21L5294-Laraib Akhtar-35302-3121800-3

Name: Muhammad Laraib Akhtar

Section: BCS-7B

Roll No: 21L-5294

SHA-512 Logic

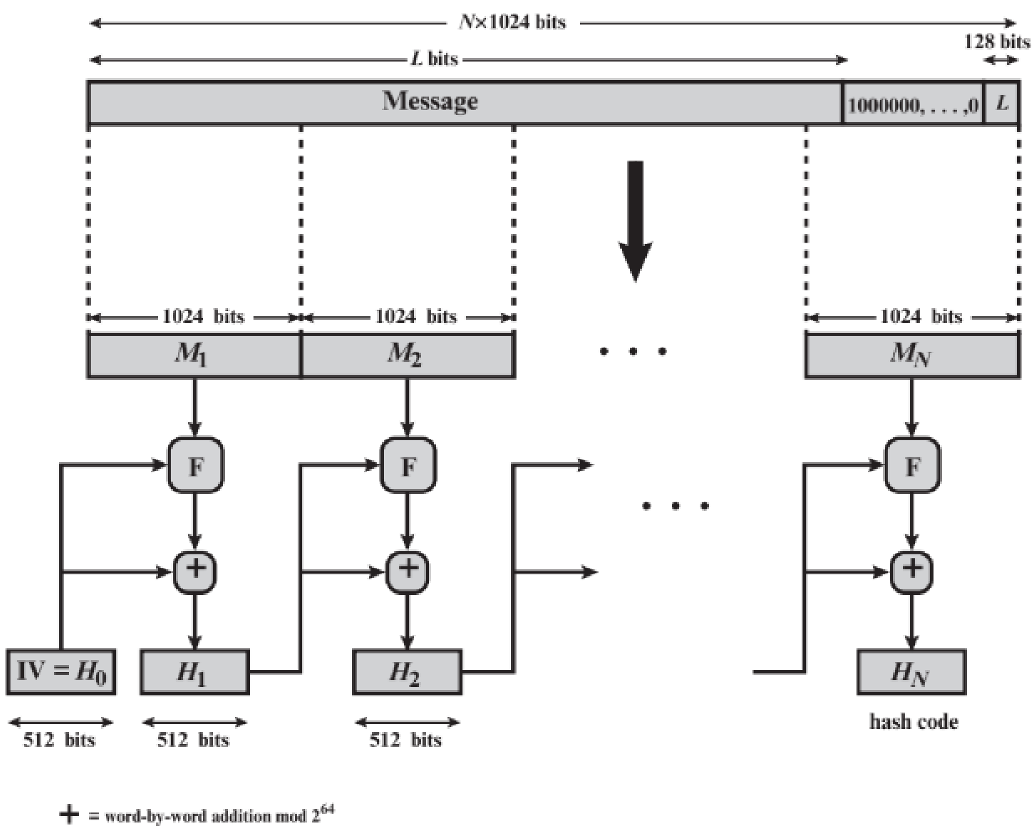

Figure 11.9 Message Digest Generation Using SHA-512

Step1

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 in bits (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 M1, M2, c , MN, so that the total length of the expanded message is N * 1024 bits.

ASCII to string

32314C353239342D 4C61726169622041 6B687461722D3335 3330322D33313231
3830302D33

Length of string before padding: 296 bits

Padding added

32314C353239342D 4C61726169622041 6B687461722D3335 3330322D33313231
3830302D33800000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000128

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-byte) blocks: The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 11.9. The logic is illustrated in Figure 11.10. 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. Table 11.4 shows these constants in hexadecimal format (from left to right).

