Cryptography and Computer Security (CSS)

Lecture # 8

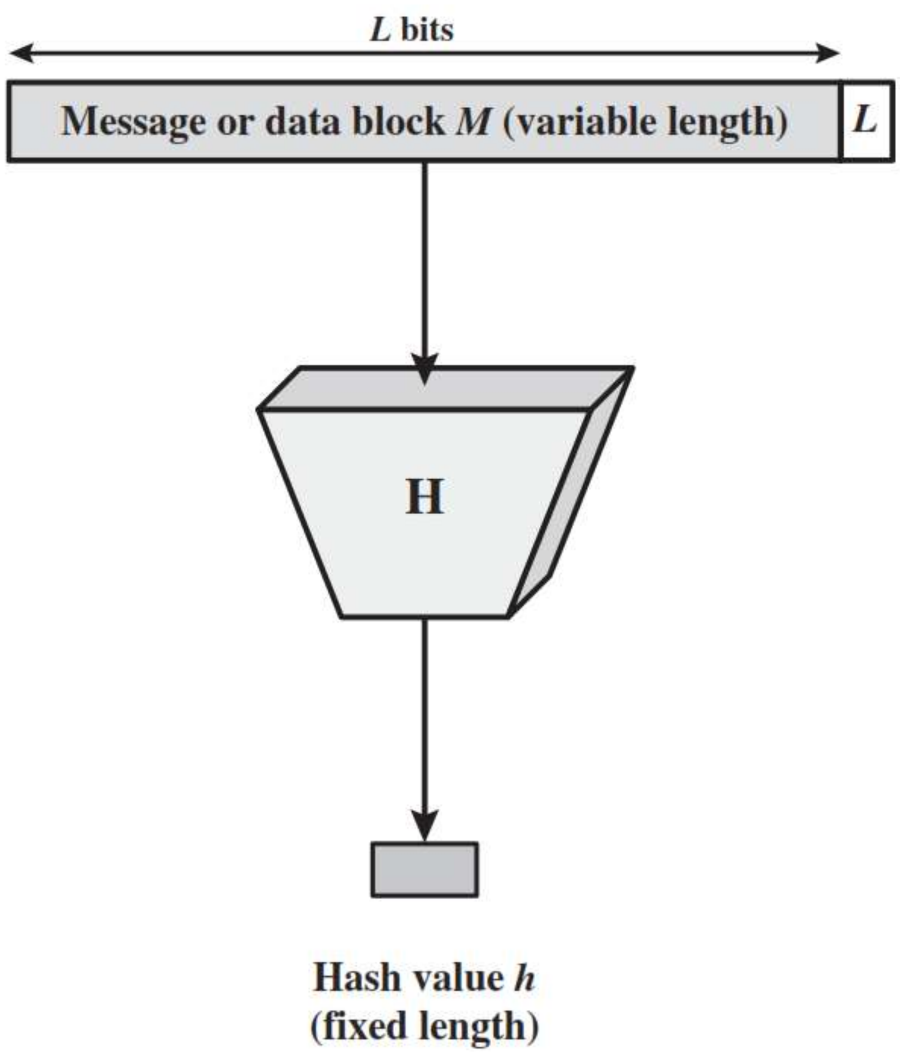
INTRODUCTION
TO
CSS COURSE
(CS401)

Unit III
Cryptographic Hash Functions

- Cryptographic Hash Functions
- Applications
- Simple hash functions
- Requirements and security
- Hash functions based on Cipher Block Chaining
- Secure Hash Algorithm (SHA)

Hash Function

- A hash function **H** accepts a variable-length block of data **M** as input and produces a fixed-size hash value **h = H(M)**.
- A “good” hash function has the property that the results of applying a change to any bit or bits in **M** results, with high probability, in a change to the **hash code**.



