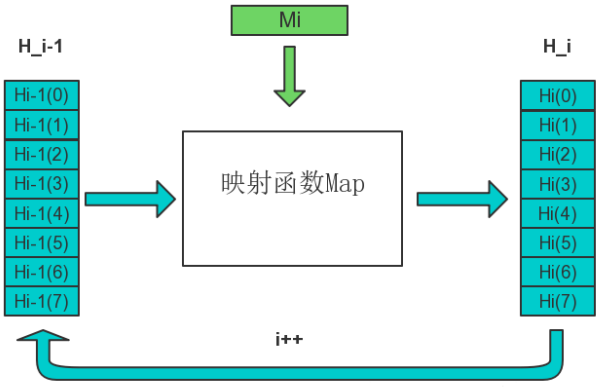


图 2：迭代过程

图中 256-bit 的 Hi 被描述 8 个小块，这是因为 **SHA256 算法中的最小运算单元称为“字”（Word），一个字是 32 位。**

第一次迭代中，映射的初值设置为前面介绍的 8 个哈希初值，如下图所示：

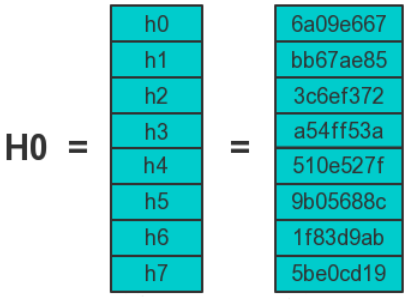


图 3：哈希初始值

下面开始介绍每一次迭代的内容，即映射 Map(H_{i-1}) = H_i 的具体算法

Map 映射函数

也就是对称加密中的轮函数 function

STEP1：扩展函数：输入：512bits，输出 2048bits；16 个字扩展为 64 个字，每个字 32bit

对于每个块 **M**(512bits=16*32)，分解为 16 个 32-bit 的 big-endian 的字 w[0], …, w[15];

1. **起始状态：前 16 个字**直接由消息的第 1 个块分解得到：512bit= **w[0]，…，w[15]**

2. **其余的 48 个字**由如下**迭代公式**得到：$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$

$W_{16}=\sigma_1(W_{14})+W_{11}+\sigma_0(W_1)+W_0$

$W_{17}=\sigma_1(W_{15})+W_{12}+\sigma_0(W_2)+W_1$

,…,

W_{64}

其中，

$\sigma 0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$

$\sigma 1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$

$\wedge$ 按位“与”

$\neg$ 按位“补”

$\oplus$ **按位“异或”**

S^n **循环**右移 n 个 bit

R^n 右移 n 个 bit

STEP2：压缩函数：进行 64 次循环；输入 64 个字和 64 个常量 k_i

映射函数 $H_i := Map(H_{i-1}, M_i)$ 包含 64 次循环

即进行 64 次循环即可完成一次迭代

每次加密循环可以由下图描述：

256 = 8 * 32

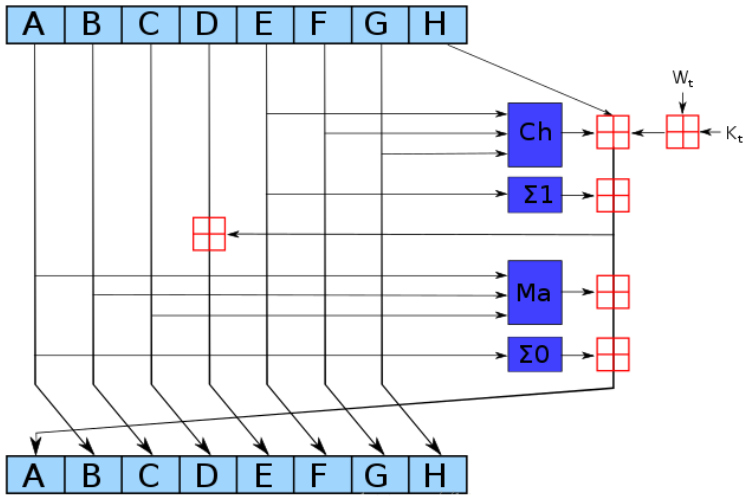


图 4：64 次循环

- $Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$
- $Ma(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- $\Sigma 0(x) = S^2(x) \oplus S^{13}(x) \oplus S^{22}(x)$
- $\Sigma 1(x) = S^6(x) \oplus S^{11}(x) \oplus S^{25}(x)$

图中，ABCDEFGH 这 8 个字（word）在按照一定的规则进行更新，其中**深蓝色方块**是事先定义好的非线性逻辑函数；
红色田字方块代是：相加后 $mod 2^{32}$，其中一个红色方框是**字与常量的模加**

$$W_t + K_t \ (mod 2^{32})$$

其中，是 K_t 是 64 个常量。

ABCDEFGH 一开始的 8 个**初始值**分别为 H_{i-1}(0),H_{i-1}(1),…,H_{i-1}(7)。
Kt 是 64 个常量，每次循环使用 1 个常量。
Wt 是本区块产生第 t 个 word。原消息被切成固定长度 512-bit 的区块，对每一个区块，产生 64 个 word，通过重复运行循环 n 次对 ABCDEFGH 这八个字循环加密。

Add the compressed chunk to the current hash value:

```
h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
h4 := h4 + e
h5 := h5 + f
h6 := h6 + g
h7 := h7 + h
```

最后一次循环所产生的八个字合起来即是第 i 个块对应到的哈希值 H_i。

hash:= h0 || h1 || h2 || h3 || h4 || h5 || h6 || h7

```
for each chunk
    create a 64-entry message schedule array w[0..63] of 32-bit words
    (The initial values in w[0..63] don't matter, so many implementations zero them here)
    copy chunk into first 16 words w[0..15] of the message schedule array
    Extend the first 16 words into the remaining 48 words w[16..63] of the message schedule array:
    for i from 16 to 63
        s0 := (w[i-15] rightrotate  7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift  3)
        s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rightshift 10)
        w[i] := w[i-16] + s0 + w[i-7] + s1
    Initialize working variables to current hash value:
    a := h0
    b := h1
    c := h2
    d := h3
```

e := h4

f := h5

g := h6

h := h7

Compression function main LOOP:

for i from 0 to 63

S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)

ch := (e and f) xor ((not e) and g)

temp1 := h + S1 + ch + k[i] + w[i]

S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)

maj := (a and b) xor (a and c) xor (b and c)

temp2 := S0 + maj

h := g

g := f

f := e

e := d + temp1

d := c

c := b

b := a

a := temp1 + temp2

Add the compressed chunk to the current hash value:

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

h4 := h4 + e

h5 := h5 + f

h6 := h6 + g

h7 := h7 + h

Produce the final hash value (big-endian):

digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7

2 SHA256 查找表技术

2.1 电路约束

2.1.1 基本约束

1. **布尔约束：** $a\cdot(1-a)=0$，将 a 约束为**布尔值**；

2. **范围约束：** $b\cdot(1-b)\cdot(2-b)\cdot(3-b)\cdot(4-b)\cdot(5-b)=0$，将 b 约束到范围 $[0,5]$；

3. **常量约束：** $(7-c)\cdot(13-c)=0$，将 c 约束为常量 7 或 13；

4. **变量约束：** $(x-d)\cdot(y-d)\cdot(z-d)=0$，将 d 约束为三个变量 x,y,z。

2.1.2 集合约束

Spread 函数：输入 16bit 的 X，在左边逐位插入零，输出 32bit 的 X'。

举例：

输入 16bitX：1111111111111111

输出 32bitY：01010101010101010101010101010101

Spread 映射的查找表约束

	输入	输出
第 0 行	0000000000000000	00000000000000000000000000000000
…	…	…
第 $2^{16}-1$ 行	1111111111111111	01010101010101010101010101010101

- **对于输入为 x，输出为 x'，证明 $(x,x')\in Table$，则确保 $x'=spread(x)$ 映射正确。**

布尔**异或**运算表 $x\oplus y=z$

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

利用 spread 函数 $x'=spread(x),y'=spread(y)$

Spread 函数表

x'	y'	$z'\&1$
00	00	00
00	01	01
01	00	01
01	01	10

布尔运算： $x\oplus y=z$
等价于以下运算： **步骤 1：spread 映射：** $x'=spread(x),y'=spread(y)$； **步骤 2：算术运算：** $x'+y'=z',z'\&1=z$。
对应的电路约束：

步骤 1：spread 查表约束：$(x,x'),(y,y')\in Table(u1,u2)$；

步骤 2：算术电路约束：$x'+y'=z',z'\&1=z$。

16bit 的异或运算 $u16(x)\oplus u16(y)=z$

等价于以下运算：

步骤 1：spread 映射：$x'=spread(x),y'=spread(y)$；

步骤 2：算术运算：$z=x'+y',z=z'_{even}+z'_{odd}*2$，计算结果**取偶数位** z_{even}。

对应的电路约束：

步骤 1：spread 查表约束：$(x,x'),(y,y')\in Table(u16,u32)$；

步骤 2：算术电路约束：$z=x'+y',z=z'_{even}+z'_{odd}*2$，计算结果**取偶数位** z_{even}。

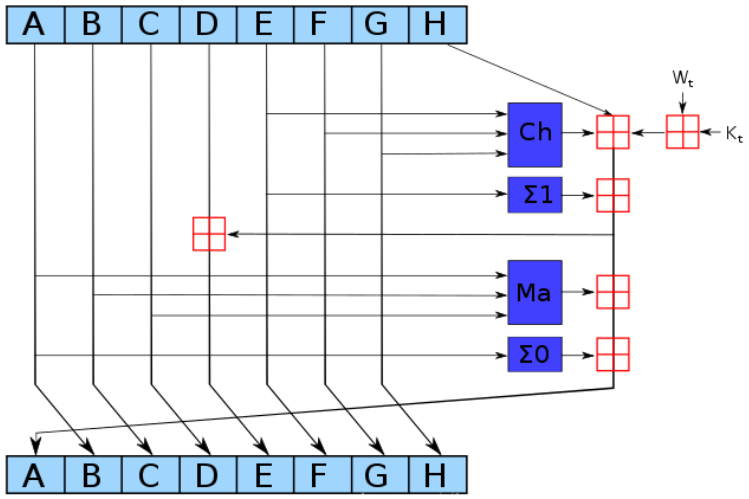
核心思想：Plonk 是算术电路，sha256 是布尔电路。

布尔运算等价于 spread Table+代数运算；

对布尔运算的约束等价于 spread 查表约束+算术电路约束，则节约电路。

2.2 SHA256 的 16bits 查表技术

以下是哈希函数 SHA256 的 64 轮函数，位宽为 32bits 的 A,B,C,D,E,F,G



$$
\begin{aligned}
Ch(E,F,G) &= (E \wedge F) \oplus (\neg E \wedge G) \\
Maj(A,B,C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\
&= count(A,B,C) \geq 2 \\
\Sigma_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22) \\
\Sigma_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25) \\
H' &= H + Ch(E,F,G) + \Sigma_1(E) + K_t + W_t \\
E_{new} &= reduce_6(H' + D) \\
A_{new} &= reduce_7(H' + Maj(A,B,C) + \Sigma_0(A))
\end{aligned}
$$

