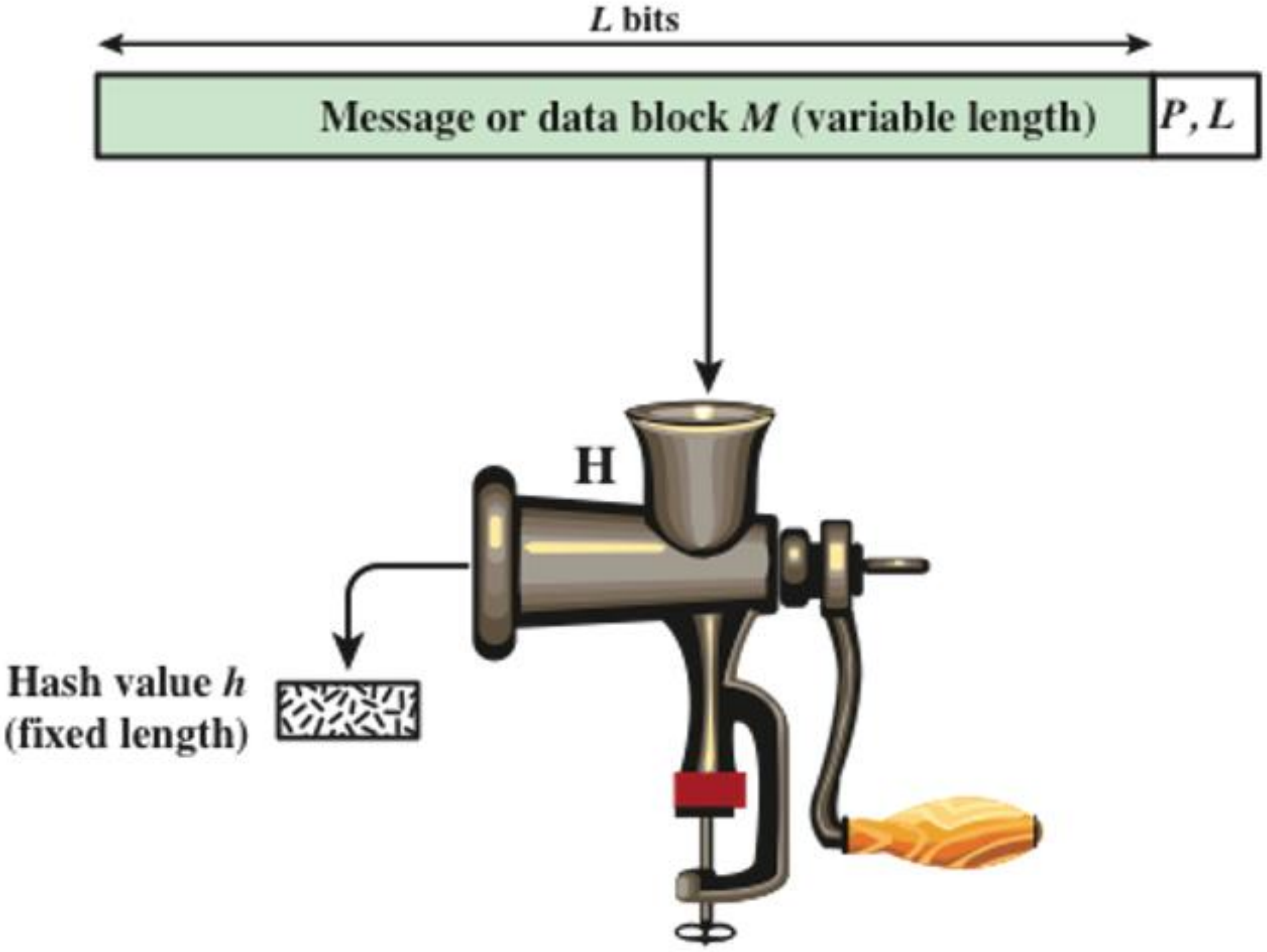
Topics to Cover

- Applications of Cryptographic Hash Functions
- Two Simple Hash Functions
- Hash Functions based on CBC
- Secure Hash Algorithm (SHA)

Cryptographic Hash Functions

- A hash function takes a variable-length data input (M) and produces a fixed-size output (h).
- A good hash function creates outputs that are evenly spread out and look random.
- Help ensure data integrity.
- A cryptographic hash function is a special type of hash function used for security purposes.
- It is computationally infeasible to find an input that produces a specific hash output (one-way property).
- It is computationally infeasible to find two different inputs that give the same hash output (collision-free property).

Cryptographic Hash Functions (Contd..)