Applications of Cryptographic Hash Functions

1. Message authentication
2. Digital Signature
3. One-way password file

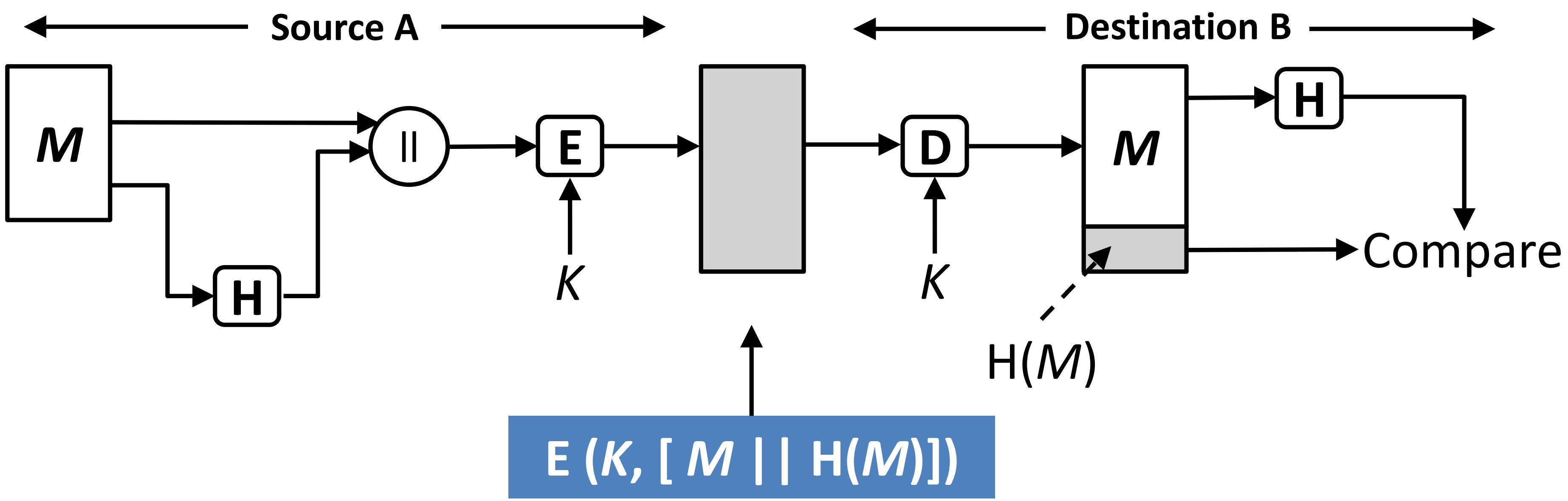
1. Message Authentication

- **Message authentication** is a mechanism or service used to verify the integrity of a message.

- Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay).

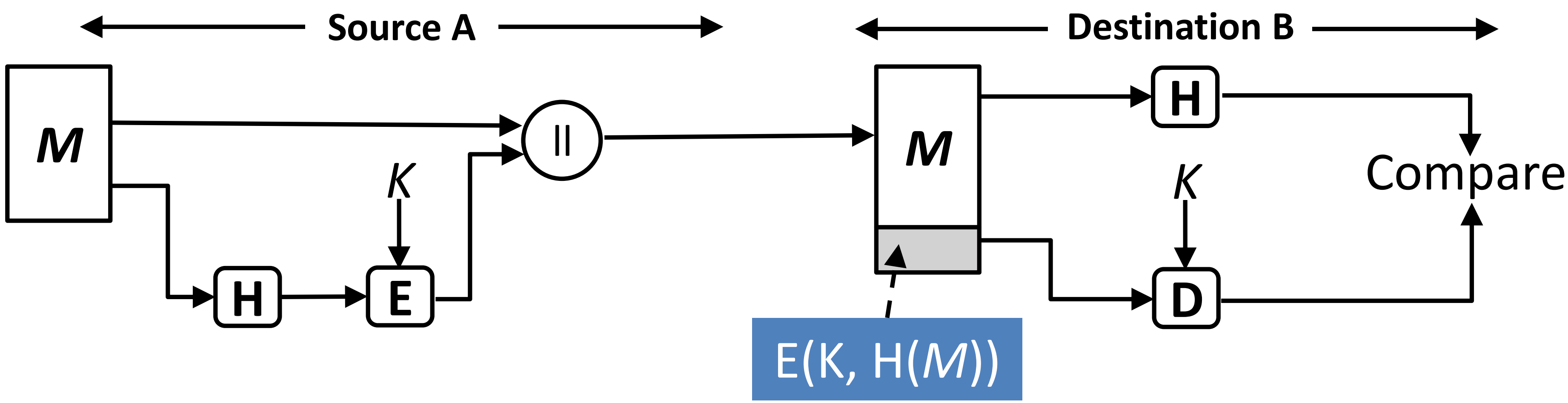
- When a hash function is used to provide message authentication, the **hash function value** is often referred to as a **message digest**.

Message authentication method - 1



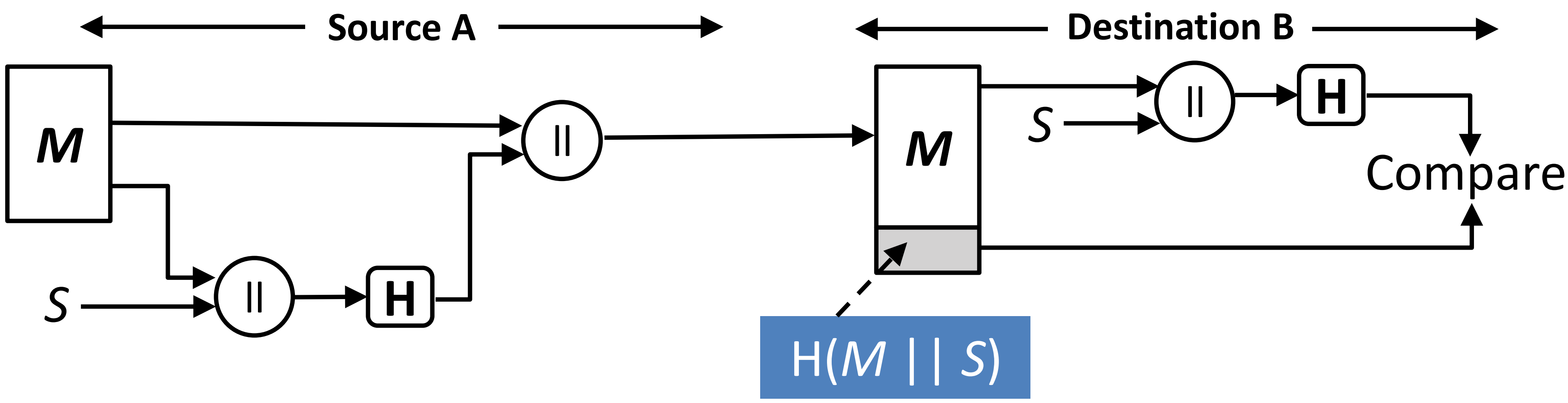
- Only A and B share the secret key, the message must have come from A and has not been altered.
- The hash code provides the structure required to achieve authentication.
- Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

