Cryptographic hash functions

Killian O'Brien

6G6Z0024 Applied Cryptography 2024/25

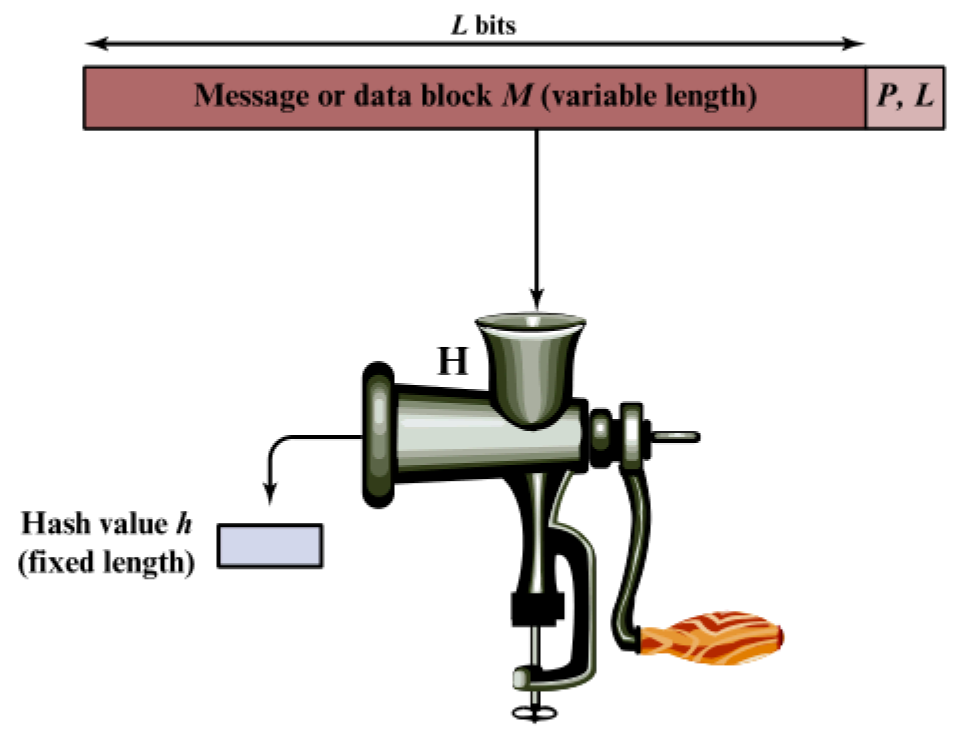
Lecture Week 09 – Wed 27 November 2024

Introduction

- Reading: Stallings, Chapter 11, *Cryptographic Hash Functions*

Basic idea of a (cryptographic) hash function H

- H takes a variable-length block of data M and computes a fixed sized output $h = H(M)$.
- h called the *hash value*, or *hash code* or *message digest*.
- Typically the input message M is split into standard sized blocks, with padding P if necessary, often including data on the length L of M.
- Typically h will come from a very large domain of possible values, and different inputs M will produce outputs h that are evenly distributed and apparently random selections from this domain.
- The inner workings of H will be somewhat complex, but still have fast implementations in software and/or hardware.
- For cryptographic hash functions, the inner workings of H will ensure security, in that it should be computationally infeasible to compute
  - a message M that will hash to a given hash value h (pre-image resistance)
  - two messages M_1 and M_2 that hash to the same hash value (collision resitance)



