CST433 SECURITY IN COMPUTING

MODULE IV

Module 4

(Message Integrity and Authentication)

Hash functions – Security requirements, Secure Hash Algorithm (SHA-512). Message Authentication Code (MAC) – Requirements, Uses, Hash-based MAC (HMAC), Cipher-based MAC (CMAC). Digital signatures – Attacks, Forgeries, Requirements, Direct vs Arbitrated digital signatures, RSA digital signature, ElGamal digital signature, Digital Signature Standard (DSS).

Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

Authentication Requirements

In communications across a network, the following attacks can be identified**.**

Disclosure: release of msg contents to any person or process not possessing the appropriate key

traffic analysis:discovery of pattern of traffic bw parties from which msg length could be determined.

Masquerade:insertion of msgs into the network from a fraudulent source .

content modification:include insertion, deletion transposition and modification

.

4

Authentication Requirements…

sequence modification: include insertion, deletion and reordering a seq of msgs

timing modification: delay or replay of msgs of some previous valid session.

source repudiation

destination repudiation

5

Message Authentication

message authentication is concerned with:

- protecting the integrity of a message
- validating identity of originator
- non-repudiation of origin (dispute resolution)

6

MA can be viewed as having 2 levels

- **Lower level:** there must be some sort of **function** that produces an authenticator which can be a value to be used to authenticate a msg
- The lower level function is used in **higher level** authentication protocol that enables a receiver to verify the authenticity of the msg.

Message Authentication…

three types of functions are used in construction of an authenticator

message encryption:cipher text of the msg serves as its authenticator

message authentication code (MAC):a public function of the msg and a secret key that produces a fixed length value that serves as authenticator

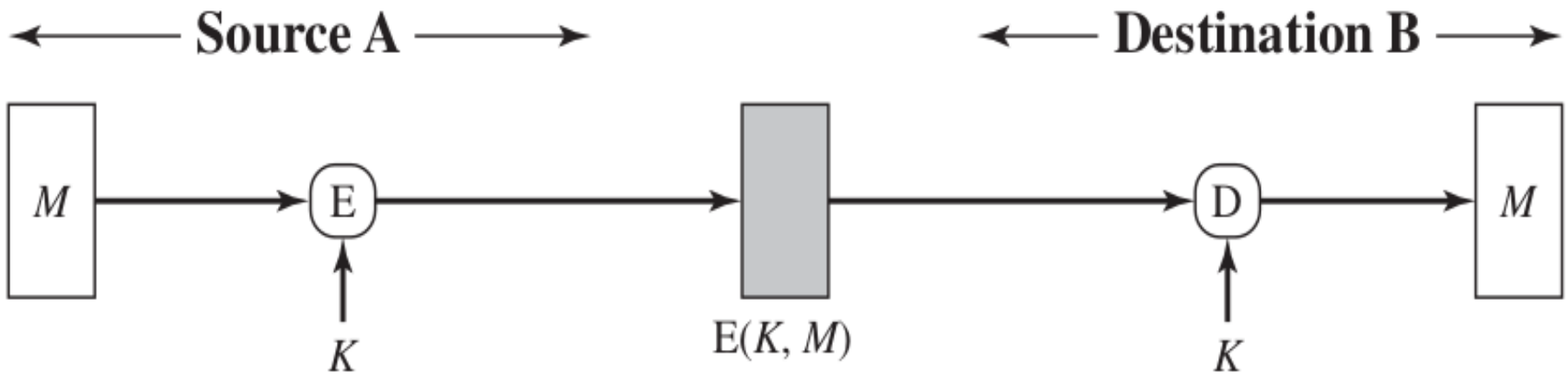
hash function:a public function that maps a msg of any length into a fixed length hash value which serves as authenticator

8

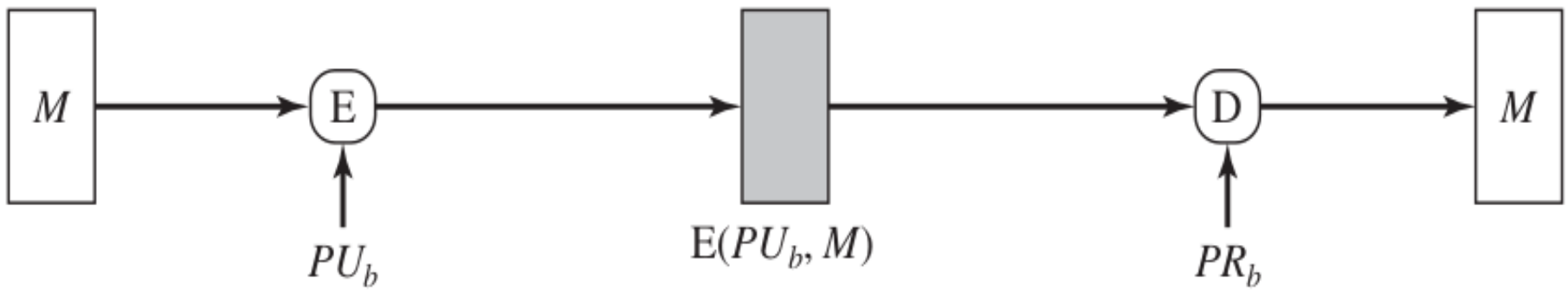
Message Encryption

- **message encryption** by itself also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver know key used
  - know content cannot have been altered
  - a checksum can be used to detect changes if any
- **generally unreliable**

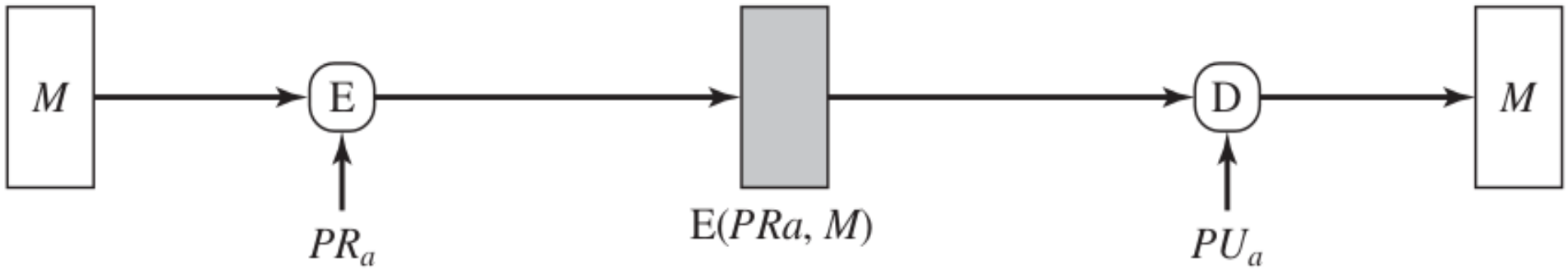
9



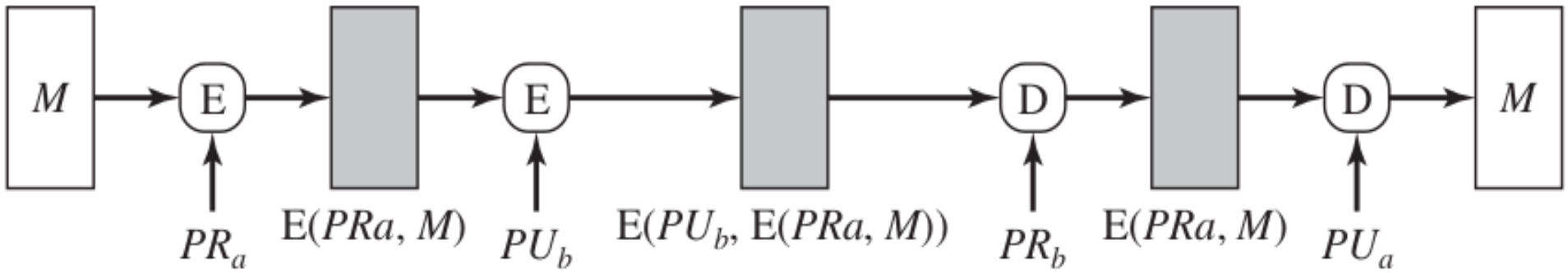
(a) Symmetric encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Figure 12.1 Basic Uses of Message Encryption

Message Encryption…

In public key encryption if private -key is used for encryption:

- encryption provides no confidence of sender
- since anyone knows public-key

if

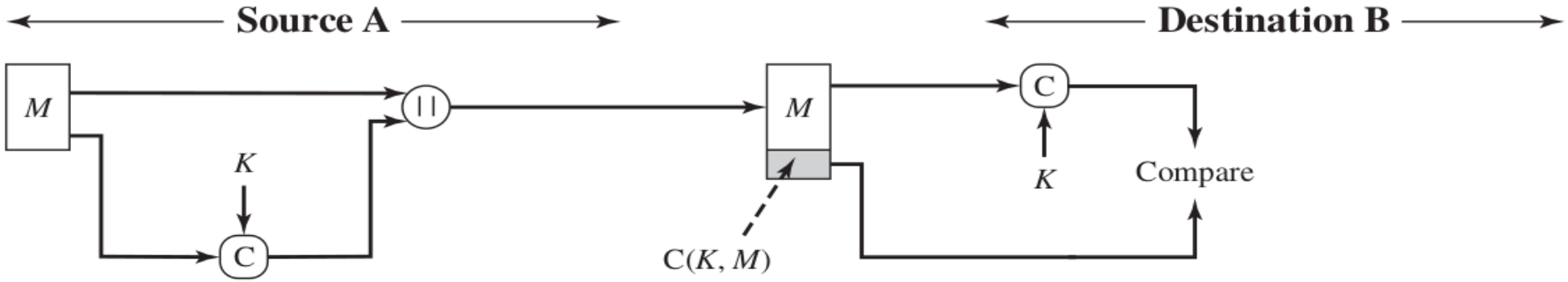
- sender **signs** message using her private-key
- then encrypts with recipients public key
- have both secrecy and authentication

12

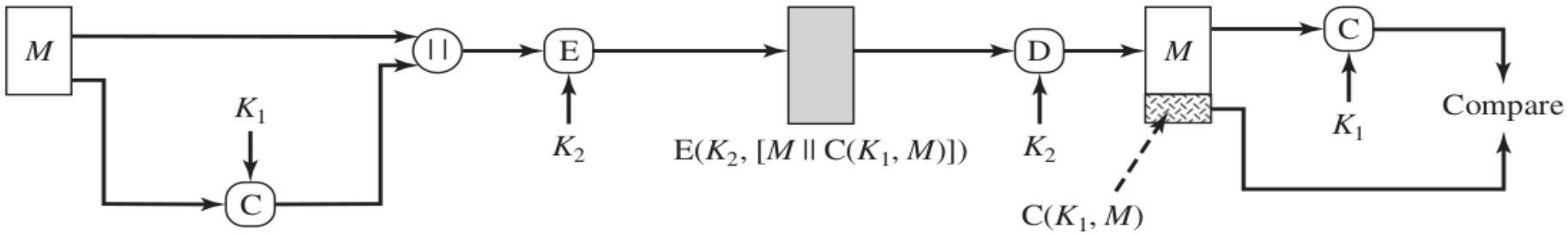
Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender who shares the key.

