UNIT-6 Cryptographic Hash Functions

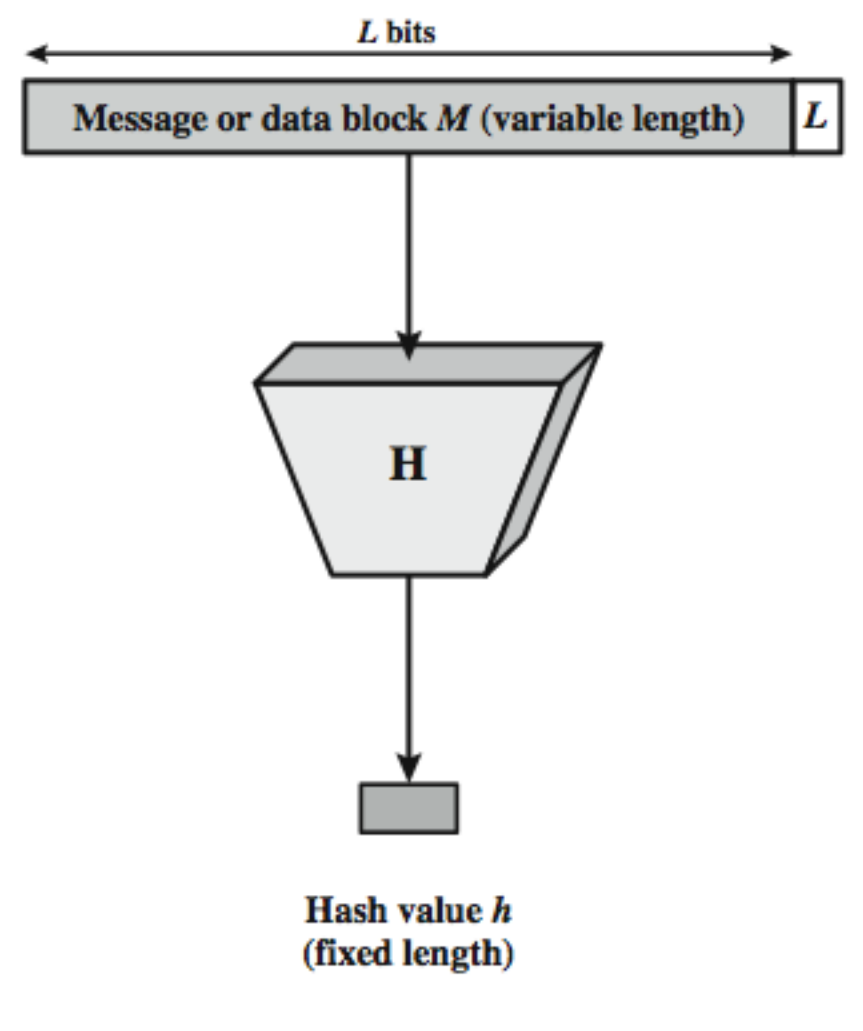
Hash Functions

- 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)