Message authentication method - 2



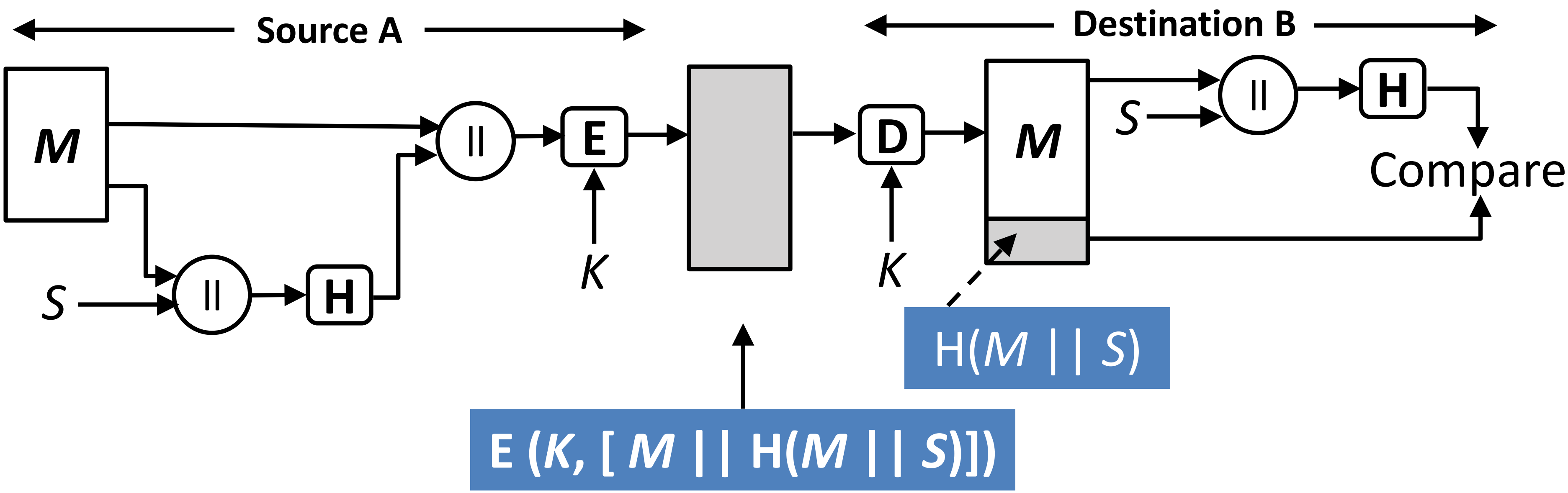
- Only the hash code is encrypted, using symmetric encryption.
- This reduces the processing burden for those applications that do not require confidentiality.

Message authentication method - 3



- It is possible to use a hash function but no encryption for message authentication.
- A and B share a common secret value S.
- A computes the hash value over the concatenation of M and S and appends the resulting hash value to M.
- Because B possesses S, it can recompute the hash value to verify.
- An opponent cannot modify an intercepted message.

Message authentication method - 4



- Confidentiality can be added to the approach of method (3) by encrypting the entire message plus the hash code.

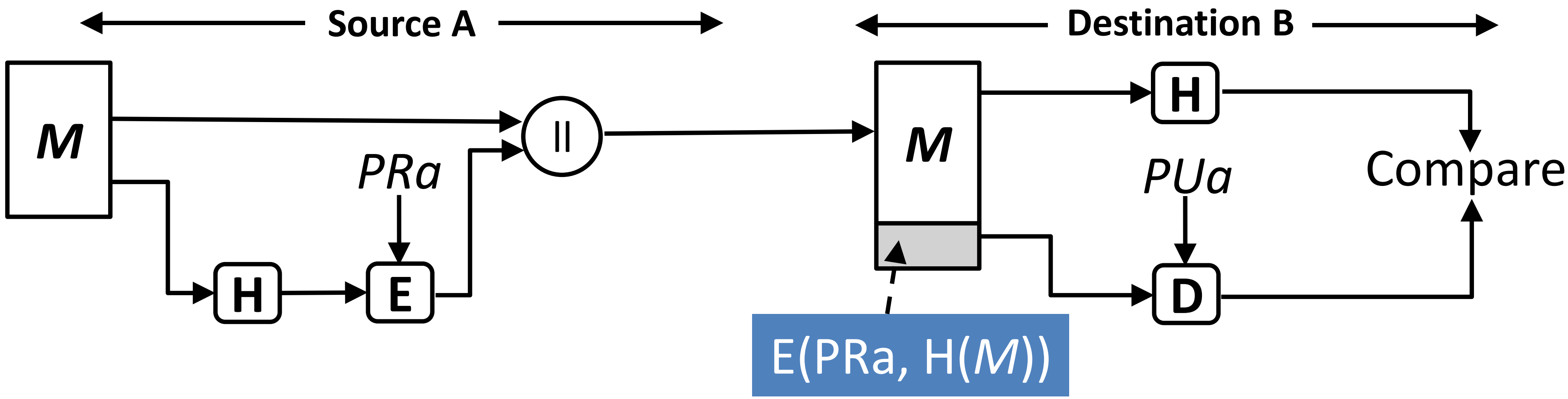
MAC (Message Authentication Code)

- More commonly, message authentication is achieved using a **MAC** also known as **keyed hash function**.
- MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.
- A **MAC** function takes as input a secret key and a data block and produces a hash value, referred to as the **MAC**.
- The combination of hashing and encryption results in an overall function that is, in fact, a MAC (Method -2 in previous slide).