P, L = padding plus length field

Figure 11.1 Cryptographic Hash Function; $h = \mathrm{H}(M)$

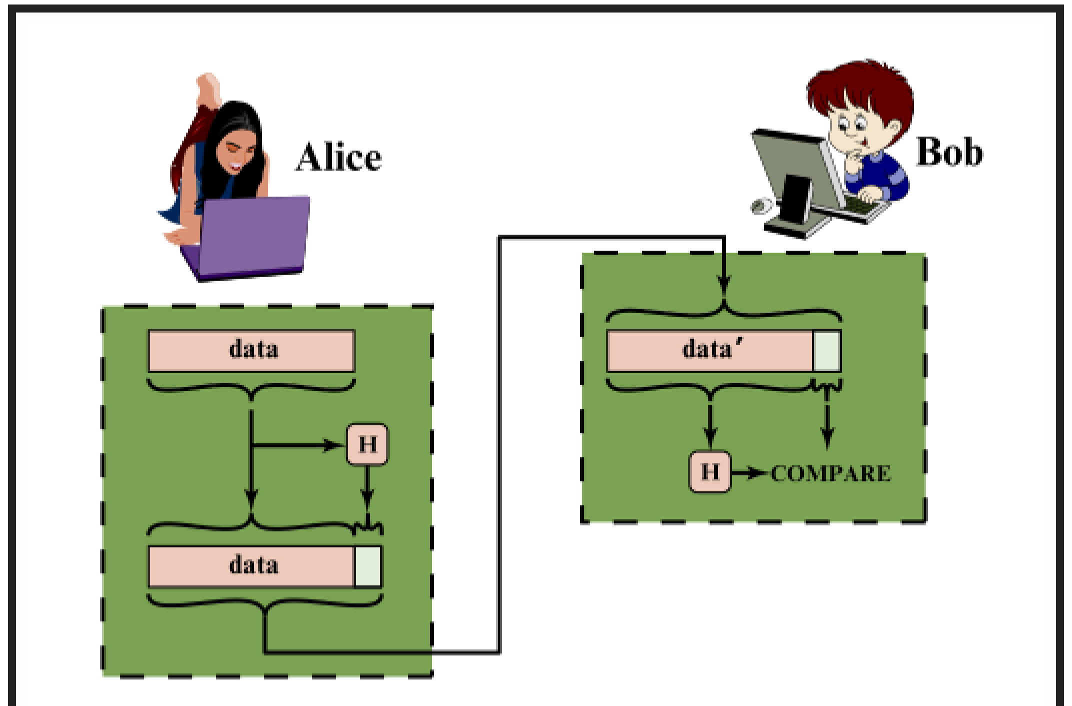
Applications in

- Message authentication, Digital signatures, Password authentication, …

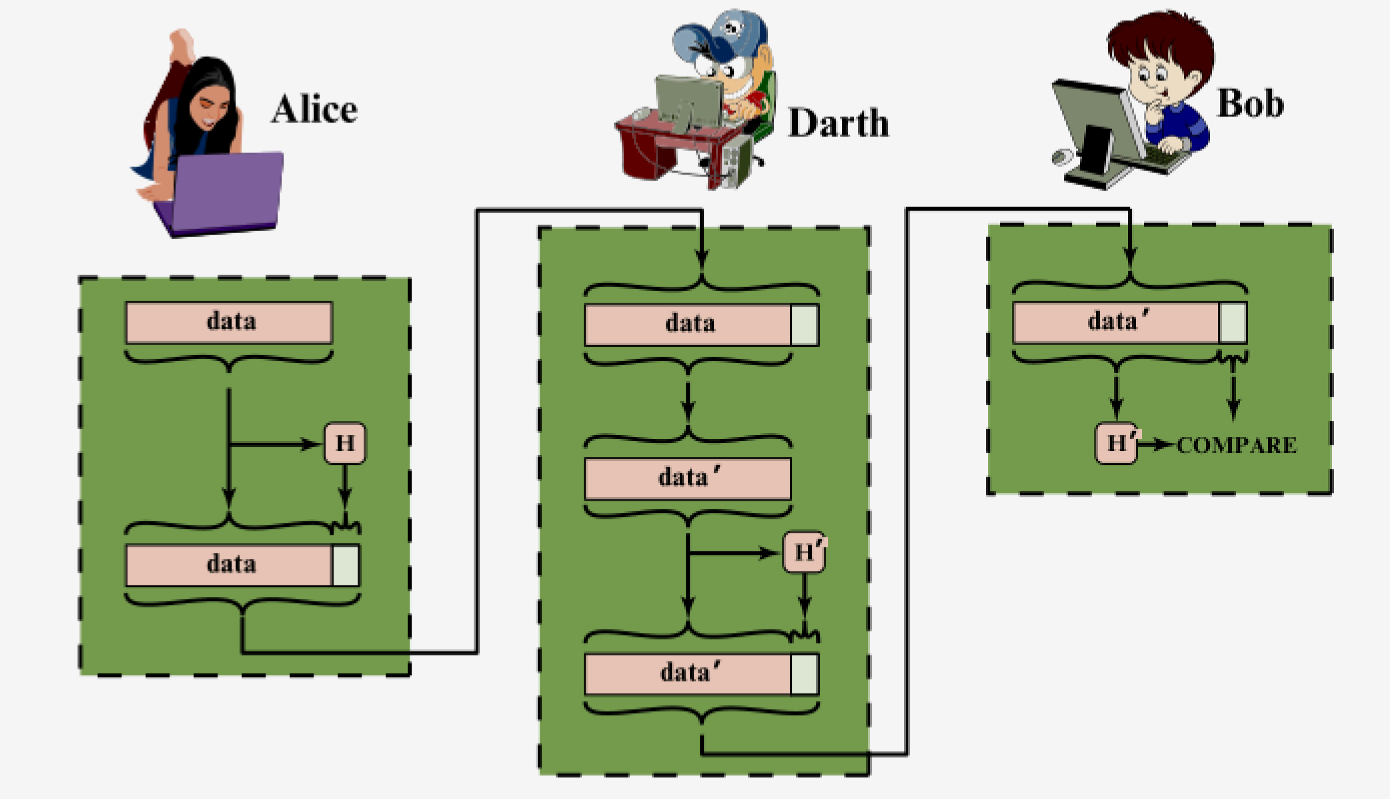
Applications of cryptographic hash functions

Message authentication

- H can be used to verify the integrity of data in storage or communication. To ensure that it has not been changed.
- In very simple terms, the hash value can be appended to the data as shown on below on left.
- This is too simple though, and susceptible to *man-in-the-middle attacks*, as shown on right.



(a) Use of hash function to check data integrity



(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

Applications of cryptographic hash functions

Message authentication possibilities

- The diagram on the right shows some ways in which H can be used to implement message authentication.
- Diagram key
  - a cryptographic hash function H
  - message M to be communicated
  - concatenation operator $||$
  - a secret symmetric key K
  - symmetric key encryption E
  - a shared secret value S