Note: We only have to implement round 1

K constant: 428a2f98d728ae22

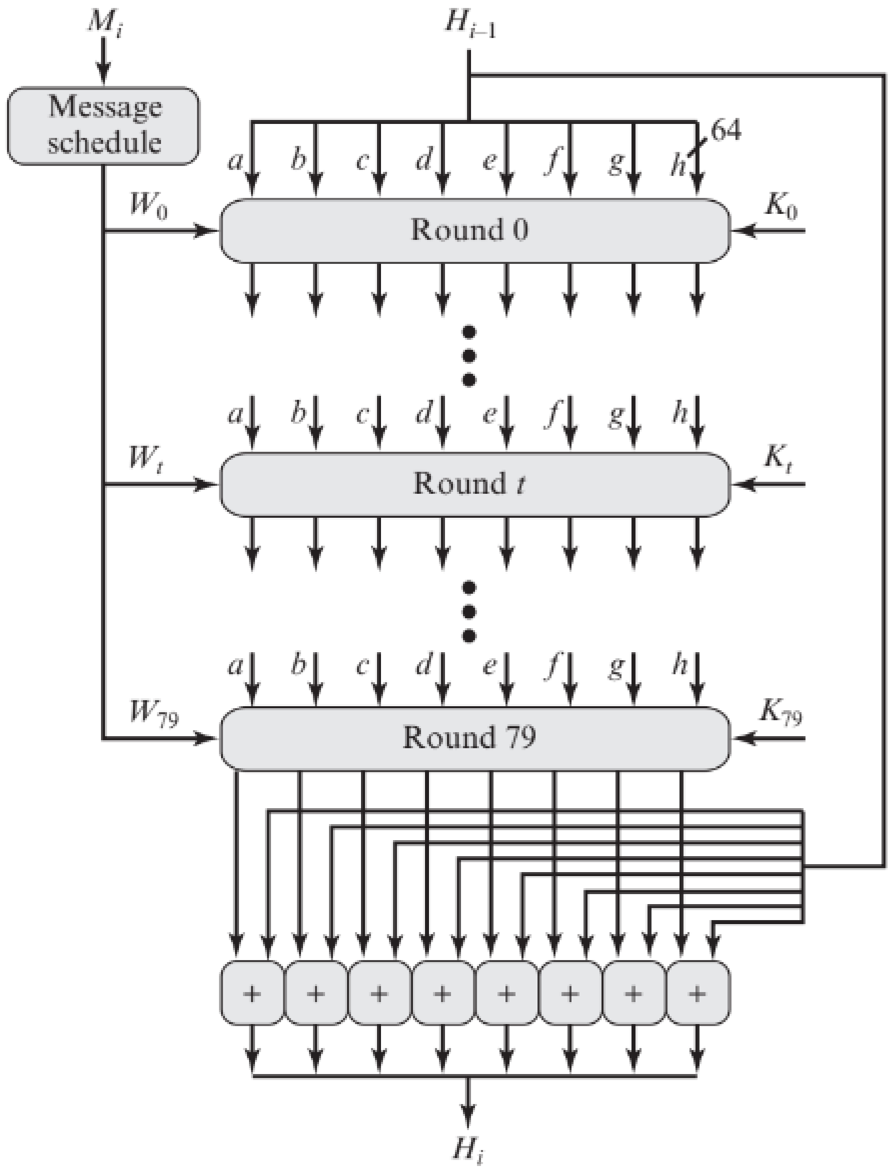

Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

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 11.11). Each round is defined by the following set of equations:

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

where

t	$=$ step number; $0 \le t \le 79$
$\text{Ch}(e, f, g)$	$=$ (e AND f) $\oplus$ (NOT e AND g) *the conditional function: If e then f else g*
$\text{Maj}(a, b, c)$	$=$ (a AND b) $\oplus$ (a AND c) $\oplus$ (b AND c) *the function is true only of the majority (two or three) of the arguments are true*
$(\Sigma_0^{512} a)$	$= \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$
$(\Sigma_1^{512} e)$	$= \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$
$\text{ROTR}^n(x)$	$=$ circular right shift (rotation) of the 64-bit argument x by n bits

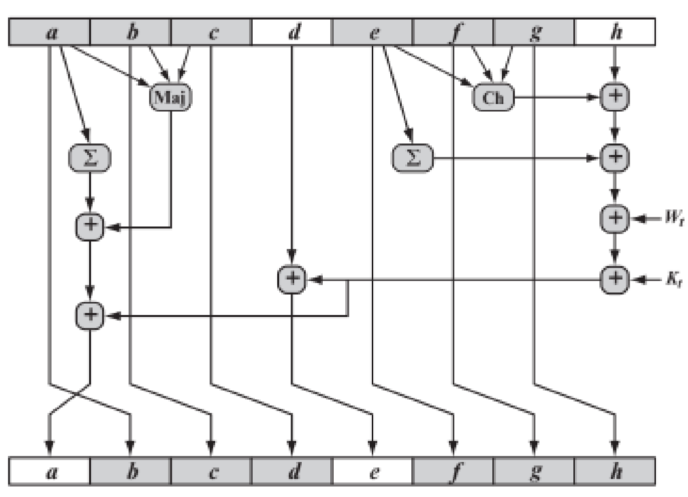

Figure 11.11 Elementary SHA-512 Operation (single round)