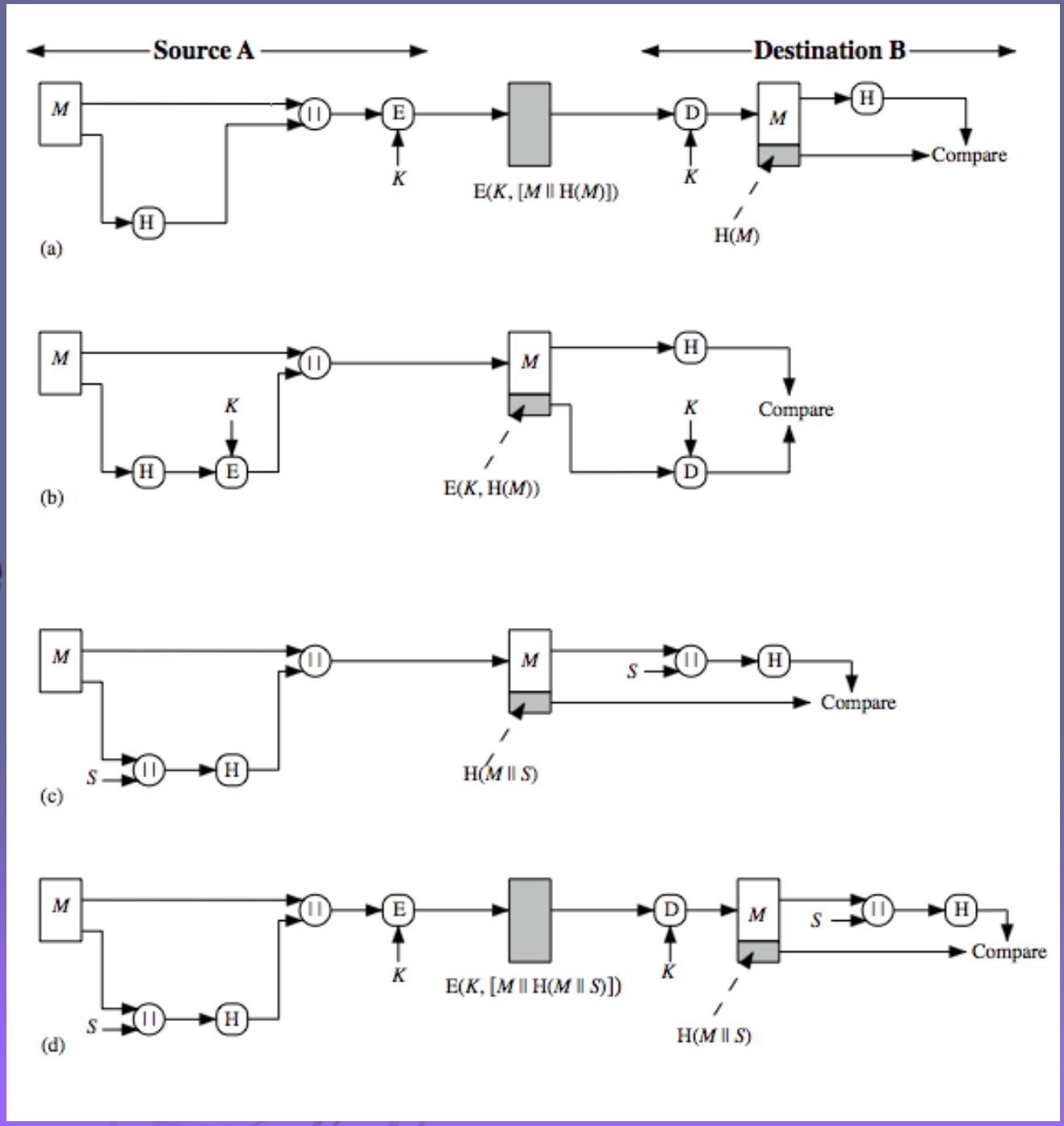
Cryptographic Hash Function



