8 Hash Functions

Ch11 in textbook

Yanwei Yu

E-mail: ywyu@ustc.edu.cn

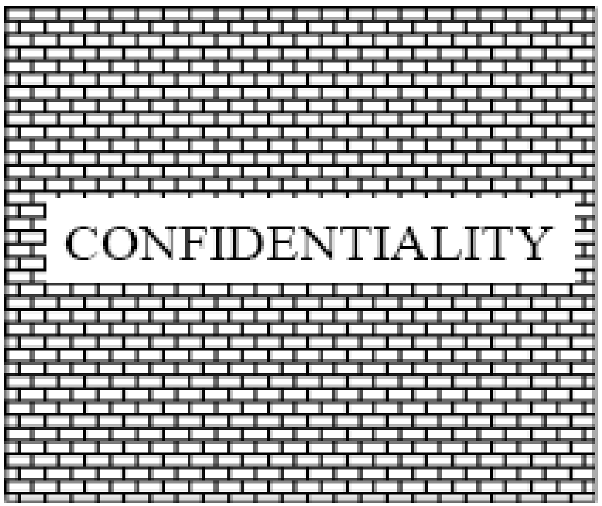
Key Points of Lecture

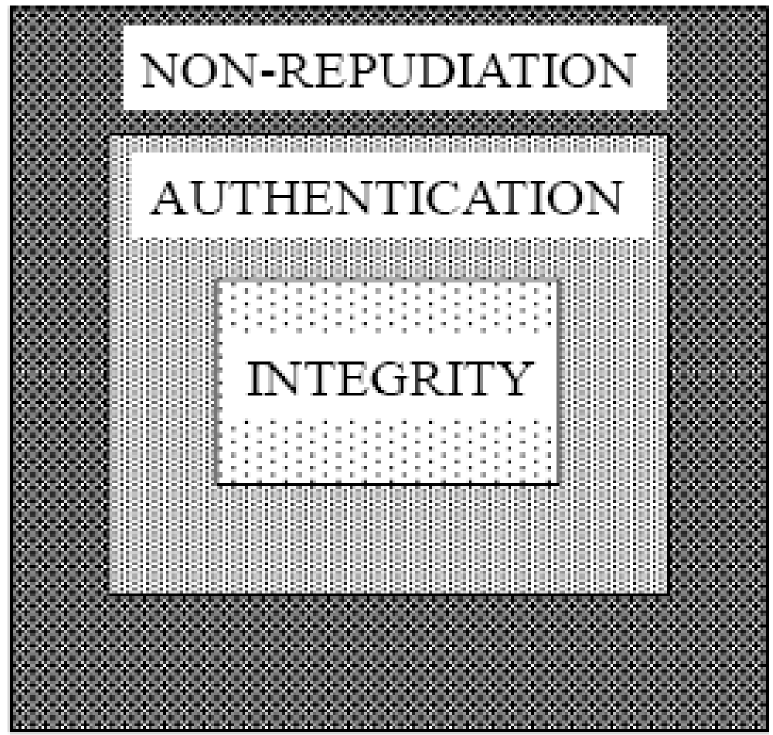
- **Two security services:**
  - **Authentication**
  - **Data Integrity**
- **Two security mechanism:** Irreversible encipherment mechanisms
  - **Hash function**
  - **Message Authentication Codes (MAC)**
    - **Based on cipher or hash function**

Security Service & Attack

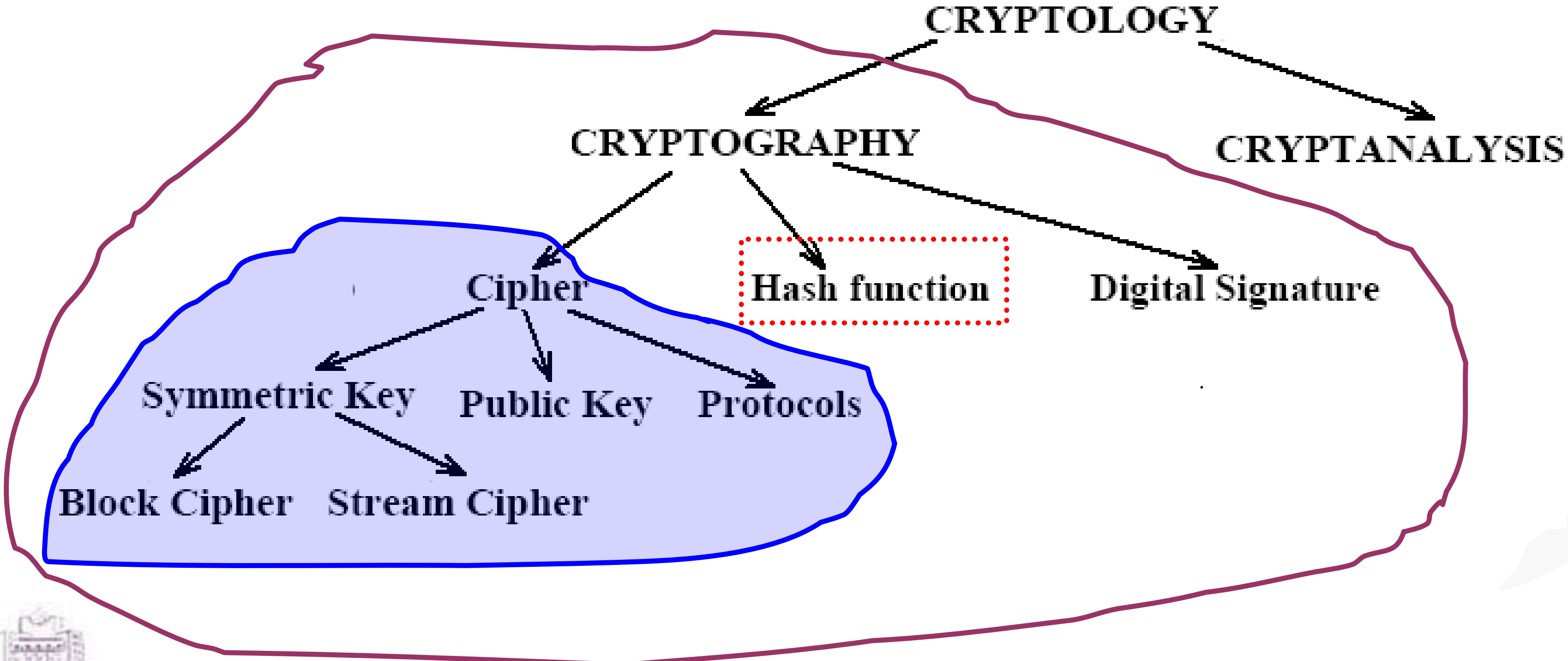
Security Service	Security Attack
• Data Confidentiality	Disclosure（泄密） Traffic analysis
• Authentication • Data Integrity	Masquerade（伪装） Content modification Sequence modification Timing modification
• Non-repudiation	Source repudiation（否认）
	Destination repudiation

Relations among Security Services





Basic security mechanism



Hash Function

- **Hash function h(m)**

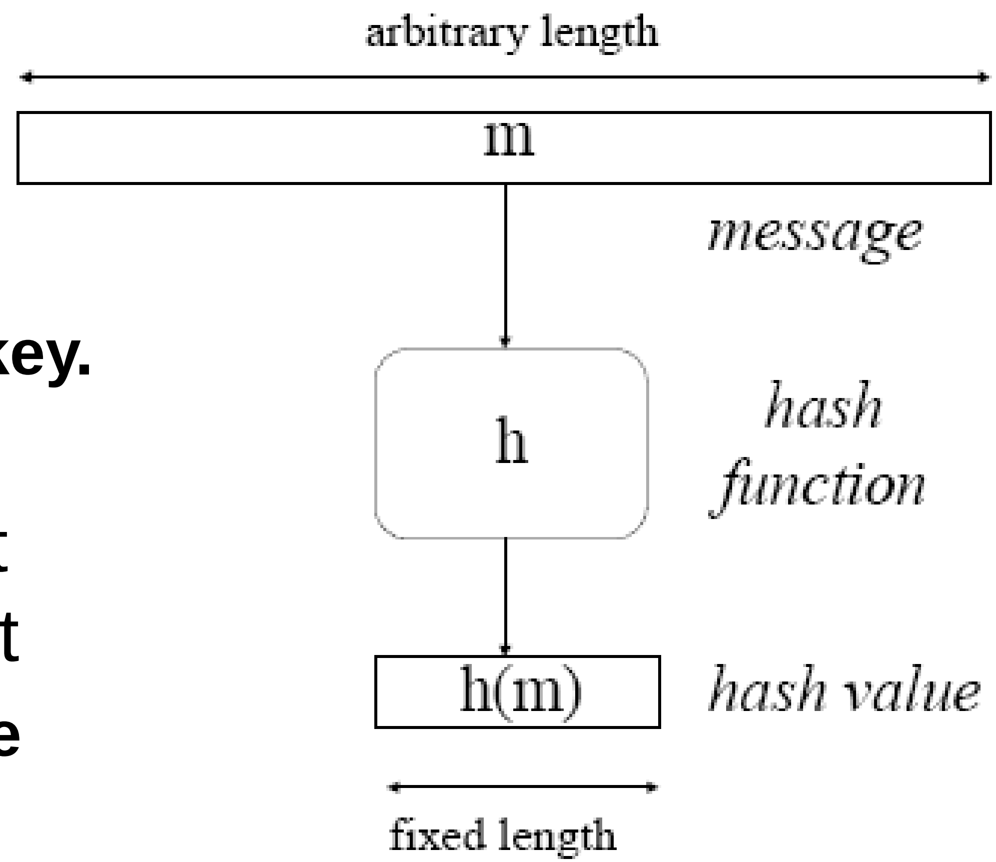
  ***Basic* Requirements**

1) Public description, no key.

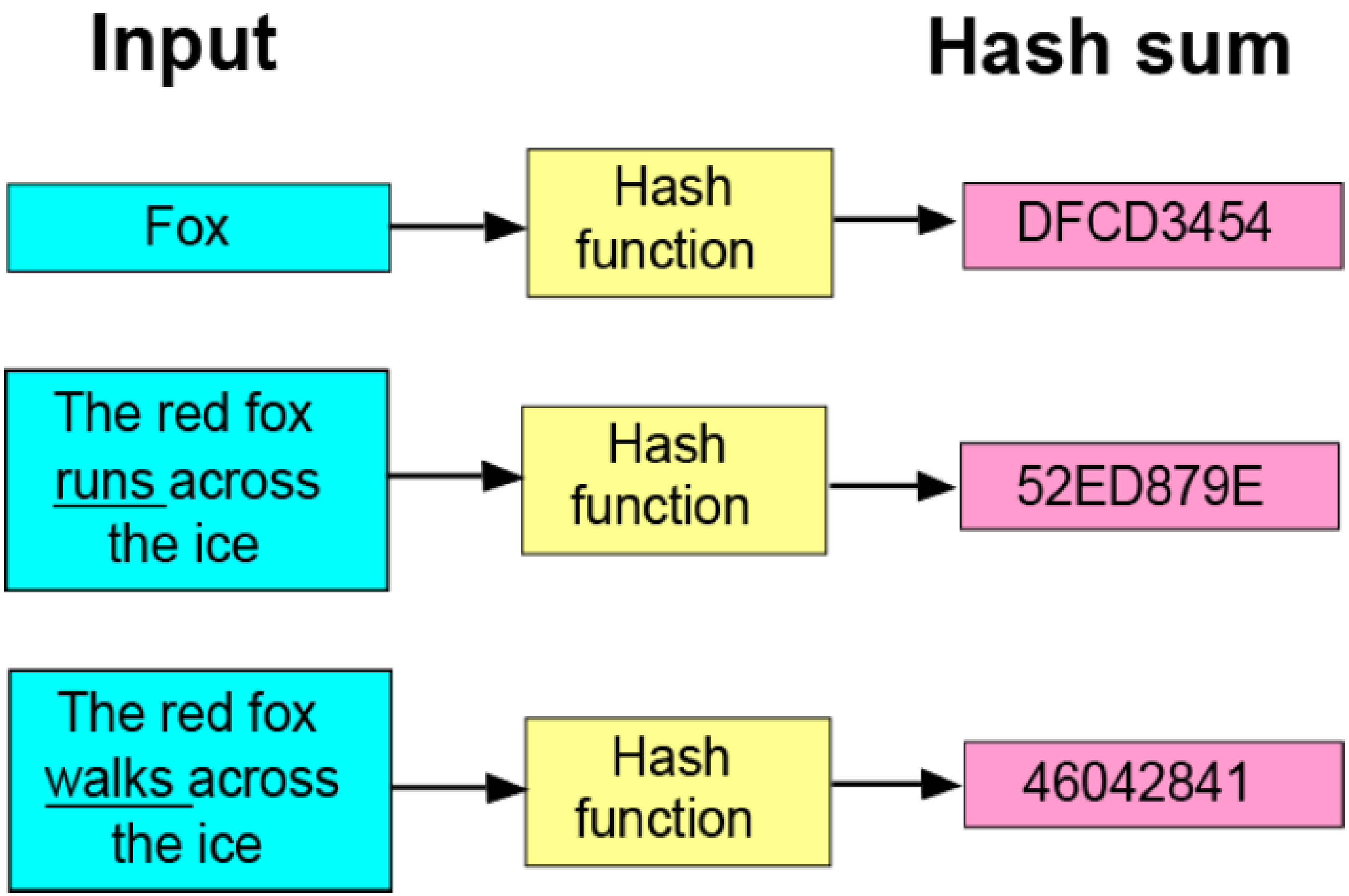
2) Compression

– arbitrary length input
  → fixed length output

4) *h(m)* is easy to compute (hw and sw).



