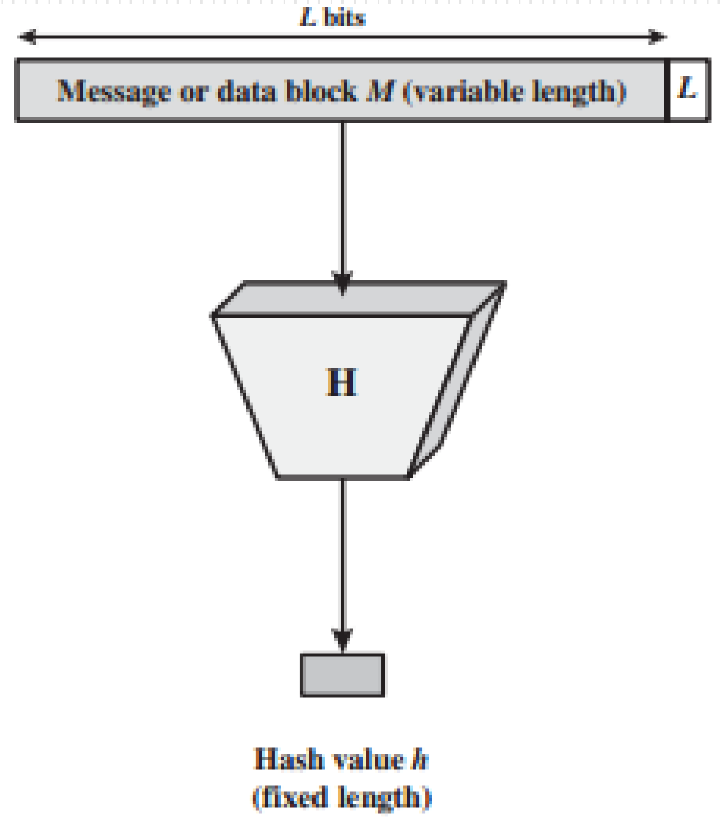
# CRYPTOGRAPHIC HASH FUNTIONS

# Contents

1) Hash function
2) Applications of Cryptographic Hash Functions
3) Simple Hash Functions
4) Requirements and Security
5) Secure Hash Algorithm (SHA)

# 1. Hash function

- A hash function maps a variable-length message into a fixed-length hash value, or message digest: h= H(M)



# 1. Cryptographic hash function.

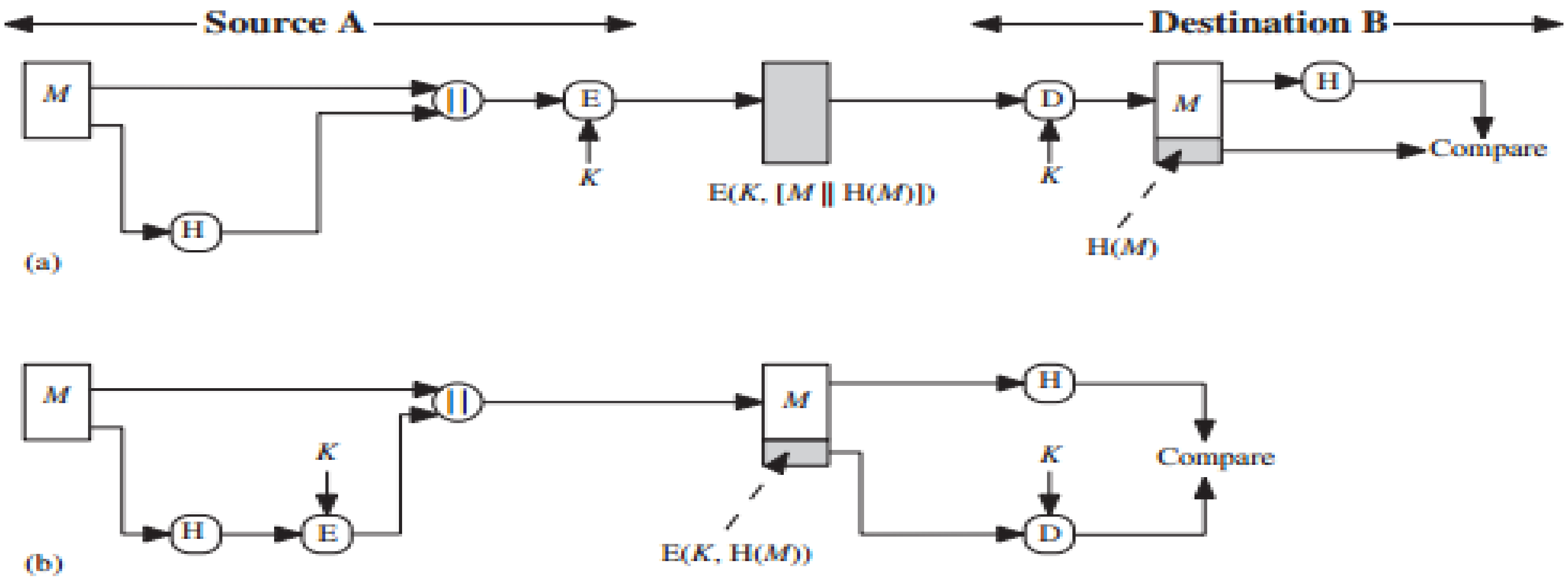
- A cryptographic hash function is an algorithm for which it is computationally infeasible to find either a data object that maps to a pre-specified hash result or two data objects that map to the same hash result

