... reluctance to depart from a proven stru
DES is based on the Feistel cipher, which in turn is based on the substitution-
mutation network proposal of Shannon. Many important subsequent block ciph
follow the Feistel design because the design can be adapted to resist ne
ered cryptanalytic threats. If, instead, an entirely new design were use
metric block cipher, there would be concern that the structure itself op
avenues of attack not yet thought of. Similarly, most important modern
tions follow the basic structure of Figure 11.10. Again, this has proved to
damentally sound structure, and newer designs simply refine the structure
to the hash code length.

In this chapter, we look at three important hash functions: MD5, SHA-1, and RIPEMD-160. We then look at an Internet-standard message authentication code, HMAC, that is based on the use of a hash function.

.1 MD5 MESSAGE DIGEST ALGORITHM

The MD5 message-digest algorithm (RFC 1321) was developed by Ron Rivest at MIT (the "R" in the RSA [Rivest-Shamir-Adleman] public-key encryption algorithm). Until the last few years, when both brute-force and cryptanalytic concerns have arisen, MD5 was the most widely used secure hash algorithm.

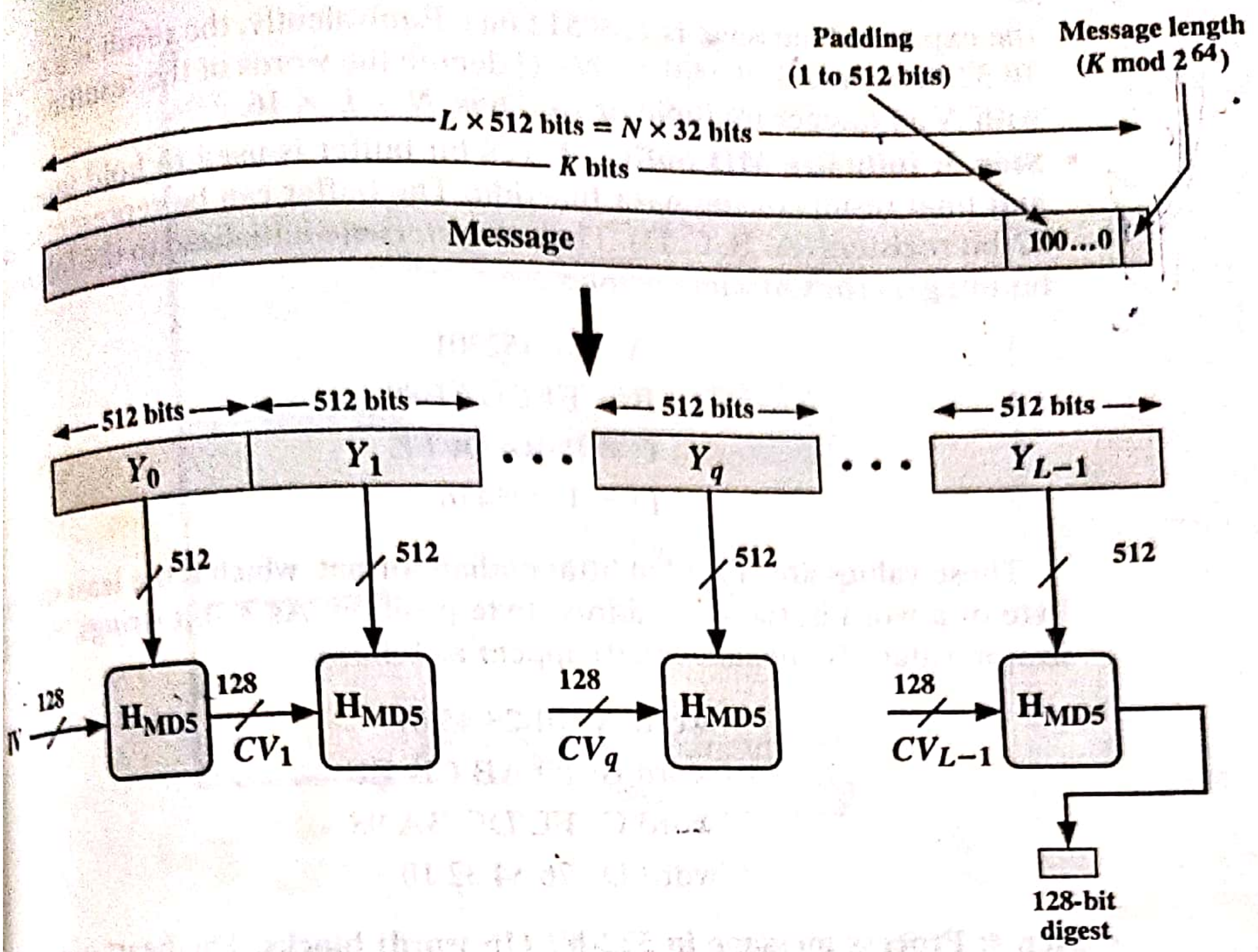


Figure 12.1 Message Digest Generation Using MD5

MD5 Logic

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest. The input is processed in 512-bit blocks.

Figure 12.1 depicts the overall processing of a message to produce a digest. This follows the general structure depicted in Figure 11.10. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length in bits is congruent to 448 modulo 512 (length $\equiv$ 448 mod 512). That is, the length of the padded message is 64 bits less than an integer multiple of 512 bits. Padding is always added, even if the message is already of the desired length. For example, if the message is 448 bits long, it is padded by 512 bits to a length of 960 bits. 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 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step 1 (least significant byte first). If the original length is greater than 2^{64}, then only the low-order 64 bits of the length are used. Thus, the field contains the length of the original message, modulo 2^{64}.

