Department of CSE, NIIT University, Neemrana Rajasthan

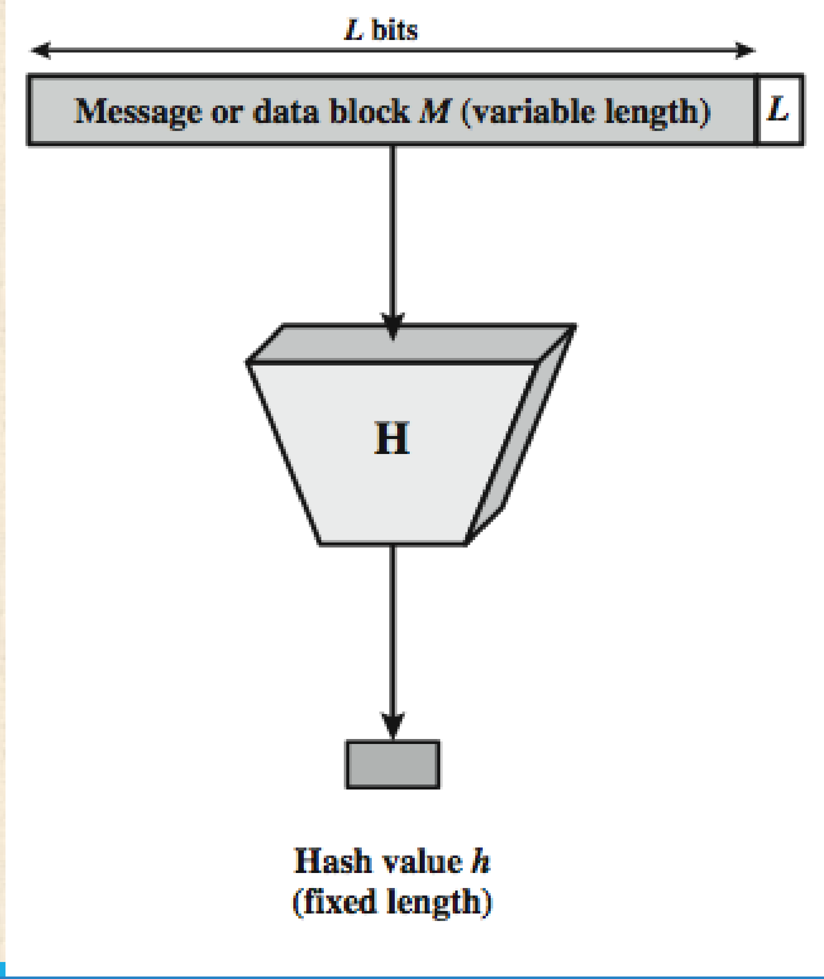
Subject- Cryptography

Faculty-
Vivek Kumar Anand

Hash Functions

- condenses arbitrary message to fixed size
  $h = H(M)$
- 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)