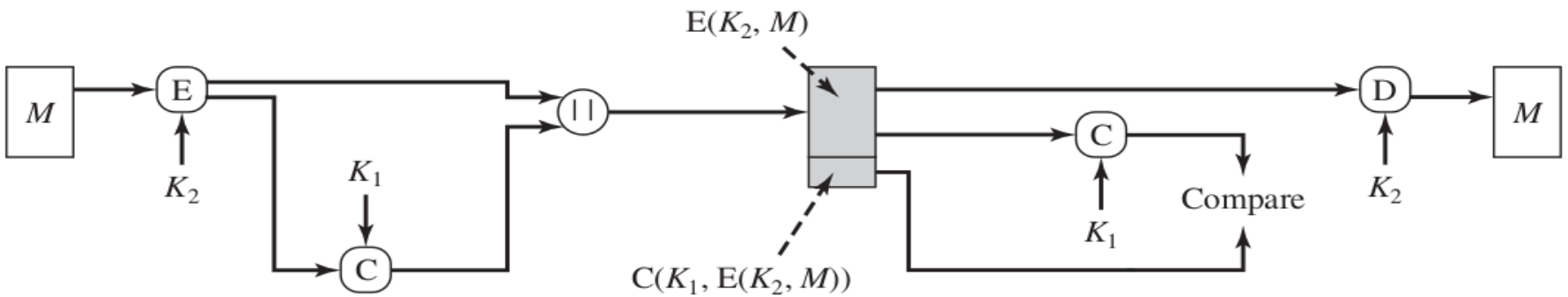
13



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

Figure 12.4 Basic Uses of Message Authentication code (MAC)

14

Message Authentication Codes

- the MAC provides authenticity
- can also use encryption for secrecy
- generally use separate keys for each
- can compute MAC either before or after encryption
- generally regarded as better done before

15

MAC…

- A MAC function is similar to encryption.
- One difference is that the MAC algorithm need not be reversible, as it must be for decryption.
- In general, the MAC function is a many-to-one function
- because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

Why MAC ?

symmetric encryption will provide authentication, why not simply use this instead of a separate message authentication code?

situations in which a message authentication code is used.

1) In the case of broadcasting the msg is broadcast in plain text with MAC.

- One destination is responsible monitoring authenticity. - if a violation occurs the other destinations are alerted.

17

2) An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming msgs. msgs chosen at random for checking.

3) computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources.

-if a message authentication code were attached to the program, it could be checked whenever assurance was required of the integrity of the program.

4) For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages.

5) Separation of authentication and confidentiality functions affords architectural flexibility. - For example, it may be desired to perform authentication at the application level but to provide confidentiality at a lower level, such as the transport layer.

6)A user may wish to prolong the period of protection beyond the time of reception yet allow processing of message contents.

-With message encryption, the protection is lost when the message is decrypted, so the message is protected against fraudulent modifications only in transit but not within the target system.

MAC Properties

- a MAC is a cryptographic checksum
- T = MAC(K,M)

MAC(K , M) is the fixed-length authenticator, sometimes called a **tag**.

- is a many-to-one function
  - many messages have samee MAC
  - -brute-force methods for key is not effective

20

Requirements for MAC

MAC function should satisfy the following requirements

1. If an opponent observes M and $\text{MAC}(K, M)$, it should be computationally infeasible for the opponent to construct a message M' such that

$$\text{MAC}(K, M') = \text{MAC}(K, M)$$

2. $\text{MAC}(K, M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $\text{MAC}(K, M) = \text{MAC}(K, M')$ is 2^{-n}, where n is the number of bits in the tag.
3. Let M' be equal to some known transformation on M. That is, $M' = \text{f}(M)$. For example, f may involve inverting one or more specific bits. In that case,

$$\Pr[\text{MAC}(K, M) = \text{MAC}(K, M')] = 2^{-n}$$

The **first requirement** deals with the attack an opponent is able to construct a new message to match a given tag, even though the opponent does not know the key.

The **second requirement** deals with the need to prevent a brute-force attack based on chosen plaintext.

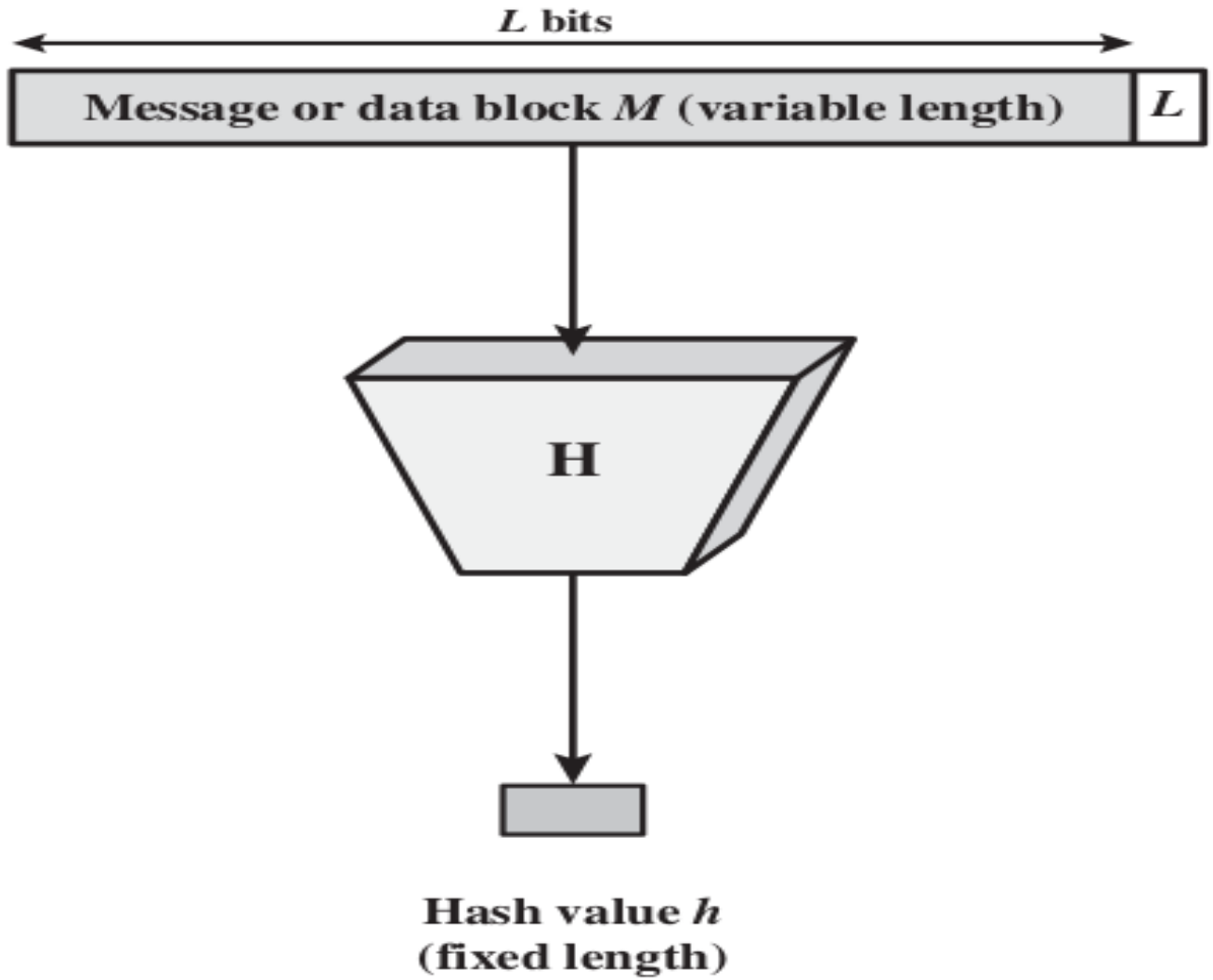
The **final requirement** dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others.