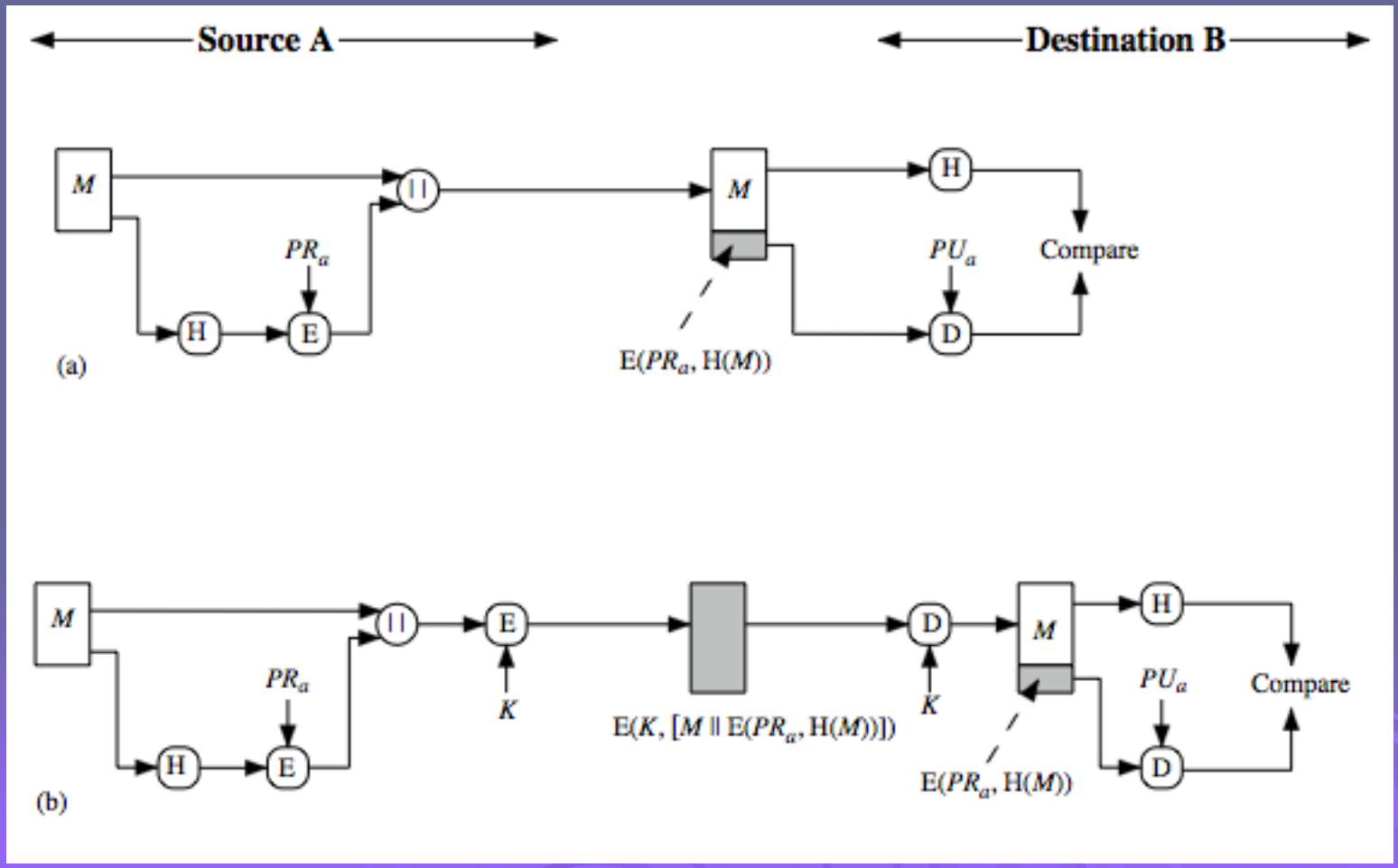
Application of hash function:

- Message Authentication
- Digital Signatures

Hash Functions & Message Authent-ication



Hash Functions & Digital Signatures



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)

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 by1bit and XOR block
  - good for data integrity but useless for security

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

Attacks on Hash Functions

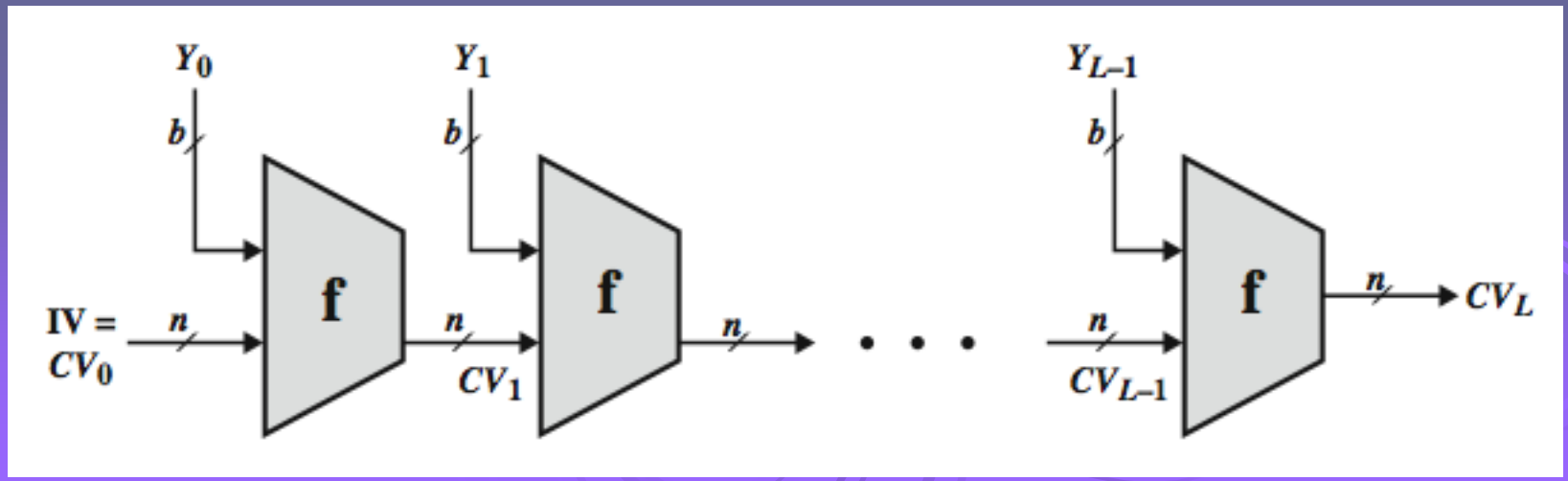
- have brute-force attacks and cryptanalysis
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

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

Hash Function Cryptanalysis

- cryptanalytic attacks exploit some property of alg so faster than exhaustive search
- hash functions use iterative structure
  - process message in blocks (incl 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
- other variants also susceptible to 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 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

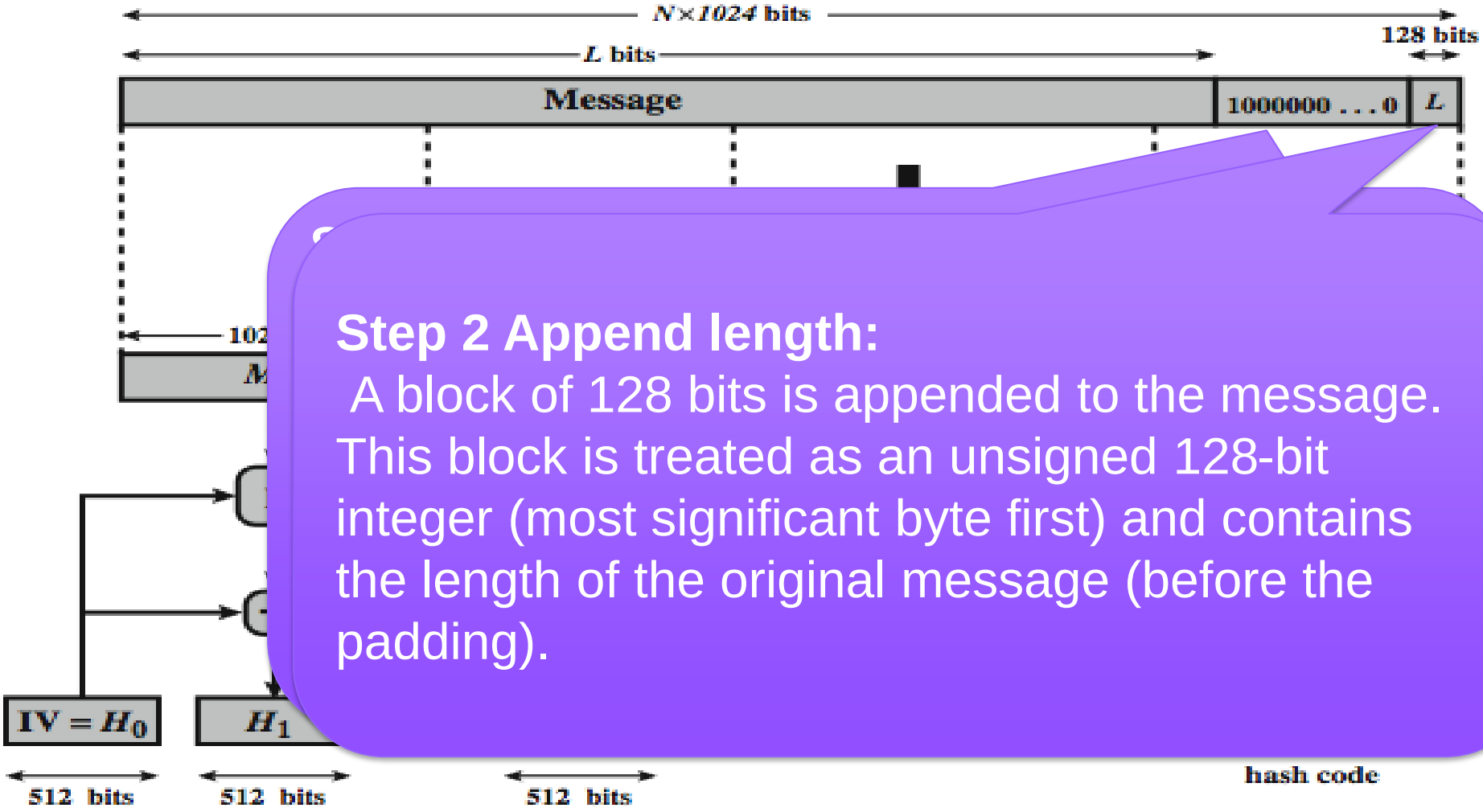
Revised Secure Hash Standard

- 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 Versions

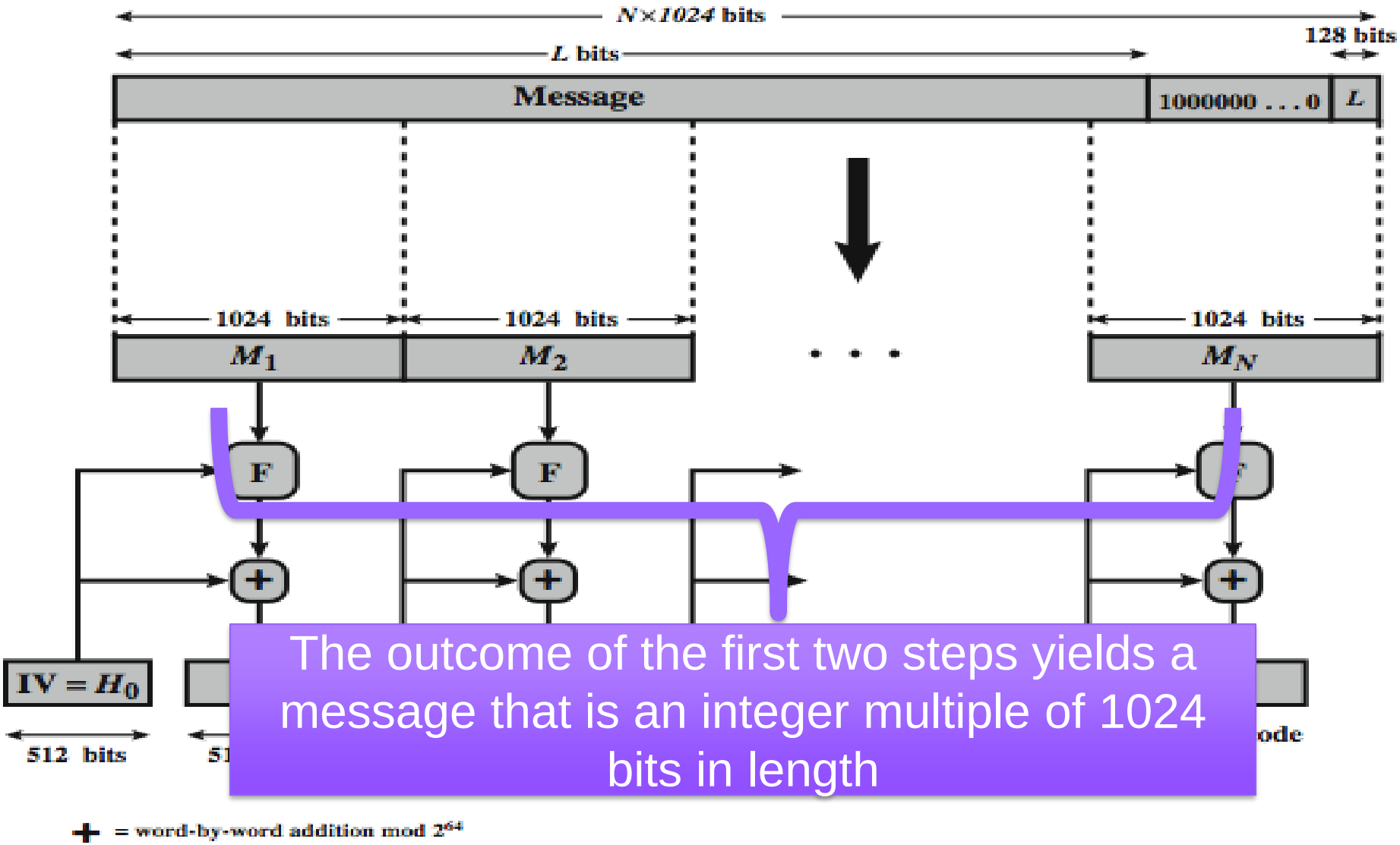
	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of steps	80	64	64	80	80