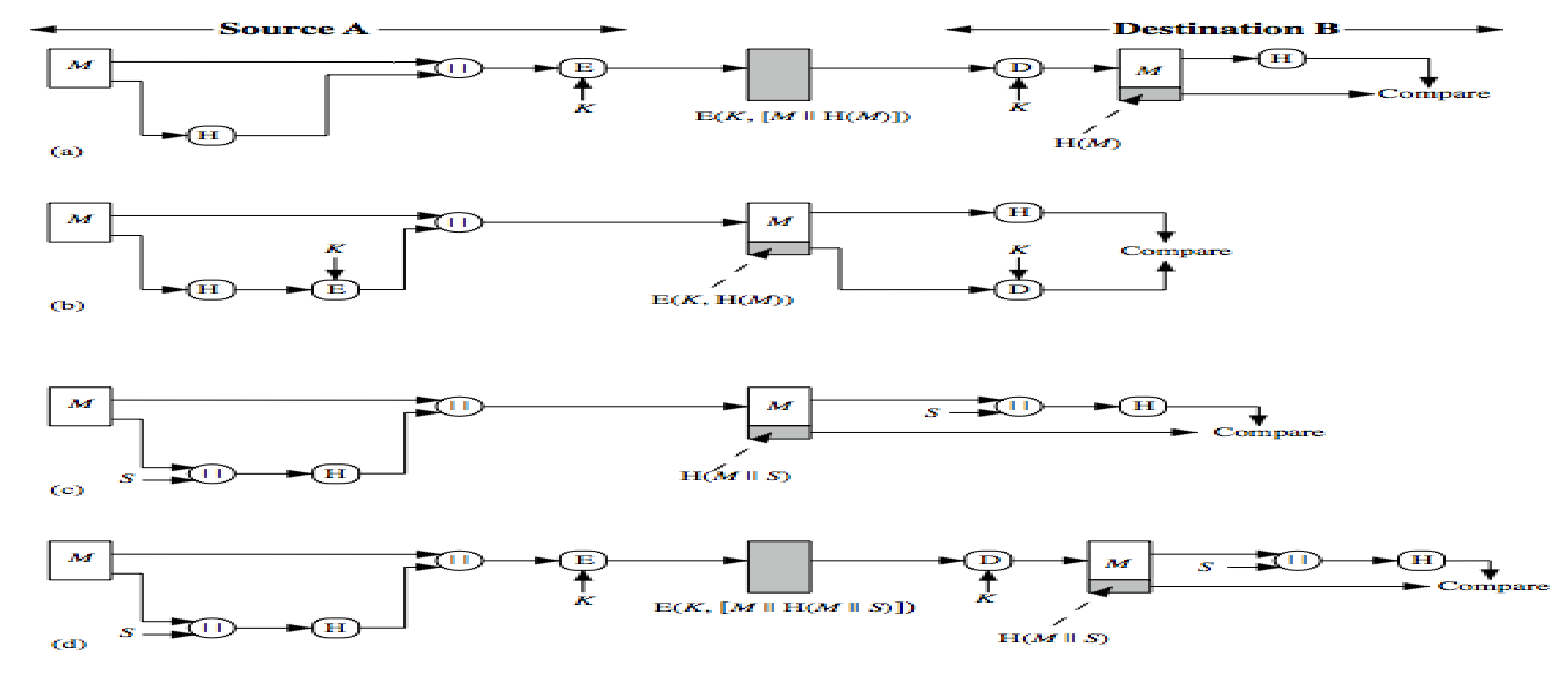
Cryptographic Hash Function



Message Authentication

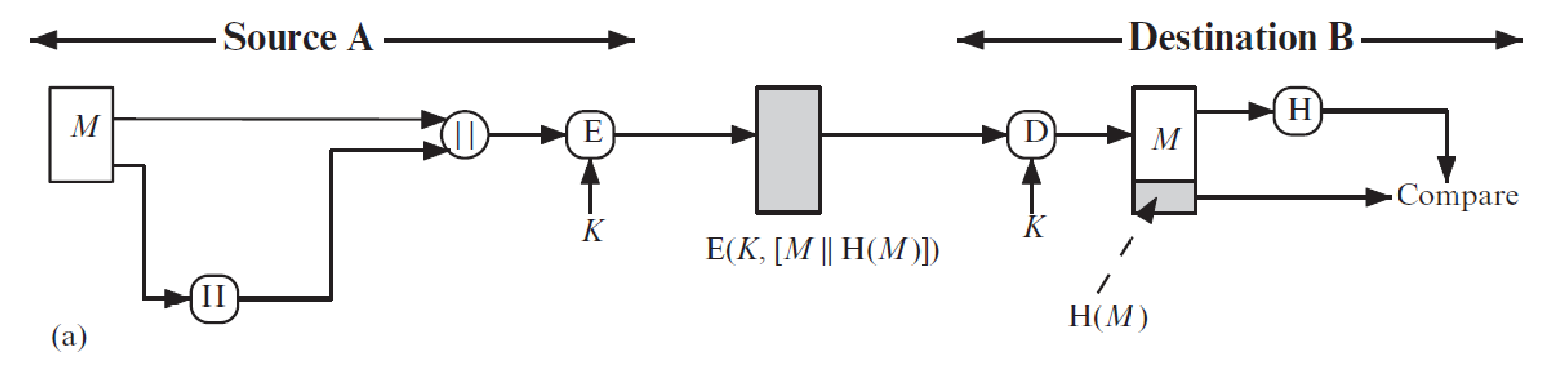
- Message authentication is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).
- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest.**

Hash Functions & Message Authentication



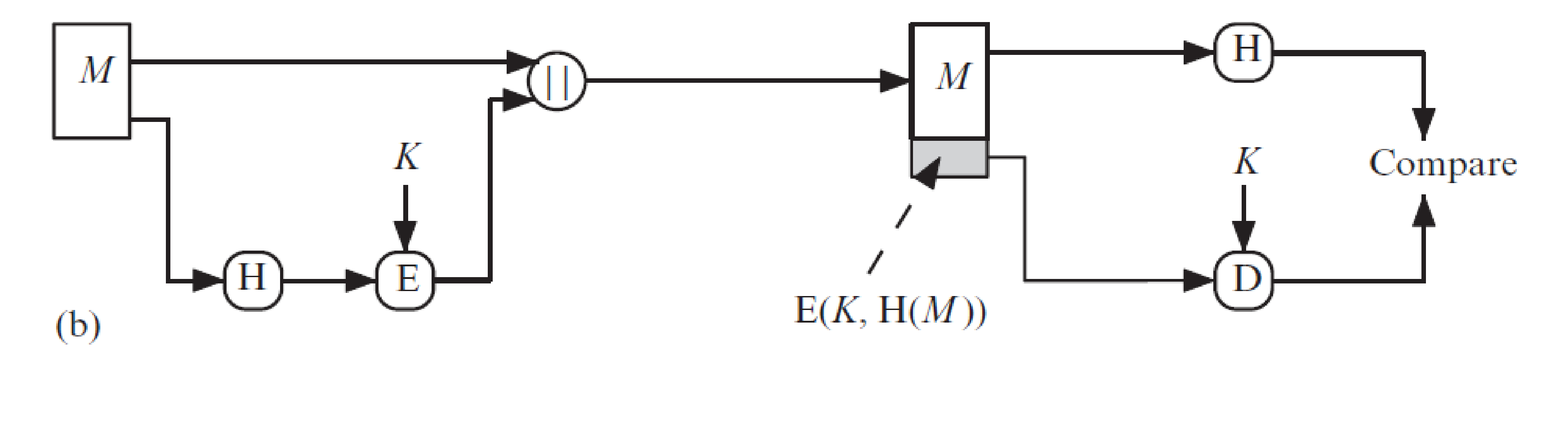
Hash Functions & Message Authentication

- 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.



Hash Functions & Message Authentication

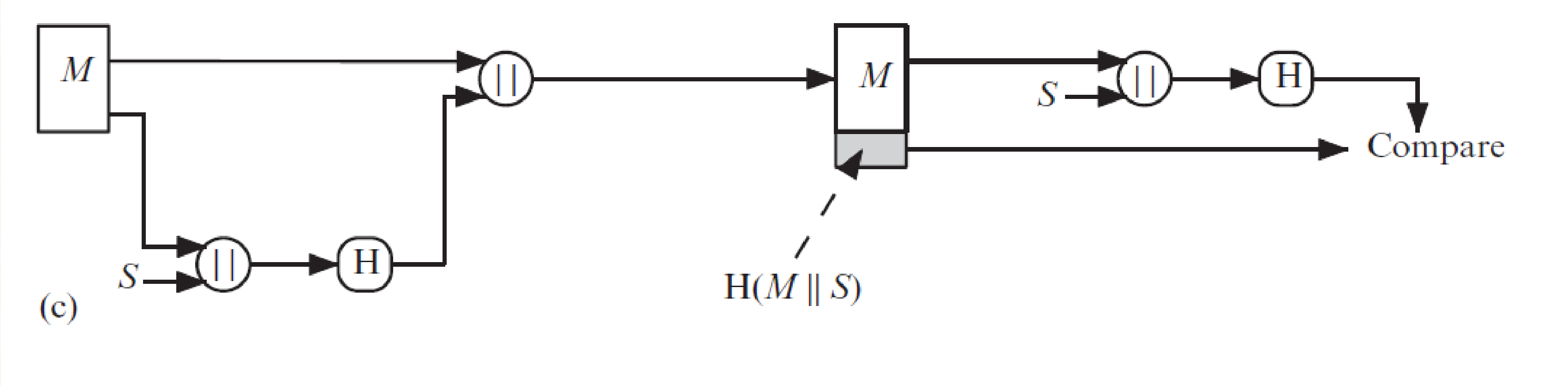
Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



