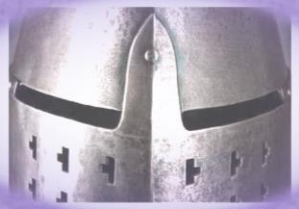


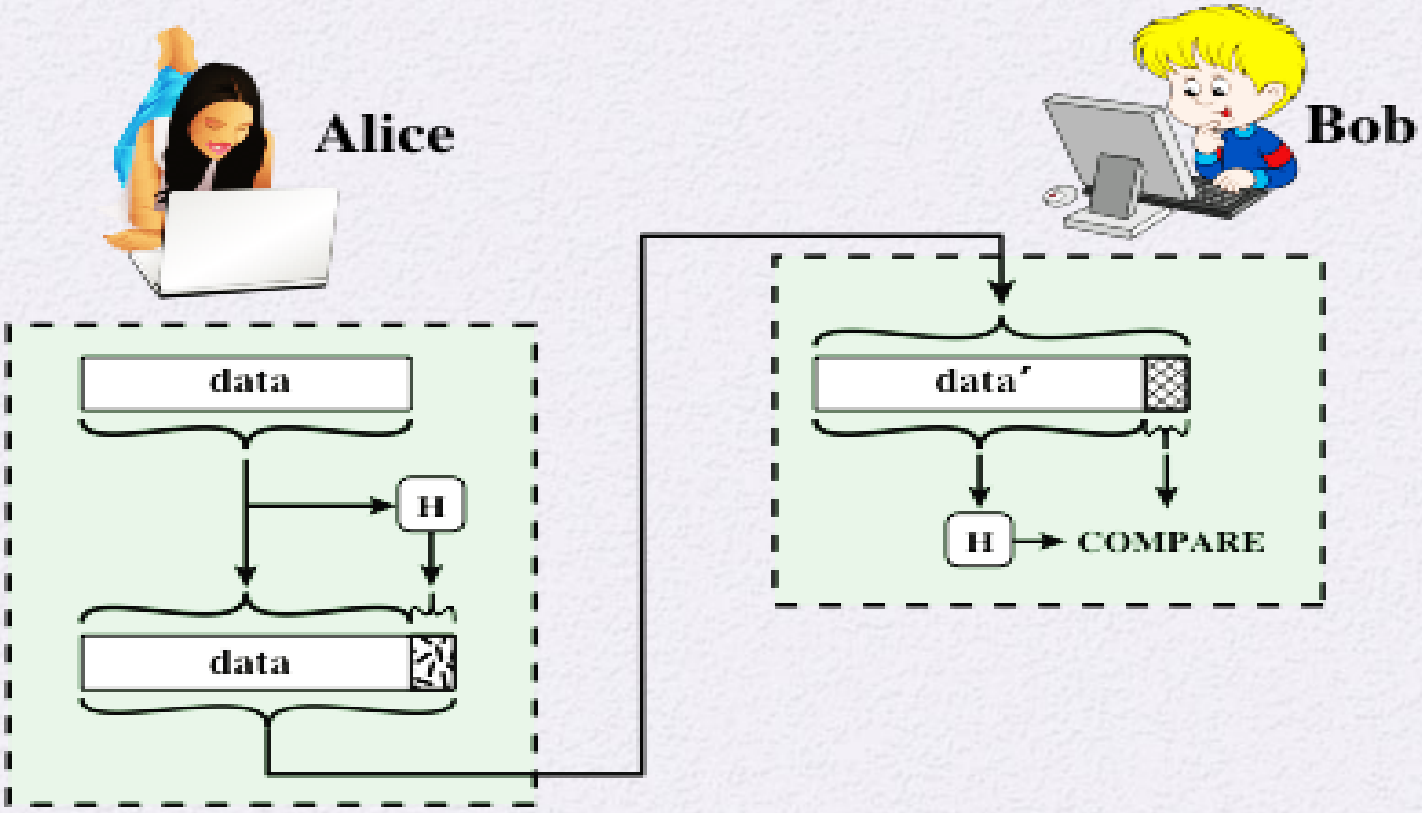
Chapter 11

Cryptographic Hash Functions

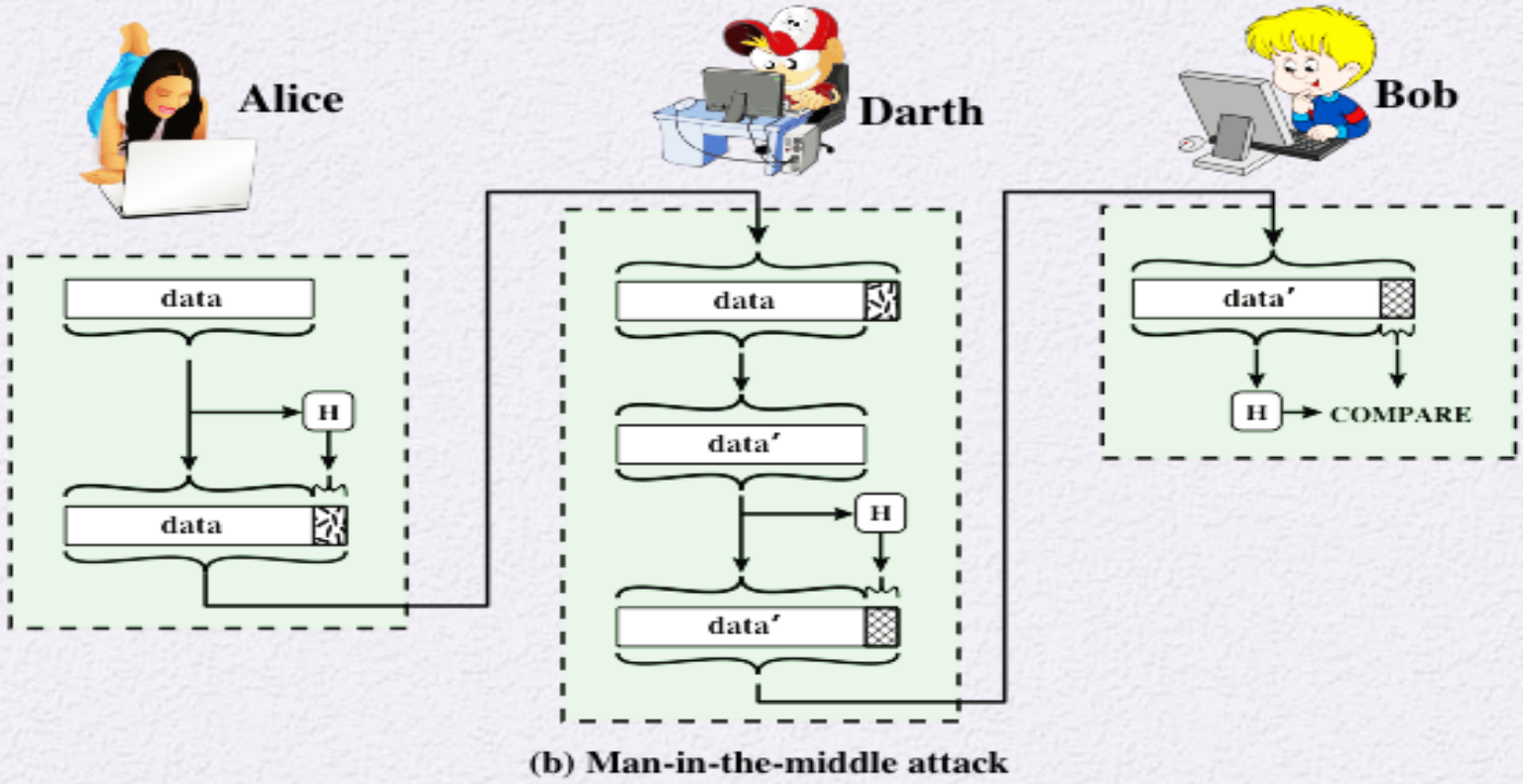
Hash Functions

- Purpose: data integrity
- No "secret key".
- Input: a variable-length block of data *M*
- Output: a fixed-size hash value h = H(*M*)
- Cryptographic hash function H
  - It is computationally infeasible to:
    - given h, find M such that H(M) = h
      - one-way property
    - find M and M' such that H(M) = H(M')
      - collision-resistant property

