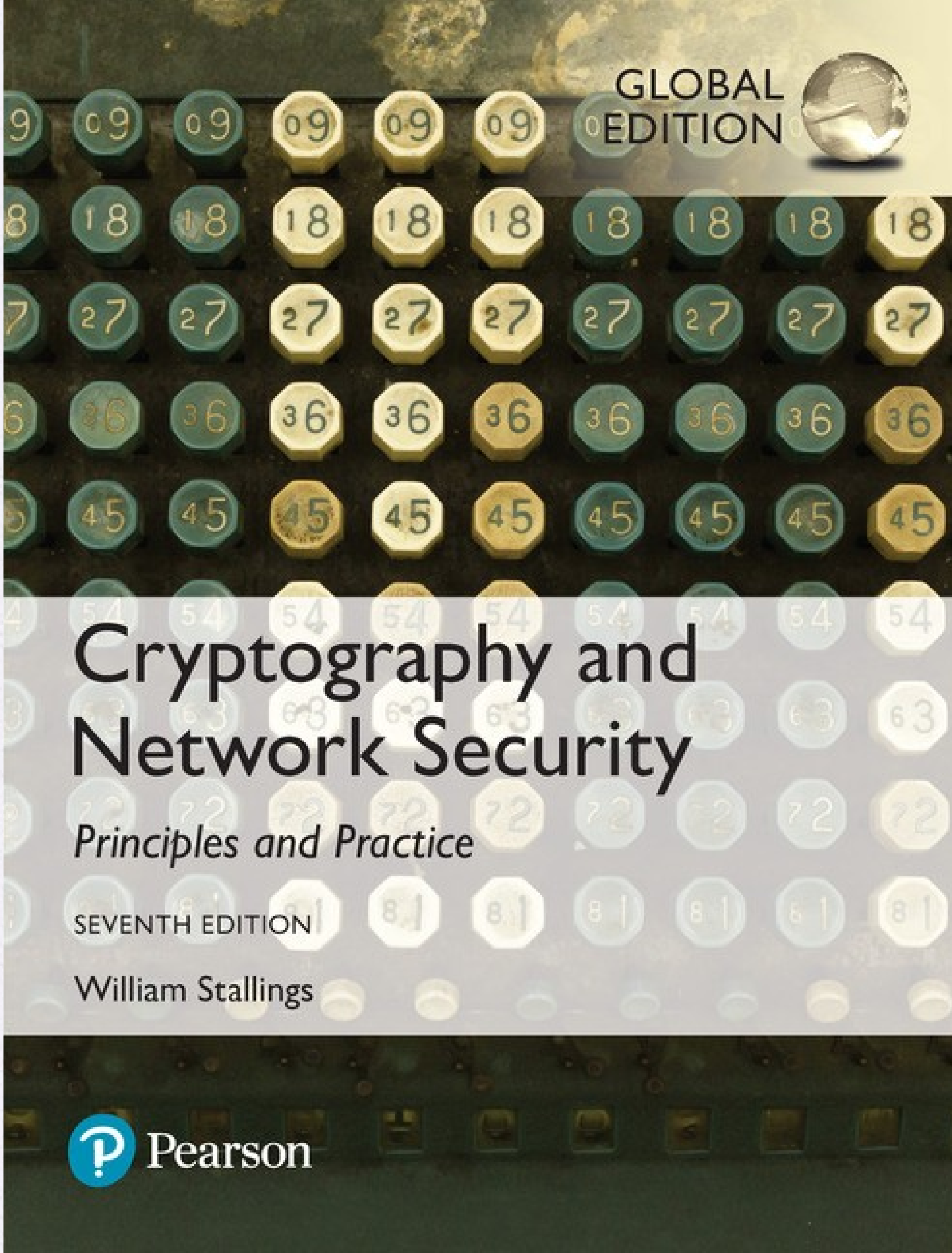


GLOBAL EDITION

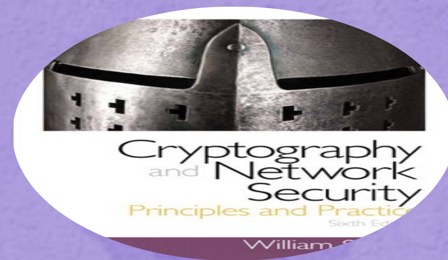
Cryptography and Network Security

Principles and Practice

SEVENTH EDITION

William Stallings

Pearson



Chapter 11

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
  - $h = \mathrm{H}(M)$
  - Principal object is data integrity
- Cryptographic hash function
  - An algorithm for which it is computationally infeasible to find either:

    (a) a data object that maps to a pre-specified hash result (the one-way property)

    (b) two data objects that map to the same hash result (the collision-free property)
- Hash functions are often used to determine whether or not data has changed