Hash Functions & Message Authentication

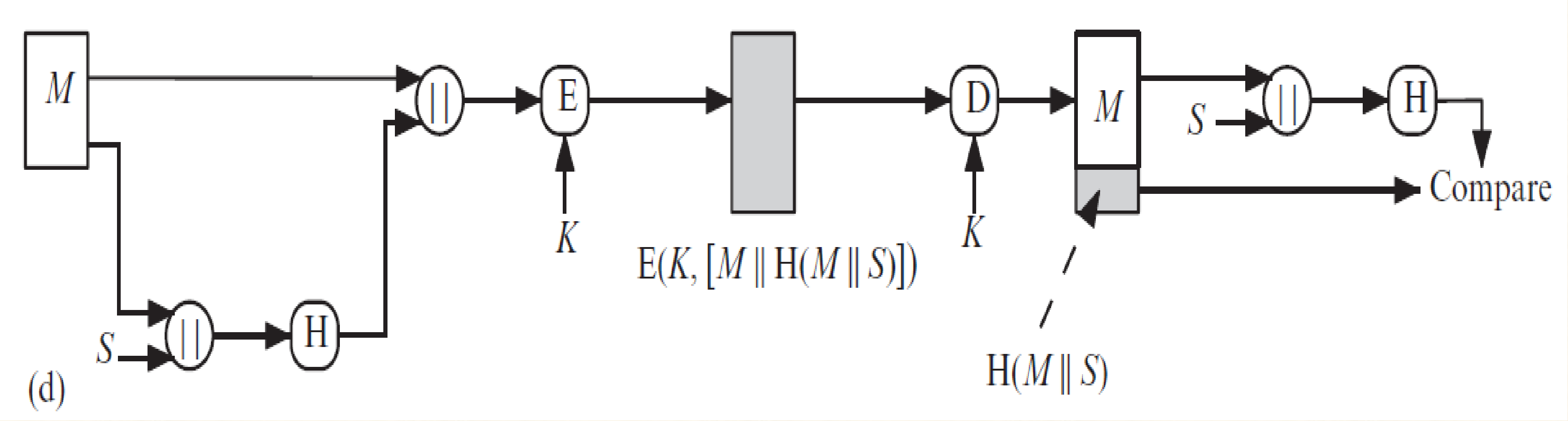
It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S.

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 recomputed 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.



