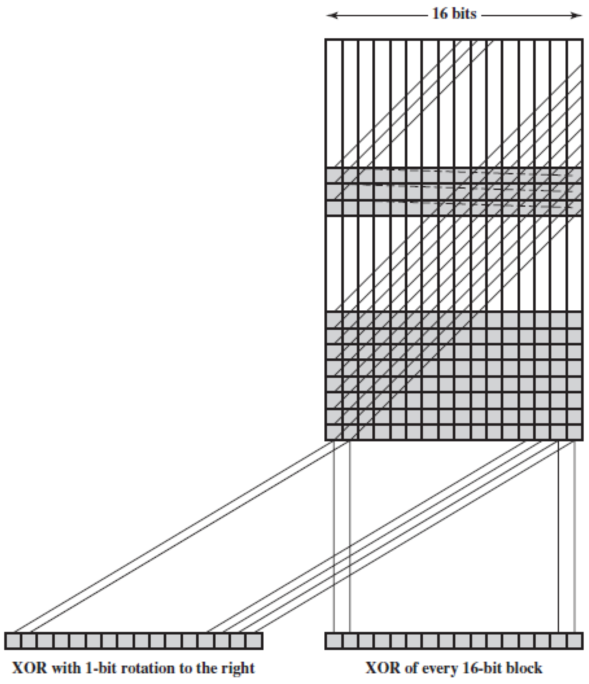
Two Simple Hash Function

The input (message, file, etc.) is viewed as a sequence of *n*-bit blocks. The input is processed one block at a time in an iterative fashion to produce an *n*-bit hash function.
One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as
$C_i = b_{i1} \oplus b_{i2} \oplus \ldots\ldots\ldots. \oplus b_{im}$
where
C_i = *i*th bit of the hash code, 1 … *i* … n
m = number of *n*-bit blocks in the input
b_{ij} = *i*th bit in *j*th block
$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check.
A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.
1. Initially set the *n*-bit hash value to zero.
2. Process each successive *n*-bit block of data as follows:
    a. Rotate the current hash value to the left by one bit.
    b. XOR the block into the hash value.


16 bits
XOR with 1-bit rotation to the right
XOR of every 16-bit block

M consisting of a sequence of 64-bit blocks $X1, X2, \ldots, XN$
$h = H(M)$ as the block-by block XOR of all blocks and append the hash code as the final block:
$h = XN+1 = X1 \oplus X2 \oplus \ldots \oplus XN$
Next, encrypt the entire message plus hash code using CBC mode to produce the encrypted message $Y1, Y2, \ldots, YN+1$.
$X1 = IV \oplus D(K, Y1)$
$Xi = Yi-1 \oplus D(K, Yi)$
$XN+1 = YN \oplus D(K, YN+1)$

But $XN+1$ is the hash code:
$XN+1 = X1 \oplus X2 \oplus \ldots \oplus XN$
$= [IV \oplus D(K, Y1)] \oplus [Y1 \oplus D(K, Y2)] \oplus c \oplus [YN-1 \oplus D(K, YN)]$

Secure Hash Algorithm (SHA)

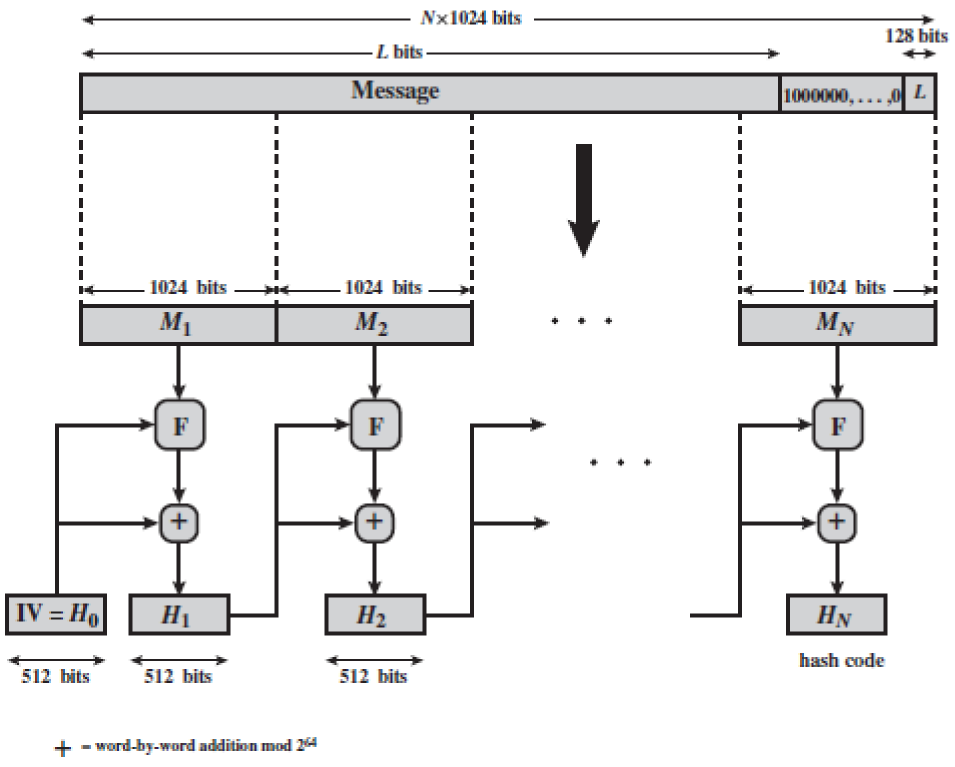
Most widely used hash function has been the Secure Hash Algorithm (SHA)