Hash example



Synonym(同义词) about Hash

- **hash function**
  - message digest
- **hash value**
  - hash total
  - fingerprint
  - imprint(印记)
  - cryptographic checksum
  - compressed encoding
  - MDC(Message Digest Code / Modify Detection Code)
  - message digest

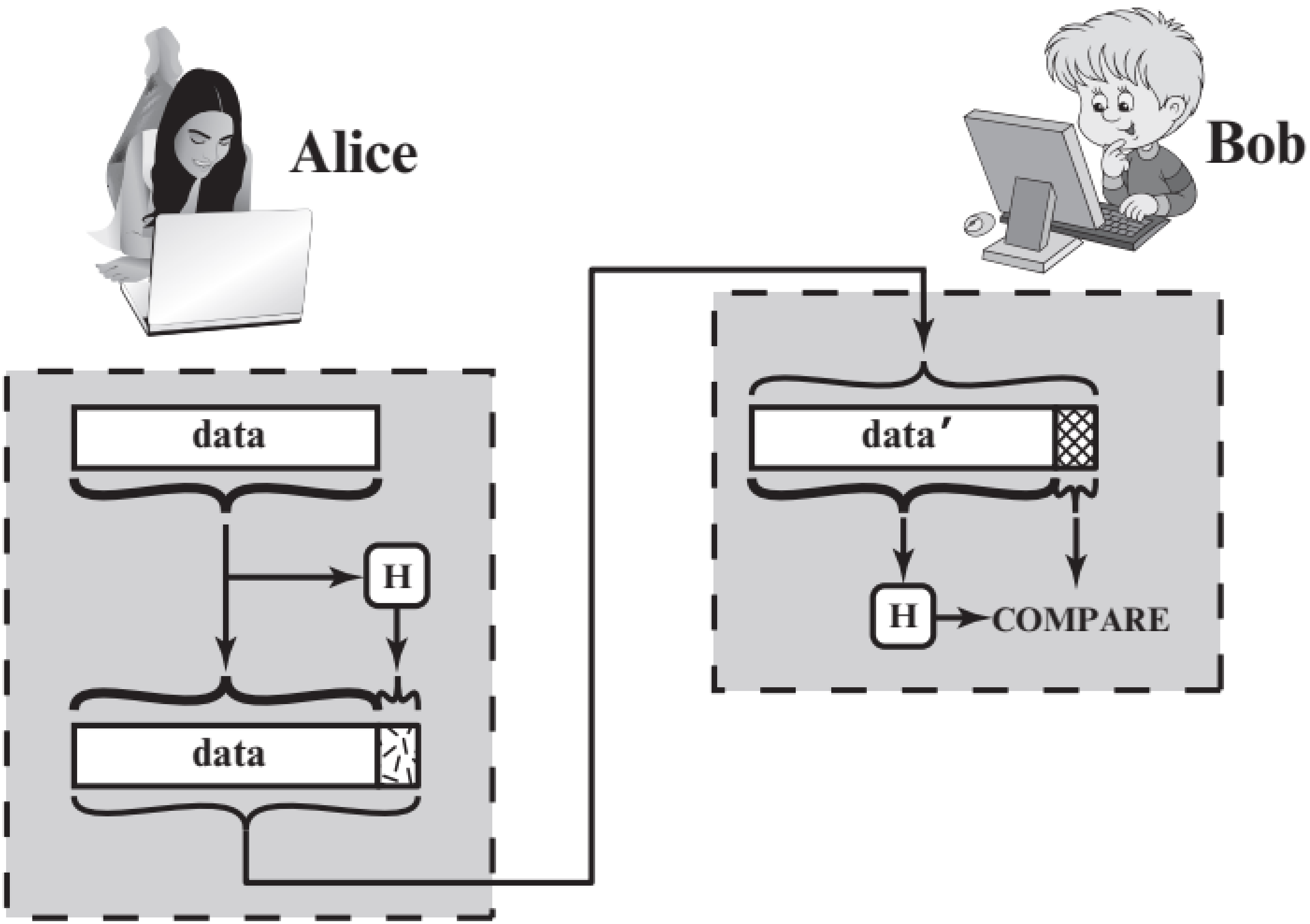
Outline

- **Applications**
- **Two simple hash functions**
- **Requirements and Security**
- **Hash function Construction**
  - **Hash functions based on cipher block chaining**
  - **Secure hash algorithm (SHA)**

Outline

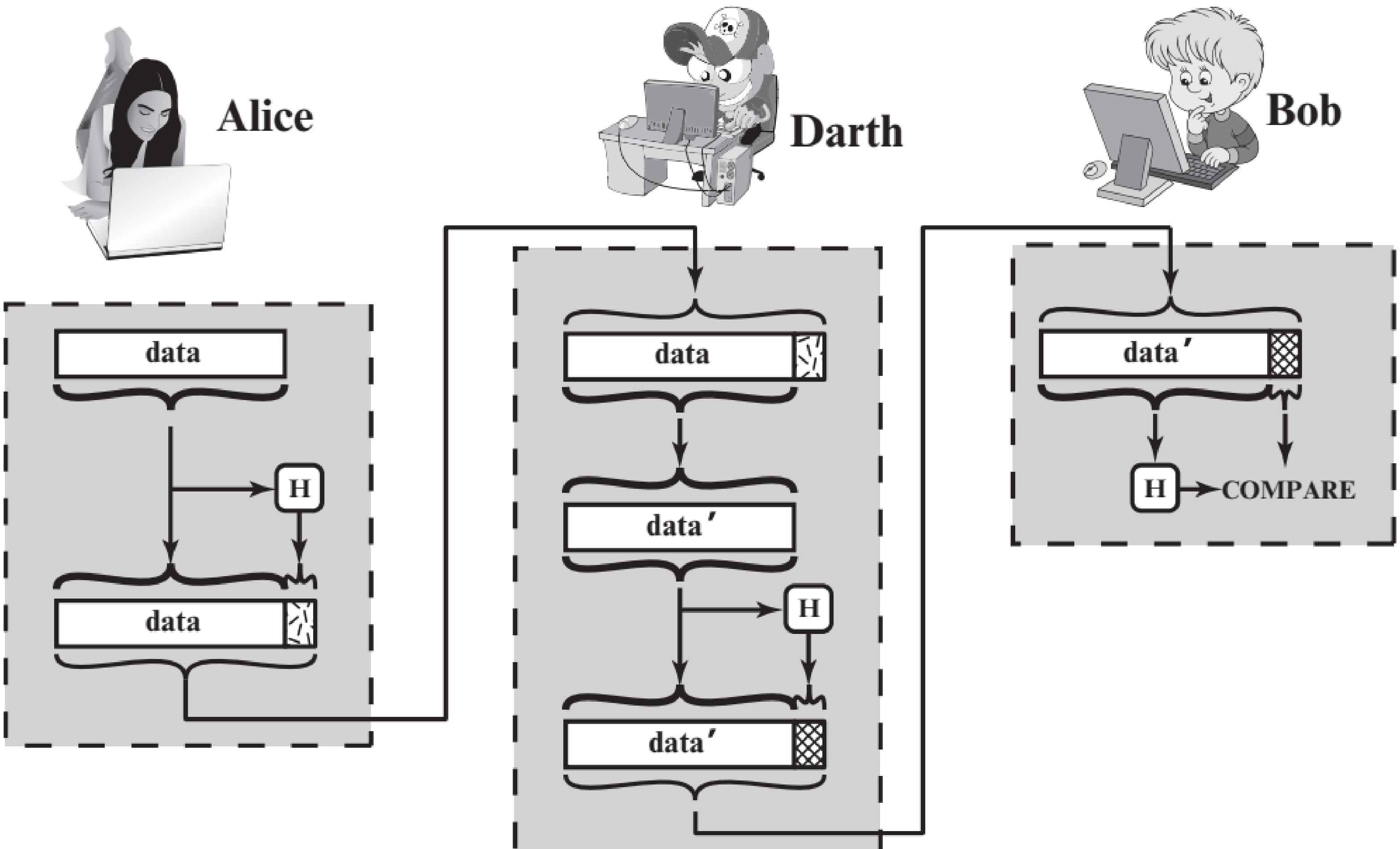
- Applications
- Two simple hash functions
- Requirements and Security
- Hash function Construction
  - Hash functions based on cipher block chaining
  - Secure hash algorithm (SHA)

Hash Functions Applications(1)



(a) Use of hash function to check data integrity

Hash Functions Applications(1)



(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

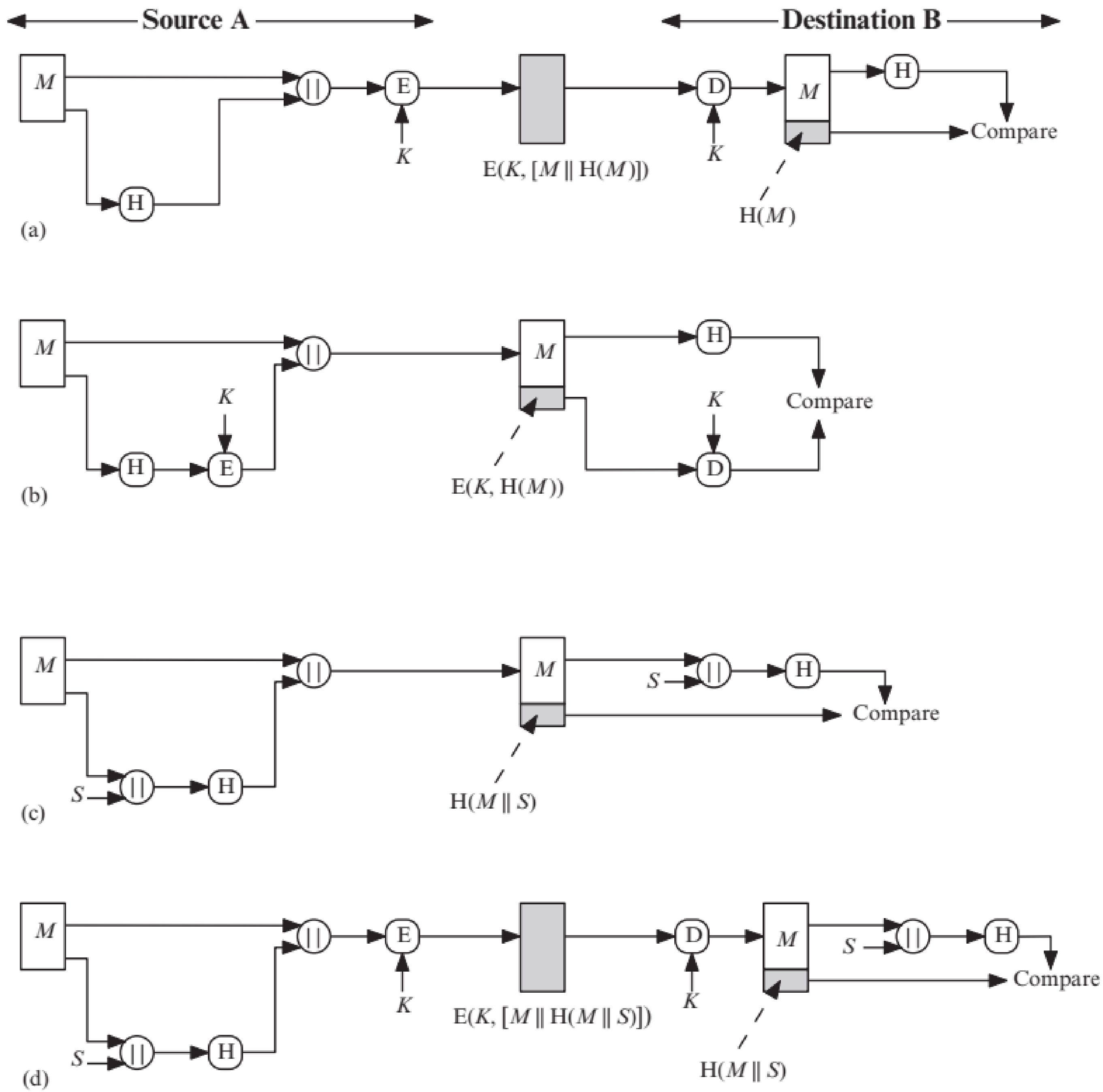


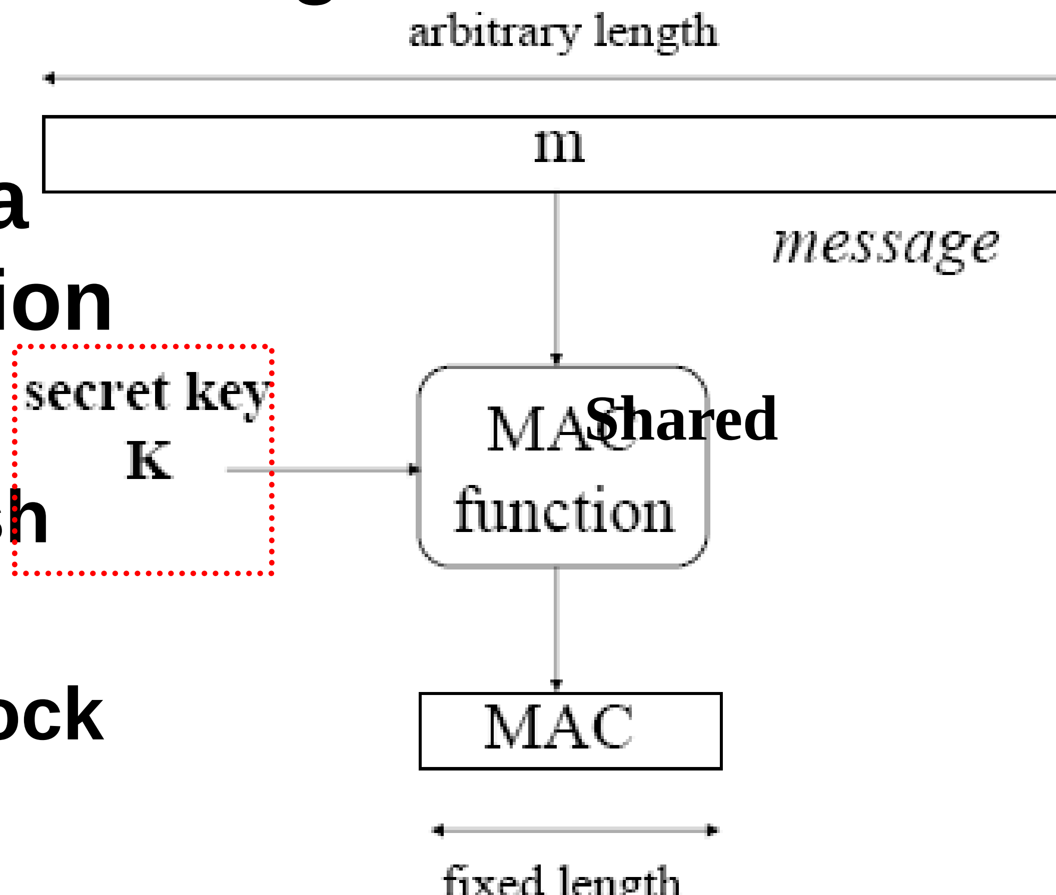
Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

