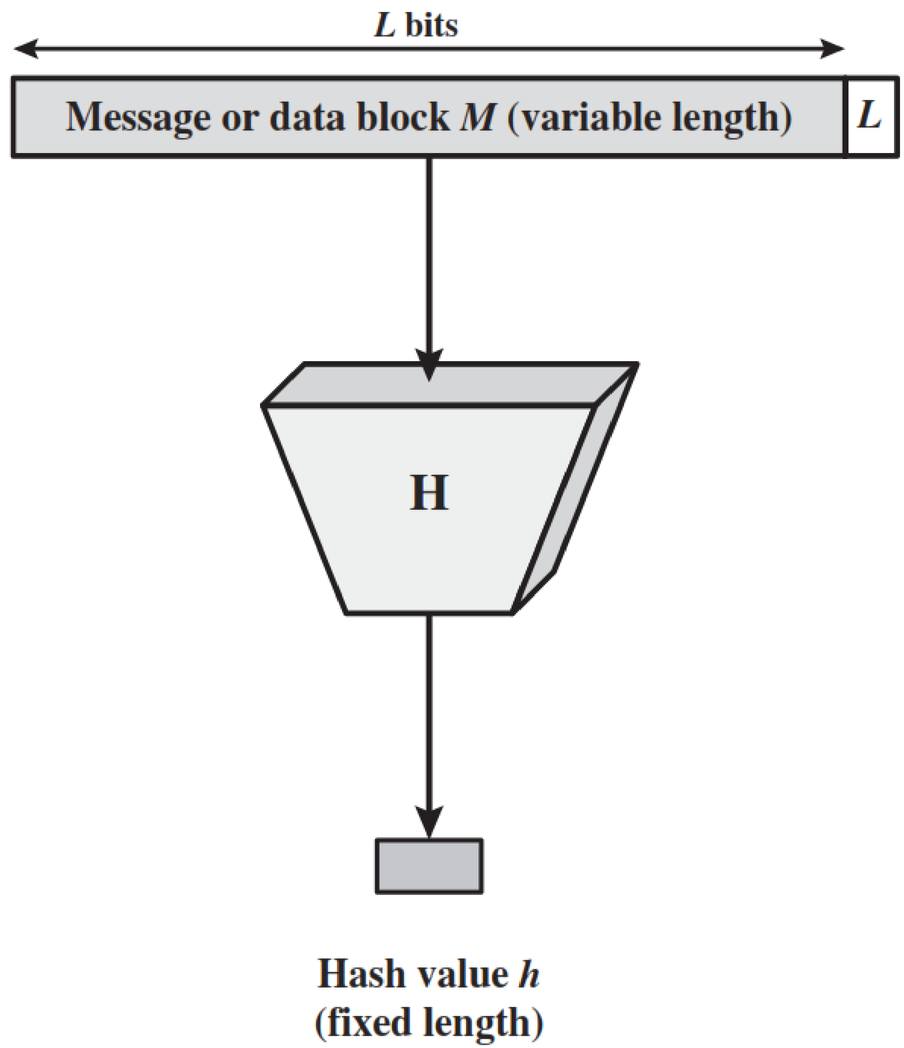
Cryptographic Hash Functions

Outline

- 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

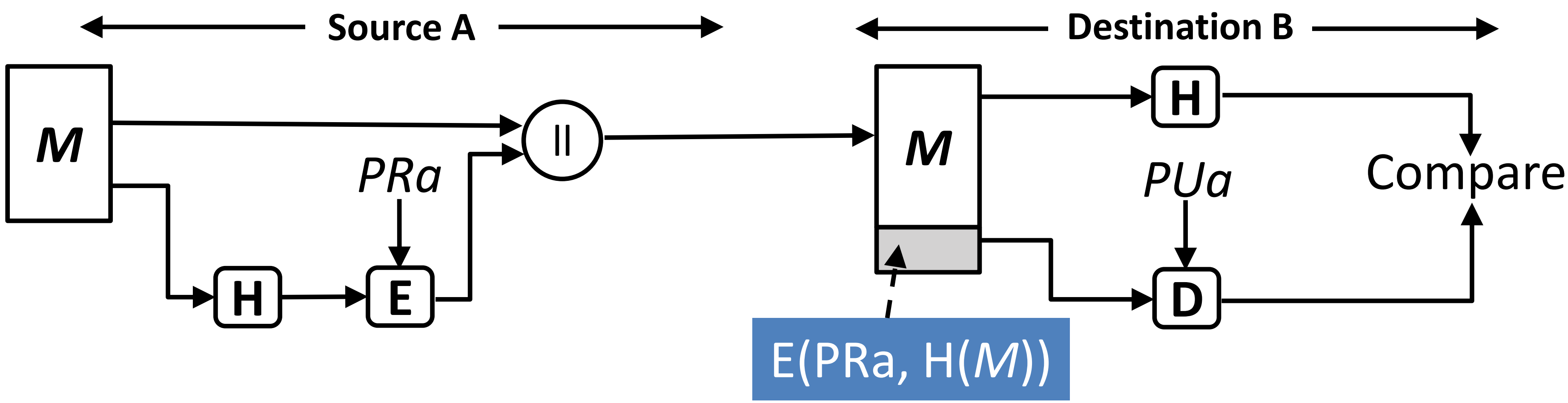
1. **Message authentication**
2. Digital Signature
3. One-way password file