-If this were not the case, then an opponent who had M and MAC(K, M) could attempt variations on M at the known **"weak spots"**

Hash Functions

- A variation of MAC
- HF accepts a var sized M as ip and produces a fixed size op,hash code H(M)
- *h = H(M)*
- usually assume that the hash function is public and not keyed
- hash used to detect changes to message
- can use in various ways with message
- often to create a digital signature

23


L bits
Message or data block M (variable length)
L
H
Hash value h
(fixed length)

HF Applications

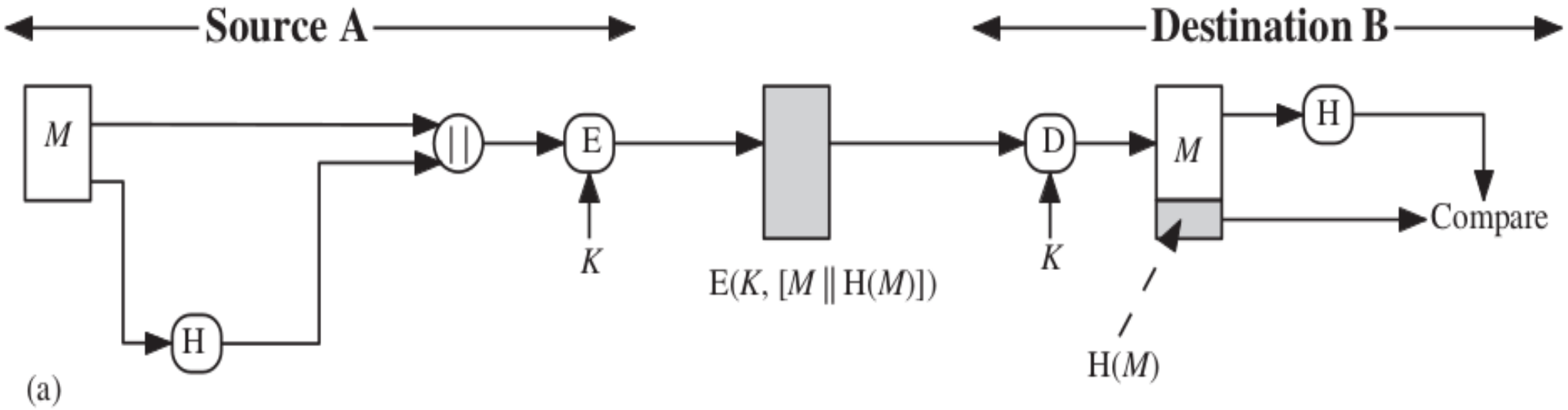
- Message authentication
- Digital Signatures
- Other Applications

  - used to create a **one-way password file**
  - used for intrusion detection and virus detection.
  - Store H(F) for each file on a system and secure the hash values

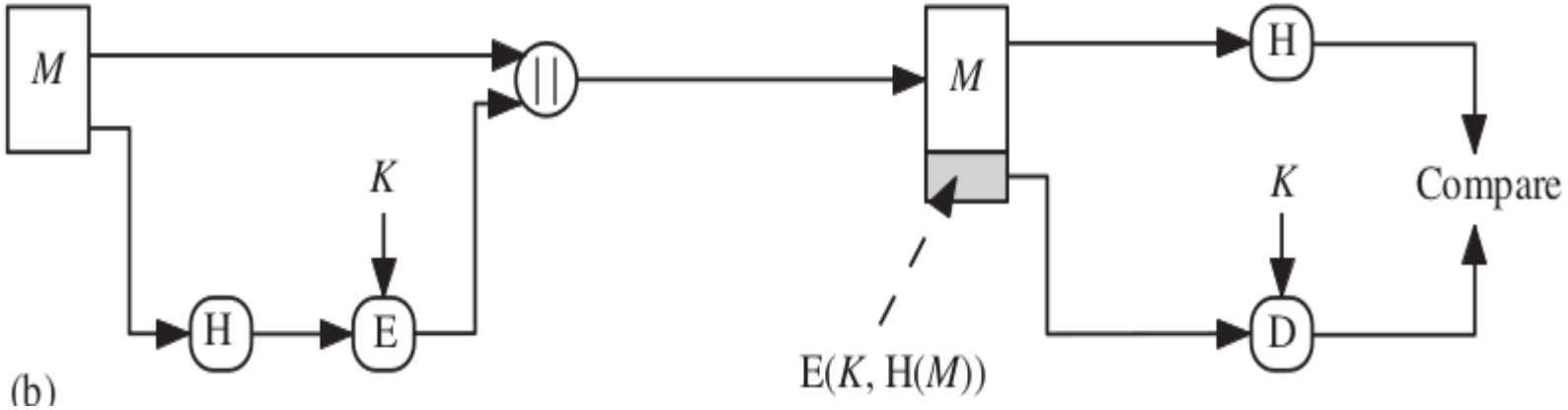
- - used to construct a pseudorandom function (PRF) or a pseudorandom number generator (PRNG).
- A common application for a hash-based PRF is for the generation of symmetric keys.

25

hash function value is often referred to as a **message digest**.

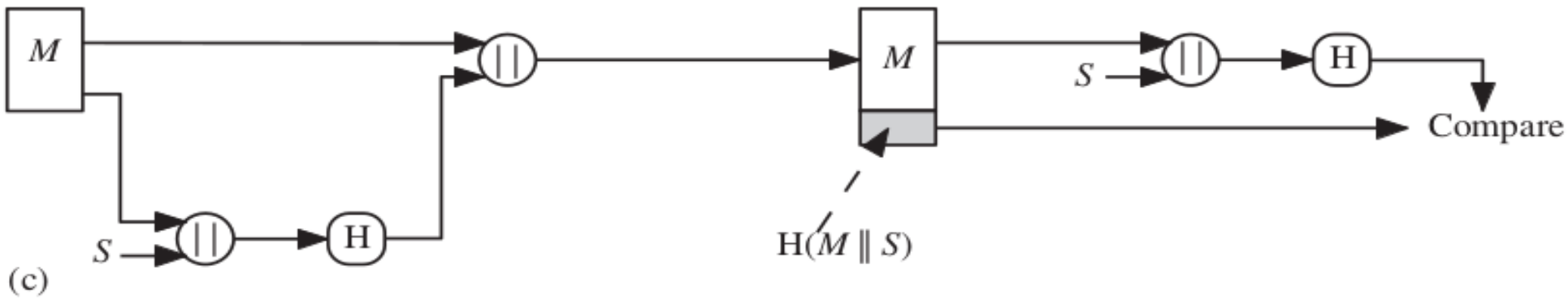


Authentication and confidentiality provided.

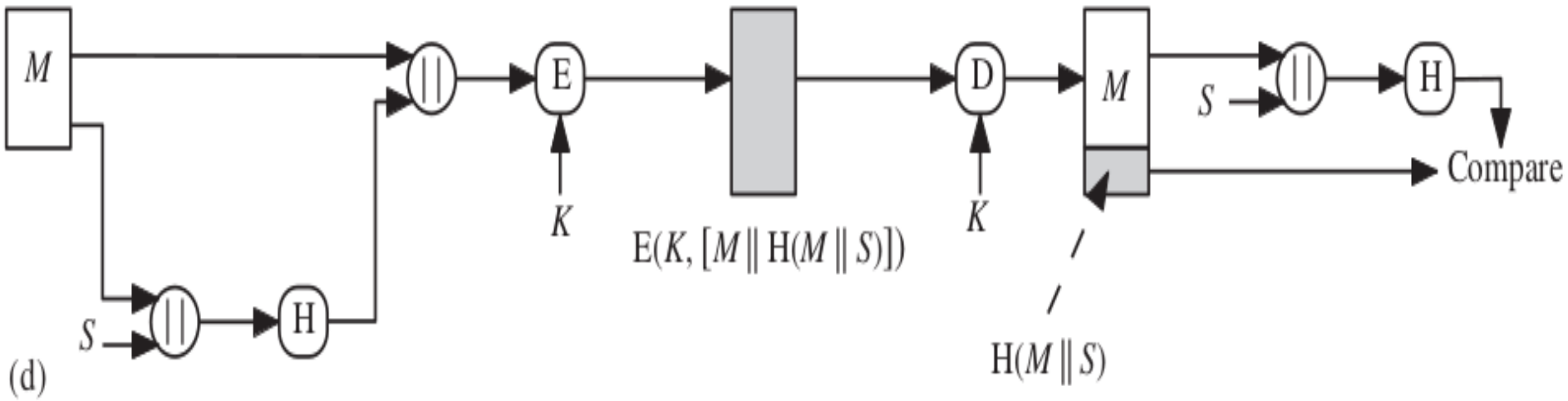


Only hash code is encrypted.

reducing the processing burden of applications those do not require confidentiality



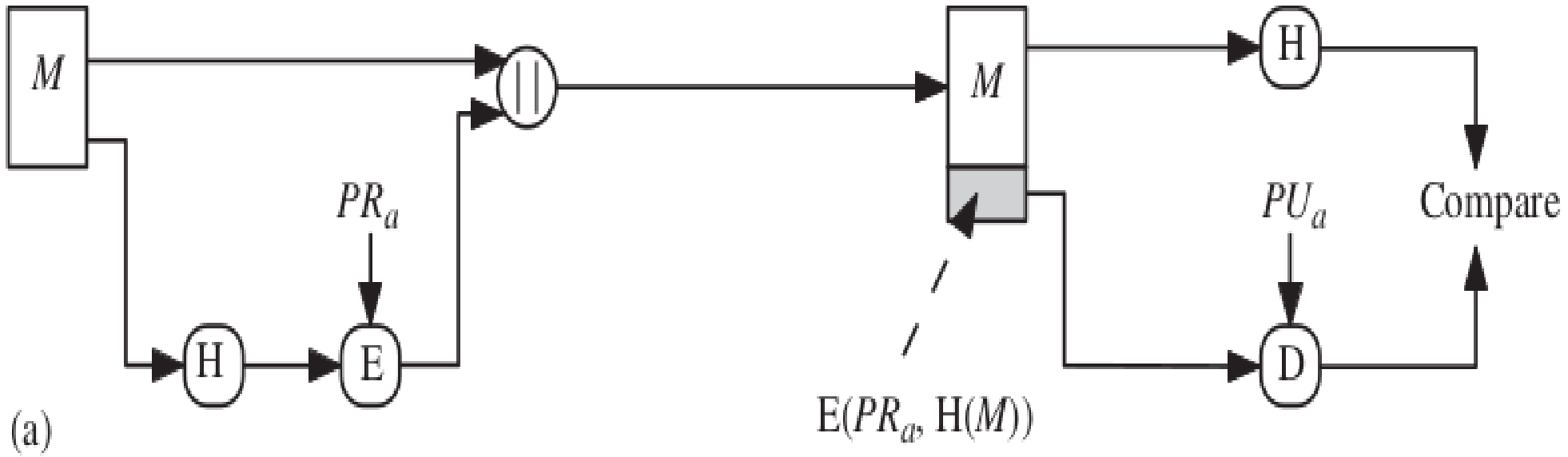
- communicating parties share a common secret value S.
- Because B possesses S , it can recompute the hash value to verify.
- Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.



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

2. Digital Signatures

Only Hash code is encrypted ,using public key encrn and using sender's private key-provides a digital signature.