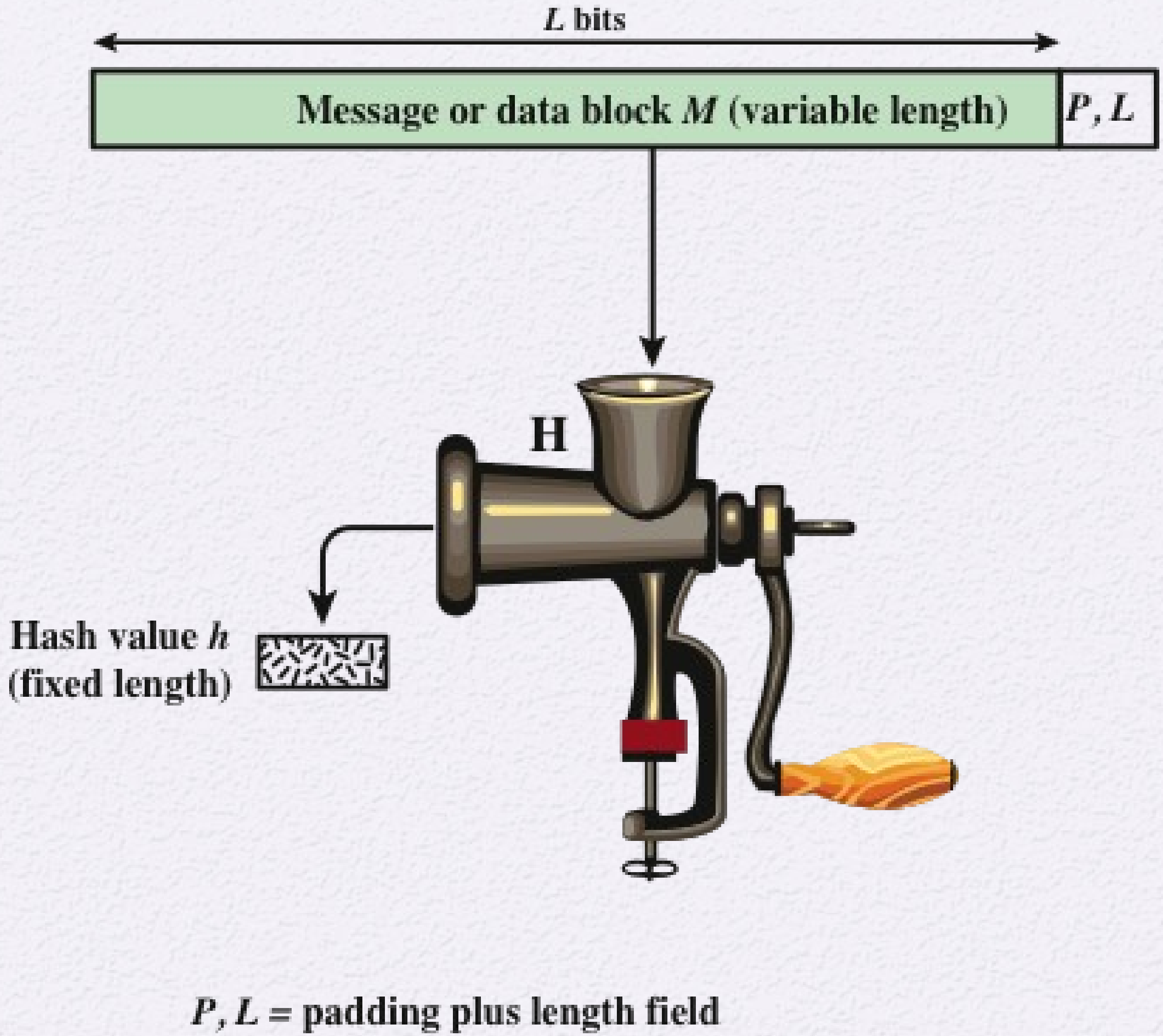


Figure 11.1 Cryptographic Hash Function; $h = H(M)$

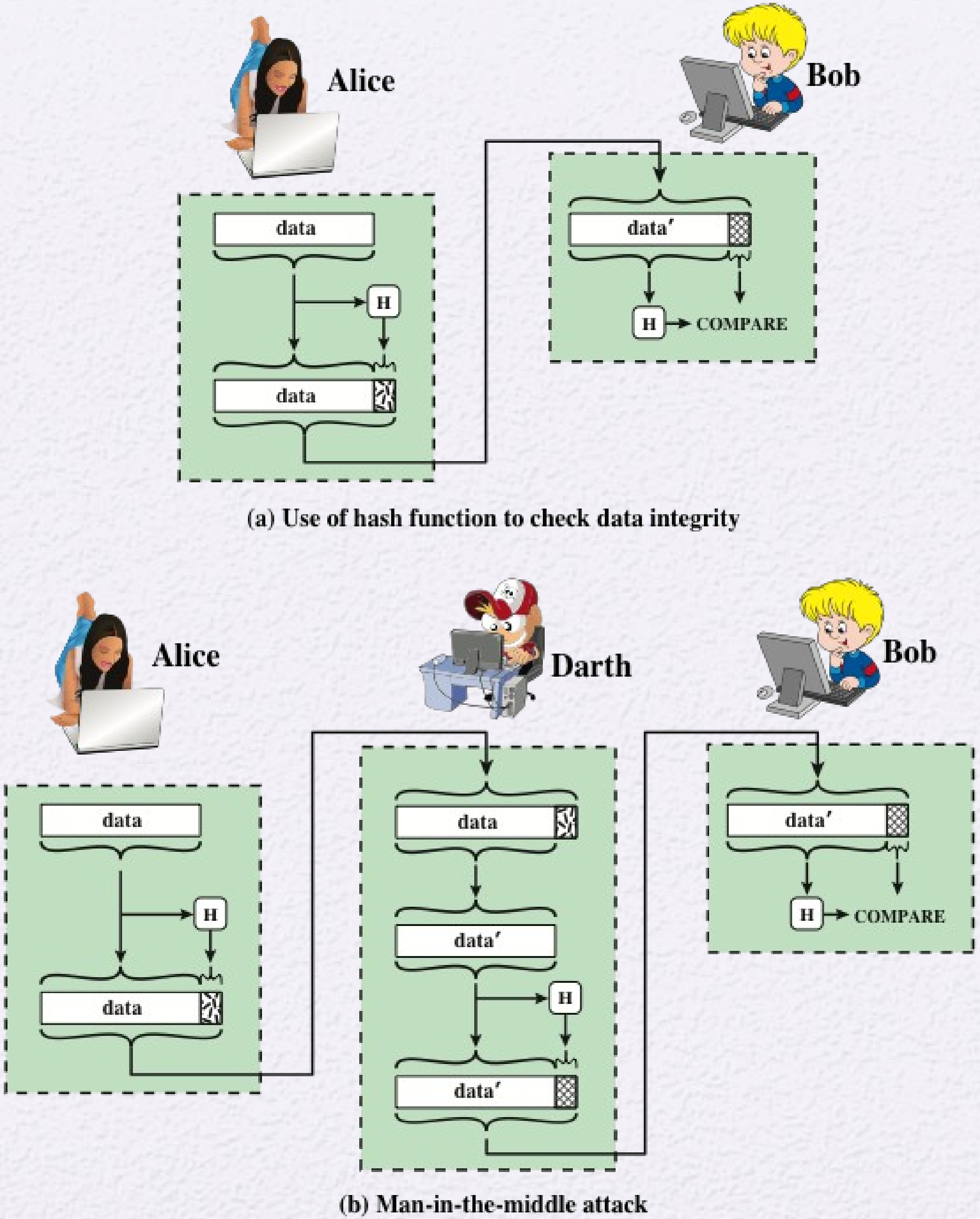


Figure 11.2 Attack Against Hash Function

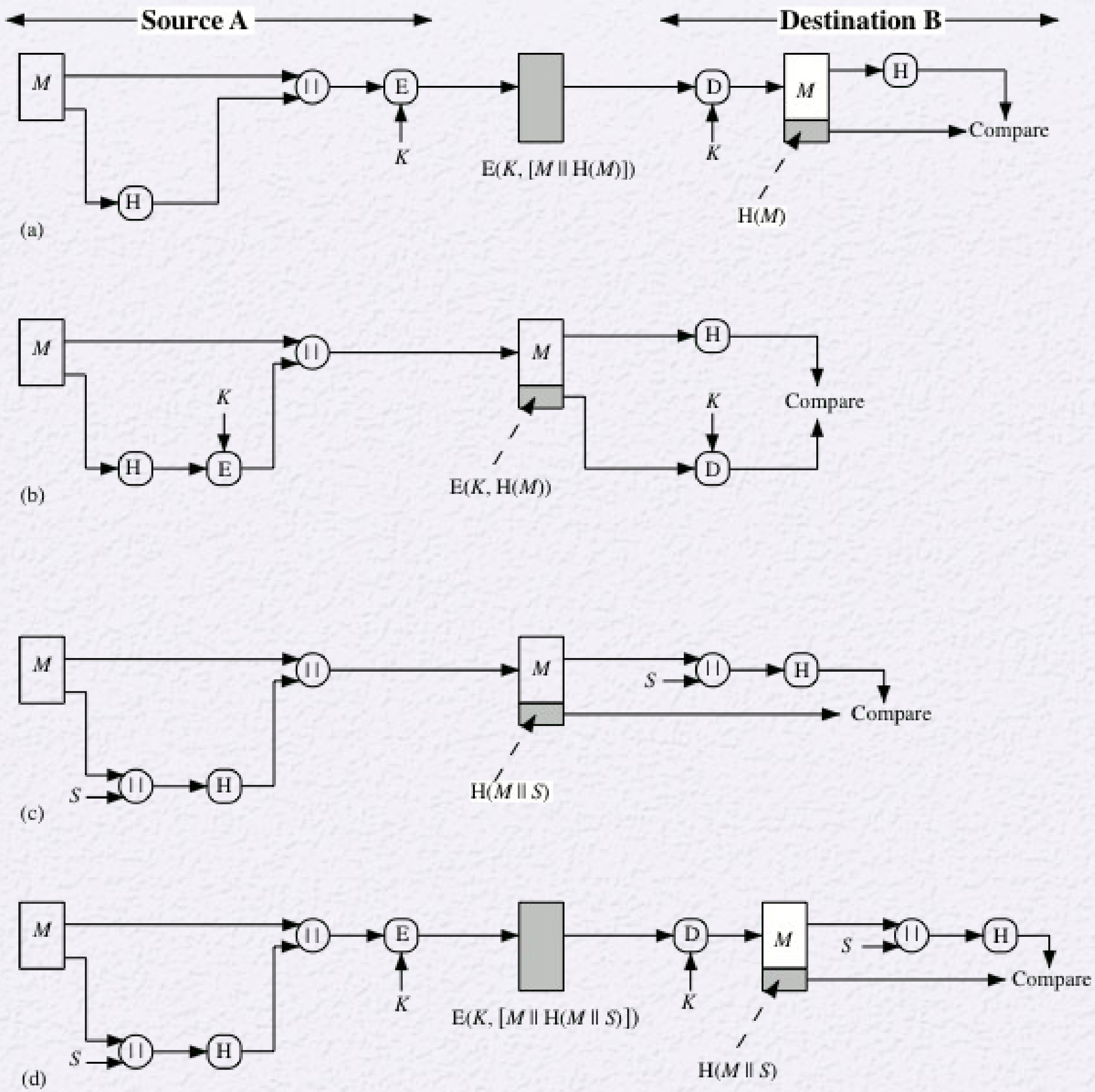


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

Message Authentication Code (MAC)

- Also known as a *keyed hash function*
- Typically used between two parties that share a secret key to authenticate information exchanged between those parties

Takes as input a secret key and a data block and produces a hash value (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

Digital Signature

- Operation is similar to that of the MAC
- 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
- An attacker who wishes to alter the message would need to know the user's private key
- Implications of digital signatures go beyond just message authentication