Some topics of Unit 3/Unit 4 will be covered by the Expert

Digital Signature

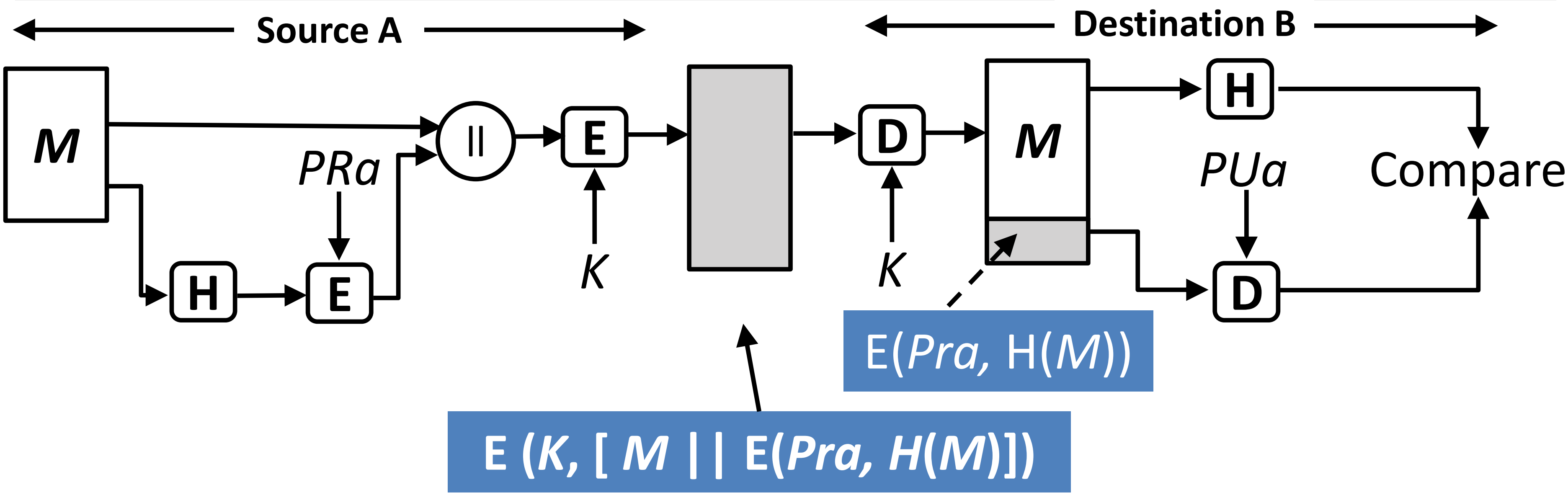
- A **digital signature** is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document.
- The operation of the digital signature is similar to that of the Message Authentication Code (**MAC**).
- In the case of the **digital signature**, the hash 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 public-key encryption with the sender's private 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

Requirements for hash functions

1. It can be applied to any *sized message M*.
2. It should produce *fixed-length output h*.
3. It should be *easy to compute h=H(M)* for any *message M*.
4. Given the hash value *h*, it is *infeasible* to find *y* such that (*H(y) = h*)
   - *One-way property*
5. For given block *x*, it is computationally infeasible to find *y*, *y* ≠ *x* with *H*(*y*) = *H*(*x*)
   - *Weak collision resistance (i.e. , it's hard to find another input 'y' that produces the same hash output)*
6. It is computationally infeasible to find messages *m*1 and *m*2 where their hash values are equal i.e. *H*(*m*1) = *H*(*m*2)
   - *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.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

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

- where

C_i = *i*th bit of the hash code, 1 … *i* … n

m = number of *n*-bit blocks in the input

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

$\oplus$ = XOR operation

SHA

- In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA) as virtually every other widely used hash function had been found to have substantial cryptanalytic weaknesses
- SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993.
- When weaknesses were discovered in SHA, different versions had been developed subsequently.