Naive use for data integrity



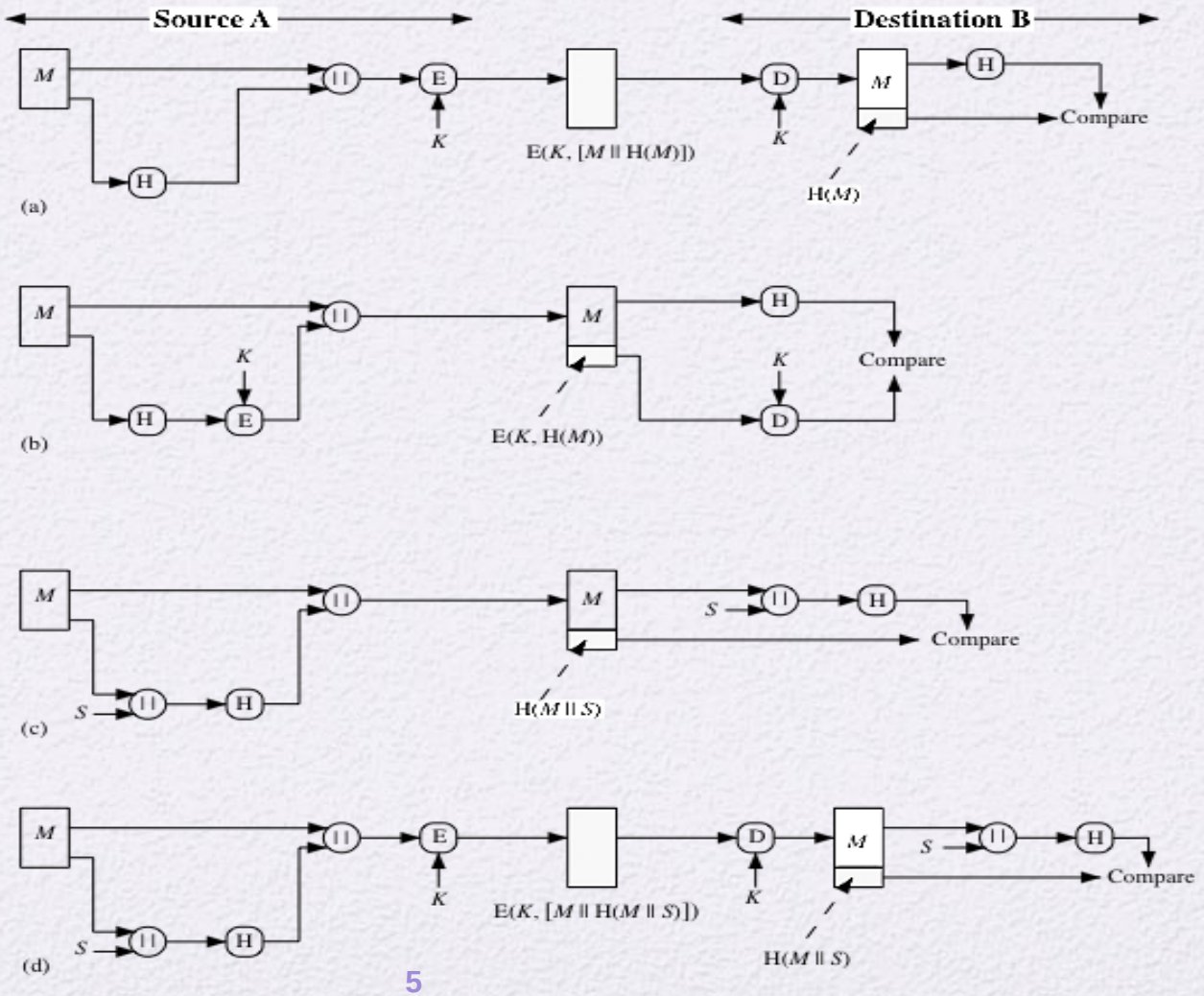
Man-in-the-Middle attack



(b) Man-in-the-middle attack

For Message Authentication

Hash function with a shared secret key K



Message Authentication Code (MAC)

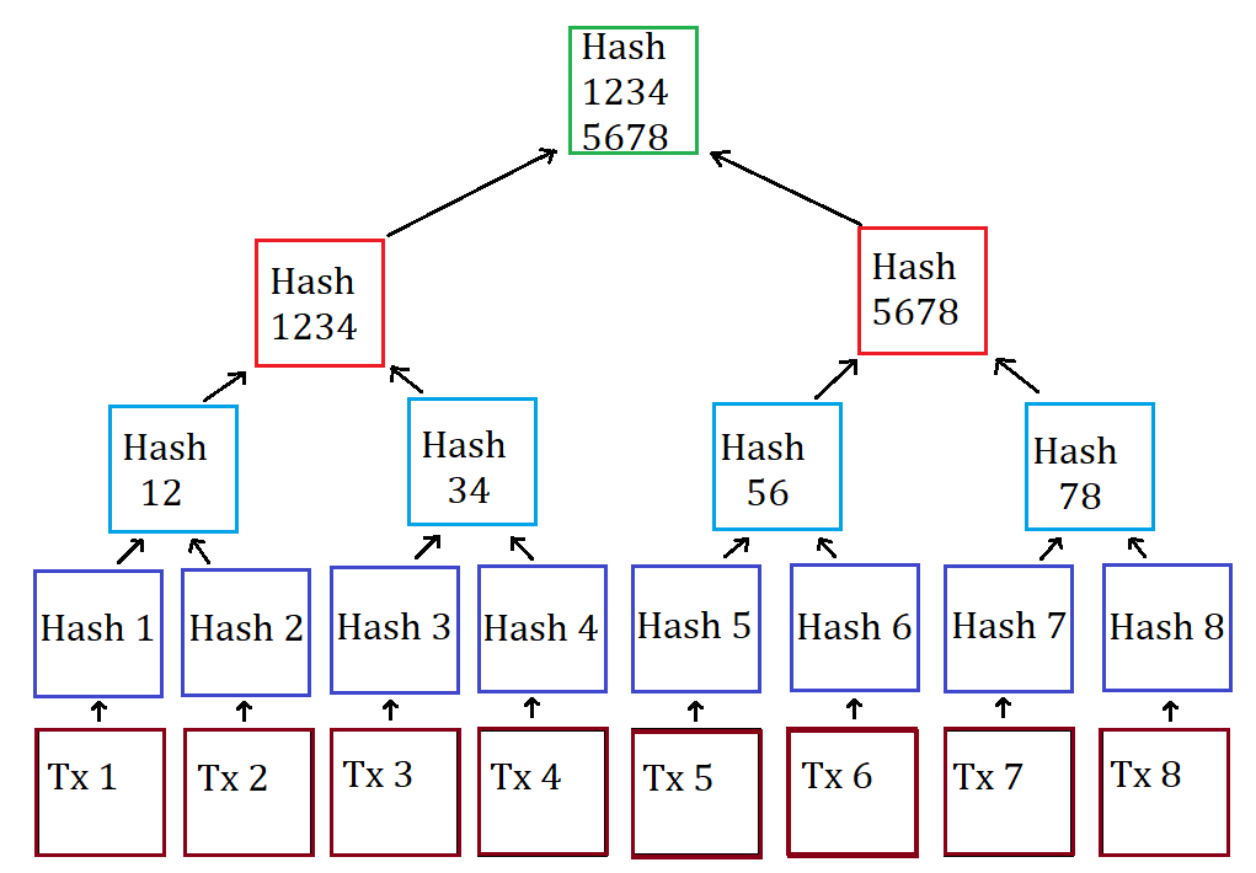
- A *keyed hash function*
- Two parties who share a secret key *k* authenticate the exchanged information
- MAC(M, k)= H(k||M)

Other Hash Function Use

- To create a one-way password file
  - Server side: (Alice, $H(PW_{alice})$), (Bob, $H(PW_{Bob})$), …
  - System intrusion ➔ the passwords of users are not disclosed
- To detect intrusion and virus
  - (F1, H(F1)), (F2, H(F2)), …
- To construct PRF and PRNG
  - $F_s(x) = H(s||x)$