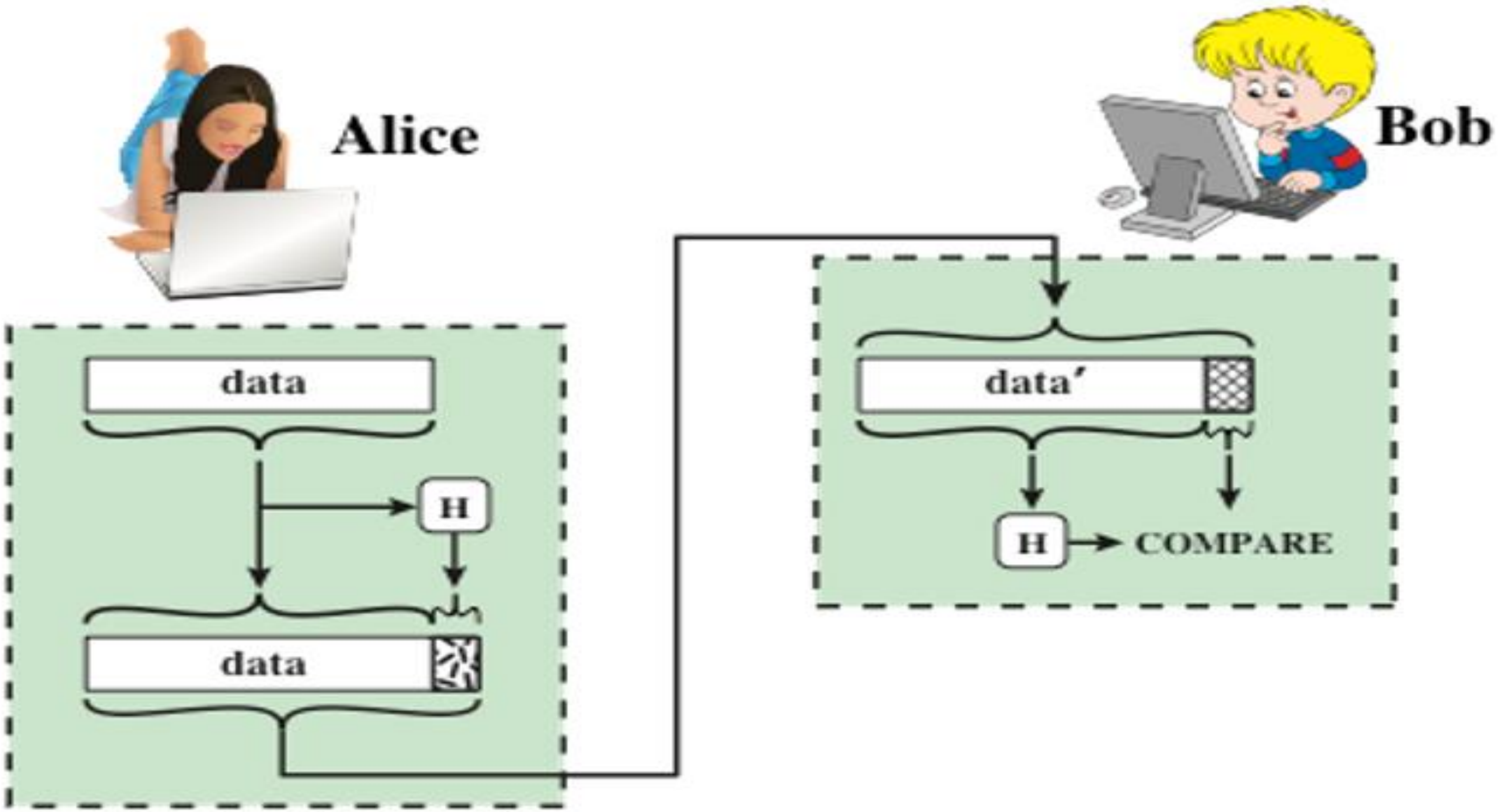
P, L = padding plus length field

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

Message Authentication

- Message authentication ensures that the received message is the same as the sent message, without any changes, additions, deletions, or replays.
- It often also checks that the sender's identity is valid.
- A hash function can be used to authenticate the message, and the result is called a message digest.
- The sender creates a hash value based on the message and sends both the message and the hash value.
- The receiver then calculates the hash value of the message they receive and compares it with the one sent by the sender.
- If the hash values don't match, the receiver knows that the message has been altered in some way.

Use of Hash Function for Message Integrity Check



MITM attack on the Cryptographic Hash Function

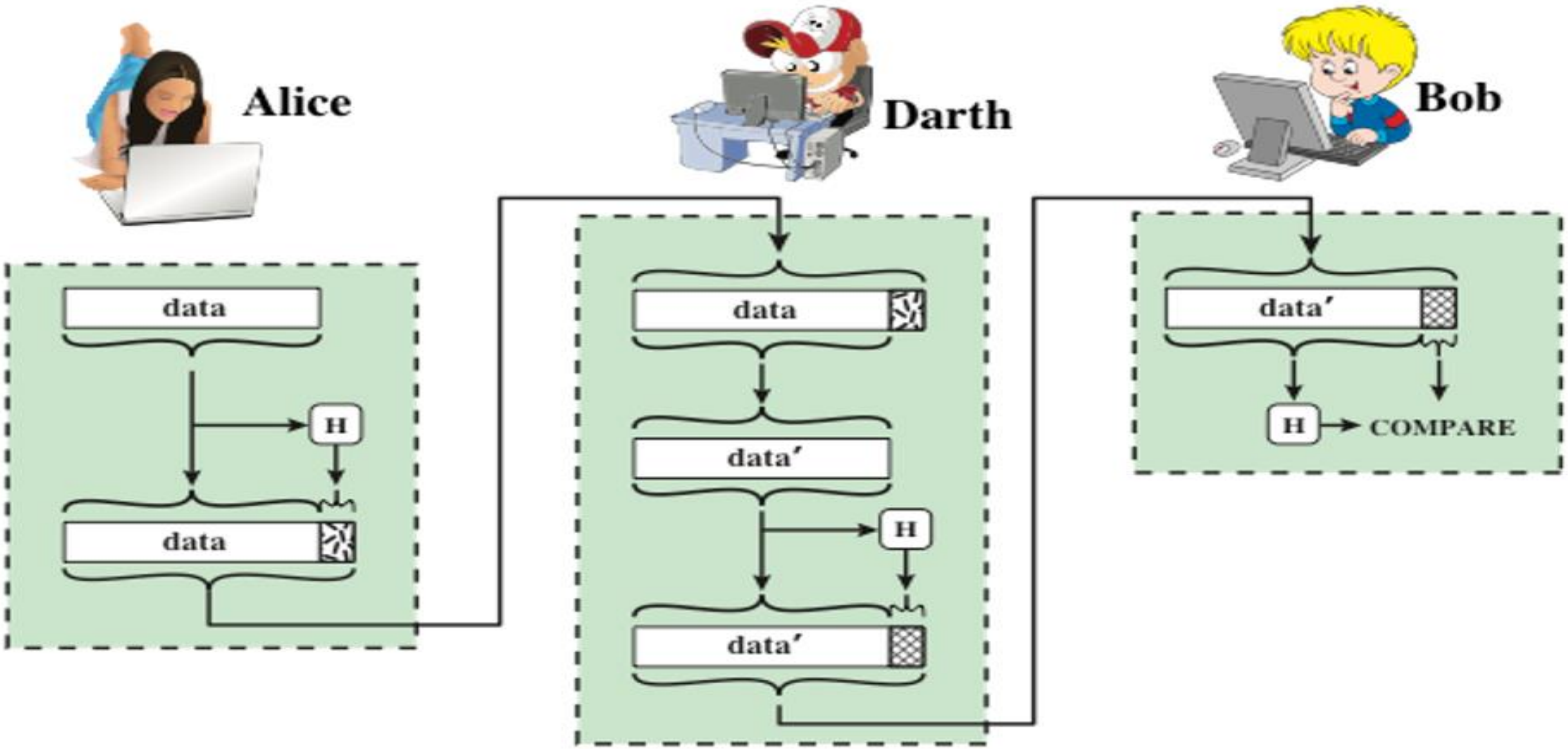
- ➢ The hash value must be sent securely to prevent tampering.

- ➢ If someone changes the message, they shouldn't be able to also change the hash value to deceive the receiver.

- ➢ Example (Alice sending data with the corresponding Hash to Bob):-
- An attacker Darth intercepts the message, changes the data, and creates a new hash value.
- Bob receives the altered data with the new hash and doesn't notice the change.

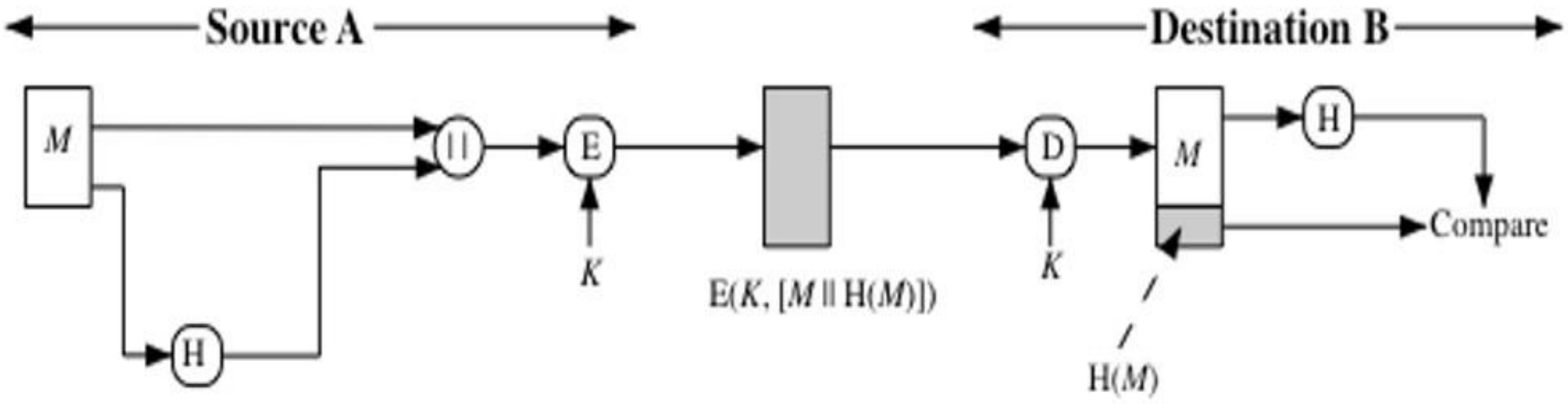
- ➢ To stop this attack, Alice's original hash value must be protected during transmission.