SHA-512 Overview



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).

SHA-512 Overview

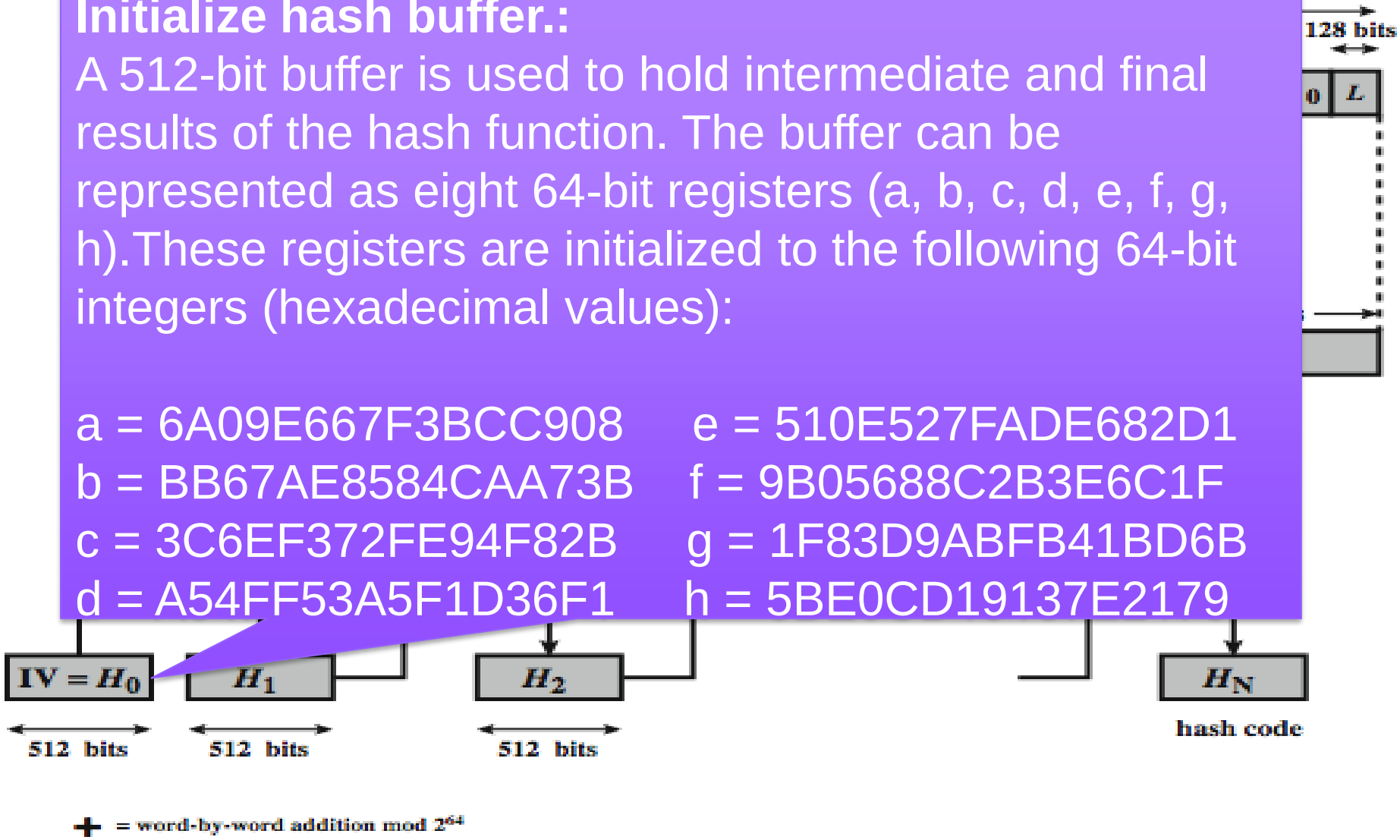


The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length

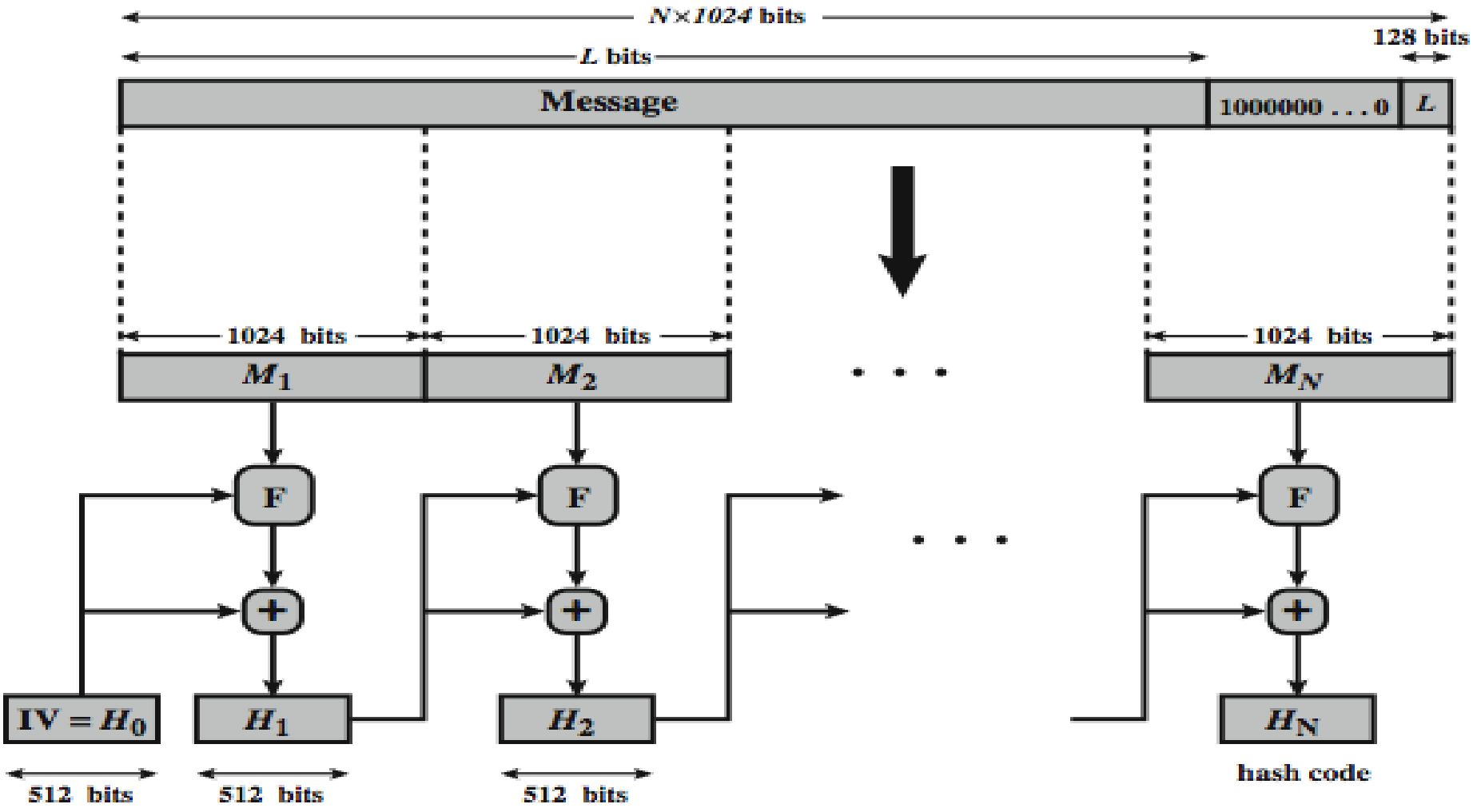
SHA-512 Overview

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 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
  - and a round constant based on cube root of first 80 prime numbers

