analysis operations.
However, there has been an ominous trend in the attacks on MD5:

1. Berson [BERS92] showed, using differential cryptanalysis, that it is possible in reasonable time to find two messages that produce the same digest for a single-round MD5. The result was demonstrated for each of the four rounds. However, the author has not been able to show how to generalize the attack to the full four-round MD5.
2. Boer and Bosselaers [BOER93] showed how to find a message block X and two related chaining variables that yield the same output state. That is, execution of MD5 on a single block of 512 bits will yield the same output for two different input values in buffer ABCD. This is referred to as a *pseudocollision.* At present, there does not seem to be any way to extend this approach to a successful attack on MD5.
3. The most serious attack on MD5 has been developed by Dobbertin [DOBB96a]. His technique enables the generation of a collision for the MD5 compression function; that is, the attack works on the operation of MD5 on a single 512-bit block of input by finding another block that produces the same 128-bit output. As of this writing, no way has been found to generalize this attack to a full message using the MD5 initial value (IV). Nevertheless, the success of this attack is too close for comfort.

Thus we see that from a cryptanalytic point of view, MD5 must now be considered vulnerable. Further, from the point of view of brute-force attack, MD5 is now vulnerable to birthday attacks that require on the order of effort of 2^{64}. As a result, there was a need to replace the popular MD5 with a hash function that has a longer hash code and is more resistant to known methods of cryptanalysis. Two alternatives are popular: SHA-1 and RIPEMD-160. These are examined in the next two sections.

2 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 MD4 algorithm 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 Logic

The algorithm takes as input a message with a maximum length of less than 2^{64} bits and produces as output a 160-bit message digest. The input is processed in 512-bit blocks.

The overall processing of a message follows the structure shown for MD5 in Figure 12.1, with a block length of 512 bits and a hash length and chaining variable length of 160 bits. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length is congruent to 448 modulo 512 (length ≡ 448 mod 512). 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 512. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length.** A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before the padding).
- **Step 3: Initialize MD buffer.** A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A, B, C, D, E). These registers are initialized to the following 32-bit integers (hexadecimal values):

  A = 67452301
  B = EFCDAB89
  C = 98BADCFE
  D = 10325476
  E = C3D2E1F0

  Note that the first four values are the same as those used in MD5. However, in the case of SHA-1, these values are stored in big-endian format, which is the most significant byte of a word in the low-address byte position. As 32-bit strings, the initialization values (in hexadecimal) appear as follows:

  word A: 67 45 23 01
  word B: EF CD AB 89
  word C: 98 BA DC FE
  word D: 10 32 54 76
  word E: C3 D2 E1 F0

- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each. The logic is illustrated in Figure 12.5. The four rounds have a similar structure, but each uses a different primitive logical function, which we refer to as f_1, f_2, f_3, and f_4.

  Each round takes as input the current 512-bit block being processed (Y_q) and the 160-bit buffer value ABCDE and updates the contents of the buffer.