MITM attack on the Cryptographic Hash Function (Contd..)



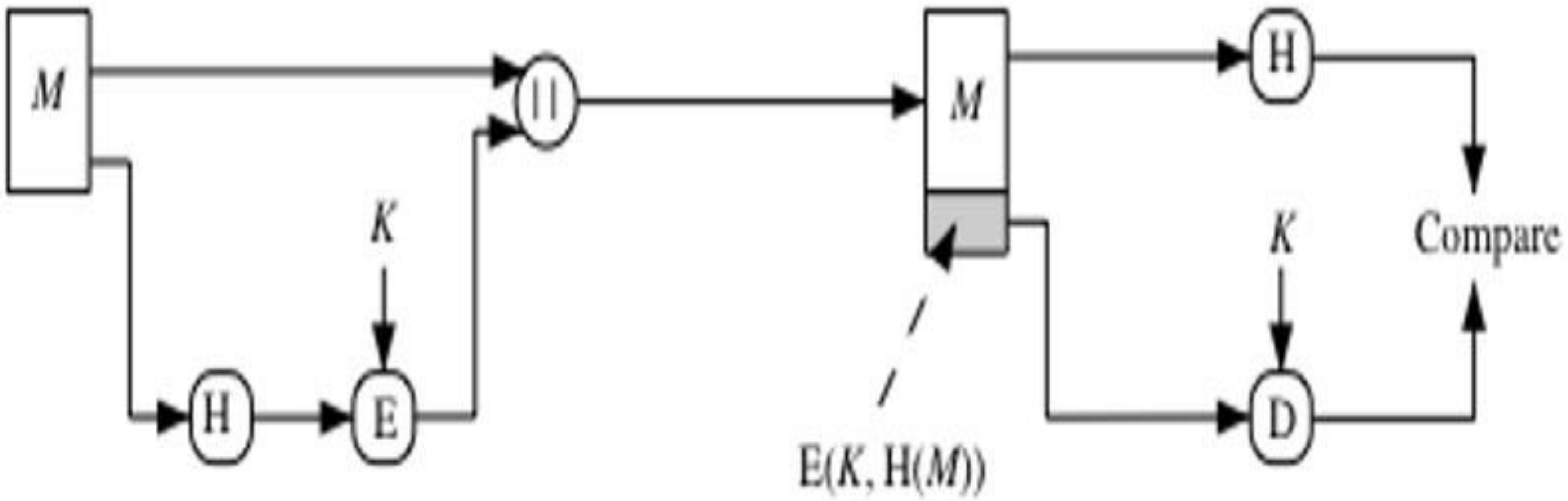
A variety of ways to authenticate messages through hash code

- *Ensuring Authentication and Confidentiality through Symmetric Encryption and Hashing:-*



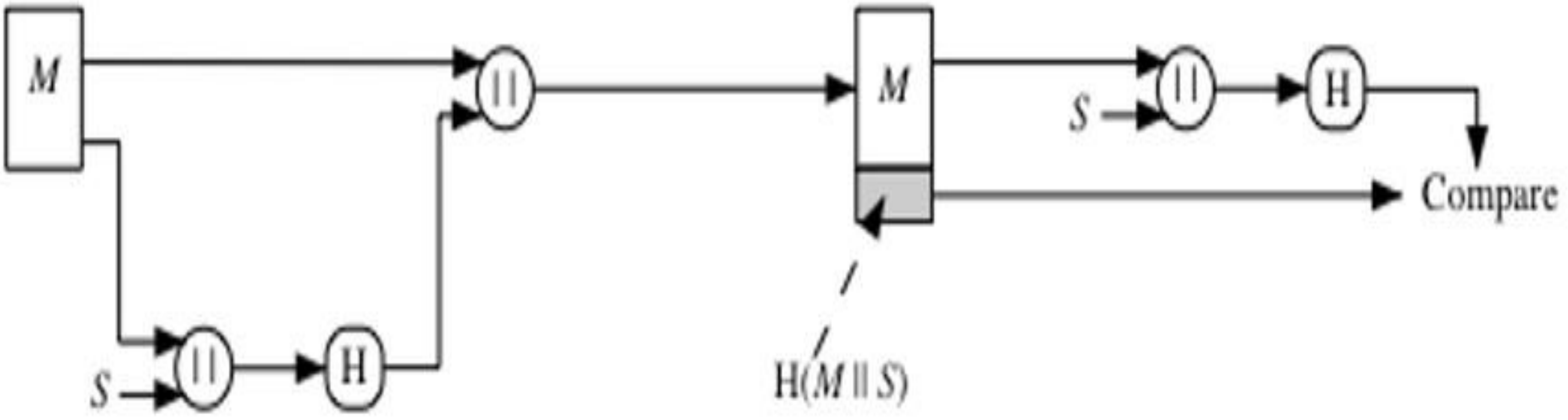
A variety of ways to authenticate messages through hash code (Contd..)

➢ *Optimizing Performance with Symmetric Encryption of Hash Codes:-*



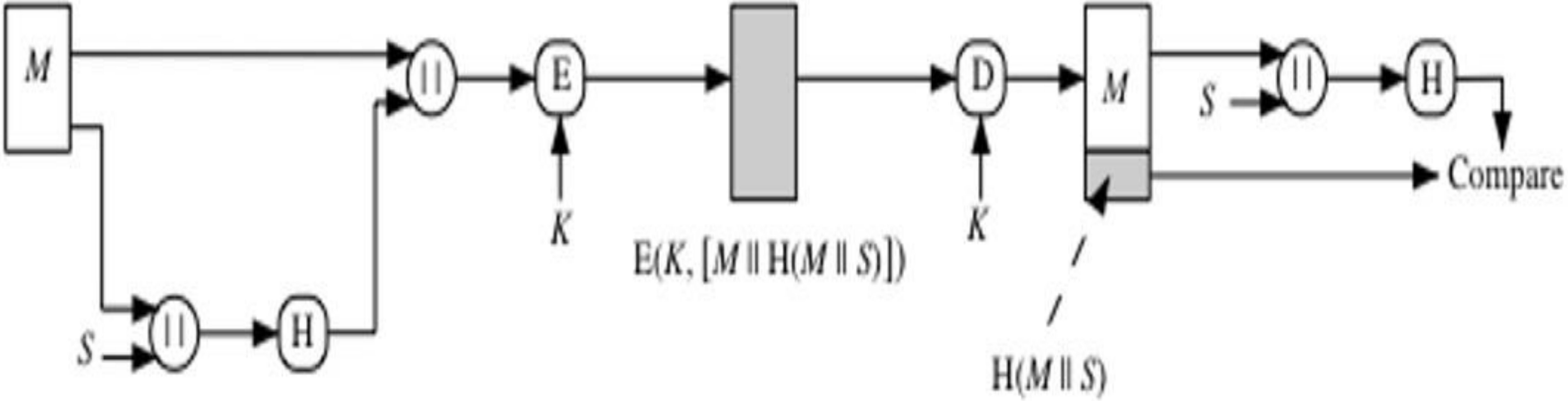
A variety of ways to authenticate messages through hash code (Contd..)