SHA - Secure Hash Algorithm

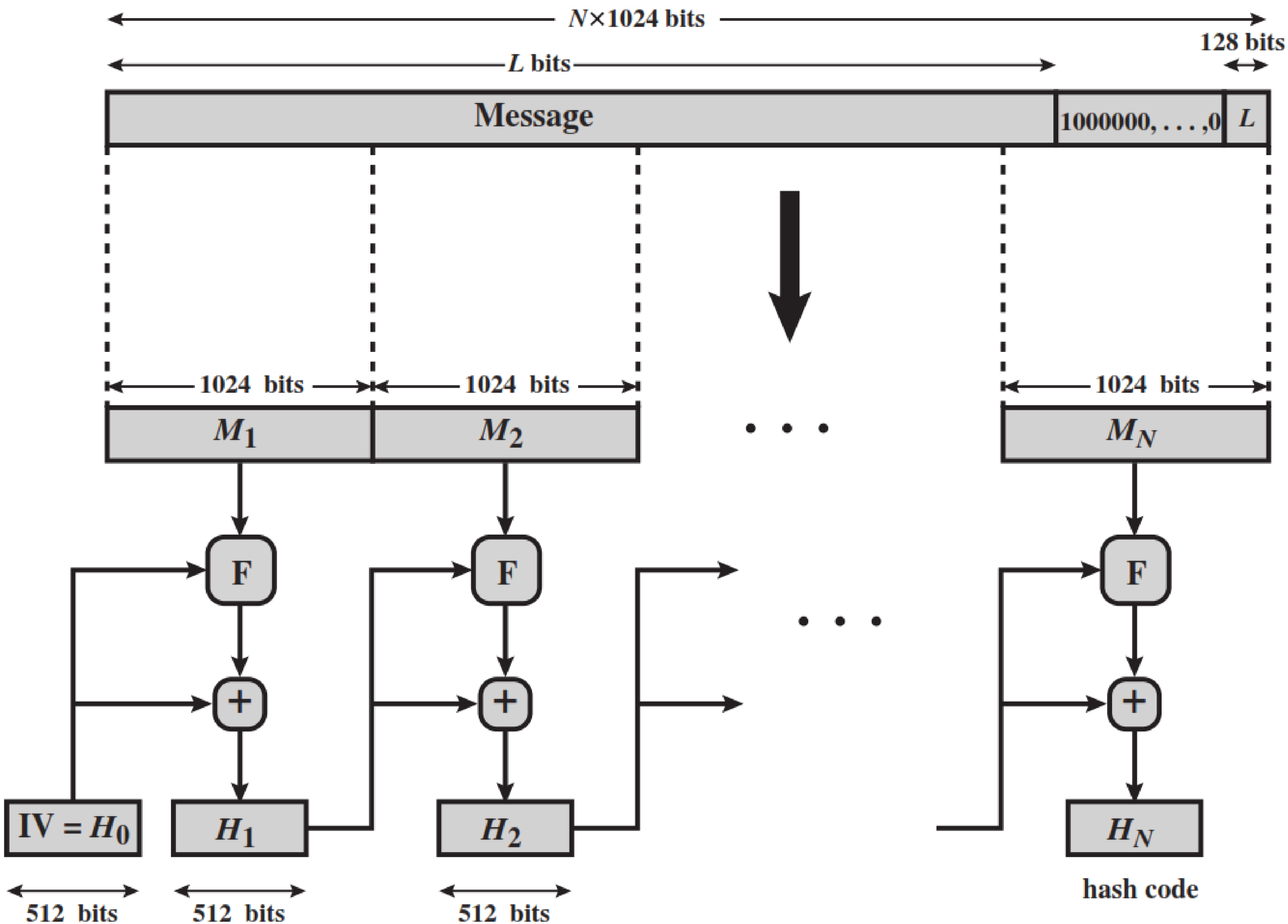
	SHA - 1	SHA - 224	SHA - 256	SHA - 384	SHA - 512
Message Digest Size (bits)	160	224	256	384	512
Message Size (bits)	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size (bits)	512	512	512	1024	1024
Word Size (bits)	32	32	32	64	64
No. of Steps	80	64	64	80	80

