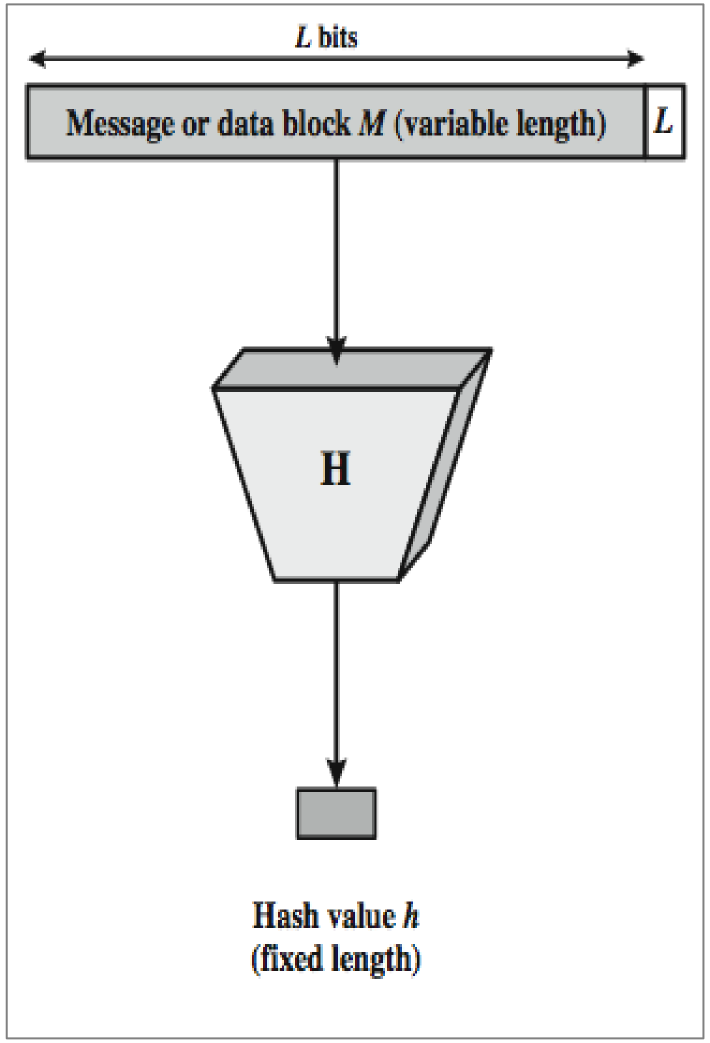
Data Security and Privacy
DSE 3258

L8 –Cryptographic Hash Functions

Hash Functions

- 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 the function to a large set of inputs will produce outputs that are evenly distributed and apparently random.
- The kind of hash function needed for security applications is referred to as ***a cryptographic hash function.***



Hash Functions (Cont..)

- usually assume hash function is public
- hash used to detect changes to message
- want a cryptographic hash function
  - computationally infeasible to find data mapping to specific hash (**one-way property)**
  - computationally infeasible to find two data to same hash (**collision-free property**)

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

- **Hash Functions & Message Authentication**

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

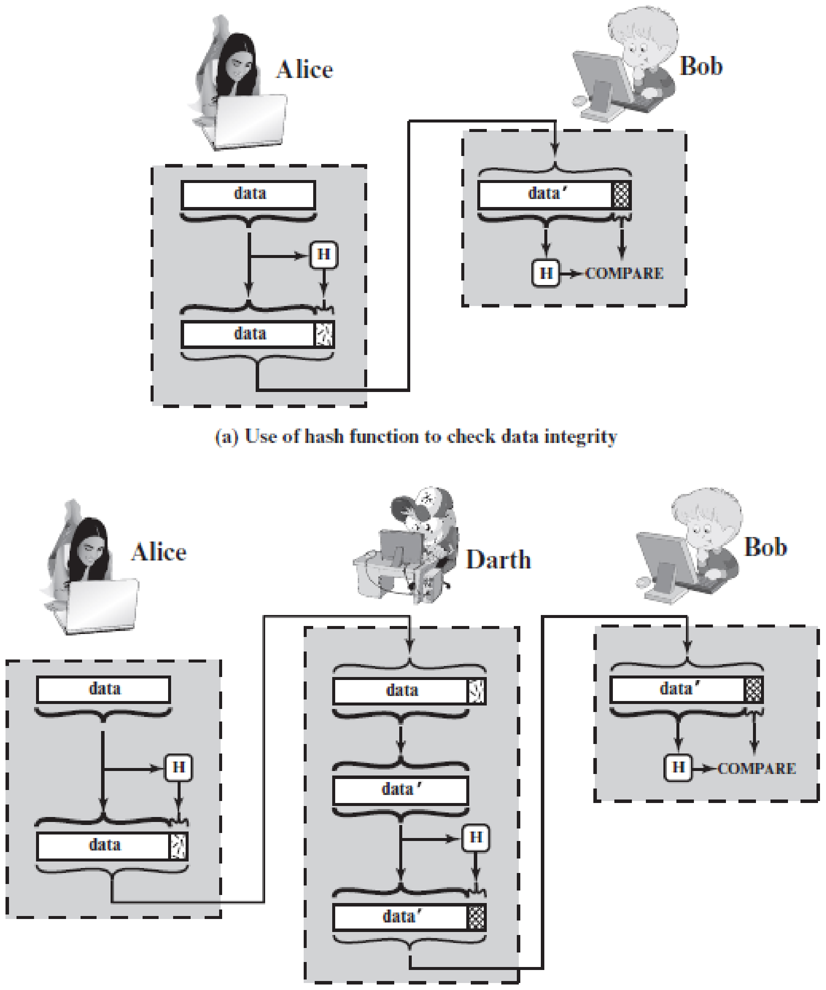
✓ Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).

✓ In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid.

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

Approach:

- The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message.
- The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
- If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered.



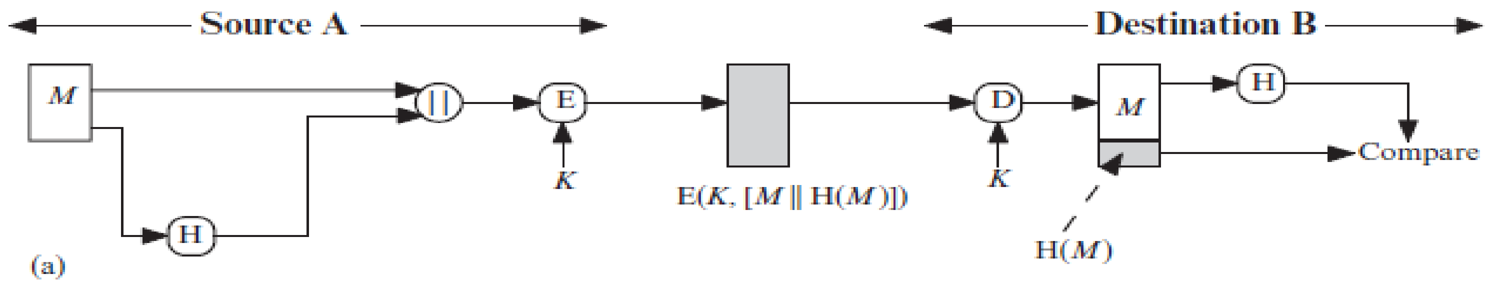
(a) Use of hash function to check data integrity