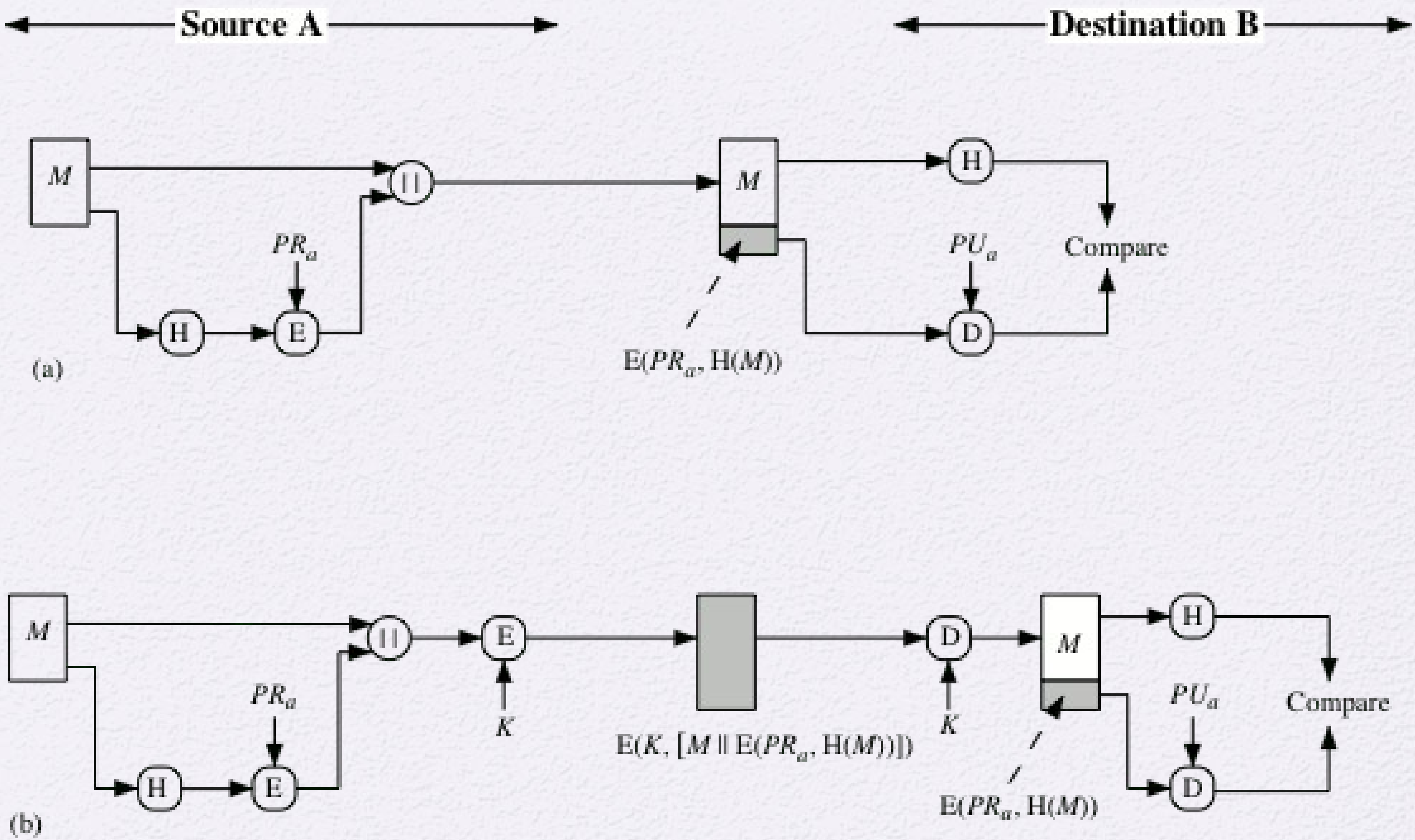


Figure 11.4 Simplified Examples of Digital Signatures

Other Hash Function Uses

Commonly used to create a one-way password file

- When a user enters a password, the hash of that password is compared to the stored hash value for verification
- This approach to password protection is used by most operating systems

Can be used for intrusion and virus detection

- Store H(F) for each file on a system and secure the hash values
- One can later determine if a file has been modified by recomputing H(F)
- An intruder would need to change F without changing H(F)

Can be 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

Requirements and Security

Preimage

- *x* is the preimage of *h* for a hash value $h = \text{H}(x)$
- 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

Collision

- Occurs if we have $x \neq y$ and $\text{H}(x) = \text{H}(y)$
- Because we are using hash functions for data integrity, collisions are clearly undesirable

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

(Table can be found on page 323 in textbook.)

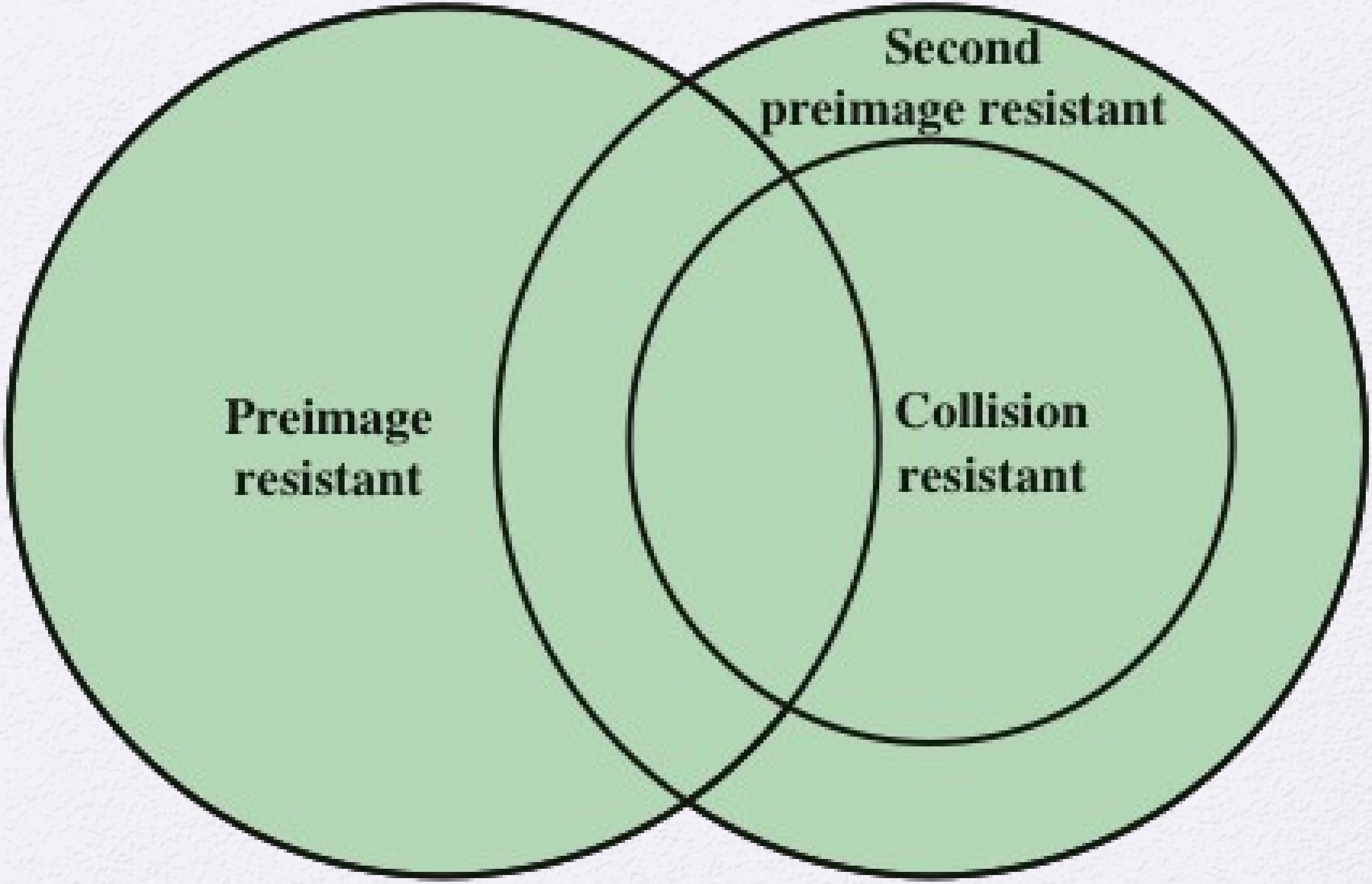


Figure 11.6 Relationship Among Hash Function Properties

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

Attacks on Hash Functions

Brute-Force Attacks

- Does not depend on the specific algorithm, only depends on bit length
- In the case of a hash function, attack depends only on the bit length of the hash value
- Method is to pick values at random and try each one until a collision occurs