- Message digest=> Hash Value
- There is technically a limit for Message Size as per the padding scheme requirement

SHA - 512

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

Message Digest Generation using SHA -512



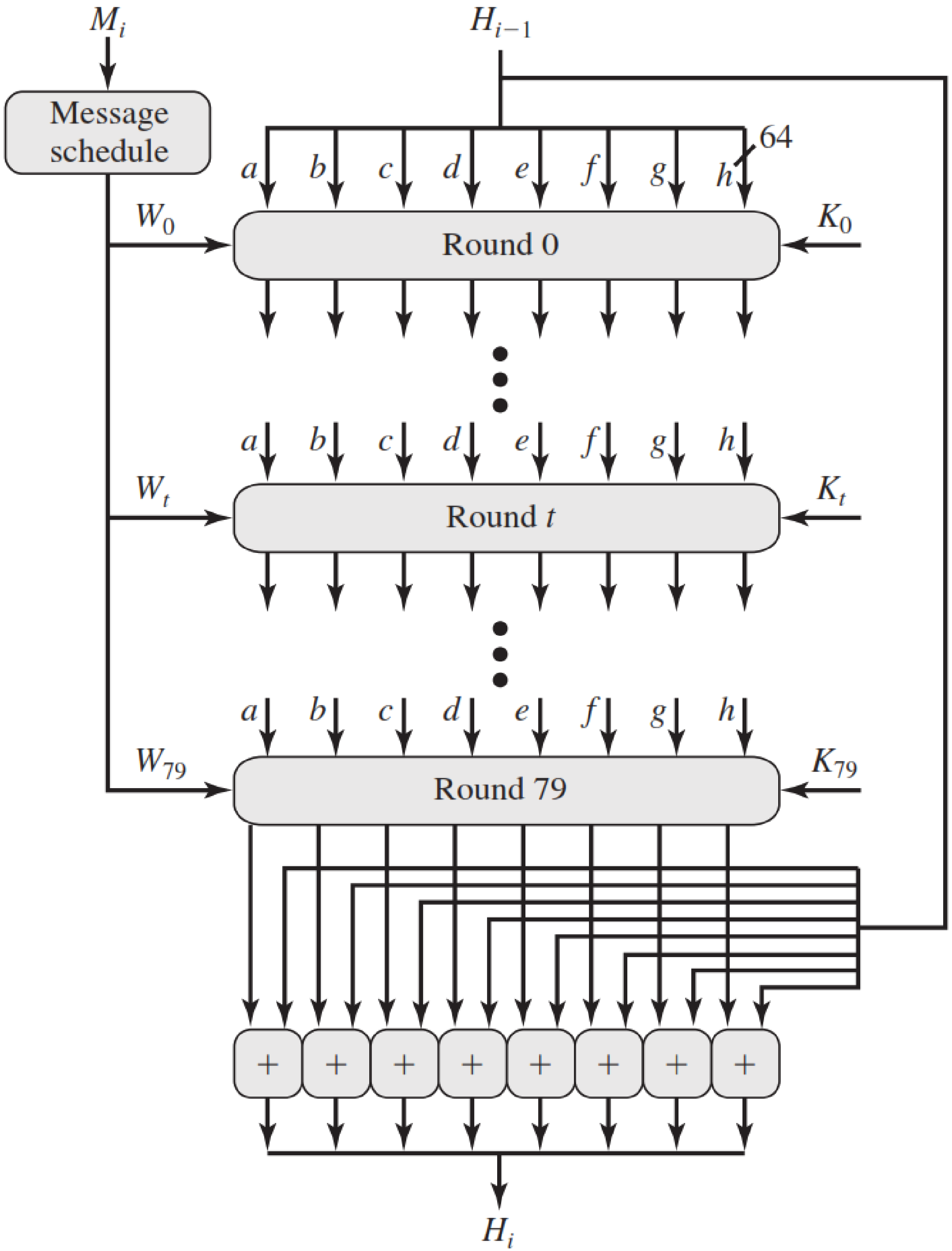
Step -1 Append Padding Bits

- The message is padded so that its length is congruent to **896 modulo 1024** [length ≡ 896(mod 1024)] . (To keep space for Appending Message Length-Check Step 2)
- 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).
- The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length.
- In Figure, the expanded message is represented as the sequence of 1024-bit blocks M_1, M_2, ..., M_N, so that the total length of the expanded message is N * 1024 bits.