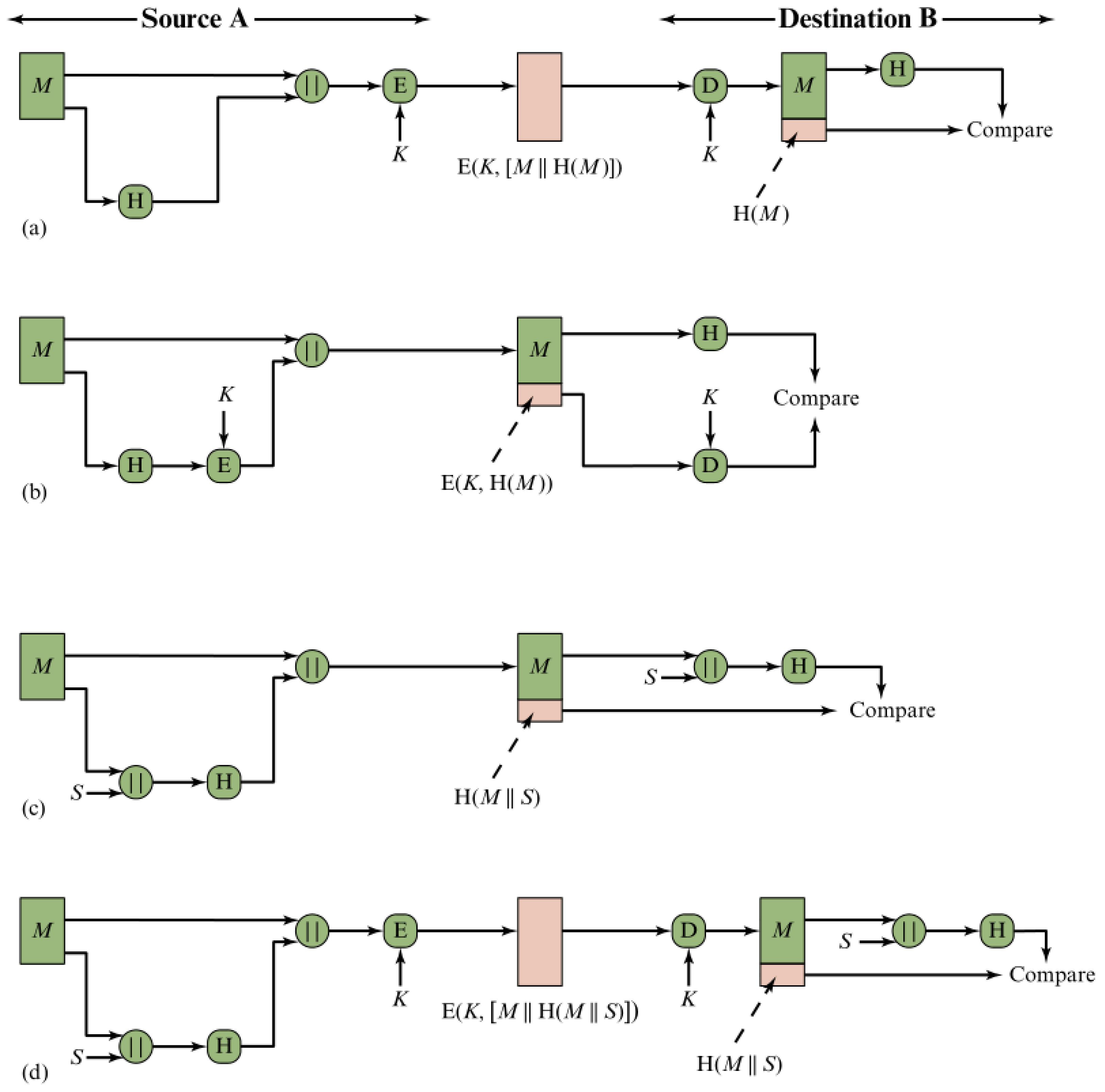


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

Applications of cryptographic hash functions

Digital signatures

- Here the emphasis is on authentication and proof of the identity of the sender of the message M
- The hash function H can be used in combination with asymmetric public key encryption E, with public PU_a, and private PR_a, keys of the sender A.
- Diagram key
  - a cryptographic hash function H
  - message M to be communicated
  - concatenation operator $||$
  - sender A's (public,private) key pair (PU_a, PR_a)
  - symmetric/asymmetric (depending on keys used) encryption E