(b) Man-in-the-middle attack

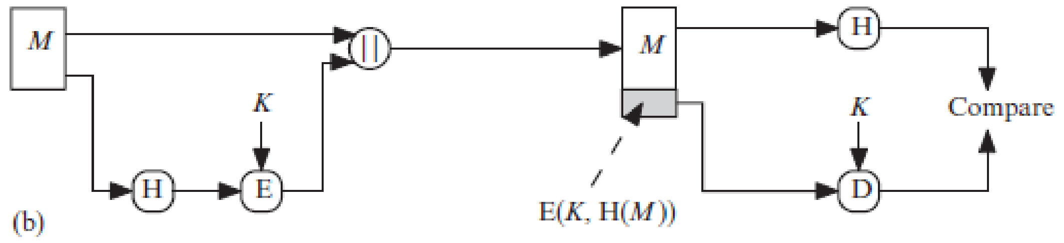
Figure 11.2 Attack Against Hash Function

The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver.

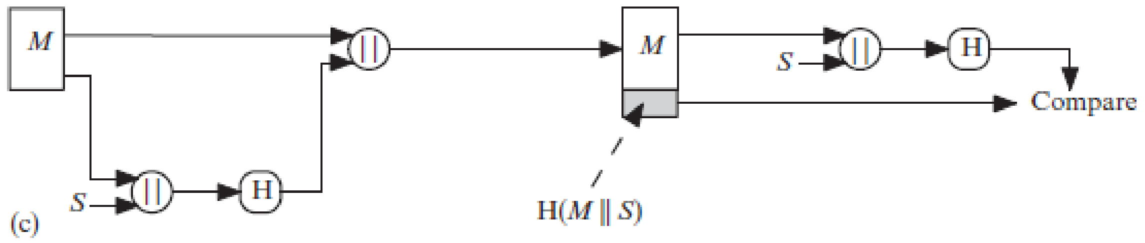
- **Different ways in which a hash code can be used to provide message authentication:**
- **a.** The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



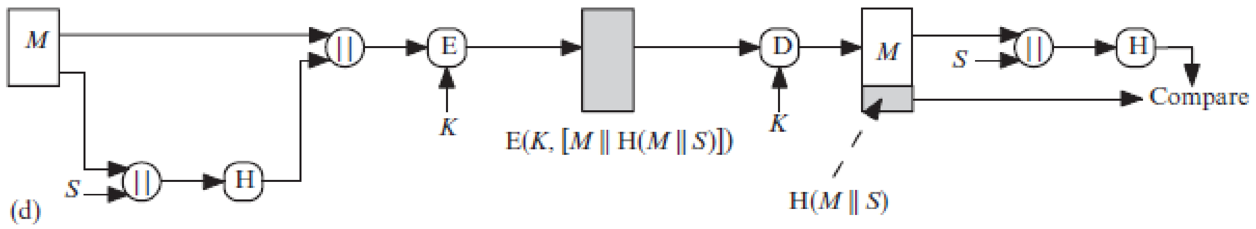
- **b.** Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



- **Different ways in which a hash code can be used to provide message authentication:**
- c. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties 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. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.



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



- More commonly, message authentication is achieved using a **message authentication code (MAC)**, also known as a **keyed hash function**.
- Typically, 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, which is associated with the protected message.
- If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value.
- An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key.

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

- **Hash Functions & Digital Signatures**
- 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.
- In this case, an attacker who wishes to alter the message would need to know the user's private key.
- **Different ways:**

a. The hash code is encrypted, using public-key encryption with the sender's private key. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

b. 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. This is a common technique.

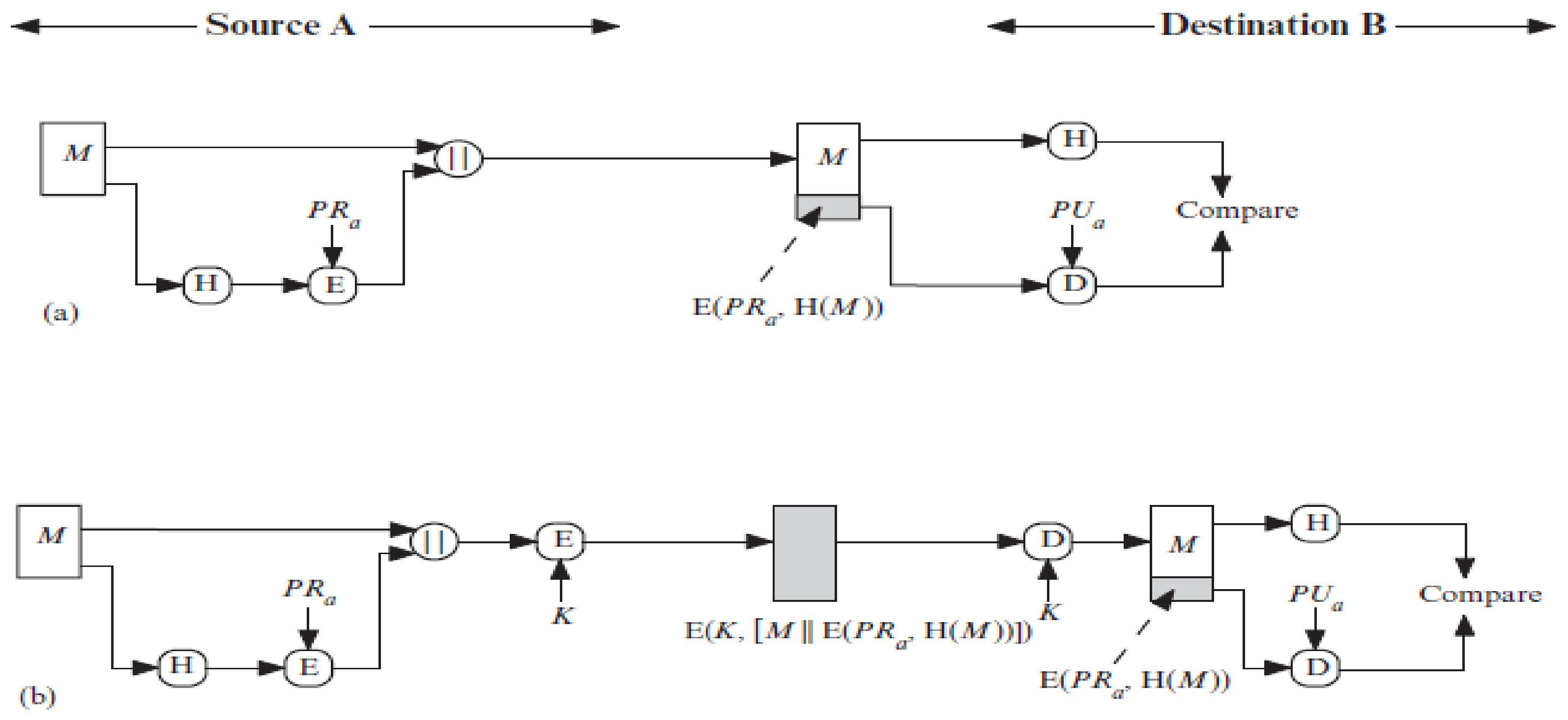


Figure 11.4 Simplified Examples of Digital Signatures

SIMPLE HASH FUNCTIONS

- **bit-by-bit exclusive-OR (XOR)**

Expressed as:

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

where

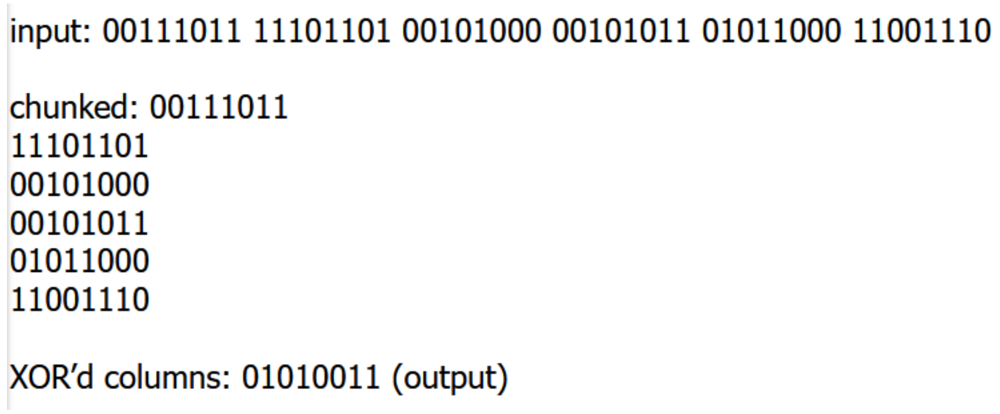
C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

- This operation produces a simple parity bit for each bit position and is known as a **longitudinal redundancy check**
- the probability that a data error will result in an unchanged hash value is 2^{-n}.
- in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2-128, the hash function on this type of data has an effectiveness of 2-112.



```
input: 00111011 11101101 00101000 00101011 01011000 11001110

chunked: 00111011
11101101
00101000
00101011
01011000
11001110

XOR’d columns: 01010011 (output)
```

SIMPLE HASH FUNCTIONS

Improve

- perform a one-bit circular shift, or rotation , on the hash value after each block is processed.
- 1. Initially set the n-bit hash value to zero.
- 2. Process each successive n-bit block of data as follows:
    - a. Rotate the current hash value to the left by one bit.
    - b. XOR the block into the hash value.

- This has the effect of “randomizing” the input more completely and overcoming
- any regularities that appear in the i

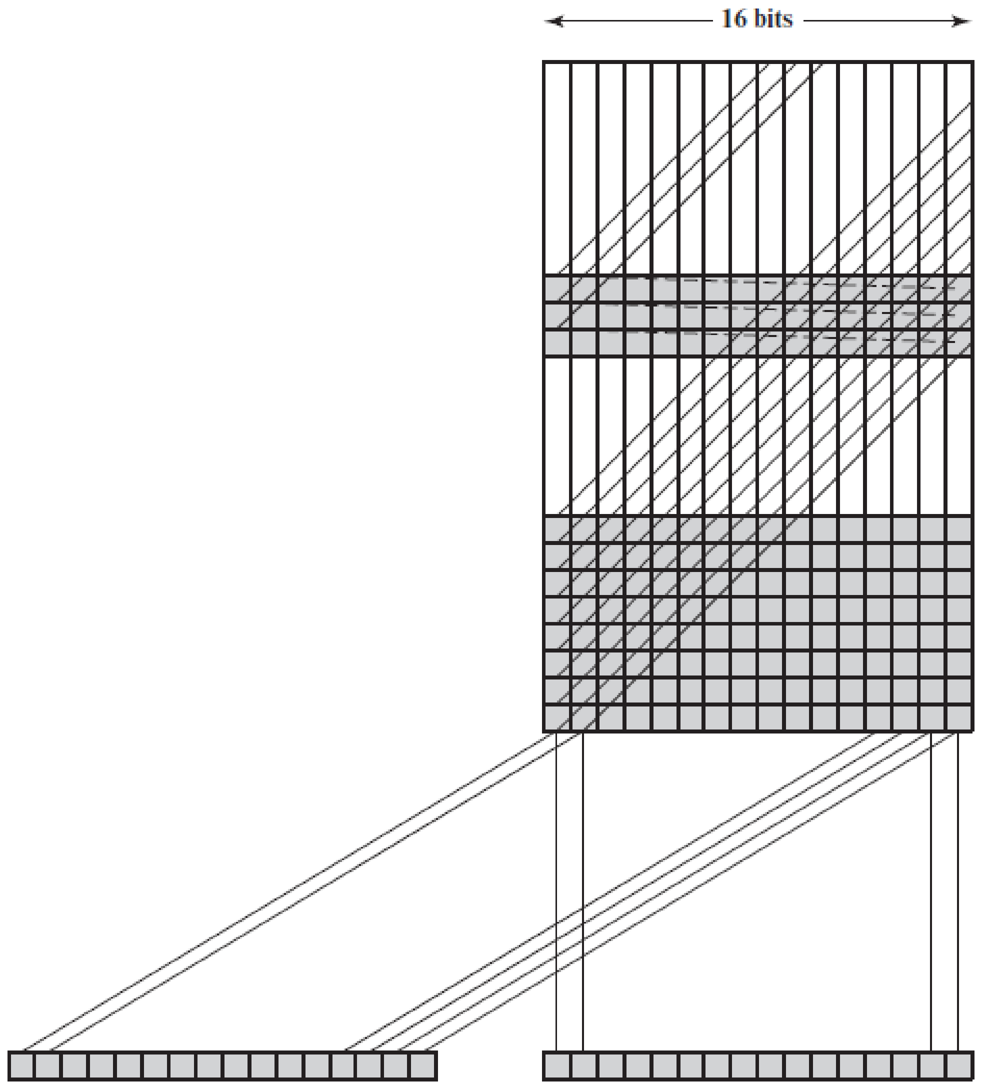
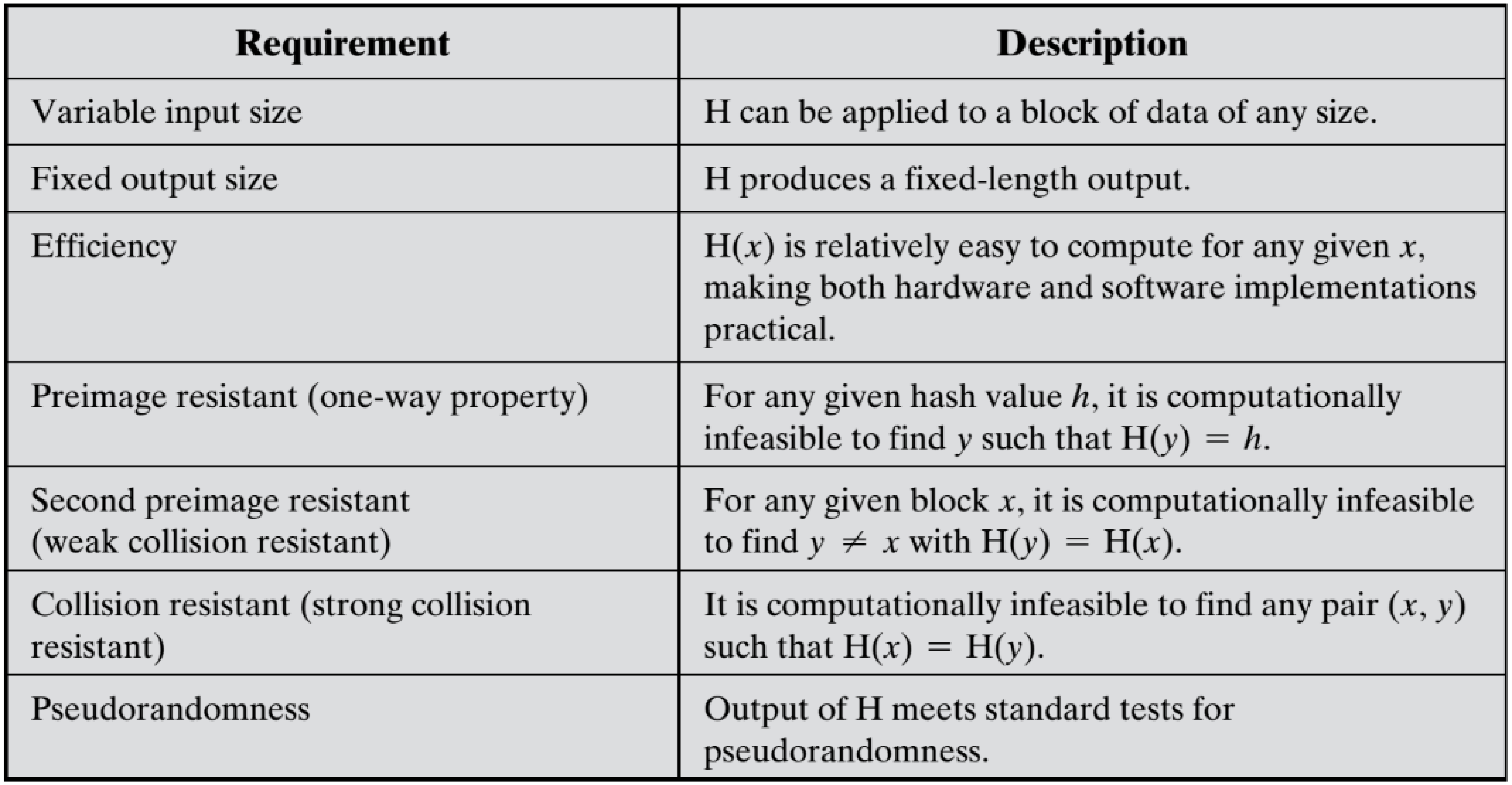

