MD5 Message Digest Algorithm

MD5 (http://www.faqs.org/rfcs/rfc1321.html) was developed by Ron Rivest at MIT in 1991. Until 1996, when a flaw was found in it, MD5 was the most widely used secure hash algorithm. In description, we follow Stallings, Cryptography and Network Security textbook.

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. The processing consists of the following steps:

1. **Append padding bits**: The message is padded so that its length in bits is congruent to 448 modulo 512 ($length \equiv 448 \bmod 512$). 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 with 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.
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 lower-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,

MD5 Logic (Cont 1)

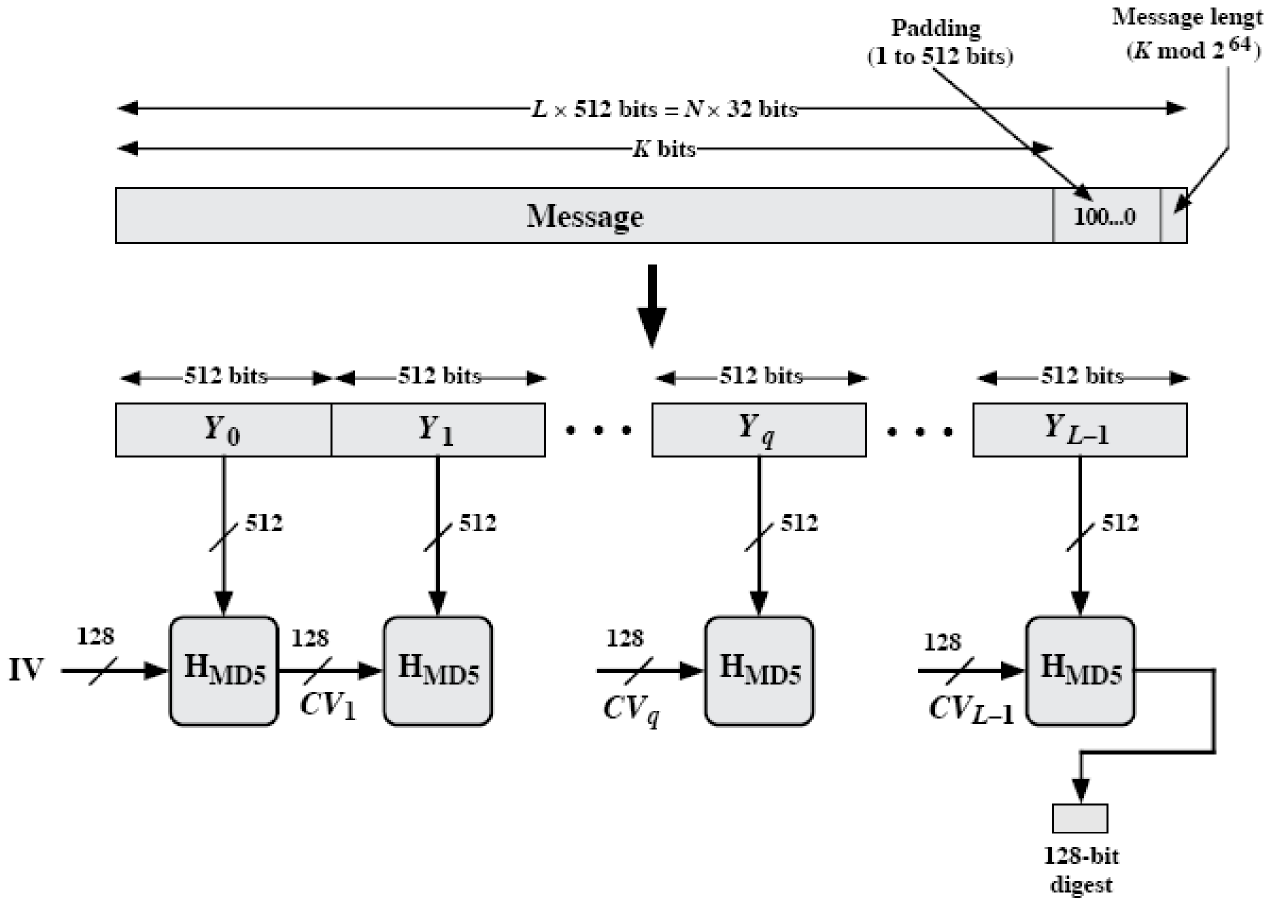

Figure 12.1 Message Digest Generation Using MD5

the expanded message is represented as the sequence of 512-bit blocks $Y_0, Y_1, .., Y_{L-1}$, so that the total length of 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, $N = L \times 16$.