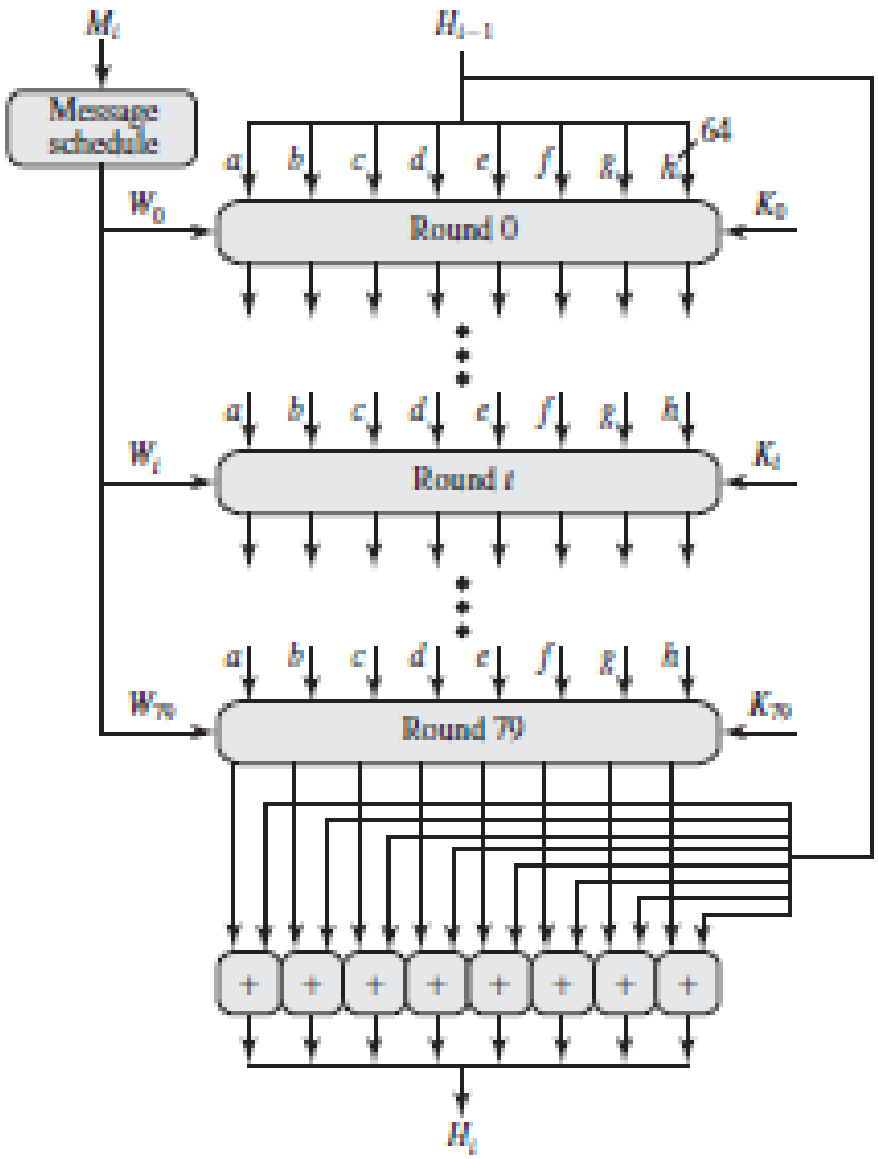
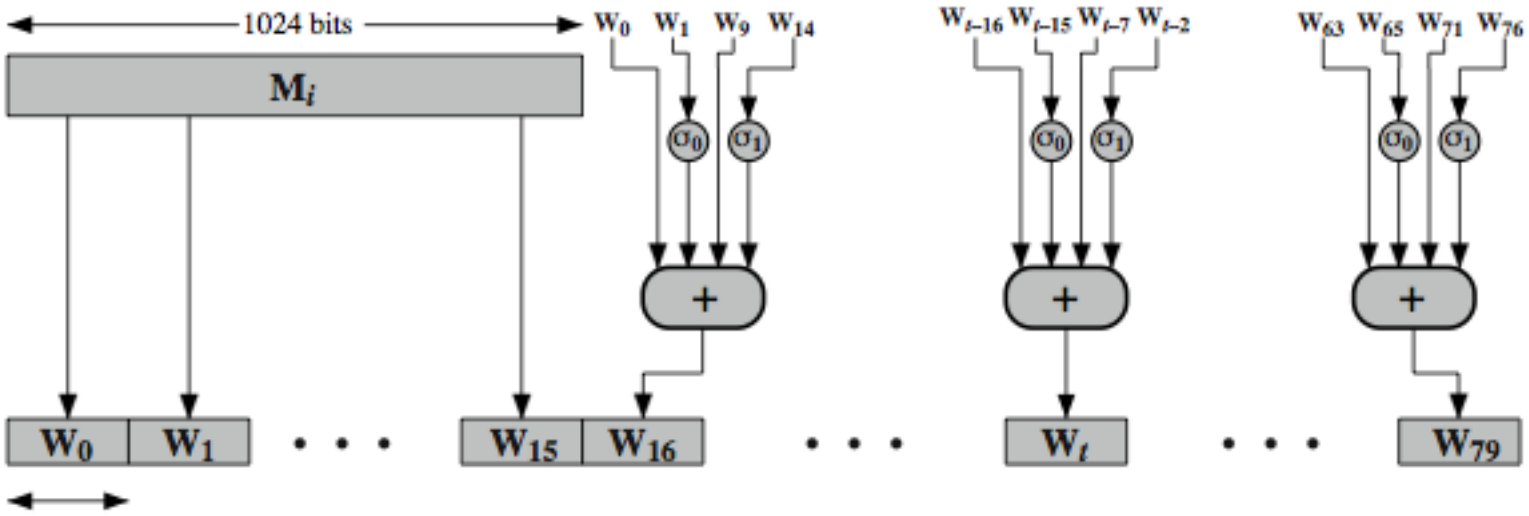


