Secure Hash Algorithm

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1.

The actual standards document is entitled Secure Hash Standard. SHA is based on the hash function MD4 and its design closely models MD4. SHA-1 is also specified in RFC 3174, which essentially duplicates the material in FIPS 180-1, but adds a C code implementation.SHA-1 produces a hash value of 160 bits.

In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.

In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

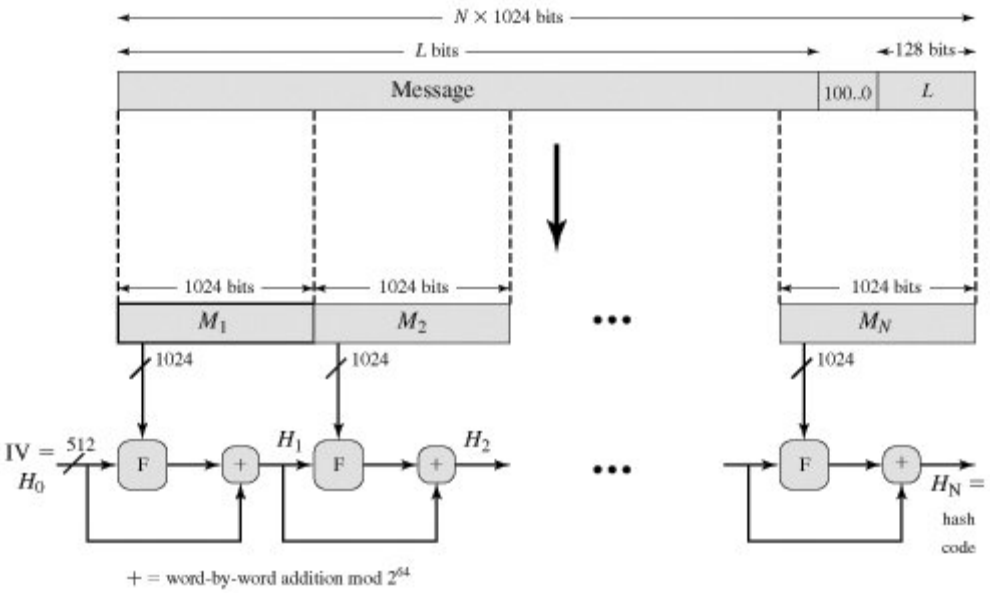
Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^{69} operations, far fewer than the 2^{80} operations previously thought needed to find a collision with an SHA-1 hash. This result should hasten the transition to the other versions of SHA.

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. Figure 12.1 depicts the overall processing of a message to produce a digest.

Figure 12.1
Message Digest Generation Using SHA-512



The processing consists of the following steps:

Step 1: Append padding bits.

The message is padded so that its length is congruent to 896 modulo 1024 [length 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 12.1, the expanded message is represented as the sequence of 1024-bit blocks *M*1, *M*2,..., *M*N, so that the total length of the expanded message is *N* x 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