  The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In Figure 12.1, the expanded message is represented as the sequence of 512-bit blocks $Y_0, Y_1, \ldots, Y_{L-1}$, so that the total length of

1024
64
960

the expanded message is $L \times 512$ bits. Equivalently, the result is a multiple of 16 32-bit words. Let M[0 . . . $N-1$] denote the words of the resulting message, with N an integer multiple of 16. Thus, $N = L \times 16$.

- **Step 3: Initialize MD buffer.** A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D). These registers are initialized to the following 32-bit integers (hexadecimal values):

$$A = 67452301$$
$$B = EFCDAB89$$
$$C = 98BADCFE$$
$$D = 10325476$$

These values are stored in little-endian format, which is the least 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: 01 23 45 67
word B: 89 AB CD EF
word C: FE DC BA 98
word D: 76 54 32 10

- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a compression function that consists of four "rounds" of processing; this module is labeled H_{MD5} in Figure 12.1, and its logic is illustrated in Figure 12.2. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H, and I in the specification.

  Each round takes as input the current 512-bit block being processed (Y_q) and the 128-bit buffer value ABCD and updates the contents of the buffer. Each round also makes use of one-fourth of a 64-element table T[1 . . . 64], constructed from the sine function. The ith element of T, denoted T[i], has the value equal to the integer part of $2^{32} \times \text{abs}(\sin(i))$, where i is in radians. Because abs(sin(i)) is a number between 0 and 1, each element of T is an integer that can be represented in 32 bits. The table provides a "randomized" set of 32-bit patterns, which should eliminate any regularities in the input data. Table 12.1b lists values of T.

  The output of the fourth round 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 four 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 128-bit message digest.

We can summarize the behavior of MD5 as follows:

$$\begin{aligned} CV_0 &= IV \\ CV_{q+1} &= \text{SUM}_{32}[CV_q, \text{RF}_I(Y_q, \text{RF}_H(Y_q, \text{RF}_G(Y_q, \text{RF}_F(Y_q, CV_q))))] \\ MD &= CV_{L-1} \end{aligned}$$