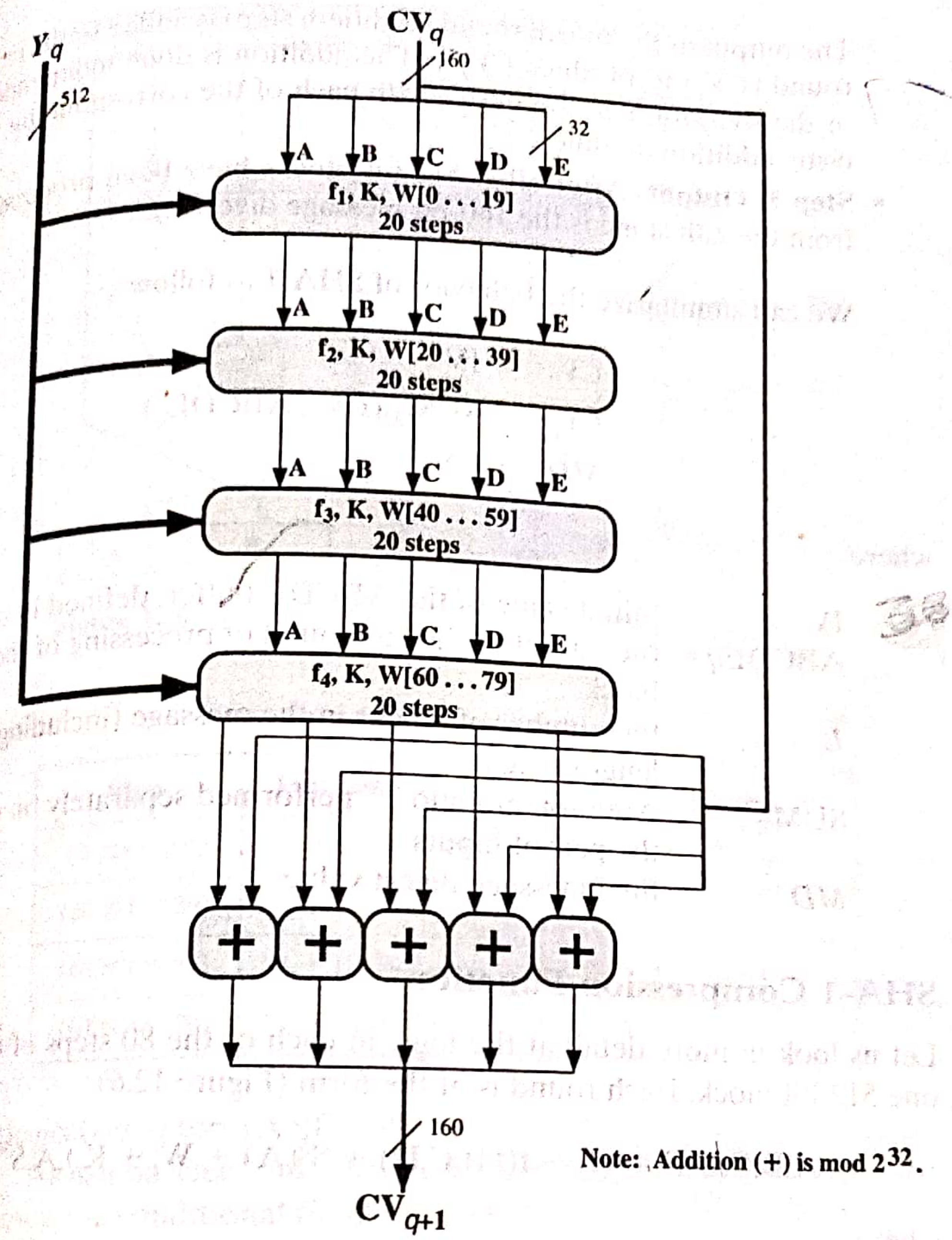

Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 compression function)

Each round also makes use of an additive constant K_t, where $0 \le t \le 79$ indicates one of the 80 steps across five rounds. In fact, only four distinct constants are used. The values, in hexadecimal and decimal, are as follows:

Step Number	Hexadecimal	Take Integer Part of:
$0 \le t \le 19$	K_t = 5A827999	$[2^{30} \times \sqrt{2}]$
$20 \le t \le 39$	K_t = 6ED9EBA1	$[2^{30} \times \sqrt{3}]$
$40 \le t \le 59$	K_t = 8F1BBCDC	$[2^{30} \times \sqrt{5}]$
$60 \le t \le 79$	K_t = CA62C1D6	$[2^{30} \times \sqrt{10}]$