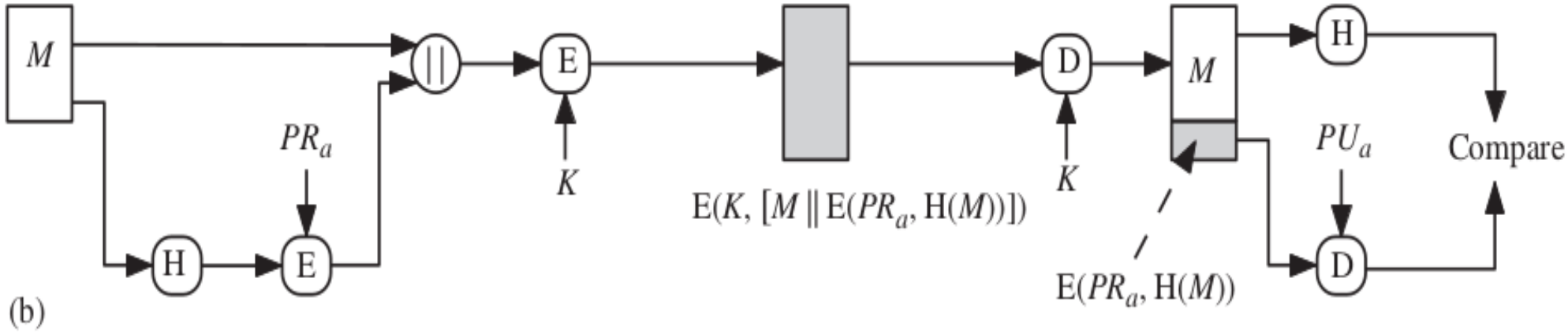


30

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.



31

Requirements for Hash Functions

Terminologies

For a hash value h = H(x) , 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 ≠ y and

$$H(x) = H(y) .$$

Because we are using hash functions for data integrity, collisions are clearly undesirable.

32

Requirements for Hash Functions

Table 11.1 Requirements for a Cryptographic Hash Function H

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.

Requirements for Hash Functions...

- If the sixth property, collision resistant, is also satisfied, then it is referred to as a **strong hash function**

- A hash function that satisfies only the first five propertiescis referred to as a **weak hash function**.

- **Pseudorandomness:** cryptographic hash functions are commonly used for key derivation and pseudorandom number generation

- In message integrity applications, the three resistant properties depend on the output of the hash function appearing to be random.
- So it is required to verify that a given hash function produces pseudorandom output.

Secure Hash Algorithm

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1

Table 11.3 Comparison of SHA Parameters

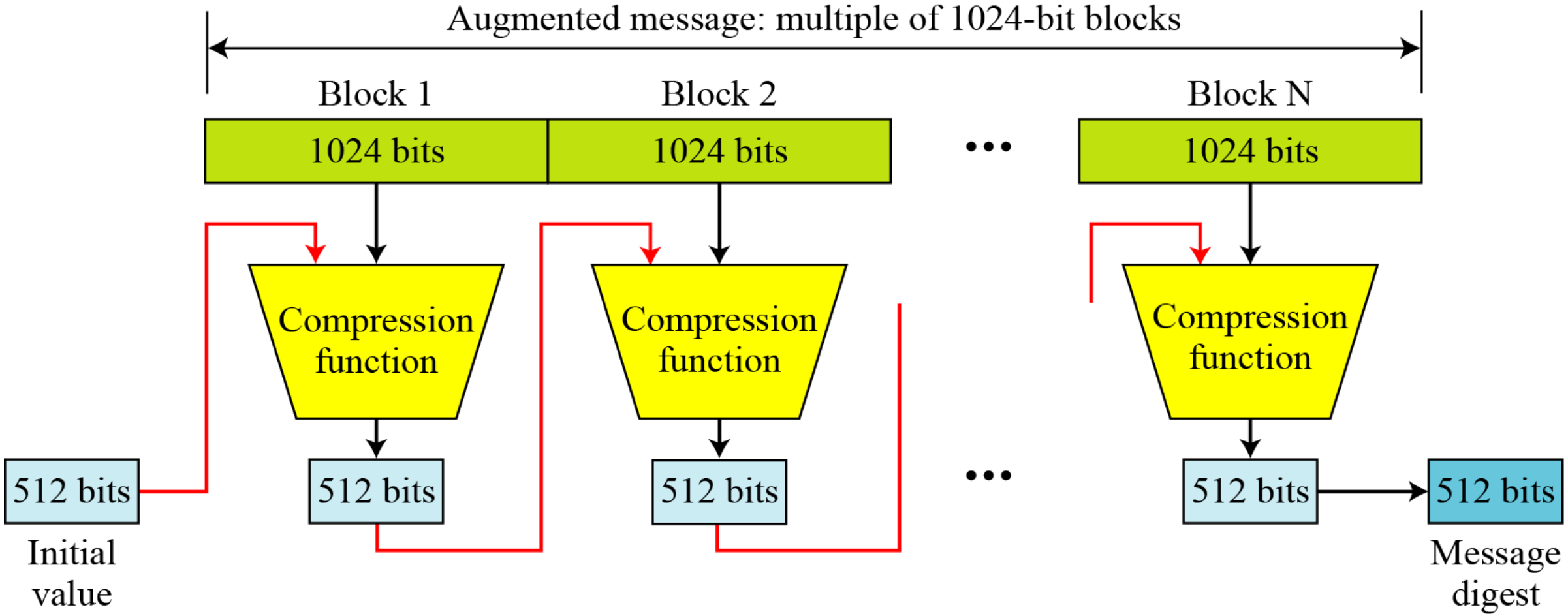
	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Note: All sizes are measured in bits.

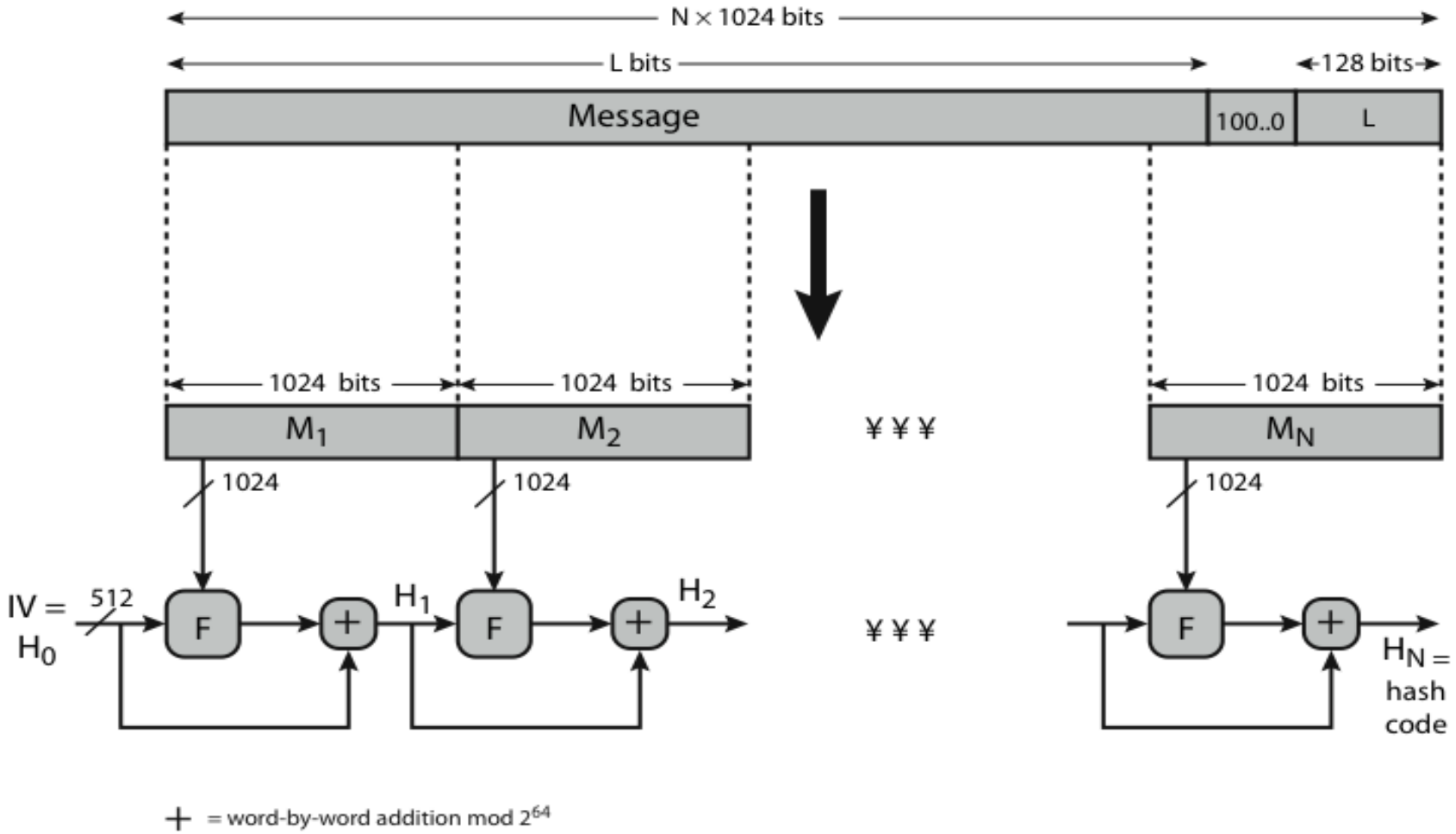
SHA-512

- The algorithm takes as input a message with a maximum length of 2^ 128 bits and produces as output a 512-bit message digest.
- The input is processed in 1024-bit blocks..
- **Steps**
- Step 1: Append padding bits
- Step 2: Append length
- Step 3: Initialize hash buffer
- Step 4: Process the message in 1024-bit (128X 8) blocks, which forms the heart of the algorithm
- Step 5: Output the final state value as the resulting hash