# 2. Applications Of Cryptographic Hash Functions

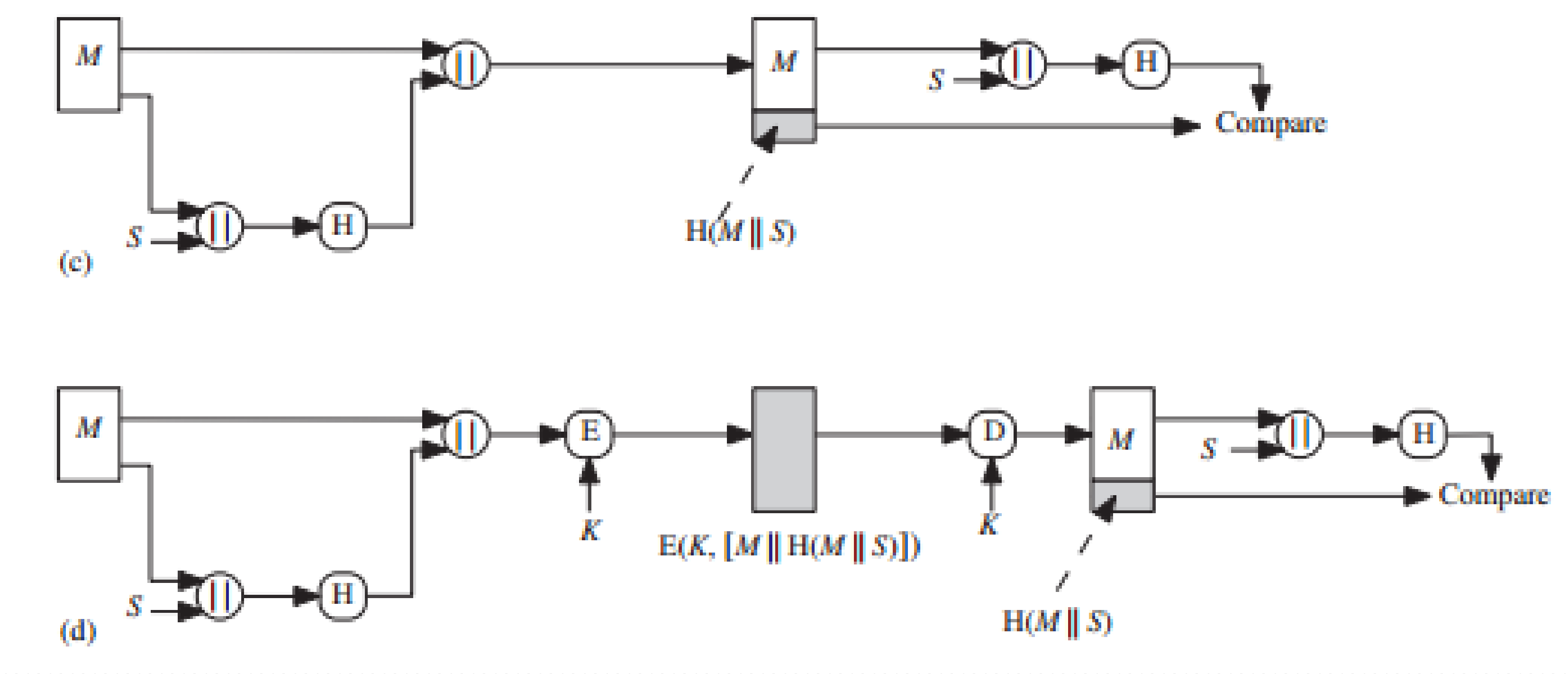
- Message Authentication
- Digital signature