Hash Functions & Message Authentication

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



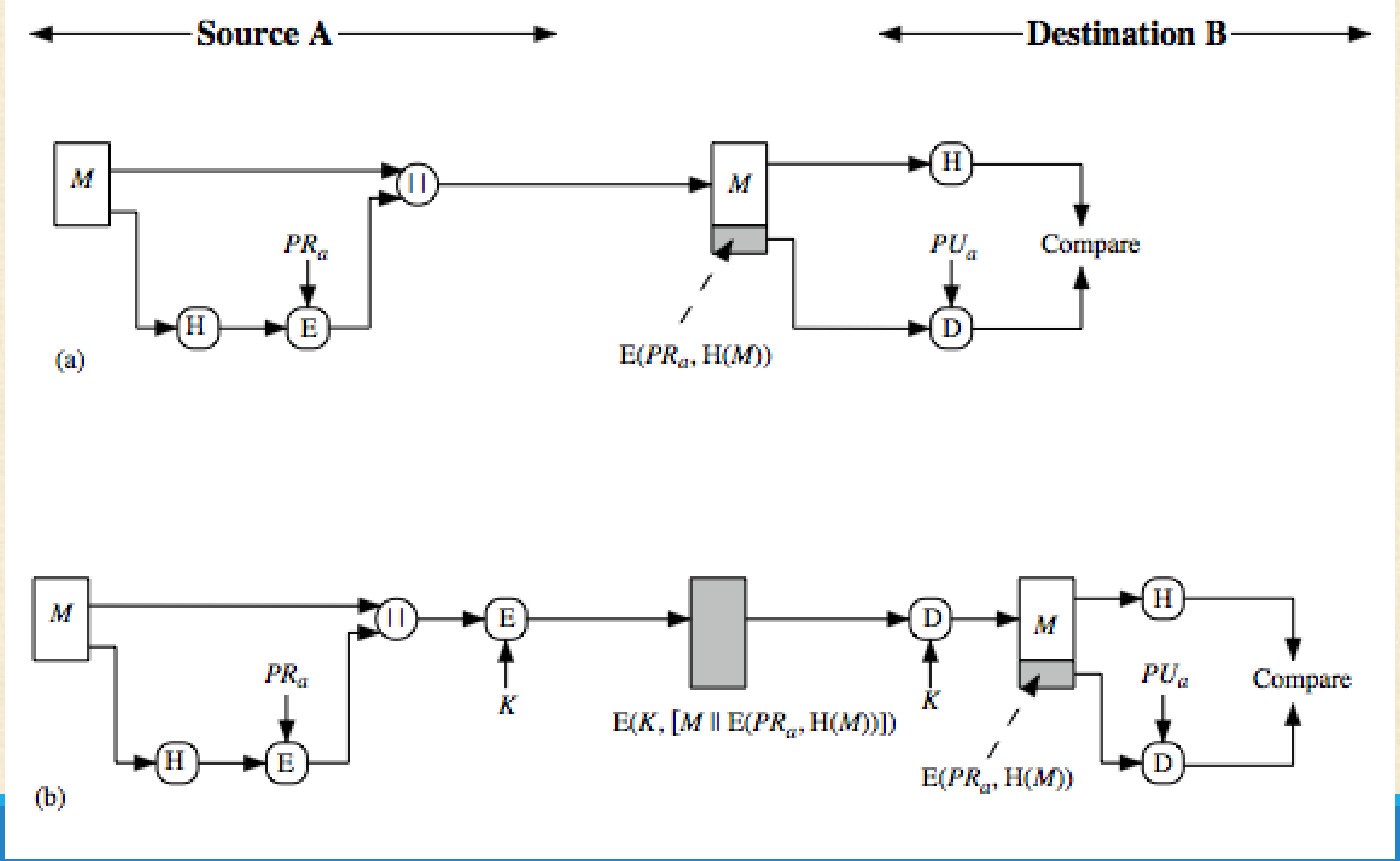
Digital Signature

- Another important application, which is similar to the message authentication application, is the **digital signature**.

- 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.

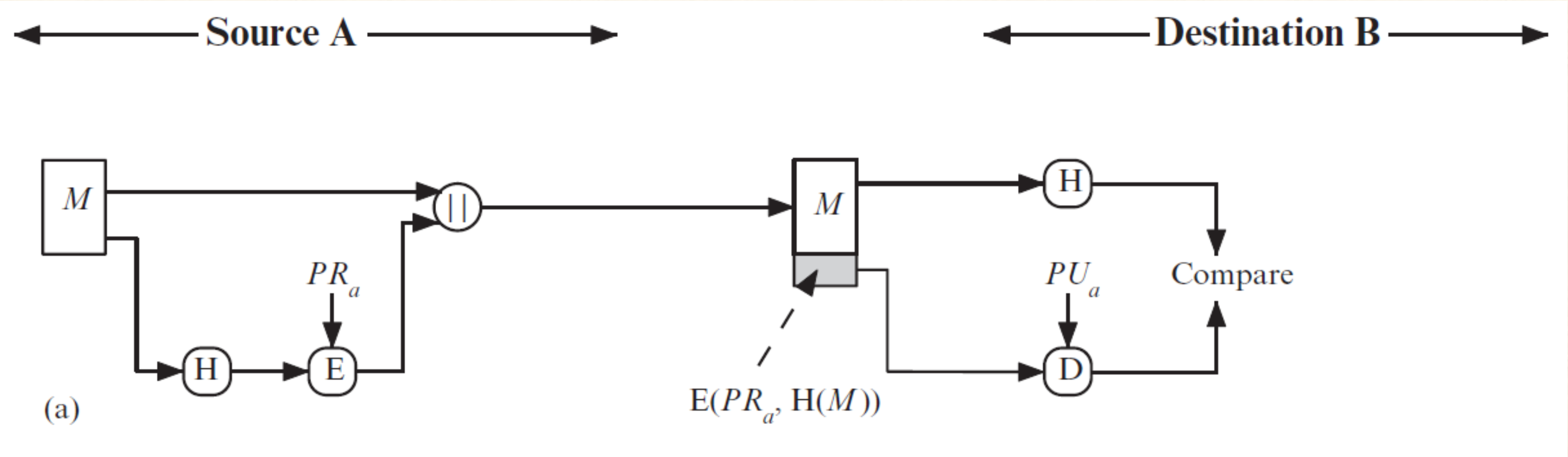
Hash Functions & Digital Signatures



Hash Functions & Digital Signatures

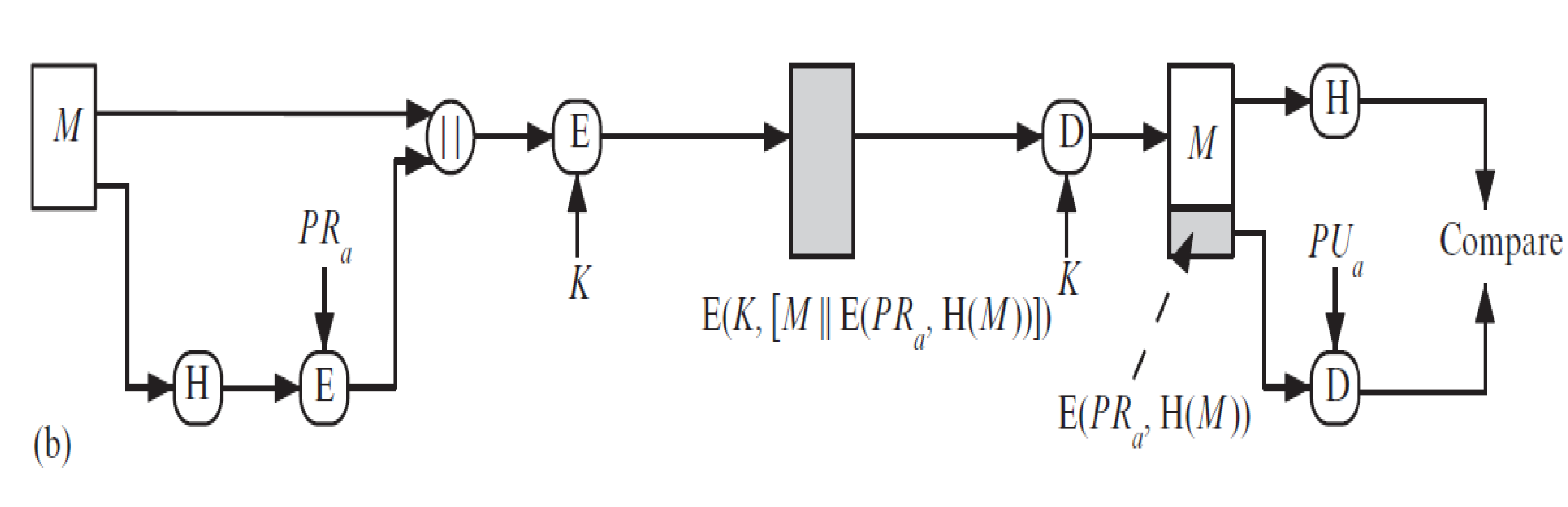
The hash code is encrypted, using public-key encryption with the sender's private key. As with Figure, this provides authentication.