Digital Signature

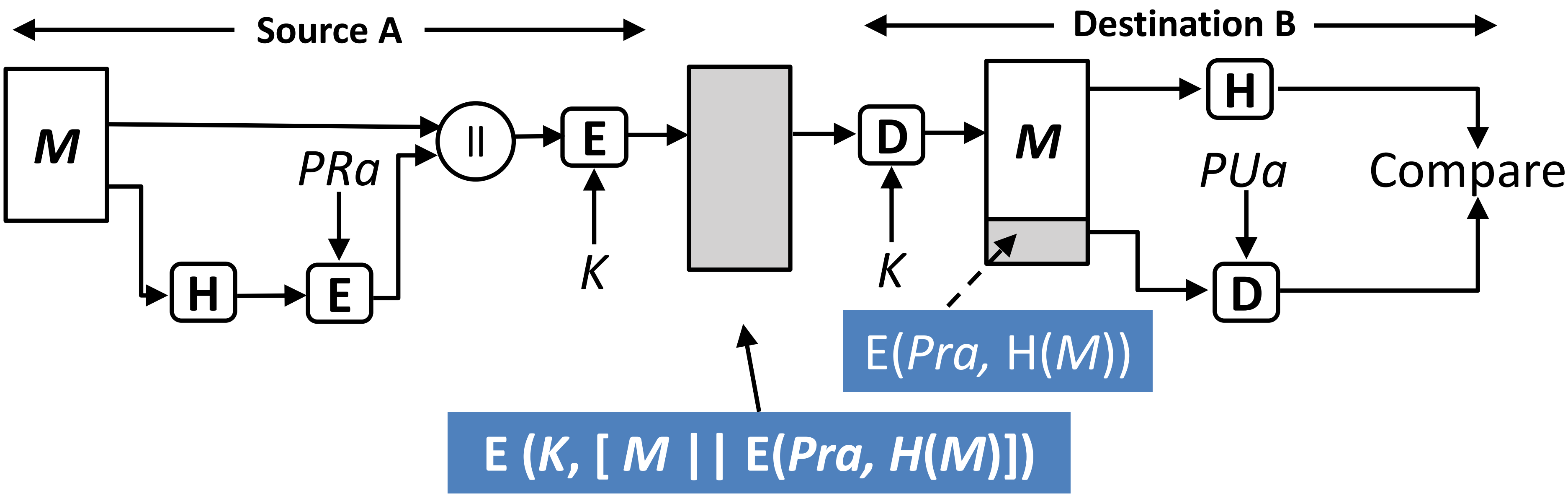
- A **digital signature** is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.
- The operation of the digital signature is similar to that of the **MAC**.
- In the case of the **digital signature**, the hash value of a message is encrypted with a user’s **private key**.
- Anyone who knows the user’s **public key** can verify the integrity of the message that is associated with the **digital signature**.

Digital Signature method - 1



- The hash code is encrypted, using the sender’s private key and decrypted using sender’s public key.
- This provides authentication.
- It also provides a **digital signature**, because only the sender could have produced the encrypted hash code.

Digital Signature method - 2



- If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key.

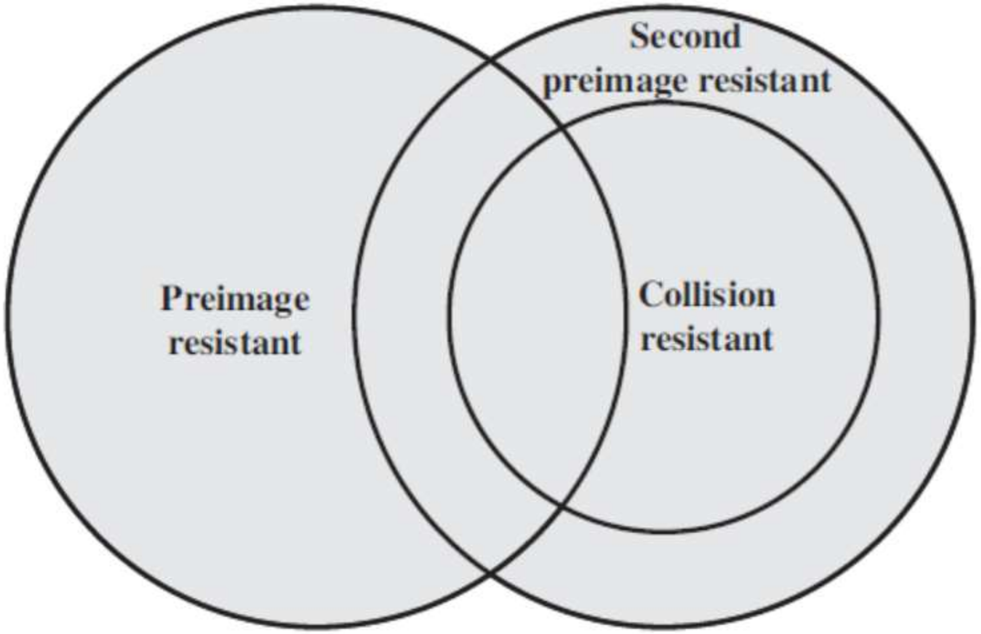
Security Requirements

1. Disclosure
2. Traffic analysis
3. Masquerade
4. Content modification
5. Sequence modification
6. Timing modification
7. Source repudiation
8. Destination repudiation

Properties Required for Cryptographic Hash Functions

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.