b = BB67AE8584CAA73B

c = 3C6EF372FE94F82B

d = A54FF53A5F1D36F1

e = 510E527FADE682D1

f = 9B05688C2B3E6C1F

g = 1F83D9ABFB41BD6B

h = 5BE0CDI9137E2179

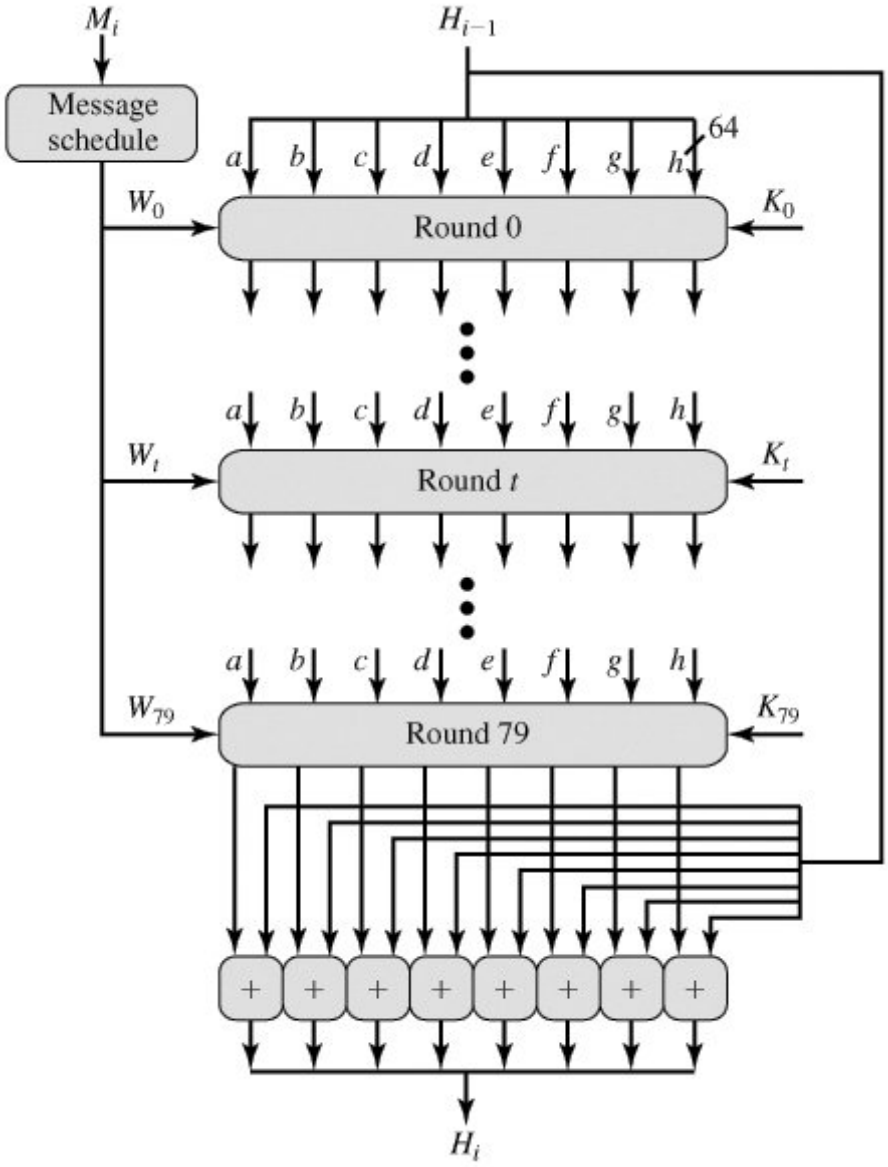
These values are stored in big-endianformat, which is the most significant byte of a word in the low-address (leftmost) byte position.

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 12.1. The logic is illustrated in Figure 12.2.

Figure 12.2
SHA-512 Processing of a Single 1024-Bit Block



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 sixty-four bits of the fractional parts of the cube roots of the first eighty 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 264.

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.

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

Let us look in more detail at the logic in each of the 80 steps of the processing of one 512-bit block
(Figure 12.3). Each round is defined by the following set of equations:

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a\right) + \text{Maj}(a, b, c)$$