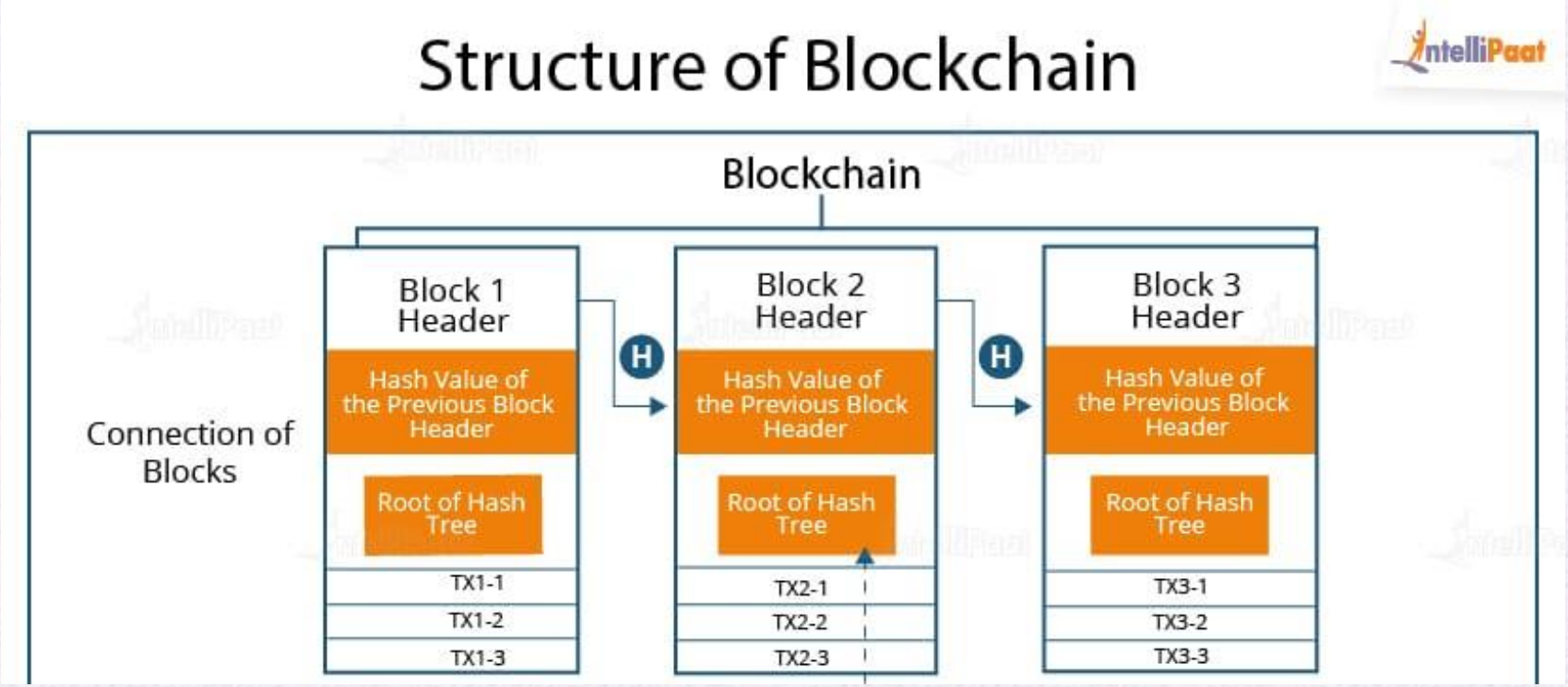
Merkle (hash) tree

- To provide data integrity flexibly and efficiently

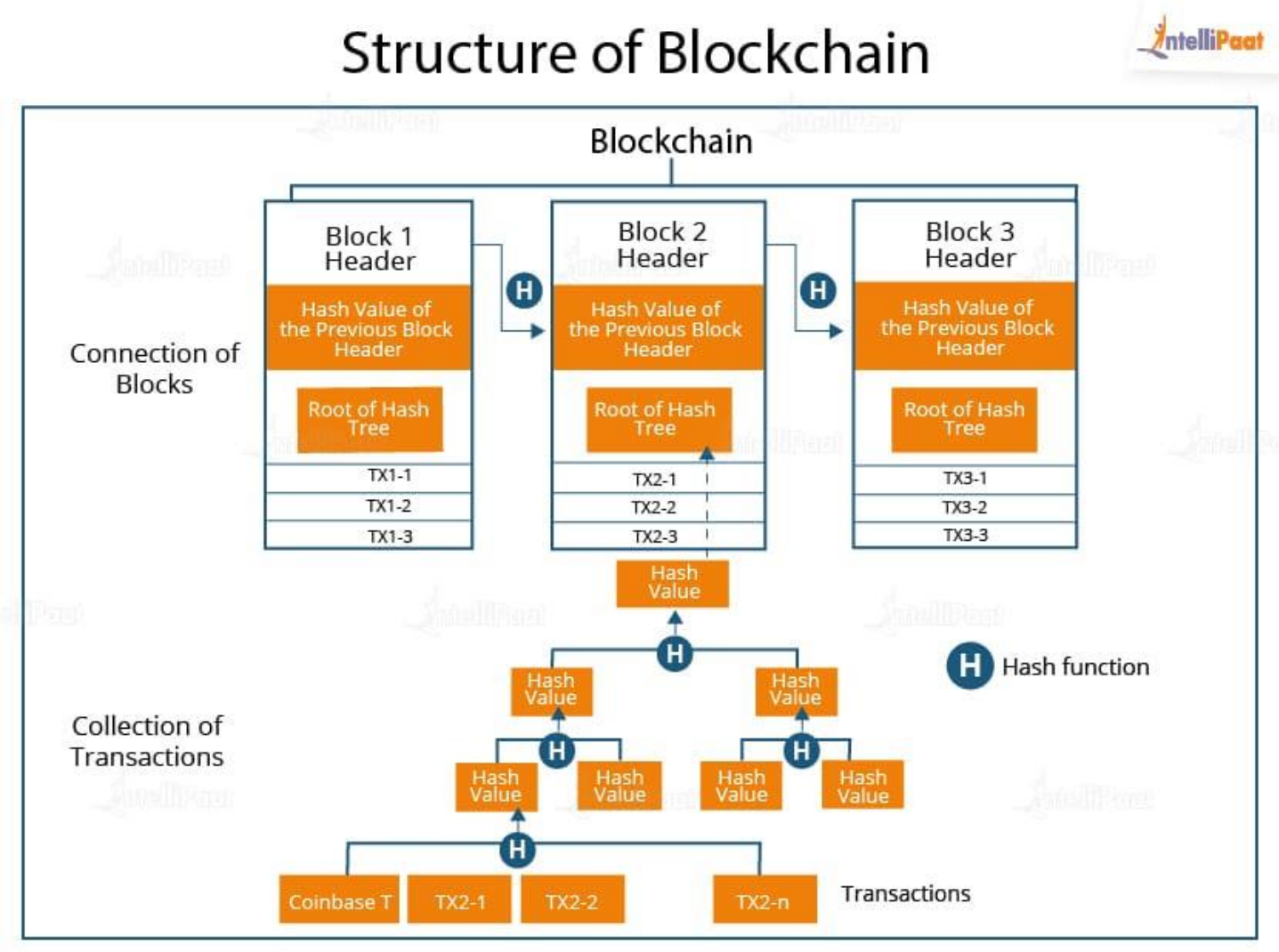


Blockchain

- To provide data integrity in temporal sequence



Combination



Security Requirements

- ***Def:*** *x* is the preimage of value *h if* *h* == H(*x*)
- Problem: given h, find x such that H(x) == h.

Preimage resistant:

- H is a many-to-one mapping. For any given hash value *h, the number of* preimages is infinite.
- But, it is computational hard to find any preimage.

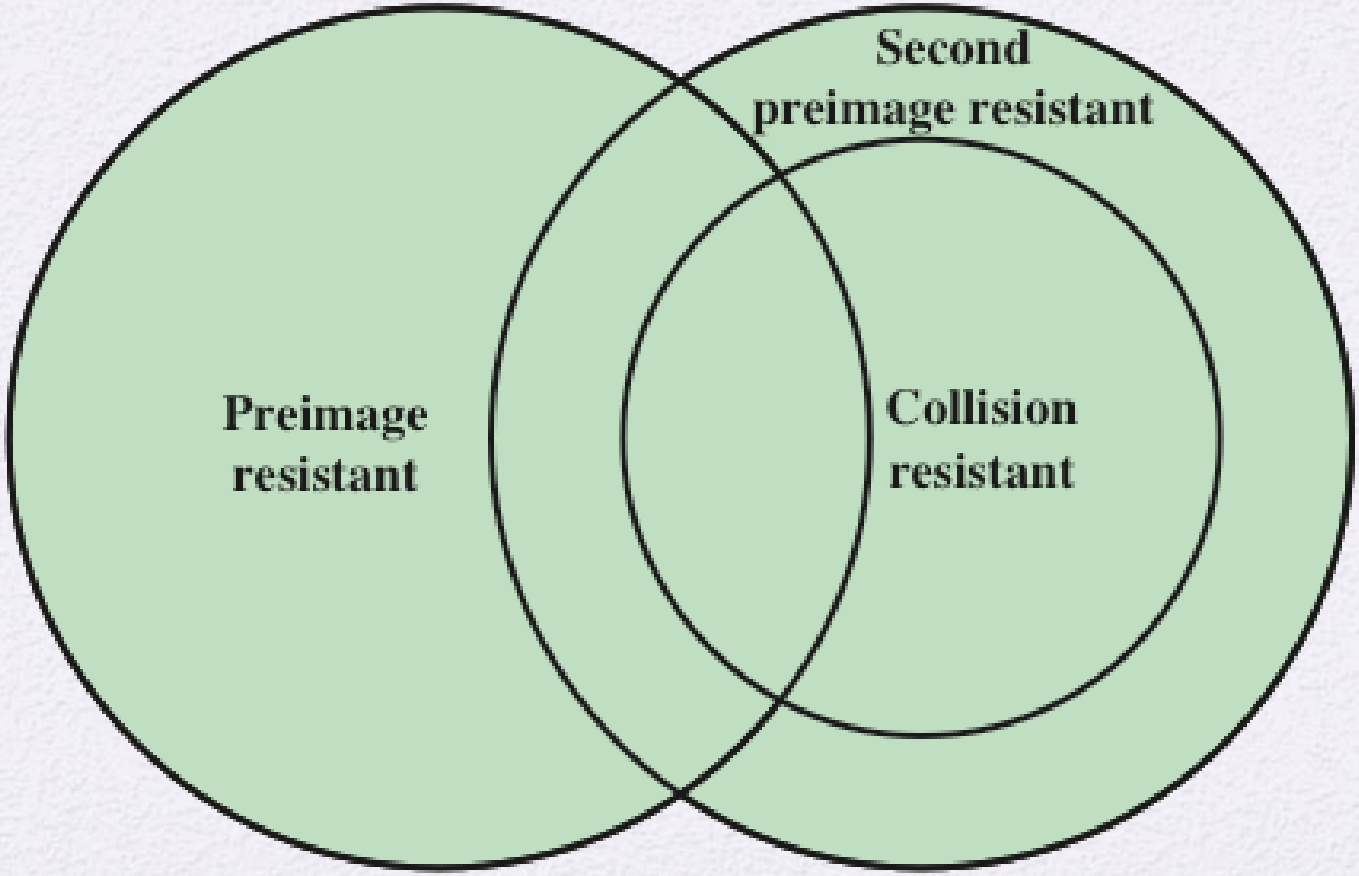
Security Requirements

- Weak Collision resistance:
  - Given x and H, it is hard to find y such H(y)=H(x)
- Strong collision resistance:
  - Given H, it is hard to find x and y such that H(x)=H(y)

Requirements for CHFs

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness

Relationship among properties



- Preimage resistance does not imply collision resistance
  - Assume that H is pre-image-resistant.
  - Let H'(xb) = H(x).
  - Then H' is also preimage-resistant, but not collision-resistant.

Why?

- Collision resistance does not imply preimage resistance
  - Assume that h is collision-resistant with 128-bit output
  - Let $H'(x) = \begin{cases} x\|0 & if\ |x| = 128 \\ H(x)\|1 & otherwise \end{cases}$
  - Then, H' is also collision-resistant, but not preimage-resistant.

Why?

