This is similar to the CBC technique, but in this case, there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings. Here is the scenario: We assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long.

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
2. Construct any desired message in the form $Q_1, Q_2, \ldots, Q_{N-2}$.
3. Compute $H_i = \mathrm{E}(Q_i, H_{i-1})$ for $1 \leq i \leq (N - 2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $\mathrm{E}(X, H_{N-2})$. Generate an additional $2^{m/2}$ random blocks; for each block Y, compute $\mathrm{D}(Y, G)$, where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $\mathrm{E}(X, H_{N-2}) = \mathrm{D}(Y, G)$.
6. Form the message $Q_1, Q_2, \ldots, Q_{N-2}, X, Y$. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a **meet-in-the-middle-attack**. A number of researchers have proposed refinements intended to strengthen the basic block chaining approach. For example, Davies and Price [DAVI89] describe the variation:

$$H_i = \mathrm{E}(M_i, \mathrm{H}_{i-1}) \oplus H_{i-1}$$

Another variation, proposed in [MEYE88], is

$$H_i = \mathrm{E}(H_{i-1}, M_i) \oplus M_i$$

However, both of these schemes have been shown to be vulnerable to a variety of attacks [MIYA90]. More generally, it can be shown that some form of birthday attack will succeed against any hash scheme involving the use of cipher block chaining without a secret key, provided that either the resulting hash code is small enough (e.g., 64 bits or less) or that a larger hash code can be decomposed into independent subcodes [JUEN87].

Thus, attention has been directed at finding other approaches to hashing. Many of these have also been shown to have weaknesses [MITC92].

11.5 SECURE HASH ALGORITHM (SHA)

In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). Indeed, because virtually every other widely used hash function had been found to have substantial cryptanalytic weaknesses, SHA was more or less the last remaining standardized hash algorithm by 2005. SHA was developed

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.

by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. When weaknesses were discovered in SHA, now known as **SHA-0**, a revised version was issued as FIPS 180-1 in 1995 and is 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 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, respectively. Collectively, these hash algorithms are known as **SHA-2**. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. A revised document was issued as FIP PUB 180-3 in 2008, which added a 224-bit version (Table 11.3). In 2015, NIST issued FIPS 180-4, which added two additional algorithms: SHA-512/224 and SHA-512/256. SHA-1 and SHA-2 are also specified in RFC 6234, which essentially duplicates the material in FIPS 180-3 but adds a C code implementation.

In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on SHA-2 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 [WANG05]. This result should hasten the transition to SHA-2.

In this section, we provide a description of SHA-512. The other versions are quite similar.

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 11.9 depicts the overall processing of a message to produce a digest.

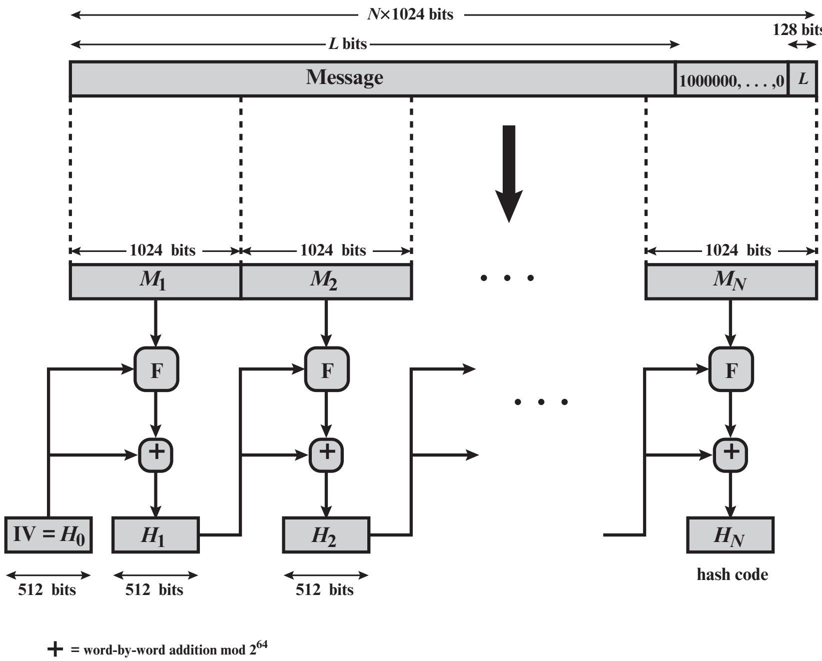

Figure 11.9 Message Digest Generation Using SHA-512

This follows the general structure depicted in Figure 11.8. 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 $\equiv$ 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 $M_1, M_2, \ldots, M_N$, so that the total length of the expanded message is $N \times 1024$ 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-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, H_{i-1}. Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ 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).