# Message Authentication

- Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent

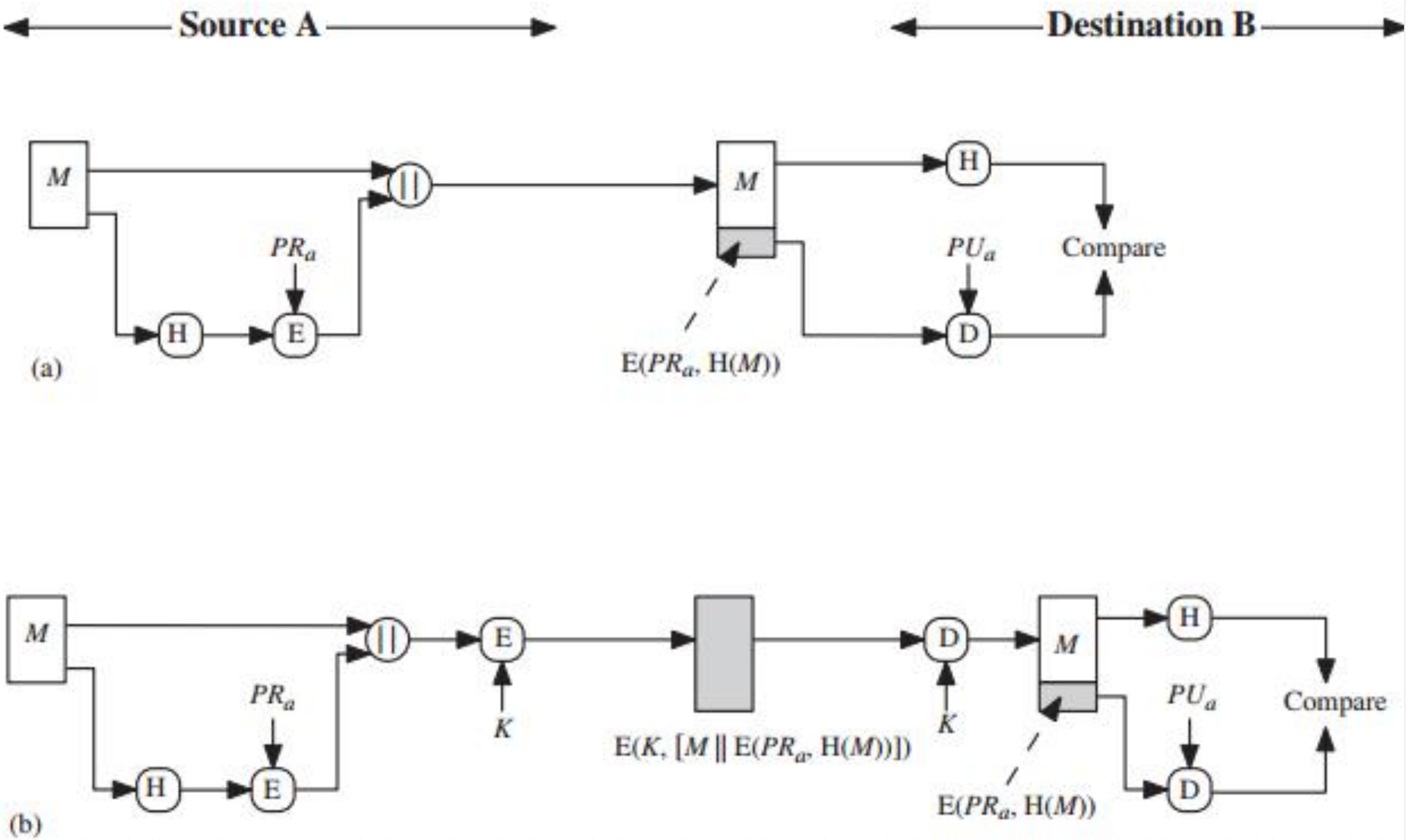


# Message Authentication



# Digital signature

- 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 that is associated with the digital signature

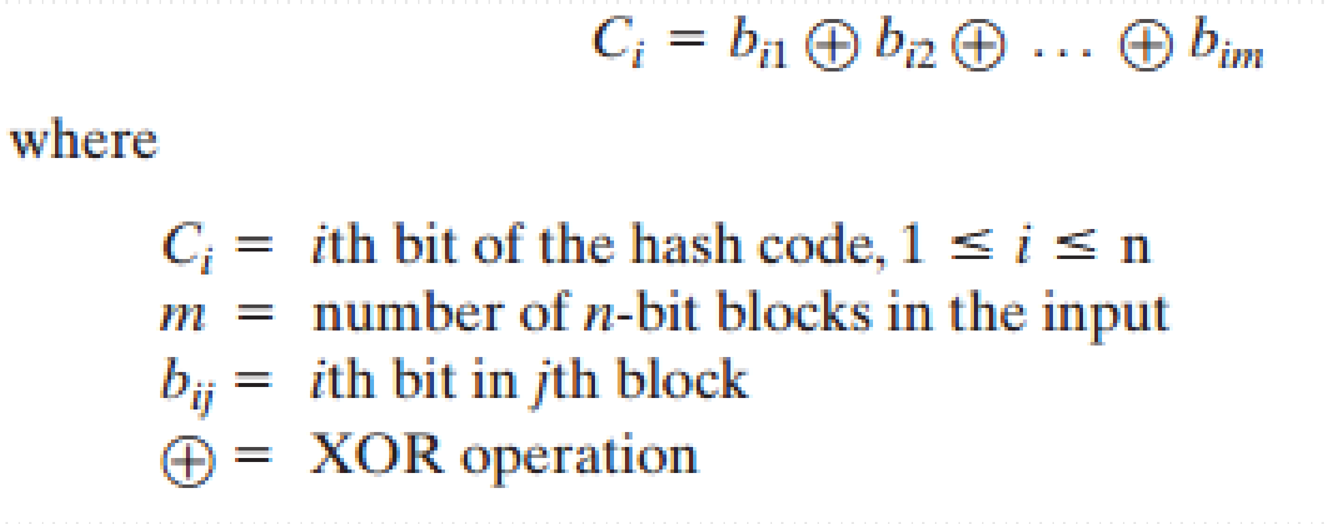

Source A
Destination B
$M$
||
H
E
$PR_a$
(a)
$M$
$\mathrm{E}(PR_a, \mathrm{H}(M))$
H
$PU_a$
D
Compare
$M$
||
H
E
$PR_a$
E
$K$
(b)
$\mathrm{E}(K, [M \,\|\, \mathrm{E}(PR_a, \mathrm{H}(M))])$
D
$K$
$M$
$\mathrm{E}(PR_a, \mathrm{H}(M))$
H
$PU_a$
D
Compare

# Other Applications

- Hash functions are commonly used to create a one-way password file
- Hash functions can be used for intrusion detection and virus detection

# 3. Simple Hash Functions

- Input (message, file,etc.) is viewed as a sequence of n-bit blocks
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as:



$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

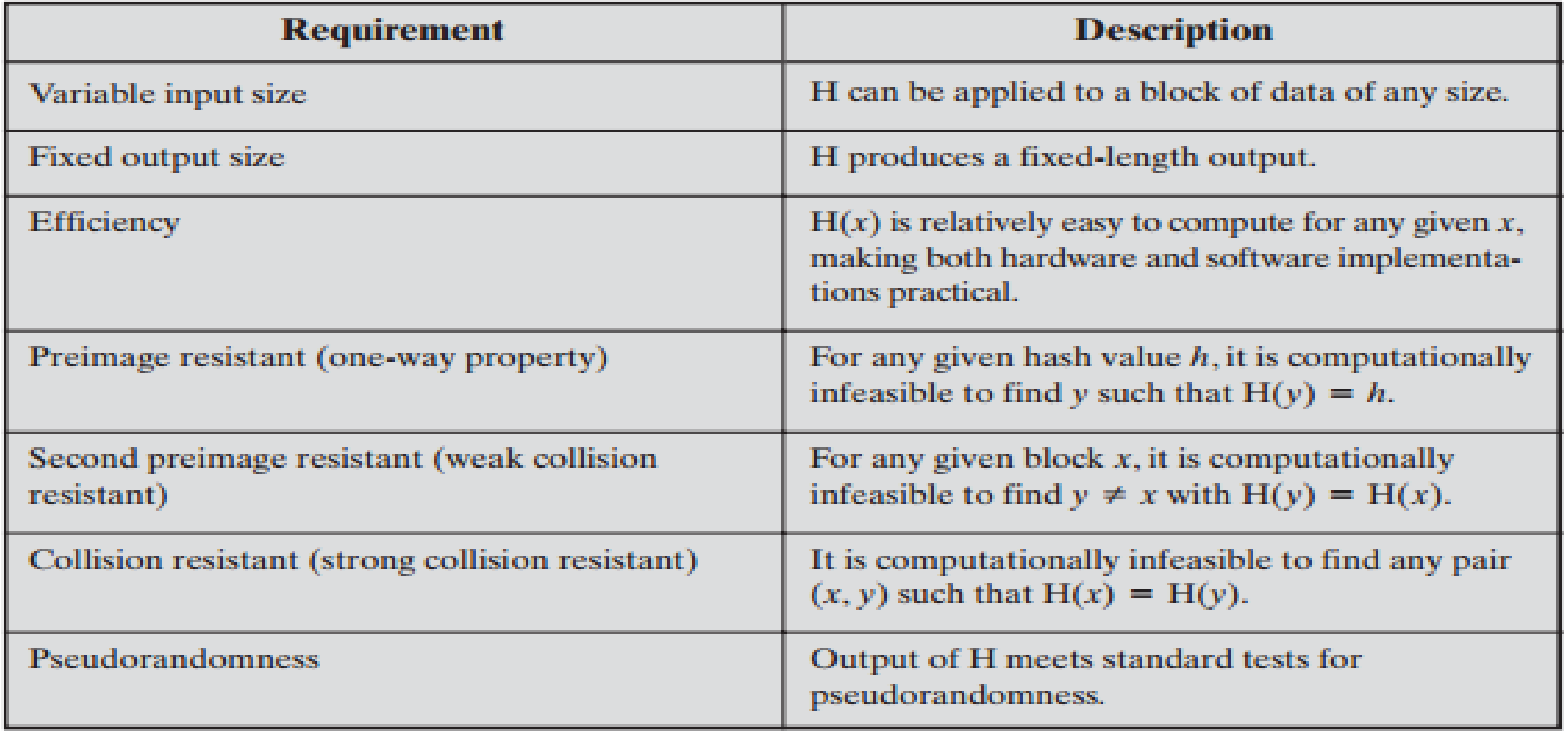
$C_i$ = $i$th bit of the hash code, $1 \leq i \leq$ n