Various Data Integrity Applications

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

Attacks on Hash Functions

Brute-Force Attacks

- Does not depend on the specific algorithm, only depends on length of hash values
- To pick values at random and try each one until a collision occurs

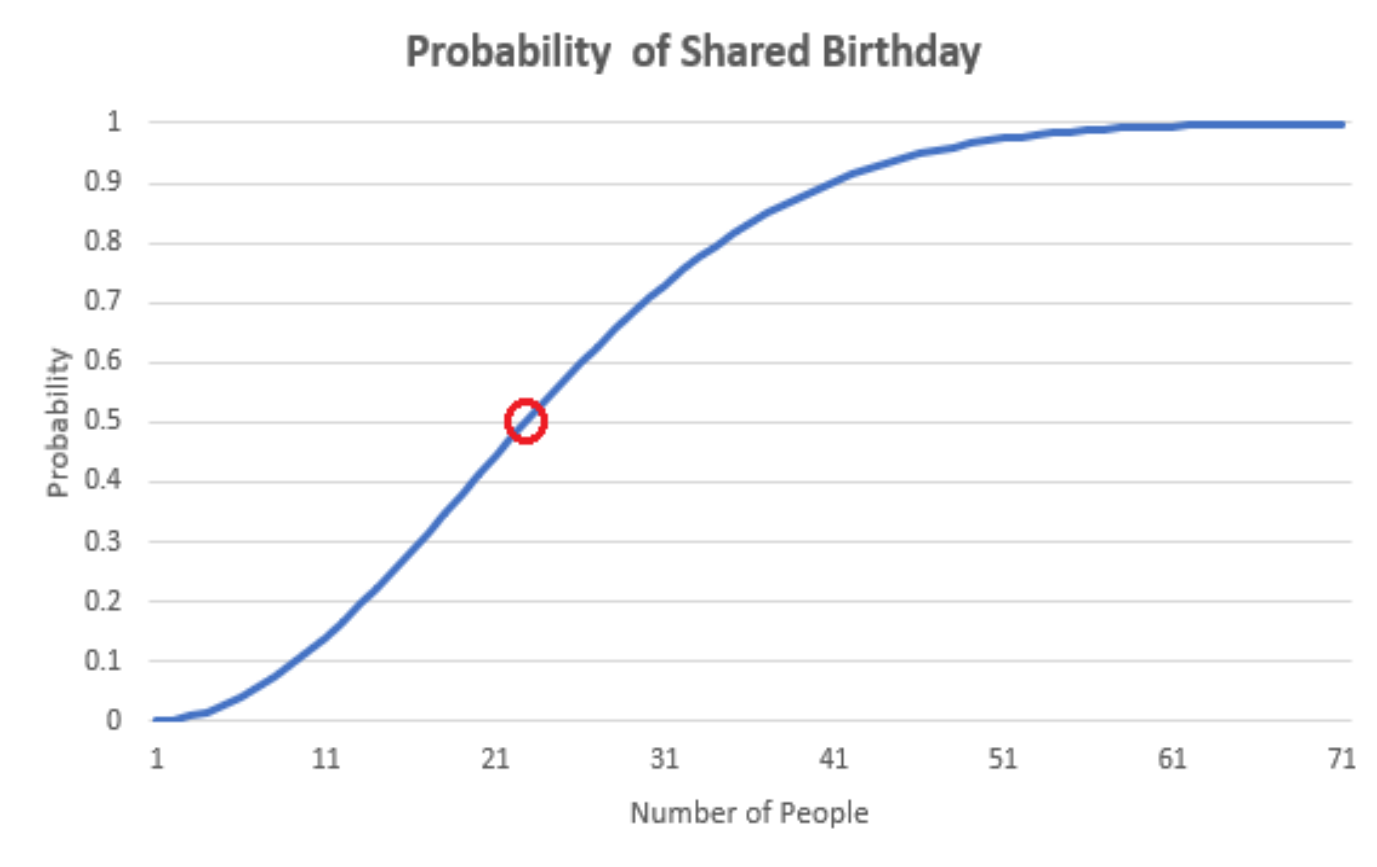
Cryptanalysis

- Seek to exploit some property of the algorithm to perform some attack other than an exhaustive search

Birthday Problem

- H: X → {1, 2, …, 365}
  - Choose $x_1, x_2, \ldots, x_r$ randomly and compute $H(x_i)=y_i$
  - What is $p=Pr[y_i=y_j$ for some $i\neq j]$ for $2 \leq r \leq 365$?
    - $1-(365/365)(364/365)(363/365)\ldots((365-r+1)/365)$
  - What r makes $p \geq 1/2$?
    - $\cong 1.18\sqrt{365} \cong 23$

Birthday Problem



General Birthday Problem

- For a set S of N different elements, r elements $x_1, x_2, \ldots, x_r$ are randomly chosen one by one and with replacement, what r makes the probability of collision greater than 0.5

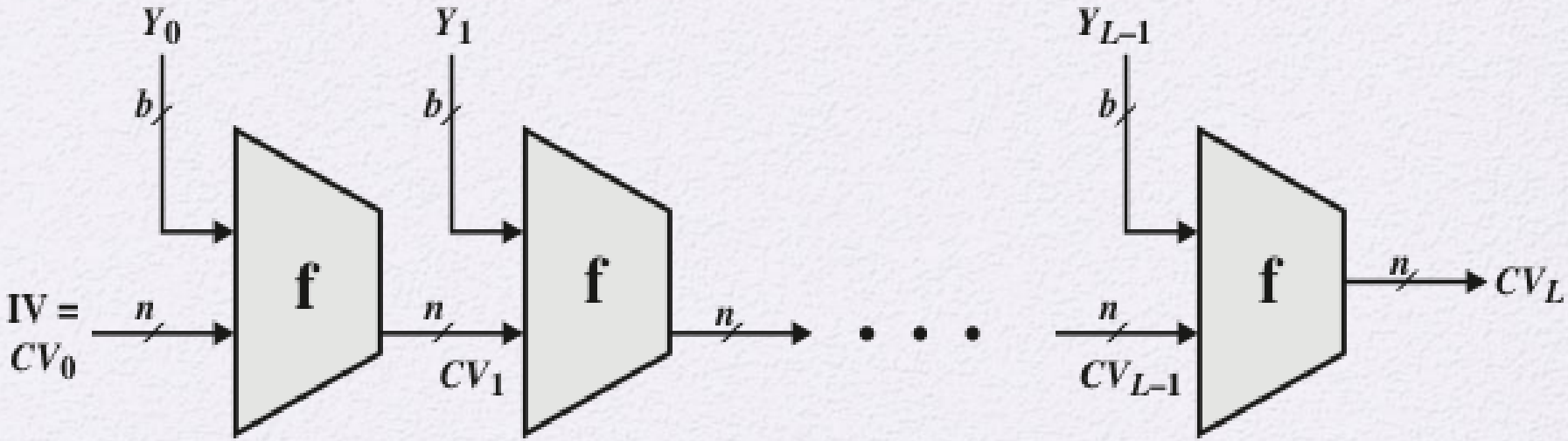
$$\text{Prob}\left(x_i = x_j \text{ for some } i \neq j\right]$$
$$= 1 - 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdots \left(1 - \frac{r-1}{N}\right)$$
$$\geq 1 - e^{\frac{-1}{N}} e^{\frac{-2}{N}} \cdots e^{\frac{-(r-1)}{N}} \quad (\because \ e^{-x} \geq 1 - x \ \text{ for all } \ x)$$
$$= 1 - e^{\frac{r(r-1)}{2N}}$$

- $r \cong 1.18\sqrt{N}$

Birthday Paradox

- Relation to the collision resistant property
  - The hash value is m-bit long
  - It takes about $2^{m/2}$ random tries to find a collision with high probability
  - Therefore, m must be at least 160-bit long
    - Sha256 has 256-bit outputs

Secure Hashing: General Structure



IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm

L = number of input blocks
n = length of hash code
b = length of input block

Hash: use Encryption-CBC

- $M=M_1M_2 \ldots M_N$
- Use a symmetric encryption, such as AES, to compute the hash code G as

  H_o= initial value

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

  $G = H_N$

  - Similar to the CBC technique, but there is no secret key here.

Man-in-the-Middle Attack

- Given a hash value G of some message, eg. "Hello"
  - Choose the desired message $M = Q_1 || Q_2 || \ldots || Q_{N-2}$, such as "I owe you NT$1000, John 12/25/2020"
  - Compute $H_i = E(Q_i, H_{i-1})$, $1 \le i \le N-2$
  - Generate $2^{m/2}$ random blocks X_i and compute $R_i = E(X_i, H_{N-2})$
  - Generate $2^{m/2}$ random blocks Y_j and compute $S_j = D(Y_j, G)$
  - If $R_i = S_j$ for some $1 \le i, j \le 2^{m/2}$, then $Q_1 Q_2 \ldots Q_{N-2} X_i Y_j$ is a message for the hash value G.

Man-in-the-Middle Attack

- Analysis
  - The number of pairs for finding a collision
    - $2^{m/2} x 2^{m/2}$
    - By the birthday paradox, we only need $1.18 x 2^{m/2}$ pairs in order to find a collision with probability > 0.5