图 2. SHA2 循环结构运算原理

其中，$reduce_i$ 必须记录进位 $carry$，$carry \in [0,i)$。

2.2.1 模加法

图中红色的⊞是指相加后模 2^{32}。

关键结论：模 2^{32} 加法等价于在域上进行加法，结果保留低 32bits。

模加法运算： $a \boxplus b = c$
等价于以下运算： **步骤 1：spread 映射：**将 3 个操作数均分解为位宽为 16bits 的 2 块（chunk），共 6 个 chunk，16bit 的 spread 映射确 $a_L, a_H, b_L, b_H, c_L, c_H$ 在 Table 内 $$(a_L, a_H) \boxplus (b_L, b_H) = (c_L, c_H)$$ **步骤 2：算术运算：**使用域加法 $$carry \cdot 2^{32} + c_H \cdot 2^{16} + c_L = (a_H + b_H) \cdot 2^{16} + a_L + b_L$$
对应的电路约束： **步骤 1：**spread 查表约束：$a_L, a_H, b_L, b_H, c_L, c_H \in Table(u16)$ **步骤 2：算术电路约束。**

注意：

1. 输出可以分解为任意位宽 n-bit，而不仅仅是位宽为 16bit；
2. **该约束要求每个块有范围约束，否则赋值将会溢出域的范围；**
3. 操作数 a,b,c 的 6 个 chunk，使用 1 个 spread 表进行**等价约束**。因此，直接获得计算结果 c 的 spread 函数值。尤其是对 A_{new} 和 E_{new} 的 spread 函数值，能够用于优化 Maj 函数和 Ch 函数；
4. carry 进行**范围约束**。

2.2.2 Maj 函数

使用 1bit 举例，16bit 类似结论

A	B	C	Maj(A,B,C)	A'+B'+C'
0	0	0	0	00
0	0	1	0	01
0	1	0	0	01
0	1	1	1	10
1	0	0	0	01
1	0	1	1	10
1	1	0	1	10
1	1	1	1	11

Maj 运算： $Maj(A,B,C)=(A\wedge B)\oplus(A\wedge C)\oplus(B\wedge C)$

等价于以下运算：

步骤 1：spread 映射：初始状态（已有约束）： 在第一轮获得 A 的 spread 函数值 A'；B 和 C 分别等于上一轮的 A 和 B，因此也获得了 spread 函数值 B',C'。**第一轮要么是初始向量 IV，**要么是上一轮的 spread 函数值。因此，此处不需要 spread 查找表约束。

步骤 2：算术运算： 对 spread 值在域范围内进行加法运算：

$$M'=A'+B'+C'$$

步骤 3：spread 映射：计算结果： M' 是 32bit 拆为 2 个 16bit。低 16bit 偶数位 M_0^{even} 与高 16bit M_1^{even} 偶数位，低 16bit 奇数 M_0^{odd} 与高 16bit 奇数 M_1^{odd}。

Maj 函数进行以下方式展开

$$M'=spread_1\left(M_0^{even}\right)+2\cdot spread_2\left(M_0^{odd}\right)+2^{32}\cdot spread_3\left(M_1^{even}\right)+2^{33}\cdot spread_4\left(M_1^{odd}\right)$$

输出奇数位 M_0^{odd},M_1^{odd} 就是最终的计算结果。

对应的电路约束：

步骤 1：spread 查表约束：无

步骤 2：算术运算电路约束。

步骤 3：spread 查表约束：需要 2chunks * 2spread，即 4 个查找表实现。

2 个 chunk：Maj 函数值的位宽为 32bits，需使用 2 个 16-bit 的 chunk 表达。

2 个 spread 函数：每个 chunk 中的奇数位和偶数位分别使用一个 spread 查找表约束。

2.2.3 Ch 函数

Ch 运算： $Ch(E,F,G)=(E\wedge F)\oplus(\neg E\wedge G)$

等价于以下运算：

步骤 1：spread 映射：初始状态：（已有约束）： 在第一轮获得 E 的 spread 函数值 E'；F 和 G 等于上一轮的 E 和 F，因此也获得了 spread 函数值 F',G'。第一轮要么是初始向量 IV，要么是上一轮的 spread 函数值。

步骤 2：算术运算： 对 spread 函数值在域范围内进行加法运算：$P'=E'+F',\ Q'=evens-E'+G'$。其中，$evens=spread(2^{32}-1)$，$evens-E'$没有借位。

步骤 3：spread 映射：计算结果： $P_0^{even},P_0^{odd},Q_0^{even},Q_0^{odd},P_1^{even},P_1^{odd},Q_1^{even},Q_1^{odd}$ 表示计算结果的奇数 bits 和偶数 bits。Ch 函数进行以下方式展开

$$P'=spread_1\left(P_0^{even}\right)+2\cdot spread_2\left(P_0^{odd}\right)+2^{32}\cdot spread_3\left(P_1^{even}\right)+2^{33}\cdot spread_4\left(P_1^{odd}\right)$$

$$Q'=spread_5\left(Q_0^{even}\right)+2\cdot spread_6\left(Q_0^{odd}\right)+2^{32}\cdot spread_7\left(Q_1^{even}\right)+2^{33}\cdot spread_8\left(Q_1^{odd}\right)$$

输出**奇数位** $P_0^{odd}+Q_0^{odd},P_1^{odd}+Q_1^{odd}$ 就是**最终的计算结果**

对应的电路约束：

步骤 1：spread 查表约束：无

步骤 2：算术运算电路约束。

步骤 3：spread 查表约束：需要 2chunks * 4spread，即 8 个查找表实现。

2 个 chunk：Ch 函数是位宽为 32bits，**需使用 2 个 16-bit 的 chunk 进行表达。**

4 个 spread 函数：Ch 有两个计算结果，分别是 P',Q'，**需要 2 个 chunk 表达。每个计算结果的奇数位和偶数为分别使用一个 spread 函数。共 4 个 spread 函数。**

2.2.4Sigma0 函数

Sigma0 运算： $\Sigma_0(A)=(A\ggg 2)\oplus(A\ggg 13)\oplus(A\ggg 22)$

等价于以下运算：

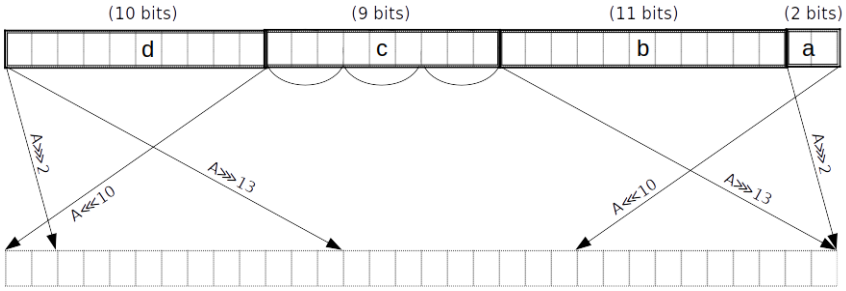
步骤 1：spread 映射：初始状态：（已有约束）：Sigma0 函数的输入是 A。将 A 分解为 (a,b,c,d) 四片，长度分别为 $(2,11,9,10)$，小端表达。（1）b,d 的长度为 10 和 11 的片使用 **2 个 spread 查找表**进行约束。（2）长度为 9 的片能够分解为 3*3 的子片、长度为 2 的片，这两片使用**范围约束**进行约束，而不需要使用 spread 函数约束，即这些小片的 spread 函数可通过拉格朗日插值多项式实现约束。因此，优化后能够使用 2 行进行长度约束。

对 4 片数据分解后进行组合，能够对 A_{new} 进行还原，而不需要额外的查找表。在最后一轮，加上进位后（进位最大为 7），则能够还原 A_{new}。

$$\Sigma_0(A)=(A\ggg 2)\oplus(A\ggg 13)\oplus(A\ggg 22)$$

等价为

$$\Sigma_0(A)=(A\ggg 2)\oplus(A\ggg 13)\oplus(A\lll 10)$$



数据初始状态 d10,c9,b11,a2

循环右移 2bit： a2,d10,c9,b11

循环右移 13bit：b11,a2,d10,c9,

循环左移 10bit：c9,b11,a2,d10,

步骤 2：算术运算：Sigma0 函数中的**循环移位、异或运算等价为**以下**位拼接的异或运算**

$$\Sigma_0(A)=\begin{pmatrix}(a\|d\|c\|d)\oplus\\(b\|a\|d\|c)\oplus\\(c\|b\|a\|d)\end{pmatrix}$$

再等价为以下域范围内的加法算术运算

$$R'=\begin{pmatrix}4^{30}a+4^{20}d+4^{11}c+b+\\4^{21}b+4^{19}a+4^{9}d+c+\\4^{23}c+4^{12}b+4^{10}a+d\end{pmatrix}$$

步骤 3：spread 映射：计算结果：Sigma0 函数使用 4 个 spread 查表

$$R'=spread_1\left(R_0^{even}\right)+2\cdot spread_2\left(R_0^{odd}\right)+2^{32}\cdot spread_3\left(R_1^{even}\right)+2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

最终结果取**偶数位** R_1^{even},R_2^{even}。

对应的电路约束：

步骤 1：spread 查表约束：2 个

步骤 2：算术电路约束。

步骤 3：spread 查表约束：4 个

总共需要 6 个 spread 函数查找表、算术电路约束、一些范围约束。

2.2.5Sigma1 函数

Sigma1 运算： $\Sigma_1(E)=(E\ggg 6)\oplus(E\ggg 11)\oplus(E\ggg 25)$

等价于以下运算：

步骤 1：spread 映射：初始状态：（已有约束）：Sigma1 函数的输入是 E，首先将 E 分解为 (a,b,c,d) 四片，长度分别为 $(6,5,14,7)$，小端表达。（1）c,d 的长度为 7 和 14 的片使用 2 个 spread **查找表**实现约束。（2）长度为 5 的片分解为长度 3 和长度为 2 的子片、长度为 6 的片可以分解为 2*3bits 子片；这 4 个小片使用**范围约束**进行约束，而不

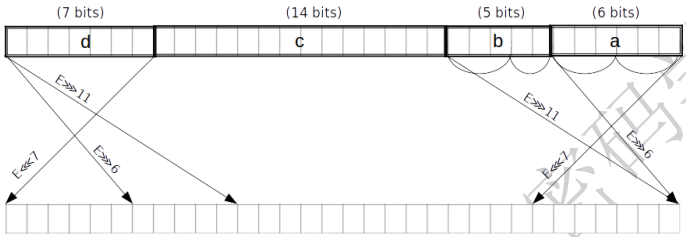
需要使用 spread 函数进行约束，即这些小片的 spread 函数可通过拉格朗日插值多项式实现约束。因此，优化后能够使用 2 个 plonk 行进行长度约束。

对 4 片数据分解后进行组合，能够对 E_{new} 进行还原，而不需要额外的查找表技术。在最后一轮，加上进位后（进位最大为 7），则能够还原 E_{new}。

$$\Sigma_1(E)=(E \ggg 6)\oplus(E \ggg 11)\oplus(E \ggg 25)$$

等价为

$$\Sigma_1(E)=(E \ggg 6)\oplus(E \ggg 11)\oplus(E \lll 7)$$



步骤 2：算术运算：Sigma1 函数中的循环移位、异或运算**等价为**以下位拼接的异或运算

$$\Sigma_0(E)=\begin{pmatrix}(a\|d\|c\|d)\oplus\\(b\|a\|d\|c)\oplus\\(c\|b\|a\|d)\end{pmatrix}$$

再等价为如下**域范围内的加法运算**

$$R'=\begin{pmatrix}4^{26}a+4^{19}d+4^5c+b+\\4^{27}b+4^{21}a+4^{14}d+c+\\4^{18}c+4^{13}b+4^7a+d\end{pmatrix}$$

步骤 3：spread 映射：计算结果：Sigma1 函数使用 4 个 spread 查找表

$$R'=spread_1\left(R_0^{even}\right)+2\cdot spread_2\left(R_0^{odd}\right)+2^{32}\cdot spread_3\left(R_1^{even}\right)+2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

最终结果取**偶数位** R_1^{even},R_2^{even} **。**

对应的电路约束：
步骤 1：spread 查表约束：
步骤 2：算术电路约束。
步骤 3：spread 查表约束：4 个
总共需要 4 个 spread 函数查找表、算术电路约束、一些范围约束。

2.3 Block 分解

512bit 的数据 M，每 32bit 记为一个 W， 32*16=512，则 512bit 记为 16 个 W。
SHA256 的扩展函数将数据 M（$16W$）扩展为 64 个 W。
（1）前 16 个 W **等于**数据 M

$$M = W_0 \| W_1 \| ... \| W_{15}$$

（2）剩余 48 个W 使用以下公式计算得出

$$W_i = \sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16}$$

其中

$$\sigma_0(X) = (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3)$$
$$\sigma_1(X) = (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10)$$

2.3.1 sigma0 函数

sigma0 运算：

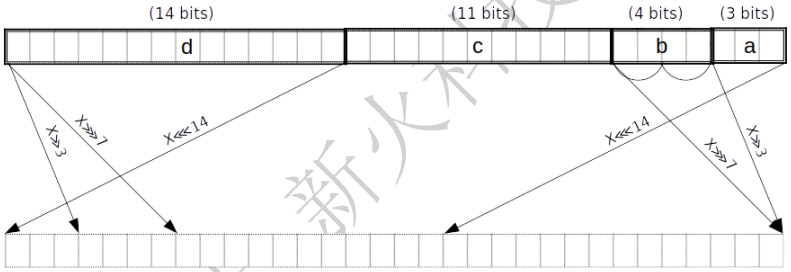
$$\sigma_0(X) = (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3)$$

等价为

$$\sigma_0(X) = (X \ggg 7) \oplus (X \lll 14) \oplus (X \gg 3)$$

等价于以下运算：

步骤 1：spread 映射：初始状态：（已有约束）：



sigma0 函数的输入是 X，首先将 X 分解为 (a,b,c,d) 四片，长度分别为 $(3,4,11,14)$，小端表达。将 b 拆为 2 个 2-bit 的片。

步骤 2：算术运算：sigma0 函数中的**循环移位、异或运算等价为**以下基于位拼接的**异或**运算

$$\sigma_0(X) = \begin{pmatrix} (0^{[3]} \| d \| c \| d) \oplus \\ (b \| a \| d \| c) \oplus \\ (c \| b \| a \| d) \end{pmatrix}$$

再等价为如下域范围内的加法运算

$$R' = \begin{pmatrix} 0 + 4^{15} d + 4^4 c + b + \\ 4^{28} b + 4^{25} a + 4^{11} d + c + \\ 4^{21} c + 4^{17} b + 4^{14} a + d \end{pmatrix}$$

步骤 3：spread 映射：计算结果：sigma0 函数使用 4 个 spread 查找表

$$R' = spread_1\left(R_0^{even}\right) + 2 \cdot spread_2\left(R_0^{odd}\right) + 2^{32} \cdot spread_3\left(R_1^{even}\right) + 2^{33} \cdot spread_4\left(R_1^{odd}\right)$$

最终结果取**偶数位** R_1^{even}, R_2^{even} 。

对应的电路约束：

步骤 1：spread 查表约束：

步骤 2：算术电路约束。

步骤 3：spread 查表约束：4 个

总共需要 4 个 spread 函数查找表、算术电路约束、一些范围约束。

2.3.1 sigma1 函数

sigma1 运算：

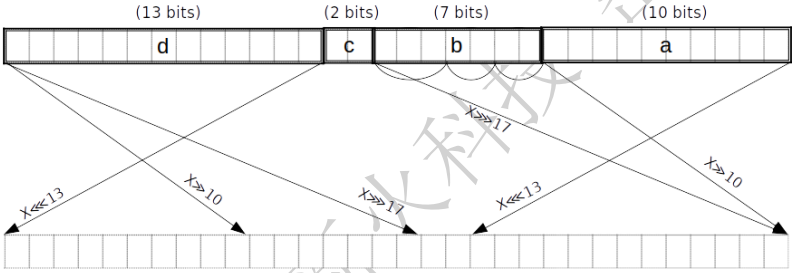
$$\sigma_1(X) = (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10)$$

等价为

$$\sigma_1(X) = (X \lll 15) \oplus (X \lll 13) \oplus (X \gg 10)$$

等价于以下运算：

步骤 1：spread 映射：初始状态：（已有约束）：



sigma1 函数的输入是 X，首先将 X 分解为 (a,b,c,d) 四片，长度分别为 $(10,7,2,13)$，

小端表达。将 b 拆为 3 个 $(3,2,2)$-bit 的片。

步骤 2：算术运算：sigma1 函数中的**循环移位、异或运算等价为**以下基于位拼接的异或运算

$$\sigma_0(X) = \begin{pmatrix} (0^{[10]} \| d \| c \| d) \oplus \\ (b \| a \| d \| c) \oplus \\ (c \| b \| a \| d) \end{pmatrix}$$

再等价为如下域范围内的加法运算

$$R' = \begin{pmatrix} 0 + 4^9 d + 4^7 c + b + \\ 4^{25} b + 4^{15} a + 4^2 d + c + \\ 4^{30} c + 4^{23} b + 4^{13} a + d \end{pmatrix}$$

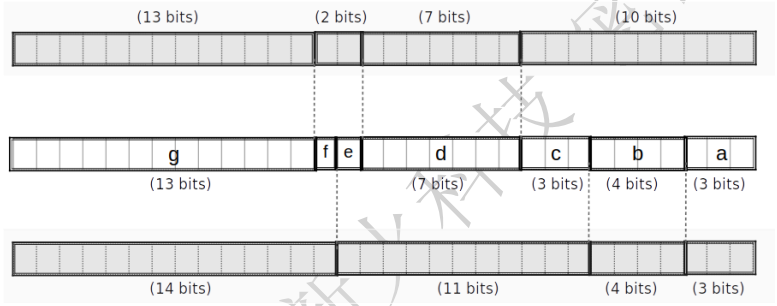
步骤 3：spread 映射：计算结果： sigma1 函数使用 4 个 spread 查找表

$$R' = spread_1\left(R_0^{even}\right) + 2 \cdot spread_2\left(R_0^{odd}\right) + 2^{32} \cdot spread_3\left(R_1^{even}\right) + 2^{33} \cdot spread_4\left(R_1^{odd}\right)$$

最终结果取**偶数位** R_1^{even}, R_2^{even} **。**
对应的电路约束： **步骤 1：spread 查表约束：** **步骤 2：算术电路约束。** **步骤 3：spread 查表约束：4 个** 总共需要 4 个 spread 函数查找表、算术电路约束、一些范围约束。

2.4 消息扩展

在消息扩展过程中，sigma0 函数应用于 W_1 到 W_{48}，sigma1 应用于 W_{14} 到 W_{61}。为防止 spread 函数重复，可以将 W_{14} 到 W_{48} 之间的 **sigma0 和 sigma1 函数合并**，即将长度为 $(3,4,11,14)$ 的 4 片和 $(10,7,2,13)$ 的 4 片合并为长度为 $(3,4,3,7,1,1,13)$ 的 7 片 **$\text{sigma}_{0,1}$ 复合函数**，如图所示：



W_{14} 到 W_{48} 是 35 个 W。sigma0 函数和 sigma1 函数是分解为 4 行，如果能够分解为 3 行，则能够节约 35 行。**每行就是一个 spread 查表。**

如上图所示，第 1 行是长度为 $(3,4,11,14)$ 的 4 片，第 3 行是长度为 $(10,7,2,13)$ 的 4 片，**中间行是两个分片合并后**长度为 $(3,4,3,7,1,1,13)$ 的 7 片。

类似于在轮函数中对 A 和 E 所做的分解合并，当它们被用来计算后面的 W 时，可以合并分解，还原模 2^{32} 的 $W_{16..61}$。

$W_{62..63}$ 不能分解，需要还原。

消息扩展开销为：一行对应一个多项式的值

1. 将 W_0 约束为 32bit=2 个 16bit，**需要 2 行。**

2. 将 $W_{1..13}$ 分解为长度为 $(3,4,11,14)$ 的 4 片（sigma0），长度为 11 和 14 的片**使用 2 个**

spread 查找表实现约束，需要 13*2 行；另外 2 个使用范围约束。

3. 将 $W_{14..48}$ 分解为长度为 $(3,4,3,7,1,1,13)$ 的 7 片（sigma0 与 sigma1 合并），长度为 7 和 13 的片使用 **3 个 spread 查找表**实现约束，需要 35*3 行。48-14+1=35
4. 将 $W_{49..61}$ 分解为长度为 $(10,7,2,13)$ 的 4 片（sigma1），长度为 7 和 13 的片**使用 2 个 spread 查找表**实现约束，需要 **13*2 行**。61-49+1=13
5. σ_0 函数对于 $W_{1..48}$，需要 **4 个 spread 查找表**，需要 48*4 行。
6. σ_1 函数对于 $W_{14..61}$，需要 **4 个 spread 查找表**，需要 48*4 行。61-14+1=48
7. $W_{62..63}$，需要 **2 个 spread 查找表，**需要 2*2 行。
8. 总共 547 行，每行就是一个 spread 查表。

2.5 总开销

1. Maj 函数 4 行，Ch 函数 8 行， Sigma0 函数 6 行，Sigma1 函数 6 行，共计 24 行。
   共计：24*64 轮=1536 行；
2. **消息扩展** 547 行；
3. 输出 32-bit 的 A,B,C,D,E,F,G,H 需要约束，32bit=2*16bit,**8*2 行；**
4. **总共：1536+547+16=2099** 行

对于 $\Sigma_0,\Sigma_1,\sigma_0,\sigma_1$ 对应长度为 $(2,11,9,10),(6,5,14,7),(3,4,11,14),(10,7,2,13)$ 的**分片**，需要一个 2^{16} 行 3 列的 spread 表，需要一个标签 tag 列选择表格中的 $(7,10,11,13,14)$ 子集。使用 spreed 函数进行约束，其中标签 tag 不需要约束，多余部分使用 0 填充。

row	tag	table (16b)	spread (32b)
0	0	0000000000000000	00000000000000000000000000000000
1	0	0000000000000001	00000000000000000000000000000001
2	0	0000000000000010	00000000000000000000000000000100
3	0	0000000000000011	00000000000000000000000000000101
...	0	...	...
2^7-1	0	0000000001111111	00000000000000000001010101010101
2^7	1	0000000010000000	00000000000000000100000000000000
...	1	...	...
$2^{10}-1$	1	0000001111111111	00000000000001010101010101010101
...	2	...	...
$2^{11}-1$	2	0000011111111111	00000000101010101010101010101010
...	3	...	...
$2^{13}-1$	3	0001111111111111	00000001010101010101010101010101
...	4	...	...
$2^{14}-1$	4	0011111111111111	00000101010101010101010101010101
...	5	...	...
$2^{16}-1$	5	1111111111111111	01010101010101010101010101010101

2.6 Maj 门--额外约束

$$Maj(A,B,C)=(A\wedge B)\oplus(A\wedge C)\oplus(B\wedge C)$$

输入上一轮的计算结果：假设上一轮 32-bit 的 A,B,C 已有约束，则仅需要基于 spread 函数，生成 64-bit 的 A',B',C' 映射值。

s_maj	a_0	a_1	a_2	a_3	a_4	a_5
0	{0,1,2,3,4,5}	M_0^{even}	spread(M_0^{even})		spread(A^{lo})	spread(A^{hi})
1	{0,1,2,3,4,5}	M_0^{odd}	spread(M_0^{odd})	spread(M_1^{odd})	spread(B^{lo})	spread(B^{hi})
0	{0,1,2,3,4,5}	M_1^{even}	spread(M_1^{even})		spread(C^{lo})	spread(C^{hi})
0	{0,1,2,3,4,5}	M_1^{odd}	spread(M_1^{odd})			

Maj 函数需要以下 **3 项额外的约束**

1. **代数运算与查表运算的相等性约束：LHS-RHS=0，**其中 LHS 是 spread 查找表计算结果；RHS 是**域范围**内的计算结果

$$LHS=spread(M_0^{even})+2\cdot spread(M_0^{odd})+2^{32}\cdot spread(M_1^{even})+2^{33}\cdot spread(M_1^{odd})$$
$$RHS=A'+B'+C'$$

2. (a_0,a_1,a_2) 的 **spread 约束**；

3. 置换约束 (a_2,a_3)。

输出：$Maj(A,B,C)=M^{odd}=M_0^{odd}+2^{16}\cdot M_1^{odd}$

2.7 Ch 门--额外约束

$$Ch(E,F,G)=(E\wedge F)\oplus(\neg E\wedge G)$$

输入上一轮的计算结果：

1. 假设上一轮 32-bit 的 E,F,G 已经有约束，则仅需要基于 spread 函数，生成 64-bit 的 E',F',G' 映射值。

2. $evens=spread(2^{32}-1),evens_0=evens_1=spread(2^{16}-1)$

如下表所示，**a0 是标签**，a1 是 16-bit 的奇数与偶数计算结果 $P_0^{even},P_0^{odd},P_1^{even},P_1^{odd}$，a2 是 a1 对应的 spread 函数值，a3 和 a4 是 spread 函数值的高位和低位截断。

E ∧ F

s_ch	a_0	a_1	a_2	a_3	a_4
0	{0,1,2,3,4,5}	P_0^{even}	spread(P_0^{even})	spread(E^{lo})	spread(E^{hi})
1	{0,1,2,3,4,5}	P_0^{odd}	spread(P_0^{odd})	spread(P_1^{odd})	
0	{0,1,2,3,4,5}	P_1^{even}	spread(P_1^{even})	spread(F^{lo})	spread(F^{hi})
0	{0,1,2,3,4,5}	P_1^{odd}	spread(P_1^{odd})		

¬E ∧ G

s_ch_neg	a_0	a_1	a_2	a_3	a_4	a_5
0	{0,1,2,3,4,5}	Q_0^{even}	spread(Q_0^{even})	spread(E_{neg}^{lo})	spread(E_{neg}^{hi})	spread(E^{lo})
1	{0,1,2,3,4,5}	Q_0^{odd}	spread(Q_0^{odd})	spread(Q_1^{odd})		spread(E^{hi})
0	{0,1,2,3,4,5}	Q_1^{even}	spread(Q_1^{even})	spread(G^{lo})	spread(G^{hi})	
0	{0,1,2,3,4,5}	Q_1^{odd}	spread(Q_1^{odd})			

Ch 函数需要以下 4 项额外的约束

1. **相等性约束：**LHS-RHS=0，其中 LHS 是 spread 函数值的高位和低位截断计算结果；RHS 是 a1 对应的 spread 函数结果

$$LHS = a_3\omega^{-1} + a_3\omega + 2^{32}(a_4\omega^{-1} + a_4\omega)$$
$$RHS = a_2\omega^{-1} + 2*a_2 + 2^{32}(a_2\omega + 2*a_3)$$

2. **负约束：**需要约束$\neg E$为负号。
3. (a_0, a_1, a_2)的 **spread 约束**。
4. 置换约束(a_2, a_3)。

输出：取奇数位：$Ch(E,F,G) = (P_0^{odd} + Q_0^{odd}) + 2^{16}\cdot(P_1^{odd} + Q_1^{odd})$

2.8 Sigma0 门--额外约束

$\Sigma_0(A)$的入参变量A是一个 32 位字，**分解**为(2,11,9,10)位 chunks，从小数端开始。将这些 chunks 分别称为(a(2),b(11),c(9),d(10))，并且进一步将 c(9)分解为 3 位的 chunks $c(9)^{lo}$,$c(9)^{mid}$,$c(9)^{hi}$。将小 chunks 的 spread 形式写成 witness：

$$\begin{aligned}\Sigma_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)\\ &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \lll 10)\\ &= \begin{pmatrix}(a \| d \| c \| d) \oplus\\ (b \| a \| d \| c) \oplus\\ (c \| b \| a \| d)\end{pmatrix}\\ &= \begin{pmatrix}0 + 4^9 d + 4^7 c + b +\\ 4^{25} b + 4^{15} a + 4^2 d + c +\\ 4^{30} c + 4^{23} b + 4^{13} a + d\end{pmatrix}\end{aligned}$$

$$R' = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

s_upp_sigma_0	a_0	a_1	a_2	a_3	a_4	a_5	a_6
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$c(9)^{lo}$	spread($c(9)^{lo}$)	$c(9)^{mid}$	spread($c(9)^{mid}$)
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($d(10)$)	spread($b(11)$)	$c(9)$
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$a(2)$	spread($a(2)$)	$c(9)^{hi}$	spread($c(9)^{hi}$)
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})				

Sigma0 门需要以下 4 项额外的约束

1. LHS−RHS+tag+decompose=0，其中

$$tag = constrain_1(a_0\omega^{-1}) + constrain_2(a_0\omega)$$

$$decompose = a(2) + 2^2 b(11) + 2^{13} c(9)^{lo} + 2^{16} c(9)^{mid} + 2^{19} c(9)^{hi} + 2^{22} d(10) - A$$

$$LHS = spread_1\left(R_0^{even}\right) + 2 \cdot spread_2\left(R_0^{odd}\right) + 2^{32} \cdot spread_3\left(R_1^{even}\right) + 2^{33} \cdot spread_4\left(R_{1RH}^{odd}\right)$$

$$\begin{aligned} RHS = {} & 4^{30} spread(a(2)) + 4^{20} spread(d(10)) + 4^{17} spread(c(9)^{hi}) + 4^{14} spread(c(9)^{mid}) + \\ & 4^{11} spread(c(9)^{lo}) + spread(b(11)) + 4^{21} spread(b(11)) + 4^{19} spread(a(2)) + \\ & 4^{9} spread(d(10)) + 4^{6} spread(c(9)^{hi}) + 4^{3} spread(c(9)^{mid}) + spread(c(9)^{lo}) + \\ & 4^{29} spread(c(9)^{hi}) + 4^{26} spread(c(9)^{mid}) + 4^{23} spread(c(9)^{lo}) + 4^{12} spread(b(11)) + \\ & 4^{10} spread(a(2)) + spread(d(10)) \end{aligned}$$

2. a0,a1,a2 的 **spread 约束**；
3. a(2)中的 2 比特位范围检查及 2 比特位 **spread 约束**；
4. $c(9)^{lo}$, $c(9)^{mid}$, $c(9)^{hi}$ 上的 **3 比特位范围约束**和 3 比特位 **spread 约束**。

输出：$\Sigma_0(A) = R^{even} = R_0^{even} + 2^{16} R_1^{even}$ 。

2.9 Sigma1 门--额外约束

$\Sigma_1(E)$ 的入参变量 E 是一个 32 位字，分解为(6,5,14,7)位 chunks，从小数端开始。将这些 chunks 分别称为(a(6),b(5),c(14),d(7))，并且进一步将 a(6)分解为两个 3 位的 chunksa$(6)^{lo}$,a$(6)^{hi}$，并更进一步将两个 3 比特位的 chunka$(6)^{lo}$,a$(6)^{hi}$ 和 b 分解为(2,3)比特位的 chunksb$(5)^{lo}$,b$(5)^{hi}$。将小 chunks 的 spread 形式写成 witness：

$$\begin{aligned} \Sigma_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25) \\ &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \lll 7) \\ &= \begin{pmatrix} (a \| d \| c \| d) \oplus \\ (b \| a \| d \| c) \oplus \\ (c \| b \| a \| d) \end{pmatrix} \\ &= \begin{pmatrix} 4^{26} a + 4^{19} d + 4^{5} c + b + \\ 4^{27} b + 4^{21} a + 4^{14} d + c + \\ 4^{18} c + 4^{13} b + 4^{7} a + d \end{pmatrix} \end{aligned}$$

$$R' = spread_1\left(R_0^{even}\right) + 2 \cdot spread_2\left(R_0^{odd}\right) + 2^{32} \cdot spread_3\left(R_1^{even}\right) + 2^{33} \cdot spread_4\left(R_1^{odd}\right)$$

s_upp_sigma_1	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$b(5)^{lo}$	spread($b(5)^{lo}$)	$b(5)^{hi}$	spread($b(5)^{hi}$)	$b(5)$
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($d(7)$)	spread($c(14)$)		
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$a(6)^{lo}$	spread($a(6)^{lo}$)	$a(6)^{hi}$	spread($a(6)^{hi}$)	$a(6)$
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})					

Sigma1 门需要以下 4 项额外的约束

1. LHS−RHS+tag+decompose=0，其中

$$tag = a_0\omega^{-1} + constrain_4(a_0\omega)$$

$$decompose = a(6)^{lo} + 23a(6)^{hi} + 26b(5)^{lo} + 28b(5)^{hi} + 211c(14) + 225d(7) - E$$

$$LHS = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

$$\begin{aligned} RHS = {} & 4^{29}spread(a(6)^{hi}) + 4^{26}spread(a(6)^{lo}) + 4^{19}spread(d(7)) + 4^{5}spread(c(14)) + \\ & 4^{2}spread(b(5)^{hi}) + spread(b(5)^{lo}) + 4^{29}spread(b(5)^{hi}) + 4^{27}spread(b(5)^{lo}) + \\ & 4^{24}spread(a(6)^{hi}) + 4^{21}spread(a(6)^{lo}) + 4^{14}spread(d(7)) + spread(c(14)) + \\ & 4^{18}spread(c(14)) + 4^{15}spread(b(5)^{hi}) + 4^{13}spread(b(5)^{lo}) + 4^{10}spread(a(6)hi) + \\ & 4^{7}spread(a(6)^{lo}) + spread(d(7)) \end{aligned}$$

2. a0,a1,a2 的 **spread 约束**；
3. $b(5)^{lo}$ 中的 2 比特位范围**约束**及 **2 比特位 spread 约束**；
4. $a(6)^{lo}$,$a(6)^{hi}$,$b(4)^{hi}$ 上的 3 比特位**范围约束**和 3 比特位 **spread 约束**；

输出：$\Sigma_1(E) = R^{even} = R_0^{even} + 2^{16}R_1^{even}$

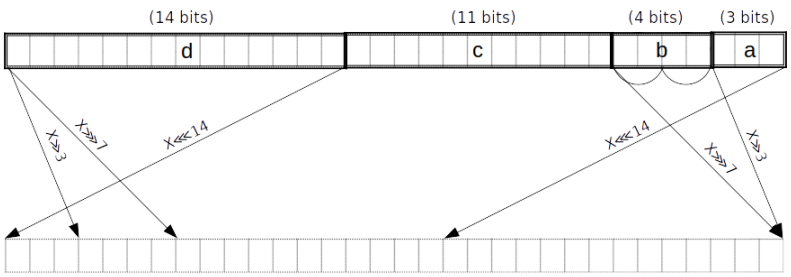
2.10 sigma0 门--额外约束

$\sigma_0(X)$ 的入参变量是一个字，可以分解为(3,4,11,14)位的 chunks（已经在消息调度中被约束）。

将这些 chunks 称为(a(3),b(4),c(11),d(14)).b(4)被进一步分解为两个 2 比特位的 chunks $b(4)^{lo}$,$b(4)^{hi}$.将小 chunks 的 spread 形式写成 witness。从消息调度中已经获得了 spread(c(11))和 spread(d(14))。

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3) \\ &= (X \ggg 7) \oplus (X \lll 14) \oplus (X \gg 3) \\ &= \begin{pmatrix} (0^{[3]} \| d \| c \| d) \oplus \\ (b \| a \| d \| c) \oplus \\ (c \| b \| a \| d) \end{pmatrix} \\ &= \begin{pmatrix} 0 + 4^{15}d + 4^{4}c + b + \\ 4^{28}b + 4^{25}a + 4^{11}d + c + \\ 4^{21}c + 4^{17}b + 4^{14}a + d \end{pmatrix} \end{aligned}$$

$$R' = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$



s_low_sigma_0	a_0	a_1	a_2	a_3	a_4	a_5	a_6
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$b(4)^{lo}$	spread($b(4)^{lo}$)	$b(4)^{hi}$	spread($b(4)^{hi}$)
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread(c)	spread(d)	$b(4)$
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	0	0	a	spread(a)
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})				

需要以下 4 项额外的约束

1. LHS−RHS=0，其中

$$LHS = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

$$RHS = 4^{15}d(14) + 4^4 c(11) + 4^2 b(4)hi + b(4)^{lo} +$$
$$4^{30}b(4)^{hi} + 4^{28}b(4)lo + 4^{25}a(3) + 4^{11}d(14) + c(11) +$$
$$4^{21}c(11) + 4^{19}b(4)hi + 4^{17}b(4)lo + 4^{14}a(3) + d(14)$$

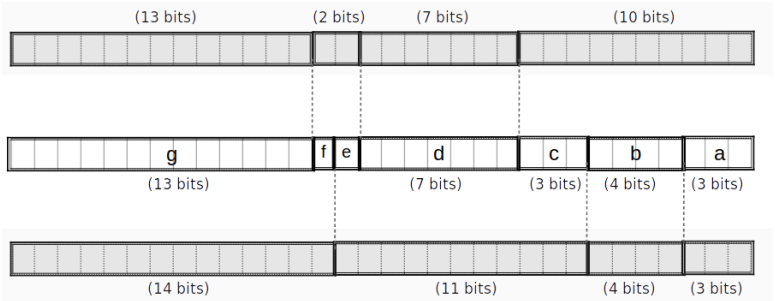
2. b **分解约束**：4 比特位的片 $W^{b(4)lo} + 2^2 W^{b(4)hi} - W = 0$；
3. $b(4)^{lo}, b(4)^{hi}$ 中的 2 比特位**范围约束**及 2 比特位 **spread 约束**；
4. a(3)上的 3 比特位**范围约束**和 **3 比特位 spread 约束**。

2.11 sigma0 门--额外约束

sigma0 和 sigma1 函数合并阶段：

$\sigma_0(X)$ 的入参变量是一个字，可以分解为(3,4,3,7,1,1,13)位的 chunks（已经在消息调度中被约束）。将这些 chunks 称为(a(3),b(4),c(3),d(7),e(1),f(1),g(13)).从消息调度中已经获得了 spread(d(7))和 spread(g(13))。1 比特位的 e(1),f(1)在 spread 操作中不做更改，可以直接使用。另外 b(4)被进一步分解为两个 2 比特位的 chunksb$(4)^{lo}$,b$(4)^{hi}$。将小 chunks 的 spread 形式写成 witness。

$$\sigma_0(X) = (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3)$$
$$= (X \ggg 7) \oplus (X \lll 14) \oplus (X \gg 3)$$



$$R' = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

s_low_sigma_0_v2	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$b(4)^{lo}$	spread($b(4)^{lo}$)	$b(4)^{hi}$	spread($b(4)^{hi}$)	
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($d(7)$)	spread($g(13)$)	$b(4)$	$e(1)$
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$a(3)$	spread($a(3)$)	$c(3)$	spread($c(3)$)	$f(1)$
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})					

需要以下 4 项额外约束

1. LHS−RHS=0，其中

$$LHS = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

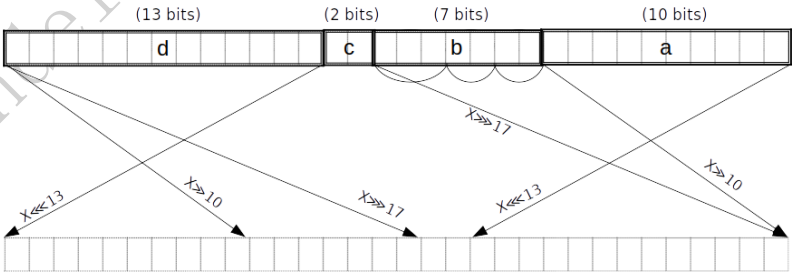
$$RHS = 4^{16}g(13) + 4^{15}f(1) + 4^{14}e(1) + 4^{7}d(7) + 4^{4}c(3) + 4^{2}b(4)^{hi} + b(4)^{lo} +$$
$$4^{30}b(4)^{hi} + 4^{28}b(4)^{lo} + 4^{25}a(3) + 4^{12}g(13) + 4^{11}f(1) + 4^{10}e(1) + 4^{3}d(7) + c(3) +$$
$$4^{31}e(1) + 4^{24}d(7) + 4^{21}c(3) + 4^{19}b(4)hi + 4^{17}b(4)^{lo} + 4^{14}a(3) + 4^{1}g(13) + f(1)$$

2. b 的**分解约束**：4 比特位的片 $W^{b(4)lo}+2^2W^{b(4)hi}-W=0$；
3. $b(4)^{lo}$,$b(4)^{hi}$ 中的 2 比特位**范围约束**及 **2 比特位 spread 约束**；
4. a(3),c(3)上的 3 比特位**范围约束**和 **3 比特位 spread 约束**。

2.12 sigma1 门--额外约束

$\sigma_1(X)$ 的入参变量是一个字，可以分解为(10,7,2,13)位的 chunks（已经在消息调度中被约束）。将这些 chunks 称为(a(10),b(7),c(2),d(13)).b(7)被进一步分解为(2,2,3)比特位的 chunksb(7)lo,b(7)mid,b(7)hi。将小 chunks 的 spread 形式写成 witness。从消息调度中已经获得 spread(a(10))和 spread(d(13))。

$$\sigma_1(X) = (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10)$$
$$= (X \lll 15) \oplus (X \lll 13) \oplus (X \gg 10)$$



s_low_sigma_1	a_0	a_1	a_2	a_3	a_4	a_5	a_6
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$b(7)^{lo}$	spread($b(7)^{lo}$)	$b(7)^{mid}$	spread($b(7)^{mid}$)
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($a(10)$)	spread($d(13)$)	$b(7)$
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$c(2)$	spread($c(2)$)	$b(7)^{hi}$	spread($b(7)^{hi}$)
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})				

需要以下 4 项额外约束

1. LHS−RHS=0，其中

$$LHS = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

$$\begin{aligned} RHS = {} & 4^9 d(13) + 4^7 c(2) + 4^4 b(7)^{hi} + 4^2 b(7)^{mid} + b(7)^{lo} + \\ & 4^{29} b(7)^{hi} + 4^{27} b(7)^{mid} + 4^{25} b(7)^{lo} + 4^{15} a(10) + 4^2 d(13) + c(2) + \\ & 4^{30} c(2) + 4^{27} b(7)^{hi} + 4^{25} b(7)^{mid} + 4^{23} b(7)^{lo} + 4^{13} a(10) + d(13) \end{aligned}$$

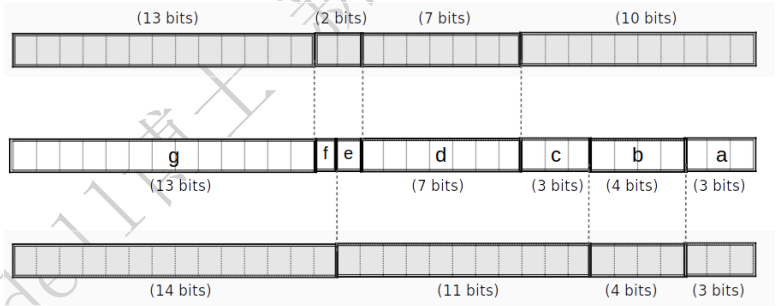
2. b 的**分解约束**：7 比特位的片 $W^{b(7)lo}+2^2W^{b(7)mid}+2^4W^{b(7)hi}-W=0$；
3. $b(7)^{lo}$,$b(7)^{mid}$,c(2)中的 2 比特位**范围约束**及 **2 比特位 spread 约束**；
4. $b(7)^{hi}$ 上的 **3 比特位范围约束**和 **3 比特位 spread 约束**。

2.13 sigma1 门--额外约束

sigma0 和 sigma1 函数合并阶段：

$\sigma_1(X)$ 的入参变量是一个字，可以分解为(3,4,3,7,1,1,13)位的 chunks（已经在消息调度中被约束）。将这些 chunks 称为(a(3),b(4),c(3),d(7),e(1),f(1),g(13)).从消息调度中已经获得了 spread(d(7))和 spread(g(13))。1 比特位的 e(1),f(1)在 spread 操作中不做更改，可以直接使用。另外 b(4)被进一步分解为两个 2 比特位的 chunksb$(4)^{lo}$,b$(4)^{hi}$。将小 chunks 的 spread 形式写成 witness。

$$\begin{aligned} \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10) \\ &= (X \lll 15) \oplus (X \lll 13) \oplus (X \gg 10) \end{aligned}$$



$$R' = spread_1\left(R_0^{even}\right) + 2\cdot spread_1\left(R_0^{odd}\right) + 2^{32}\cdot spread_2\left(R_1^{even}\right) + 2^{33}\cdot spread_2\left(R_1^{odd}\right)$$

s_low_sigma_1_v2	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$b(4)^{lo}$	spread($b(4)^{lo}$)	$b(4)^{hi}$	spread($b(4)^{hi}$)	
1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($d(7)$)	spread($g(13)$)	$b(4)$	$e(1)$
0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$a(3)$	spread($a(3)$)	$c(3)$	spread($c(3)$)	$f(1)$
0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})					

需要以下 4 项额外约束

1. LHS-RHS=0，其中

$$LHS = spread_1\left(R_0^{even}\right) + 2\cdot spread_2\left(R_0^{odd}\right) + 2^{32}\cdot spread_3\left(R_1^{even}\right) + 2^{33}\cdot spread_4\left(R_1^{odd}\right)$$

$$RHS = 4^9 g(13) + 4^8 f(1) + 4^7 e(1) + d(7) +$$
$$4^{25} d(7) + 4^{22} c(3) + 4^{20} b(4)hi + 4^{18} b(4)lo + 4^{15} a + 42g(13) + 4^1 f(1) + e(1)$$
$$4^{31} f(1) + 4^{30} e(1) + 4^{23} d(7) + 4^{20} c(3) + 4^{18} b(4)^{hi} + 4^{16} b(4)^{lo} + 4^{13} a + g(13)$$

2. b 的**分解约束**：4 比特位的片 $W^{b(4)lo}+2^2W^{b(4)hi}-W=0$；

3. $b(4)^{lo}, b(4)^{hi}$ 中的 **2 比特位范围约束**及 **2 比特位 spread 约束**；

4. a(3)，c(3)上的 **3 比特位范围约束**和 **3 比特位 spread 约束**。

2.14 辅助门约束

2.14.1 小范围约束

令 $constrain_n(x) = \prod_{i=0}^{n}(x-i)$ **约束**该表达式的值为 0，相当于**约束** x∈[0..n].

2.24.2 两比特位范围约束

(a−3)(a−2)(a−1)(a)=0

sr2	a_0
1	a

2.14.3 两比特位 spread 约束

$l_1(a)+4*l_2(a)+5*l_3(a)-a'=0$

ss2	a_0	a_1
1	a	a'

插值多项式：

$$l_0(a) = \frac{(a-3)(a-2)(a-1)}{(-3)(-2)(-1)}\left(spread(00)=0000\right)$$
$$l_1(a) = \frac{(a-3)(a-2)(a)}{(-2)(-1)(1)}\left(spread(01)=0001\right)$$
$$l_2(a) = \frac{(a-3)(a-1)(a)}{(-1)(1)(2)}\left(spread(10)=0100\right)$$
$$l_3(a) = \frac{(a-2)(a-1)(a)}{(1)(2)(3)}\left(spread(11)=0101\right)$$

2.14.4 三比特位范围约束

(a−7)(a−6)(a−5)(a−4)(a−3)(a−2)(a−1)(a)=0

sr3	a_0
1	a

2.14.5 三比特位 spread 约束

$l_1(a)+4*l_2(a)+5*l_3(a)+16*l_4(a)+17*l_5(a)+20*l_6(a)+21*l_7(a)-a'=0$

ss3	a_0	a_1
1	a	a'

插值多项式：

$$l_0(a)=\frac{(a-7)(a-6)(a-5)(a-4)(a-3)(a-2)(a-1)}{(-7)(-6)(-5)(-4)(-3)(-2)(-1)}\left(spread(000)=000000\right)$$

$$l_1(a)=\frac{(a-7)(a-6)(a-5)(a-4)(a-3)(a-2)(a)}{(-6)(-5)(-4)(-3)(-2)(-1)(1)}\left(spread(001)=000001\right)$$

$$l_2(a)=\frac{(a-7)(a-6)(a-5)(a-4)(a-3)(a-1)(a)}{(-5)(-4)(-3)(-2)(-1)(1)(2)}\left(spread(010)=000100\right)$$

$$l_3(a)=\frac{(a-7)(a-6)(a-5)(a-3)(a-2)(a-1)(a)}{(-4)(-3)(-2)(-1)(1)(2)(3)}\left(spread(011)=000101\right)$$

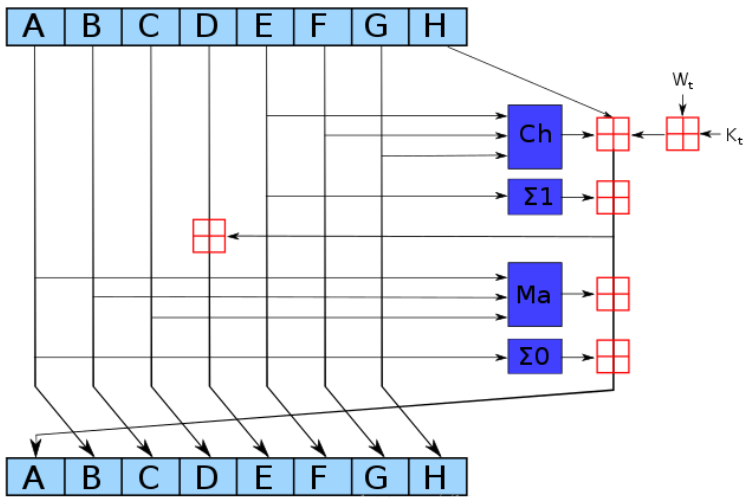
$$l_4(a)=\frac{(a-7)(a-6)(a-5)(a-3)(a-2)(a-1)(a)}{(-3)(-2)(-1)(1)(2)(3)(4)}\left(spread(100)=010000\right)$$

$$l_5(a)=\frac{(a-7)(a-6)(a-4)(a-3)(a-2)(a-1)(a)}{(-2)(-1)(1)(2)(3)(4)(5)}\left(spread(101)=010001\right)$$

$$l_6(a)=\frac{(a-7)(a-5)(a-4)(a-3)(a-2)(a-1)(a)}{(-1)(1)(2)(3)(4)(5)(6)}\left(spread(110)=010100\right)$$

$$l_7(a)=\frac{(a-6)(a-5)(a-4)(a-3)(a-2)(a-1)(a)}{(1)(2)(3)(4)(5)(6)(7)}\left(spread(111)=010101\right)$$

