Hash functions Based on CBC (Cipher Block Chaining)

↳ Many such proposals have been made.

↳ One of them was by Rabin et al.

↳ Divide a message M into fixed-size blocks $M_1, M_2 \cdots M_N$

↳ Use the symmetric encryption system (e.g. DES) to compute the hash code G, as

$$H_0 = \text{initial value}$$

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

$$G = H_N$$

↳ This technique is similar to CBC but no secret key

CBC =

$$C_j = E(\underset{\text{symmetric Key}}{K}, [C_{j-1} \oplus \underset{\text{message Block } j}{M_j}])$$

here, c_0 = IV (Initialization vector).

↳ However the above simple hash function design is vulnerable to birthday attack and meet-in-the-middle attack.

↳ Many modifications have been proposed thereafter

↳ However, all ~~are~~ remain vulnerable.

Secure Hash Algorithm (SHA)

↳ SHA was developed by NIST in 1993. Standard doc. FIPS 180.

History

1993 → SHA-0 } following structure of ~~MD4~~ MD4.

1995 → SHA-1. It produces a hash value of 160 bits. (SHA-160) ≈ MD5

2002 → SHA-2. (SHA-256, SHA-384, SHA-512) RFC 6234

2008 → (SHA-224)

2015 → SHA-3 (SHA-512/224, SHA-512/256). Standard doc. FIPS 180-4

⇓

- Message size $< 2^{128}$
- Block size ~~≤~~ 1024
- Word size 64
- Message Digest Size. 224, 256

SHA-3

↳ SHA-1 has not yet been broken i.e. no one has demonstrated a technique for producing collisions in a practical amount of time.

However, structure of SHA-1 is similar to MD5 and SHA-0 — these are brocken.

Also, SHA-2 shares the same structure.

↳ So, in 2007, NIST declares competition for new design.

↳ selected one is Keccak

The Sponge Construction

↳ It follow itterative hash function design of general structure

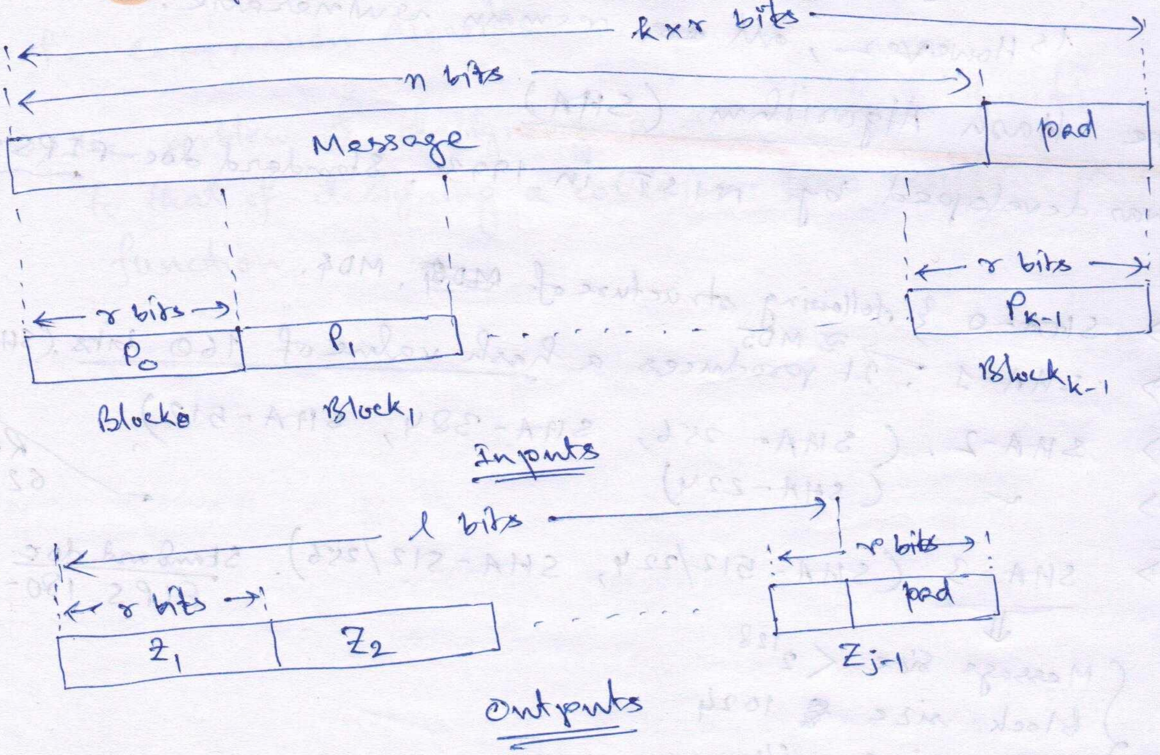
↳ It takes an input message and partitions it into fixed-size blocks.