Choice for Only Authentication

- **When confidentiality is not required, encryption to the entire message should be avoid:**
  - **Encryption software is relatively slow**
  - **Encryption hardware costs are not negligible**
  - **Encryption hardware is optimized toward large data sizes**
  - **Encryption algorithms may be covered by patents**

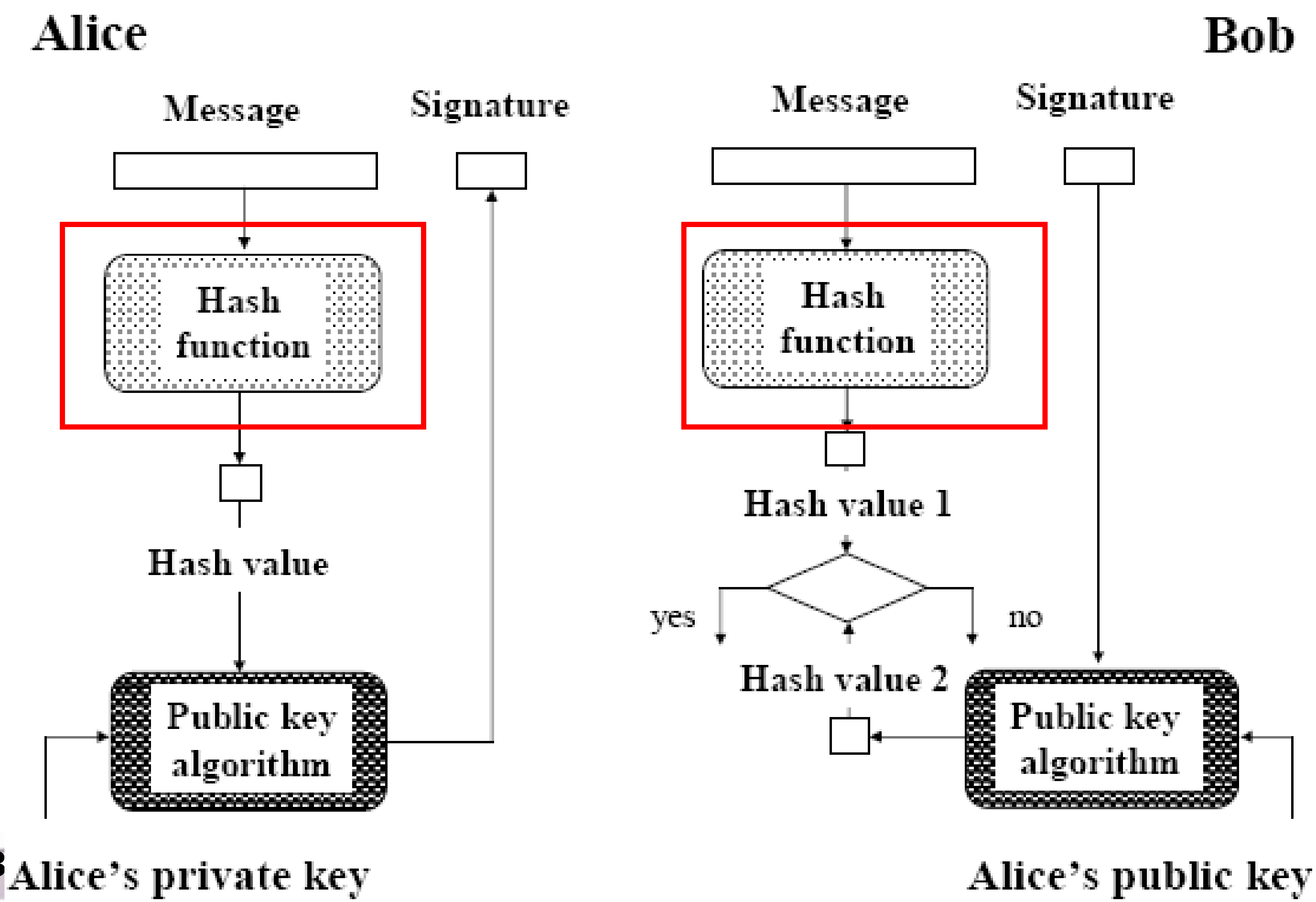
More commonly, message authentication is achieved using message authentication code (MAC)

- **MAC： known as a keyed hash function MAC=C(k,m)**
  - **HMAC: keyed hash functions**
  - **CMAC: Cipher Block Chaining MAC**



Hash Functions Applications(2)

- **Digital Signatures**



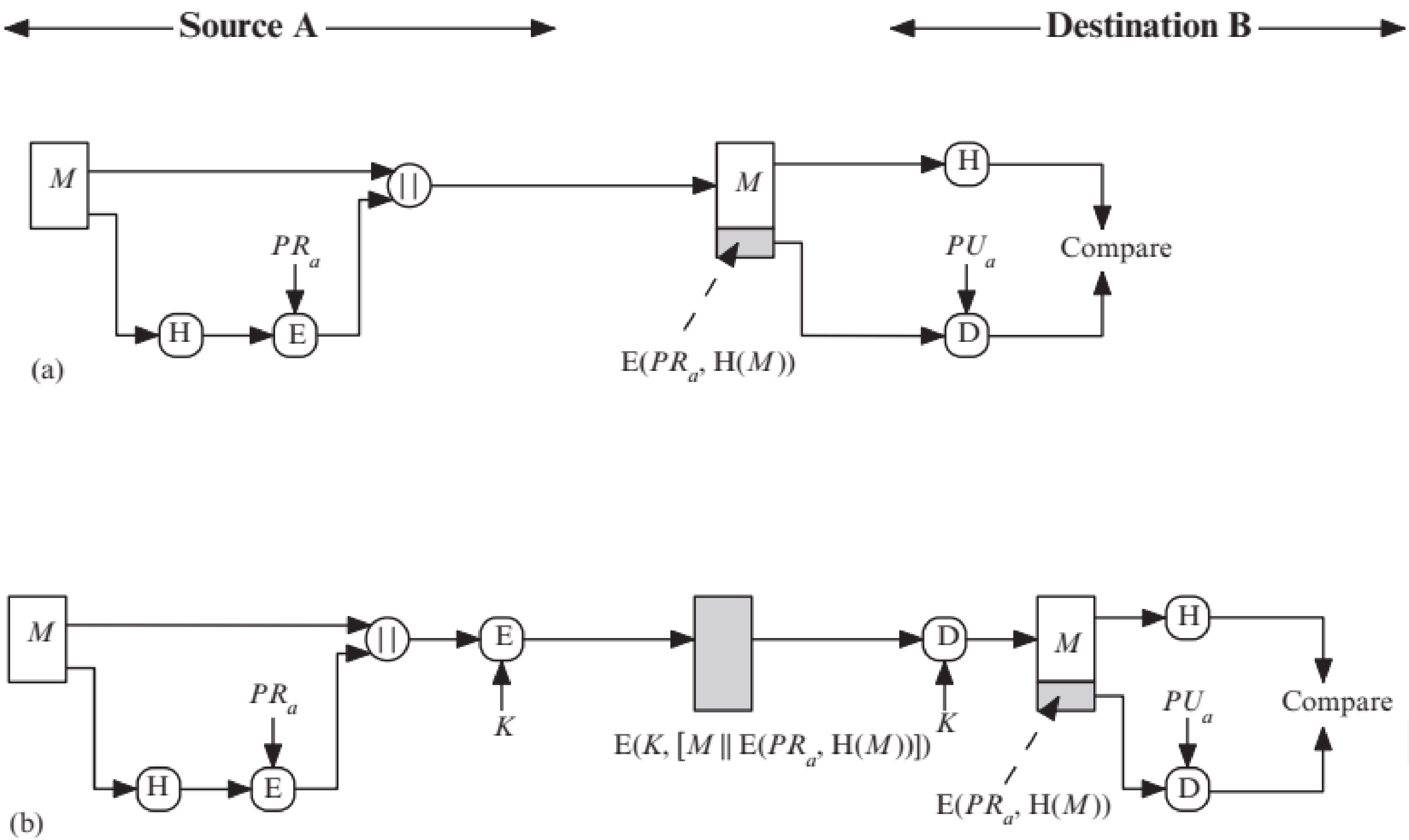
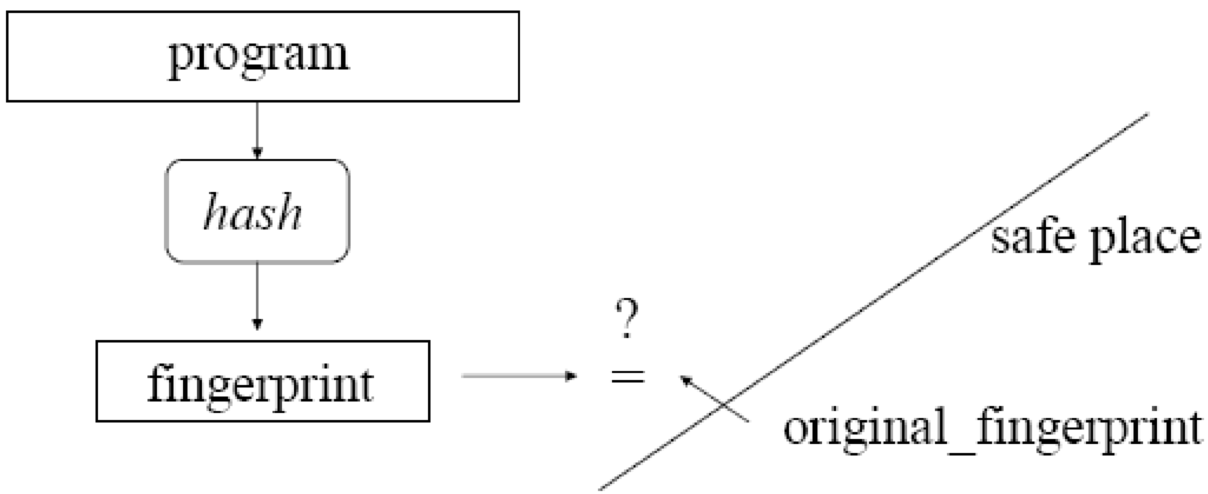


Figure 11.4 Simplified Examples of Digital Signatures

Hash Functions Applications(3)

- **Fingerprint of a program or a document**

(e.g., to detect a modification by a virus or an intruder)



UltraEdit-32 V16.20.0.1011 汉化版(30天免费使用) | 纠正官方中文多

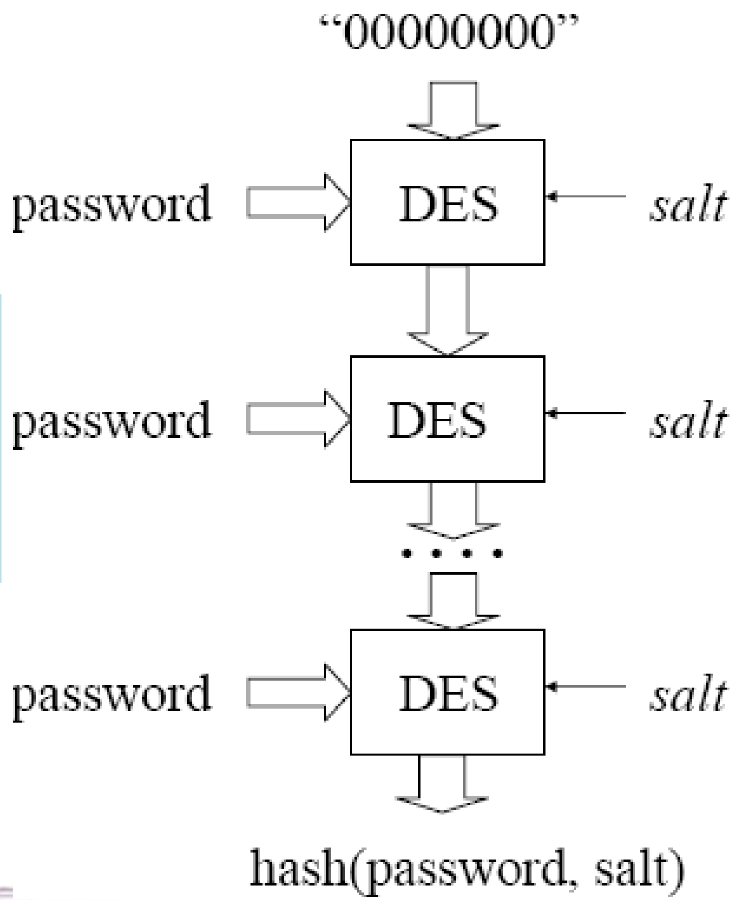
金山检测 瑞星检测 卡巴检测 NOD检测 360检测

无插件 Windows7兼容 多特认证 100%纯净软件

软件大小：13.16MB	下载次数：744714
软件授权： 共享软件(可试用,有时间限制)	软件语言：简体中文
开 发 商：Home Page	评价等级：4
更新时间： 2010-09-25 18:02:03	软件MD5：点击复制
应用平台： Win7/Vista/Win2003/WinXP/Win2000/Win9X/	

Hash Functions Applications(4)

- **Storing password: (ID, hash(password))**



UNIX password scheme

ID, *salt*,
hash(password, salt)

salt modifies the expansion function E of DES

Outline

- Applications
- Two simple hash functions
- Requirements and Security
- Hash function Construction
  - Hash functions based on cipher block chaining
  - Secure hash algorithm (SHA)