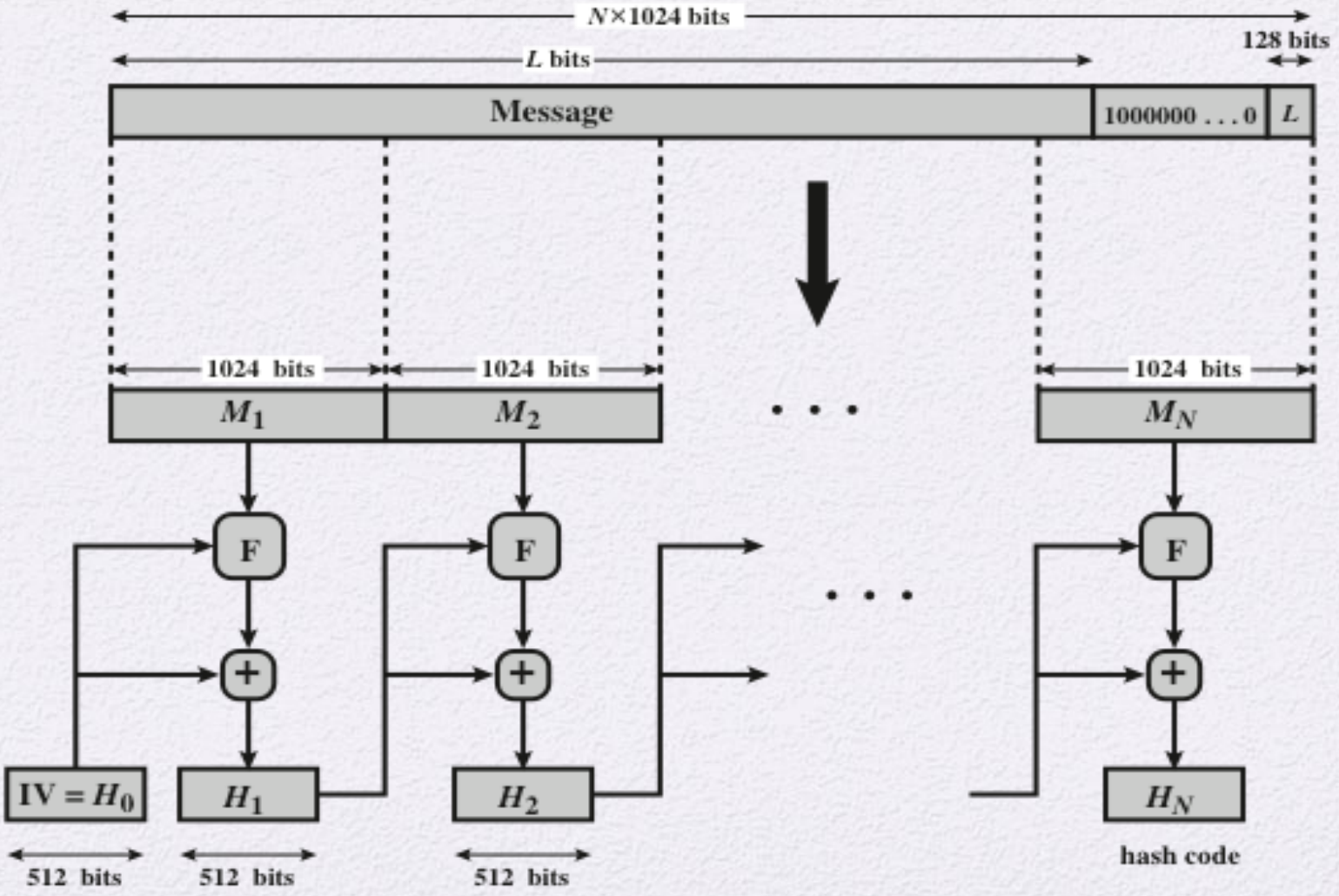
Secure Hash Algorithm (SHA)

- SHA: NIST, FIPS 180, 1993
- SHA-1: 1995
  - Collisions are found for SHA-0 and SHA-1
- SHA-2: {SHA-256, SHA-384, SHA-512}, 2002
  - Secure and widely used
- ShA-3: 2012
  - Derived from Keccak with different structure from Sha-0, sha-1 and sha-2
  - Recommended: SHA3-224, SHA3-256, SHA3-384, SHA3-512

SHA Parameters

Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

SHA-512: structure



The padded message consists blocks $M_1, M_2, \ldots M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1} \ldots M_{i,15}$. All addition is performed modulo 2^{64}.

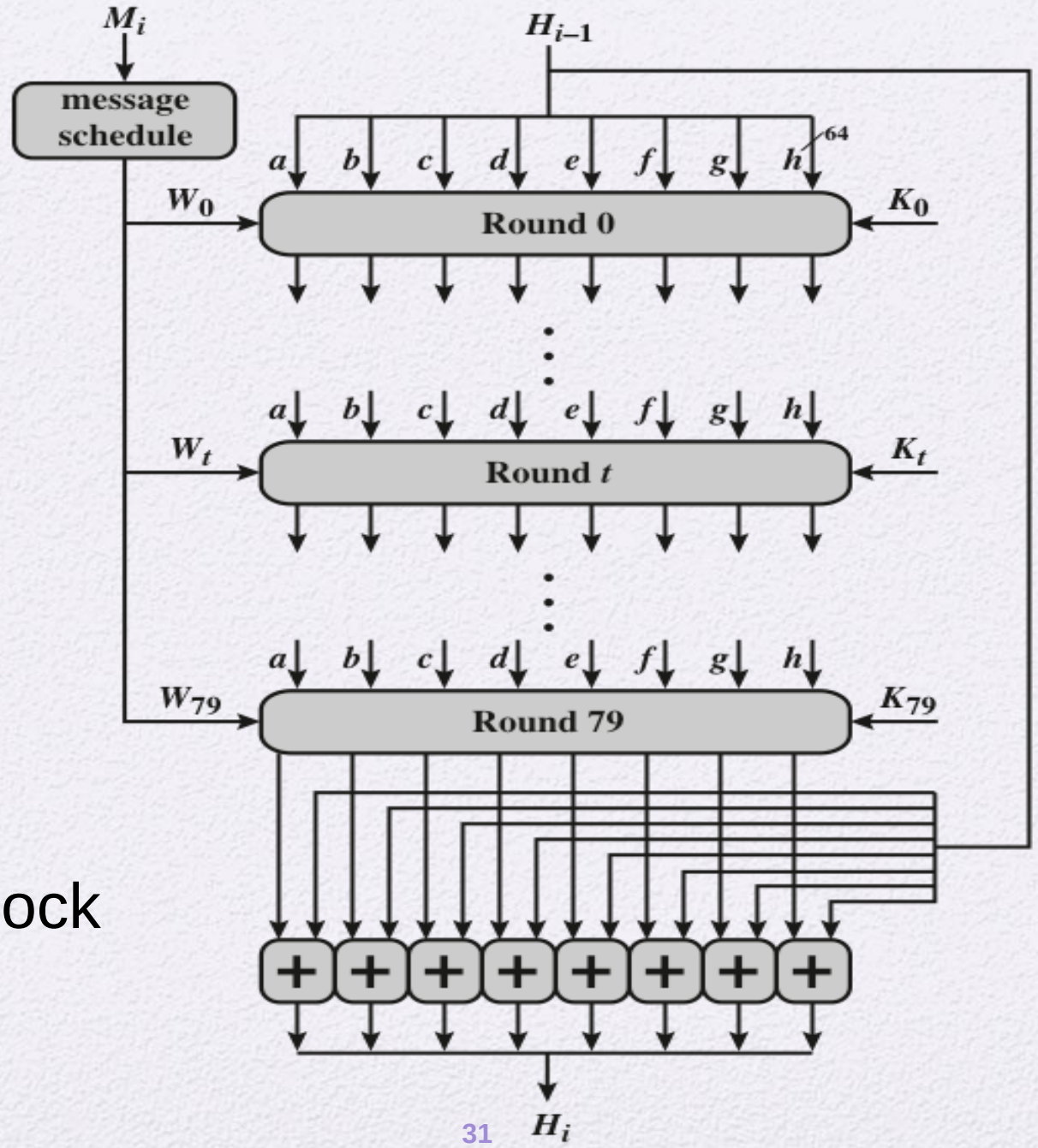
$H_{0,0} = 6A09E667F3BCC908$ $H_{0,4} = 510E527FADE682D1$
$H_{0,1} = BB67AE8584CAA73B$ $H_{0,5} = 9B05688C2B3E6C1F$
$H_{0,2} = 3C6EF372FE94F82B$ $H_{0,6} = 1F83D9ABFB41BD6B$
$H_{0,3} = A54FF53A5F1D36F1$ $H_{0,7} = 5BE0CDI9137E2179$

for $i = 1$ **to** N

1. Prepare the message schedule W:
   for $t = 0$ **to** 15
   $W_t = M_{i,t}$
   for t = 16 **to** 79
   $W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$
2. Initialize the working variables
   $a = H_{i-1,0}$ $e = H_{i-1,4}$
   $b = H_{i-1,1}$ $f = H_{i-1,5}$
   $c = H_{i-1,2}$ $g = H_{i-1,6}$
   $d = H_{i-1,3}$ $h = H_{i-1,7}$
3. Perform the main hash computation
   for $t = 0$ **to** 79
   $T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$
   $T_2 = \left(\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c)$
   $h = g$
   $g = f$
   $f = e$
   $e = d + T_1$
   $d = c$
   $c = b$
   $b = a$
   $a = T_1 + T_2$
4. Compute the intermediate hash value
   $H_{i,0} = a + H_{i-1,0}$ $H_{i,4} = e + H_{i-1,4}$
   $H_{i,1} = b + H_{i-1,1}$ $H_{i,5} = f + H_{i-1,5}$
   $H_{i,2} = c + H_{i-1,2}$ $H_{i,6} = g + H_{i-1,6}$
   $H_{i,3} = d + H_{i-1,3}$ $H_{i,7} = h + H_{i-1,7}$