SHA-512 Overview



SHA-512 Overview



$H_0 = \text{IV}$

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

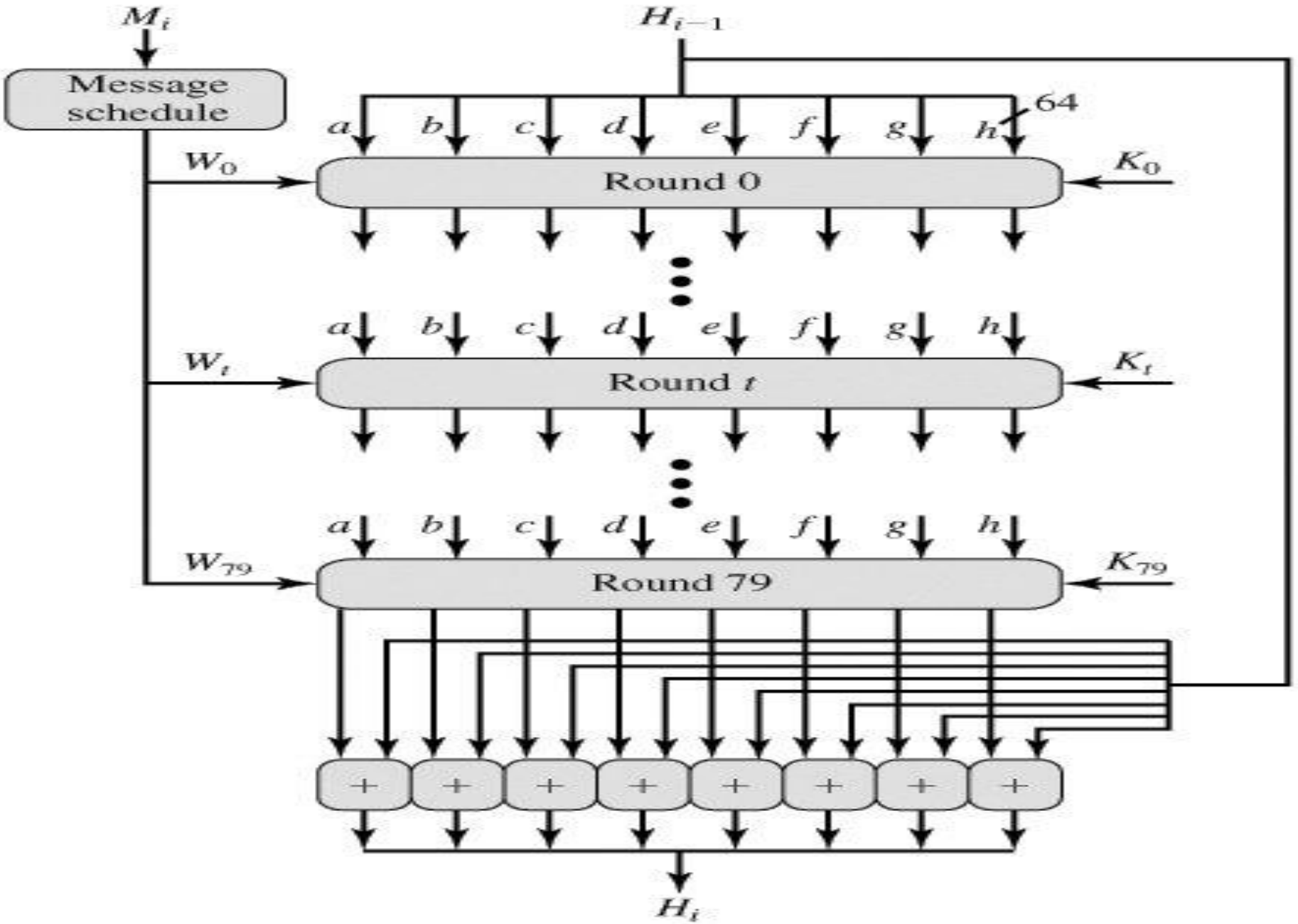
$MD = H_N$

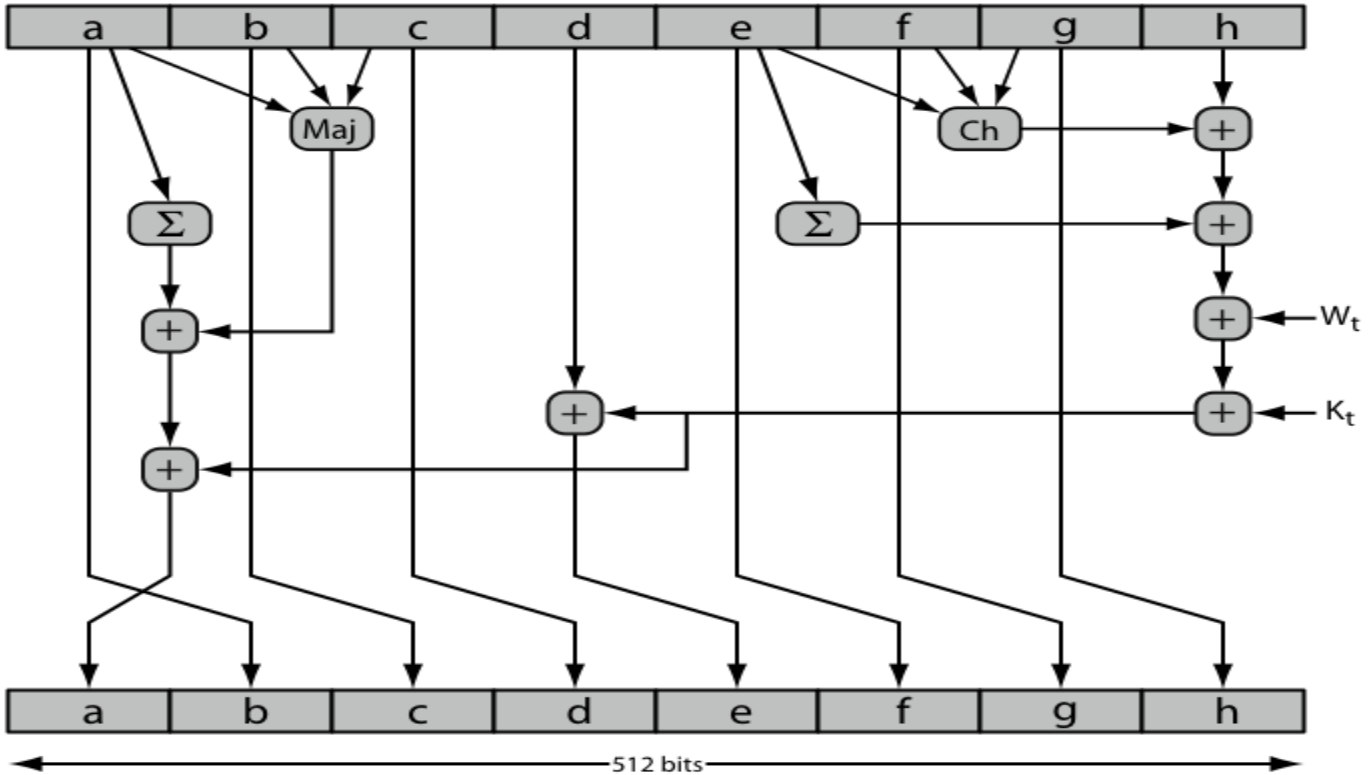
where

IV	= initial value of the abcdefgh buffer, defined in step 3
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 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
  - and a round constant based on cube root of first 80 prime numbers


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



$$T_1 = h + \mathrm{Ch}(e,f,g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a\right) + \mathrm{Maj}(a,b,c)$$

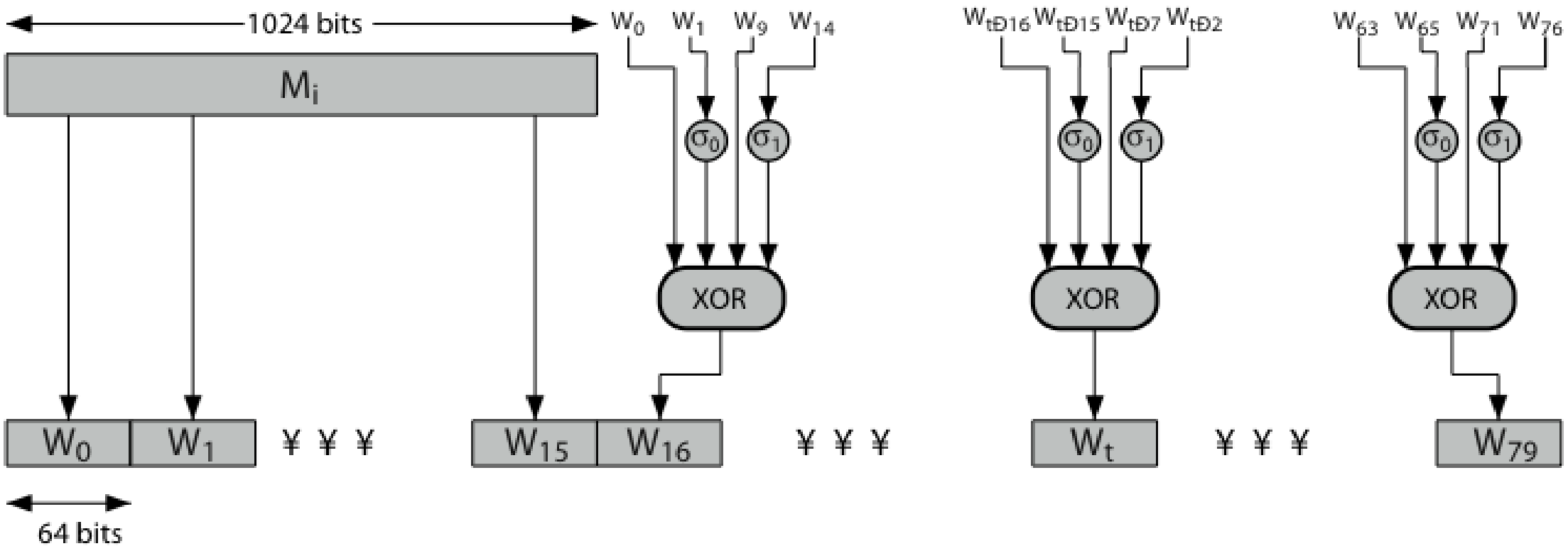
$$a = T_1 + T_2$$

Ch(e, f, g) = (e AND f) $\oplus$ (NOT e AND g) *the conditional function*: *If* e then f else g