Cryptanalysis

- An attack based on weaknesses in a particular cryptographic algorithm
- Seek to exploit some property of the algorithm to perform some attack other than an exhaustive search

Collision Resistant Attacks

The birthday paradox
If we choose random variables from a uniform distribution in the range 0 through *N* - 1, then the probability that a repeated element is encountered exceeds 0.5 after choices have been made. Thus, for an *m*-bit hash value, if we pick data blocks at random, we can expect to find two data blocks with the same hash value within attempts.

- Yuval proposed the following strategy to exploit the birthday paradox in a collision resistant attack:
  - The source (A) is prepared to sign a legitimate message *x* by appending the appropriate *m*-bit hash code and encrypting that hash code with A's private key
  - Opponent generates $2^{m/2}$ variations *x'* of *x*, all with essentially the same meaning, and stores the messages and their hash values
  - Opponent prepares a fraudulent message *y* for which A's signature is desired
  - Opponent generates minor variations y' of *y*, all of which convey essentially the same meaning. For each y', the opponent computes H (y'), checks for matches with any of the H (x') values, and continues until a match is found. That is, the process continues until a y' is generated with a hash value equal to the hash value of one of the x' values
  - The opponent offers the valid variation to A for signature which can then be attached to the fraudulent variation for transmission to the intended recipient
    - Because the two variations have the same hash code, they will produce the same signature and the opponent is assured of success even though the encryption key is not known

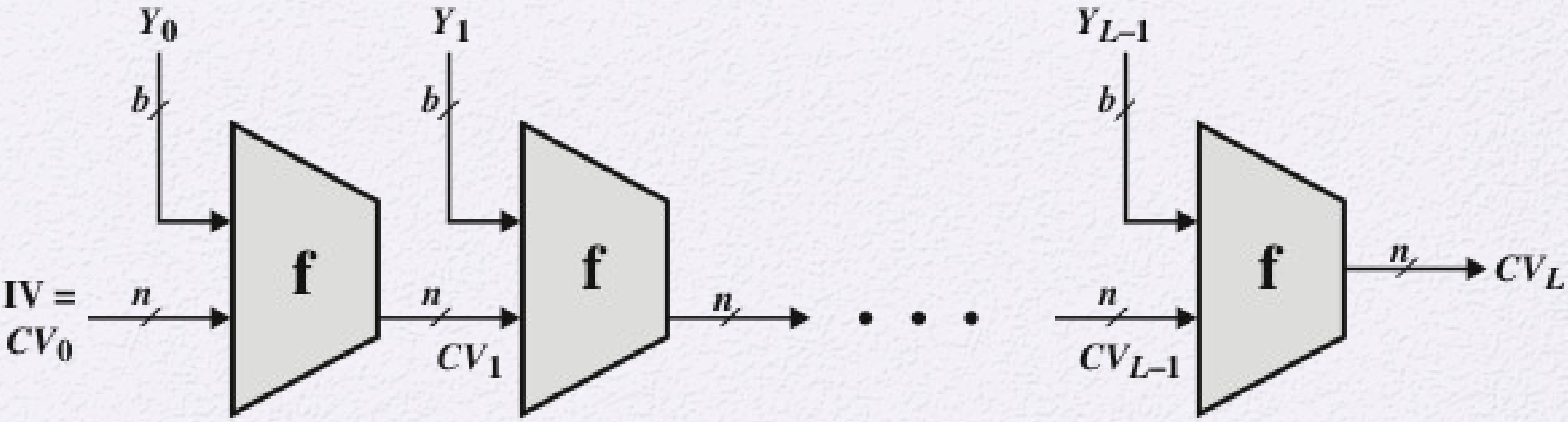
A Letter in 2^{38} Variation

(Letter is located on page 334 in textbook)

As {the / --} Dean of Blakewell College, I have {had the pleasure of knowing / known} Cherise Rosetti for the {last / past} four years. She {has been / was} {a tremendous / an outstanding} {asset to / role model in} {our / the} school. I {would like to take this opportunity to / wholeheartedly} recommend Cherise for your {school's / --} graduate program. I {am / feel} {confident / certain} {that / - -} {she / Cherise} will {continue to / - -} succeed in her studies. {She / Cherise} is a dedicated student and {thus far her grades / her grades thus far} {have been / are} {exemplary / excellent}. In class, {she / Cherise} {has proven to be / has been} a take-charge {person / individual} {who is / - -} able to successfully develop plans and implement them.

{She / Cherise} has also assisted {us / - -} in our admissions office. {She / Cherise} has {successfully / - -} demonstrated leadership ability by counseling new and prospective students. {Her / Cherise's} advice has been {a great / of considerable} help to these students, many of whom have {taken time to share / shared} their comments with me regarding her pleasant and {encouraging / reassuring} attitude. {For these reasons / It is for these reasons that} I {highly recommend / offer high recommendations for} Cherise {without reservation / unreservedly}. Her {ambition / drive} and {abilities / potential} will {truly / surely} be an {asset to / plus for} your {establishment / school}.