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 format) 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

MD5 Logic (Cont 2)

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.

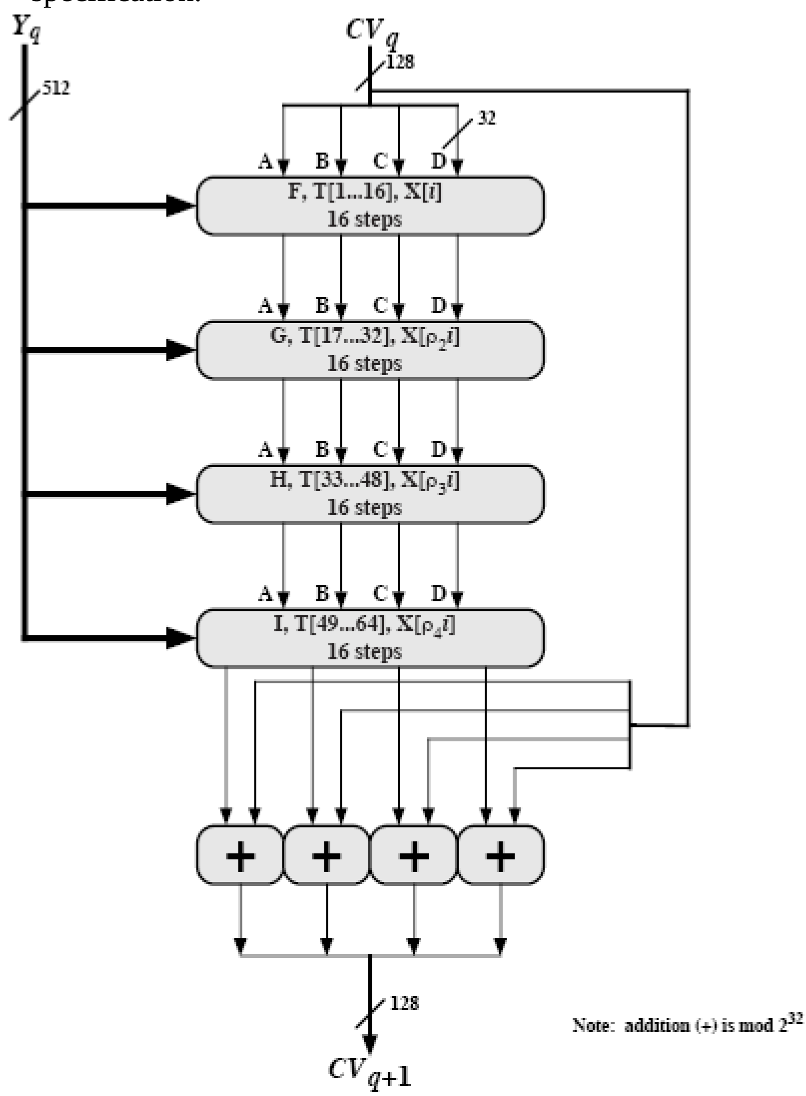

Figure 12.2 MD5 Processing of a Single 512-bit Block

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 *i*-th element of T, denoted T[i], has the value equal to the integer part of $2^{32} \times abs(\sin(i))$, where *i* is in radians. Because $abs(\sin(i)) \in [0,1]$, 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.

MD5 Logic (Cont 3)

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}.

5. Output: After all L 512-bit blocks have been processed, the output from the L-th stage is the 128-bit message digest.

We can summarize the behavior of MD5 as follows:

$CV_0 = IV$

$CV_{q+1} = SUM_{32}(CV_q, RF_I(Y_q, RF_H(Y_q, RF_G(Y_q, RF_F(Y_q, CV_q)))))$

$MD = CV_{L-1}$

Where

IV - initial value of the ABCD buffer, defined in Step 3

Y_q - the *q*-th 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 *q*-th block of the message

RF_x - round function using primitive logic function *x*

MD - final message digest value

SUM_{32} - addition modulo 2^{32} performed separately on each word of the pair of inputs