Properties Required for Cryptographic Hash Functions



Requirements for hash functions

1. Can be applied to any *sized message M*.
2. Produces *fixed-length output h*.
3. It is *easy to compute h=H(M)* for any *message M*.
4. Given hash value *h* is *infeasible* to find *y* such that ($H(y) = h$)
   - *One-way property*
5. For given block *x*, it is computational infeasible to find

   $y \neq x$ with $H(y) = H(x)$
   - *Weak collision resistance*
6. It is computationally infeasible to find messages *m1* and *m2* with $H(m1) = H(m2)$
   - *Strong collision resistance*

Simple Hash Function

- The input (message, file, etc.) is viewed as a sequence of n-bit blocks.
- The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block.

$$\boldsymbol{C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}}$$

Where,

C_i = i^{th} bit of the hash code $1 \le i \le n$

m = number of n-bit blocks in the input

b_{ij} = i^{th} bit in j^{th} block

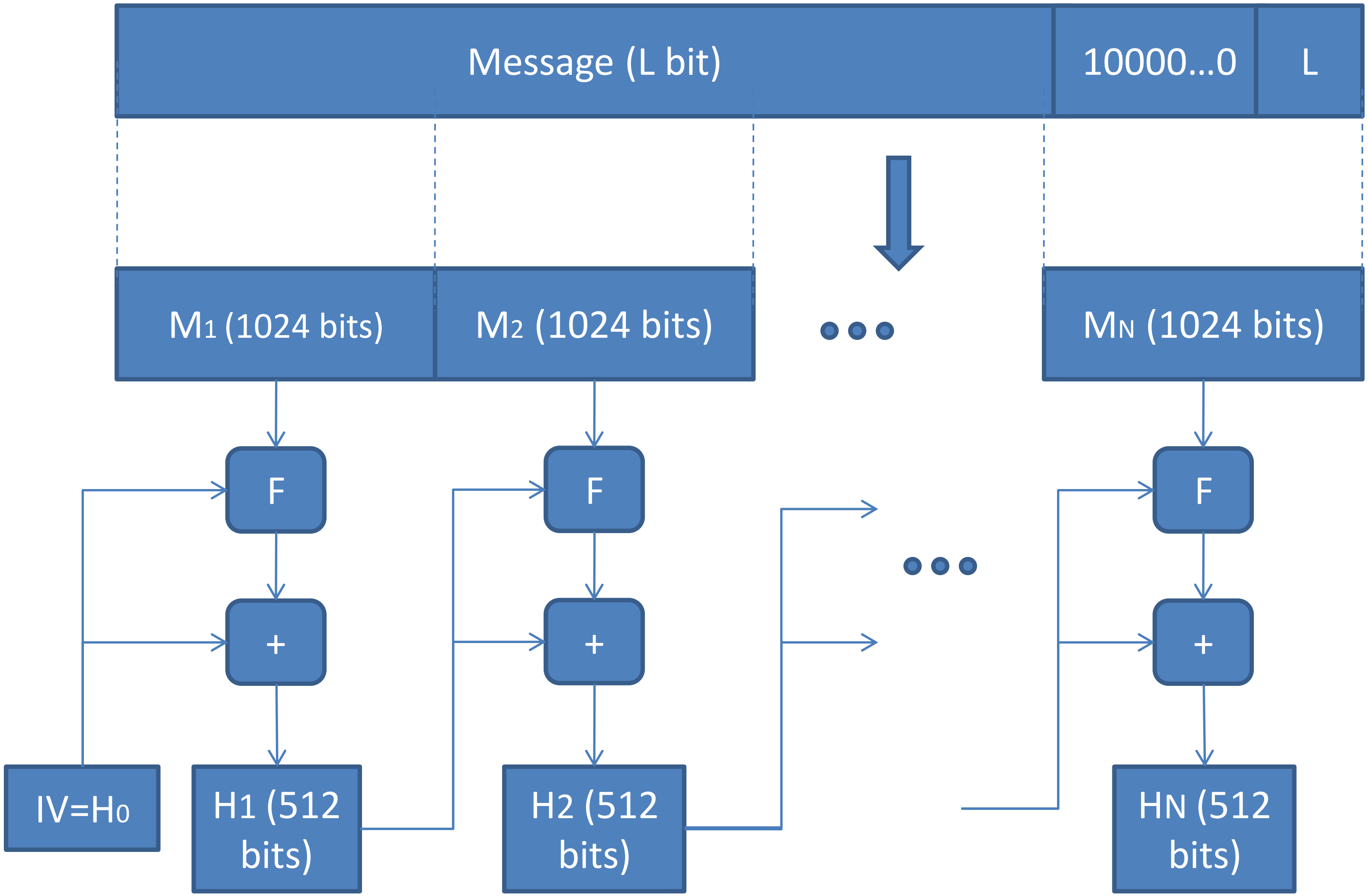
SHA - Secure Hash Algorithm

	SHA - 1	SHA - 224	SHA - 256	SHA - 384	SHA - 512
Message Digest Size	**160**	**224**	**256**	**384**	**512**
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	**512**	**512**	**512**	**1024**	**1024**
Word Size	**32**	**32**	**32**	**64**	**64**
Number of Steps	**80**	**64**	**64**	**80**	**80**

SHA - 512

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a **512-bit** message digest.
- The input is processed in **1024-bit** blocks.

Message Digest Generation using SHA -512



Step -1 Append Padding Bits

- The message is padded so that its length is congruent to **896 modulo 1024** [length ≡ 896(mod 1024)] .
- Padding is always added, even if the message is already of the desired length.
- Thus, the number of padding bits is in the range of **1 to 1024**.
- The padding consists of a single 1 bit followed by the necessary number of 0 bits.

Step -2 Append Length

- A block of 128 bits is appended to the message.
- This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the **length of the original message** (before the padding).