↳ Each block is processed in turn with each iteration.

↳ So, it allows both variable length input and output
  ↳ so, flexible structure.

Sponge function is defined by three parameters.

1. f: iteration function, used to process each input block
2. r: input block size in bits, also called bitrate.
3. pad: the padding algorithm



Padding:

↳ for uniformity, padding is always added.

so, if $n \bmod r = 0$, a padding block of r bits is added.

↳ The sponge specification proposes two padding schemes:

① Simple padding: denoted by pad10*

↳ appends a single bit 1 followed by minimum but sufficient number of bits 0

② Multirate padding: denoted by pad10*1

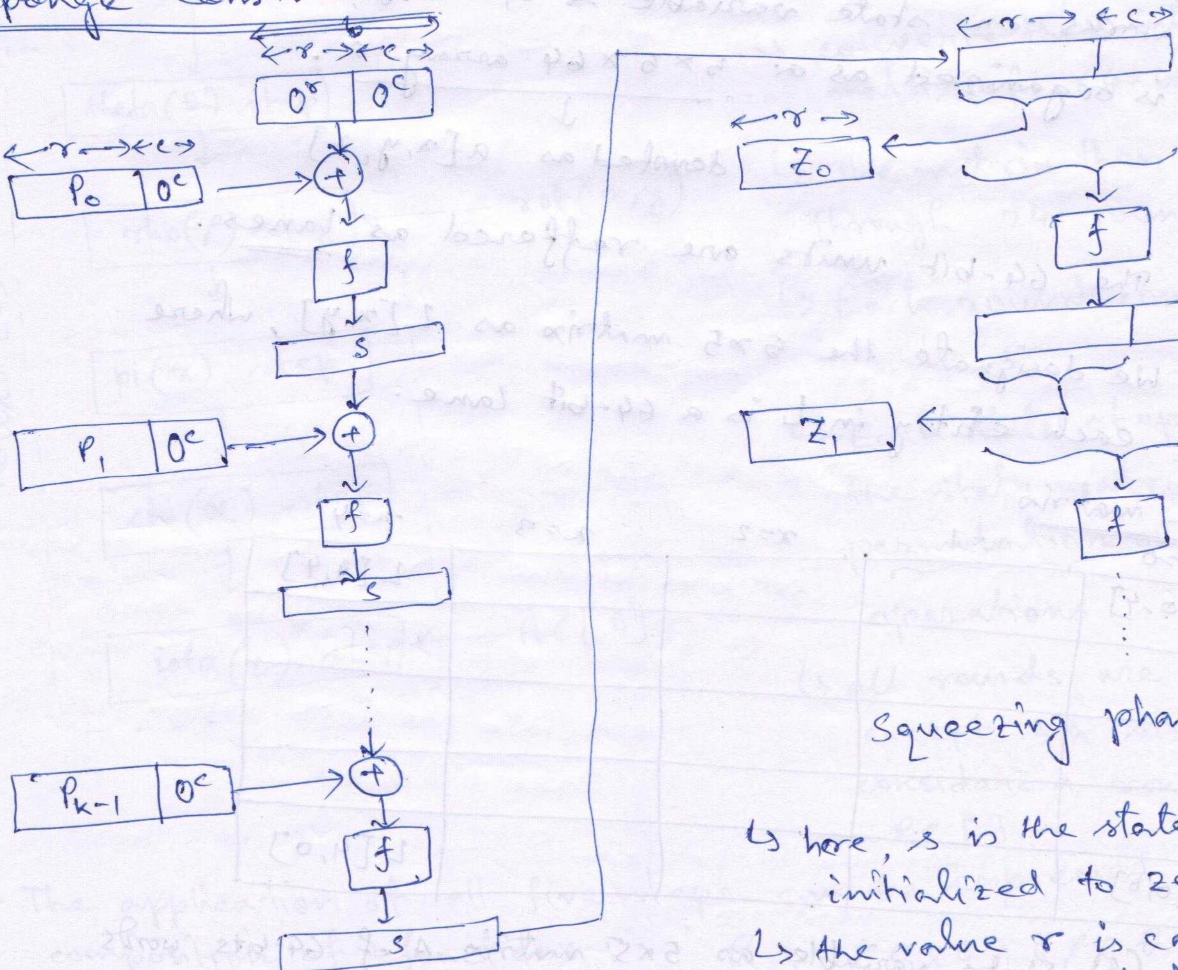
↳ appends a single bit 1 followed by minimum number of bits 0 followed by a single bit 1.