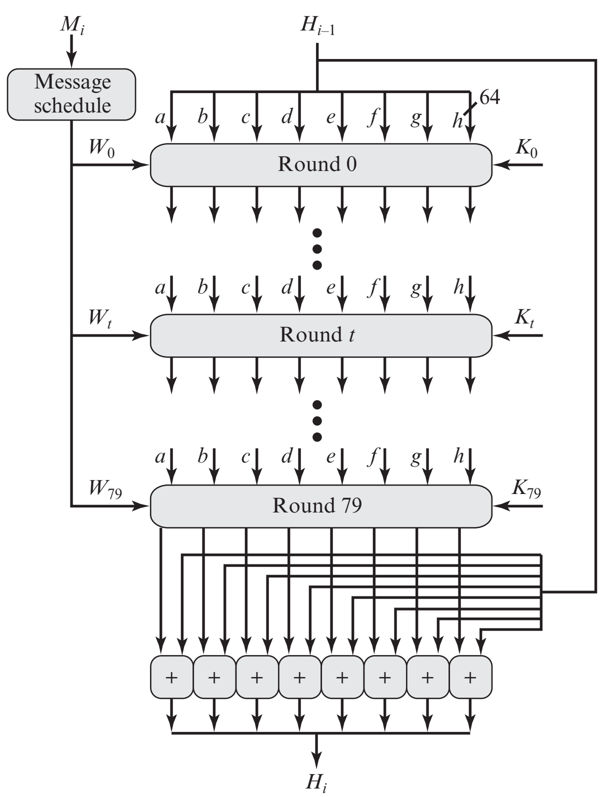

Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

Table 11.4 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

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

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:

$$H_0 = \mathrm{IV}$$
$$H_i = \mathrm{SUM}_{64}(H_{i-1}, \mathrm{abcdefgh}_i)$$
$$MD = H_N$$

where

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

$\mathrm{abcdefgh}_i$ = 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)

SUM_{64} = addition modulo 2^{64} 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 11.11). Each round is defined by the following set of equations:

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum\nolimits_1^{512} e\right) + W_t + K_t$$
$$T_2 = \left(\sum\nolimits_0^{512} a\right) + \text{Maj}(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

where

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

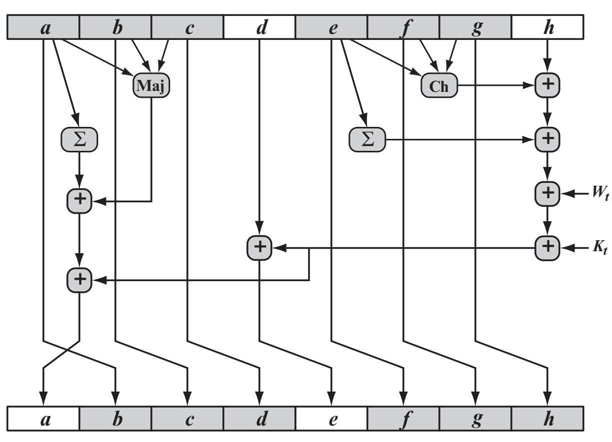

Figure 11.11 Elementary SHA-512 Operation (single round)

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 W_t and the constant K_t. Word a is a function of all of the input variables except d, as well as the round word W_t and the constant K_t.

It remains to indicate how the 64-bit word values W_t are derived from the 1024-bit message. Figure 11.12 illustrates the mapping. The first 16 values of W_t 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

$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$
$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$
$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits
$\text{SHR}^n(x)$ = right shift of the 64-bit argument x by n bits with padding by zeros on the left
$+$ = addition modulo 2^{64}

Thus, in the first 16 steps of processing, the value of W_t is equal to the corresponding word in the message block. For the remaining 64 steps, the value of W_t consists of the circular left shift by one bit of the XOR of four of the preceding values of W_t, with two of those values subjected to shift and rotate operations.