$m$ = number of $n$-bit blocks in the input

$b_{ij}$ = $i$th bit in $j$th block

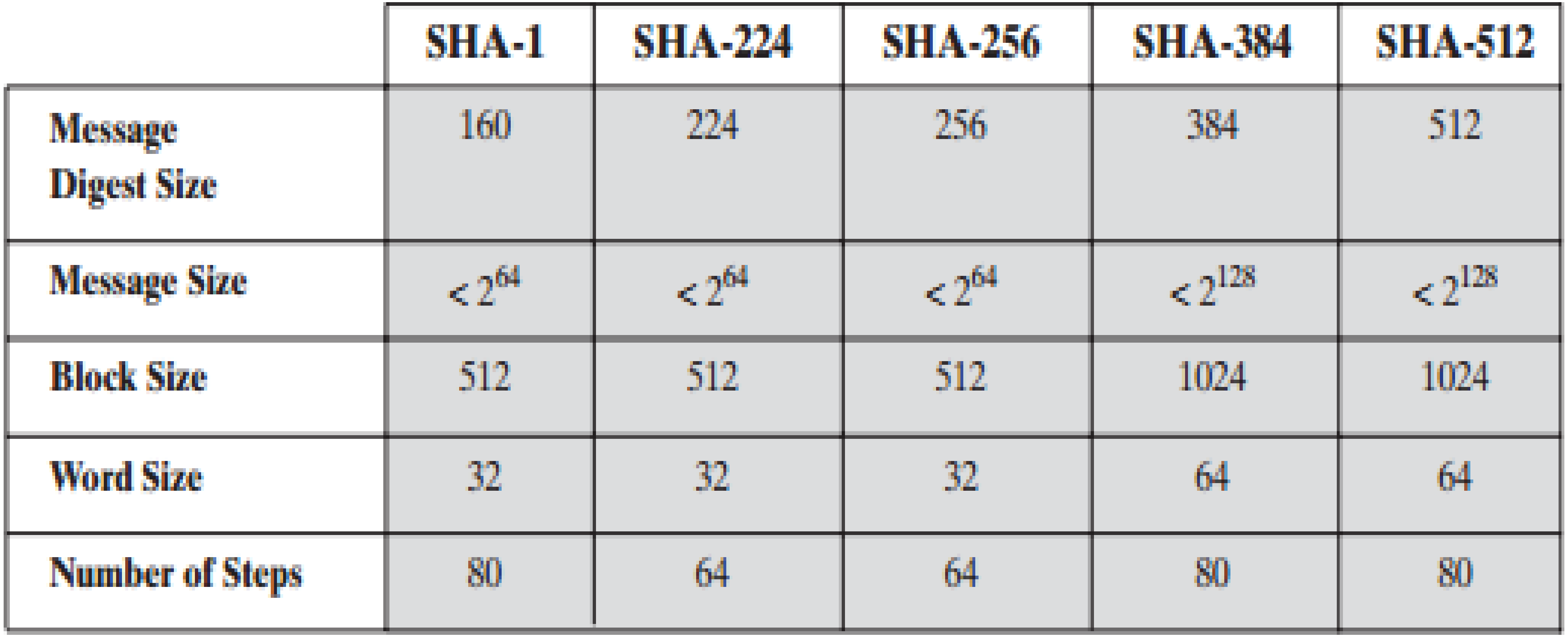
$\oplus$ = XOR operation

# 4. Requirements And Security



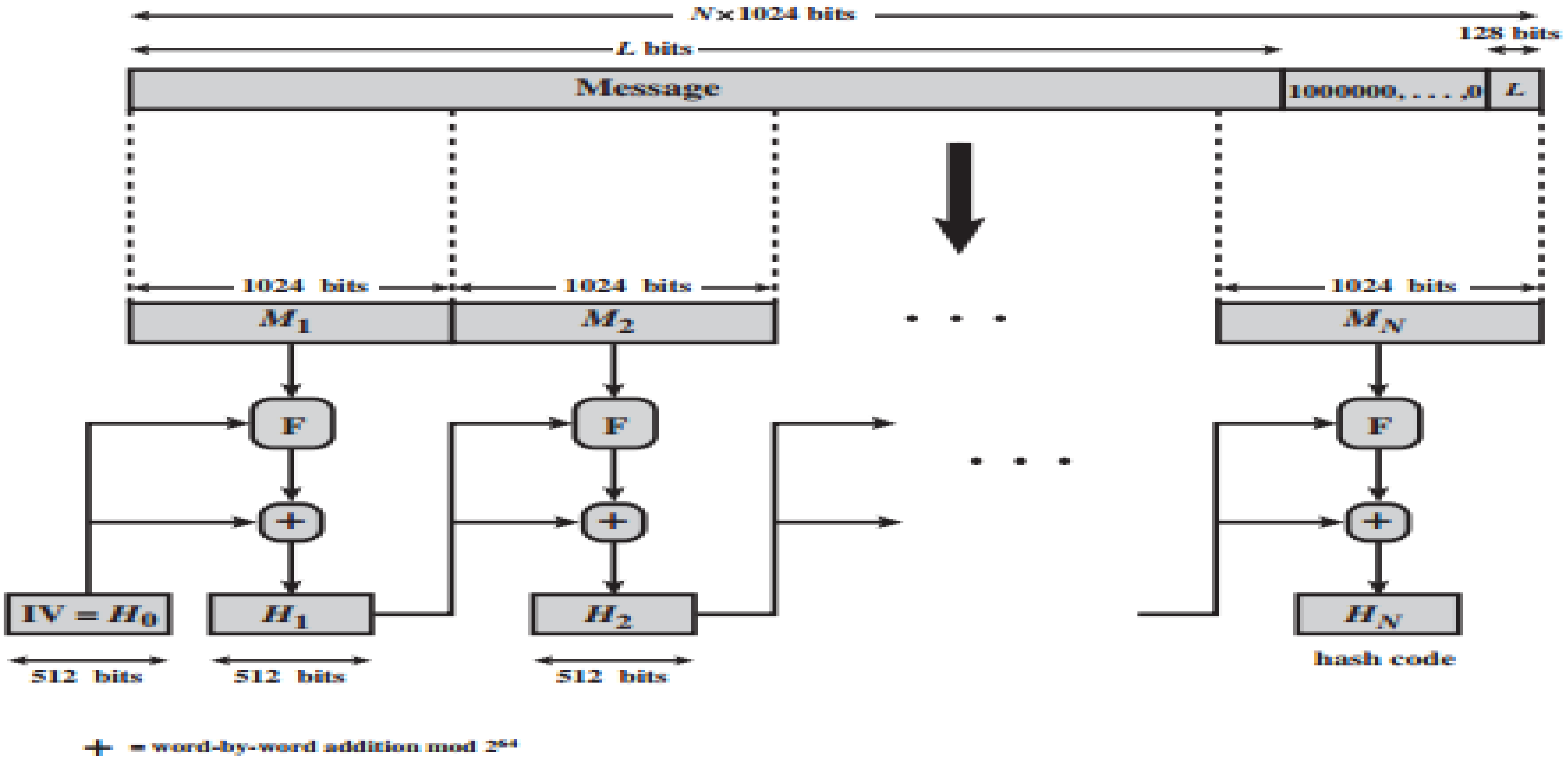
| Requirement | Description |
|---|---|
| Variable input size | H can be applied to a block of data of any size. |
| Fixed output size | H produces a fixed-length output. |
| Efficiency | H($x$) is relatively easy to compute for any given $x$, making both hardware and software implementations practical. |
| 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. |

# 5. Secure Hash Algorithm (SHA)



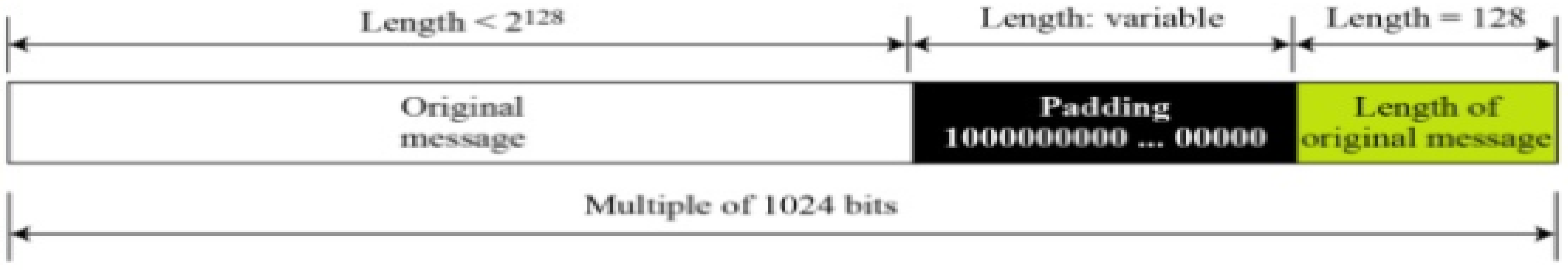
| | SHA-1 | SHA-224 | SHA-256 | SHA-384 | SHA-512 |
|---|---|---|---|---|---|
| **Message Digest Size** | 160 | 224 | 256 | 384 | 512 |
| **Message Size** | $< 2^{64}$ | $< 2^{64}$ | $< 2^{64}$ | $< 2^{128}$ | $< 2^{128}$ |
| **Block Size** | 512 | 512 | 512 | 1024 | 1024 |
| **Word Size** | 32 | 32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

# SHA-512


N × 1024 bits
L bits
128 bits
Message
1000000, . . . ,0
L
1024 bits
1024 bits
1024 bits
$M_1$
$M_2$
$M_N$
F
F
F
+
+
+
IV = $H_0$
$H_1$
$H_2$
$H_N$
512 bits
512 bits
512 bits
hash code
+ = word-by-word addition mod $2^{64}$