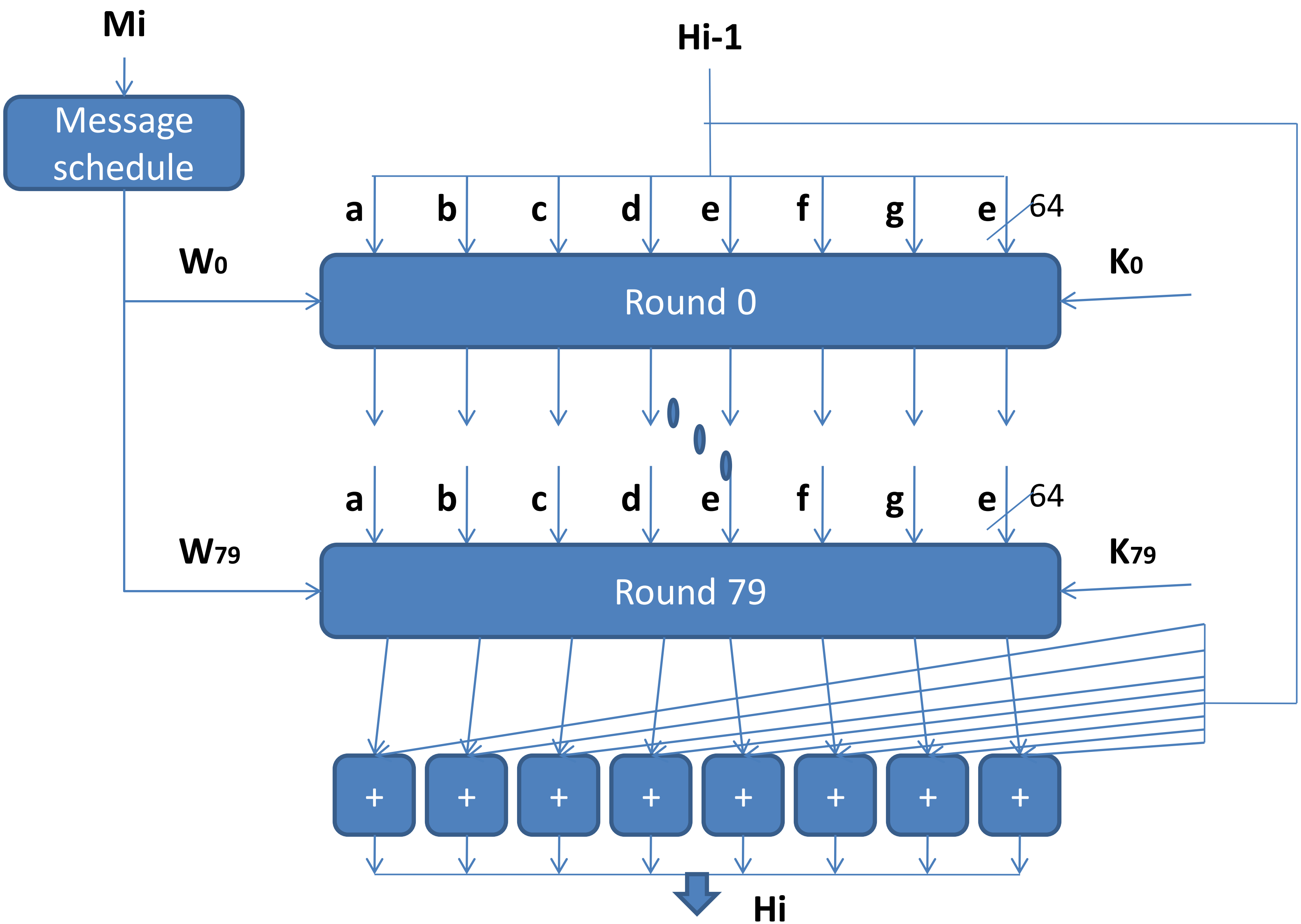
Step -3 Initialize hash buffer

- The outcome of the first two steps produces a message that is an integer multiple of 1024 bits in length.
- the expanded message is represented as the sequence of 1024-bit blocks $\mathbf{M_1}$ **,** $\mathbf{M_2}$**, … ,** $\mathbf{M_N}$, so that the total length of expanded message is N X 1024 bits.
- A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as **eight 64-bit registers (a, b, c, d, e, f, g, h)**.

Step -4 Process message in 1024-bit (128-word) blocks

- The heart of the algorithm is a module that consists of **80 rounds**; this module is labelled F