MD5 Logic (Cont 4)

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's look in more detail at the logic 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
$X[k] - M[q \times 16+k]$ – *k*-th 32-bit word in the *q*-th 512-bit of the message
T[i] – the *i*-th 32-bit word in matrix *T*
+ - addition modulo 2^{32}
Figure 12.3 illustrates the step operation

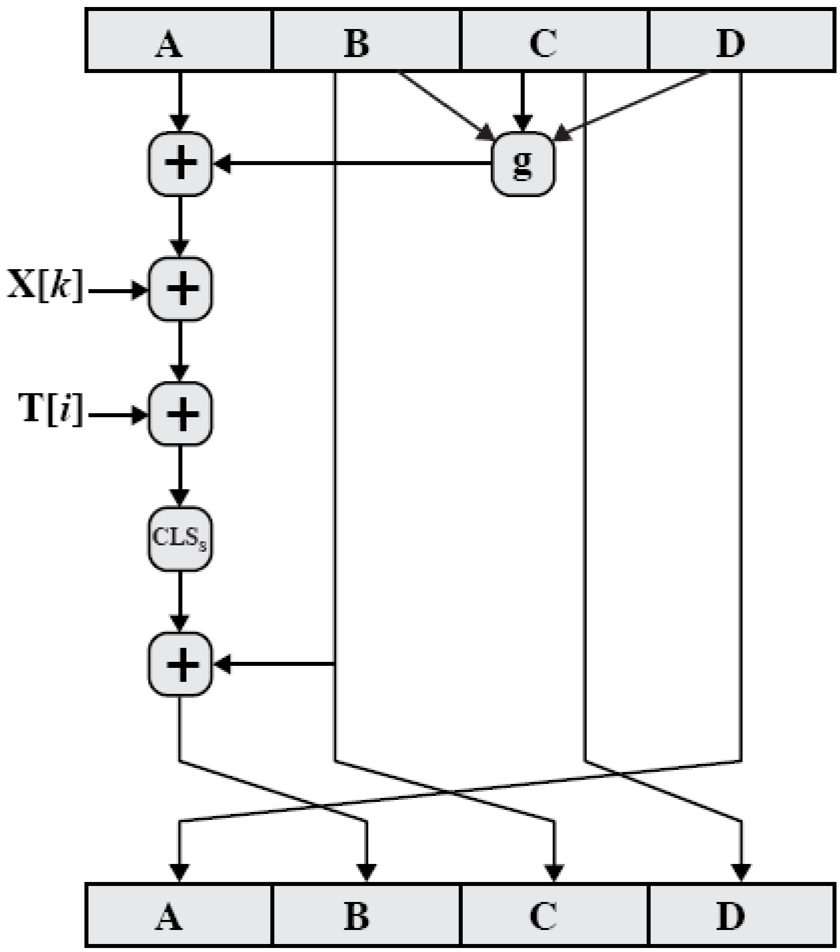

Figure 12.3 Elementary MD5 Operation (single step)

The order in which the four words (*a*,*b*,*c*,*d*) are used produces a word-level circular right shift for each step.
One of the four primitive logical functions is used for each of the four rounds of the algorithm. Each primitive function takes three 32-bit words as input and produces a 32-bit output. Each

MD5 Compression Function (Cont 1)

function performs a set of logical operations; that is, the *n*-th bit of the output is a function of the three inputs. The functions can be summarized as follows:

Round	Primitive function g	g(b,c,d)
1	F(b,c,d)	$(b \wedge c) \vee (\bar{b} \wedge d)$
2	G(b,c,d)	$(b \wedge d) \vee (c \wedge \bar{d})$
3	H(b,c,d)	$b \oplus c \oplus d$
4	I(b,c,d)	$c \oplus (b \vee \bar{d})$

Figure 12.4 adapted from RFC 1321, defines the processing algorithm of step 4. The array of 32-bit words *X[0..15]* holds the value of the current 512-bit input block being processed. Within a round, each of the 16 words of *X[i]* is used exactly once, during one step; the order in which these words are used varies from round to round. In the first round, they are used in the original order. The following permutations are defined for rounds 2 through 4:

$$\rho_2(i) = (1 + 5i) \bmod 16$$

$$\rho_3(i) = (5 + 3i) \bmod 16$$

$$\rho_4(i) = 7i \bmod 16$$

Consider permutation ρ_2:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12

Each of the 64 32-bit word elements of *T* is used exactly once, during one step of one round. Also, note that for each step, only one of the 4 bytes of the *ABCD* buffer is updated. Hence, each byte of the buffer is updated four times during the round and then a final time at the end to produce the final output of this block. Finally, note that four different circular left shift amounts are used each round and are different from round to round. The point of all this complexity is to make it very difficult to generate collisions (two 512-bit blocks that produce the same output).