Output:

↳ sponge function generates sequence of output blocks $Z_0, Z_1 \cdots Z_{j-1}$

↳ so, if the desired output bits is l bits, then the value of j must satisfy $(j-1) \times r < l \leq j \times r$.

Sponge Construction



Absorbing phase.

Squeezing phase

↳ here, s is the state variable initialized to zero.

↳ the value r is called bitrate.

↳ the value c is called capacity.

↳ The default values of $c = 1024$ bits
$r = 576$ bits } so, $b = 1600$ bits

↳ If the desired output length l satisfies $l \leq r$, then at the end of absorbing phase, the first l bits of s are retured. (i.e. no squeezing phase is required).

↳ Otherwise, it goes through squeezing phase.

↳ in each iteration, r bits of S are retained as z_i and concatenated with previous z bocks.

↳ This process continues through $(j-1)$ iteration until we have $(j-1) \times r < l \leq j \times r$.

↳ Then, first l bits of concatenated block z are returned.

The SHA-3 Iteration function f

↳ Keccak-$f[r+c]$, $b = r+c$, in general $b = 1600$ bits

bitrate capacity

↳ It works on state variable s of length b bits.

↳ s is organized as a $5 \times 5 \times 64$ array a.

denoted as $a[x, y, z]$

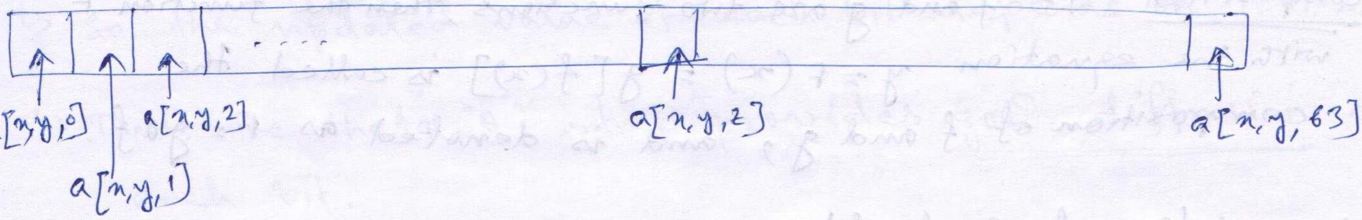
↳ The 64-bit units are reffered as lanes.

↳ We designate the 5×5 matrix as $L[x, y]$, where each entry in L is a 64-bit lane.

L matrix

	x=0	x=1	x=2	x=3	x=4
y=5	L[0,4]				L[4,4]
y=4					
y=3					
y=1					
y=0	L[0,0]				L[4,0]

(a) State variable as 5×5 matrix A of 64-bits words.



(b) Bit labeling of 64-bits words.