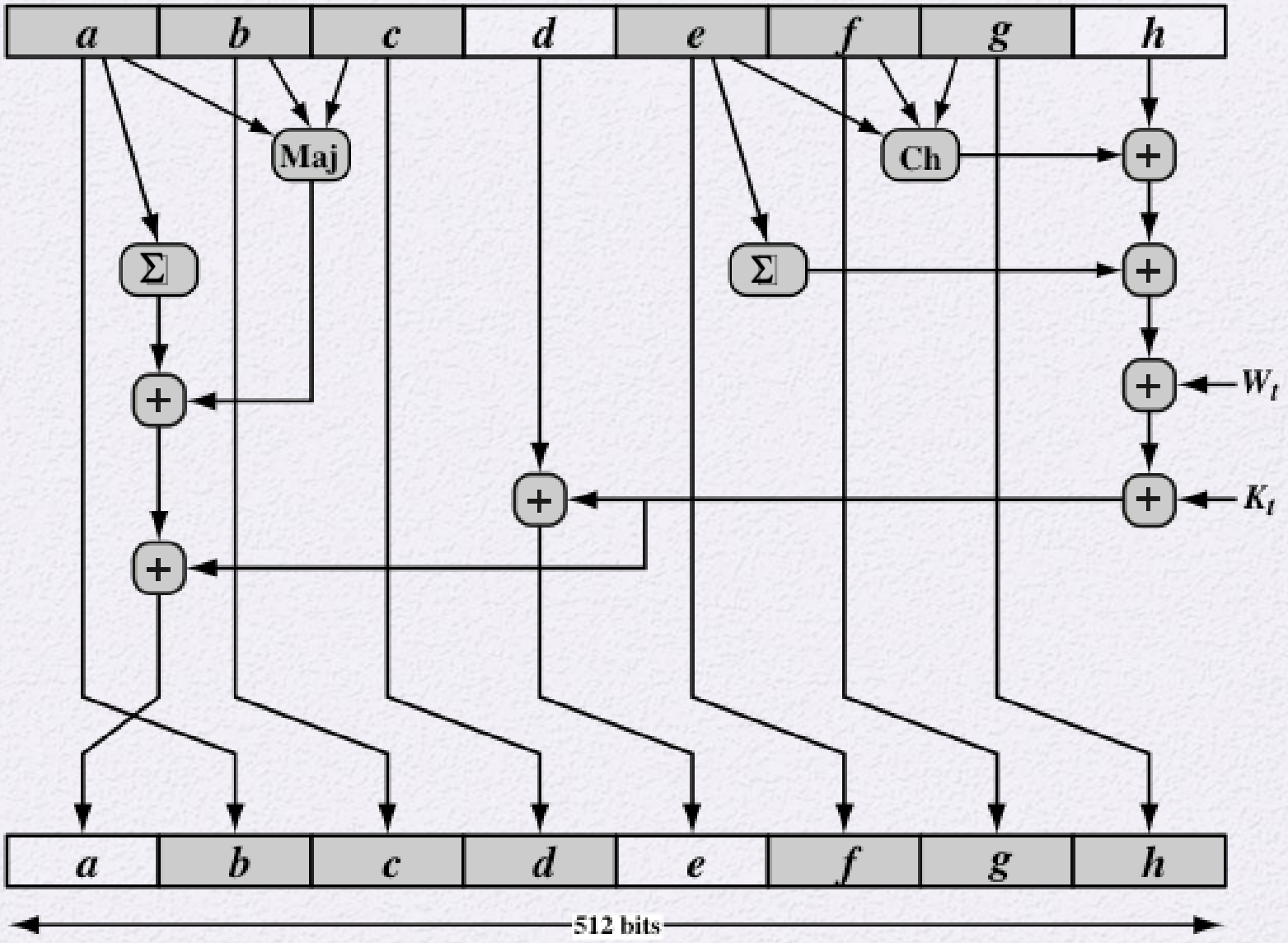
return $\{H_{N,0} \| H_{N,1} \| H_{N,2} \| H_{N,3} \| H_{N,4} \| H_{N,5} \| H_{N,6} \| H_{N,7}\}$

Single block

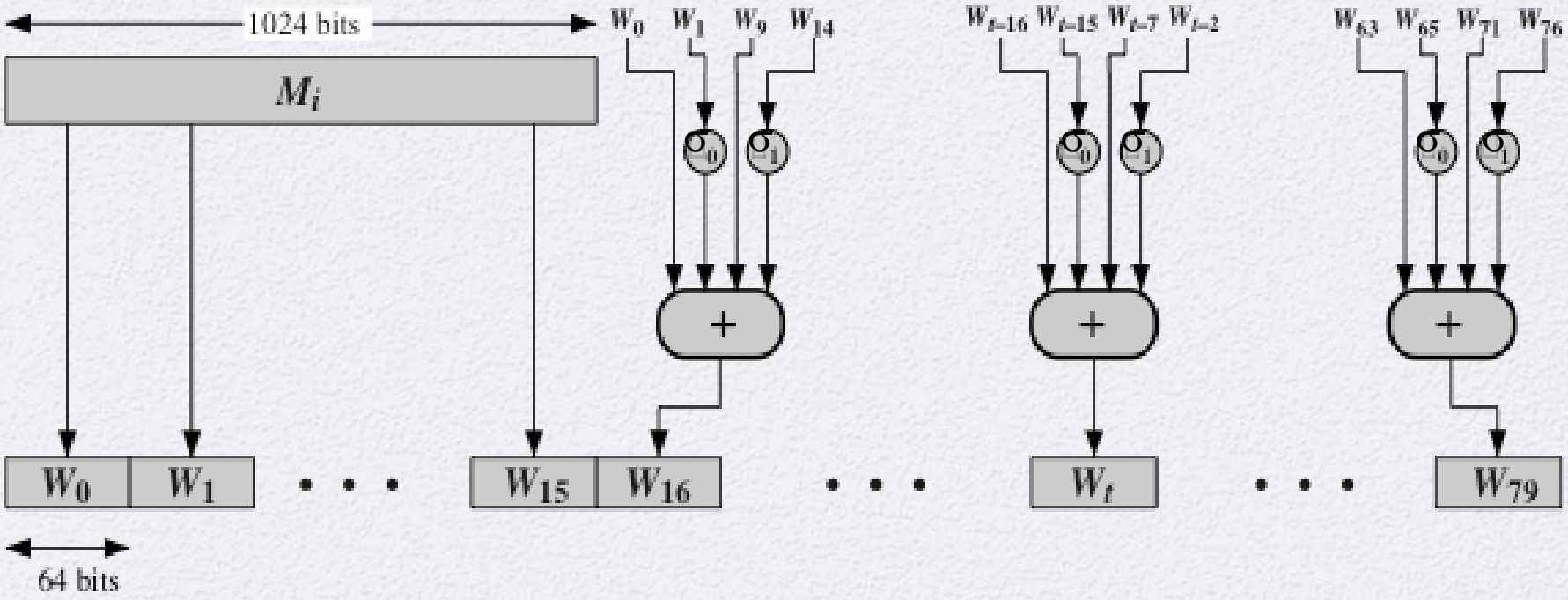


SHA-512 Constants

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817



Single round



80-word input sequences for SHA-512 in each round

SHA-2 Example

```
SHA224("") 0x d14a028c2a3a2bc9476102bb288234c4
              15a2b01f828ea62ac5b3e42f
SHA256("") 0x e3b0c44298fc1c149afbf4c8996fb924
              27ae41e4649b934ca495991b7852b855
SHA384("") 0x 38b060a751ac96384cd9327eb1b1e36a
              21fdb71114be07434c0cc7bf63f6e1da
              274edebfe76f65fbd51ad2f14898b95b
SHA512("") 0x cf83e1357eefb8bdf1542850d66d8007
              d620e4050b5715dc83f4a921d36ce9ce
              47d0d13c5d85f2b0ff8318d2877eec2f
              63b931bd47417a81a538327af927da3e
```

SHA-3

- SHA-1 has not yet been "broken" in practice
  - Differential analysis
- But, considered to be insecure

Concerns about SHA-2: share the same structure and mathematical operations as its predecessors

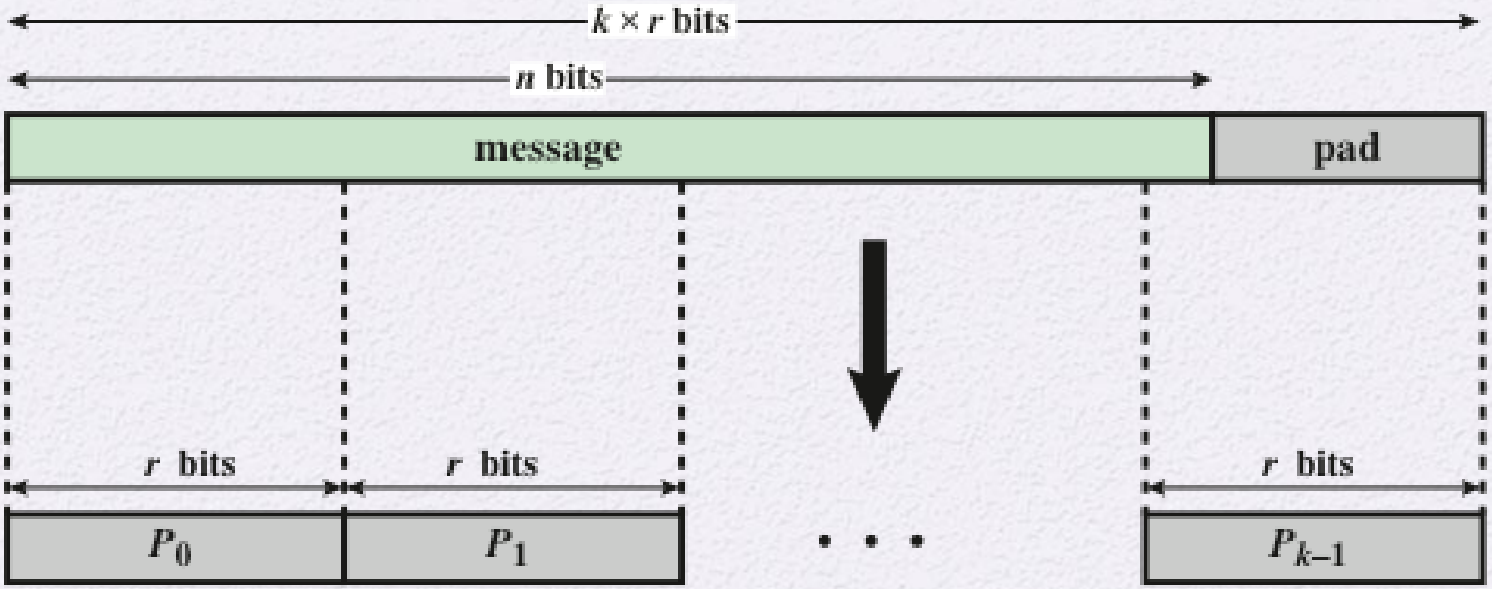
NIST, 2007, call for competition for SHA-3 next generation hash function.