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.*

$$\left(\sum_0^{512} a\right) \quad = \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$$

$$\left(\sum_1^{512} e\right) \quad = \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$$



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

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

where

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

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

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

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

$+$ = addition modulo 2^{64}

Keyed Hash Functions as MACs (HMAC)

MAC based on a hash function

- Hash function alone do not provide authentication
  - hash functions are generally faster
  - hash includes a key along with message

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

HMAC Algorithm

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC (including the padding specified in the embedded hash function)

Y_i = i th block of M, $0 \leq i \leq (L - 1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key; recommended length is $\geq n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key

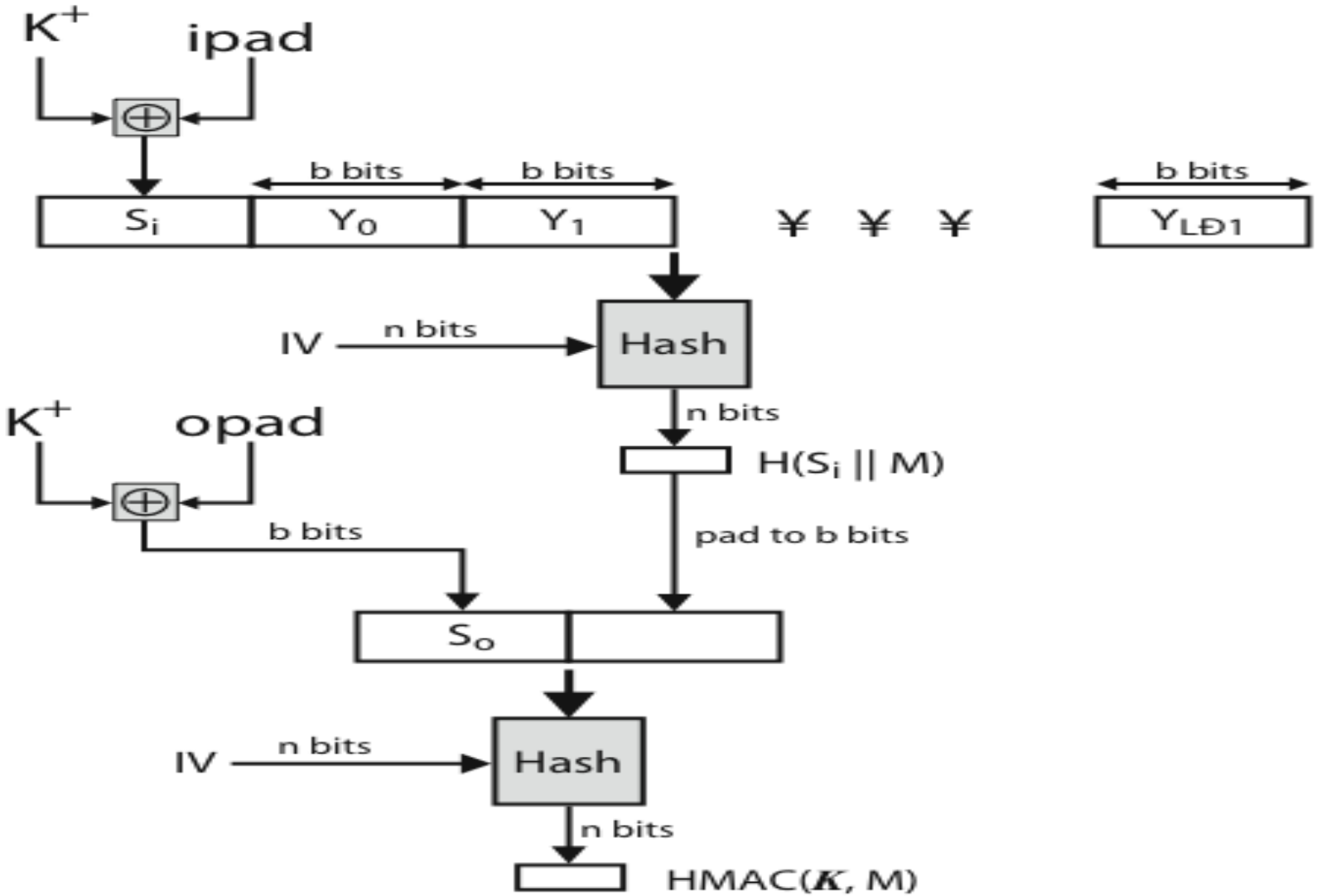
HMAC…

K^+ = K padded with zeros on the left so that the result is b bits in length

ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times

opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \,||\, \text{H}[(K^+ \oplus \text{ipad}) \,||\, M]]$$



$$\mathrm{HMAC}(K, M) = \mathrm{H}[(K^{+} \oplus \mathrm{opad}) \| \mathrm{H}[(K^{+} \oplus \mathrm{ipad}) \| M]]$$

HMAC Algorithm

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zeroes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

Cipher-based Message Authentication Code (CMAC)

I) the message is an integer multiple *n* of the cipher block length *b*.

For AES, $b = 128$ and for triple DES, $b = 64$

k-bit encryption key K

and an n-bit constant $K1$.

T = message authentication code, also referred to as the tag

$Tlen$ = bit length of T

$MSB_s(X)$ = the s leftmost bits of the bit string X

$$C_1 = E(K, M_1)$$

$$C_2 = E(K, [M_2 \oplus C_1])$$

$$C_3 = E(K, [M_3 \oplus C_2])$$

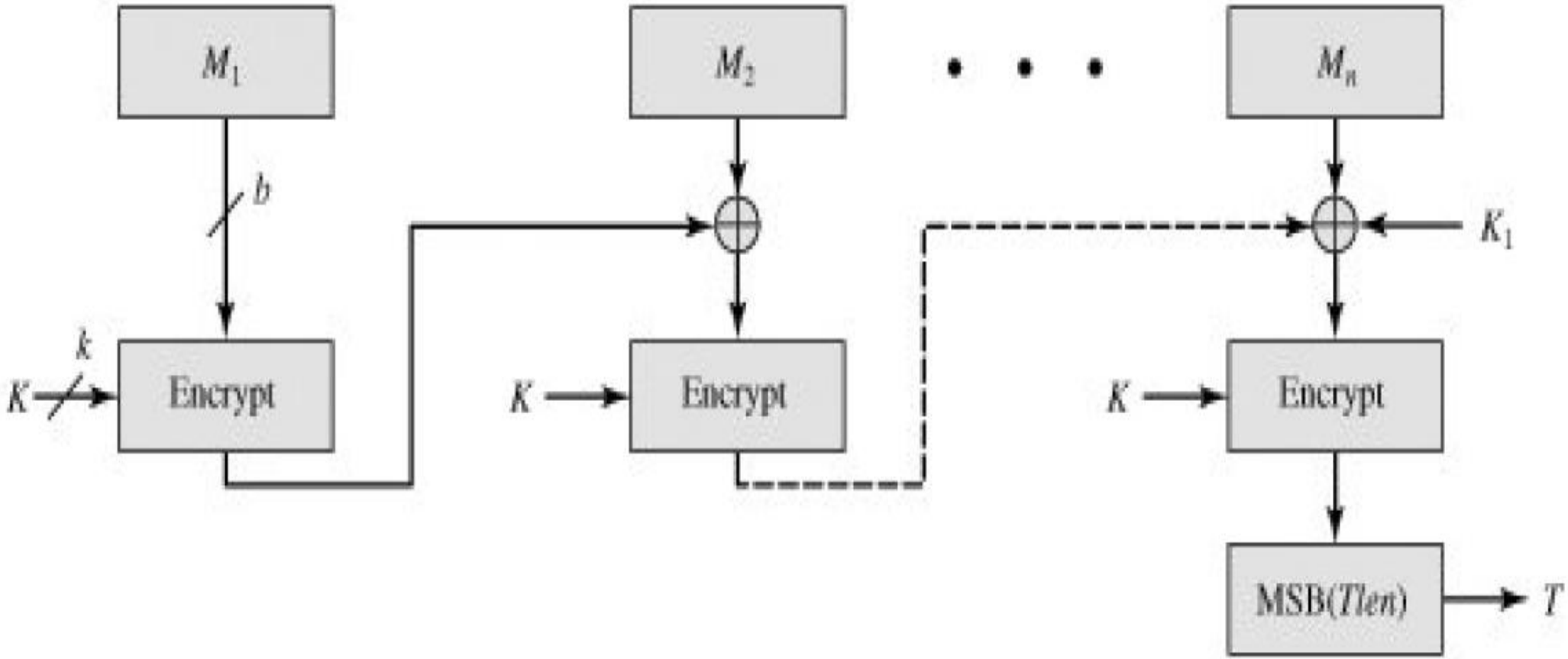
.

.