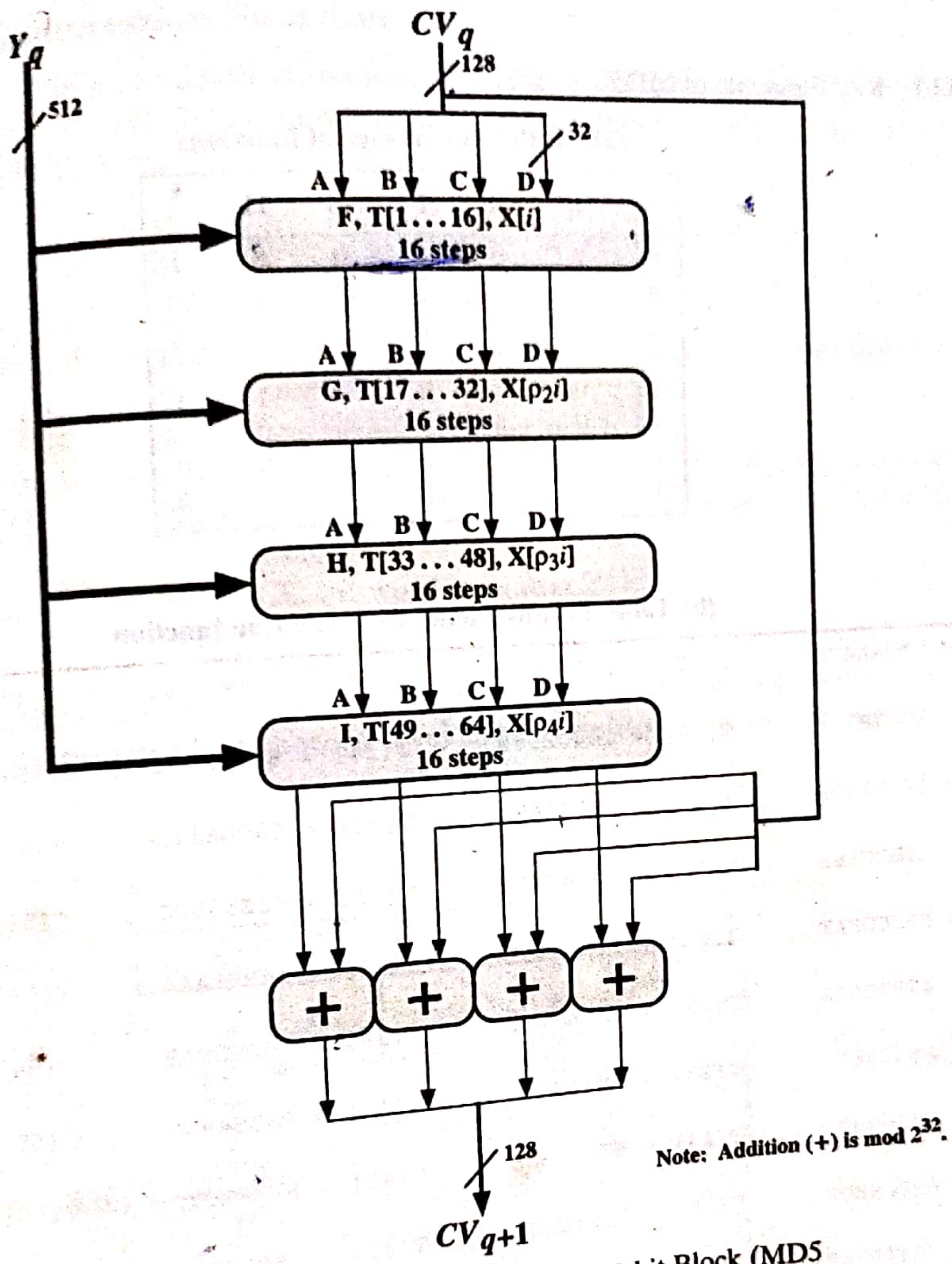


Figure 12.2 MD5 Processing of a Single 512-bit Block (MD5 compresion function)

where

IV	= initial value of the ABCD buffer, defined in step 3
Y_q	= the qth 512-bit block of the message
L	= the number of blocks in the message (including padding and length fields)
CV_q	= chaining variable processed with the qth block of the message
RF_x	= round function using primitive logical function x
MD	= final message digest value
SUM_{32}	= Addition modulo 2^{32} performed separately on each word of the pair of inputs