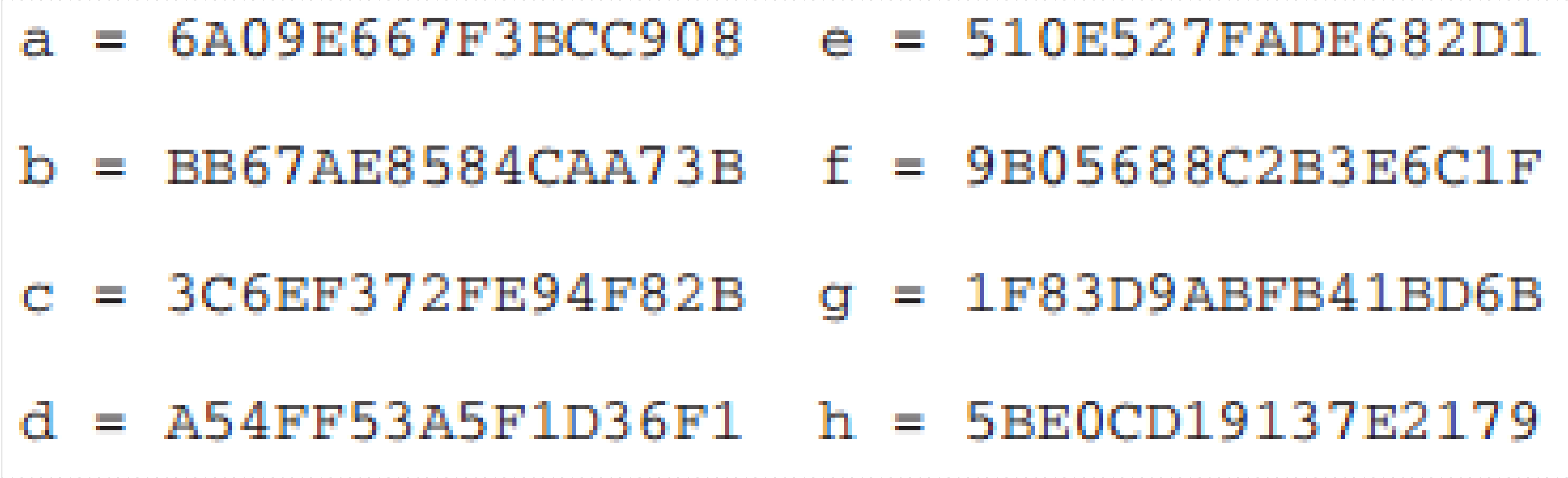
# SHA-512

- **Step 1** Append padding bits: The padding consists of a single 1 bit followed by the necessary number of 0 bits
- **Step 2** Append length: A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer and contains the length of the original message (before the padding).