Example: Simple Hash Functions ------using XOR

- **Case 1:**
  - **$b \rightarrow C$, where $C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$**

    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

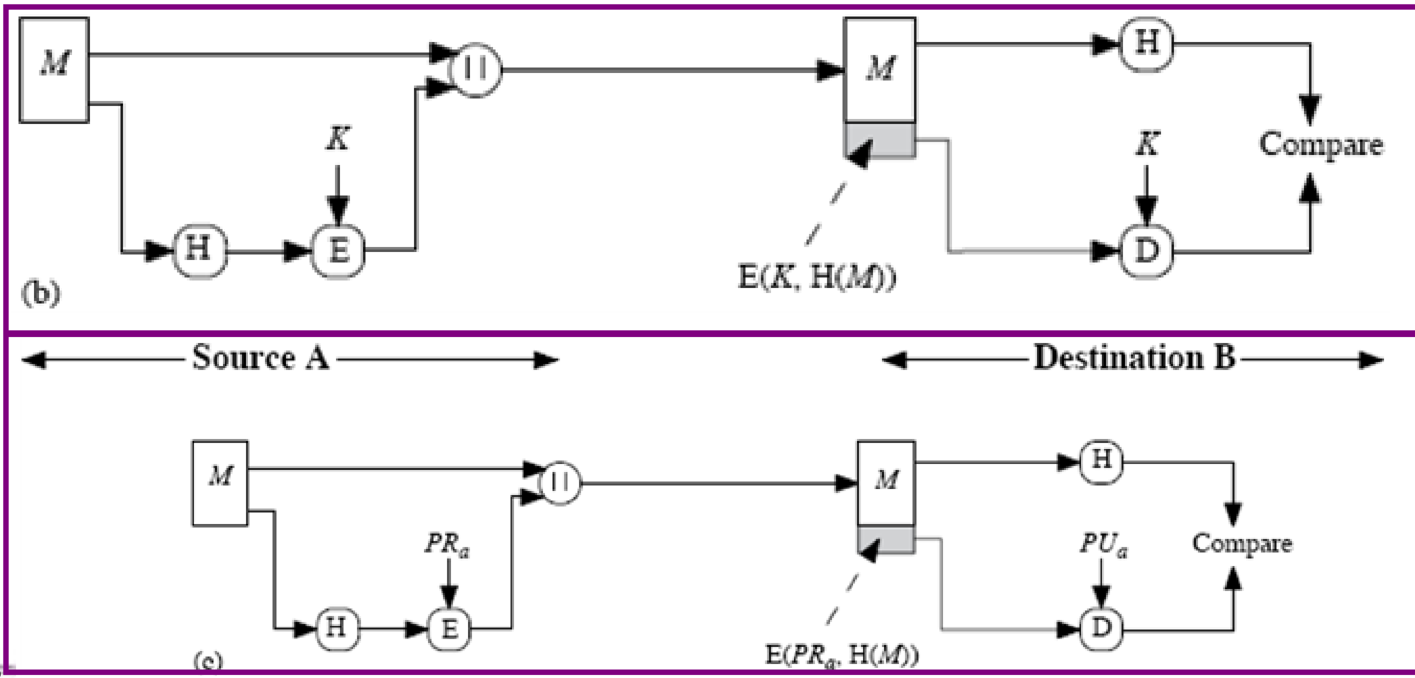
    $\oplus$ = XOR operation
  - **produces a simple parity for each bit position as a longitudinal(经度) redundancy check which is effective for random data as a data integrity check.**
  - **easy to affected by data format**

Example: Simple Hash Functions ------using XOR

- **Case 2: (improvement based on case 1)**
  - Process each successive n-bit block of data as follows:
    - Rotate the current hash value to the left by one bit.
    - XOR the block into the hash value.
    - E.g. if $m= m_1||m_2$, then $H(m_1||m_2||pad(m_2))=LR_1(H(m_1))\oplus(m_2||pad(m_2))$
  - **Easy to forge by appending a block**
    - **Assume m and H(m) is known, easy to find x s.t. H(m||pad(m)||x)=H(m).**
    - **$H(m||pad(m)||\ x)=LR_1(H(m))\oplus x = H(m)$ => $x= LR_1(H(m))\oplus H(m)$.**

1st improvement based on case 2 ——Append encrypted hash code

- **Case3:using encrypted Hash code (improvement based on case 2)**
  - **where hash code is generated according method in case 2.**
  - **Easy to forge by appending a block, the same as Case 2.**
    - **easy to find x s.t. H(m||pad(m)||x)=H(m)**
    - **if H(m||pad(m)||x)=H(m), then E(H(m))=E(H(m||pad(m)||x))**



2^{nd} improvement based on case 2 ——Authenticate then Encrypt

Case 4: transit $Y=E(K, [M \| H(M)])=Y_1\|Y_2\|Y_3\|\ldots$

1) where hash code is generated according method in case 2.

For a message M of 64-bit blocks $X_1, X_2,\ldots, X_N$, appending hash value $X_1 \oplus X_2 \oplus \ldots \oplus X_N$ as H(M) (denoted as X_{N+1})

But X_{N+1} is the hash code:

$$X_{N+1} = X_1 \oplus X_2 \oplus \ldots \oplus X_N$$
$$= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \ldots \oplus [Y_{N1} \oplus \ldots \oplus D(K, Y_N)]$$

2) Encrypt $X= X_1X_2\ldots X_{N+1}$ by CBC mode as Y,then

$$\begin{cases} X_1 = IV \oplus D(K, Y_1) \\ X_i = Y_{i1} \oplus D(K, Y_i) \\ X_{N+1} = Y_N \oplus D(K, Y_{N+1}) \end{cases}$$

- **Case 4 Is Not secure**
- **Hash code would not change if the ciphertext blocks were permuted**

$Y'=Y2||Y1||Y3||...$

$Y=Y1||Y2||Y3||...$ $Y'=Y2||Y1||Y3||...$

- **A** --> **B**

$Y=E(K, [M || H(M)])$
$=Y1||Y2||Y3||...||Yn+1$

Compute $D(K,Y') =M'|| Xn+1'$, and Verify $Xn+1'==H(M')$?
If $Xn+1'==H(M')$, B think Y is not changed during transmission.

if $M=X1 ||X2 ||X3||...Xn$,
$H(M)=Xn+1=X1\oplus X2\oplus \ldots \oplus Xn$

- if Attacker can intercept $Y = E(K, [M \| H(M)]) = Y_1 \| Y_2 \| Y_3 \| \ldots$ from A to B and modify Y to $Y' = Y_2 \| Y_1 \| Y_3 \| \ldots$, then send Y' to B.
- B will decrypt and obtain M' and H(M')
  - $X_1' = IV \oplus D(K,Y_2) \neq X_1$
  - $X_2' = Y_2 \oplus D(K,Y_1) \neq X_2$
  - $X_3' = Y_1 \oplus D(K,Y_3) = X_3$
  - $X_4' = Y_3 \oplus D(K,Y_4) = X_4$
  - ……
  - $X_{n+1}' = Y_n \oplus D(K,Y_{n+1}) = X_{n+1}$

Obviously, Xn+1'==H(M'). But M' was modified in fact

- and $X_{n+1} = (IV \oplus D(K,Y_1)) \oplus (Y_1 \oplus D(K,Y_2)) \oplus (Y_2 \oplus D(K,Y_3)) \oplus (Y_3 \oplus D(K,Y_4)) \oplus \ldots \oplus (Y_{n-1} \oplus D(K,Y_n)) = (IV \oplus D(K,Y_2)) \oplus (Y_2 \oplus D(K,Y_1)) \oplus (Y_1 \oplus D(K,Y_3)) \oplus (Y_3 \oplus D(K,Y_4)) \oplus \ldots \oplus (Y_{n-1} \oplus D(K,Y_n)) = X_1' \oplus X_2' \oplus \ldots \oplus X_n'$
- so, $X_{n+1}' == X_1' \oplus X_2' \oplus \ldots \oplus X_n'$, B verify M' is not modified during transimission. But M' was modified in fact.

$$X_1 = IV \oplus D(K, Y_1)$$
$$X_i = Y_{i1} \oplus D(K, Y_i)$$
$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

Outline

- Applications
- Two simple hash functions
- Requirements and Security
- Hash function Construction
  - Hash functions based on cipher block chaining
  - Secure hash algorithm (SHA)

Requirements for Hash Functions *P349*

1. can be applied to any sized message `M`
2. produces fixed-length output `h`
3. is easy to compute `h=H(M)` for any message `M`

Basic Requirements

4. <u>One-way property</u>: given `h` is infeasible to find `x` s.t.(满足) `H(x)=h`
5. <u>Weak collision resistance</u>: given `x` is infeasible to find `y` s.t. `H(y)=H(x)`
6. <u>Strong collision resistance</u>: is infeasible to find any `x,y` s.t. `H(y)=H(x)`

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) with $x \neq y$, such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Birthday Attack *Appendix 11A P250*

Q: How many students must be in a class so that there is a greater than 50% chance that

1. one of the students shares the teacher's birthday (up to the day and month)?

   366/2=183

2. any two of the students share the same birthday (up to the day and month)?——Birthday paradox(悖论)

   $1.18*(366)^{1/2}\approx 23$

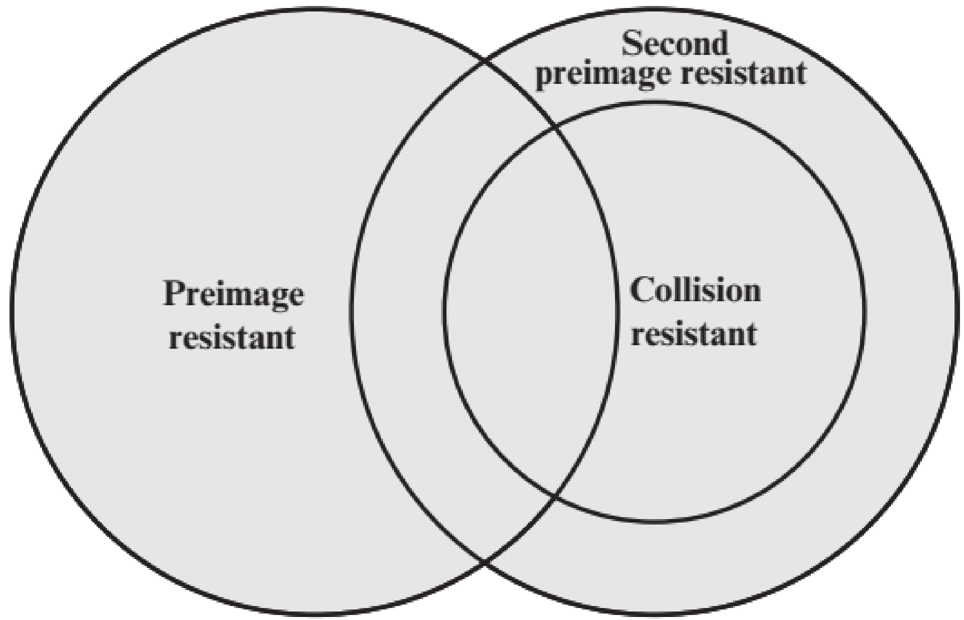


Figure 11.6 Relationship Among Hash Function Properties

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

Table 11.2 shows the resistant properties required for various hash function applications.

General Scheme for Constructing a Secure Hash Function

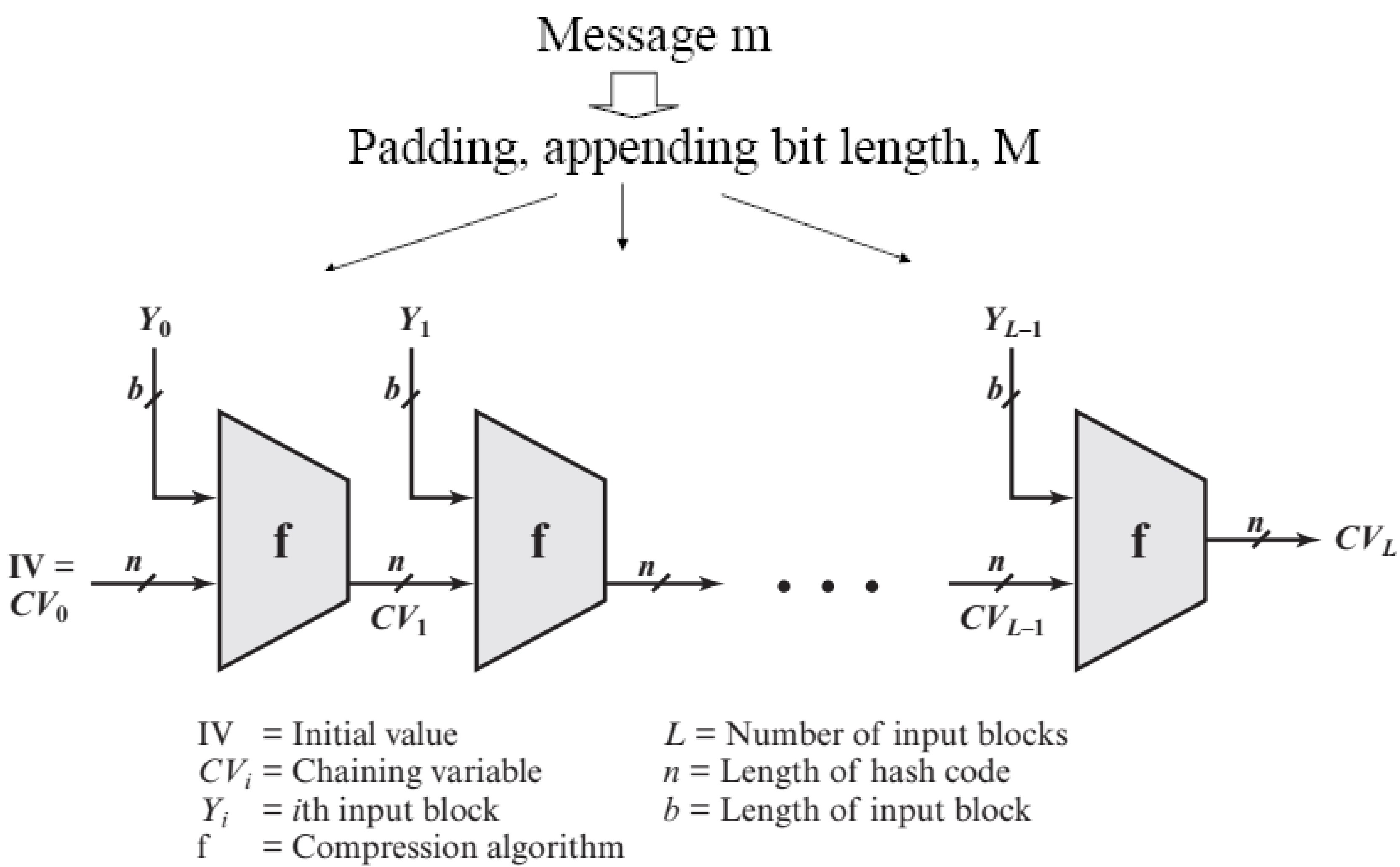


Figure 11.8 General Structure of Secure Hash Code

Outline

- Applications
- Two simple hash functions
- Requirements and Security
- Hash function Construction
  - Hash functions based on cipher block chaining
  - Secure hash algorithm (SHA)