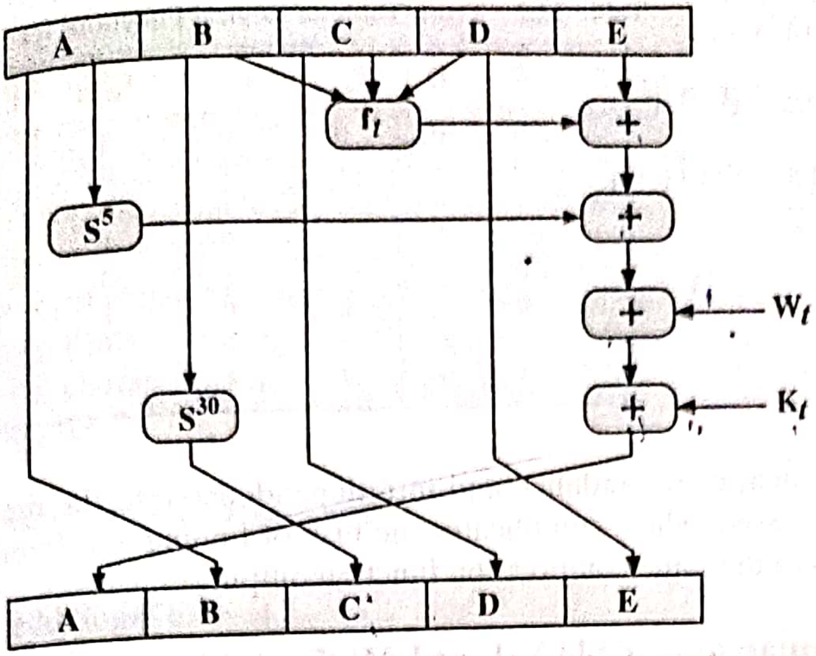

Figure 12.6 Elementary SHA Operation (single step)

Step	Function Name	Function Value
$(0 \le t \le 19)$	$f_1 = f(t, B, C, D)$	$(B \wedge C) \vee (\bar{B} \wedge D)$
$(20 \le t \le 39)$	$f_2 = f(t, B, C, D)$	$B \oplus C \oplus D$
$(40 \le t \le 59)$	$f_3 = f(t, B, C, D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \le t \le 79)$	$f_4 = f(t, B, C, D)$	$B \oplus C \oplus D$

The logical operators (AND, OR, NOT, XOR) are represented by the symbols ($\wedge, \vee, \bar{}, \oplus$). As can be seen, only three different functions are used. For $0 \le t \le 19$, the function is the conditional function: If B, then C else D. For $20 \le t \le 39$ and $60 \le t \le 79$, the function produces a parity bit. For $40 \le t \le 59$, the function is true if two or three of the arguments are true. Table 12.2 is a truth table of these functions.

It remains to indicate how the 32-bit word values W_t are derived from the 512-bit message. Figure 12.7 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 follows:

$$W_t = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$

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. This is a notable difference from MD5 and RIPEMD-160, both of which use one of the 16 words of a message block directly as input to each step function; only the order of the words is permuted from round to round. SHA-1 expands the 16 block words to 80 words for use in the compression function. This introduces a

SHA-1 | MD5
160 bit o/p → 128-bit o/p
big-endian → little-endian
→ slow → fast
(80 steps) (64 steps)

Table 12.2 Truth Table of Logical Functions for SHA-1

B	C	D	$f_{0..19}$	$f_{20..39}$	$f_{40..59}$	$f_{60..79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

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.

Comparison of SHA-1 and MD5

Because both are derived from MD4, SHA-1 and MD5 are quite similar to one another. Accordingly, their strengths and other characteristics should be similar. We compare the two algorithms using the design goals cited earlier for MD4:

- **Security against brute-force attacks:** The most obvious and most important difference is that the SHA-1 digest is 32 bits longer than the MD5 digest. Using a brute-force technique, the difficulty of producing any message having a given message digest is on the order of 2^{128} operations for MD5 and 2^{160} for SHA-1. Again, using a brute-force technique, the difficulty of producing two messages having the same message digest is on the order of 2^{64} operations for MD5 and 2^{80} for SHA-1. Thus, SHA-1 is considerably stronger against brute-force attacks.
- **Security against cryptanalysis:** As was discussed in the previous section, MD5 is vulnerable to cryptanalytic attacks discovered since its design. SHA-1 appears not to be vulnerable to such attacks. However, little is publicly known about the design criteria for SHA-1, so its strength is more difficult to judge than would otherwise be the case.