Table 12.1 Key Elements of MD5

(a) Truth table of logical functions

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

(b) Table T, constructed from the sine function

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

MD5 Compression Function

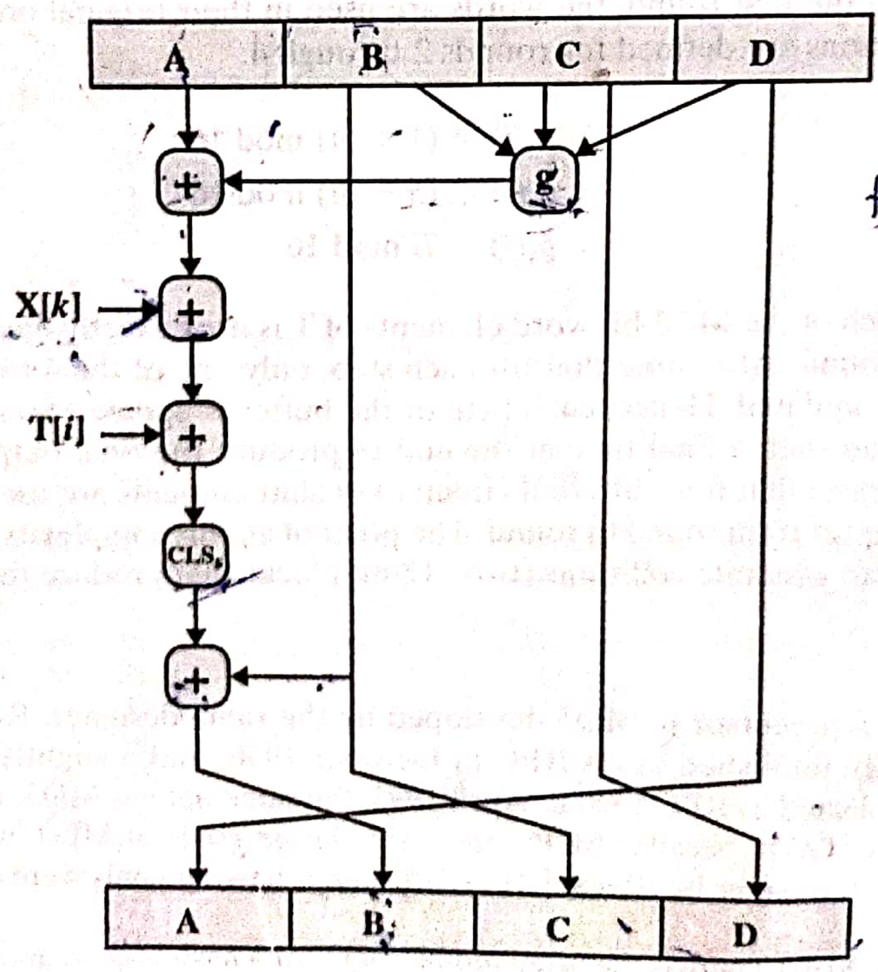
Let us look in more detail at the logic in each of the four rounds of the processing of one 512-bit block. Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form

$$a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i]) <<< s)$$

where

a, b, c, d	=	the four words of the buffer, in a specified order that varies across steps
g	=	one of the primitive functions F, G, H, I
$<<< s$	=	circular left shift (rotation) of the 32-bit argument by s bits
$X[k]$	=	$M[q \times 16 + k]$ = the kth 32-bit word in the qth 512-bit block of the message
$T[i]$	=	the ith 32-bit word in matrix T
+	=	addition modulo 2^{32}

Figure 12.3 illustrates the step operation. The order in which the four words (a, b, c, d) are used produces a word-level circular right shift of one word for each step.



F, G, H, I

Figure 12.3 Elementary MD5 Operation (single step)