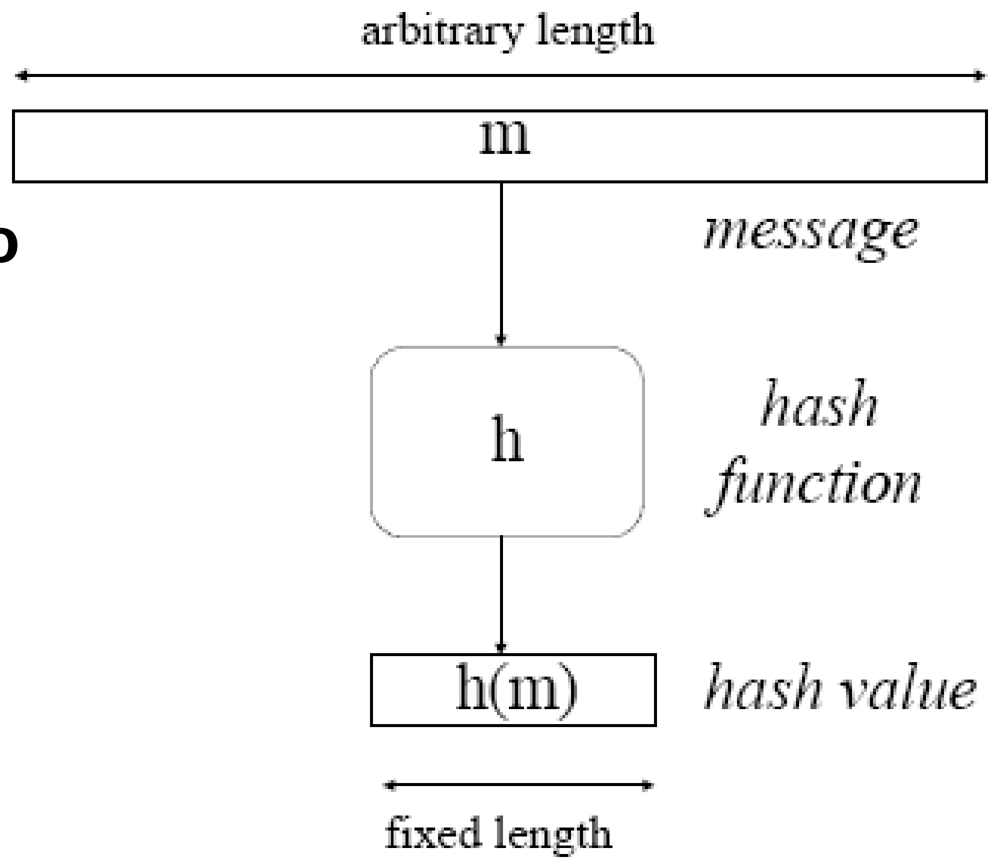
Hash Function

- **Hash function h(m)**
  ***Basic* Requirements**

1) **Public description, no key.**
2) **Compression**

– arbitrary length input → fixed length output

3) ***h(m)* is easy to compute (hw and sw).**



Outline

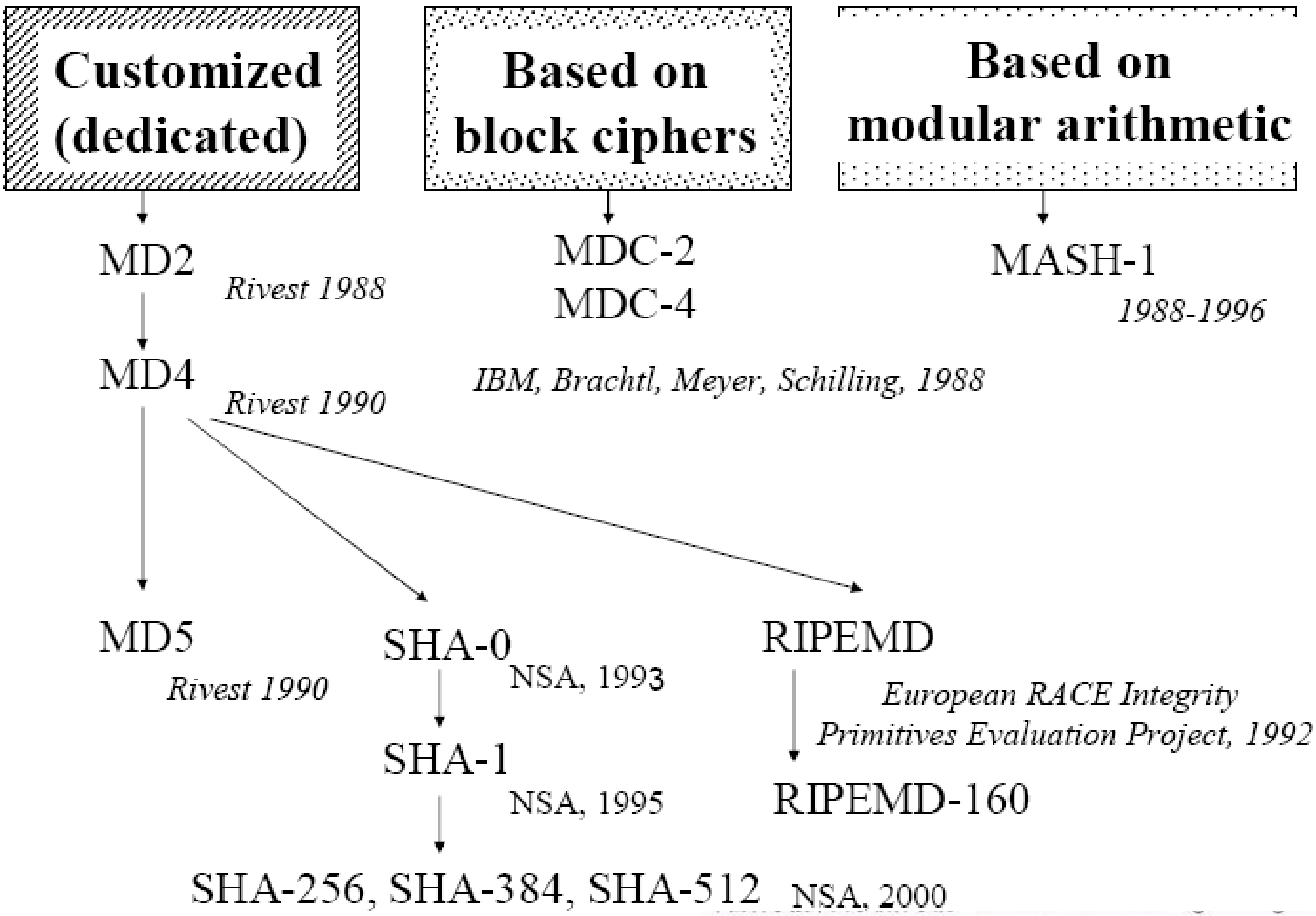
- Applications
- Two simple hash functions
- Requirements and Security
- Hash function Construction
  - Hash functions based on cipher block chaining
  - Secure hash algorithm (SHA)

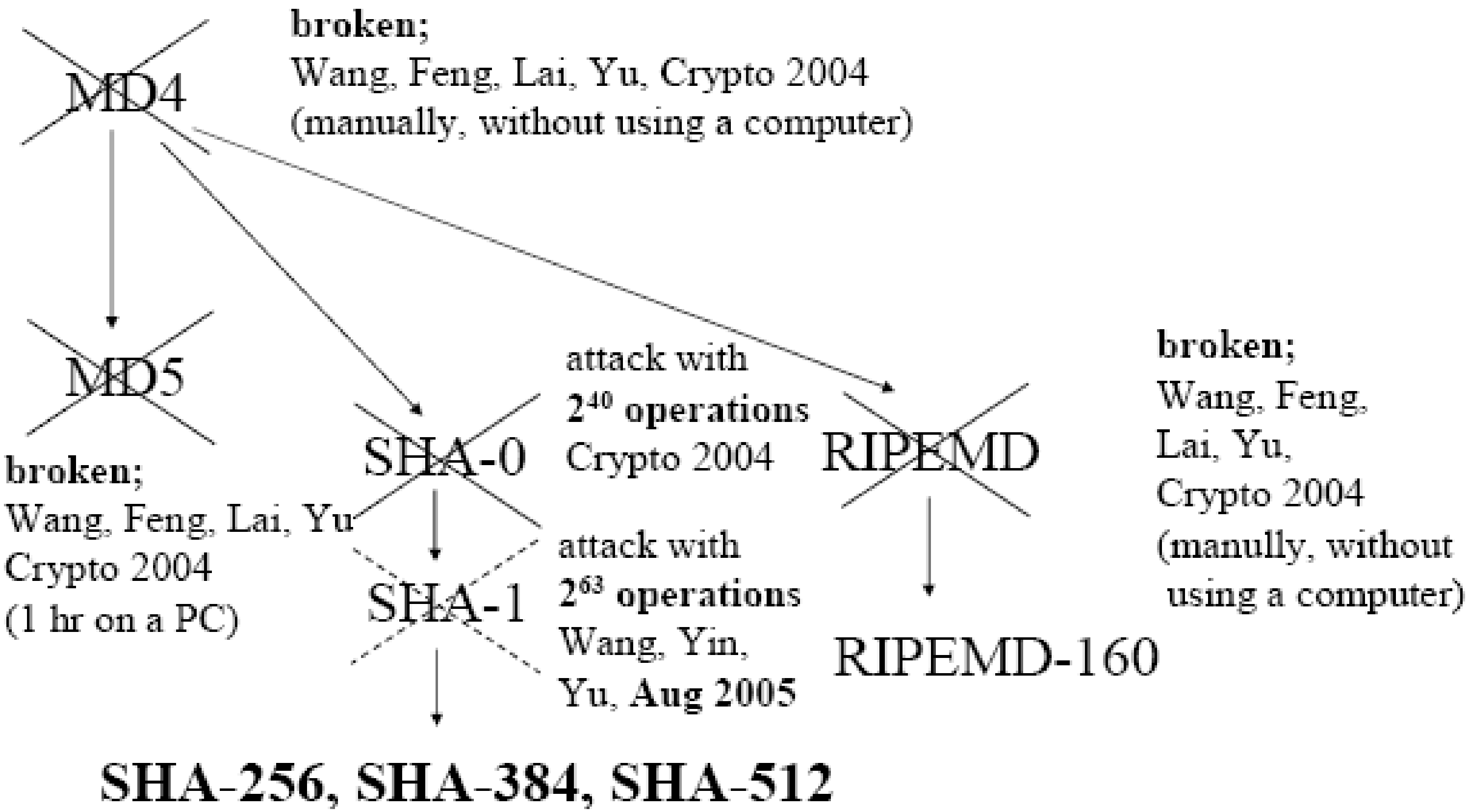
Hash functions based on cipher block chaining

- Divide a message M into fixed-size blocks $M_1, M_2, \ldots, M_N$ and use a symmetric encryption system such as DES to compute the hash code G as

$$
\begin{aligned}
H_0 &= \text{initial value} \\
H_i &= \mathrm{E}(M_i, H_{i-1}) \\
G &= H_N
\end{aligned}
$$

Hash Function Algorithms





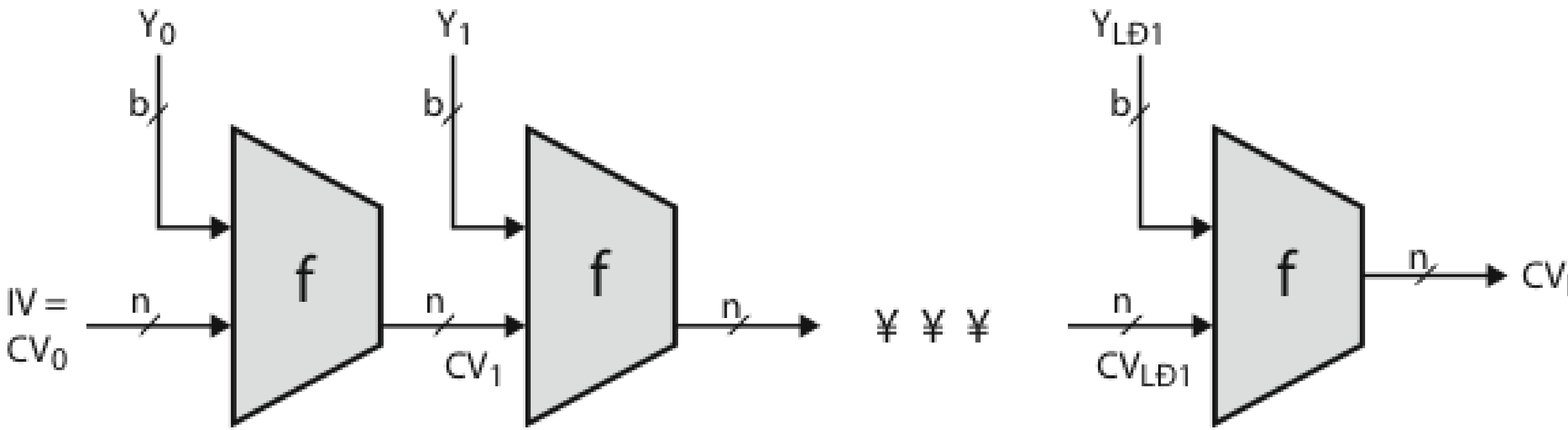
- http://csrc.nist.gov/groups/ST/hash/index.html