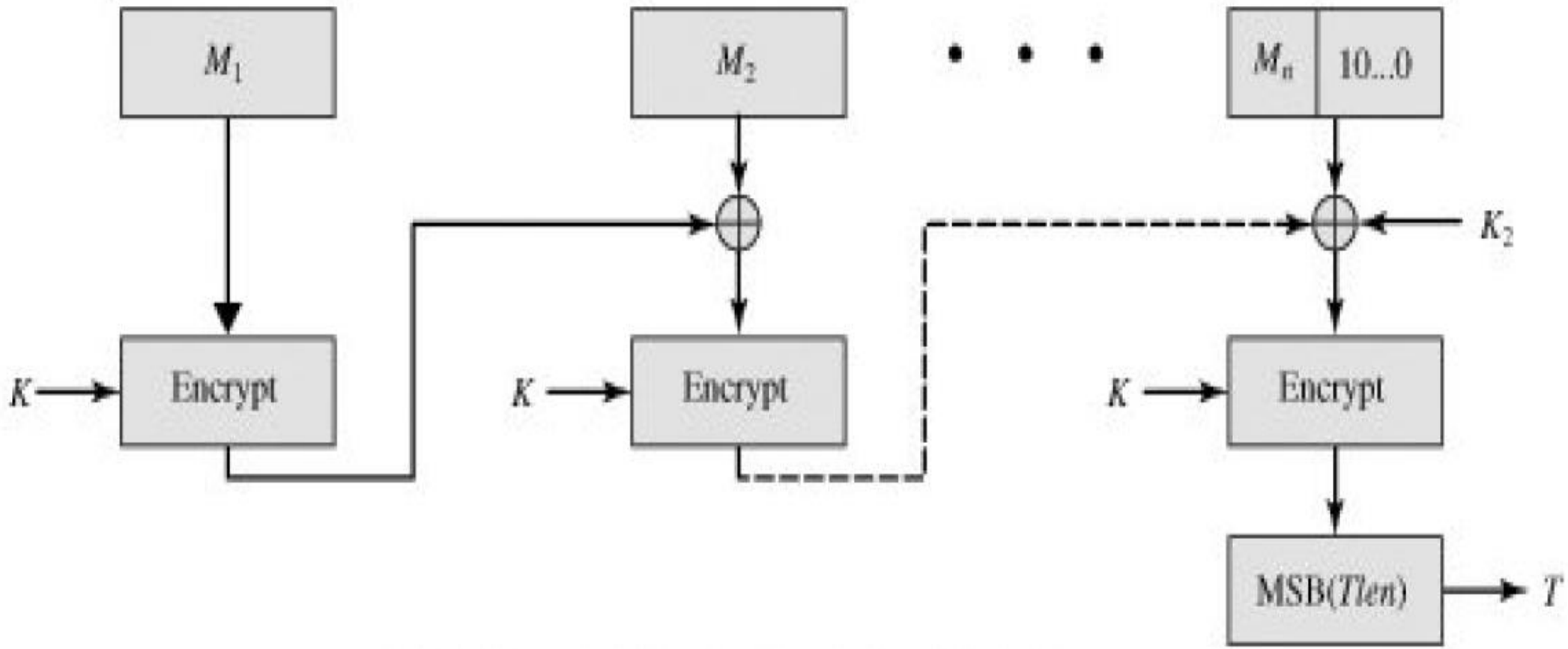
.

$$C_n = E(K, [M_n \oplus C_{n1} \oplus K_1])$$

$$T = MSB_{Tlen}(C_n)$$



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Digital Signatures

- Digital signatures provide the ability to:
  - Verify author, date & time of signature
  - Authenticate message contents
  - Be verified by third parties to resolve disputes
- Hence include authentication function with additional capabilities

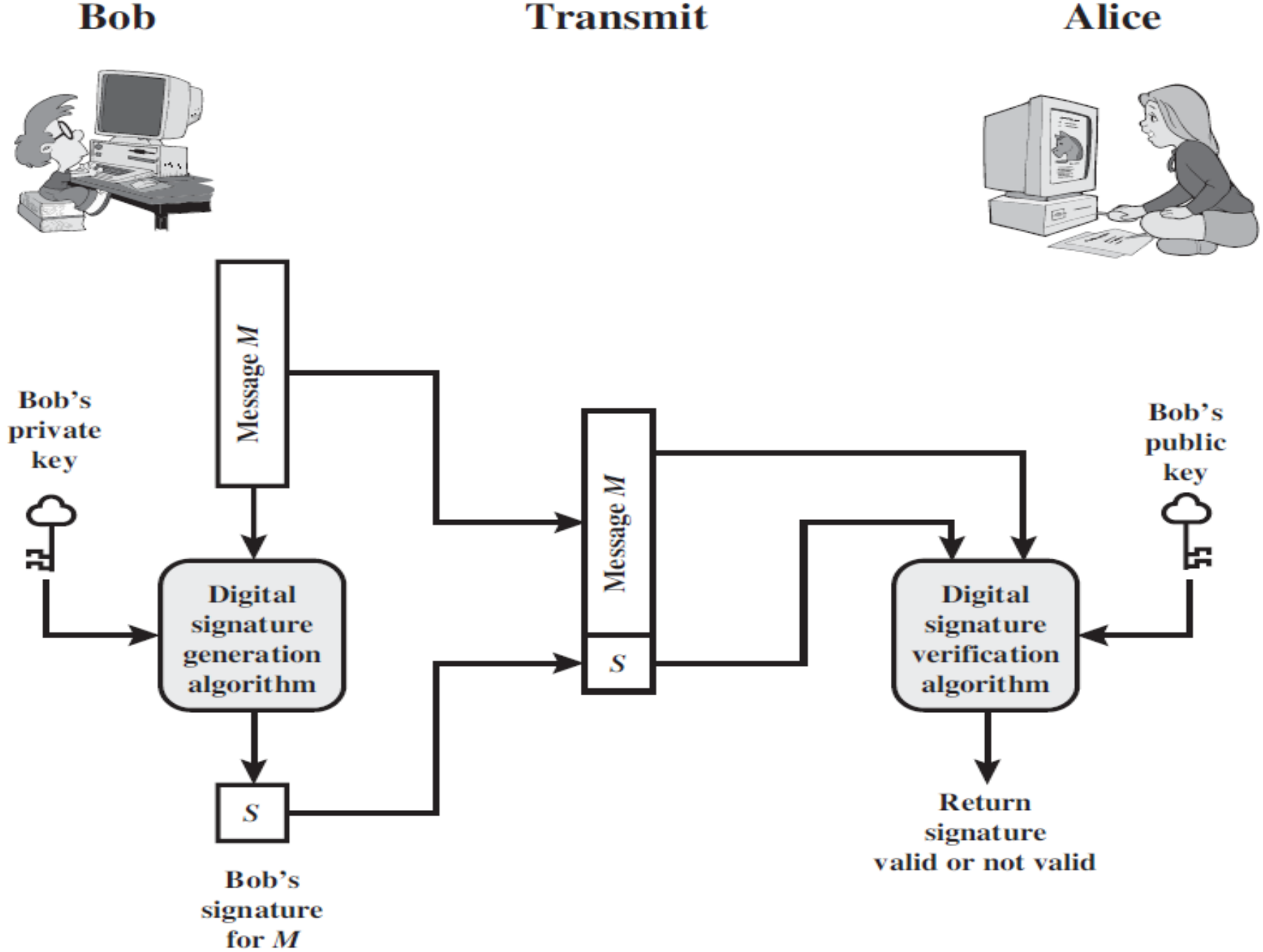


Figure 13.1 Generic Model of Digital Signature Process

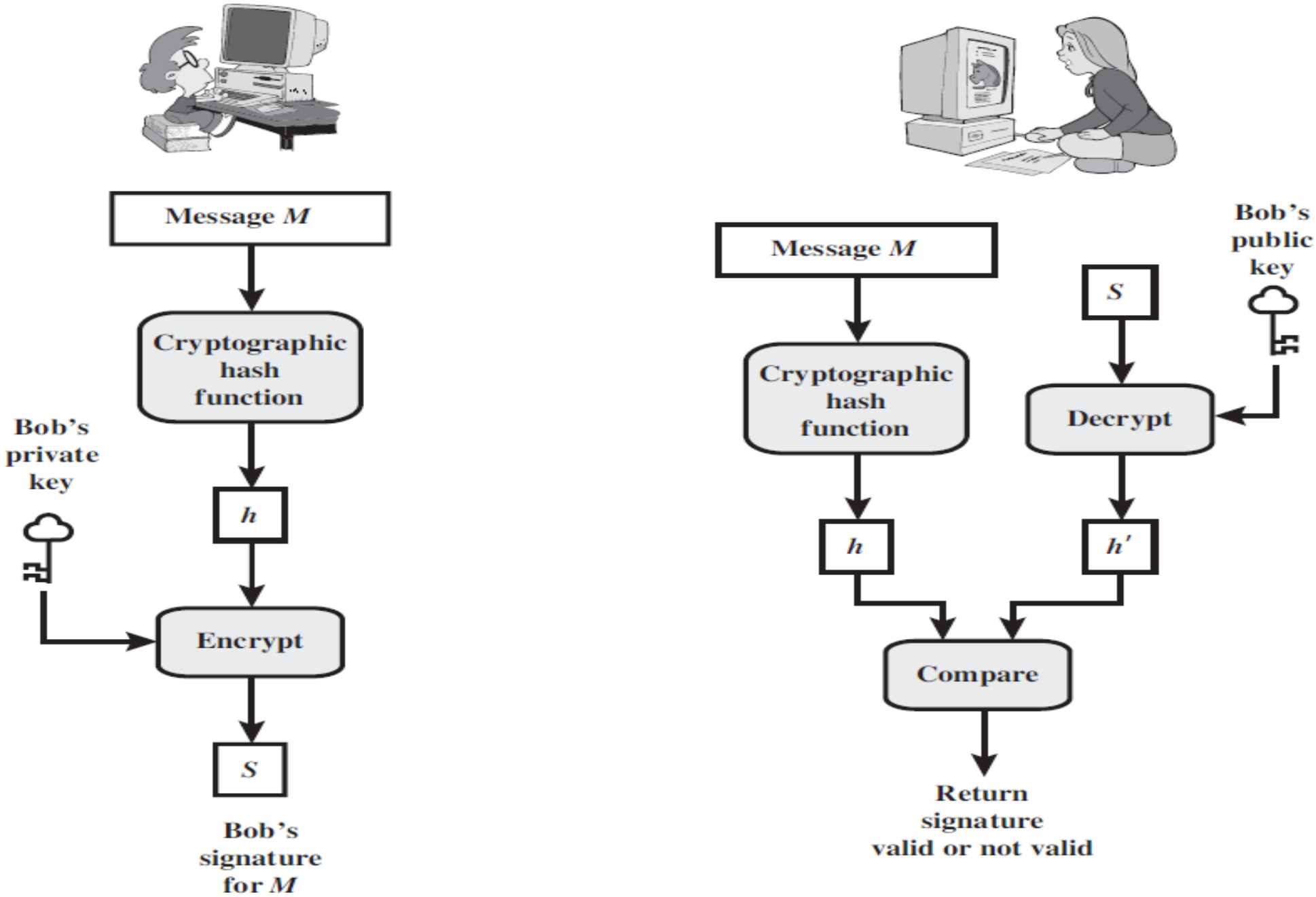


Figure 13.2 Simplified Depiction of Essential Elements of Digital Signature Process

Attacks and Forgeries

- **Key-only attack:** C only knows A's public key.
- **Known message attack:** C is given access to a set of messages and their signatures.
- **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key.
  - C then obtains from A valid signatures for the chosen messages.
  - The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.