So, the mapping between the bits of s and those of a is

$$s[64(5y+x)+z] = a[x,y,z]$$

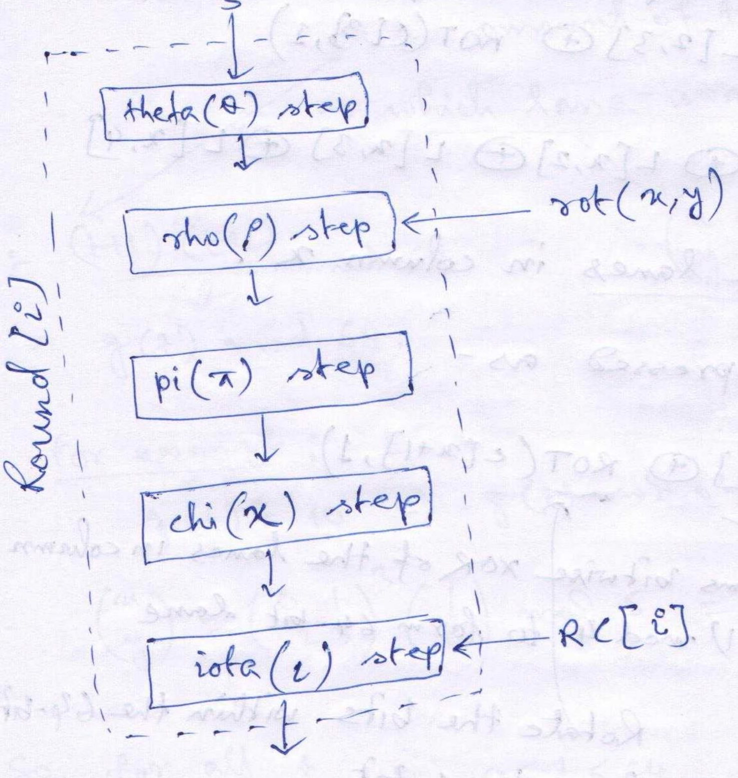
$x = 0,1,\cdots 4$
$y = 0,1,\cdots 4$
$z = 0,1,\cdots 63$

↳ In brief, the first lane in the lower left corner, $L[0,0]$, correspond to the first 64 bits of s.

↳ The lane in second column, lowest row, $L[1,0]$, corresponds to the next 64 bits of s.

↳ and so on.

Structure of f



↳ f takes input 1600-bit s

↳ converts the s into a 5×5 matrix of 64-bit lanes.

↳ The matrix then passes ~~thru~~ through 24 rounds

↳ Each round consists of 5 steps.

↳ Each ~~state~~ step updates the state matrix by permutation or substitution operations.

↳ all rounds are identical except 5th step. which considers a round constant RC[ ].

↳ The application of all five steps can be expressed as the composition of functions : $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

↳ <u>composition</u>: If f and g are two functions, then the function F with the equation $y = F(x) = g[f(x)]$ is called the composition of f and g, and is denoted as $F = g \circ f$.

Description of Each Step

① <u>Theta (θ): Type → Substitution</u>

↳ New value of each bit in each word depends on its current value and on one bit in each word of preceding column and one bit of each word in succeeding column.

$$\theta: a[x,y,z] \leftarrow a[x,y,z] \oplus \sum_{y'=0}^{4} a[(x-1), y', z] \oplus \sum_{y'=0}^{4} a[(x+1), y', (z-1)]$$

	x=0	x=1	x=2	x=3	x=4
y=4		L[1,4]		L[3,4]	
y=3		L[1,3]	L[2,3]	L[3,3]	
y=2		L[1,2]		L[3,2]	
y=1		L[1,1]		L[3,1]	
y=0		L[1,0]		L[3,0]	

e.g. ⇒ $L[2,3] \leftarrow C[1] \oplus L[2,3] \oplus ROT(C[3], 1)$

$C[x] = L[x,0] \oplus L[x,1] \oplus L[x,2] \oplus L[x,3] \oplus L[x,4]$

⇗ This is bitwise XOR of the lanes in column x.

<u>Operation on $L[x,y]$ is expressed as</u> -

$$L[x,y] \leftarrow L[x,y] \oplus \underbrace{C[x-1]}_{\text{performs bitwise XOR of the lanes in column } (x-1) \bmod 4 \text{ to form 64-bit lane}} \oplus ROT(C[x+1], 1)$$

Rotate the bits within the 64-bit lane by 1 bit.

↳ The same operation is performed on all of the other lanes in the matrix.

↳ So, the updated value of each bit depends on 11 bits.

↳ Thus, the theta (θ) step provides good diffusion on each bit.

(2) **Rho (ρ)** : Type → Permutation ↳ The bits of each word are permuted using a circular bit shift. W[0,0] is not affected.