Figure 11.7 A Letter in 2^{38} Variations



IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	ith input block	b	=	length of input block
f	=	compression algorithm			

Figure 11.8 General Structure of Secure Hash Code

Hash Functions Based on Cipher Block Chaining

Meet-in-the-middle attack

1. Use the algorithm defined at the beginning of this section to calculate the unencrypted hash code .
2. Construct any desired message in the form .
3. Compute .
4. Generate $2^{m/2}$ random blocks; for each block X, compute $E(X, H_{N-2})$. Generate an additional $2^{m/2}$ random blocks; for each block Y, compute D(Y, G), where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $E(X, H_{N-2}) = D(Y, G)$.
6. Form the message . This message has the hash code G and therefore can be used with the intercepted encrypted signature.

- Meet-in-the-middle-attack
  - Another version of the birthday attack used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings
- It can be shown that some form of birthday attack will succeed against any hash scheme involving the use of cipher block chaining without a secret key, provided that either the resulting hash code is small enough or that a larger hash code can be decomposed into independent subcodes

Secure Hash Algorithm (SHA)

- SHA was originally designed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993
- Was revised in 1995 as SHA-1
- Based on the hash function MD4 and its design closely models MD4
- Produces 160-bit hash values
- In 2002 NIST produced a revised version of the standard that defined three new versions of SHA with hash value lengths of 256, 384, and 512
  - Collectively known as SHA-2

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.