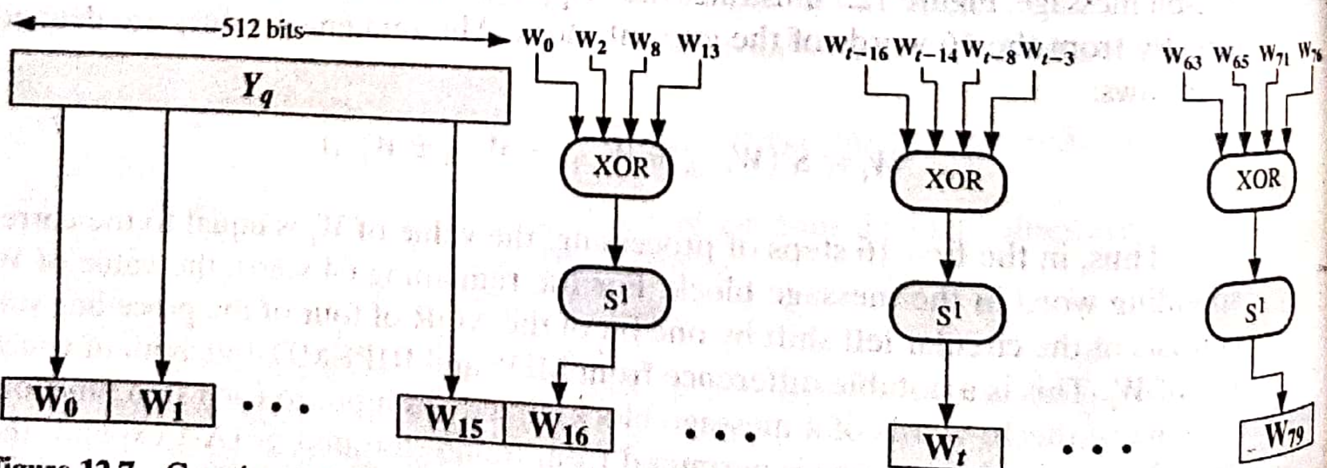

Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

- **Speed:** Because both algorithms rely heavily on addition modulo 2^{32}, both do well on a 32-bit architecture. SHA-1 involves more steps (80 versus 64) and must process a 160-bit buffer compared to MD5's 128-bit buffer. Thus, SHA-1 should execute more slowly than MD5 on the same hardware.
- **Simplicity and compactness:** Both algorithms are simple to describe and simple to implement and do not require large programs or substitution tables.
- **Little-endian versus big-endian architecture:** MD5 uses a little-endian scheme for interpreting a message as a sequence of 32-bit words, whereas SHA-1 uses a big-endian scheme.[1] There appears to be no significant advantage to either approach.

The output of the fourth round (eightieth step) is added to the input to the first round (CV_q) to produce CV_{q+1}. The addition is done independently for each of the five words in the buffer with each of the corresponding words in CV_q, using addition modulo 2^{32}.

- **Step 5: Output.** After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.

We can summarize the behavior of SHA-1 as follows:

$$\begin{aligned} CV_0 &= IV \\ CV_{q+1} &= SUM_{32}(CV_q, ABCDE_q) \\ MD &= CV_L \end{aligned}$$

where

IV	= initial value of the ABCDE buffer, defined in step 3
ABCDEq	= the output of the last round of processing of the qth message block
L	= the number of blocks in the message (including padding and length fields)
SUM_{32}	= Addition modulo 2^{32} performed separately on each word of the pair of inputs
MD	= final message digest value

SHA-1 Compression Function

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 of the form (Figure 12.6)

$$A,B,C,D,E \leftarrow (E + f(t,B,C,D) + S^5(A) + W_t + K_t),A,S^{30}(B),C,D$$

where

A, B, C, D, E	= the five words of the buffer
t	= step number; $0 \leq t \leq 79$
f(t, B, C, D)	= primitive logical function for step t
S^k	= circular left shift (rotation) of the 32-bit argument by k bits
W_t	= a 32-bit word derived from the current 512-bit input block
K_t	= an additive constant; four distinct values are used, as defined previously
+	= addition modulo 2^{32}

Each primitive function takes three 32-bit words as input and produces a 32-bit word output. Each function performs a set of bitwise logical operations; that is, the nth bit of the output is a function of the nth bit of the three inputs. The functions can be summarized as follows: