SHA3

前置知识：B 、W、L 对照表

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
ℓ	0	1	2	3	4	5	6

Table 1: KECCAK-*p* permutation widths and related quantities

对于SHA3家族：B固定为1600。所以 W、L也固定了。

第一步

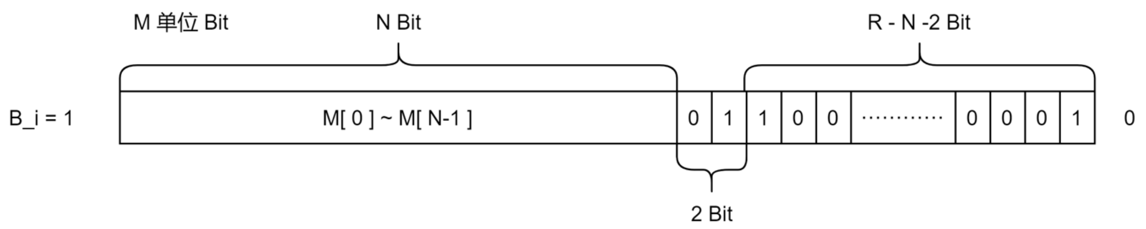
设 需要散列的 文本为"M"，M的长度为N Bit 。

需要将M 填充到 N mod r = 0;

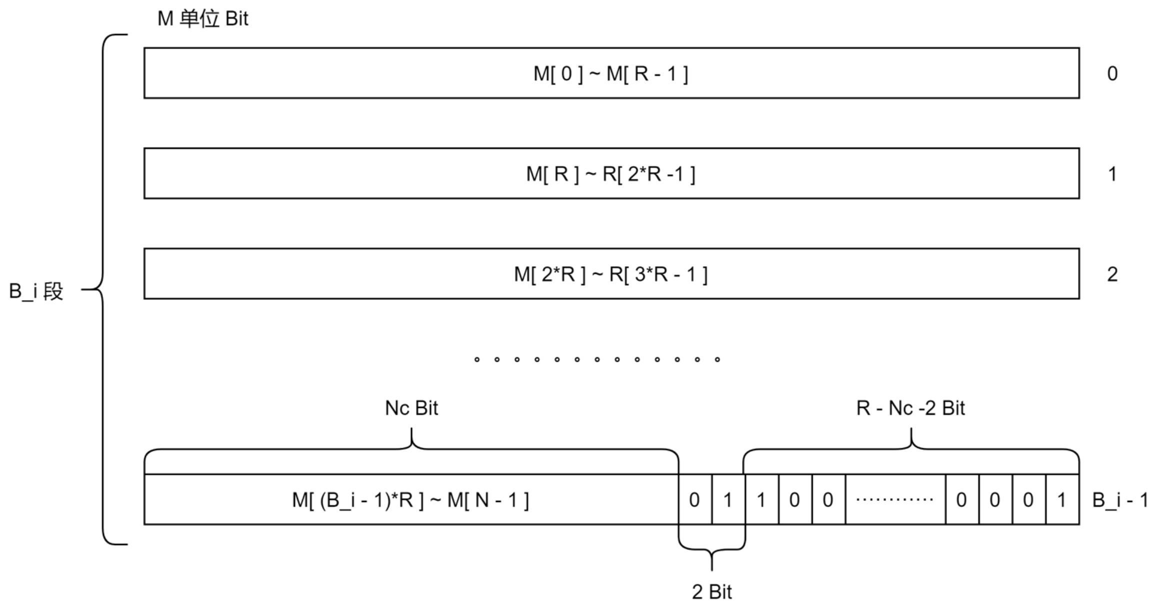
```
SHA3-224: r = 1152          SHA3-256: r = 1088
SHA3-384: r = 832           SHA3-512: r = 576
```

填充规则：先串接 01 ， 再串接 100....001。图例：

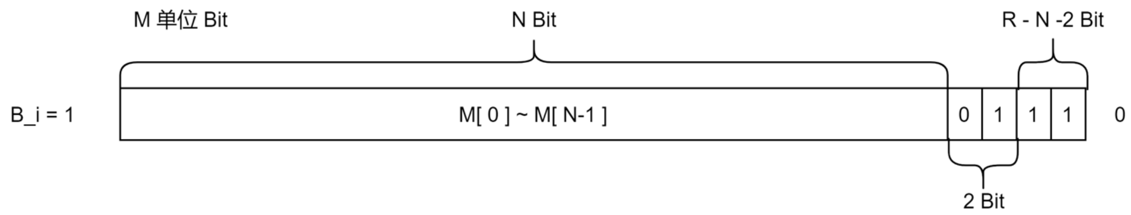
例子1：M 很短时



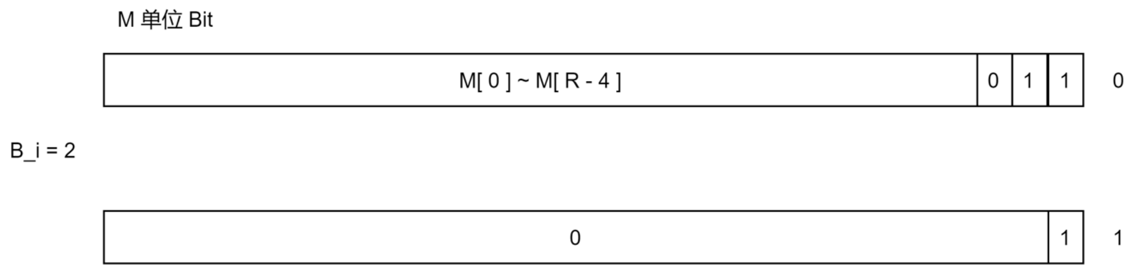
例子2：M 长时



例子3：填充最少



例子4：填充最多

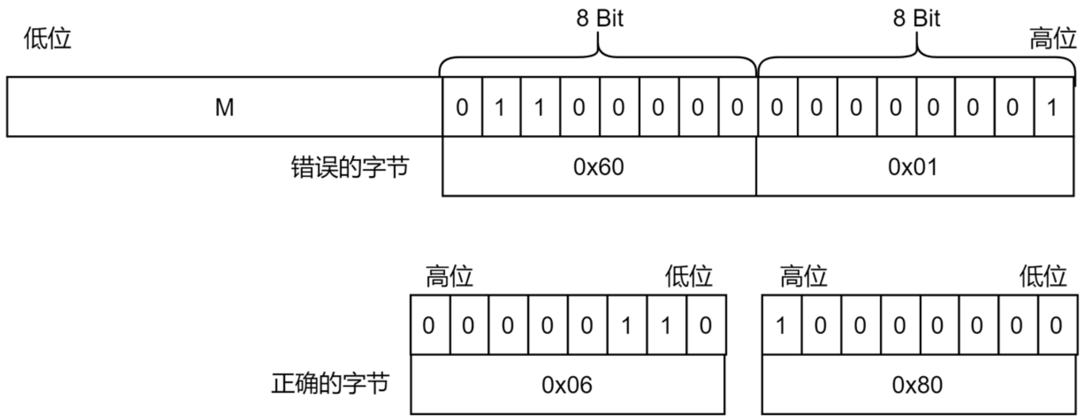


注意：

在计算机的实际应用中，基本单位是字节，因此填充的部分，也应该是字节。以 8Bit 为单位：

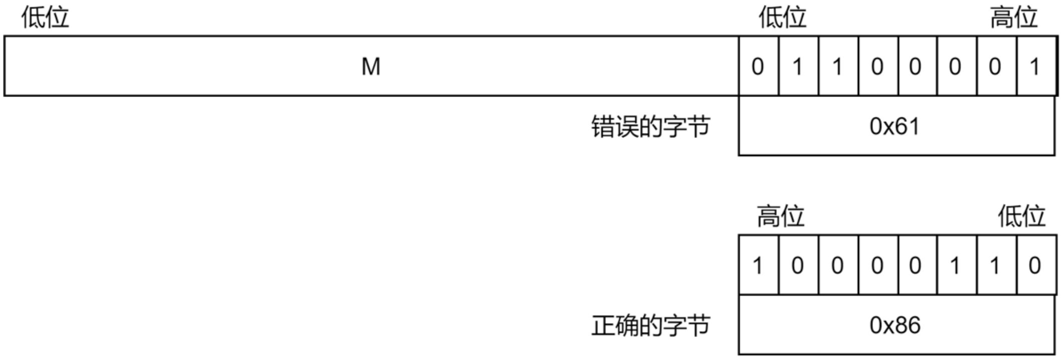
在上图中 M[0] 为最低位的 Bit，最后填补的 Bit 1 为最高位的 Bit。若以 字节的形式表示如下：

例如我需要填补 2个字节才能补足 长度 R Bit：



可以看出：我们补的字节并不是 0x60 与 0x01 ，而补的是 0x06 与 0x80 。这一部分和SHA1、SHA2、MD家族有些许区别。

若只补1个字节：



补的不是 0x61 而是 0x86。

这一部分是一个大坑。

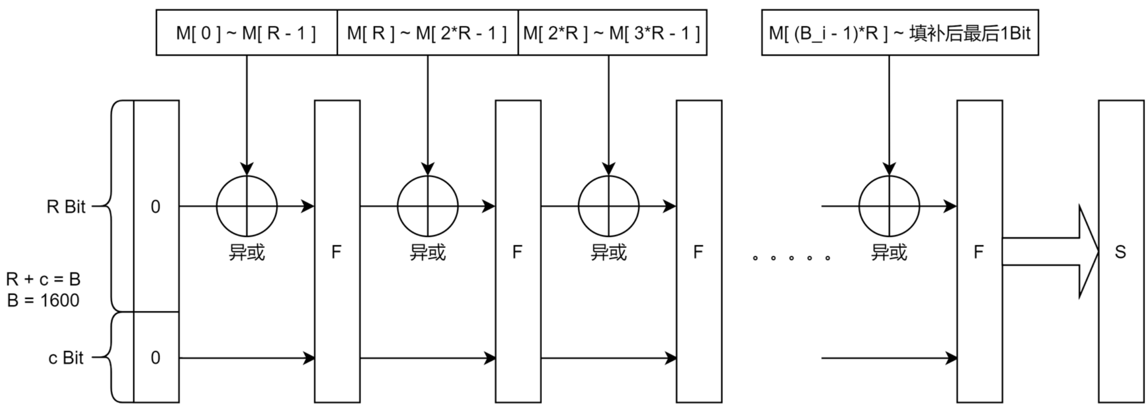
第二步

迭代：

对上面的每一段 M 进行迭代。初始向量 S 为 长度为B的全 0 Bit串。

S 会经过多次迭代变化。经过最后一个 函数F 得到的 S 为HASH结果。

取 S 前 （224、256、384、512）Bit 即为HASH输出。



详解 迭代函数 F：

前置知识：

A[x,y,z] = S[W(5y+x)+z]， S 即为上图的 S 。还有两个工具函数，使用代码表示：

```
#define A(x,y,z) (W*(5*y+x)+z)

//获得 S 的第 A(x,y,z) 个 Bit 的值（UCHAR 类型表示）
UCHAR Get_Bit(S,A(x,y,z));

//将 S 的第 A(x+1,y+1,z+1) 个 Bit 的值 赋值为 S 的第 A(x,y,z) 个 Bit 的值
VOID Set_Bit(S,A(x+1,y+1,z+1),Get_Bit(S,A(x,y,z)));
```

```
VOID SHA3_512_Data::KECCAK_P(PUCHAR S) { //这个函数就是 上述流程图中的 函数 F
    for (int ir = 12 + 2 * L - Nr; ir <= 12 + 2 * L - 1; ir++) {
        Rnd(S, ir);
    }
}
```

其中：

Rnd：

$$\mathrm{Rnd}(\mathbf{A}, i_r) = \iota(\chi(\pi(\rho(\theta(\mathbf{A})))), i_r).$$

θ(theat)：

Steps:

1. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$, let
   $C[x,z] = \mathbf{A}[x,0,z] \oplus \mathbf{A}[x,1,z] \oplus \mathbf{A}[x,2,z] \oplus \mathbf{A}[x,3,z] \oplus \mathbf{A}[x,4,z]$.
2. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$ let
   $D[x,z] = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$.
3. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
   $\mathbf{A}'[x,y,z] = \mathbf{A}[x,y,z] \oplus D[x,z]$.

代码（节选）：

```
for (int x = 0; x < 5; x++) {
        for (int z = 0; z < W; z++) {
            Set_Bit(C, W * x + z, Get_Bit(S, A(x, 0, z)) ^ Get_Bit(S, A(x, 1,
z)) ^ Get_Bit(S, A(x, 2, z)) ^ Get_Bit(S, A(x, 3, z)) ^ Get_Bit(S, A(x, 4, z)));
        }
    }
    for (int x = 0; x < 5; x++) {
        for (int z = 0; z < W; z++) {
            Set_Bit(D, W * x + z, Get_Bit(C, W * mod(x - 1, 5) + z) ^ Get_Bit(C,
W * mod(x + 1, 5) + mod(z - 1, W)));
```

```
        }
    }

    for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int z = 0; z < W; z++) {
                Set_Bit(Sc, A(x, y, z), Get_Bit(D, W * x + z) ^ Get_Bit(S, A(x,
y, z)));
            }
        }
    }
```

ρ(rho):

Steps:

1. For all z such that $0 \le z < w$, let $\mathbf{A}'[0, 0, z] = \mathbf{A}[0, 0, z]$.
2. Let $(x, y) = (1, 0)$.
3. For t from 0 to 23:
   a. for all z such that $0 \le z < w$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, (z-(t+1)(t+2)/2) \bmod w]$;
   b. let $(x, y) = (y, (2x+3y) \bmod 5)$.
4. Return $\mathbf{A}'$.

代码（节选）：

```
for (int z = 0; z < W; z++) {
        Set_Bit(Sc, A(0, 0, z), Get_Bit(S, A(0, 0, z)));
    }
    int x = 1, y = 0;
    int newx, newy;
    for (int t = 0; t < 24; t++) {
        for (int z = 0; z < W; z++) {
            Set_Bit(Sc, A(x, y, z), Get_Bit(S, A(x, y, mod(z - ((t + 1) * (t +
2) / 2), W))));
        }
        newx = y%5;
        newy =(2 * x + 3 * y)% 5;
        x = newx;
        y = newy;
    }
```

π(pi):

Steps:

1. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
   $\mathbf{A}'[x, y, z] = \mathbf{A}[(x + 3y) \bmod 5, x, z]$.
2. Return $\mathbf{A}'$.

代码（节选）：

```
for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int z = 0; z < W; z++) {
                j = mod(x + (3 * y), 5);
                Set_Bit(Sc, A(x, y, z), Get_Bit(S, A(j,x,z)));
            }
        }
    }
```

χ(chi):

Steps:

1. For all triples (x, y, z) such that $0\leq x<5$, $0\leq y<5$, and $0\leq z<w$, let
$$\mathbf{A'}[x,y,z] = \mathbf{A}[x,y,z] \oplus ((\mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x+2) \bmod 5, y, z]).$$
2. Return $\mathbf{A'}$.

代码（节选）：

```
for (int x = 0; x < 5; x++) {
        for (int y = 0; y < 5; y++) {
            for (int z = 0; z < W; z++) {
                Set_Bit(Sc, A(x, y, z), Get_Bit(S, A(x, y, z)) ^ ((Get_Bit(S,
A(mod(x + 1, 5), y, z)) ^ 1) * Get_Bit(S, A(mod(x + 2, 5), y, z))));
            }
        }
    }
```

ι(iota):

Steps:

1. For all triples (x, y, z) such that $0\leq x<5$, $0\leq y<5$, and $0\leq z<w$, let $\mathbf{A'}[x,y,z] = \mathbf{A}[x,y,z]$.
2. Let $RC=0^w$.
3. For j from 0 to ℓ, let $RC[2^j-1]=rc(j+7i_r)$.
4. For all z such that $0\leq z<w$, let $\mathbf{A'}[0, 0, z]=\mathbf{A'}[0,0,z] \oplus RC[z]$.
5. Return $\mathbf{A'}$.

代码（节选）：

```
    memcpy(Sc, S, B / 8);
    UCHAR RC[W] = { 0 };
    for (int j = 0; j <= L; j++) {
        int ls = pow(2, j) - 1;
        RC[ls] = rc(j + 7 * Ir);
    }

    for (int z = 0; z < W; z++) {
        Set_Bit(Sc, A(0, 0, z), Get_Bit(Sc, A(0, 0, z)) ^ RC[z]);
    }
```

rc:

Steps:

1. If t mod 255 = 0, return 1.
2. Let R = 10000000.
3. For i from 1 to t mod 255, let:
   a. $R = 0 \parallel R$;
   b. $R[0] = R[0] \oplus R[8]$;
   c. $R[4] = R[4] \oplus R[8]$;
   d. $R[5] = R[5] \oplus R[8]$;
   e. $R[6] = R[6] \oplus R[8]$;
   f. $R = \mathrm{Trunc}_8[R]$.
4. Return $R[0]$.