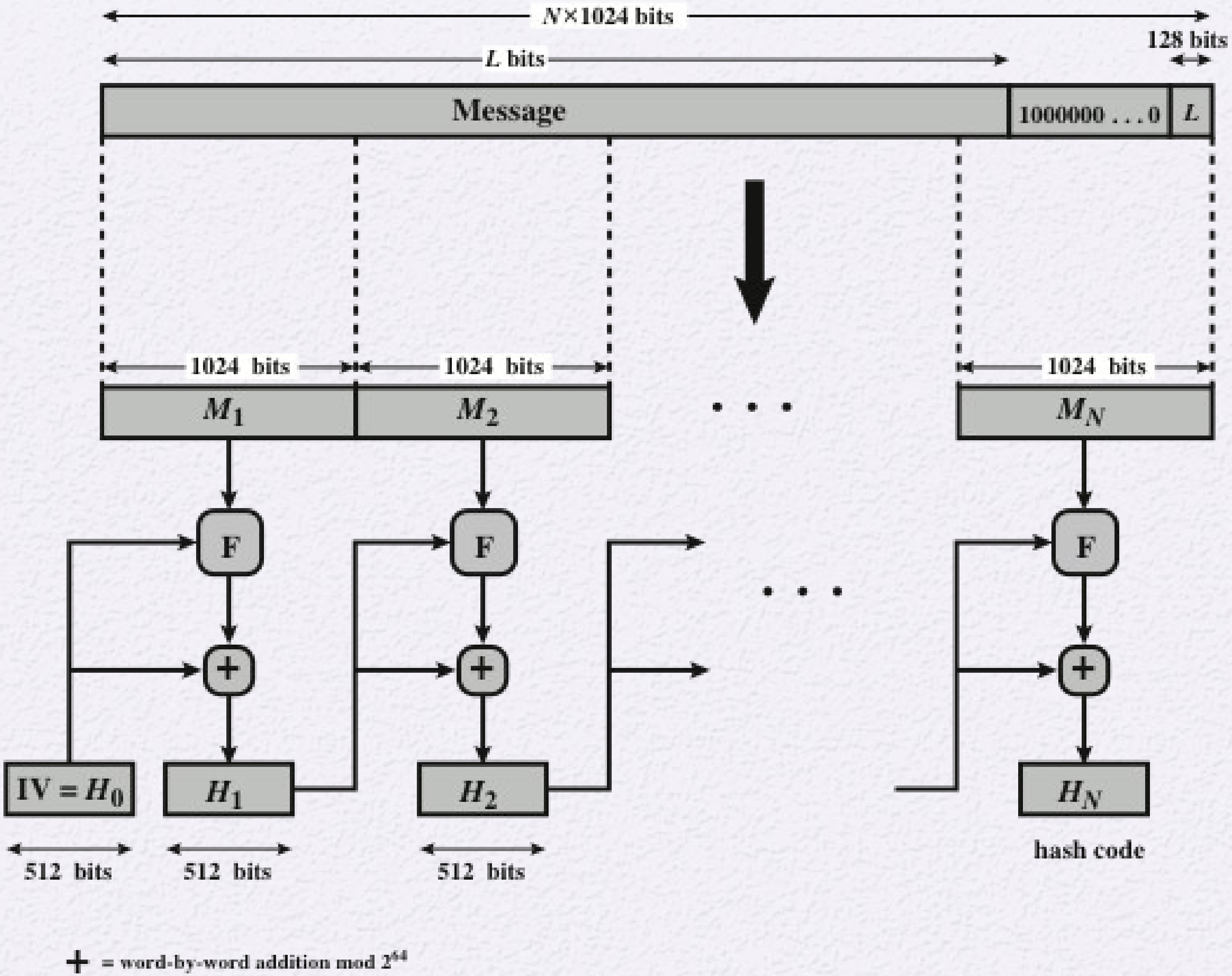


Figure 11.9 Message Digest Generation Using SHA-512

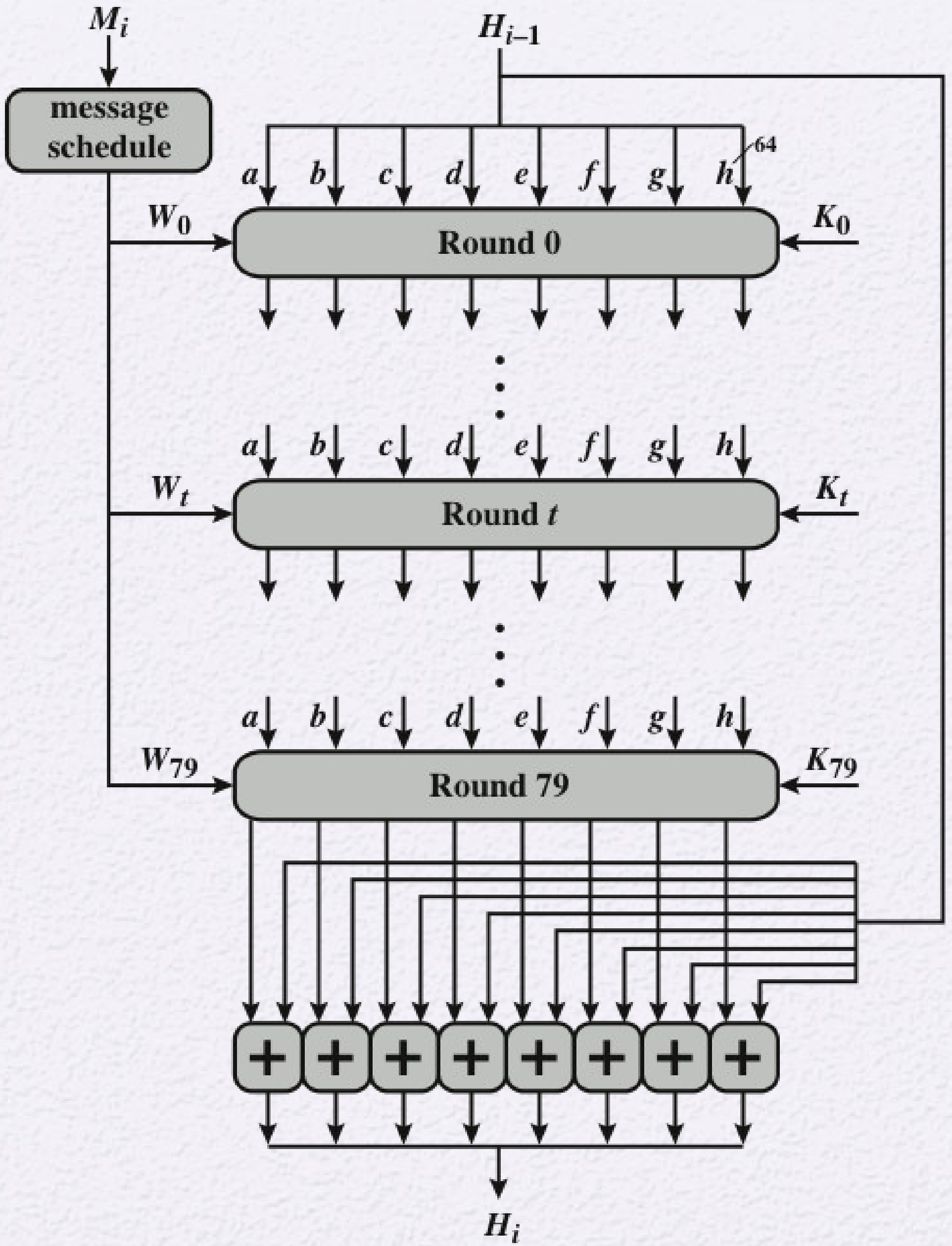


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

Table 11.4 ---- SHA-512 Constants

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

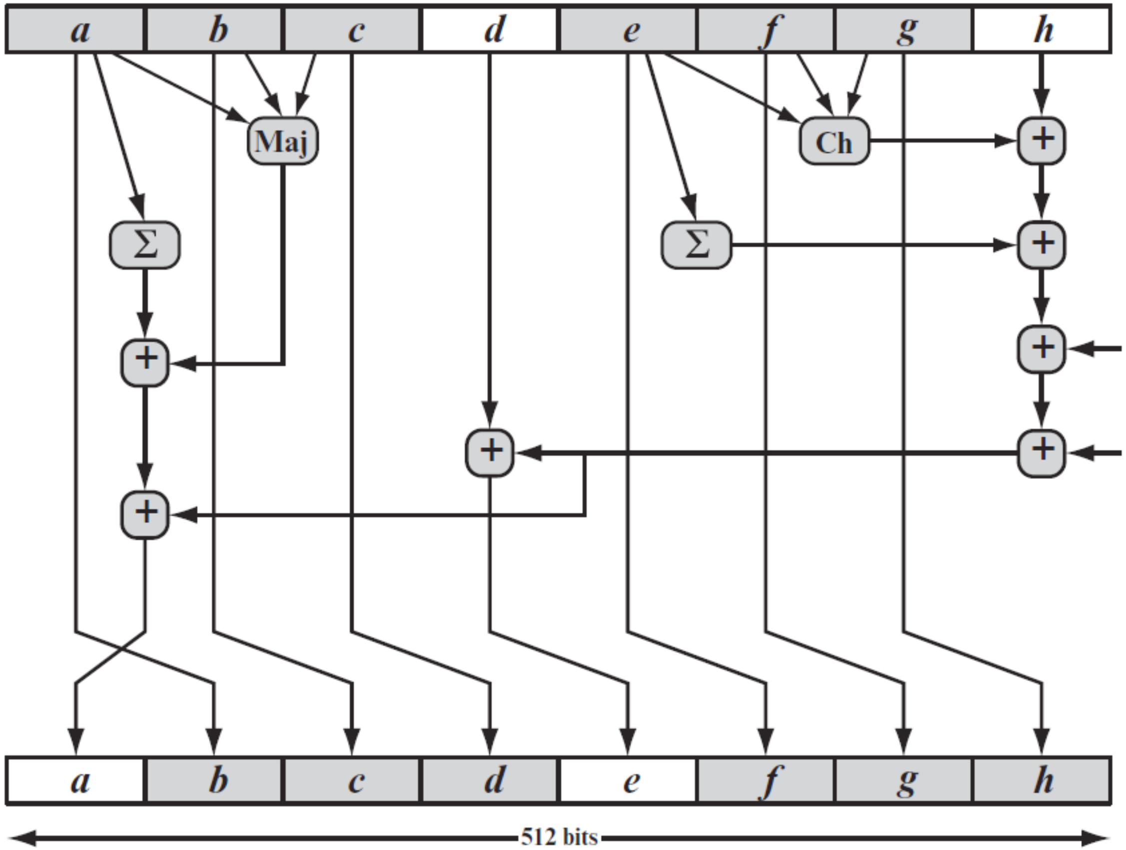


Figure 11.11 Elementary SHA-512 Operation (single round)