The ρ function is defined as follows:

$$\rho : a[x,y,z] \leftarrow a[x,y,z] \text{ if } x=y=0$$

otherwise,

$$\rho : a[x,y,z] \leftarrow a\left[x, y, \left(z - \frac{(t+1)(t+2)}{2}\right)\right]$$

with t satisfying $0 \leq t < 24$ and $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ in $GF(5)^{2\times 2}$

Meaning

↳ Lane L[0,0] is unaffected.

↳ For all other words, a circular bit shift within the lane is performed.

↳ The variable t, $0 \leq t < 24$, is used to determine both the circular bit shift, and

→ the amount of the circular bit shift

→ which lane is assigned which shift value.

$$= \frac{(t+1)(t+2)}{2} \bmod 64 \qquad \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} \bmod 5$$

$= g(t) \bmod 64.$

For example: $t = 3$

$g(t) = 10 \Rightarrow g(t) \bmod 64 = 10$ ← amount of bit shift

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^3 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \bmod 5 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

← Lane L[1,2]

So, for all t i.e. $0 \leq t < 24$, we will actually get 24 different rotation amount and all 24 lanes, excluding the L[0,0].

Thus, rho (ρ) function consists of a simple permutation (circular shift) within each lane. The intent is to provide diffusion within each lane.

③ **Pi (π) : Type → Permutation**

↳ Words are permuted in the 5×5 matrix.

w[0,0] is not affected.

The π function is defined as

$$\pi : a[x,y] \leftarrow a[x',y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

i.e. $[x,y] \leftarrow [y, (2x+3y)]$

Thus the lanes within the 5×5 matrix are moved so that

⇒ new x position equals the old y position

⇒ new y position is determined by $(2x+3y) \bmod 5$.

For example: Let lane L[2,3]

Now modified [x,y] are [3,3]. This is new position.

So, the word Z[2,3] corresponding to L[2,3] will now move to lane L[3,3].

Lane position after permutation

	x=0	x=1	x=2	x=3	x=4
y=4	Z[2,0]	Z[3,1]	Z[4,2]	Z[0,3]	Z[1,4]
y=3	Z[4,0]	Z[0,1]	Z[1,2]	Z[2,3]	Z[3,4]
y=2	Z[1,0]	Z[2,1]	Z[3,2]	Z[4,3]	Z[0,4]
y=1	Z[3,0]	Z[4,1]	Z[0,2]	Z[1,3]	Z[2,4]
y=0	Z[0,0]	Z[1,1]	Z[2,2]	Z[3,3]	Z[4,4]

Thus, the π step is a permutation of lanes within the matrix.

the ρ step was a permutation of bits within a lane.

④ **Chi (χ)** : Type → Substitution

↳ New value of each bit in each word depends on its current value and on one bit in next word in the same row and one bit in the second next word in the same row.

The χ function is defined as

$$\chi : a[x] \leftarrow a[x] \oplus ((a[x+1] \oplus 1) \text{ AND } a[x+2])$$

i.e. $$a[x,y,z] \leftarrow a[x,y,y] \oplus (\text{NOT}(a[x+1,y,z]) \text{ AND } a[x+2,y,z])$$

	x=0	x=1	x=2	x=3	x=4
y=4					
y=3			L[2,3]	L[3,3]	L[4,3]
y=2					
y=1					
y=0					

e.g. $L[2,3] \leftarrow L[2,3] \oplus \underbrace{L[3,3]}$ AND $L[4,3]$

see NOT operation

↳ This is the only one of the step functions that is a non-linear mapping.

⑤ **Iota (ι)** : Type → Substitution

↳ W[0,0] is updated by XOR with a round constant.

The iota function is defined as –

$$\iota : a \leftarrow a \oplus RC[i_r]$$

ith round.

more precisely, $L[0,0] \leftarrow L[0,0] \oplus RC[i_r]$, $0 \leq i_r \leq 24$

as round constant (RC) is applied only ~~on~~ to the first lane of the internal state array.

↳ Iota function breaks up any symmetry induced by the other four steps functions.

↳ Note that disruption applied on first lane diffuses through θ and χ to all lanes of the state after a single round.

↳ SHA-3 defines a RC table for all 24 rounds.

see Chapter 11 in Stallings book for the table.