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

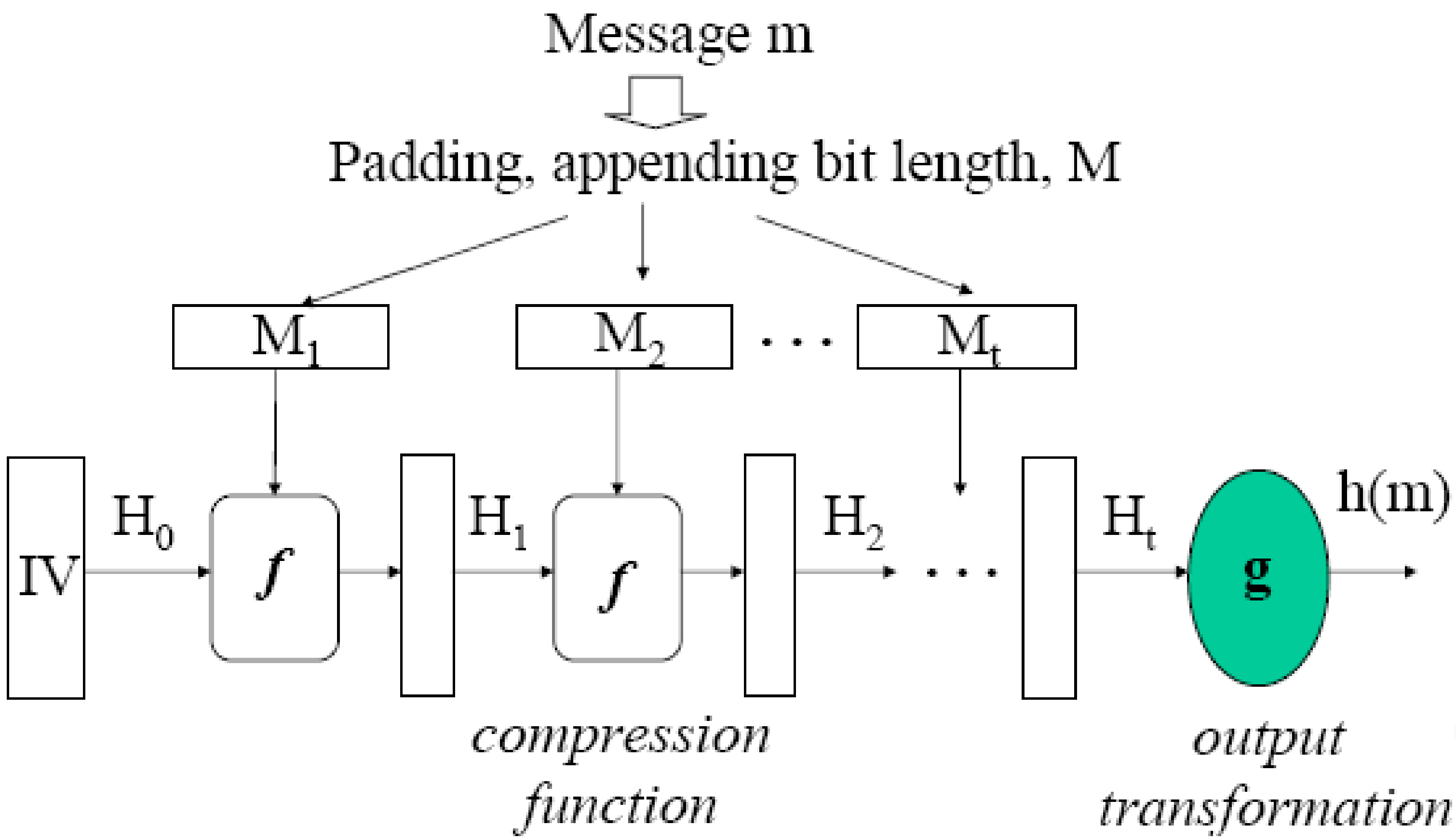
Note: All sizes are measured in bits.

Hash Algorithm Structure



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			

General Scheme for Constructing a Secure Hash Function



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 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- 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

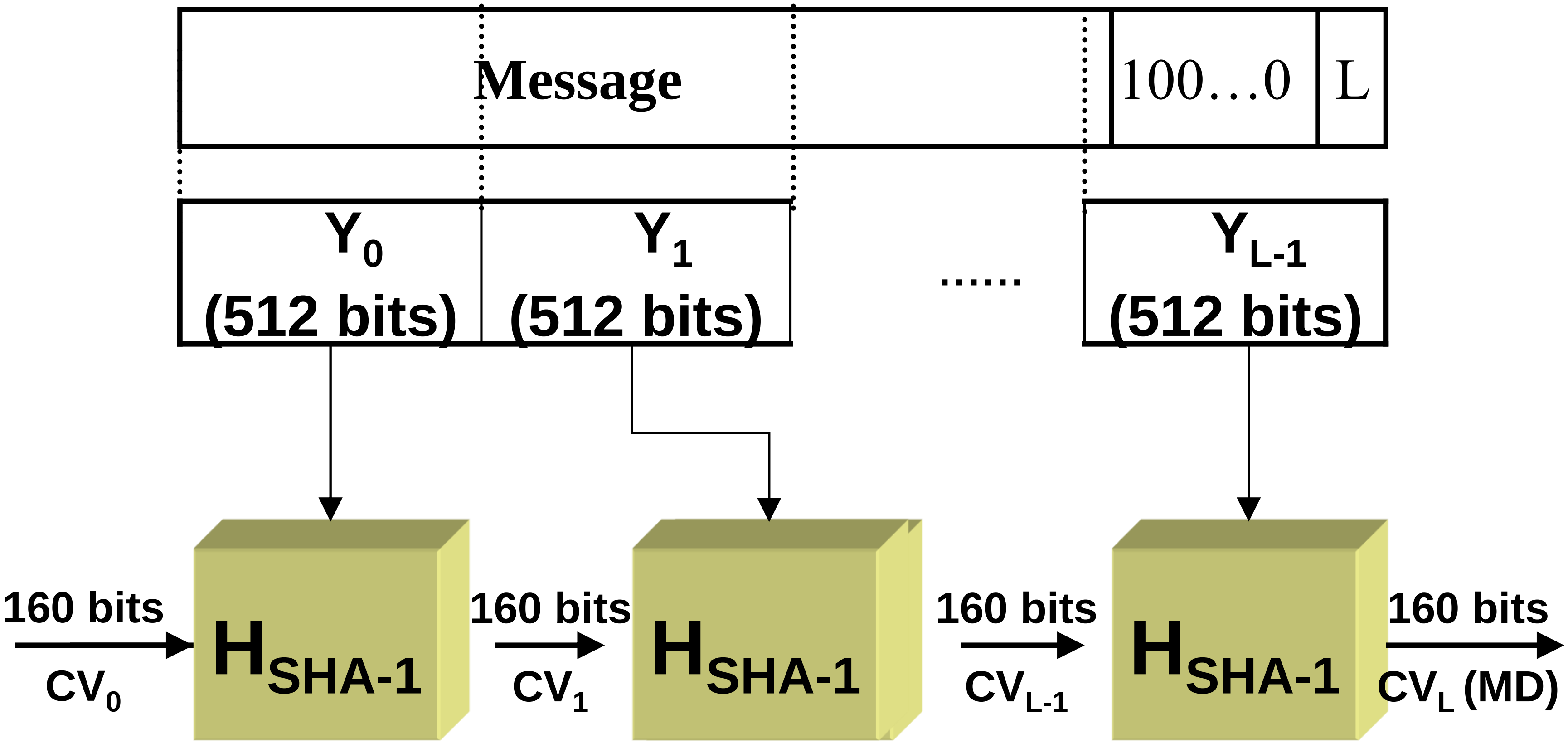
Revised Secure Hash Standard

- 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-1 Overview

- Input： message with length of less than 2^{64} bits
- Output: a 160-bit message digest

- Process unit: input is processed in 512-bit blocks



$Y=Y_0Y_1\ldots Y_{L-1}$ (Y_i has 16 words=512bits)

CV_0 (Initial Value of Buffer(ABCDE)) (5 word)

$CV_{q+1}=H_{SHA-1}(CV_q,Y_q)$

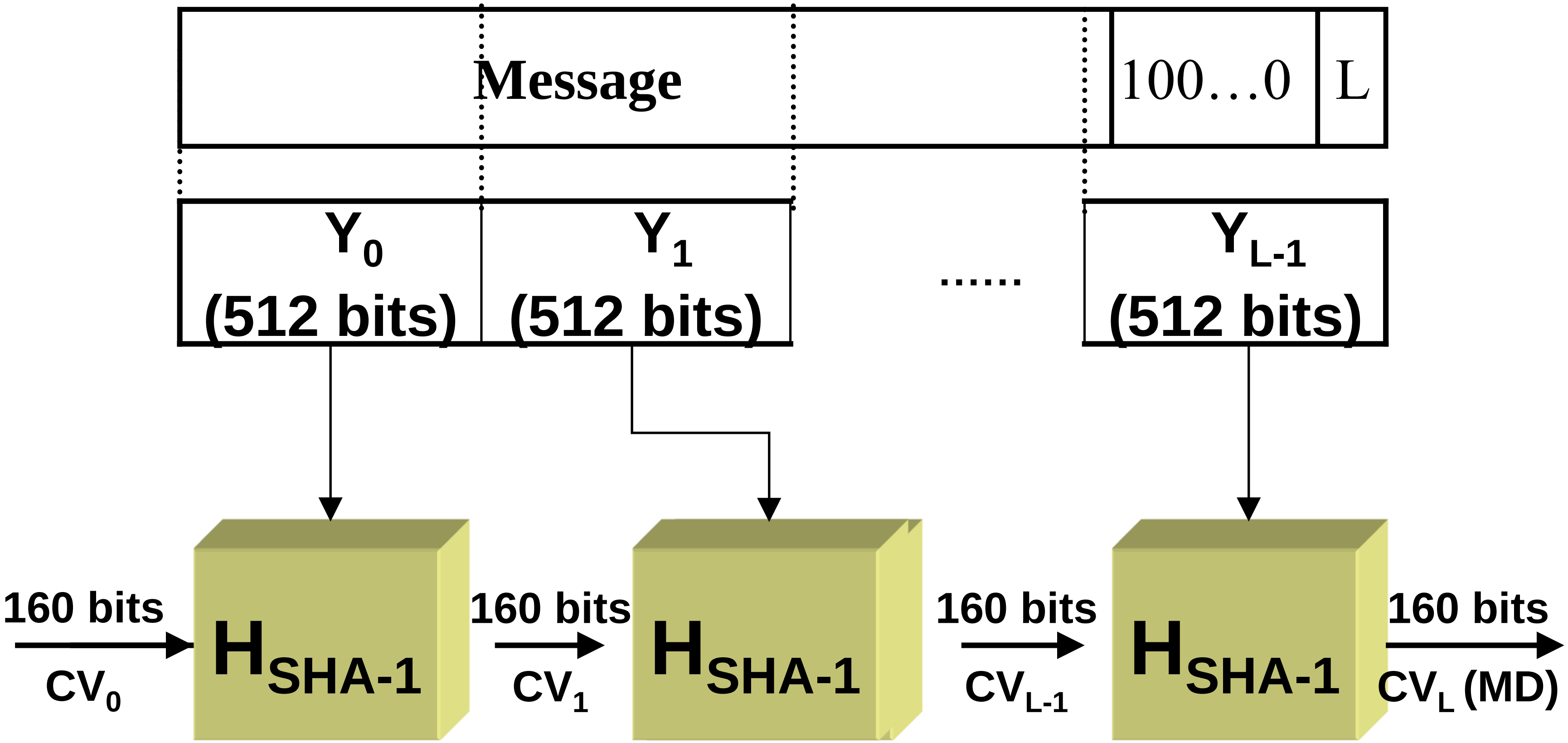
$MD=CV_{q+1}$（Output）

- **Step 1: Append padding bits**
- **Step 2: Append length**
- **Step 3: Initialize hash buffer**

  A=0x67453210，B=0xEFCDAB89

  C=0x98BADCFE，D=0x10325476

  E=0xC3D2E1F0
- **Step 4: Process message in 512-bit (16-word) blocks**



$Y=Y_0Y_1…Y_{L-1}$ (Y_i has 16 words=512bits)

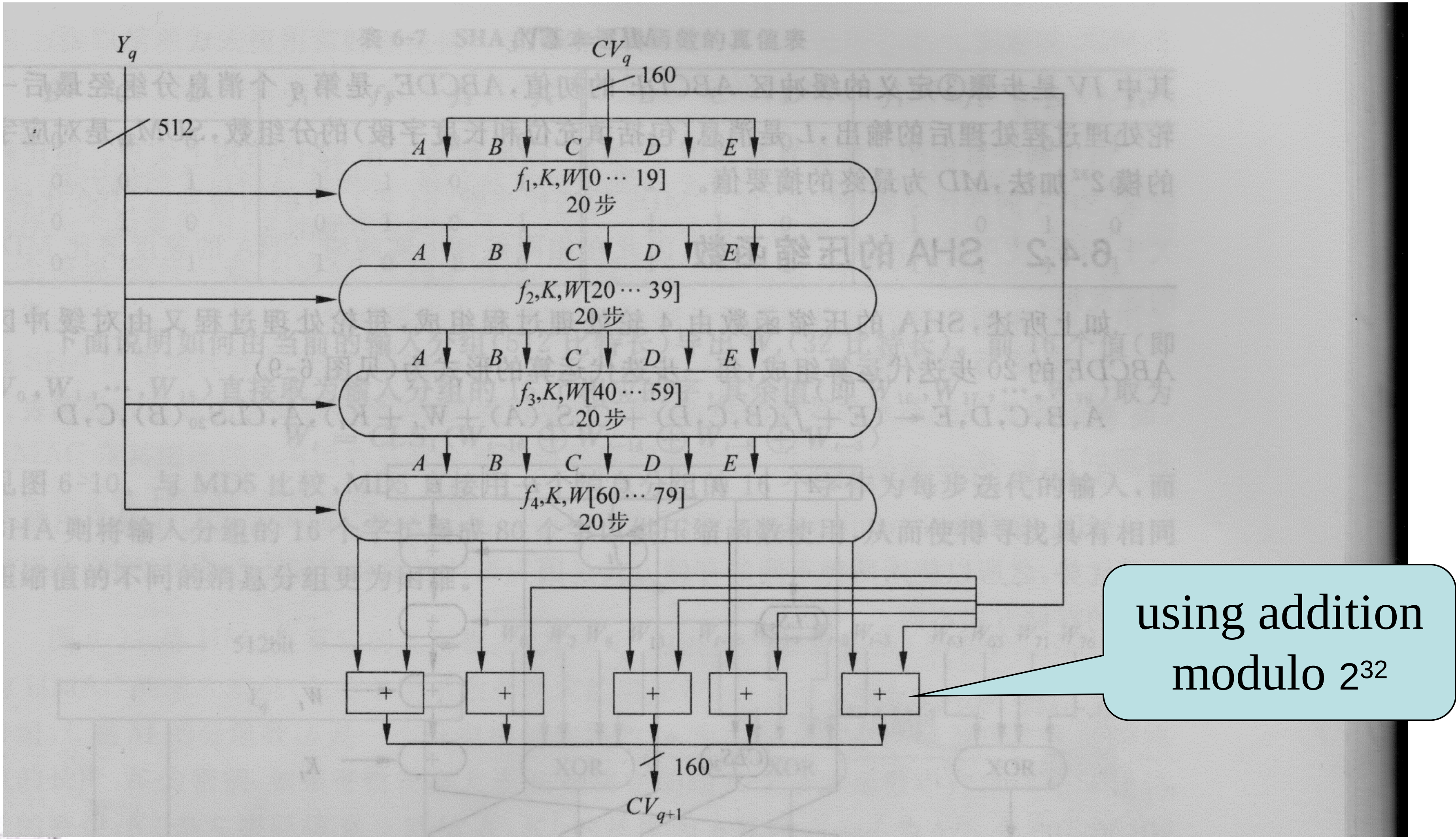
CV_0 (Initial Value of Buffer(ABCDE)) (5 word)

$CV_{q+1}=H_{SHA-1}(CV_q,Y_q)$

$MD=CV_{q+1}$（Output）

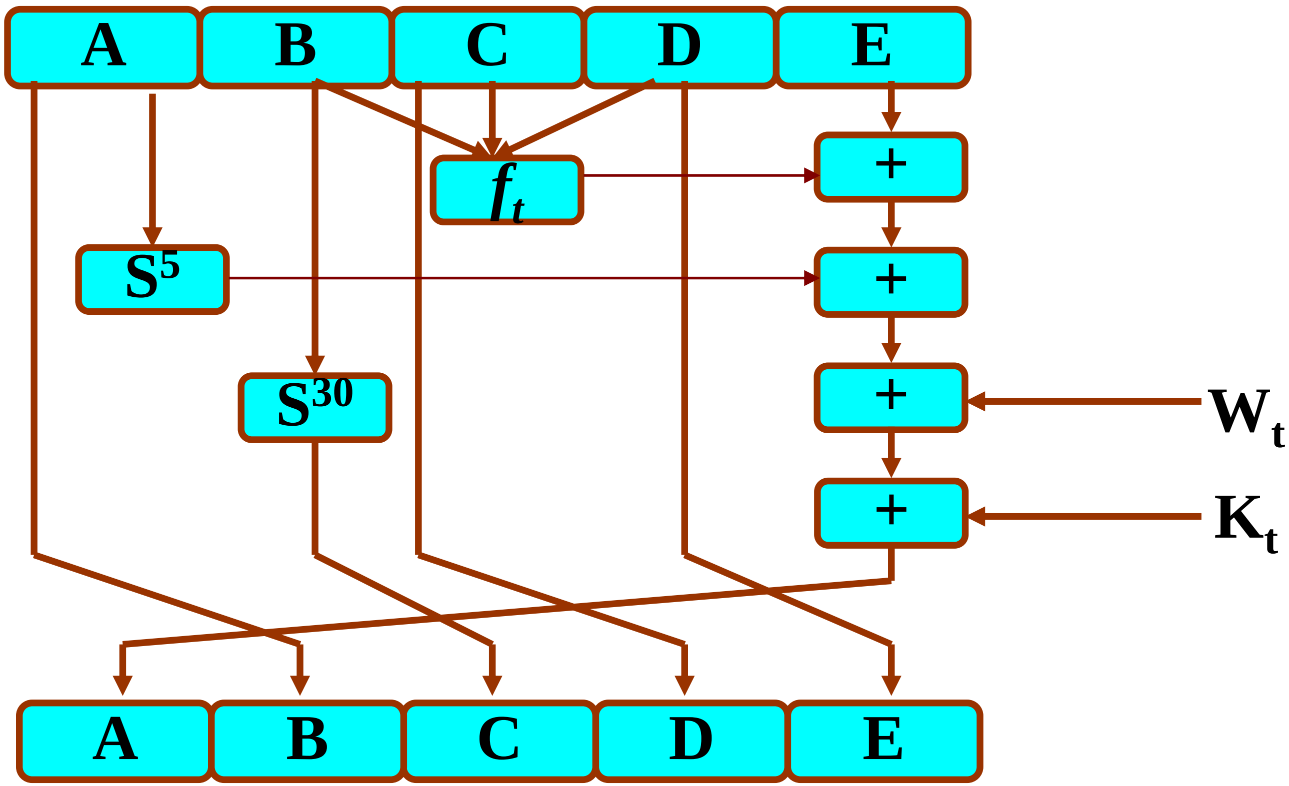
SHA-1 Compression Function

- heart of the algorithm
- processing message in 512-bit blocks
- consists of 80 rounds, for each round t (0≤t≤79):
  - updating a 160-bit buffer
  - using a 160-bit value Wt derived from the current message block
  - has a round constant Kt



$$CV_{q+1}=H_{SHA\text{-}1}(CV_q,Y_q)$$

SHA-1 Round Function



① $B \leftarrow S^{30}(B)$

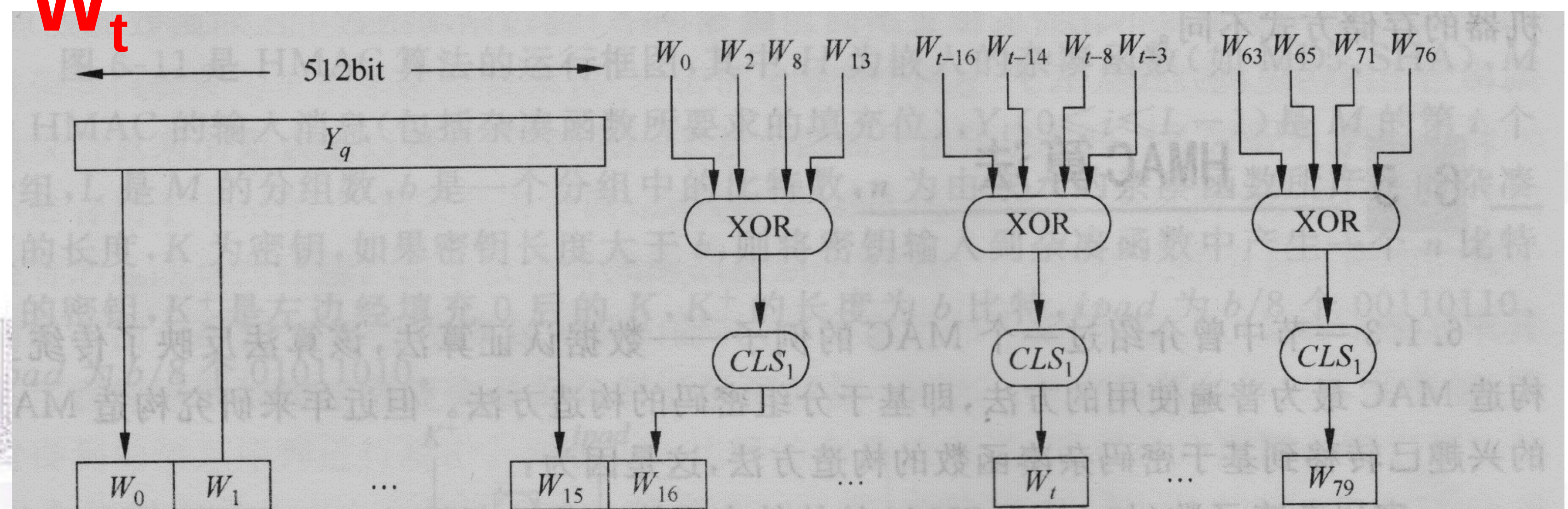
② $E \leftarrow (E+f_t(B,C,D))+S^5(A)+W_t+K_t)$,

③ circular right shift (rotation) of ABCDE by 1 word

f_t

迭代的步数	函数名	定义
$0 \leqslant t \leqslant 19$	$f_1 = f_t(B,C,D)$	$(B \wedge C) \vee (\bar{B} \wedge D)$
$20 \leqslant t \leqslant 39$	$f_2 = f_t(B,C,D)$	$B \oplus C \oplus D$
$40 \leqslant t \leqslant 59$	$f_3 = f_t(B,C,D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \leqslant t \leqslant 79$	$f_4 = f_t(B,C,D)$	$B \oplus C \oplus D$

W_t



$K_t, 0 \le t \le 79$

$$K_t = \begin{cases} \text{0x5A827999} & 0 \le t \le 19 \\ \text{0x6ED9EBA1} & 20 \le t \le 39 \\ \text{0x8F1BBCDC} & 40 \le t \le 59 \\ \text{0xCA62C1D6} & 60 \le t \le 79 \end{cases}$$