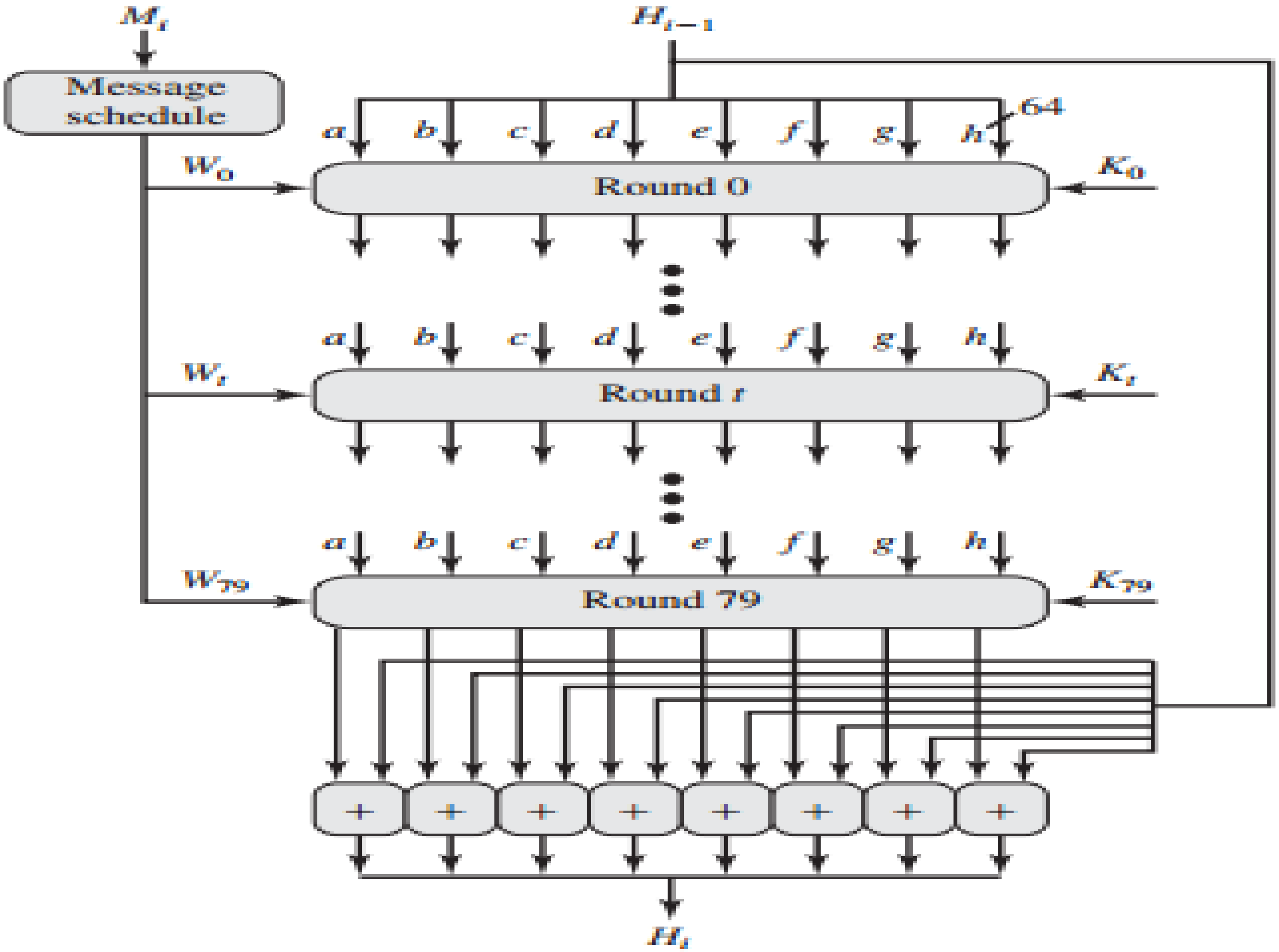


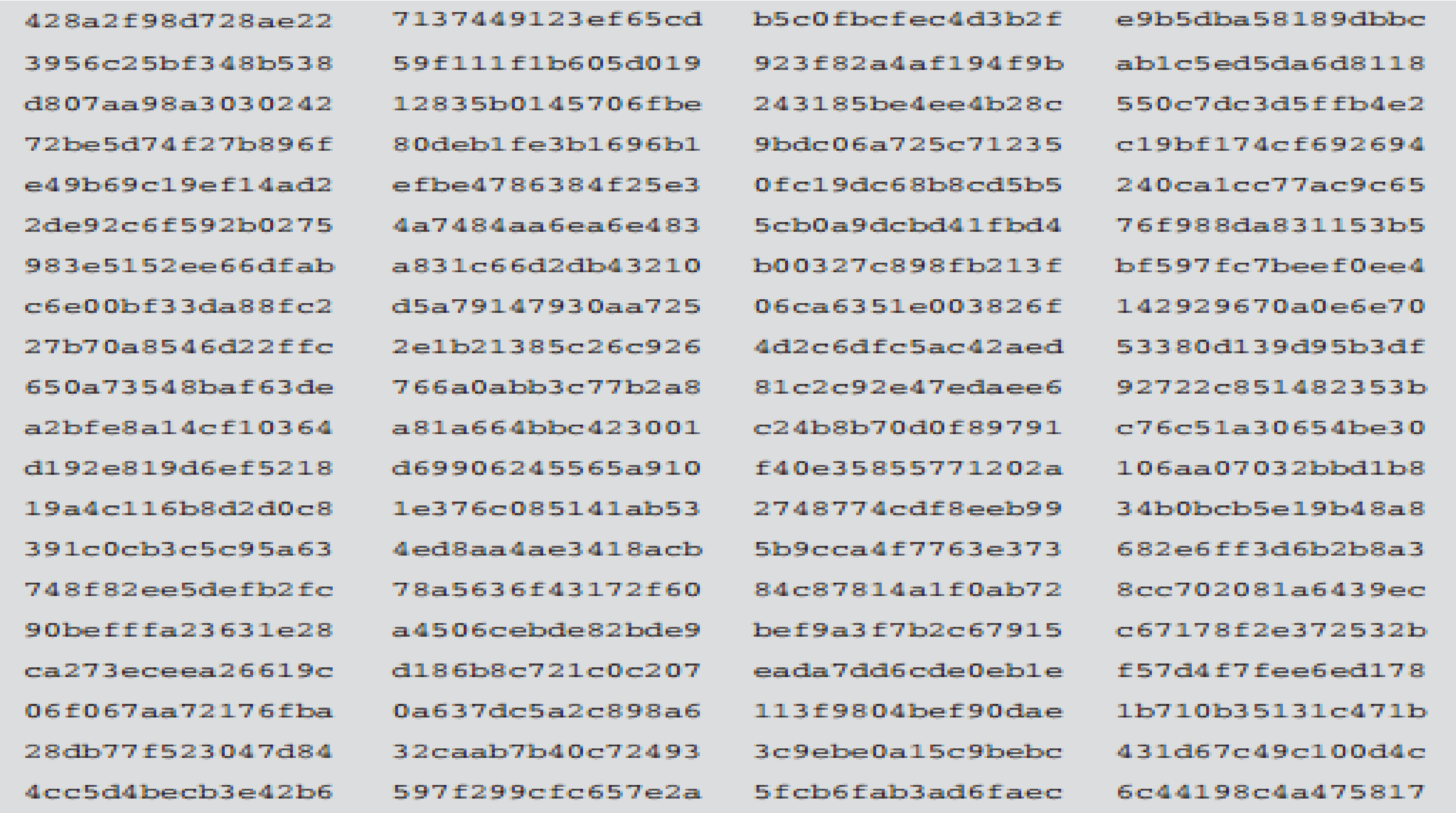
# SHA-512

- **Step 3** 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):