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

SHA-512 Round Function



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

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}

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:

$$H_0 = \text{IV}$$
$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$
$$MD = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

abcdefgh_i = the output of the last round of processing of the 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

Let us look in more detail at the logic in each of the 80 steps of the processing of one 512-bit block (Figure 11.10). Each round is defined by the following set of equations:

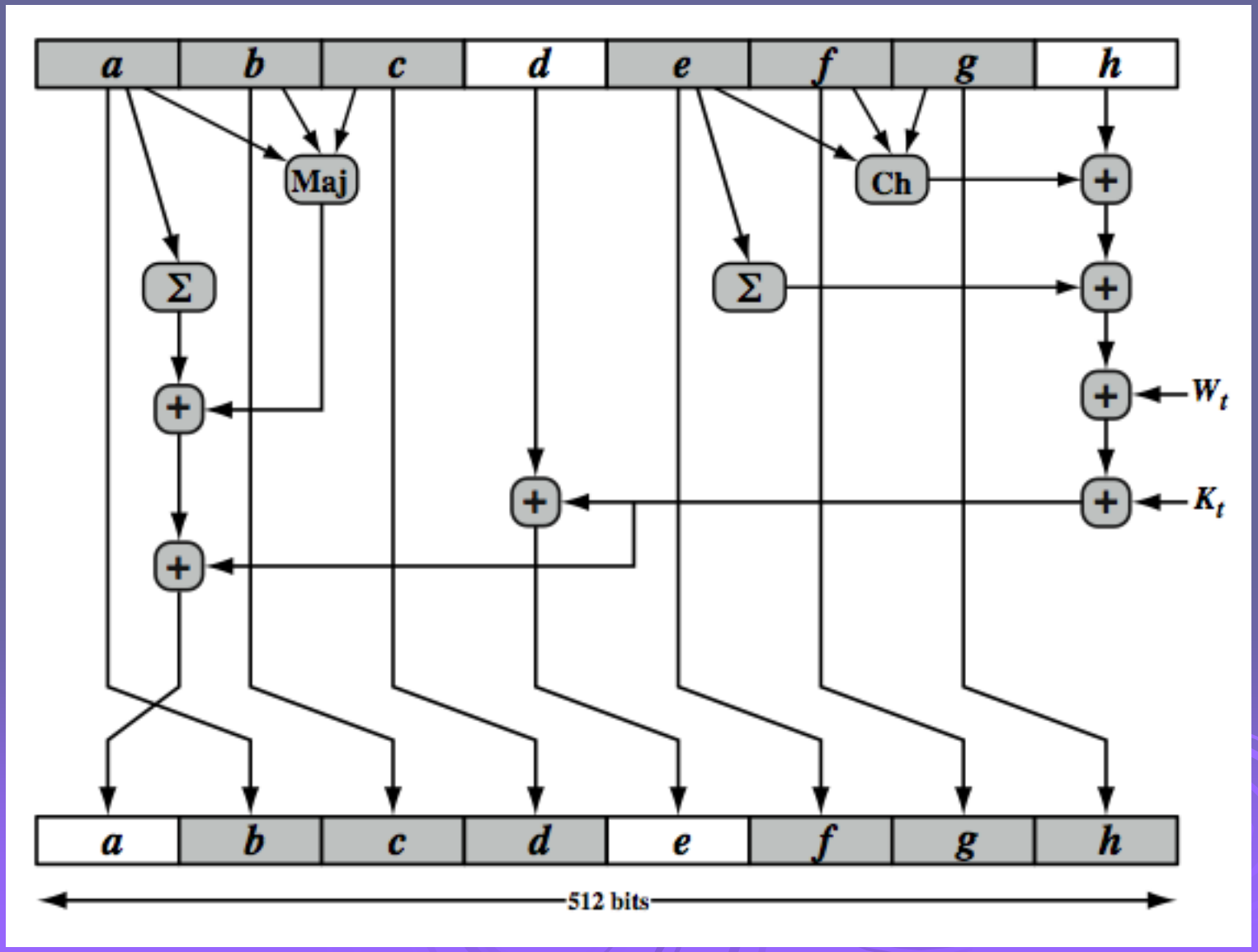
$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$
$$T_2 = \left(\sum_0^{512} a\right) + \text{Maj}(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

where

t = step number; $0 \leq t \leq 79$

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

SHA-512 Round Function



$\mathrm{Maj}(a, b, c)$ – $(a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

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

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

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

W_t – a 64-bit word derived from the current 512-bit input block

K_t – a 64-bit additive constant

$+$ – addition modulo 2^{64}

Two observations can be made about the 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
  - goal to have in place by 2012 but not fixed

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