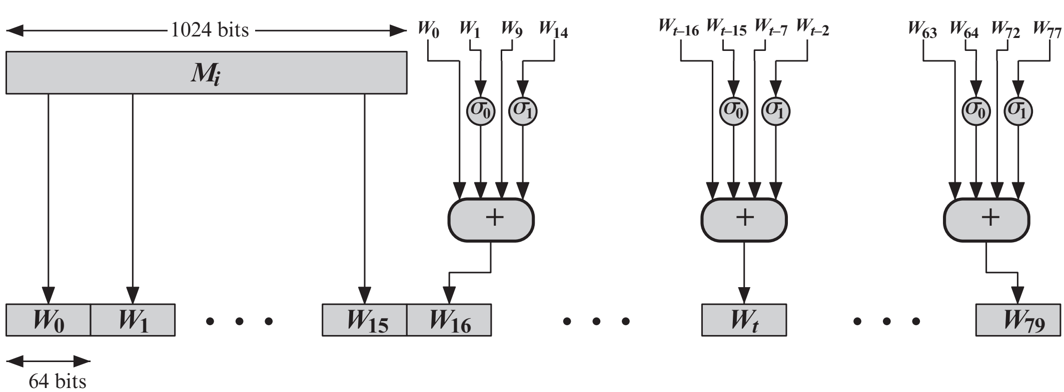

Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

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. Figure 11.13 summarizes the SHA-512 logic.

The SHA-512 algorithm has the property that every bit of the hash code is a function of every bit of the input. The complex repetition of the basic function F produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA-512, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of 2^{256} operations, while the difficulty of finding a message with a given digest is on the order of 2^{512} operations.

Example

We include here an example based on one in FIPS 180. We wish to hash a one-block message consisting of three ASCII characters: “abc,” which is equivalent to the following 24-bit binary string:

```
01100001 01100010 01100011
```

Recall from step 1 of the SHA algorithm, that the message is padded to a length congruent to 896 modulo 1024. In this case of a single block, the padding consists of $896 - 24 = 872$ bits, consisting of a “1” bit followed by 871 “0” bits. Then a 128-bit length value is appended to the message, which contains the length of the original message in bits (before the padding). The original length is 24 bits, or a hexadecimal value of 18. Putting this all together, the 1024-bit message block, in hexadecimal, is

```
6162638000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000018
```

This block is assigned to the words W0, . . . , W15 of the message schedule, which appears as follows.

$$
\begin{array}{ll}
W_0 = 6162638000000000 & W_8 = 0000000000000000 \\
W_1 = 0000000000000000 & W_9 = 0000000000000000 \\
W_2 = 0000000000000000 & W_{10} = 0000000000000000 \\
W_3 = 0000000000000000 & W_{11} = 0000000000000000 \\
W_4 = 0000000000000000 & W_{12} = 0000000000000000 \\
W_5 = 0000000000000000 & W_{13} = 0000000000000000 \\
W_6 = 0000000000000000 & W_{14} = 0000000000000000 \\
W_7 = 0000000000000000 & W_{15} = 0000000000000018
\end{array}
$$

The padded message consists blocks $M_1, M_2, \ldots, M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1}, \ldots, M_{i,15}$. All addition is performed modulo 2^{64}.

$$
\begin{array}{ll}
H_{0,0} = \text{6A09E667F3BCC908} & H_{0,4} = \text{510E527FADE682D1} \\
H_{0,1} = \text{BB67AE8584CAA73B} & H_{0,5} = \text{9B05688C2B3E6C1F} \\
H_{0,2} = \text{3C6EF372FE94F82B} & H_{0,6} = \text{1F83D9ABFB41BD6B} \\
H_{0,3} = \text{A54FF53A5F1D36F1} & H_{0,7} = \text{5BE0CD19137E2179}
\end{array}
$$

for $i = 1$ **to** N

1. Prepare the message schedule W
   for $t = 0$ **to** 15
   $W_t = M_{i,t}$
   for $t = 16$ **to** 79
   $W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$
2. Initialize the working variables

$$
\begin{array}{ll}
a = H_{i-1,0} & e = H_{i-1,4} \\
b = H_{i-1,1} & f = H_{i-1,5} \\
c = H_{i-1,2} & g = H_{i-1,6} \\
d = H_{i-1,3} & h = H_{i-1,7}
\end{array}
$$

3. Perform the main hash computation
   for $t = 0$ **to** 79

$$
\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\Sigma_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\Sigma_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