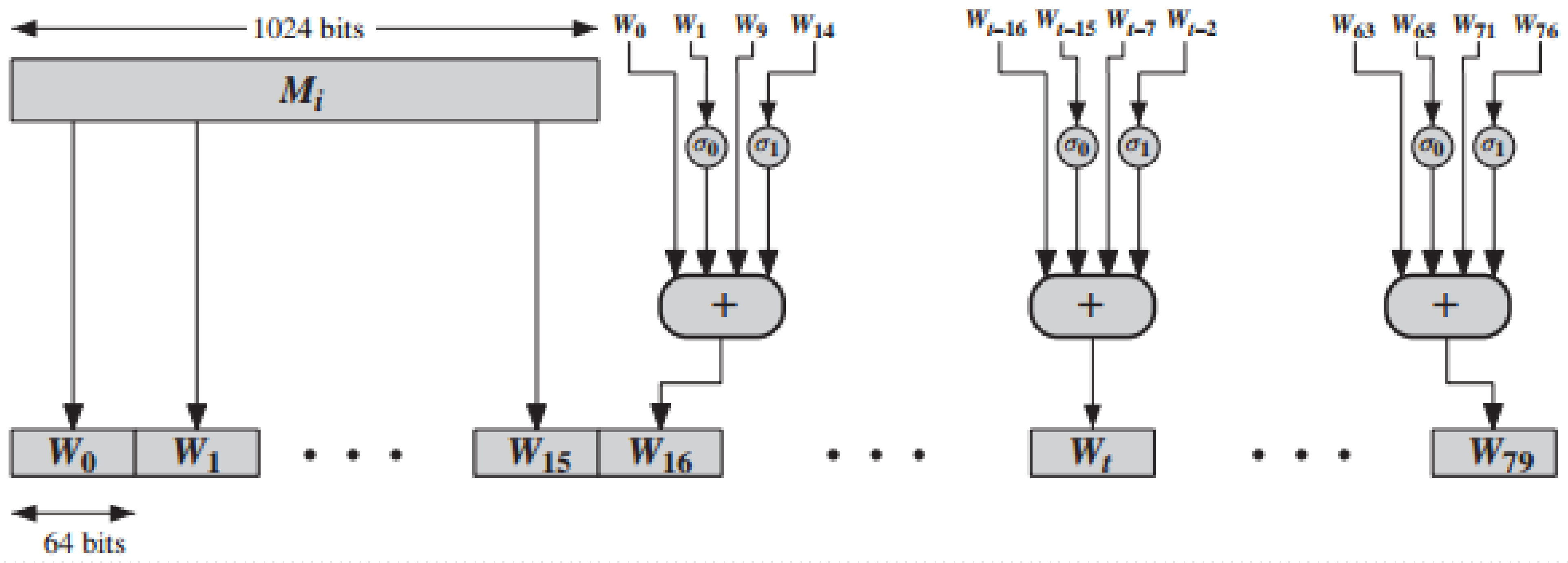
```
a = 6A09E667F3BCC908   e = 510E527FADE682D1

b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F

c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B

d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```


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



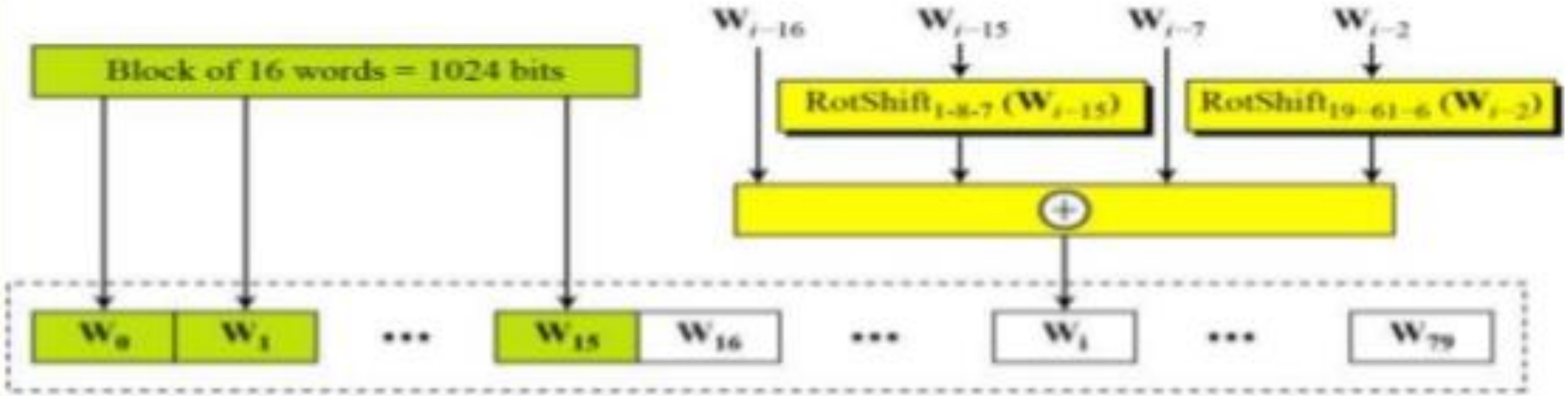
| | | | |
|---|---|---|---|
| 428a2f98d728ae22 | 7137449123ef65cd | b5c0fbcfec4d3b2f | e9b5dba58189dbbc |
| 3956c25bf348b538 | 59f111f1b605d019 | 923f82a4af194f9b | ab1c5ed5da6d8118 |
| d807aa98a3030242 | 12835b0145706fbe | 243185be4ee4b28c | 550c7dc3d5ffb4e2 |
| 72be5d74f27b896f | 80deb1fe3b1696b1 | 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 |

# SHA-512



1024 bits
$W_0$ $W_1$ $W_9$ $W_{14}$
$W_{t-16}$ $W_{t-15}$ $W_{t-7}$ $W_{t-2}$
$W_{63}$ $W_{65}$ $W_{71}$ $W_{76}$
$M_i$
$\sigma_0$
$\sigma_1$
+
$W_0$ $W_1$ • • • $W_{15}$ $W_{16}$ • • • $W_t$ • • • $W_{79}$
64 bits

# SHA-512: Word Expansion


$W_{i-16}$
$W_{i-15}$
$W_{i-7}$
$W_{i-2}$
Block of 16 words = 1024 bits
$\text{RotShift}_{1\text{-}8\text{-}7}(W_{i-15})$
$\text{RotShift}_{19\text{-}61\text{-}6}(W_{i-2})$
⊕
$W_0$
$W_1$
...
$W_{15}$
$W_{16}$
...
$W_i$
...
$W_{79}$

$\text{RotShift}_{l\text{-}m\text{-}n}(x)$: $\text{RotR}_l(x) \oplus \text{RotR}_m(x) \oplus \text{ShL}_n(x)$

$\text{RotR}_i(x)$: Right-rotation of the argument $x$ by $i$ bits
$\text{ShL}_i(x)$: Shift-left of the argument $x$ by $i$ bits and padding the left by 0's.

# SHA-512

- **Step 5** Output: After all 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

# Thanks