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.



Hash Functions & Digital Signatures

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.



Two Simple Insecure Hash Functions

- consider two simple insecure hash functions
- bit-by-bit exclusive-OR (XOR) of every block
  - $C_i = b_{i1}$ *xor* b_{i2} *xor* . . . *xor* b_{im}
  - a longitudinal redundancy check
  - reasonably effective as data integrity check
- one-bit circular shift on hash value
  - for each successive *n-bit* block
    - rotate current hash value to left by 1bit and XOR block
  - good for data integrity but useless for security

Other Hash Function Uses

To create a one-way password file

- store hash of password not actual password

For intrusion detection and virus detection

- keep & check hash of files on system

Pseudorandom function (PRF) or pseudorandom number generator (PRNG)

Define : Preimage

- For a hash value ***h*=H(*x*)**, we say that ***x*** is the **preimage** of ***h***. That is, ***x*** is a data block whose hash function, 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 **H(*x*)=H(*y*)**. Because we are using hash functions for data integrity, collisions are clearly undesirable.

Hash Function Requirements

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 ! 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

Hash Function Requirements

Second pre-image resistance

Given an input m_1 it should be difficult to find another input m_2 such that $m_1 \neq m_2$ and $\mathrm{hash}(m_1) = \mathrm{hash}(m_2)$. Functions that lack this property are vulnerable to second-preimage attacks.

Collision resistance

It should be difficult to find two different messages m_1 and m_2 such that $\mathrm{hash}(m_1) = \mathrm{hash}(m_2)$. Such a pair is called a cryptographic hash collision.

The difference is in the choice of m_1.

- In the first case (second preimage resistance), the attacker is **handed a fixed** m_1 to which he has to find a different m_2 with equal hash. In particular, he **can't choose** m_1.
- In the second case (collision resistance), the attacker can **freely choose both messages** m_1 and m_2, with the only requirement that they are different (and hash to the same value).

(From this, it is also obvious that collision resistance implies second preimage resistance: An attacker can just choose an arbitrary m_1 and compute a second preimage m_2 to obtain a collision.)

Attacks on Hash Functions

- Two categories of attacks on hash functions: brute-force attacks and cryptanalysis

- a preimage or second preimage attack
  - find *y* s.t. *H(y)* equals a given hash value

- collision resistance
  - find two messages x & *y* with same hash so H(x) = H(y)

- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks

m bit hash value

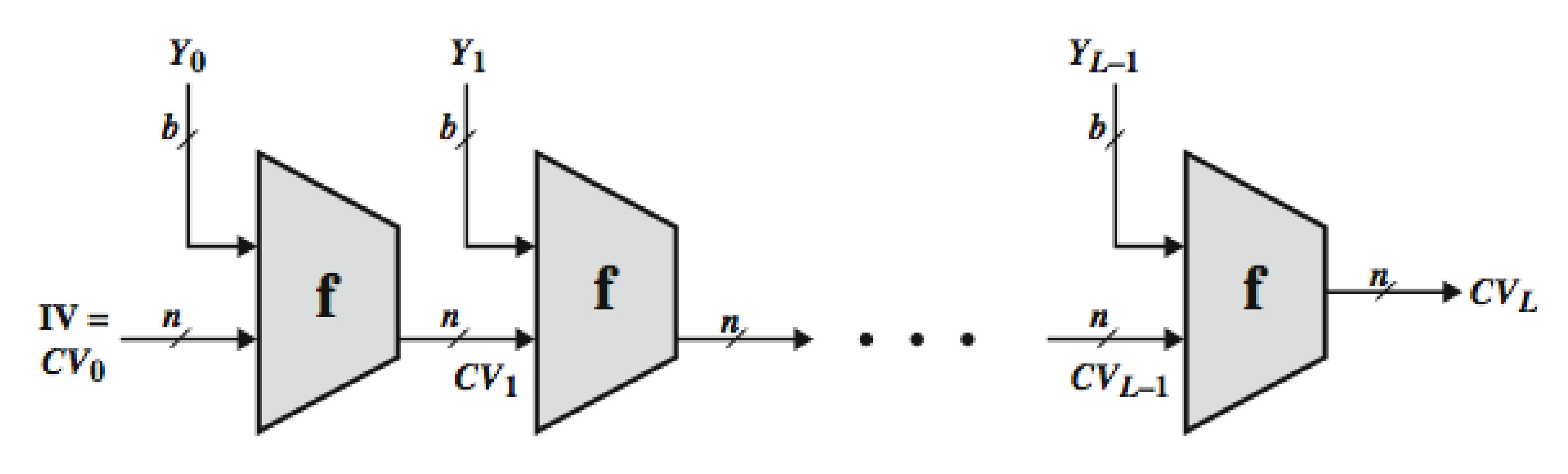
Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - given user prepared to sign a valid message x
  - opponent generates $2^{m/2}$ variations x' of x, all with essentially the same meaning, and saves them
  - opponent generates $2^{m/2}$ variations y' of a desired fraudulent message y
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use larger MAC/hash

link

Hash Function Cryptanalysis

- Cryptanalytic attacks exploit some property of algorithm so faster than exhaustive search
- Hash functions use iterative structure
  - process message in blocks (include length)
- Attacks focus on collisions in function f



Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - using $H_0=0$ and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key

- resulting hash is too small (64-bit)
  - both due to direct birthday attack
  - and to “meet-in-the-middle” attack

Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 in 1995, also Internet RFC3174
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

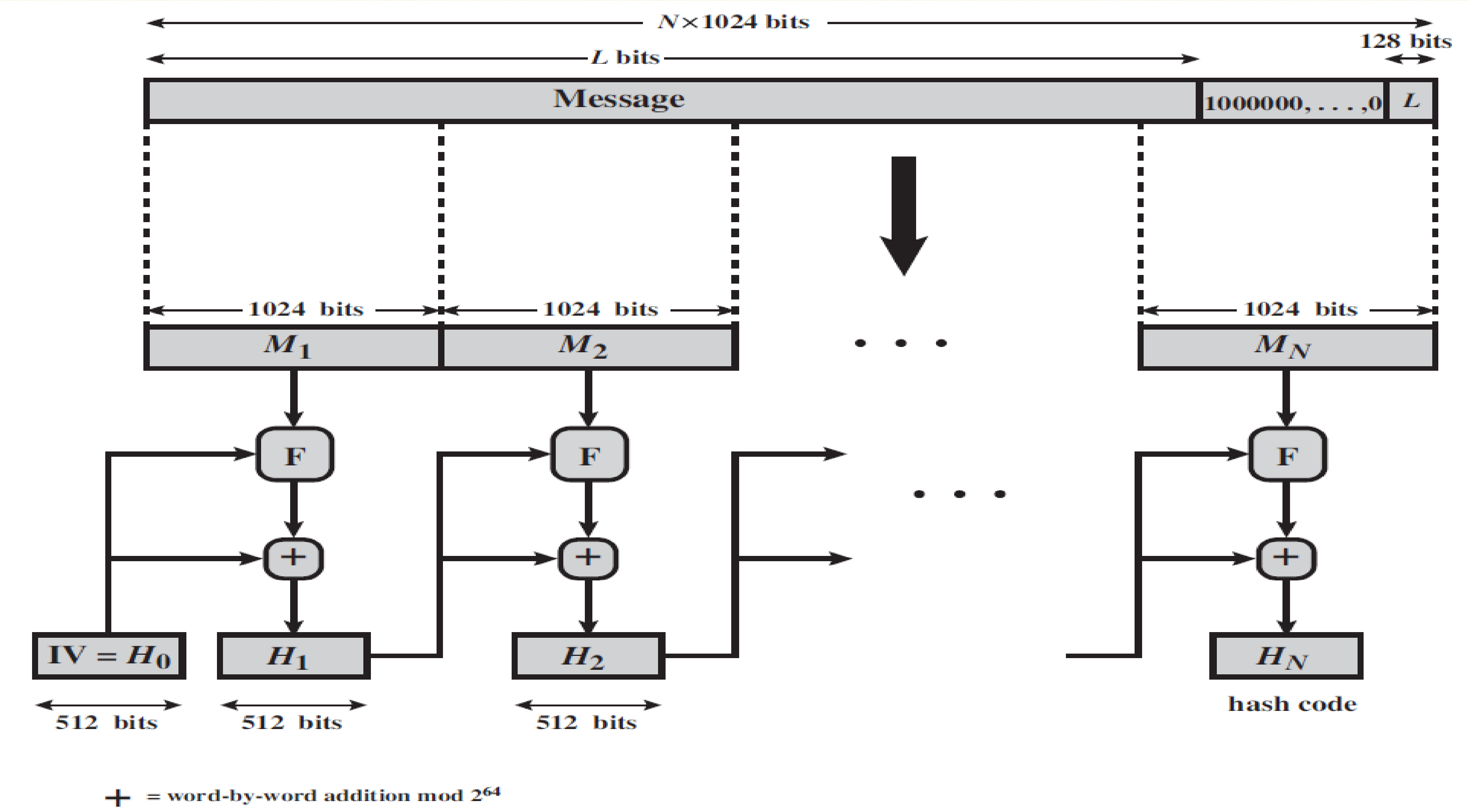
Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA Versions

	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 Overview



SHA-512 Compression Function

Heart of the algorithm-

- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value Wt derived from the current message block.

SHA-512

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.

SHA-512

- **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.

SHA-512

Step 3 Initialize hash buffer: A 512-bit buffer is used to hold intermediate and final results of the hash function.