- Winner: October 2012
- SHA-3 : to complement SHA-2 for a wide range of applications

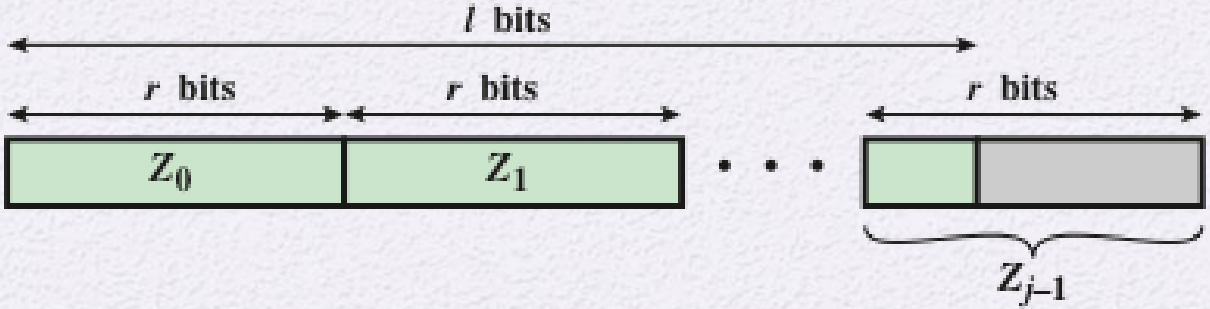
Structure of SHA-3: Sponge

- $M=M_1M_2 \ldots M_N$
- Sponge function is defined by three parameters:
  - f = the internal "permutation" function for processing each input block: b bits →b bits
  - r = the size in bits of the input blocks, called the *bitrate*
  - pad = the padding algorithm

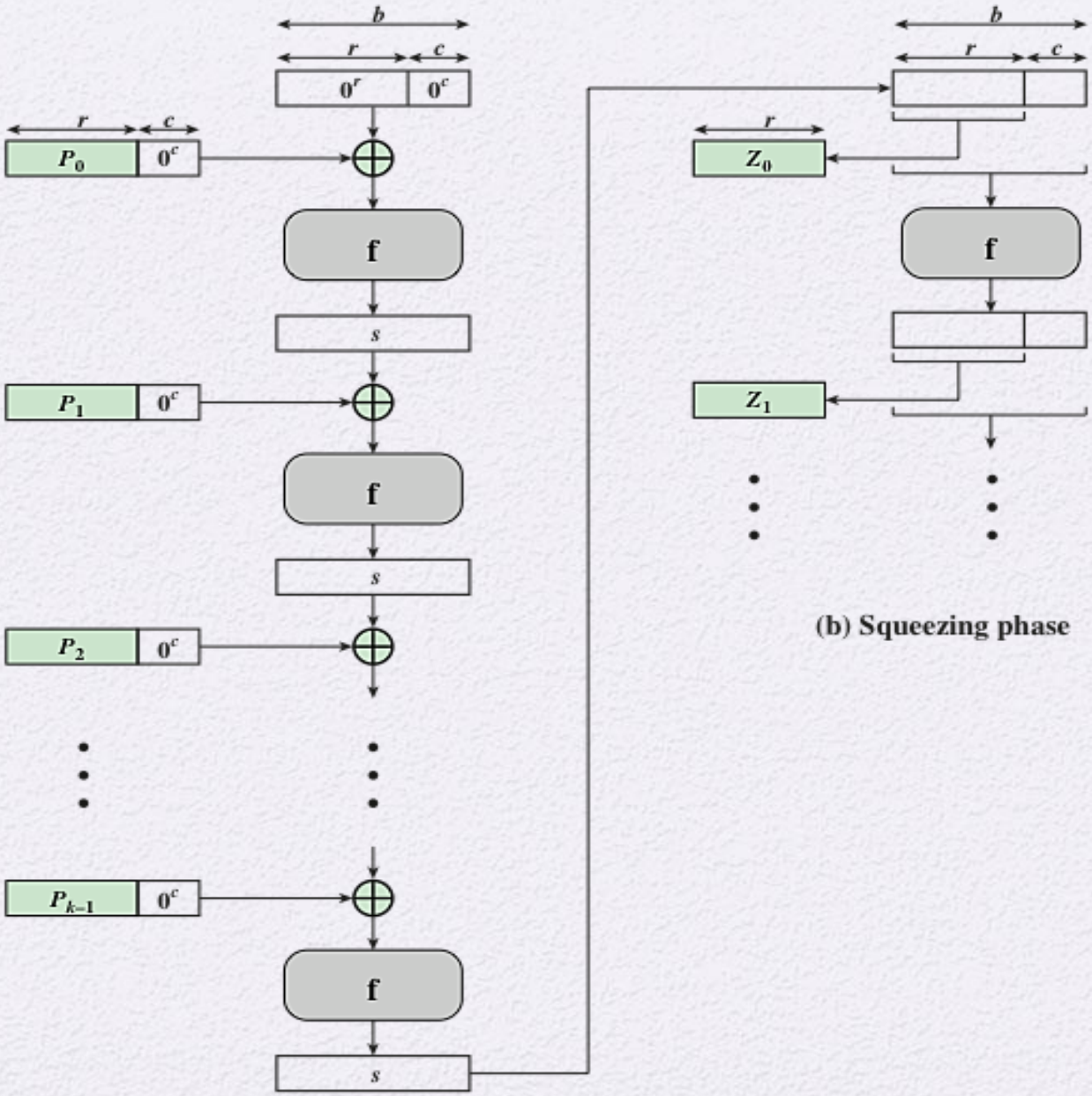
```
SHA3-224("") 6b4e03423667dbb73b6e15454f0eb1ab
             d4597f9a1b078e3f5b5a6bc7
SHA3-256("") a7ffc6f8bf1ed76651c14756a061d662
             f580ff4de43b49fa82d80a4b80f8434a
SHA3-384("") 0c63a75b845e4f7d01107d852e4c2485
             c51a50aaaa94fc61995e71bbee983a2a
             c3713831264adb47fb6bd1e058d5f004
SHA3-512("") a69f73cca23a9ac5c8b567dc185a756e
             97c982164fe25859e0d1dcc1475c80a6
             15b2123af1f5f94c11e3e9402c3ac558
             f500199d95b6d3e301758586281dcd26
```



(a) Input



(b) Output



(a) Absorbing phase

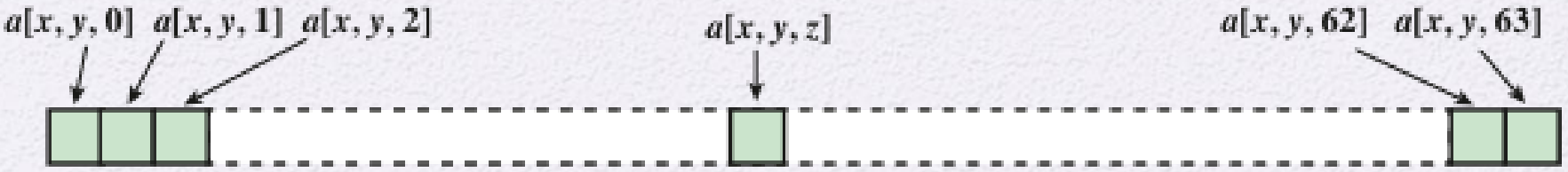
(b) Squeezing phase

SHA-3 Parameters

Message Digest Size	224	256	384	512
Message Size	no maximum	no maximum	no maximum	no maximum
Block Size (bitrate *r*)	1152	1088	832	576
Word Size	64	64	64	64
Number of Rounds	24	24	24	24
Capacity *c*	448	512	768	1024
Collision resistance	2^{112}	2^{128}	2^{192}	2^{256}
Second preimage resistance	2^{224}	2^{256}	2^{384}	2^{512}

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

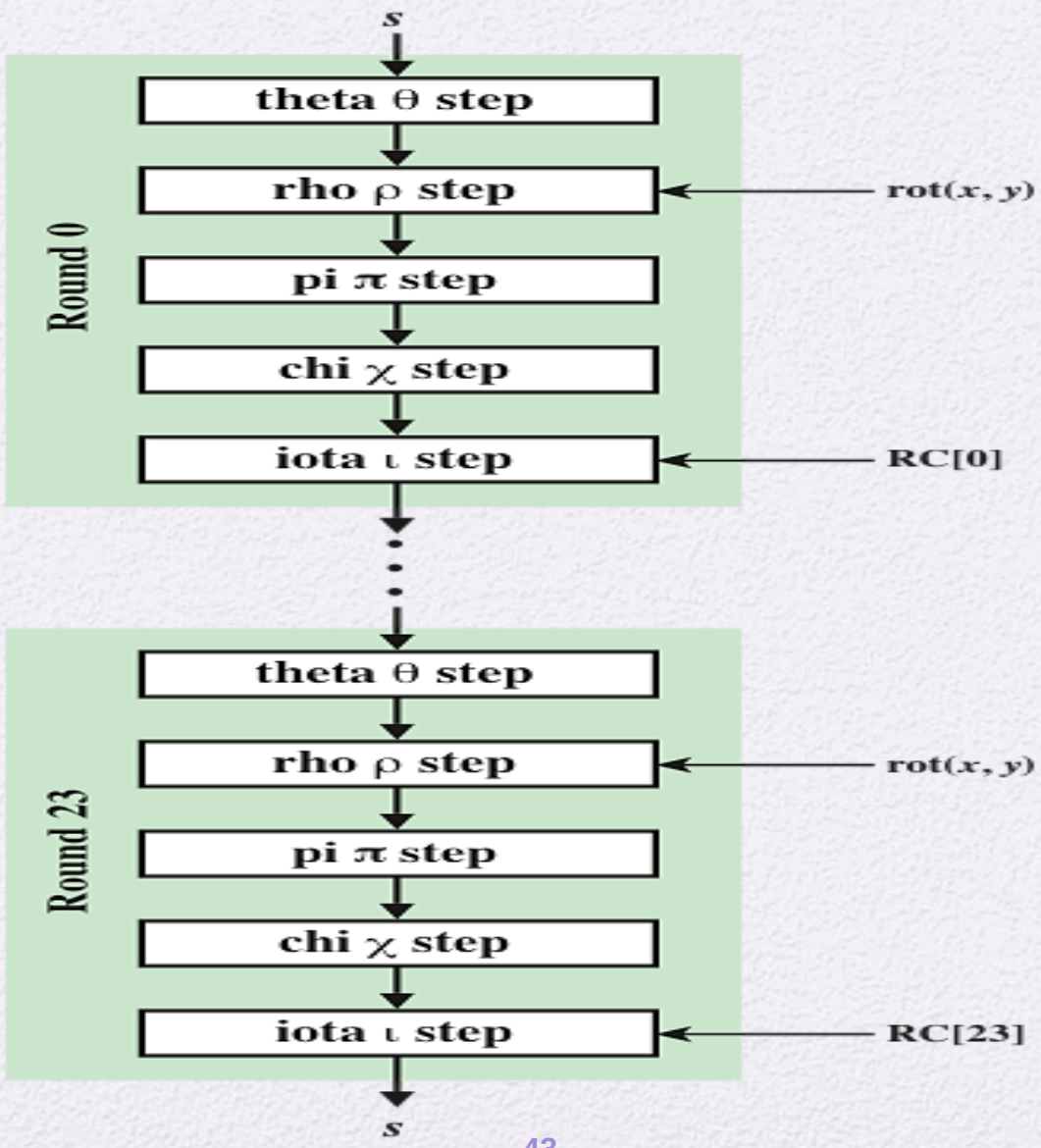
(a) State variable as 5 ×5 matrix A of 64-bit words