- Ch(*e*, *f*, *g*) = (*e* AND *f*) ⊕(NOT *e* AND *g*)
  the conditional function: If e then f else g
- Maj(*a*, *b*, *c*) = (*a* AND *b*) ⊕(*a* AND *c*) ⊕(*b* AND *c*)
  the function is true only of the majority (two or three) of the arguments are true
- () = ROTR^{28}(a) ⊕ROTR^{34}(a) ⊕ROTR^{39}(a)
- () = ROTR^{14}(e) ⊕ROTR^{18}(e) ⊕ROTR^{41}(e)
- ROTR^{n}(x) = circular right shift (rotation) of the 64-bit argument *x* by *n* bits

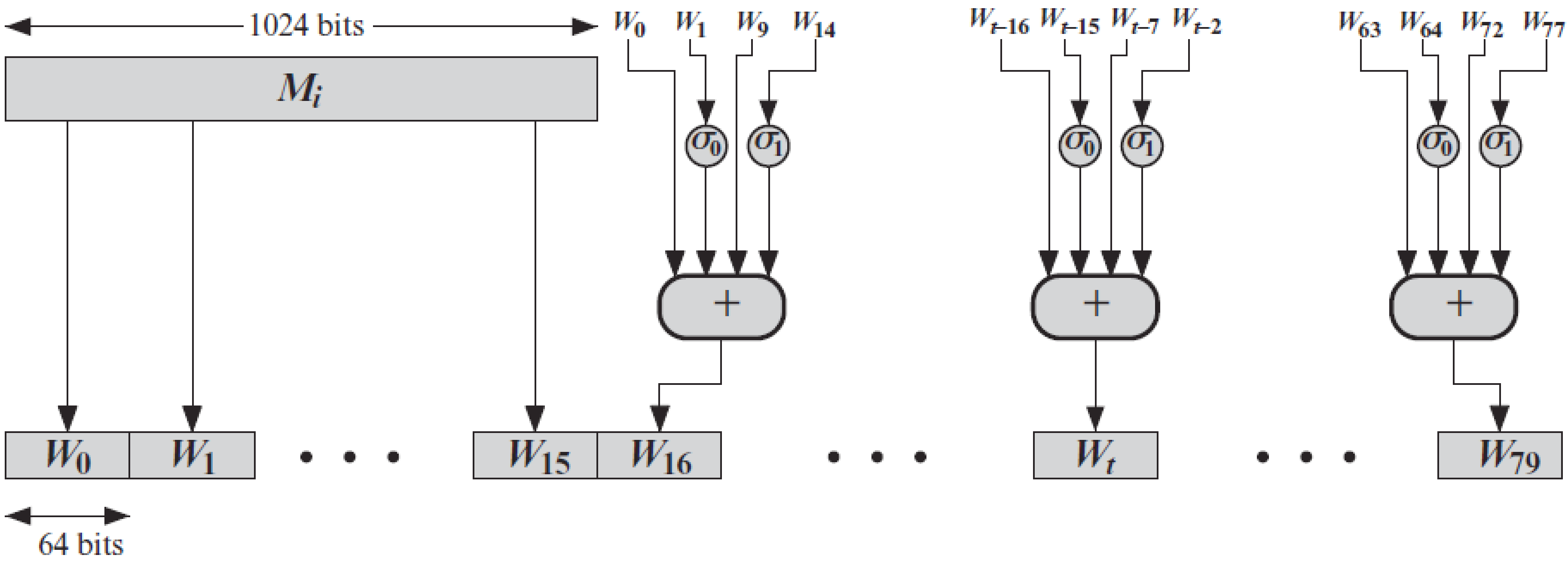


Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

The padded message consists blocks $M_1, M_2, \ldots M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1} \ldots M_{i,15}$. All addition is performed modulo 2^{64}.

$H_{0,0}$ = 6A09E667F3BCC908 $H_{0,4}$ = 510E527FADE682D1
$H_{0,1}$ = BB67AE8584CAA73B $H_{0,5}$ = 9B05688C2B3E6C1F
$H_{0,2}$ = 3C6EF372FE94F82B $H_{0,6}$ = 1F83D9ABFB41BD6B
$H_{0,3}$ = A54FF53A5F1D36F1 $H_{0,7}$ = 5BE0CDI9137E2179

for $i = 1$ **to** N

1. Prepare the message schedule W:
   for $t = 0$ **to** 15
   $W_t = M_{i,t}$
   for t = 16 **to** 79
   $W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$
2. Initialize the working variables
   $a = H_{i-1,0}$ $e = H_{i-1,4}$
   $b = H_{i-1,1}$ $f = H_{i-1,5}$
   $c = H_{i-1,2}$ $g = H_{i-1,6}$
   $d = H_{i-1,3}$ $h = H_{i-1,7}$
3. Perform the main hash computation
   for $t = 0$ **to** 79

$$\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\sum\nolimits_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}$$

4. Compute the intermediate hash value
   $H_{i,0} = a + H_{i-1,0}$ $H_{i,4} = e + H_{i-1,4}$
   $H_{i,1} = b + H_{i-1,1}$ $H_{i,5} = f + H_{i-1,5}$
   $H_{i,2} = c + H_{i-1,2}$ $H_{i,6} = g + H_{i-1,6}$
   $H_{i,3} = d + H_{i-1,3}$ $H_{i,7} = h + H_{i-1,7}$

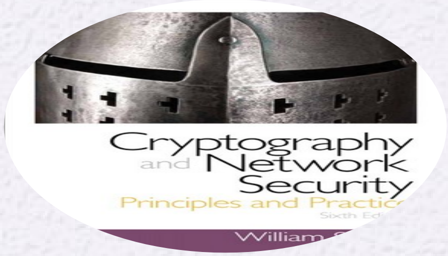
return $\{H_{N,0} \,\|\, H_{N,1} \,\|\, H_{N,2} \,\|\, H_{N,3} \,\|\, H_{N,4} \,\|\, H_{N,5} \,\|\, H_{N,6} \,\|\, H_{N,7}\}$

(Figure can be found on page 345 in textbook)

Figure 11.13 SHA-512 Logic

Summary

- Applications of cryptographic hash functions
  - Message authentication
  - Digital signatures
  - Other applications
- Requirements and security
  - Security requirements for cryptographic hash functions
  - Brute-force attacks
  - Cryptanalysis



- Hash functions based on cipher block chaining
- Secure hash algorithm (SHA)
  - SHA-512 logic
  - SHA-512 round function