	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

“Secure Hash Standard.” SHA is based on the hash function MD4

SHA-512 Logic

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks.



Step 1 Append padding bits. The message is padded so that its length is congruent to 896 modulo 1024 [length K 896(mod 1024)]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. 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 (most significant byte first) and contains the length of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure 11.9, the expanded message is represented as the sequence of 1024-bit blocks *M*1, *M*2, c, *MN*, so that the total length of the expanded message is *N* * 1024 bits.

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

These values are stored in **big-endian** format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

Step 4 Process message in 1024-bit (128-word) blocks. The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure.

Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, *Hi*-1. Each round *t* makes use of a 64-bit value *Wt*, derived from the current 1024-bit block being processed (*Mi*). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant *Kt*, where 0 <= *t* <= 79 indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a “randomized” set of 64-bit patterns, which should eliminate any regularities in the input data.

The output of the eightieth round is added to the input to the first round (*Hi*-1) to produce *Hi*. The addition is done independently for each of the eight words in the buffer with each of the corresponding words in *Hi*-1, using addition modulo 2^{64}.

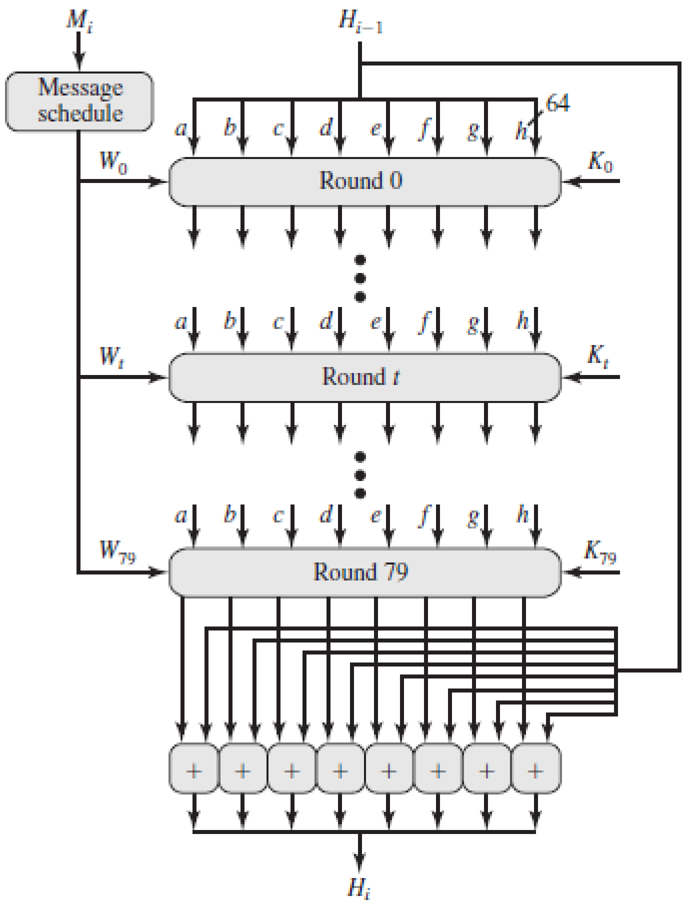


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

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

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
```

We can summarize the behavior of SHA-512 as follows:

*H*0 = IV

Hi = SUM64(*Hi*-1, abcdefgh*i*)

MD = *HN*

where

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

abcdefgh$_i$ = the output of the last round of processing of the *i*th message block

N = the number of blocks in the message (including padding and length fields)

SUM64 = addition modulo 264 performed separately on each word of the pair of inputs

MD = final message digest value

SHA-512 Round Function

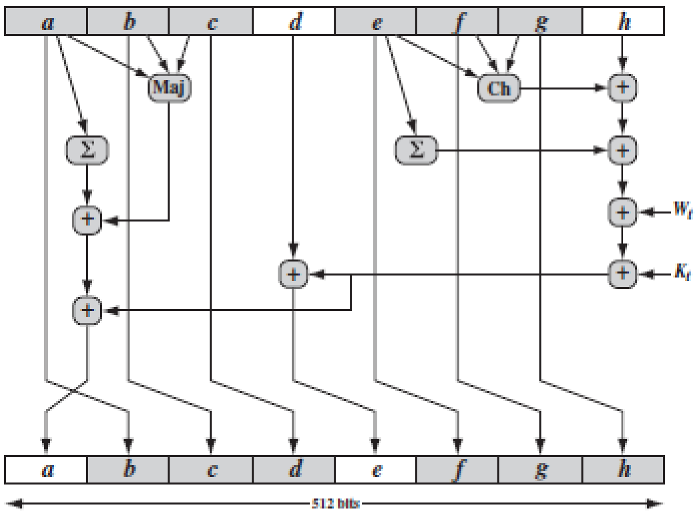


Figure 11.11 Elementary SHA-512 Operation (single round)

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

*T*1 = *h* + Ch(*e*, *f*, *g*) + 1a

512

1 *e*2 + *Wt* + *Kt*

*T*2 = 1a

512

0 *a*2 + 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

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

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

$\mathrm{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
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 1024-bit input block

K_t = a 64-bit additive constant

$+$ = addition modulo 2^{64}

Two observations can be made about the round function.
1. Six of the eight words of the output of the round function involve simply permutation (*b*, *c*, *d*, *f* , *g*, *h*) by means of rotation. This is indicated by shading in Figure 11.11.

2. Only two of the output words (*a*, *e*) are generated by substitution. Word *e* is a function of input variables (*d*, *e*, *f* , *g*, *h*), as well as the round word *Wt* and the constant *Kt*. Word *a* is a function of all of the input variables except *d*, as well as the round word *Wt* and the constant *Kt*.

It remains to indicate how the 64-bit word values *Wt* are derived from the 1024-bit message. Figure 11.12 illustrates the mapping. The first 16 values of *Wt* are taken directly from the 16 words of the current block. The remaining values are defined as

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

where

$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$

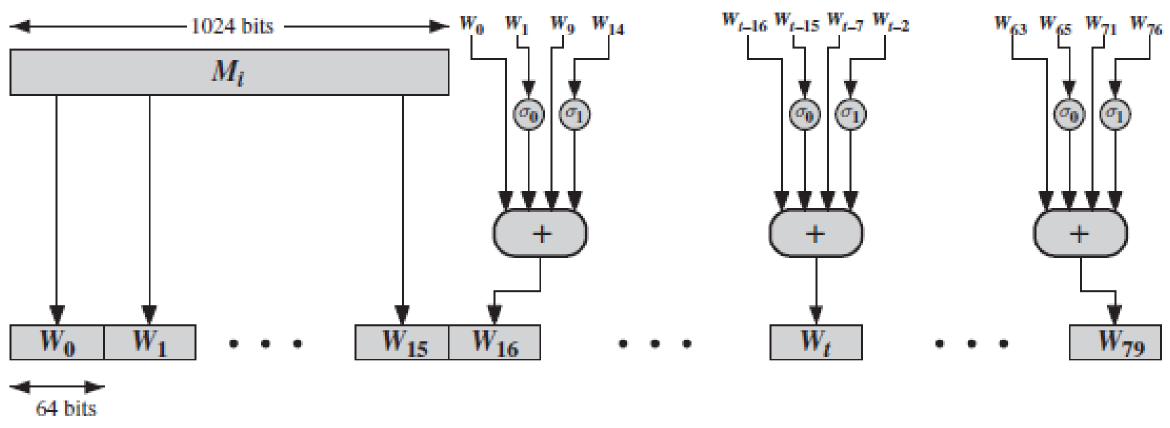
$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$

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

$\mathrm{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}

Thus, in the first 16 steps of processing, the value of *Wt* is equal to the corresponding word in the message block. For the remaining 64 steps, the value of *Wt* consists of the circular left shift by one bit of the XOR of four of the preceding values of *Wt*, with two of those values subjected to shift and rotate operations. This introduces a great deal of redundancy and interdependence into the message blocks that are compressed, which complicates the task of finding a different message block that maps to the same compression function output.



1024 bits
M_i
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}
σ_0 σ_1
+
W_0 W_1 • • • W_{15} W_{16} • • • W_t • • • W_{79}
64 bits

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