NOTE: + is addition modulo 2^64

B = 6A09E667F3BCC908

C = BB67AE8584CAA73B

D = 3C6EF372FE94F82B

F = 510E527FADE682D1

G = 9B05688C2B3E6C1F

H = 1F83D9ABFB41BD6B

Maj(a,b,c) = (a AND b) + (a AND c) + (b AND c)

= 2A01A60580888108 + 2808E262F294C808 + 3866A2008480A02B

= 520a8868731d4910 + 3866A2008480A02B

= 8A712A68F79DE93B

Ch(e,f,g) = (e AND f) + (NOT e AND g)

= 1104400c29260011 + e81898052013d2a

= 1f85c98c7b273d3b

E(e) = ROTR(e,14) + ROTR(e,18) + ROTR(e,41)

= b45443949feb79a + a0b45443949feb79 + a0b45443949feb79

= abf9987cde9ea313 + a0b45443949feb79

= 4cadecc0733e8e8c

E(a) = ROTR(a,28) + ROTR(a,34) + ROTR(a,39)

= 3bcc9086a09e667f + fcef32421a827999 + cfe7799210d413cc

= 38bbc2c8bb20e018 + cfe7799210d413cc

= 8a33c5acbf4f3e4

Wt = 32314C353239342D

Kt = 428a2f98d728ae22

T1 = h + ch(e,f,g) + E(e) + Wt + Kt

= 5BE0CD19137E2179 + 1f85c98c7b273d3b + 4cadecc0733e8e8c + 32314C353239342D + 428a2f98d728ae22

= 7b6696a58ea55eb4 + 4cadecc0733e8e8c + 32314C353239342D + 428a2f98d728ae22

= c814836601e3ed40 + 32314C353239342D + 428a2f98d728ae22

= fa45cf9b341d216d + 428a2f98d728ae22

= 3ccfff340b45cf8f

T2 = E(a) + Maj(a,b,c)

= 8a33c5acbf4f3e4 + 8A712A68F79DE93B

= 931466c3c392dd1f

A = T1 + T2

= 3ccfff340b45cf8f + 931466c3c392dd1f

=cfe465f7ced8acae

E = d + T1

= A54FF53A5F1D36F1 + 3ccfff340b45cf8f

= e21ff46e6a630680

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:

H0 = IV

Hi = SUM64(Hi-1, 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)

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

MD = final message digest value

A0 = 6A09E667F3BCC908 e0 = 510E527FADE682D1

B0 = BB67AE8584CAA73B f0 = 9B05688C2B3E6C1F

C0 = 3C6EF372FE94F82B g0 = 1F83D9ABFB41BD6B

D0 = A54FF53A5F1D36F1 h0 = 5BE0CD19137E2179

A1 = **cfe465f7ced8acae**

B1 = 6A09E667F3BCC908

C1 = BB67AE8584CAA73B

D1 = 3C6EF372FE94F82B

E1 = **e21ff46e6a630680**

F1 = 510E527FADE682D1

G1 = 9B05688C2B3E6C1F

H1 = 1F83D9ABFB41BD6B

Ha = A0+A1

= 6A09E667F3BCC908 + **cfe465f7ced8acae**

39ee4c5fc29575b6

Hb = B0 + B1

= BB67AE8584CAA73B + 6A09E667F3BCC908

=257194ed78877043

Hc = C0 + C1

= 3C6EF372FE94F82B + BB67AE8584CAA73B

= f7d6a1f8835f9f66

Hd = D0 + D1

= A54FF53A5F1D36F1 + 3C6EF372FE94F82B

= e1bee8ad5db22f1c

He = E0 + E1

= 510E527FADE682D1 + **e21ff46e6a630680**

= 332e46ee18498951

Hf = F0 + F1

= 9B05688C2B3E6C1F + 510E527FADE682D1

= ec13bb0bd924eef0

Hg = G0 + G1

= 1F83D9ABFB41BD6B + 9B05688C2B3E6C1F

= ba8942382680298a

Hh = H0 + H1

= 5BE0CD19137E2179 + 1F83D9ABFB41BD6B

= 7b64a6c50ebfdee4

HASH CODE =

39ee4c5fc29575b6 257194ed78877043 f7d6a1f8835f9f66 e1bee8ad5db22f1c

332e46ee18498951 ec13bb0bd924eef0 ba8942382680298a 7b64a6c50ebfdee4