SHA-512

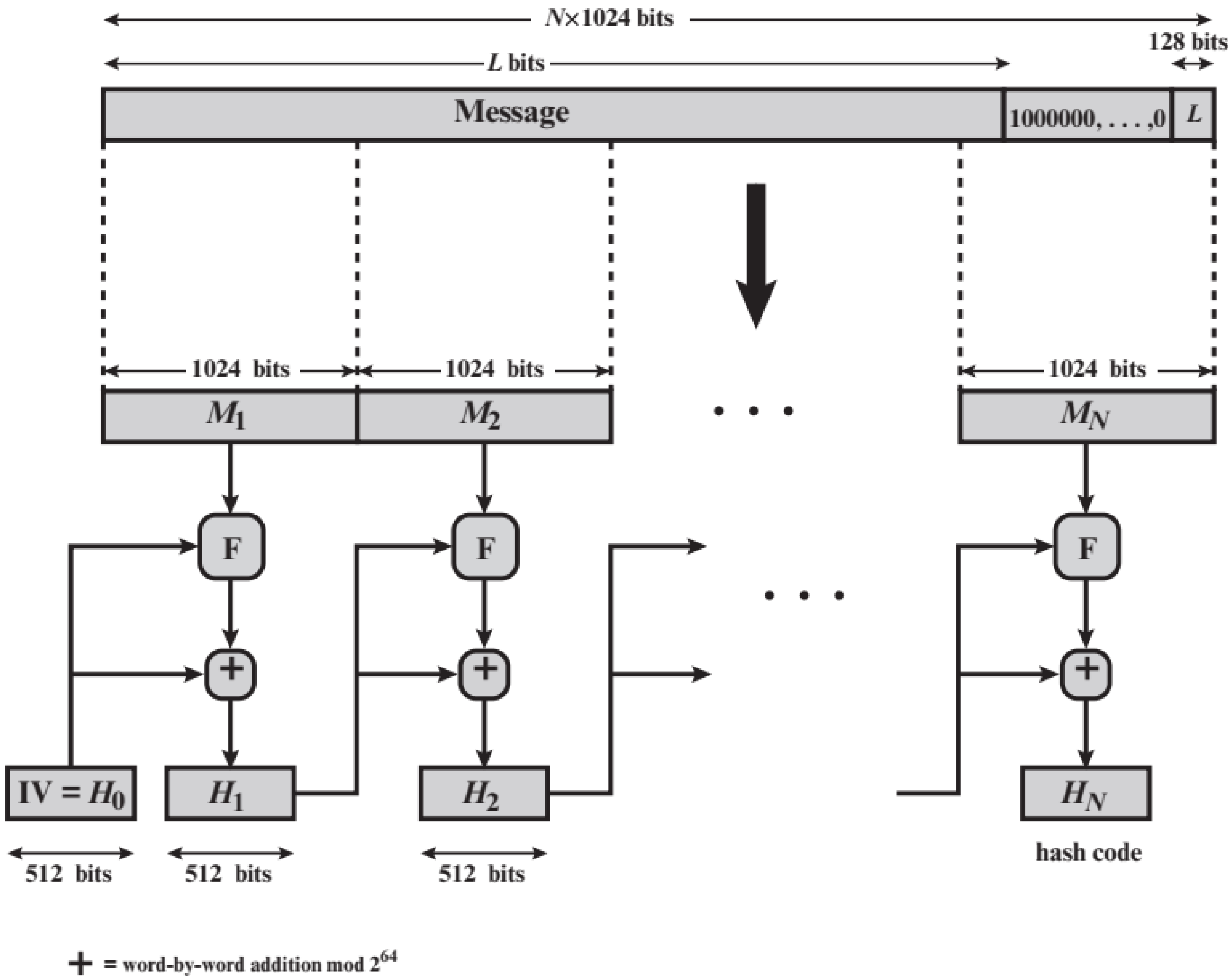


Figure 11.9 Message Digest Generation Using SHA-512

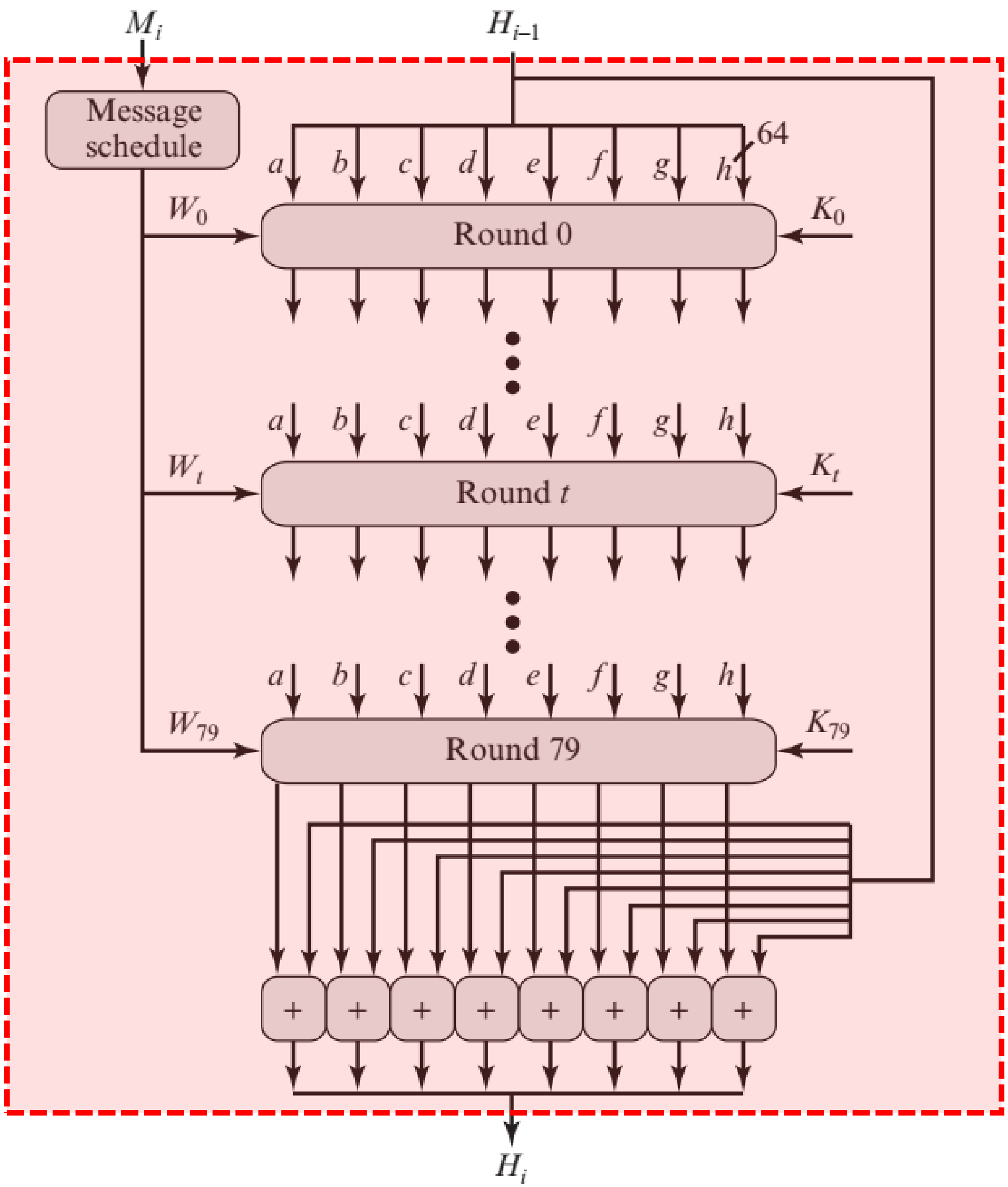


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

SHA-3

- **The Sponge Construction**
  - **allows both variable length input and output**
  - **Used for hash function, PRNG, etc.**
  - **defined by three parameters:**

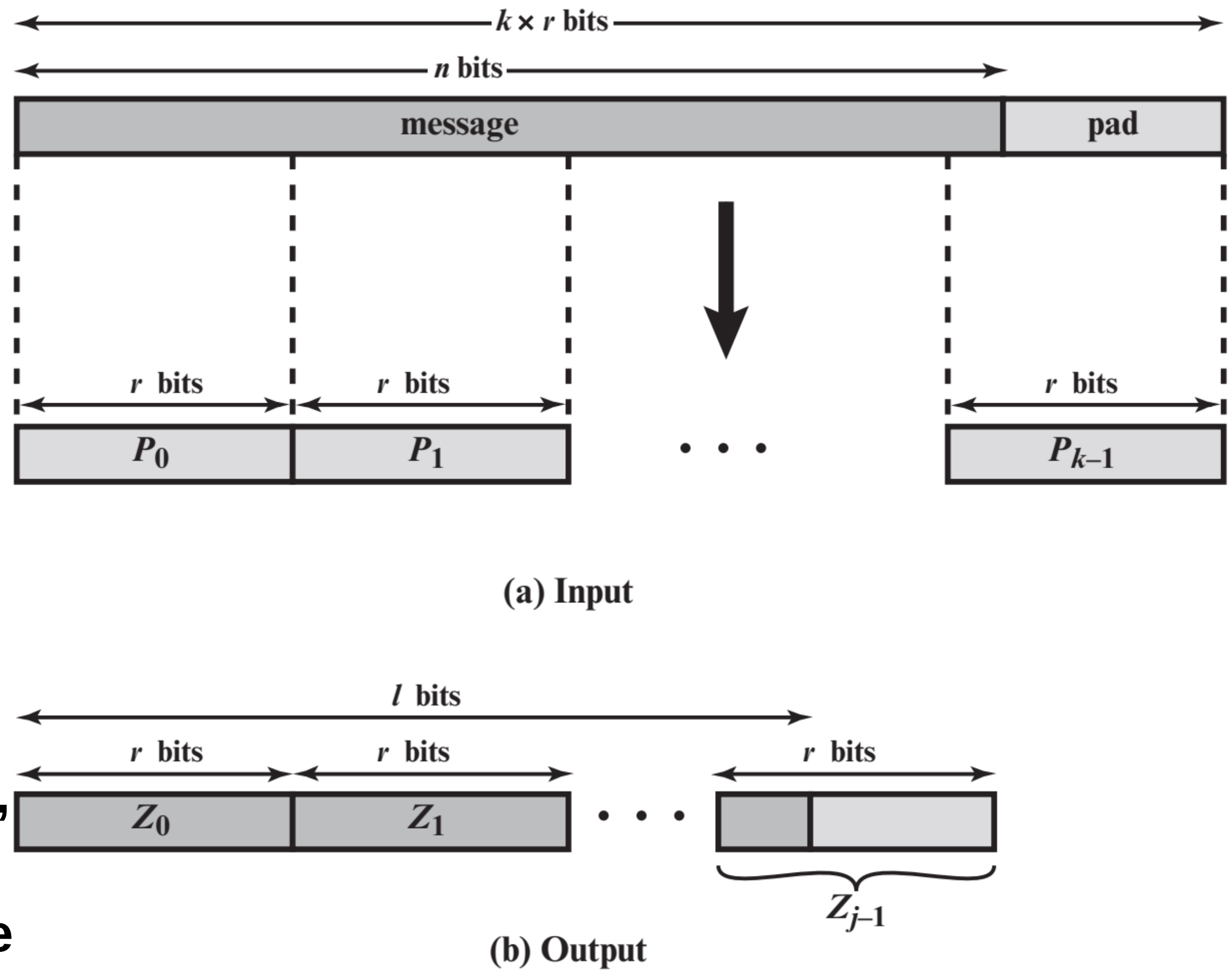


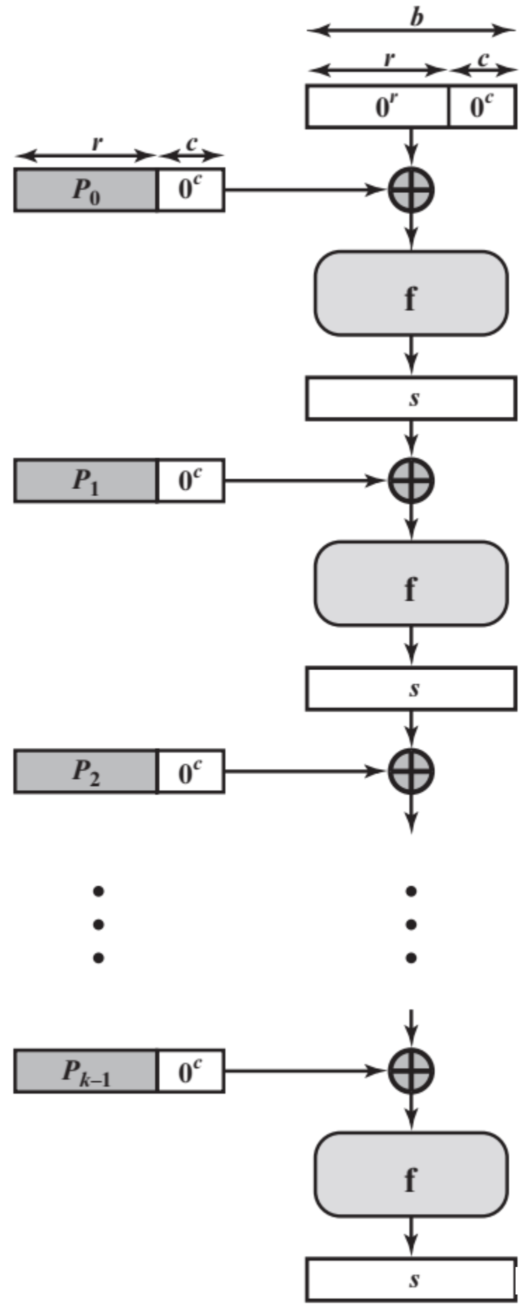
Figure 11.14 Sponge Function Input and Output

f = the internal function used to process each input block

r = the size in bits of the input blocks, called the **bitrate**

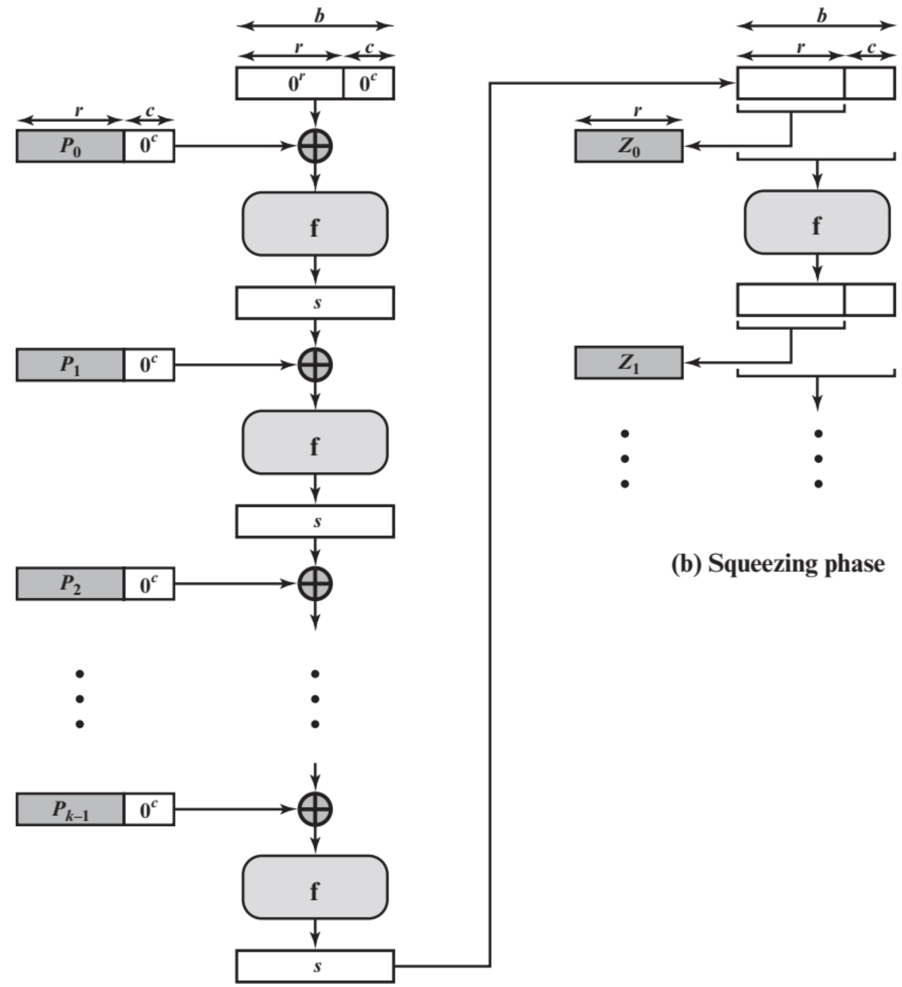
pad = the padding algorithm

- The Sponge construction consists of
  - Only one phase: the absorbing phase (if output length l satisfies: $l \le b$)
  - two phases: the absorbing phase and the squeezing phase. (if output length l satisfies: $l > b$)



Absorbing phase

- **The Sponge construction consists of**
  - **Only one phase: the absorbing phase (if output length *l* satisfies: $l \le b$)**
  - **two phases: the absorbing phase and the squeezing phase. (if output length *l* satisfies: $l > b$)**



(a) Absorbing phase

(b) Squeezing phase

Key Terms

absorbing phase big endian birthday attack birthday paradox bitrate capacity Chi step function collision resistant compression function cryptographic hash function hash code hash function hash value	Iota step function Keccak keyed hash function lane little endian MD4 MD5 message authentication code (MAC) message digest one-way hash function Pi step function preimage resistant	Rho step function second preimage resistant SHA-1 SHA-224 SHA-256 SHA-3 SHA-384 SHA-512 sponge construction squeezing phase strong collision resistance Theta step function weak collision resistance

Review Questions

11.1 What characteristics are needed in a secure hash function?

11.2 What is the difference between weak and strong collision resistance?

11.3 What is the role of a compression function in a hash function?

11.5 What basic arithmetical and logical functions are used in SHA?

Thanks!