2.14.6 Reduce_6 门约束



6 个元素的(mod2^{32})加法 $E_{new} \leftarrow K_t \oplus W_t \oplus H \oplus Ch \oplus \Sigma_1 \oplus D$

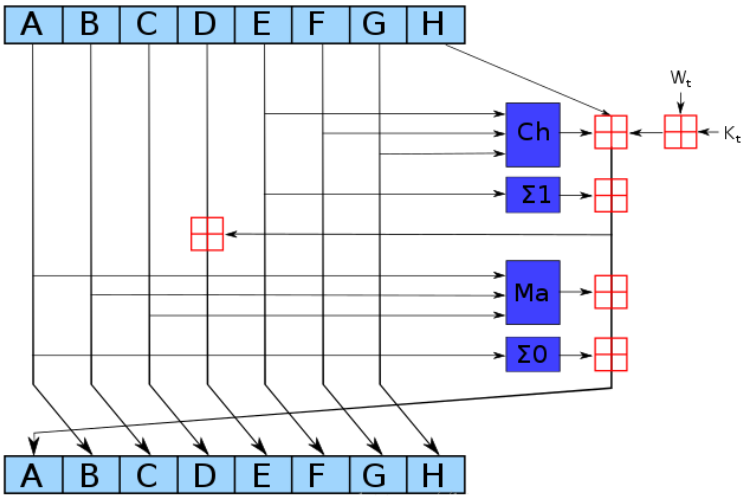
输入： 运算结果 E_new，每位 $\{e_i^{lo}, e_i^{hi}\}_{i=0}^{5}$，进位 carry

计算：E=e0+e1+e2+e3+e4+e5(mod32)
假设输入被约束为 16 个比特位

- **加法门约束**(sa):
  - $a_0+a_1+a_2+a_3+a_4+a_5+a_6-a_7=0$
- **进位门约束**(sc): 6 个数相加，最多有 5 个进位
  - $2^{16}a_6\omega^{-1}+a_6+[(a_6-5)(a_6-4)(a_6-3)(a_6-2)(a_6-1)(a_6)]=0$

sa	sc	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	0	e_0^{lo}	e_1^{lo}	e_2^{lo}	e_3^{lo}	e_4^{lo}	e_5^{lo}	$-carry*2^{16}$	E^{lo}
1	1	e_0^{hi}	e_1^{hi}	e_2^{hi}	e_3^{hi}	e_4^{hi}	e_5^{hi}	$carry$	E^{hi}

2.14.7 Reduce_7 门约束



7 个元素的(mod2^{32})加法 $A_{new} \leftarrow K_t \oplus W_t \oplus H \oplus Ch \oplus \Sigma_1 \oplus Ma \oplus \Sigma_0$

输入：运算结果 A_new，每位 $\{e_i^{lo}, e_i^{hi}\}_{i=0}^{6}$，进位 carry

计算：E=e0+e1+e2+e3+e4+e5+e6(mod32)
假设输入被约束为 16 个比特位

- **加法门约束**(sa): 7 个数相加
  - $a_0+a_1+a_2+a_3+a_4+a_5+a_6+a_7-a_8=0$
- **进位门约束**(sc): 7 个数相加，最多进位为 6
  - $2^{16}a_7\omega^{-1}+a_7+[(a_7-6)(a_7-5)(a_7-4)(a_7-3)(a_7-2)(a_7-1)(a_7)]=0$

sa	sc	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1	0	e_0^{lo}	e_1^{lo}	e_2^{lo}	e_3^{lo}	e_4^{lo}	e_5^{lo}	e_6^{lo}	$-carry*2^{16}$	E^{lo}
1	1	e_0^{hi}	e_1^{hi}	e_2^{hi}	e_3^{hi}	e_4^{hi}	e_5^{hi}	e_6^{hi}	$carry$	E^{hi}

2.14.8 消息调度区域

对于消 M∈{0,1}，512 中的每一个 block，其 64 个 32 位字以如下方式构造：
通过将 M 以每 32 个比特进行划分，得到前 16 个

$$M=W_0||W_1||\cdots||W_{14}||W_{15}$$

剩下的 48 个字使用如下的方式构造：

$$\text{for16}\le i<64.\quad W_i=\sigma_1(W_{i-2})\boxplus W_{i-7}\boxplus\sigma_0(W_{i-15})\boxplus W_{i-16}$$

sw	sd0	sd1	sd2	sd3	ss0	ss0_v2	ss1	ss1_v2	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
0	1	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_0^{lo}	spread(W_0^{lo})	W_9^{lo}	W_2^{hi}	W_9	$\sigma_0(W_1)^{lo}$	$\sigma_1(W_{14})^{lo}$	W_2^{lo}	
1	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_0^{hi}	spread(W_0^{hi})			W_{16}	$\sigma_0(W_1)^{hi}$	$\sigma_1(W_{14})^{hi}$	W_2^{hi}	$carry_{16}$
0	1	1	0	0	0	0	0	0	{0,1,2,3,4}	$W_1^{a(14)}$	spread($W_1^{a(14)}$)	W_1^{lo}	W_1^{hi}	W_1	$\sigma_0(W_2)^{lo}$	$\sigma_1(W_{15})^{lo}$	W_{10}^{lo}	
1	0	0	0	0	0	0	0	0	{0,1,2}	$W_1^{c(11)}$	spread($W_1^{c(11)}$)	$W_1^{w(3)}$	$W_1^{w(4)}$	W_{17}	$\sigma_0(W_2)^{hi}$	$\sigma_1(W_{15})^{hi}$	W_{10}^{hi}	$carry_{17}$
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$W_1^{w(4)lo}$	spread($W_1^{w(4)lo}$)	$W_1^{w(4)hi}$	spread($W_1^{w(4)hi}$)			
0	0	0	0	0	1	0	0	0	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($W_1^{c(11)}$)	spread($W_1^{a(14)}$)	$W_1^{w(4)}$			
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_0^{odd}	spread(R_1^{even})	0	0	$W_1^{w(3)}$	spread($W_1^{w(3)}$)			
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{odd})	$\sigma_0 v1 R_0$	$\sigma_0 v1 R_1$	$\sigma_0 v1 R_0^{even}$	$\sigma_0 v1 R_0^{odd}$			
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
0	0	0	0	0	0	0	0	0	{0,1,2,3}	$W_{14}^{a(13)}$	spread($W_{14}^{a(13)}$)	$W_{14}^{w(3)}$	$W_{14}^{w(3)}$					
0	1	0	1	0	0	0	0	0	0	$W_{14}^{a(7)}$	spread($W_{14}^{a(7)}$)	W_{14}^{lo}	W_{14}^{hi}	W_{14}	$\sigma_0(W_{15})^{lo}$	$\sigma_1(W_{28})^{lo}$	W_{23}^{lo}	
1	0	0	0	0	0	0	0	0	0	$W_{14}^{a(4)}$	spread($W_{14}^{a(4)}$)	$W_{14}^{c(1)}$	$W_{14}^{f(1)}$	W_{29}	$\sigma_0(W_{15})^{hi}$	$\sigma_1(W_{28})^{hi}$	W_{23}^{hi}	
$carry_{30}$																		
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$W_{14}^{w(4)lo}$	spread($W_{14}^{w(4)lo}$)	$W_{14}^{w(4)hi}$	spread($W_{14}^{w(4)hi}$)			
0	0	0	0	0	0	1	0	0	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread($W_{14}^{a(7)}$)	spread($W_{14}^{a(13)}$)	$W_{14}^{a(14)}$	$W_{14}^{c(1)}$		
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$W_{14}^{w(3)}$	spread($W_{14}^{a(3)}$)	$W_{14}^{c(3)}$	spread($W_{14}^{c(3)}$)	$W_{14}^{f(1)}$		
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})	$\sigma_0 v2 R_0$	$\sigma_0 v2 R_1$	$\sigma_0 v2 R_0^{even}$	$\sigma_0 v2 R_0^{odd}$			
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$W_{14}^{w(4)lo}$	spread($W_{14}^{w(4)lo}$)	$W_{14}^{w(4)hi}$	spread($W_{14}^{w(4)hi}$)			
0	0	0	0	0	0	0	0	1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread(d)	spread(g)		$W_{14}^{c(1)}$		
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$W_{14}^{w(3)}$	spread($W_{14}^{w(3)}$)	$W_{14}^{c(3)}$	spread($W_{14}^{c(3)}$)	$W_{14}^{f(1)}$		
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})	$\sigma_1 v2 R_0$	$\sigma_1 v2 R_1$	$\sigma_1 v2 R_0^{even}$	$\sigma_1 v2 R_0^{odd}$			
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
0	1	0	0	1	0	0	0	0	{0,1,2,3}	$W_{49}^{a(13)}$	spread($W_{49}^{a(13)}$)	W_{49}^{lo}	W_{49}^{hi}	W_{49}				
0	0	0	0	0	0	0	0	0	{0,1}	$W_{49}^{c(10)}$	spread($W_{49}^{c(10)}$)	$W_{49}^{c(2)}$	$W_{49}^{f(7)}$					
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_0^{even}	spread(R_0^{even})	$W_{49}^{w(7)lo}$	spread($W_{49}^{w(7)lo}$)	$W_{49}^{w(7)mid}$	spread($W_{49}^{w(7)mid}$)			
0	0	0	0	0	0	0	0	1	{0,1,2,3,4,5}	R_0^{odd}	spread(R_0^{odd})	spread(R_1^{odd})	spread(a)	spread(d)	$W_{49}^{w(49)}$			
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{even}	spread(R_1^{even})	$W_{49}^{c(2)}$	spread($W_{49}^{c(2)}$)	$W_{49}^{w(7)hi}$	spread($W_{49}^{w(7)hi}$)			
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	R_1^{odd}	spread(R_1^{odd})	$\sigma_1 v1 R_0$	$\sigma_1 v1 R_1$	$\sigma_1 v1 R_0^{even}$	$\sigma_1 v1 R_0^{odd}$			
..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
0	1	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_{62}^{a}	spread(W_{62}^{a})	W_{62}^{lo}	W_{62}^{hi}	W_{62}				
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_{62}^{hi}	spread(W_{62}^{hi})							
0	1	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_{63}^{a}	spread(W_{63}^{a})	W_{63}^{lo}	W_{63}^{hi}	W_{63}				
0	0	0	0	0	0	0	0	0	{0,1,2,3,4,5}	W_{63}^{hi}	spread(W_{63}^{hi})							