(b) Bit labeling of 64-bit words

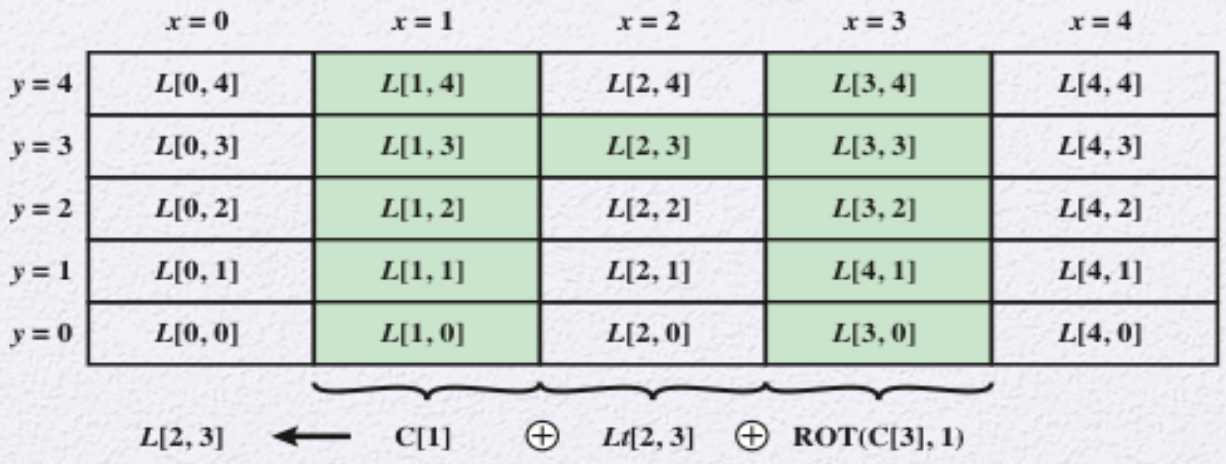
Figure 11.16 SHA-3 State Matrix

Iteration Function *f*

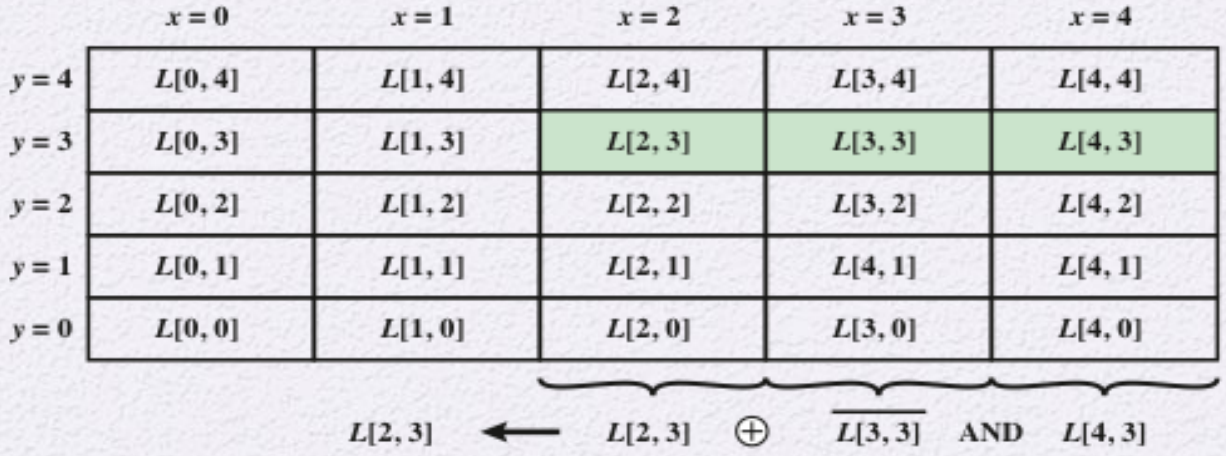


Step functions in SHA-3

Function	Type	Description
θ	substitution	New value of 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.
ρ	permutation	The bits of each word are permuted using a circular bit shift W[0, 0] is not affected.
π	permutation	Words are permuted in the 5x5 matrix. W[0, 0] is not affected.
χ	substitution	New value o each bit 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
ι	substitution	W[0, 0] is updated by XOR with a round constant.

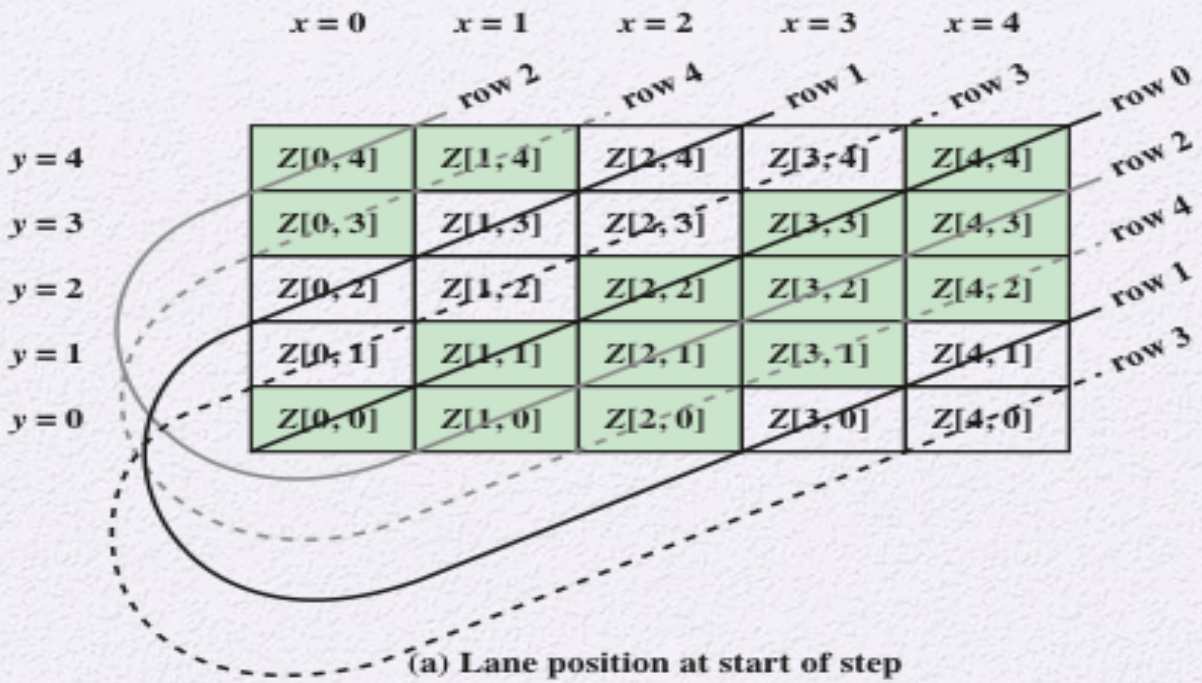


(a) θ step function



(b) χ step function

Figure 11.18 Theta and Chi Step Functions



(a) Lane position at start of step

	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]

(b) Lane position after permutation

Figure 11.19 Pi Step Function

Round Constants in SHA-3

Round	Constant (hexadecimal)	Number of 1 bits
0	0000000000000001	1
1	0000000000008082	3
2	800000000000808A	5
3	8000000080008000	3
4	000000000000808B	5
5	0000000080000001	2
6	8000000080008081	5
7	8000000000008009	4
8	000000000000008A	3
9	0000000000000088	2
10	0000000080008009	4
11	000000008000000A	3

Round	Constant (hexadecimal)	Number of 1 bits
12	000000008000808B	6
13	800000000000008B	5
14	8000000000008089	5
15	8000000000008003	4
16	8000000000008002	3
17	8000000000000080	2
18	000000000000800A	3
19	800000008000000A	4
20	8000000080008081	5
21	8000000000008080	3
22	0000000080000001	2
23	8000000080008008	4

Summary

- Applications of cryptographic hash functions
  - Message authentication
  - Digital signatures
  - Other applications
- Requirements and security
  - Security requirements functions
  - Brute-force attacks
  - Cryptanalysis
- Hash functions based on cipher block chaining
- Secure hash algorithm (SHA)
  - SHA-512 logic
  - SHA-512 round function
- SHA-3
  - The sponge construction
  - The SHA-3 Iteration Function *f*