Attacks and Forgeries…

- **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- **Adaptive chosen message attack:** C is allowed to use A as an "oracle."

This means A may request signatures of messages that depend on previously obtained message–signature pairs.

Attacks and Forgeries…

- If C succeeds at breaking a signature scheme C can do any of the following with a non-negligible probability
- **Total break:** C determines A's private key.
- **Universal forgery:** C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery:** C forges a signature for a particular message chosen by C.
- **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Digital Signature Requirements

- Must depend on the message signed
- Must use some information unique to sender
  - To prevent both forgery and denial
- Must be relatively easy to produce
- Must be relatively easy to recognize & verify
- Be computationally infeasible to forge
- Be practical to save digital signature in storage

Direct Digital Signatures

- Involve only sender & receiver
- Assumed receiver has sender’s public-key
- Digital signature made by sender signing entire message or hash with private-key
- Can encrypt using receivers public-key
- Important that sign first then encrypt message & signature
- Security depends on sender’s private-key

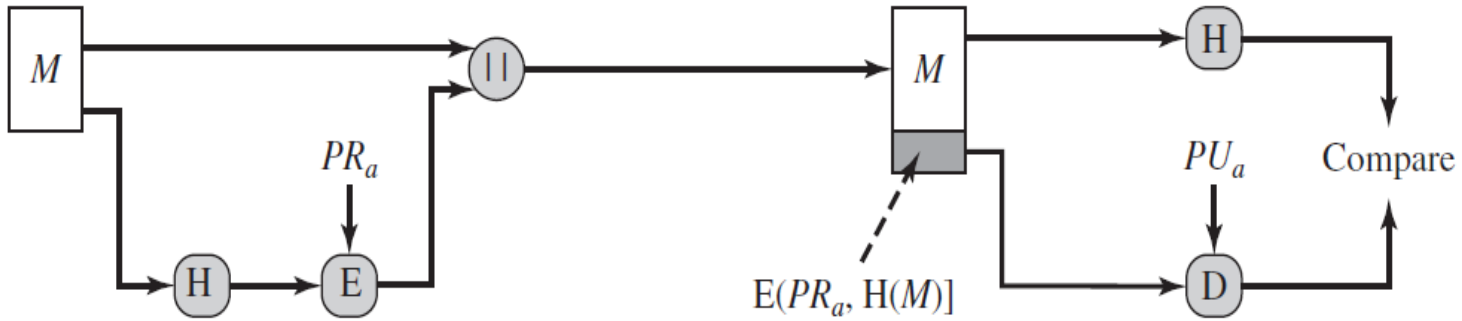
Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

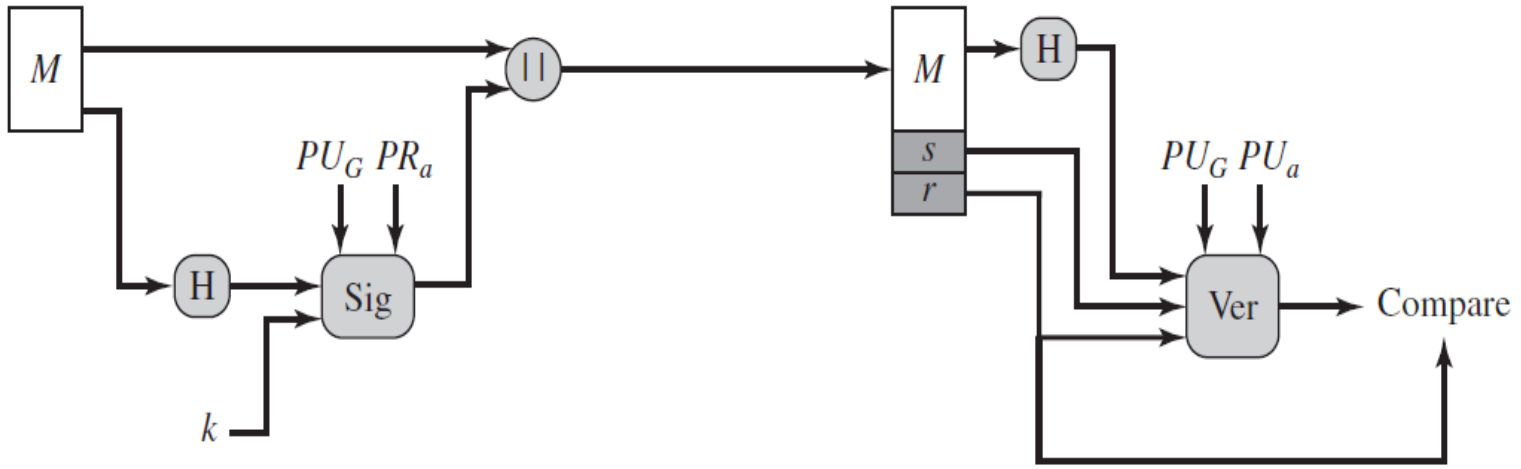
Digital Signature Standard (DSS)

- US govt approved signature scheme
- Designed by NIST in early 90's
- Revised in 1993, 1996 & then 2000
- Uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm
- It is a public key technique

Digital Signature Algorithm (DSA)



(a) RSA approach



(b) DSS approach

Figure 13.3 Two Approaches to Digital Signatures

Digital Signature Algorithm (DSA)

- Smaller and faster than RSA
- A digital signature scheme only
- Security depends on difficulty of computing discrete logarithms

DSA

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length of between 512 and 1024 bits
in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$;
i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1} (\mathrm{H}(M) + xr)] \bmod q$

Signature = (r, s)

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [\mathrm{H}(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

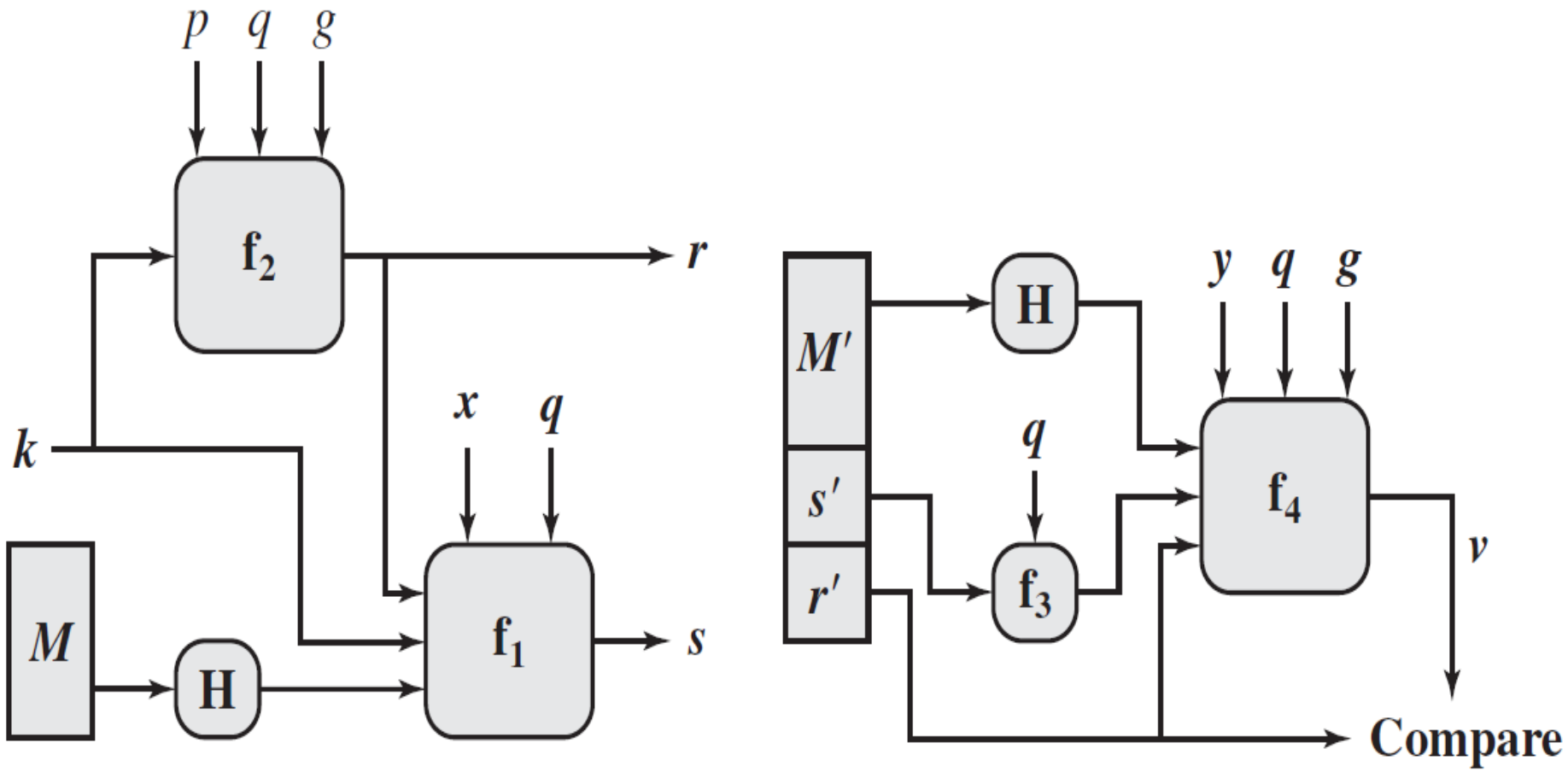
TEST: $v = r'$

M = message to be signed

$\mathrm{H}(M)$ = hash of M using SHA-1

M', r', s' = received versions of M, r, s

DSA



$$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

(a) Signing

$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r')$$

$$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$$

(b) Verifying

Figure 13.5 DSS Signing and Verifying