如下约束：

- sw: 使用 reduce_4 门约束：**4 个元素的(mod2^{32})加法约束**

  $$W_i=\sigma_1(W_{i-2})\boxplus W_{i-7}\boxplus\sigma_0(W_{i-15})\boxplus W_{i-16}$$

- sd0: W0,W62,W63 对应的**分解约束** $W^{lo}+2^{16}W^{hi}-W=0$
- sd1: W1..13 的 sigma0 **分解约束**(分解为(3,4,11,14)位的片)

  $$W^{a(3)}+2^3W^{b(4)lo}+2^5W^{b(4)hi}+2^7W^{c(11)}+2^{18}W^{d(14)}-W=0$$

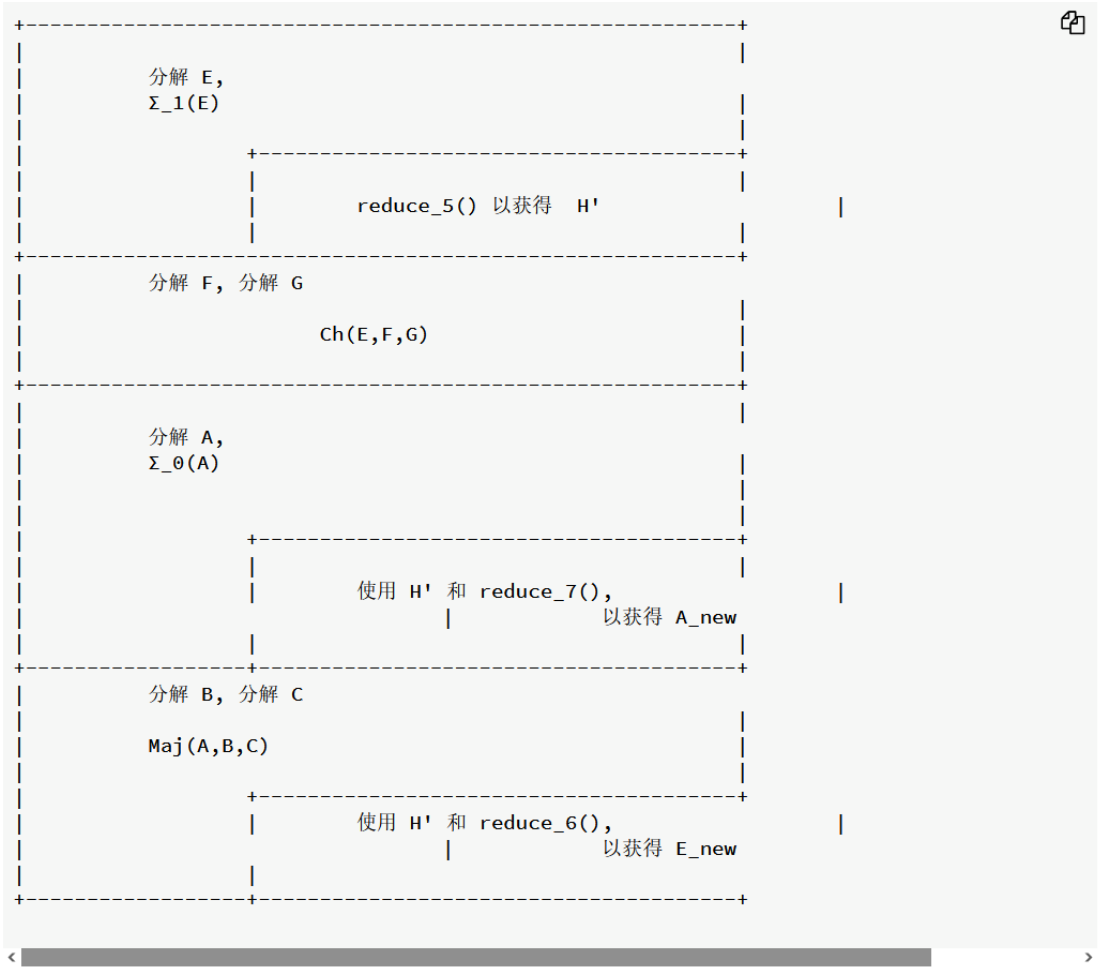
- sd2: W14..48 的 sigma0 与 sigma1 **分解约束**(分解为(3,4,3,7,1,1,13)位的片)

  $$W^{a(3)}+2^3W^{b(4)lo}+2^5W^{b(4)hi}+2^7W^{c(11)}+2^{10}W^{d(14)}+2^{17}W^{e(1)}+2^{18}W^{f(1)}+2^{19}W^{g(13)}-W=0$$

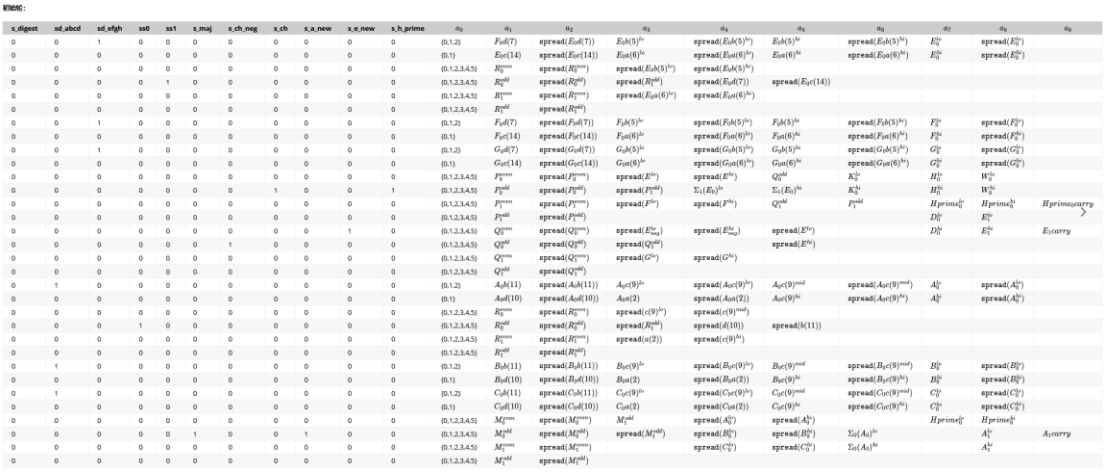
- sd3: W49..61 的 sigma1 **分解约束**(分解为(10,7,2,13)位的片)

  $$W^{a(10)}+2^{10}W^{b(7)lo}+2^{12}W^{b(7)mid}+2^{15}W^{b(7)hi}+2^{17}W^{c(2)}+2^{19}W^{d(13)}-W=0$$

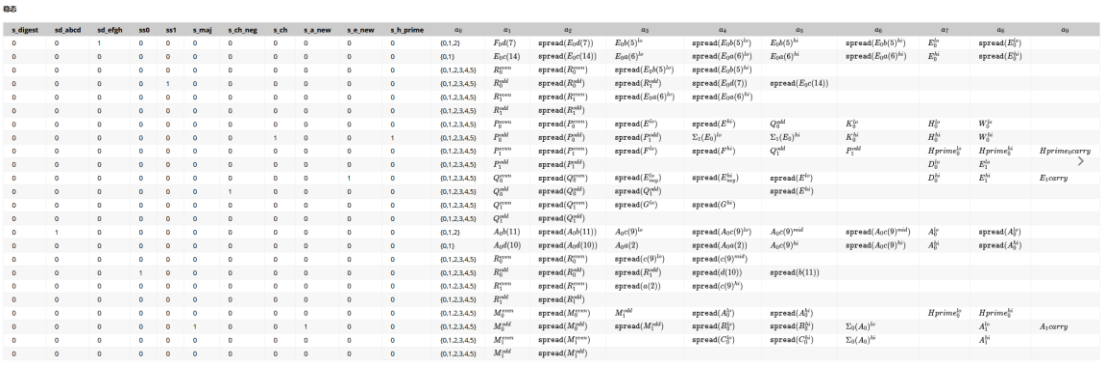
2.14.9 Compression 区域



初始态



稳态



s_digest	sd_abcd	sd_efgh	ss0	ss1	s_maj	s_ch_neg	s_ch	s_a_new	s_e_new	s_h_prime	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
1	0	0	0	0	0	0	0	0	0	0	0	0	0	A_{63}^{lo}	A_{63}^{hi}	A_{63}	B_{63}^{lo}	B_{63}^{hi}	B_{63}	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	C_{63}^{lo}	C_{63}^{hi}	C_{63}	C_{63}^{lo}	C_{63}^{hi}	C_{63}	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	E_{63}^{lo}	E_{63}^{hi}	E_{63}	G_{63}^{lo}	G_{63}^{hi}	G_{63}	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	F_{63}^{lo}	F_{63}^{hi}	F_{63}	H_{63}^{lo}	H_{63}^{hi}	H_{63}	

最终输出结果

最后的摘要输出:

s_digest	sd_abcd	sd_efgh	ss0	ss1	s_maj	s_ch_neg	s_ch	s_a_new	s_e_new	s_h_prime	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
1	0	0	0	0	0	0	0	0	0	0	0	0	0	A_{63}^{lo}	A_{63}^{hi}	A_{63}	B_{63}^{lo}	B_{63}^{hi}	B_{63}	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	C_{63}^{lo}	C_{63}^{hi}	C_{63}	C_{63}^{lo}	C_{63}^{hi}	C_{63}	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	E_{63}^{lo}	E_{63}^{hi}	E_{63}	G_{63}^{lo}	G_{63}^{hi}	G_{63}	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	F_{63}^{lo}	F_{63}^{hi}	F_{63}	H_{63}^{lo}	H_{63}^{hi}	H_{63}	

lynndell 博士 新火科技 密码学专家 lynndell2010@gmail.com