16 bits
XOR with 1-bit r otation to the right
XOR of every 16-bit block

Figure 11.5 Two Simple Hash Functions

Requirements

- For a hash value $h = \mathrm{H}(x)$, we say that x is the **preimage** of h. That is, x is a data block whose hash value, using the function H, is h.
- Because H is a many-to-one mapping, for any given hash value h, there will in general be multiple preimages. A **collision** occurs if we have $x \neq y$ and $\mathrm{H}(x) = \mathrm{H}(y)$
- Suppose the length of the hash code is n bits, and the function H takes as input messages or data blocks of length b bits with $b > n$. Then, the total number of possible messages is 2^b and the total number of possible hash values is 2^n. On average, each hash value corresponds to 2^{b-n} preimages.

Requirements for a Cryptographic Hash Function



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.

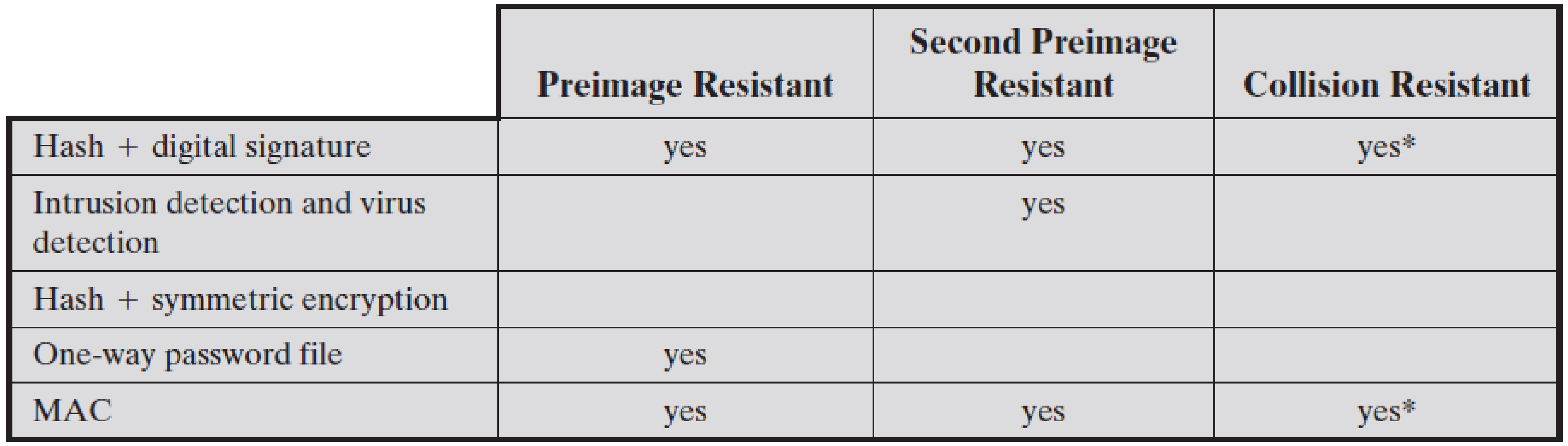


Table 11.2 Hash Function Resistance Properties Required for Various Data Integrity Applications