SHA-512 Processing of a Single 1024-Bit Block



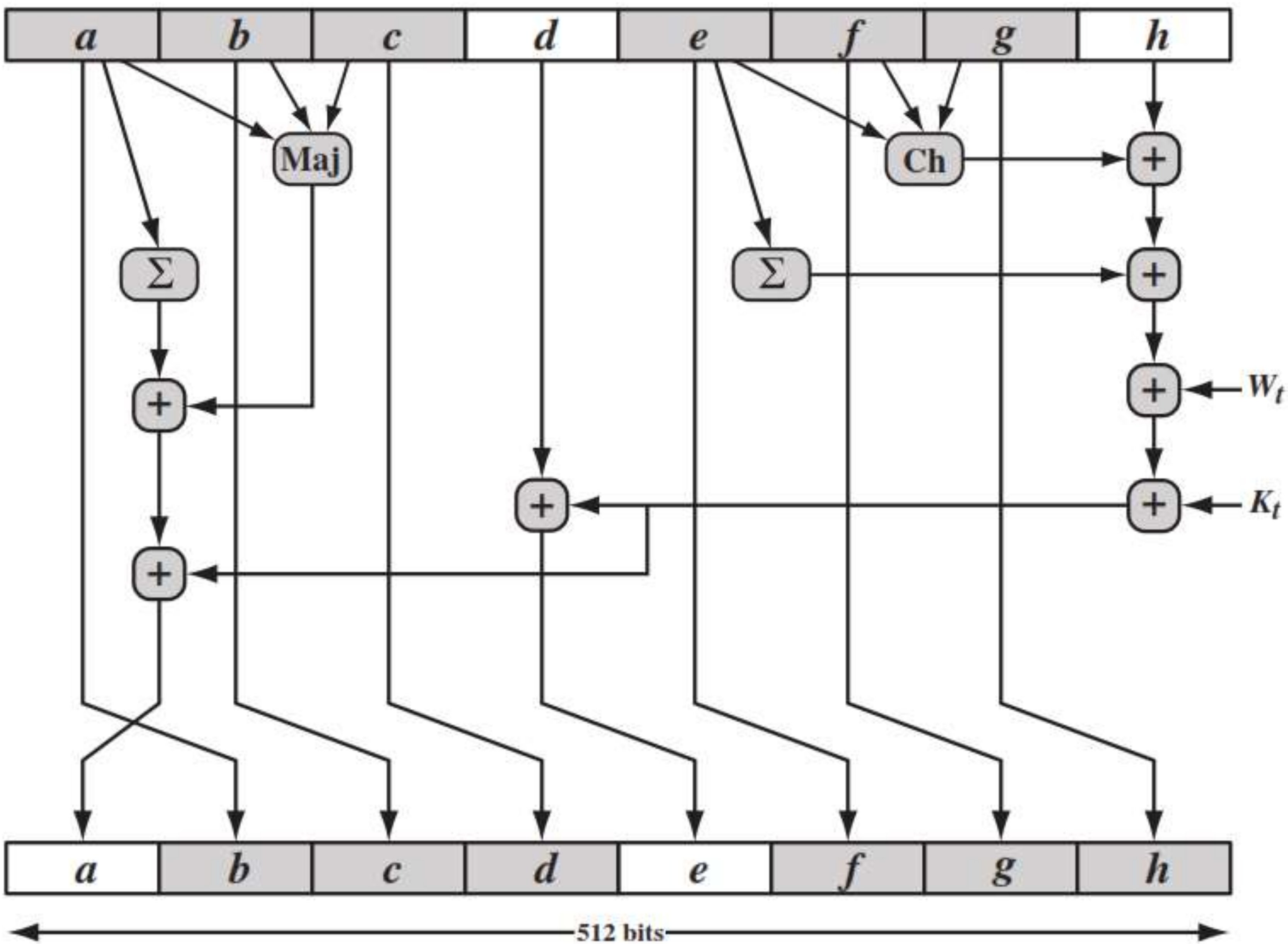
SHA-512 Processing of a Single 1024-Bit Block

- Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, $\boldsymbol{H_{i\text{-}1}}$.
- Each round ***t*** makes use of a **64-bit value Wt**, derived from the current 1024-bit block being processed.
- The output of the eightieth round is added to the input to the first round **(H_i-1)** to produce $\boldsymbol{H_i}$.