SHA-512 Processing of a Single 1024-Bit Block



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 **M_1 , M_2, … , 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

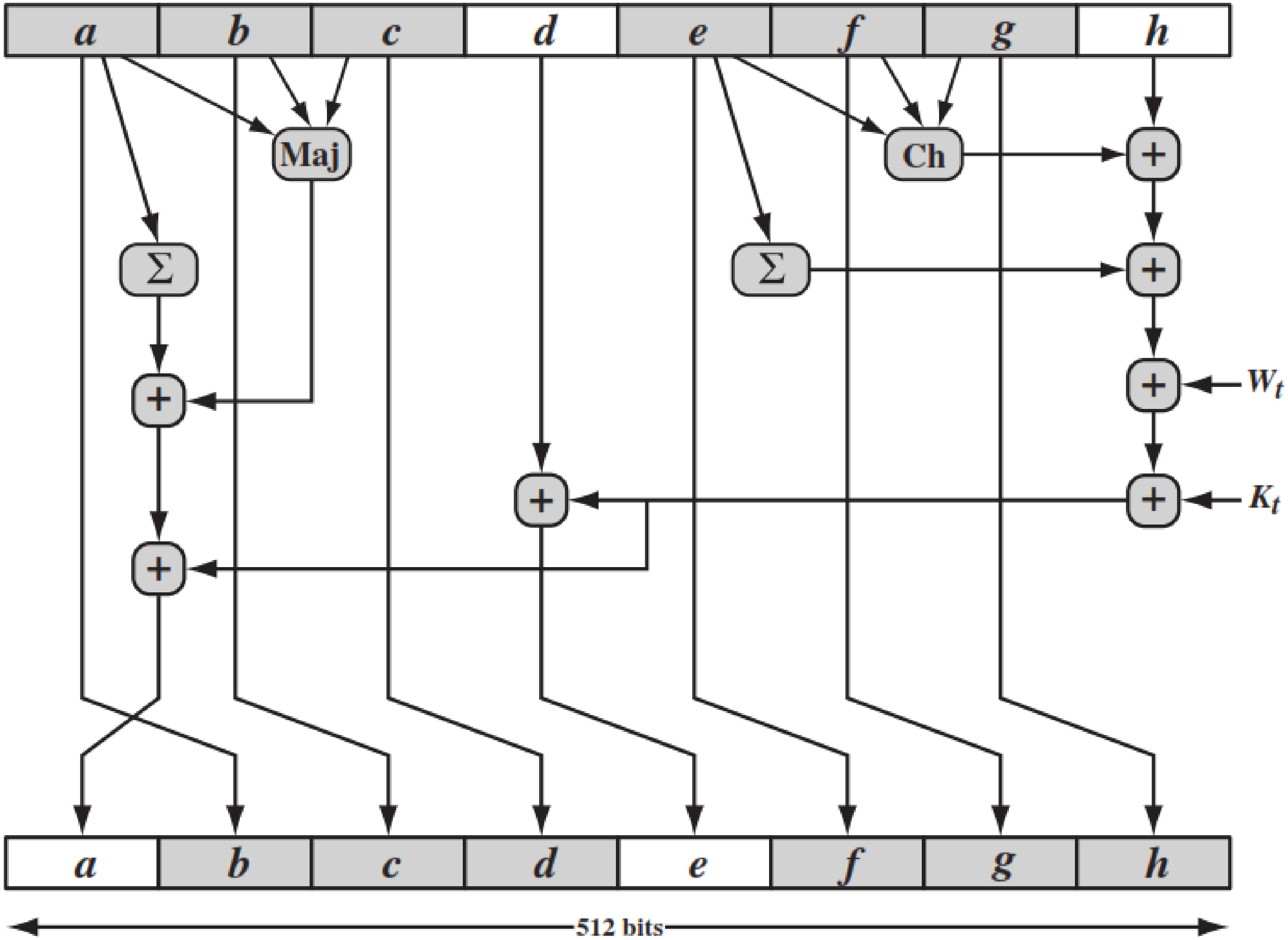
- The heart of the algorithm is a module that consists of **80 rounds**; this module is labelled F

SHA-512 Processing of a Single 1024-Bit Block

- 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 intermediate hash value, H_{i-1}.
- Each round *t* makes use of a **64-bit value** W_t, 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 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



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

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

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

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

SHA-512 Round Function

SHA-512 Round Function Elements

- Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c) returns a result based on majority value among these inputs
- ∑(a) = ROTR(a,28) **XOR** ROTR(a,34) **XOR** ROTR(a,39) (ROTR(a,28) means rotate right by 28 positions)
- ∑(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 input block.