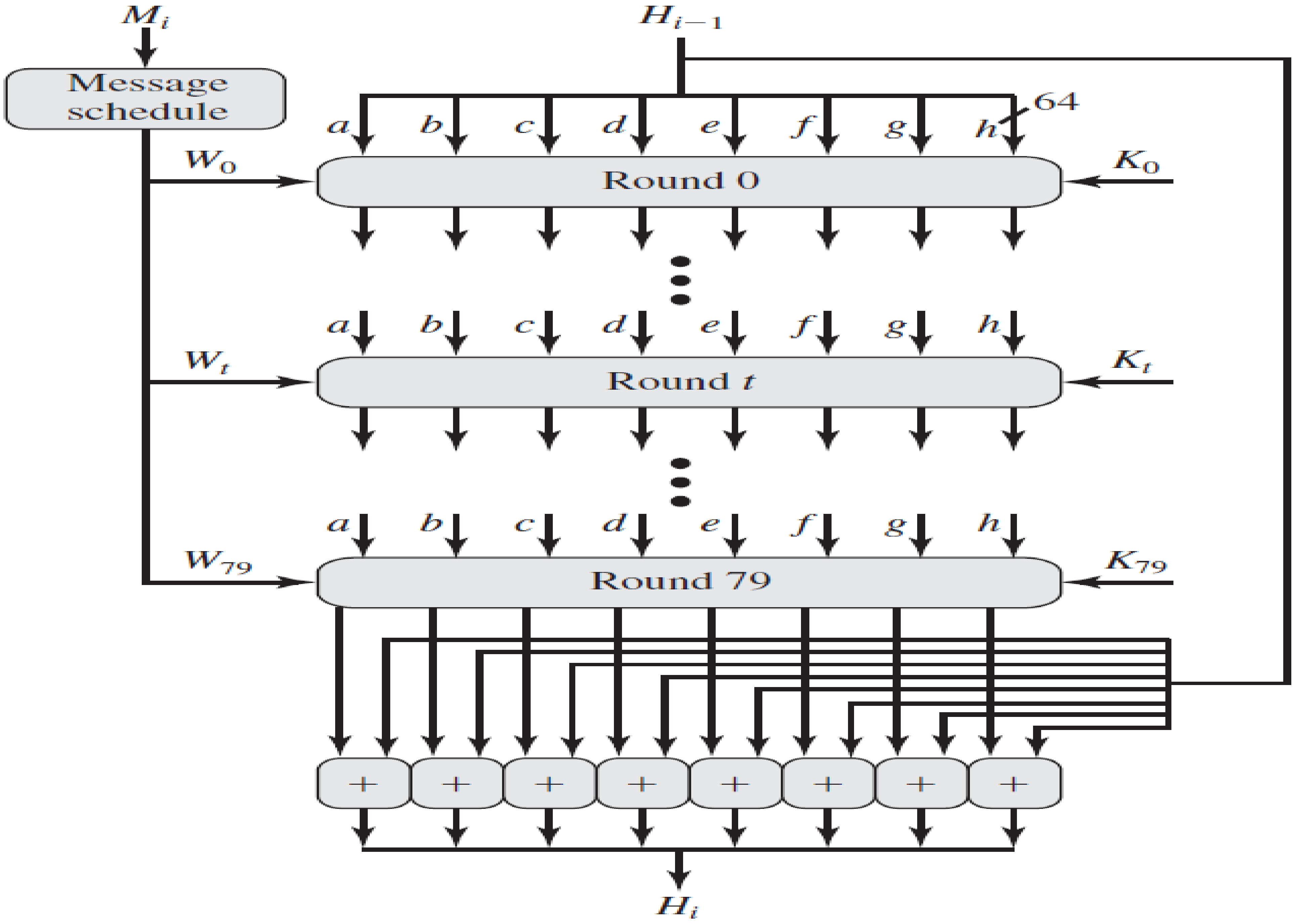
The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).These registers are initialized to the following 64-bit integers (hexadecimal values):

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

SHA-512

- **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 labeled F in Figure.

- The logic is illustrated in Following Figure.



M_i
H_{i-1}
Message schedule
64
a b c d e f g h
W_0
Round 0
K_0
W_t
Round t
K_t
W_{79}
Round 79
K_{79}
+ + + + + + + +
H_i

SHA-512

Step 5 Output. After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

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

$$H_0 = \mathrm{IV}$$

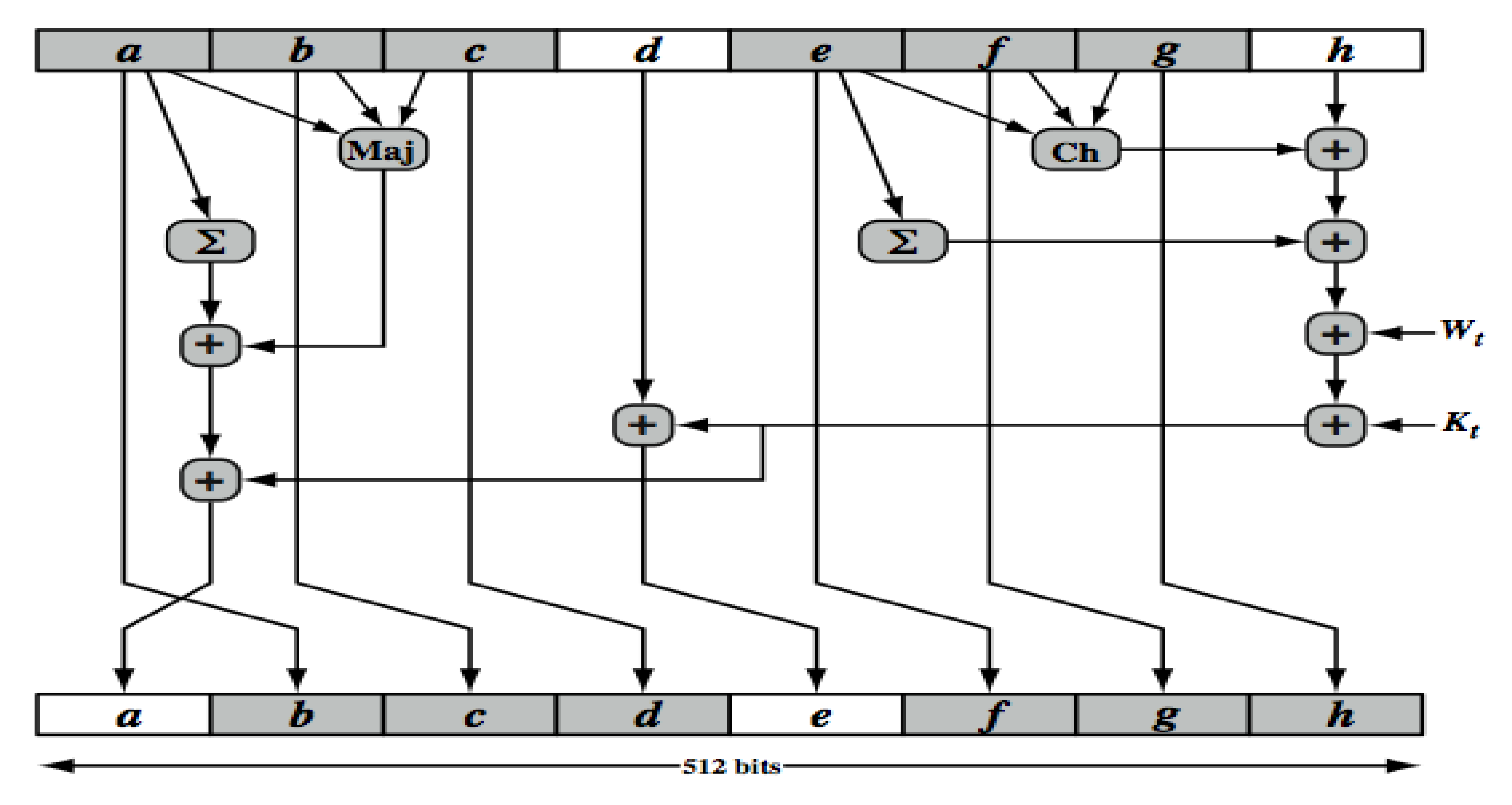
$$H_i = \mathrm{SUM}_{64}(H_{i-1}, \mathrm{abcdefgh}_i)$$

$$MD = H_N$$

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 ith 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



SHA-512 Round Function

The logic in each of the 80 steps of the processing of one 512-bit block. Each round is defined by the following set of Equations.

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

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

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

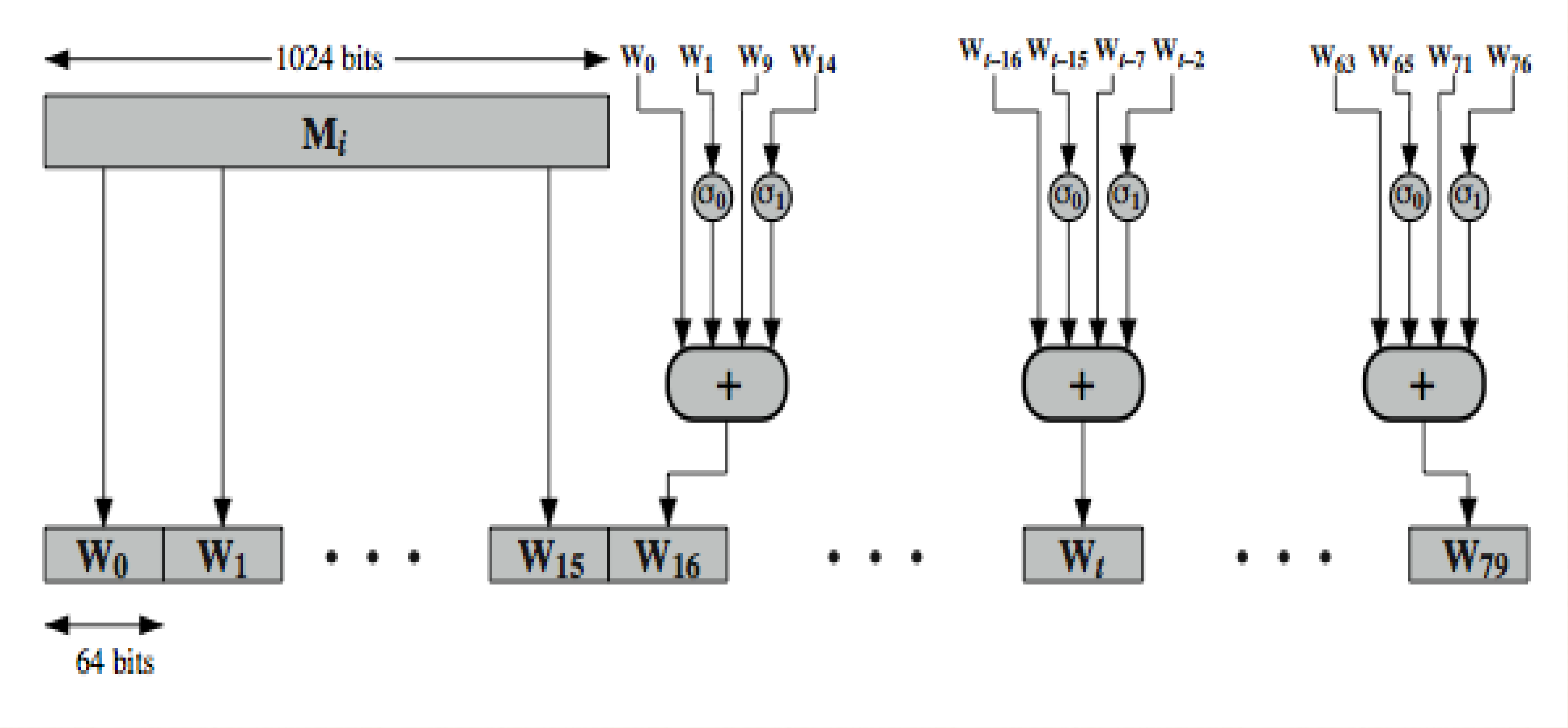
$$a = T_1 + T_2$$

SHA-512 Round Function

where

$$
\begin{aligned}
t &= \text{step number}; 0 \le t \le 79 \\
\mathrm{Ch}(e, f, g) &= (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g) \\
&\quad \textit{the conditional function: If e then f else g} \\
\mathrm{Maj}(a, b, c) &= (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c) \\
&\quad \textit{the function is true only of the majority (two or three) of the} \\
&\quad \textit{arguments are true} \\
\left(\textstyle\sum_0^{512} a\right) &= \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a) \\
\left(\textstyle\sum_1^{512} e\right) &= \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e) \\
\mathrm{ROTR}^n(x) &= \text{circular right shift (rotation) of the 64-bit argument } x \text{ by } n \text{ bits} \\
W_t &= \text{a 64-bit word derived from the current 512-bit input block} \\
K_t &= \text{a 64-bit additive constant} \\
+ &= \text{addition modulo } 2^{64}
\end{aligned}
$$

SHA-512 Round Function



SHA-3

- SHA-1 not yet "broken”
  - but similar to broken MD5 & SHA-0
  - so considered insecure
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function

SHA-3 Requirements

- replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the online nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory
  - characteristics: such as flexibility & simplicity

Summary

have considered:

- hash functions
  - uses, requirements, security
- hash functions based on block ciphers
- SHA-1, SHA-2, SHA-3