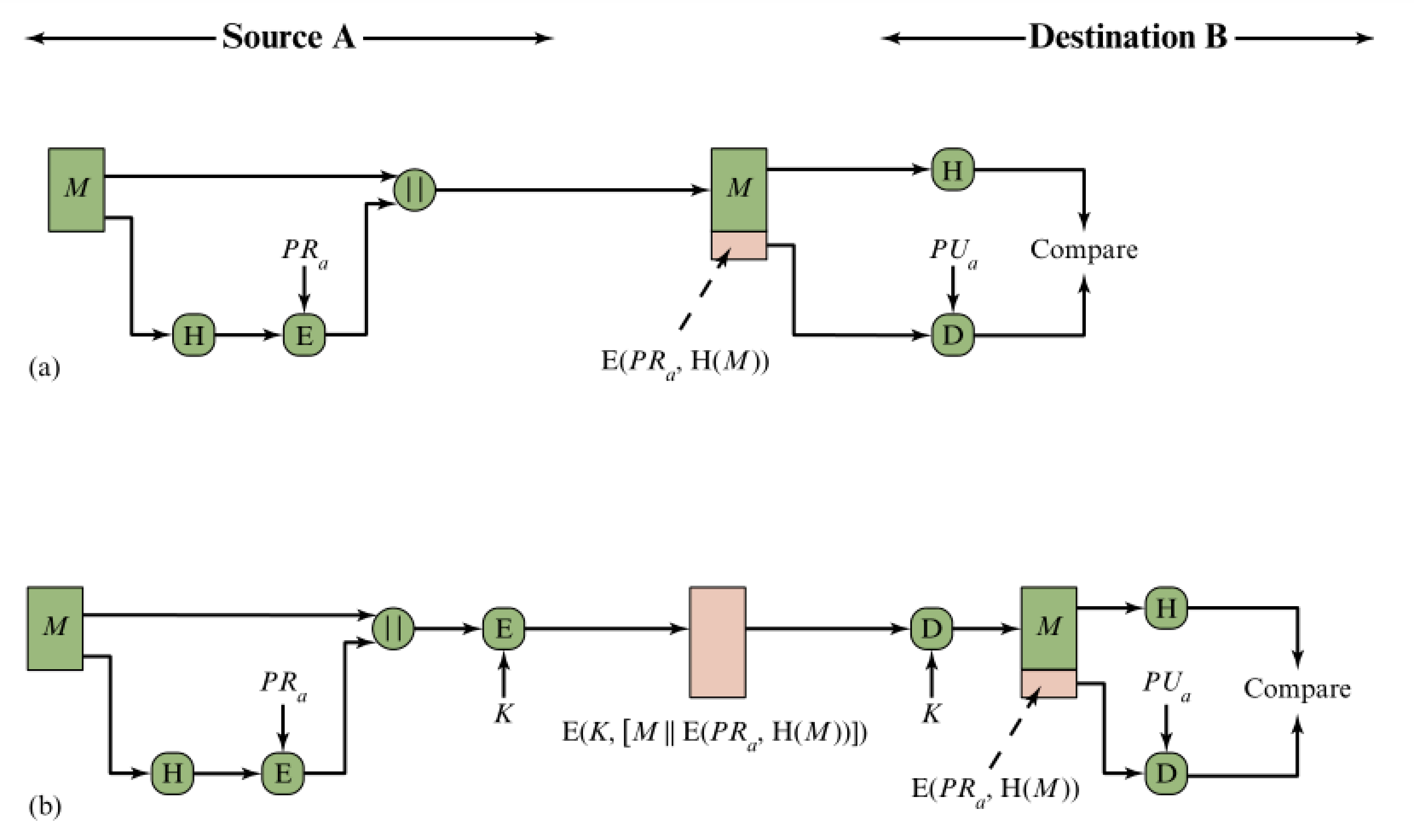


Figure 11.4 Simplified Examples of Digital Signatures

Merkle-Damgard iterated hash function design

- Merkle and Damgard proposed this iterated design based on a *compression function* f that produces an n-bit hash value.
- The input message is split into L fixed-size blocks Y_i of b bits each.
- The final block Y_{L-1} also contains the length information of the original message and any necessary padding to make it up to b bits.
- The i^{th} compression function f takes two inputs, the message block Y_{i-1} and an n bit *chaining variable* CV_{i-1}, and produces an n-bit output chaining variable CV_i.
- $CV_0 = IV = \text{ standard initial } n\text{-bit value}$
- $CV_i = f(CV_{i-1}, Y_{i-1}), \quad 1 \leq i \leq L$
- $H(M) = CV_L$

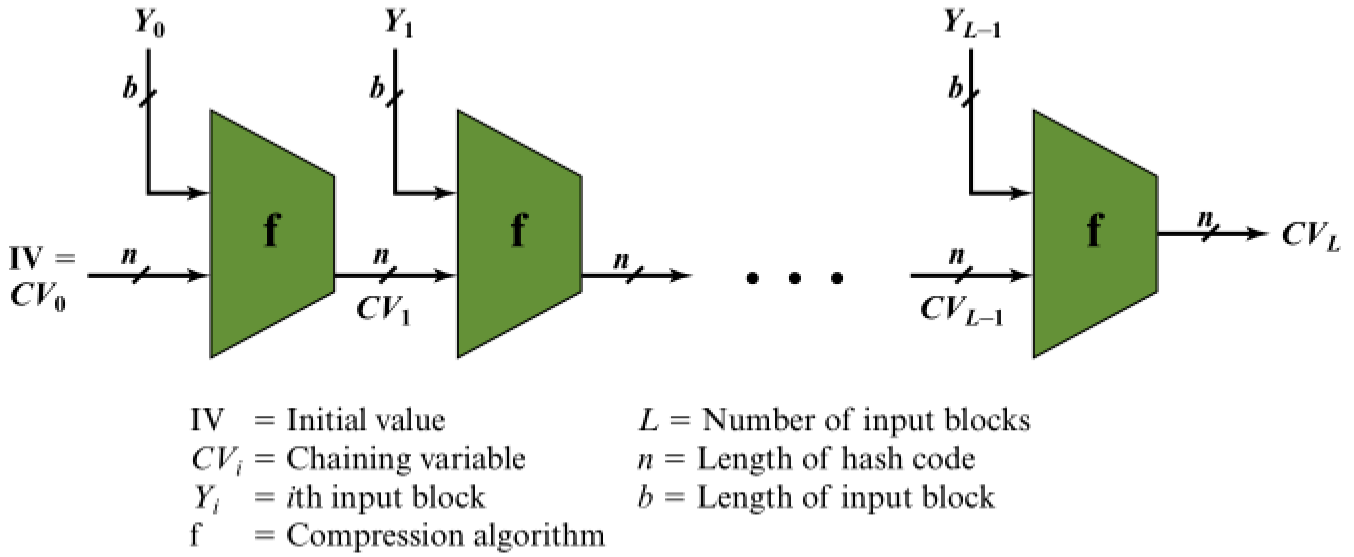

Y_0
b
Y_1
b
Y_{L-1}
b
f
f
f
IV = CV_0
n
n
CV_1
n
. . .
n
CV_{L-1}
n
CV_L
IV = Initial value
CV_i = Chaining variable
Y_i = ith input block
f = Compression algorithm
L = Number of input blocks
n = Length of hash code
b = Length of input block

Figure 11.8 General Structure of Secure Hash Code

The SHA family and SHA-512

- Secure Hash Algorithm (SHA) family produced by NIST have been the most widely used cryptographic hash functions.
- Offers hash functions of increasing complexity and security.
- As earlier versions become vulnerable to discovered weaknesses in combination with brute-force attack, then later versions adopted.
- Currently SHA-256 and SHA-512 seen as sensible.

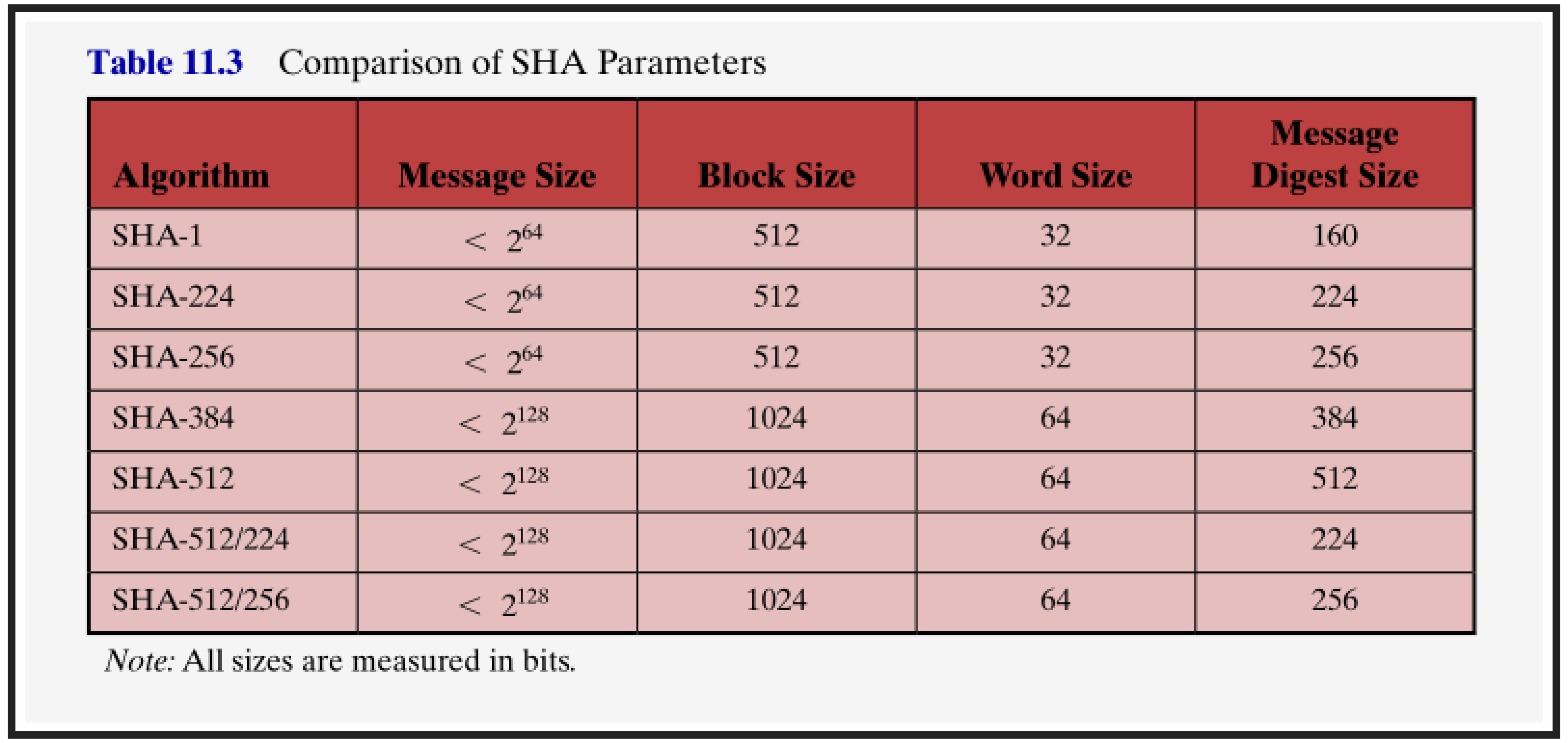


Table 11.3 Comparison of SHA Parameters

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.

SHA-512

Overall design

- Input message M is padded with `1000...0` until its length is $\equiv 896 \pmod{1024}$.
- Then the bit-length L of M is appended as an unsigned 128-bit integer.
- We now have some number N of 1024-bit blocks, $M_1, M_2, \dots, M_N$.
- The intermediate hash buffer is initialized with standard initial values $H_0 = IV$, derived from the fractional parts of the square roots of the first eight prime numbers (i.e. essentially random bits).
- An intermediate step of SHA-512 takes the block M_i and previous intermediate hash buffer H_{i-1} and processes it with an 80-round processing module F.
- This produces a 512-bit output that is combined with an addition-like operation with H_{i-1}.
- The final 512-bit output H_N is the hash value of SHA-512.

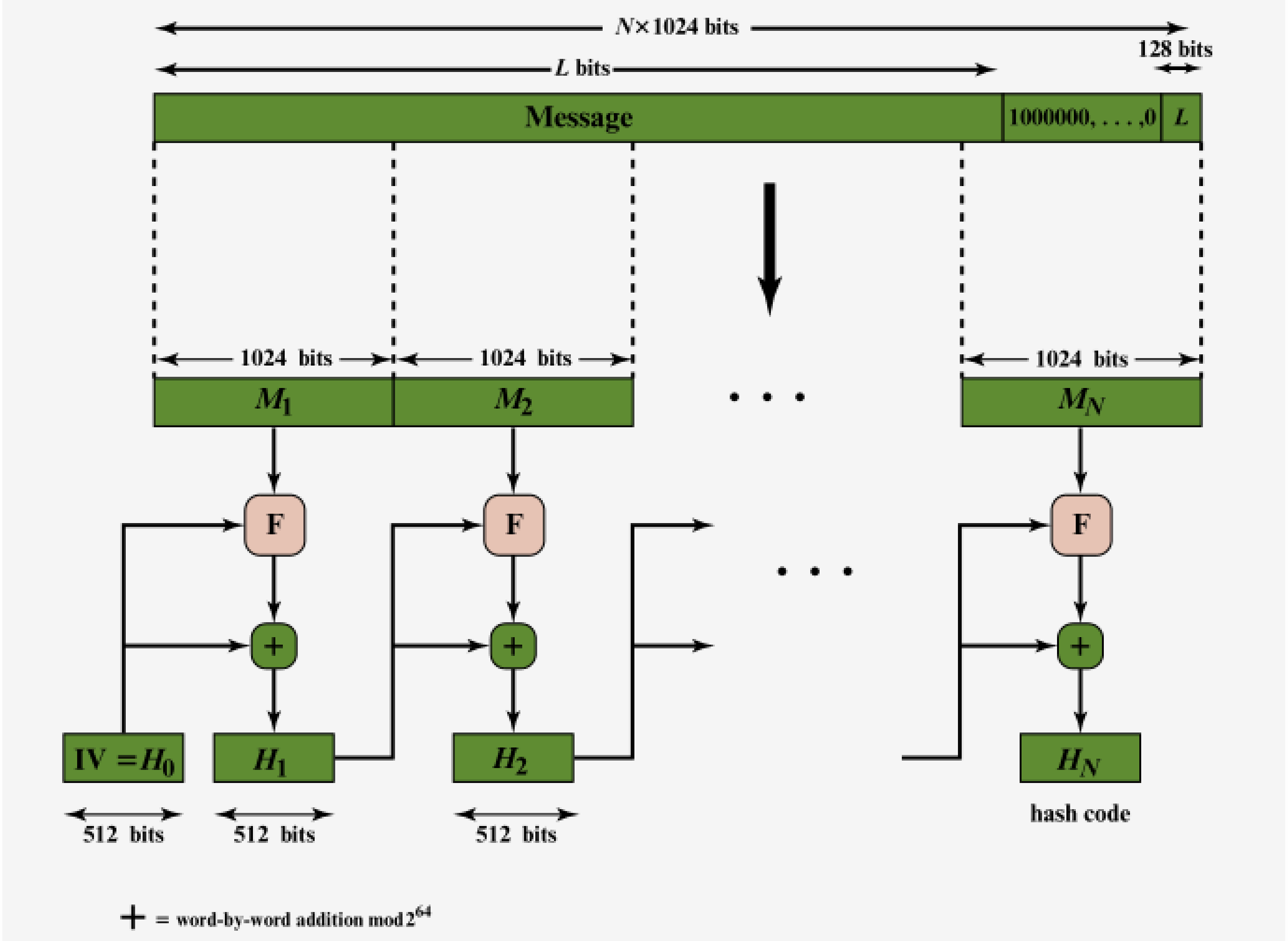


Figure 11.9 Message Digest Generation Using SHA-512

The 80-round processing module F

- This diagram shows the makeup of the processing module F.
- Operates on the 8 64-bit words making up the 512-bit intermediate hash buffer H_{i-1}.
- The message block M_i supplies 80 different 64-bit words W_t via a *message schedule* algorithm, one for each round.
- 80 different 64-bit constants K_t are used, one for each round. These are derived from the fractional parts of the cube roots of the first 80 prime numbers, again this produces essentially random bits for input.
- The additions $+$ shown here are carried out word-by-word on the 8 64-bit pairs of inputs, one from H_{i-1}, one from the corresponding output of the last of the 80 rounds. The additions are carried out modulo 2^{64}.

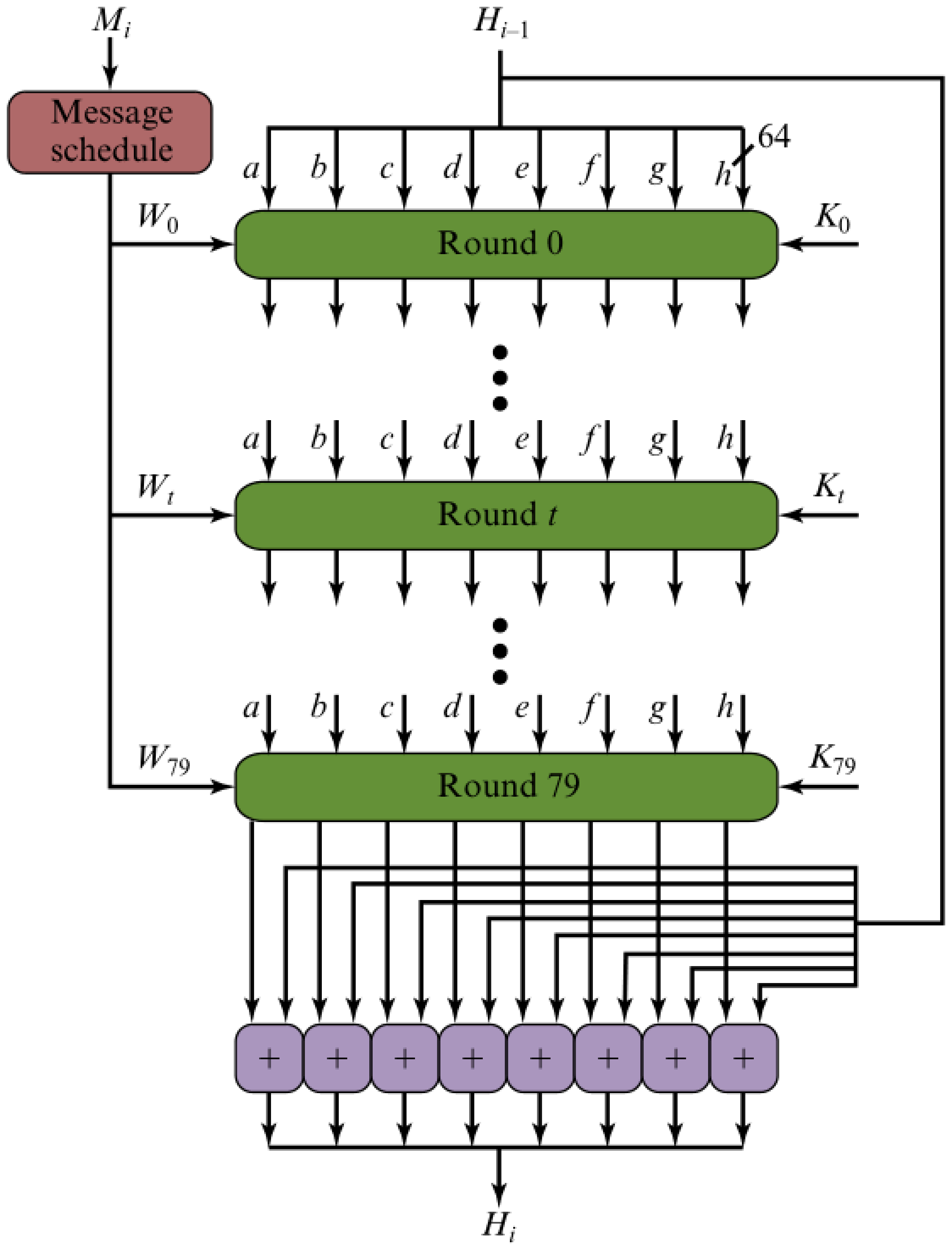


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

The detail of the 80 rounds

- This shows the operations within the one of the 80 rounds.
- The 8 input 64-bit words a, b, c, d, e, f, g, h are processed to the 8 output 64-bit words a, b, c, d, e, f, g, h. Some by relabelling, and others by more complex operations.
- The additions $+$ shown here are word additions modulo 2^{64}.
- Employs bit-wise logical functions Ch and Maj, $\oplus$ denotes bit-wise XOR
- $\mathrm{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{ NOT } e \text{ AND } g)$. This is equivalent to *If e then f, else g.*
- $\mathrm{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$. This is true only if the majority of the arguments (i.e. 2 or 3 of them) are true.
- $\Sigma(a) = \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$, where ROTR^{n} denotes circular right shift of the bits by n bits.
- $\Sigma(e) = \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$, where ROTR^{n} denotes circular right shift of the bits by n bits.

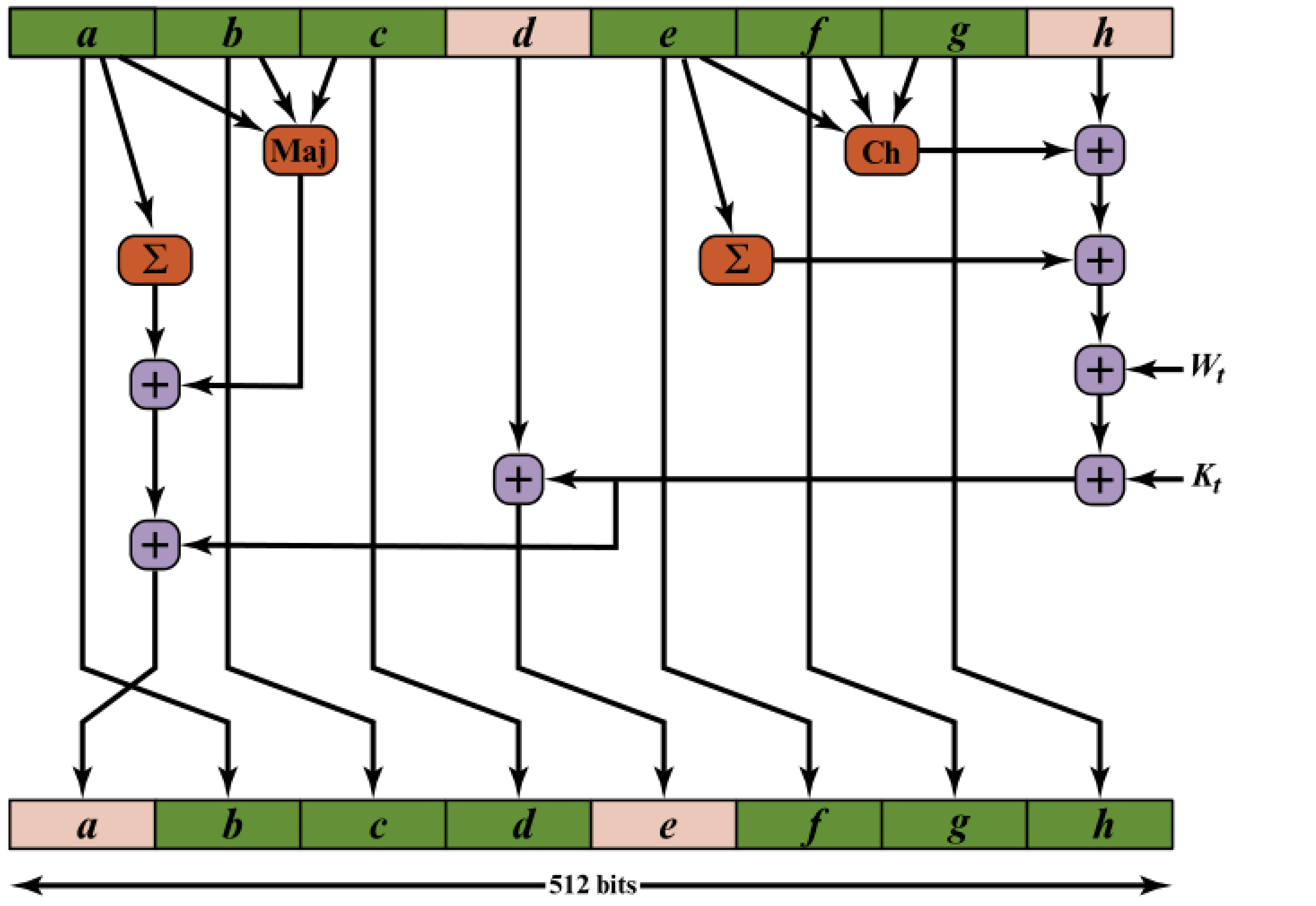


Figure 11.11 Elementary SHA-512 Operation (single round)

SHA-512 message schedule algorithm

- This shows the operations within the one of the 80 rounds.
- First 16 64-bit words $W_0, \ldots, W_{15}$ obtained directly from M_i.
- Subsequent words obtained by processing previous words.
- $W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$
- These are word by word additions modulo 2^{64}
- $\sigma_0(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$

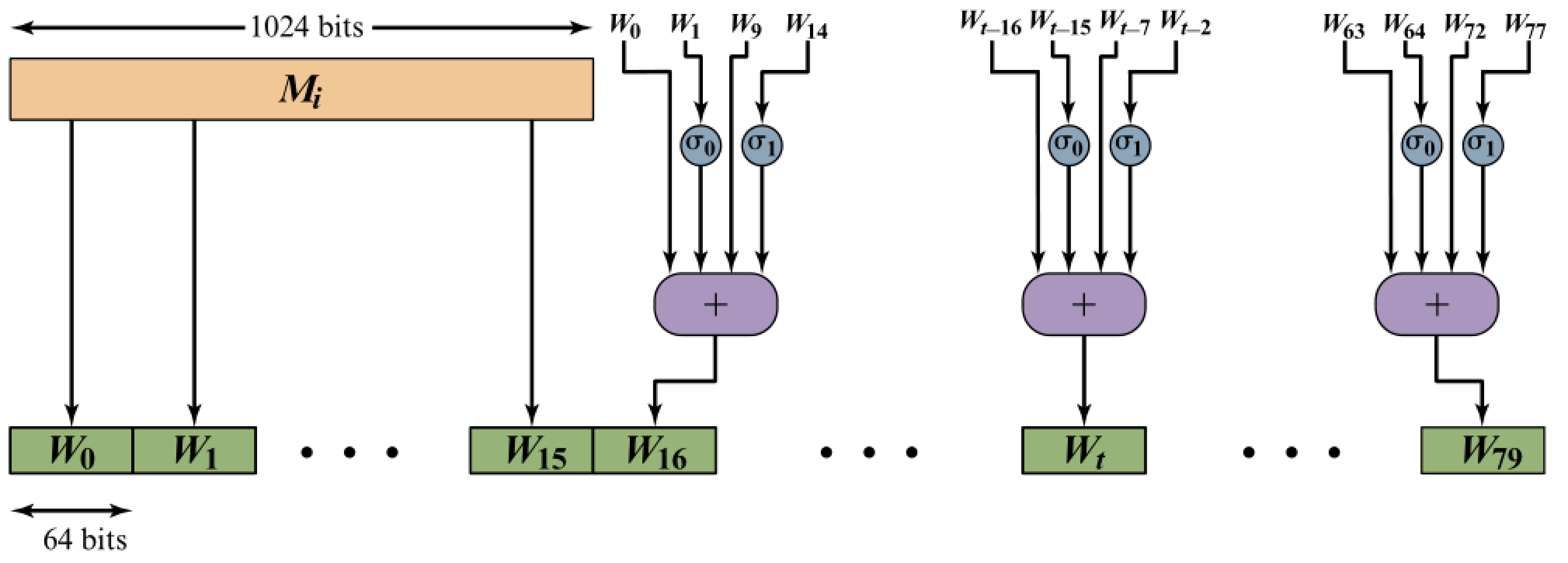


Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

- $\sigma_1(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$
- where ROTR^n is as before, a right circular shift by n bits, and SHR^n is right shift of the 64-bit word by n bits, and padding with $64 - n$ zeros on the left. So SHR^n loses bit information.