$$a = T_1 + T_2$$

$$b = a$$

$$c = b$$

$$d = c$$

$$e = d + T_1$$

$$f = e$$

$$g = f$$

$$h = g$$

where
t=step number; 0 <=t <=79

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

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.*

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

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

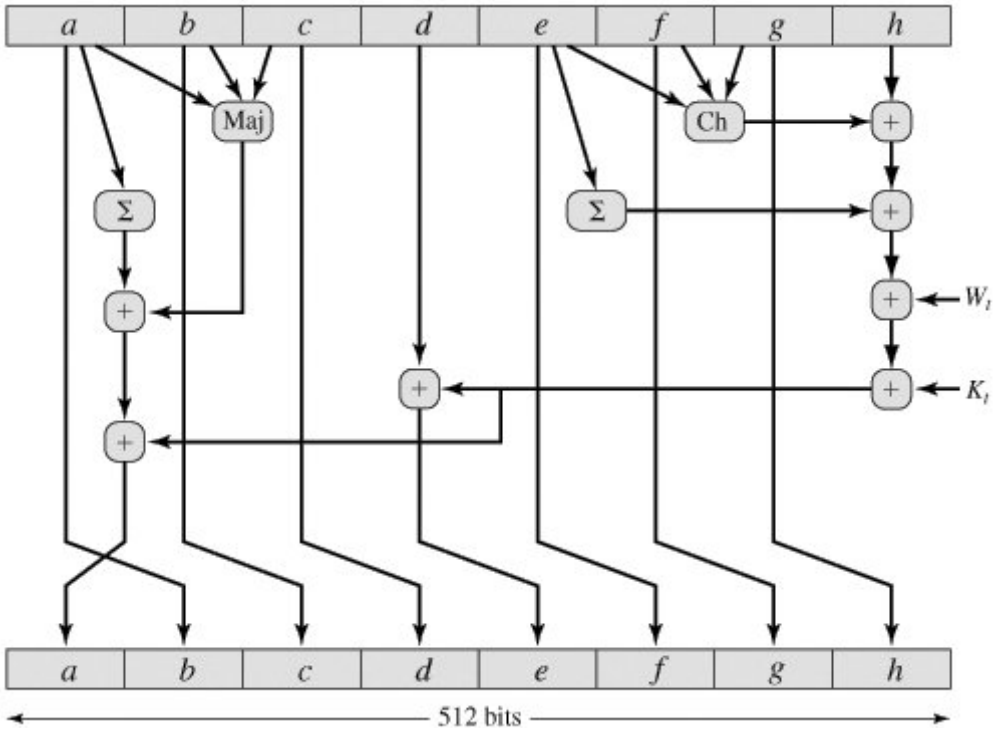
$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

Wt = a 64-bit word derived from the current 512-bit input block

Kt = a 64-bit additive constant

+ = addition modulo 2^{64}

Figure 12.3
Elementary SHA-512 Operation (single round)



It remains to indicate how the 64-bit word values *Wt* are derived from the 1024-bit message. Figure12.4 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 follows:

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

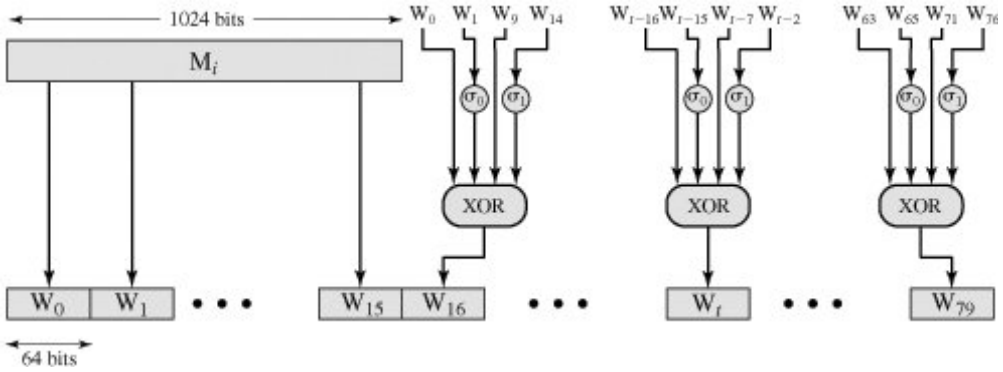
Where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

ROTR$n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

SHR$n(x)$ = left shift of the 64-bit argument x by n bits with padding *by zeros on the right*

Figure 12.4
Creation of 80-word Input Sequence for SHA-512 Processing of Single Block



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.

**********STAY HOME, BE SAFE**********