代码（节选）：

```
UCHAR r[9] = { 1,0,0,0,0,0,0,0,0 };
if (mod(t, 255) == 0) {
    return 1;
}
for (int i = 1; i <= mod(t, 255); i++) {
    memcpy(&r[1], &r[0], 8);
    r[0] = 0;
    r[0] = r[0] ^ r[8];
    r[4] = r[4] ^ r[8];
    r[5] = r[5] ^ r[8];
    r[6] = r[6] ^ r[8];
}
return r[0];
```

SHA3官方步骤（选读）：

1.

$$\text{SHA3-224}(M) = \text{KECCAK}[448]\,(M \parallel 01, 224);$$
$$\text{SHA3-256}(M) = \text{KECCAK}[512]\,(M \parallel 01, 256);$$
$$\text{SHA3-384}(M) = \text{KECCAK}[768]\,(M \parallel 01, 384);$$
$$\text{SHA3-512}(M) = \text{KECCAK}[1024]\,(M \parallel 01, 512).$$

2.

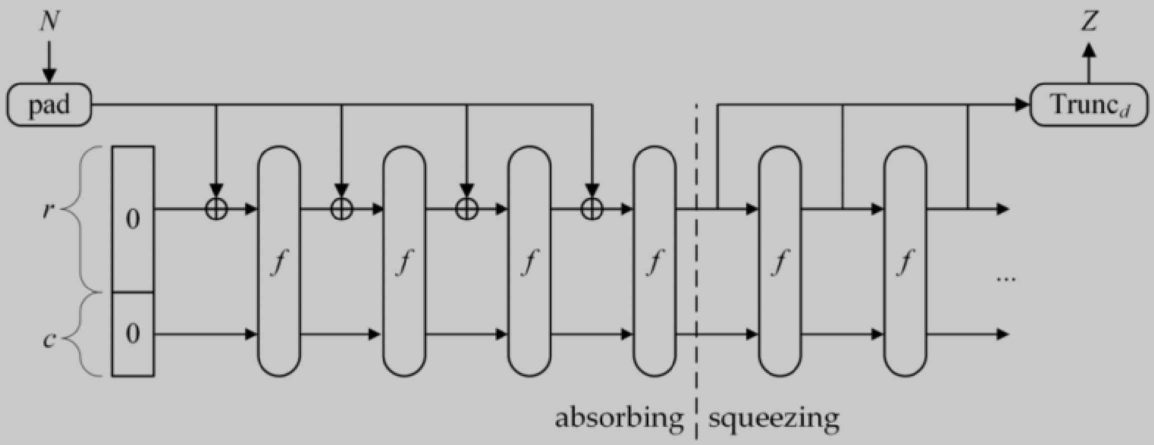


Figure 7: The sponge construction: Z=SPONGE[f, pad, r](N, d) [4]

$$\text{KECCAK}[c]\ (N, d) = \text{SPONGE}[\text{KECCAK-}p[1600, 24],\ \text{pad10*1},\ 1600-c]\ (N, d).$$

Steps:

1. Let $P=N \,||\, \text{pad}(r, \text{len}(N))$.
2. Let $n=\text{len}(P)/r$.
3. Let $c=b-r$.
4. Let $P_0, \ldots, P_{n-1}$ be the unique sequence of strings of length r such that $P = P_0 \,||\, \ldots \,||\, P_{n-1}$.
5. Let $S=0^b$.
6. For i from 0 to $n-1$, let $S=f(S \oplus (P_i \,||\, 0^c))$.
7. Let Z be the empty string.
8. Let $Z=Z \,||\, \text{Trunc}_r(S)$.
9. If $d \leq |Z|$, then return $\text{Trunc}_d(Z)$; else continue.
10. Let $S=f(S)$, and continue with Step 8.

3.

Algorithm 7: KECCAK-$p[b, n_r](S)$

Input:
string S of length b;
number of rounds n_r.

Output:
string S' of length b.

Steps:

1. Convert S into a state array, $\mathbf{A}$, as described in Sec. 3.1.2.
2. For i_r from $12+2\ell-n_r$ to $12+2\ell-1$, let $\mathbf{A}=\text{Rnd}(\mathbf{A}, i_r)$.
3. Convert $\mathbf{A}$ into a string S' of length b, as described in Sec. 3.1.3.
4. Return S'.

$$\text{Rnd}(\mathbf{A}, i_r) = \iota(\chi(\pi(\rho(\theta(\mathbf{A})))), i_r).$$