- *Message Authentication Using a Shared Secret Value and Hash Function:-*



A variety of ways to authenticate messages through hash code (Contd..)

➢ *Message Authentication and Confidentiality Using a Shared Secret, Hash Function, and Encryption:-*



Reasons to provide Hash Code-based Message Authentication that avoid encryption

- Any technique patented and '*Message Authentication and Confidentiality Using a Shared Secret, Hash Function, and Encryption*'.

- Reasons:-

- *Encryption software is slow, and despite small data sizes per message, a constant stream of messages can impact system performance.*
- *Encryption hardware can be expensive, especially when every network node requires it.*
- *Encryption hardware works best with large data sizes, as small data blocks incur more overhead during initialization.*
- *Encryption algorithms may be patented and using them often requires a licensing fee.*

Message Authentication (Contd..)

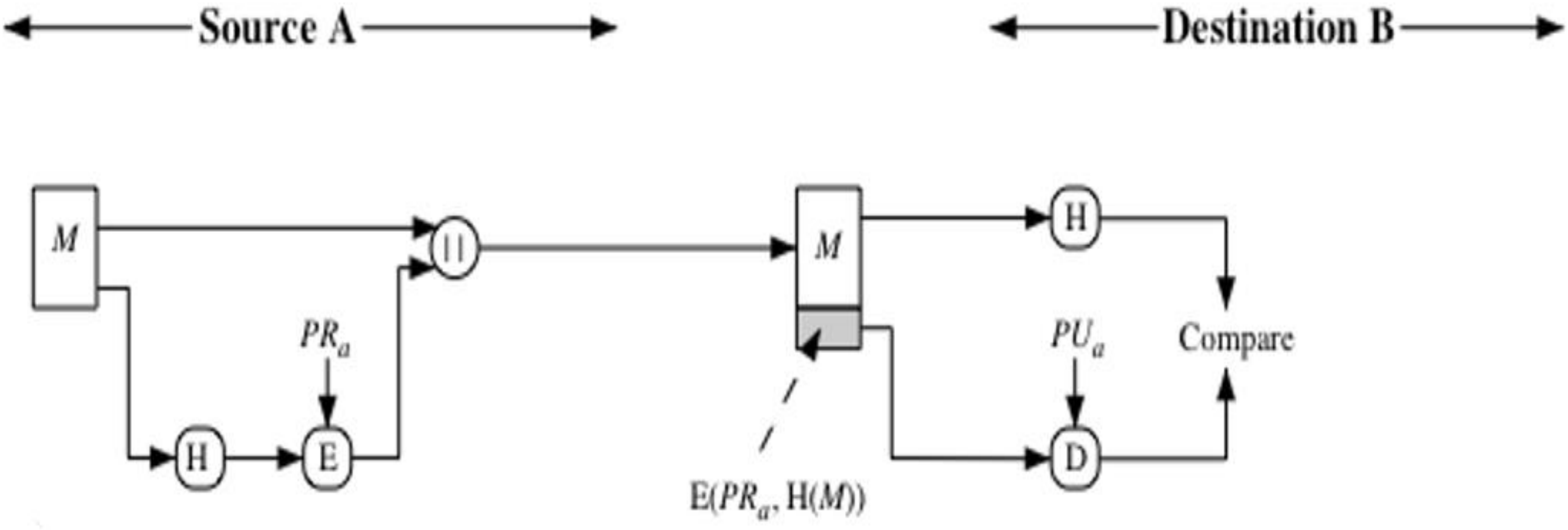
- Message Authentication is often done using a message authentication code (MAC), which involves a secret key.
- The MAC function combines the secret key and the message data to create a unique hash value (MAC).
- To verify the message, the MAC function is applied again, and the result is compared to the original MAC.
- If the message is altered, the attacker can't change the MAC without knowing the secret key.
- The verifying party knows the sender because only they share the secret key.

Digital Signatures

- Digital signature is like message authentication and works like a MAC.
- The hash value of the message is encrypted using the user's private key.
- Anyone who has the user's public key can check if the message is correct (its integrity).
- To change the message, an attacker would need to know the user's private key.

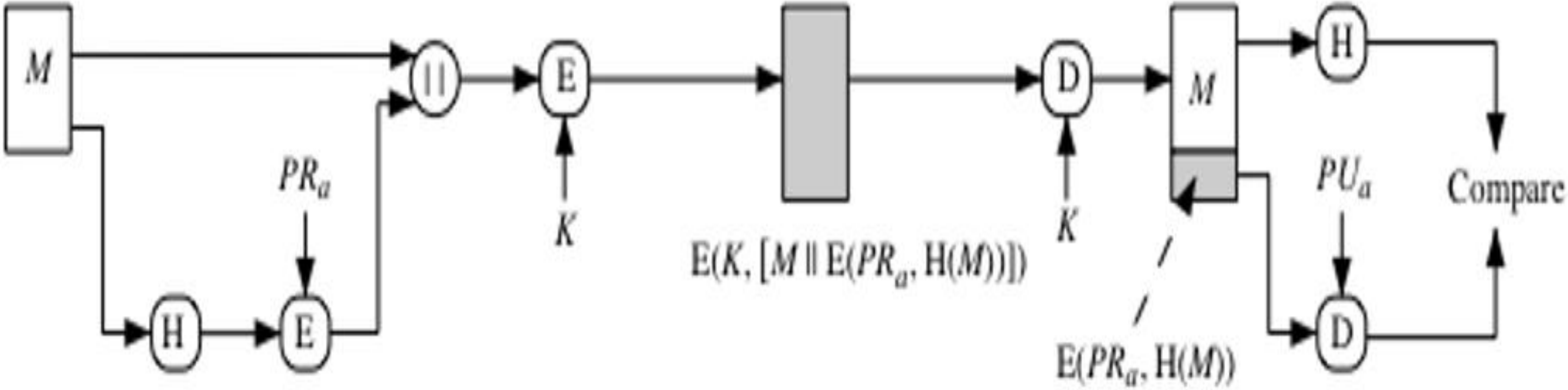
Methods of using Hash Code to provide Digital Signature

- Digital Signatures and Authentication through Public-Key Encryption:-



Methods of using Hash Code to provide Digital Signature (Contd..)

- Securing Messages with Confidentiality and Digital Signatures Using Symmetric Encryption:-

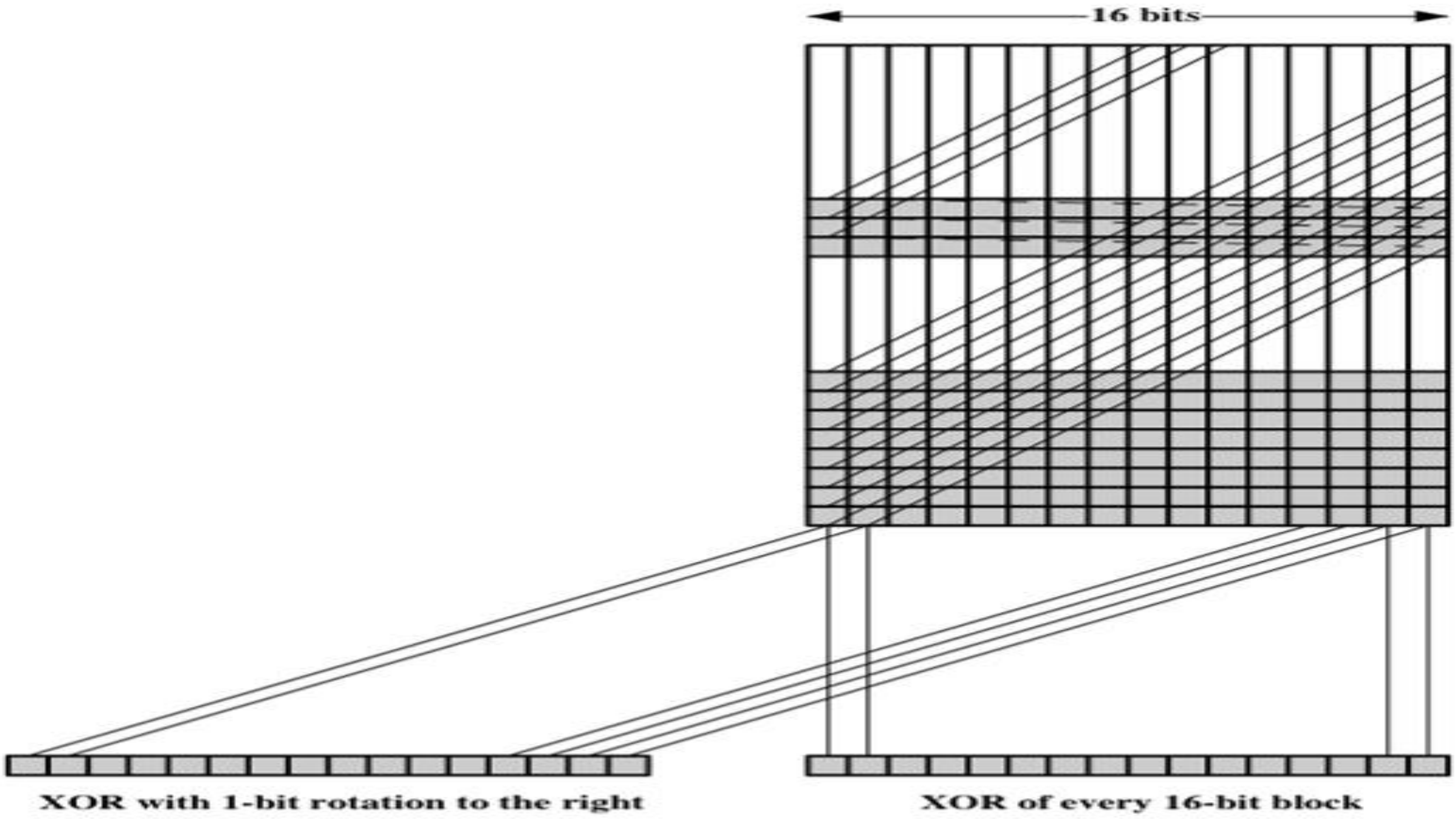


Other Applications

- One-Way Password file
- Intrusion detection and Virus detection
- PRNG
- PRF

TWO SIMPLE HASH FUNCTIONS

Two Simple Hash Functions



Bitwise XOR of every Block of data

- $C_i = b_{i1} \oplus b_{i2} \oplus b_{i3} \oplus \ldots\ldots\ldots\ldots\ldots \oplus b_{im}$; where C_i is the i^{th} bit of Hash Code, and b_{ij} is the i^{th} bit in the j^{th} block.
- This method checks data integrity by generating a simple parity bit for each bit position.
- The probability of an error not affecting the hash is 2^{-n}, where n is the number of bits in the hash.
- Less effective for predictable or patterned data.
- For example, in text files, the high-order bit is usually zero, making the hash function less effective (with an effectiveness of 2^{-112} instead of 2^{-128}).

Enhancing Hash Function Performance with a One-Bit Circular Shift

- ➢ Improve the hash by rotating it after processing each block of data.

- ➢ Steps:-
- Start with the hash value set to zero.
- Rotate the current hash value of each block of data left by 1 bit.
- XOR each data block with the hash value.

- ➢ This process mixes the data better, removing patterns or regularities in the input.

Enhancing Hash Function Performance with a One-Bit Circular Shift (Contd..)

- This hash process works well for checking data integrity but isn't strong enough for overall security.

- If the hash is used with a plaintext message:-

- It's easy to create a new message that matches the original hash.
- A message can be changed, and a specific block can be added that makes the new message match the original hash.

Two Simple Hash Functions (Contd..)

- The 2 simple hash functions are basic and fast operations used in encryption, but they're not very strong by modern security standards.
- If only the hash code is encrypted (without the message), it's not secure enough and might be vulnerable to attacks.
- Encrypting both the message and its hash together using either of the 2 hash functions might provide some protection, but it's still weak.
- Despite seeming useful, the 2 hash functions are not strong enough for secure encryption, so stronger methods should be used to protect both the message and its hash.

CBC Mode with XOR-Based Block Hashing

- The message 'M' is made up of 64-bit blocks (denoted as X_1, X_2, X_3, ..., X_N).

- Hash Code (h) = $X_1 \oplus X_2 \oplus X_3 \oplus \ldots\ldots\ldots \oplus X_N$

- 'h' is then added as an extra block at the end of the message, making the total message length (N+1) blocks.

- The entire message (including the hash code) is encrypted using CBC operating mode.

CBC Mode with XOR-Based Block Hashing (Contd..)

- <u>Application of CBC:-</u>
- Encrypted Message:- $Y_1, Y_2, Y_3, \ldots\ldots\ldots.., Y_N, Y_{N+1}$
- $X_1 = IV \oplus D(K,Y_1)$
- $X_i = Y_{i-1} \oplus D(K, Y_i)$
- $X_{N+1} = Y_N \oplus D(K, Y_{N+1})$
- $X_{N+1} = [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \ldots\ldots \oplus [Y_{N-1} \oplus D(K, Y_N)]$

- The hash code remains the same if the ciphertext blocks are permuted, since XORing the terms is order-independent.

HASH FUNCTIONS BASED ON CBC

Hash Functions based on CBC

- Proposals exist for hash functions using the CBC technique, but without a secret key.

- One early proposal was by Rabin:
- Message (M):- M_1, M_2, M_3, ………., M_N
- A symmetric encryption system like DES is used to compute the hash code G.
- $H_i = E(M_i, H_{i-1})$; where H_0 is the initial value.
- $G = H_N$

- Rabin method is similar to the CBC technique but does not use a secret key.

- Like any hash code, this method is vulnerable to the birthday attack.

- If the encryption algorithm is DES and only a 64-bit hash code is produced, the system is vulnerable.

Meet In the Middle (MIM) attack

- A variation of the birthday attack can be used even if the opponent only has one message and its valid signature.

- The opponent cannot obtain multiple signings.

- The scenario assumes that the opponent intercepts a message with a signature in the form of an encrypted hash code.

- The unencrypted hash code is ‘m’ bits long.

Meet In the Middle (MIM) attack (Contd..)

- Steps:-
- Calculate G.
- Construct the desired message (Q_1, Q_2, Q_3, …….., Q_{N-2}).
- $H_i = E(Q_i, H_{i-1})$; where H_0 is the initial value., $1 \leq I \leq (N-2)$.
- Generate $2^{m/2}$ random blocks. For each block X, compute $E(X, H_{N-2})$.
- Generate additional $2^{m/2}$ random blocks. For each block Y, compute D(Y, G).
- According to birthday paradox, there will be a X and Y such that $E(X, H_{N-2}) = D(Y, G)$.
- Construct the final message (Q_1, Q_2, Q_3, …….., Q_{N-2}, X, Y).
- This message will have the same hash code G and can be used with the intercepted encrypted signature.

Refinements for CBC-based Hash Functions

- $H_i = E(M_i, H_{i-1}) \oplus H_{i-1}$
- $H_i = E(H_{i-1}, M_i) \oplus M_i$
- Both schemes are vulnerable to various attacks.
- As a result, there has been a shift towards finding alternative hashing methods.

SECURE HASH ALGORITHM (SHA)

History and Evolution of SHA

- SHA has been the most used hash function in recent years.
- Many other widely used hash functions had serious weaknesses, so SHA became the main standard by 2005.
- SHA was developed by NIST and published as a standard (FIPS 180) in 1993.
- The original version of SHA, called SHA-0, had weaknesses, so it was updated to SHA-1 in 1995.
- SHA-1 creates a hash value of 160 bits.
- SHA-2 was introduced in 2002 with three new versions: SHA-256, SHA-384, and SHA-512 (with hash sizes of 256, 384, and 512 bits).

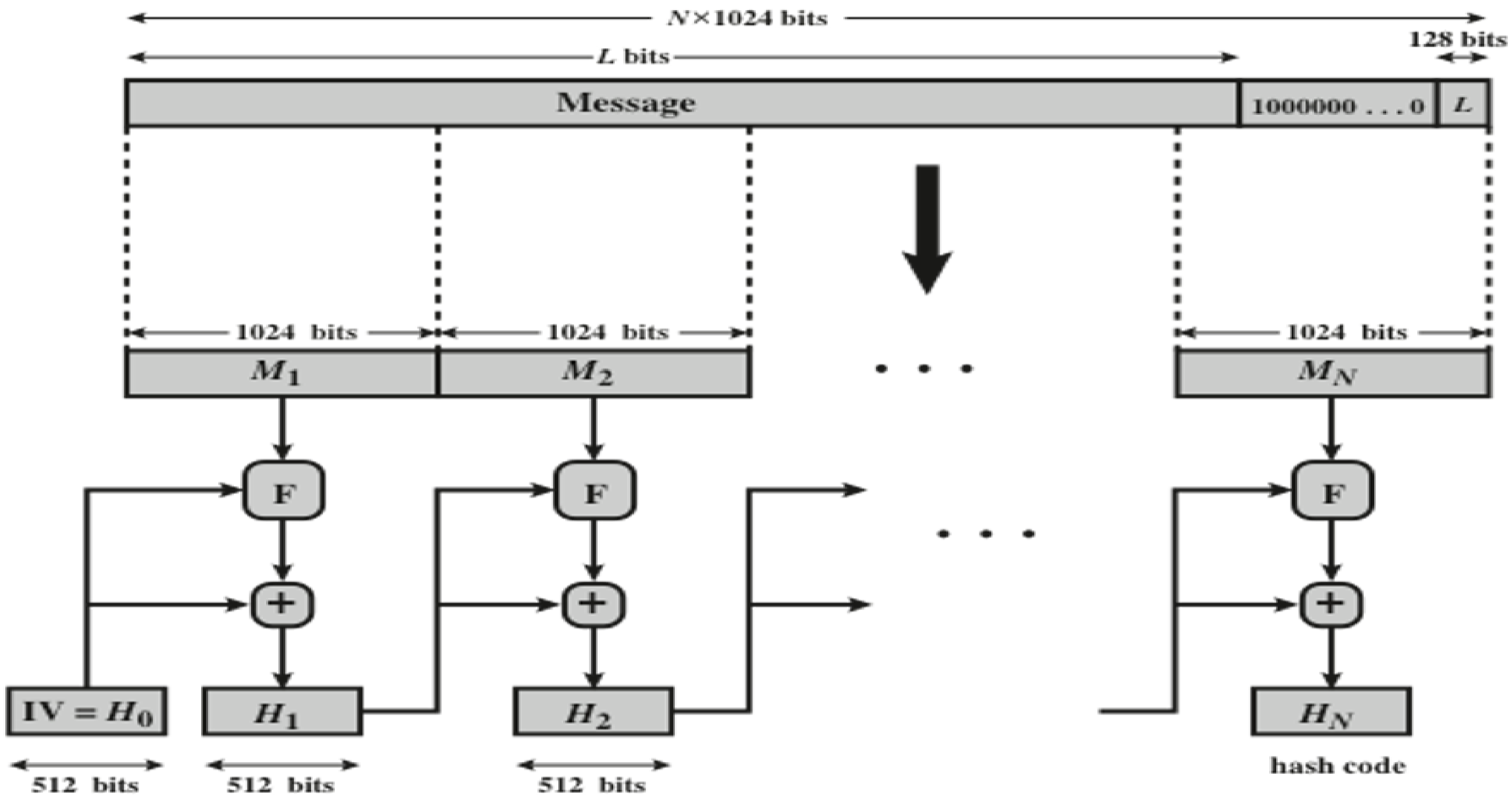
History and Evolution of SHA (Contd..)

- These SHA-2 versions are based on the same structure and operations as SHA-1 but are stronger.
- In 2008, a version called SHA-224 was added to SHA-2.
- In 2015, two more versions, SHA-512/224 and SHA-512/256, were added.
- In 2005, NIST started planning to move from SHA-1 to SHA-2, expecting the transition to be completed by 2010.
- In 2005, researchers found that finding two messages with the same SHA-1 hash (a collision) could be done with 2^{69} operations, which was much fewer than expected, speeding up the move to SHA-2.