MD5 Compression Function (Cont 2)

```
/* Process each 16-word (512-bit) block. */
For q = 0 to (N/16) - 1 do
   /* Copy block q into X. */
   For j = 0 to 15 do
        Set X[j] to  M[q*16 + j].
   end /* of loop on j */

   /* Save A as AA, B as BB, C as CC, and
   D as DD. */
   AA = A
   BB = B
   CC = C
   DD = D

   /* Round 1. */
   /* Let [abcd  k  s  i] denote the operation
   a = b + ((a + F(b,c,d) + X[k] + T[i]) <<<s).
   Do the following 16 operations. */
   [ABCD    0    7    1]
   [DABC    1   12    2]
   [CDAB    2   17    3]
   [BCDA    3   22    4]
   [ABCD    4    7    5]
   [DABC    5   12    6]
   [CDAB    6   17    7]
   [BCDA    7   22    8]
   [ABCD    8    7    9]
   [DABC    9   12   10]
   [CDAB   10   17   11]
   [BCDA   11   22   12]
   [ABCD   12    7   13]
   [DABC   13   12   14]
   [CDAB   14   17   15]
   [BCDA   15   22   16]

   /* Round 2. */
   /* Let [abcd  k  s  i] denote the operation
   a = b + ((a + G(b,c,d) + X[k] + T[i]) <<<s).
   Do the following 16 operations. */
   [ABCD    1    5   17]
   [DABC    6    9   18]
   [CDAB   11   14   19]
   [BCDA    0   20   20]
   [ABCD    5    5   21]
   [DABC   10    9   22]
   [CDAB   15   14   23]
   [BCDA    4   20   24]
   [ABCD    9    5   25]
   [DABC   14    9   26]
   [CDAB    3   14   27]
   [BCDA    8   20   28]
   [ABCD   13    5   29]
   [DABC    2    9   30]
   [CDAB    7   14   31]
   [BCDA   12   20   32]

   /* Round 3. */
   /* Let [abcd  k  s  i] denote the operation
   a = b + ((a + H(b,c,d) + X[k] + T[i]) <<<s).
   Do the following 16 operations. */
   [ABCD    5    4   33]
   [DABC    8   11   34]
   [CDAB   11   16   35]
   [BCDA   14   23   36]
   [ABCD    1    4   37]
   [DABC    4   11   38]
   [CDAB    7   16   39]
   [BCDA   10   23   40]
   [ABCD   13    4   41]
   [DABC    0   11   42]
   [CDAB    3   16   43]
   [BCDA    6   23   44]
   [ABCD    9    4   45]
   [DABC   12   11   46]
   [CDAB   15   16   47]
   [BCDA    2   23   48]

   /* Round 4. */
   /* Let [abcd  k  s  i] denote the operation
   a = b + ((a + I(b,c,d) + X[k] + T[i]) <<<s).
   Do the following 16 operations. */
   [ABCD    0    6   49]
   [DABC    7   10   50]
   [CDAB   14   15   51]
   [BCDA    5   21   52]
   [ABCD   12    6   53]
   [DABC    3   10   54]
   [CDAB   10   15   55]
   [BCDA    1   21   56]
   [ABCD    8    6   57]
   [DABC   15   10   58]
   [CDAB    6   15   59]
   [BCDA   13   21   60]
   [ABCD    4    6   61]
   [DABC   11   10   62]
   [CDAB    2   15   63]
   [BCDA    9   21   64]

   /* Then increment each of the four registers by the
   value it had before this block was started. */
   A = A + AA
   B = B + BB
   C = C + CC
   D = D + DD

end /* of loop on q */
```

Figure 12.4 Basic MD5 Update Algorithm (RFC 1321)

We see that in Figure 12.4, order of *k* values in Round 2 follows specified above permutation ρ_2.