	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*

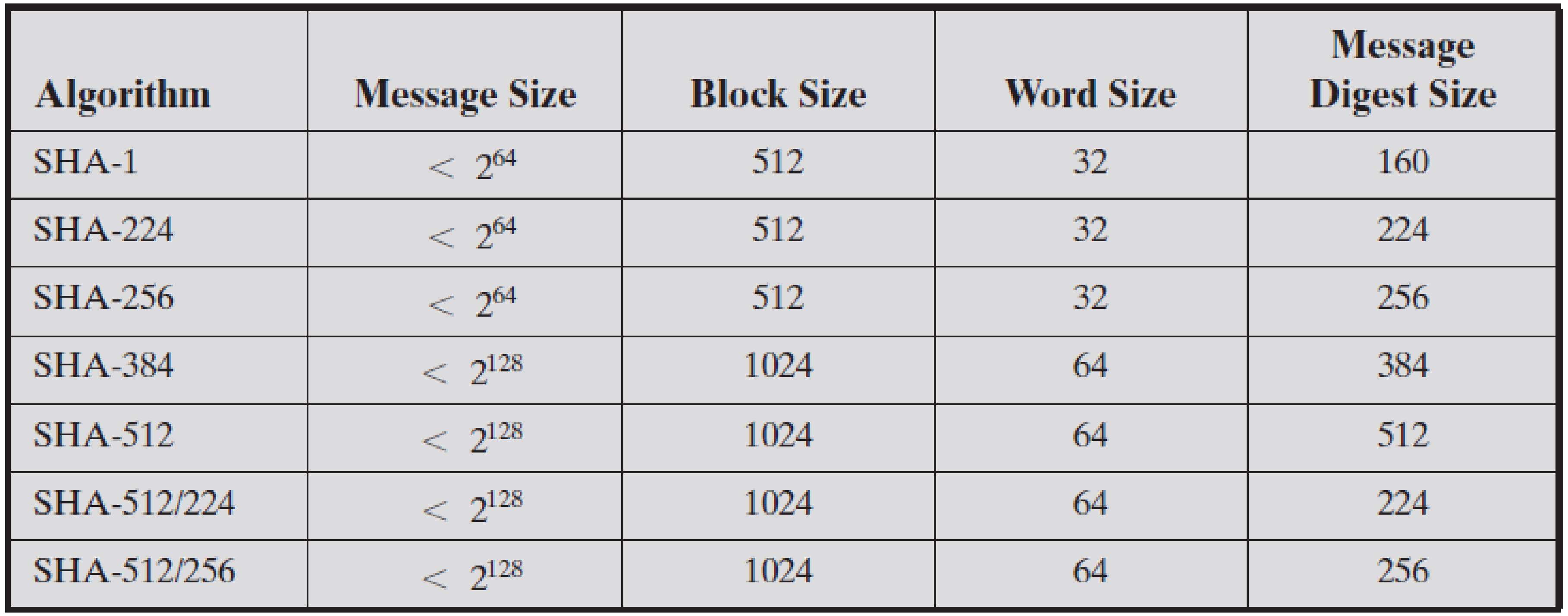
*Resistance required if attacker is able to mount a chosen message attack

Secure Hash Algorithm (SHA)

- 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, now known as **SHA-0**, a revised version was issued as FIPS 180-1 in 1995 and is referred to as **SHA-1**.
- The actual standards document is entitled “Secure Hash Standard.”

Secure Hash Algorithm (SHA)

Table 11.3 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

Note: All sizes are measured in bits.

SHA-512 Logic

- 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.

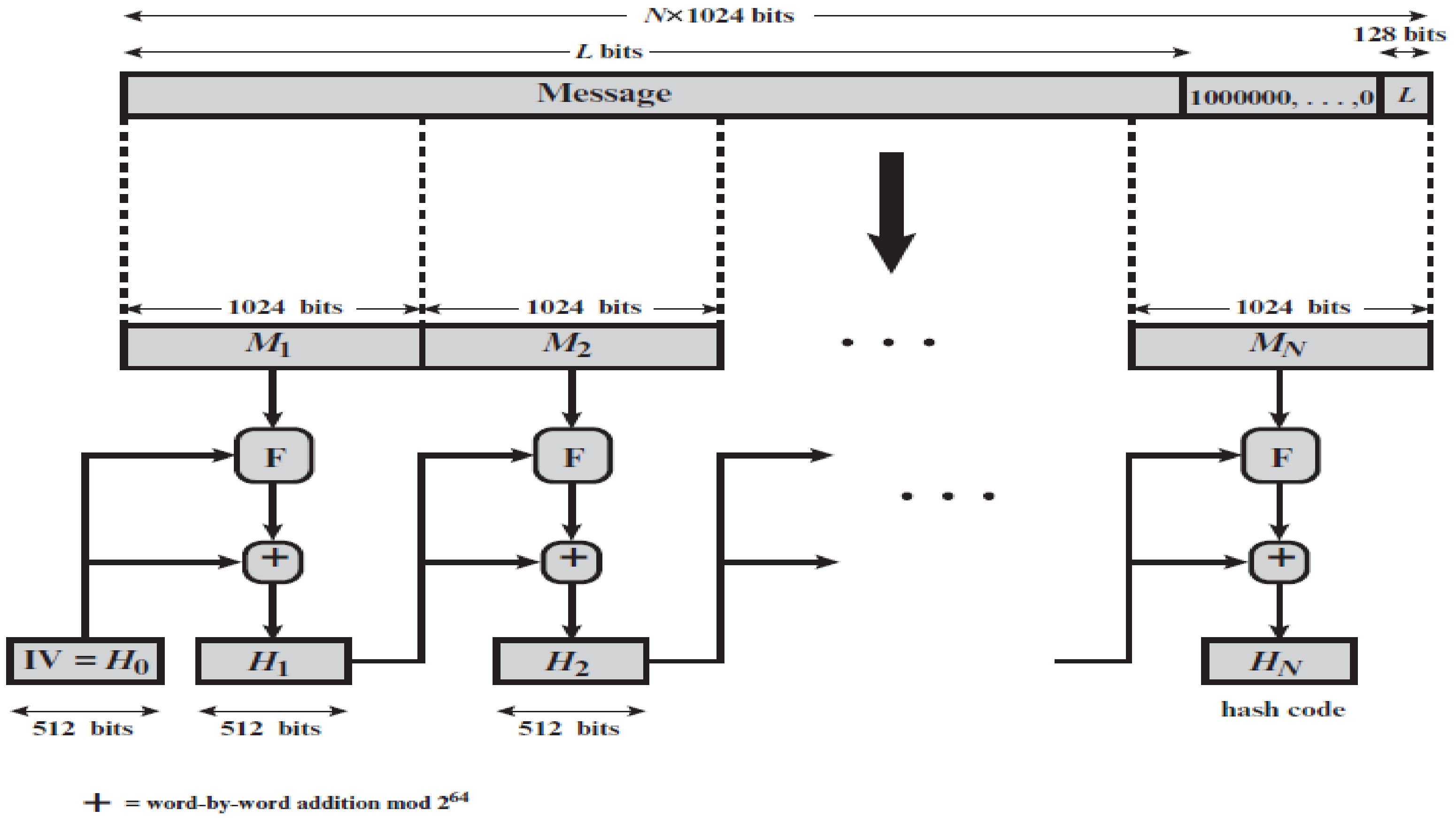


Figure 11.9 Message Digest Generation Using SHA-512

SHA-512 Logic

- **Step 1: Append padding bits.**

The message is padded so that its length is congruent to 896 modulo 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 and contains the length of the original message in bits (before the padding). The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length.

- **Step 3 Initialize hash buffer.**

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). These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1
b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

SHA-512 Logic

- **Step 4 Process message in 1024-bit (128-byte) blocks.**

The heart of the algorithm is a module that consists of 80 rounds

- **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.

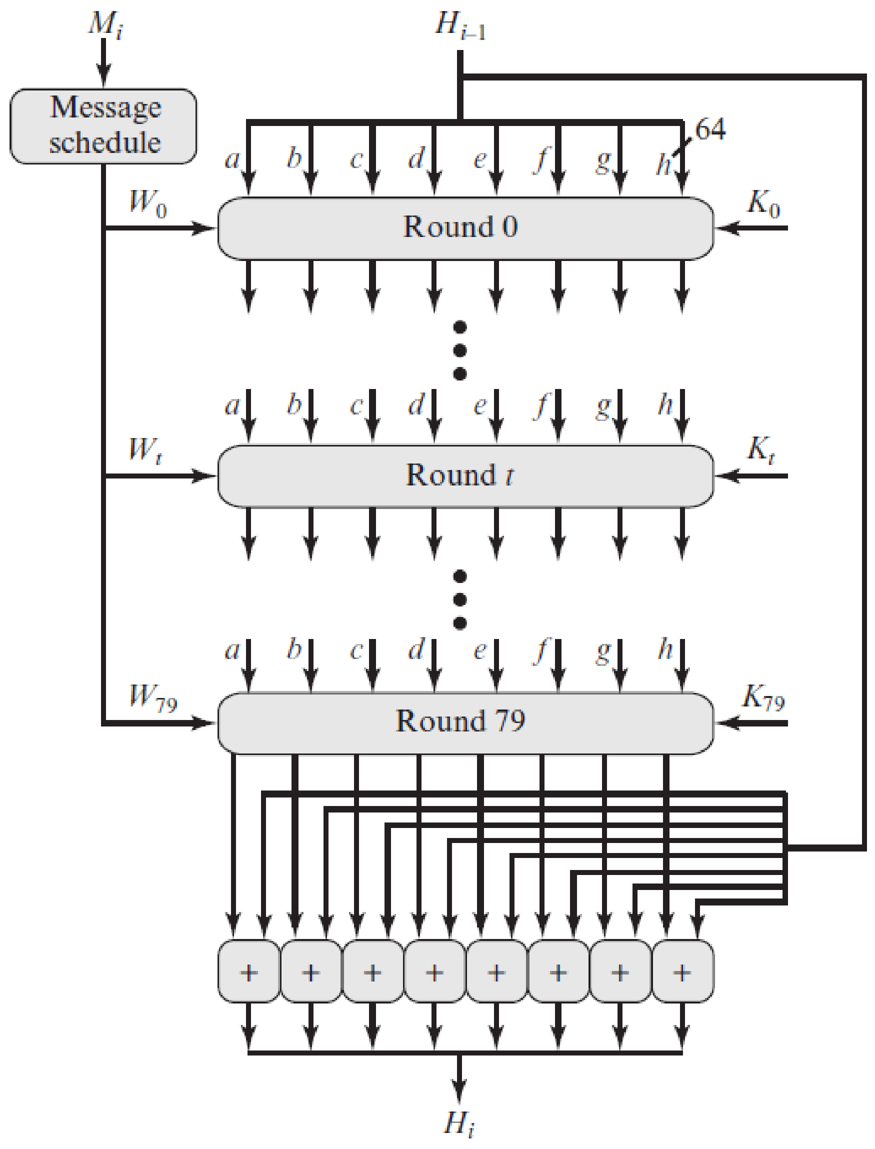
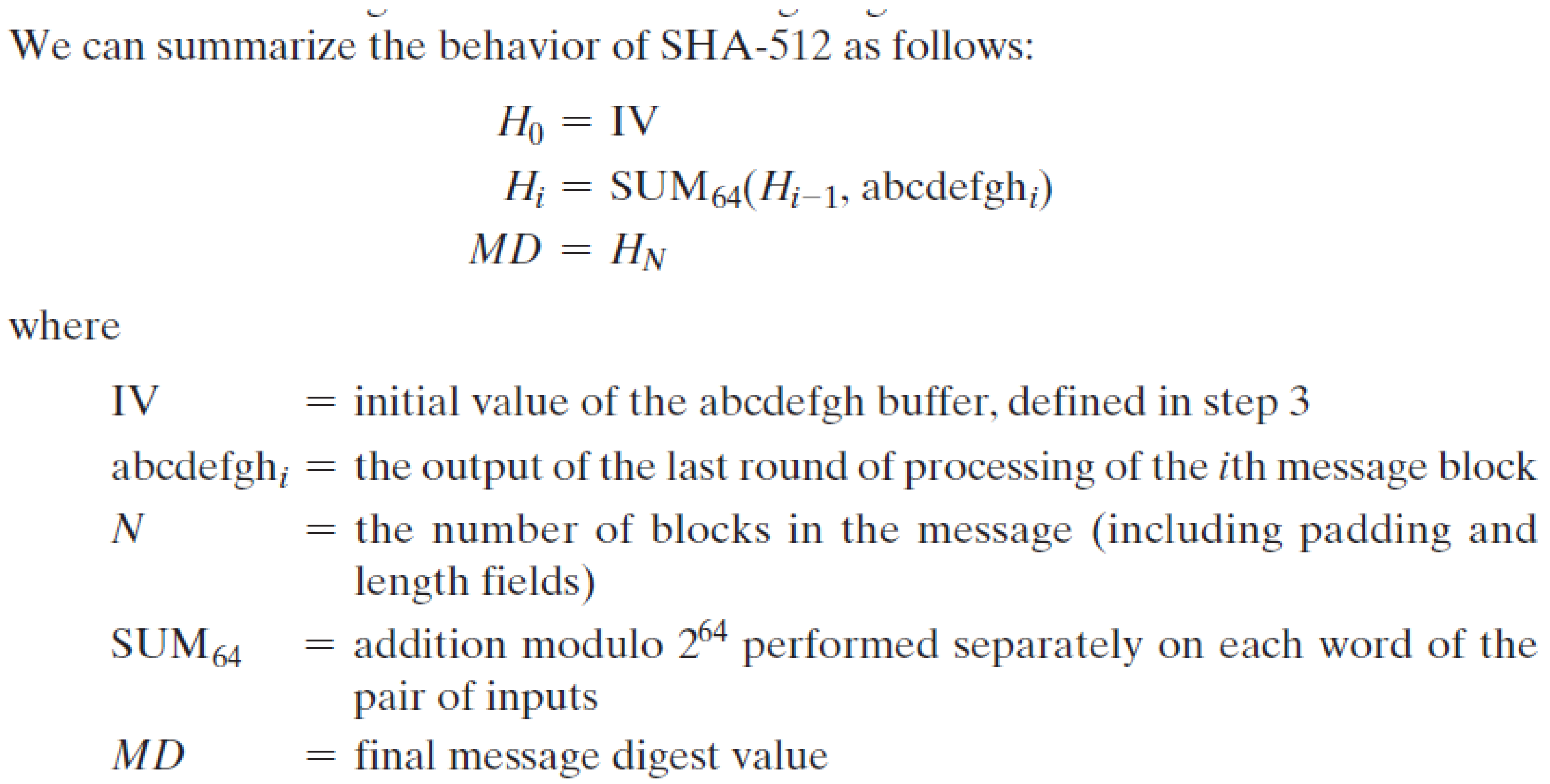


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

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, $H_{i\text{-}1}$.
Each round *t* makes use of a 64-bit value W_t, derived from the current 1024-bit block being (M_i).
These values are derived using a message schedule.
Each round also makes use of an additive constant K_t, where 0 ... *t* ... 79 indicates one of the 80 rounds.
The output of the eightieth round is added to the input to the first round (*Hi*-1) to produce *Hi*.
The addition is done independently for each of the eight words in the buffer with each of the corresponding words in *Hi*-1, using addition modulo 2^{64}.



We can summarize the behavior of SHA-512 as follows:

$$\begin{aligned} H_0 &= \mathrm{IV} \\ H_i &= \mathrm{SUM}_{64}(H_{i-1}, \mathrm{abcdefgh}_i) \\ MD &= H_N \end{aligned}$$

where

IV	=	initial value of the abcdefgh buffer, defined in step 3
$\mathrm{abcdefgh}_i$	=	the output of the last round of processing of the *i*th message block
N	=	the number of blocks in the message (including padding and length fields)
SUM_{64}	=	addition modulo 2^{64} performed separately on each word of the pair of inputs
MD	=	final message digest value

SHA-512 Round Function

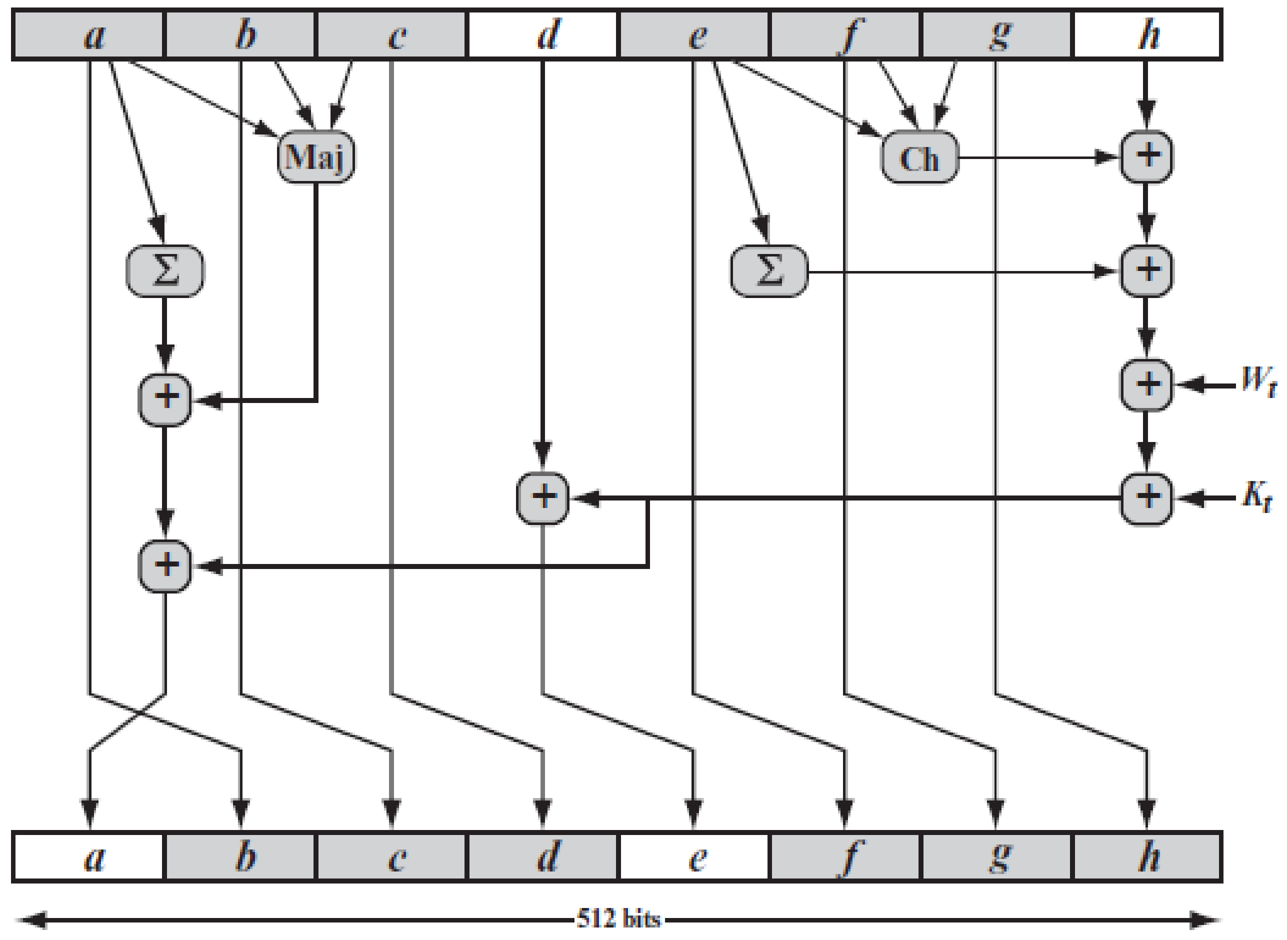


Figure 11.11 Elementary SHA-512 Operation (single round)

SHA-512 Round Function Summarized as follows:

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

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

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

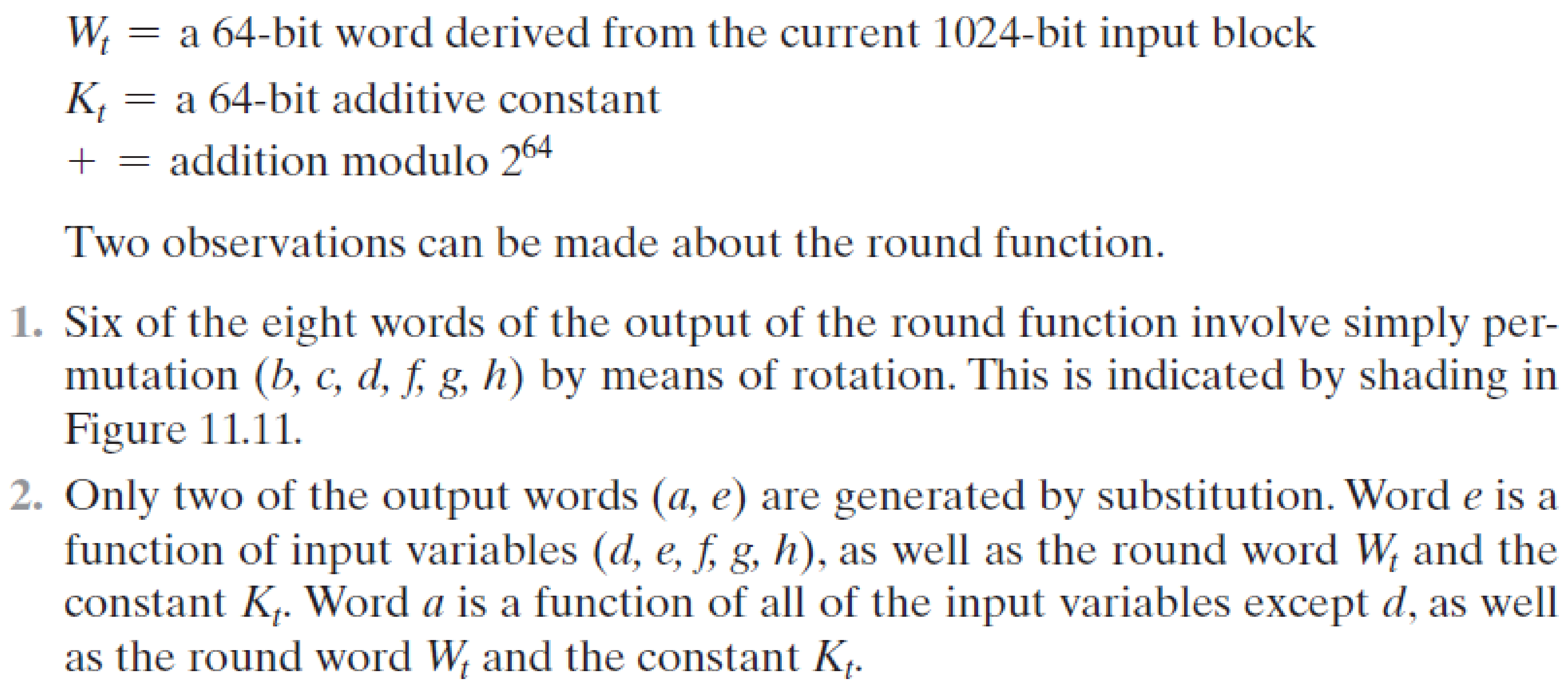
$$b = a$$

$$a = T_1 + T_2$$

where

t	= step number; $0 \leq t \leq 79$
$\mathrm{Ch}(e, f, g)$	= (e AND f) $\oplus$ (NOT e AND g) *the conditional function: If e then f else g*
$\mathrm{Maj}(a, b, c)$	= (a AND b) $\oplus$ (a AND c) $\oplus$ (b 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 Summarized as follows (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}

Two observations can be made about the round function.

1. Six of the eight words of the output of the round function involve simply permutation (b, c, d, f, g, h) by means of rotation. This is indicated by shading in Figure 11.11.
2. Only two of the output words (a, e) are generated by substitution. Word e is a function of input variables (d, e, f, g, h), as well as the round word W_t and the constant K_t. Word a is a function of all of the input variables except d, as well as the round word W_t and the constant K_t.

How the 64-bit word values *Wt* are derived from the 1024-bit message. ?

- The first 16 values of *Wt* are taken directly from the 16 words of the current block. The remaining values aredefined as

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

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}

How the 64-bit word values *Wt* are derived from the 1024-bit message. ?

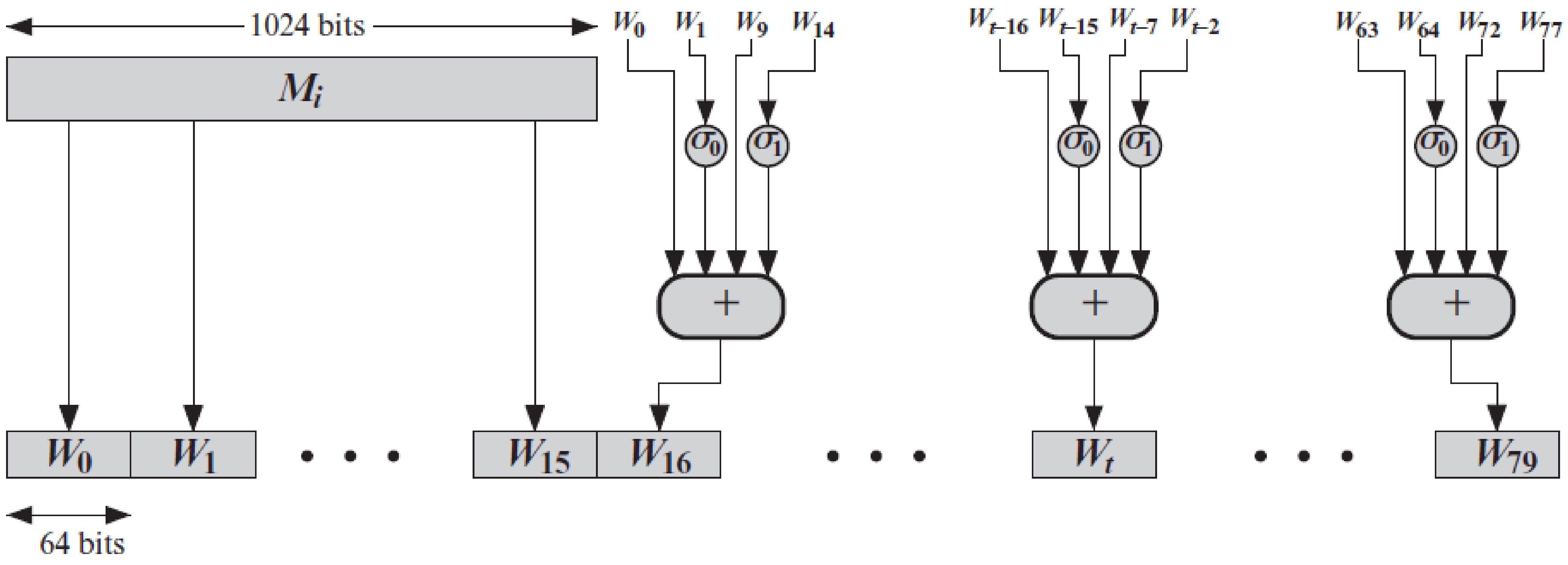

1024 bits
M_i
W_0 W_1 W_9 W_{14}
W_{t-16} W_{t-15} W_{t-7} W_{t-2}
W_{63} W_{64} W_{72} W_{77}
σ_0
σ_1
+
W_0
W_1
W_{15}
W_{16}
W_t
W_{79}
64 bits

Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block