4. Compute the intermediate hash value

$$
\begin{array}{ll}
H_{i,0} = a + H_{i-1,0} & H_{i,4} = e + H_{i-1,4} \\
H_{i,1} = b + H_{i-1,1} & H_{i,5} = f + H_{i-1,5} \\
H_{i,2} = c + H_{i-1,2} & H_{i,6} = g + H_{i-1,6} \\
H_{i,3} = d + H_{i-1,3} & H_{i,7} = h + H_{i-1,7}
\end{array}
$$

return $\{H_{N,0} \| H_{N,1} \| H_{N,2} \| H_{N,3} \| H_{N,4} \| H_{N,5} \| H_{N,6} \| H_{N,7}\}$

Figure 11.13 SHA-512 Logic

As indicated in Figure 11.13, the eight 64-bit variables, a through h, are initialized to values $H_{0,0}$ through $H_{0,7}$. The following table shows the initial values of these variables and their values after each of the first two rounds.

a	6a09e667f3bcc908	f6afceb8bcfcddf5	1320f8c9fb872cc0
b	bb67ae8584caa73b	6a09e667f3bcc908	f6afceb8bcfcddf5
c	3c6ef372fe94f82b	bb67ae8584caa73b	6a09e667f3bcc908
d	a54ff53a5f1d36f1	3c6ef372fe94f82b	bb67ae8584caa73b
e	510e527fade682d1	58cb02347ab51f91	c3d4ebfd48650ffa
f	9b05688c2b3e6c1f	510e527fade682d1	58cb02347ab51f91
g	1f83d9abfb41bd6b	9b05688c2b3e6c1f	510e527fade682d1
h	5be0cd19137e2179	1f83d9abfb41bd6b	9b05688c2b3e6c1f

Note that in each of the rounds, six of the variables are copied directly from variables from the preceding round.

The process continues through 80 rounds. The output of the final round is

```
73a54f399fa4b1b2  10d9c4c4295599f6  d67806db8b148677  654ef9abec389ca9
d08446aa79693ed7  9bb4d39778c07f9e  25c96a7768fb2aa3  ceb9fc3691ce8326
```

The hash value is then calculated as

$$
\begin{aligned}
H_{1,0} &= \texttt{6a09e667f3bcc908} + \texttt{73a54f399fa4b1b2} = \texttt{ddaf35a193617aba}\\
H_{1,1} &= \texttt{bb67ae8584caa73b} + \texttt{10d9c4c4295599f6} = \texttt{cc417349ae204131}\\
H_{1,2} &= \texttt{3c6ef372fe94f82b} + \texttt{d67806db8b148677} = \texttt{12e6fa4e89a97ea2}\\
H_{1,3} &= \texttt{a54ff53a5f1d36f1} + \texttt{654ef9abec389ca9} = \texttt{0a9eeee64b55d39a}\\
H_{1,4} &= \texttt{510e527fade682d1} + \texttt{d08446aa79693ed7} = \texttt{2192992a274fc1a8}\\
H_{1,5} &= \texttt{9b05688c2b3e6c1f} + \texttt{9bb4d39778c07f9e} = \texttt{36ba3c23a3feebbd}\\
H_{1,6} &= \texttt{1f83d9abfb41bd6b} + \texttt{25c96a7768fb2aa3} = \texttt{454d4423643ce80e}\\
H_{1,7} &= \texttt{5be0cd19137e2179} + \texttt{ceb9fc3691ce8326} = \texttt{2a9ac94fa54ca49f}
\end{aligned}
$$

The resulting 512-bit message digest is

```
ddaf35a193617aba  cc417349ae204131  12e6fa4e89a97ea2  0a9eeee64b55d39a
2192992a274fc1a8  36ba3c23a3feebbd  454d4423643ce80e  2a9ac94fa54ca49f
```

Suppose now that we change the input message by one bit, from "abc" to "cbc." Then, the 1024-bit message block is

```
6362638000000000  0000000000000000  0000000000000000  0000000000000000
0000000000000000  0000000000000000  0000000000000000  0000000000000000
0000000000000000  0000000000000000  0000000000000000  0000000000000000
0000000000000000  0000000000000000  0000000000000000  0000000000000018
```

And the resulting 512-bit message digest is

```
531668966ee79b70  0b8e593261101354  4273f7ef7b31f279  2a7ef68d53f93264
319c165ad96d9187  55e6a204c2607e27  6e05cdf993a64c85  ef9e1e125c0f925f
```

The number of bit positions that differ between the two hash values is 253, almost exactly half the bit positions, indicating that SHA-512 has a good avalanche effect.