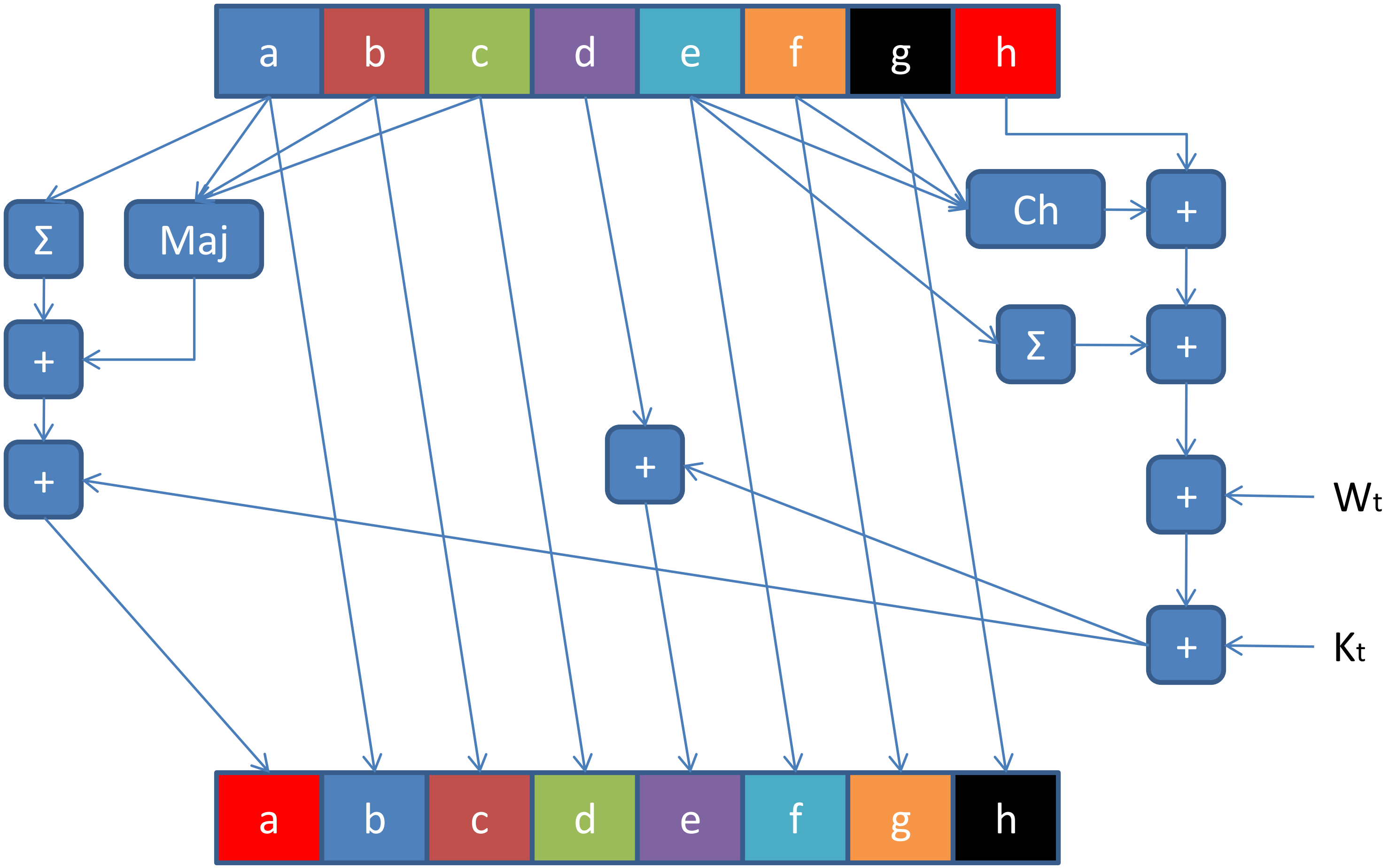
Step – 5 Output

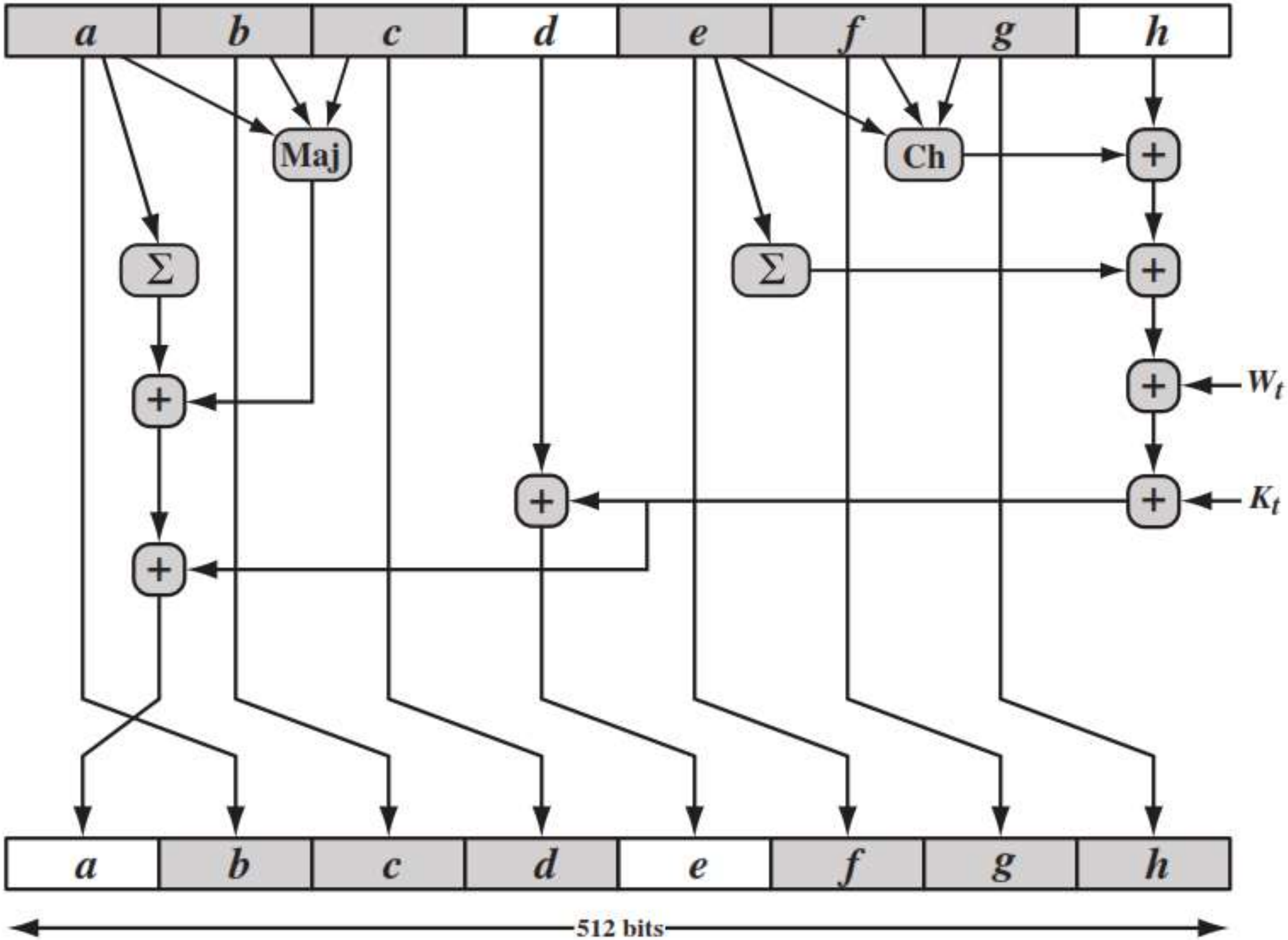
- After all ***N*** **1024-bit** blocks have been processed, the output from the ***N*th** stage is the **512-bit** message digest

SHA-512 Round Function



SHA-512 Round Function – Cont…





$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$

$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

$$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum\nolimits_0^{512} a\right) + \mathrm{Maj}(a, b, c)$$

SHA-512 Round Function Elements

- Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c) Majority of arguments are true
- ∑(a) = ROTR(a,28) **XOR** ROTR(a,34) **XOR** ROTR(a,39)
- ∑(e) = ROTR(e,14) **XOR** ROTR(e,18) **XOR** ROTR(e,41)
- + = addition modulo 2^{64}
- K_t = a 64-bit additive constant
- W_t = a 64-bit word derived from the current 512-bit input block.