Comparison of SHA parameters

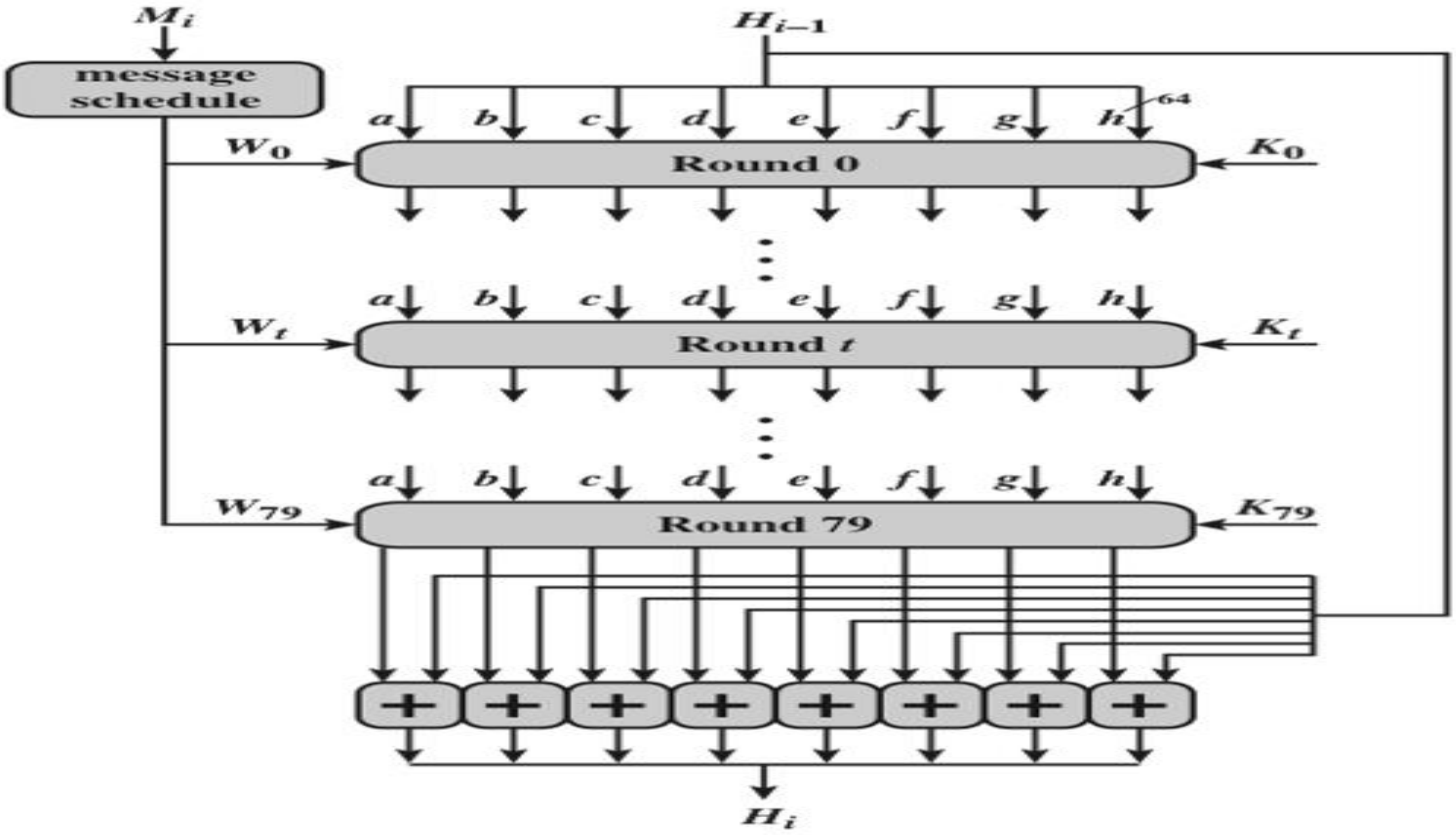
Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Message Digest Generation using SHA-512



- ‘+’ stands for word-by-word addition mod 2^{64}

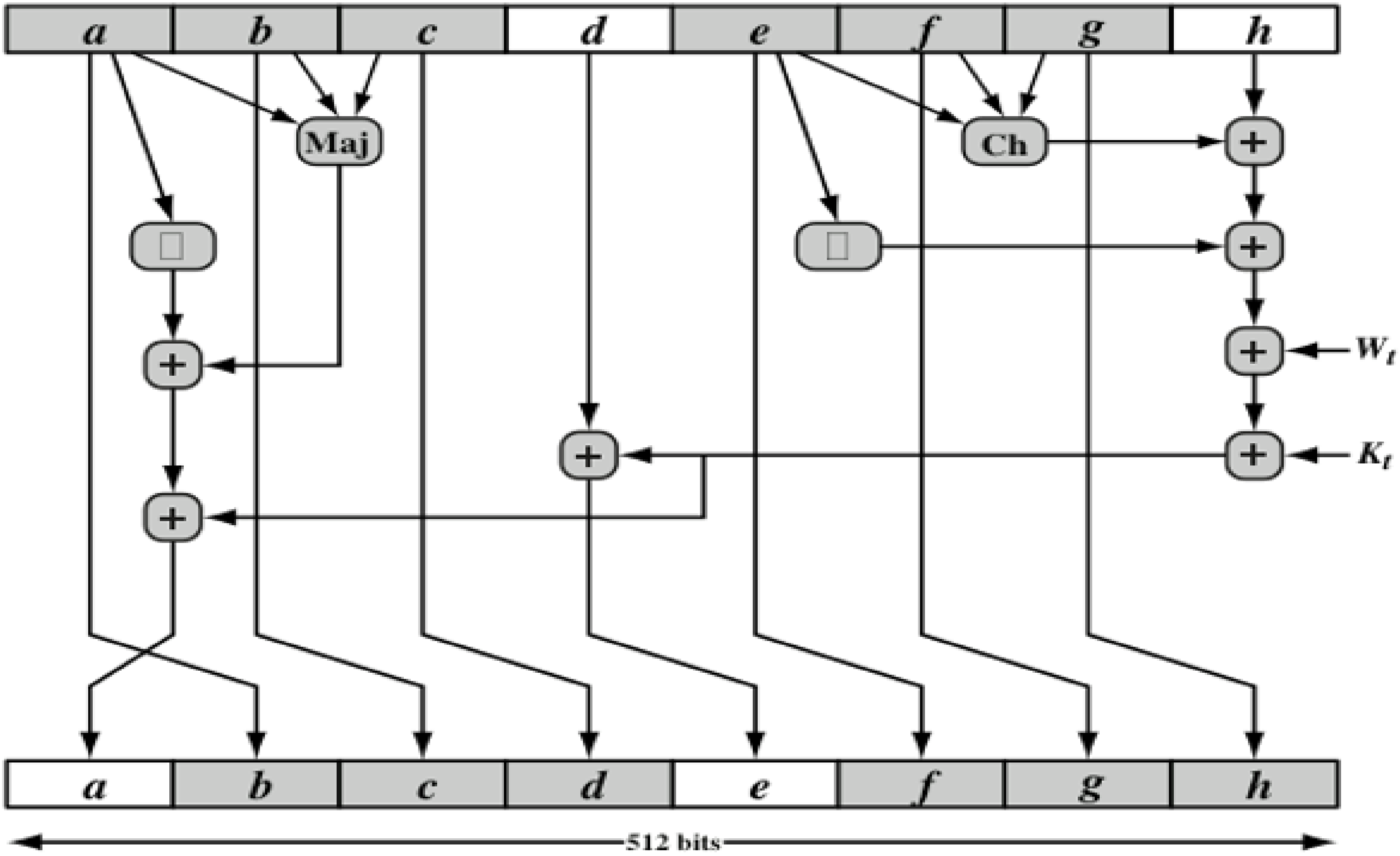
Processing of a single 1024-bit block in SHA-512



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

SHA-512 Round Function



SHA-512 Round Function (Contd..)

$$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t$$
$$T_2 = \left(\sum\nolimits_0^{512} a\right) + \mathrm{Maj}(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

where

t	$=$ step number; $0 \le t \le 79$
$\mathrm{Ch}(e, f, g)$	$= (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$ *the conditional function: If e then f else g*
$\mathrm{Maj}(a, b, c)$	$= (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$ *the function is true only of the majority (two or three) of the arguments are true*
$(\Sigma_0^{512} a)$	$= \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$
$(\Sigma_1^{512} e)$	$= \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$
$\mathrm{ROTR}^n(x)$	$=$ circular right shift (rotation) of the 64-bit argument x by n bits

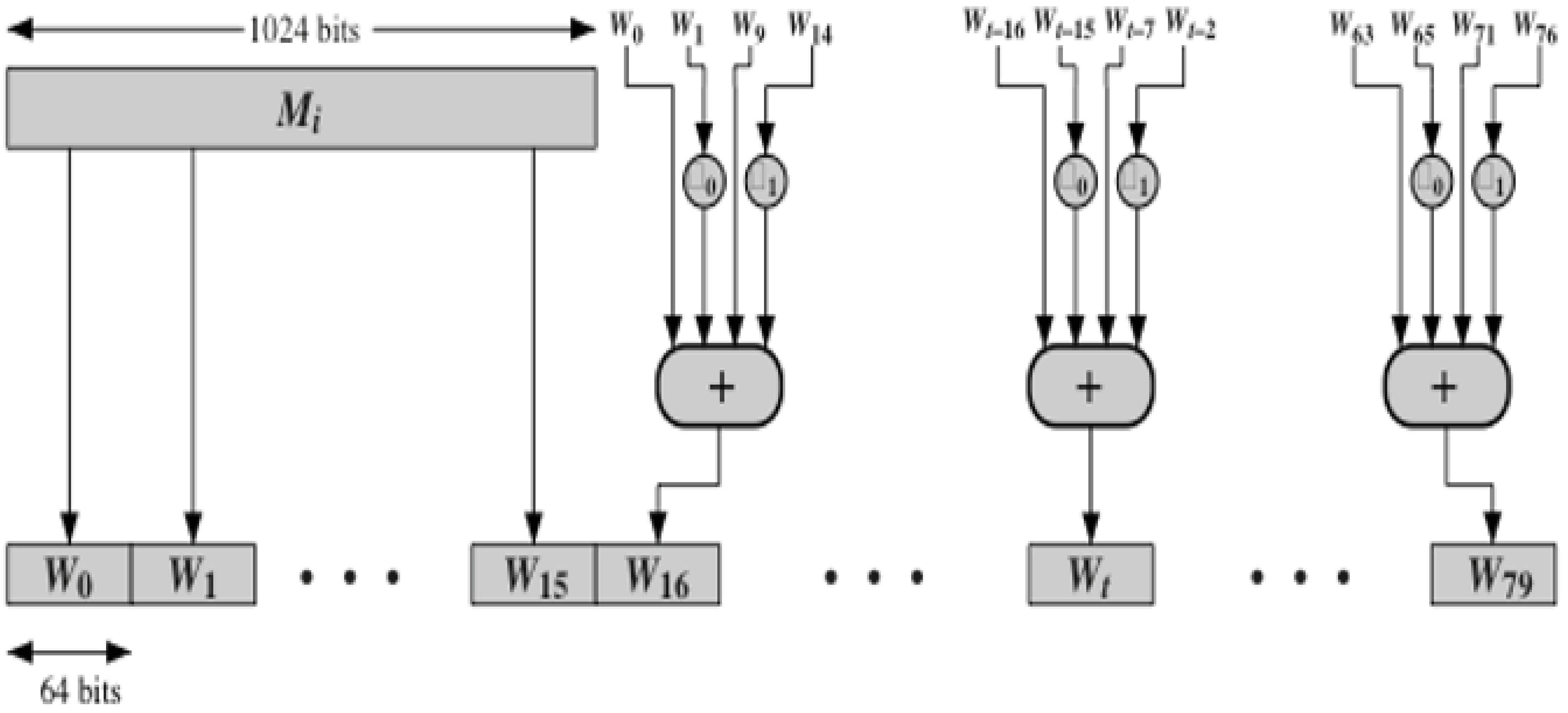
SHA-512 Round Function (Contd..)

- W_t = a 64-bit word derived from the current 1024-bit input block
- K_t = a 64-bit additive constant
- $+$ = addition modulo 2^{64}

➢ Observations:-

- 6 out of the 8 output values in the round function are just rearranged (rotated) versions of the input values.
- Only 2 output values, *a* and *e*, are created through substitution (changing the input values).
- *e* depends on the variables *d*, *e*, *f*, *g*, and *h*, as well as the round word W_t and constant K_t.
- *a* depends on all the input variables except *d*, as well as the round word W_t and constant K_t.

Generation of the 80-Word Input Sequence for SHA-512 Block Processing



Generation of the 80-Word Input Sequence for SHA-512 Block Processing (Contd..)

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$$

$$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

$\mathrm{SHR}^n(x)$ = right shift of the 64-bit argument x by n bits with padding by zeros on the left

$+$ = addition modulo 2^{64}

SHA-512 Logic

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

$H_{0,0} = $ 6A09E667F3BCC908 $\quad H_{0,4} = $ 510E527FADE682D1
$H_{0,1} = $ BB67AE8584CAA73B $\quad H_{0,5} = $ 9B05688C2B3E6C1F
$H_{0,2} = $ 3C6EF372FE94F82B $\quad H_{0,6} = $ 1F83D9ABFB41BD6B
$H_{0,3} = $ A54FF53A5F1D36F1 $\quad H_{0,7} = $ 5BE0CD19137E2179

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

   $a = H_{i-1,0} \qquad e = H_{i-1,4}$
   $b = H_{i-1,1} \qquad f = H_{i-1,5}$
   $c = H_{i-1,2} \qquad g = H_{i-1,6}$
   $d = H_{i-1,3} \qquad h = H_{i-1,7}$

3. Perform the main hash computation

   for $t = 0$ **to** 79

   $$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t$$
   $$T_2 = \left(\sum\nolimits_0^{512} a\right) + \mathrm{Maj}(a, b, c)$$
   $h = g$
   $g = f$
   $f = e$
   $e = d + T_1$
   $d = c$
   $c = b$
   $b = a$
   $a = T_1 + T_2$

4. Compute the intermediate hash value

   $H_{i,0} = a + H_{i-1,0} \qquad H_{i,4} = e + H_{i-1,4}$
   $H_{i,1} = b + H_{i-1,1} \qquad H_{i,5} = f + H_{i-1,5}$
   $H_{i,2} = c + H_{i-1,2} \qquad H_{i,6} = g + H_{i-1,6}$
   $H_{i,3} = d + H_{i-1,3} \qquad H_{i,7} = h + H_{i-1,7}$

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