Appendix N
MD5 Hash Function

William Stallings

Copyright 2013

Supplement to
Cryptography and Network Security, Sixth Edition
Prentice Hall 2013
ISBN: 0133354695
http://williamstallings.com/Cryptography

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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). For many years, until both brute-force and cryptanalytic concerns arose, MD5 was the most widely used secure hash algorithm. It is still in use today as a component of other cryptographic functions.

N.1 MD5 LOGIC

MD5 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 N.1 depicts the overall processing of a message to produce a digest. 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 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.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

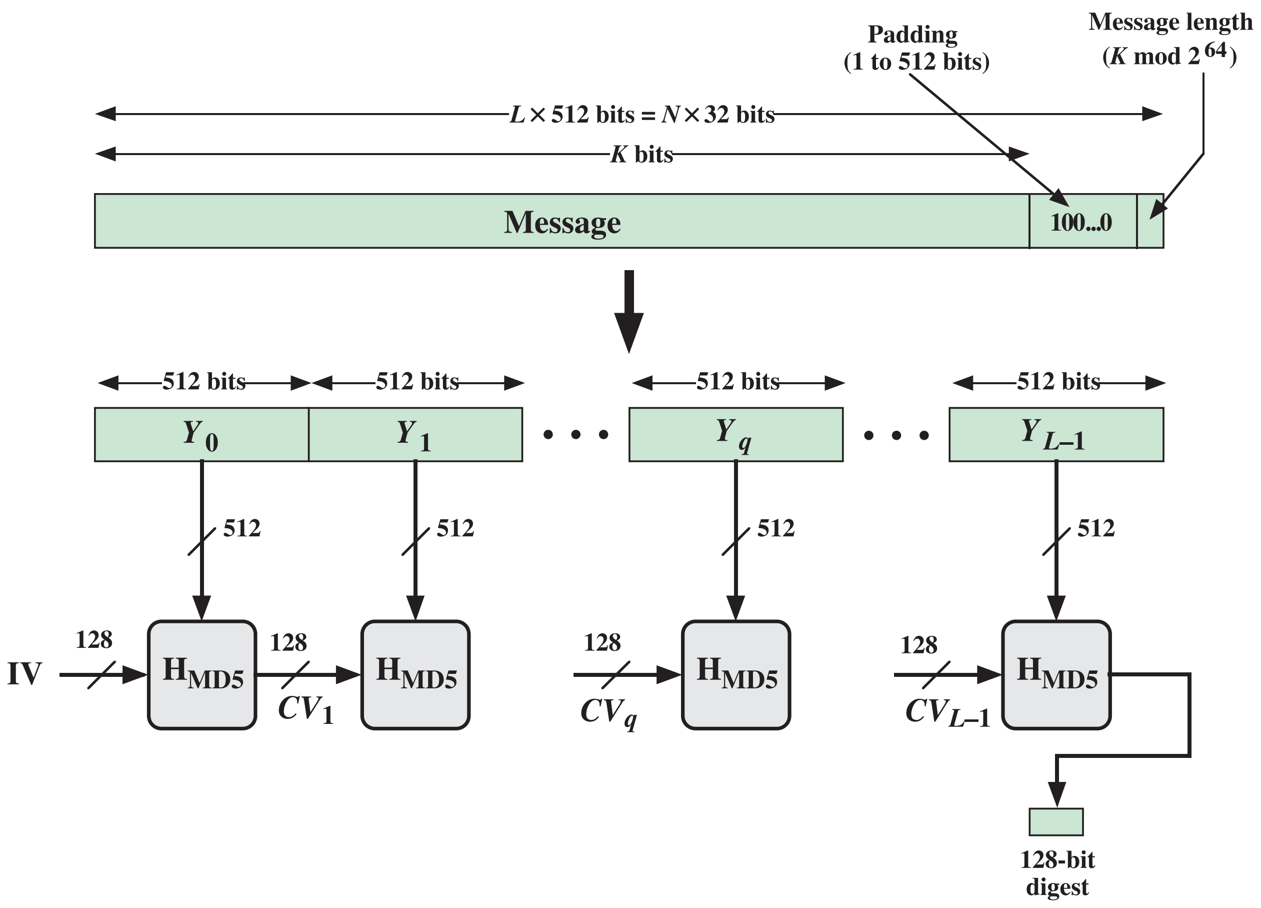

Figure N.1 Message Digest Generation Using MD5

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 N.1, 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, $N = L \times 16$.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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 N.1, and its logic is illustrated in Figure N.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.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

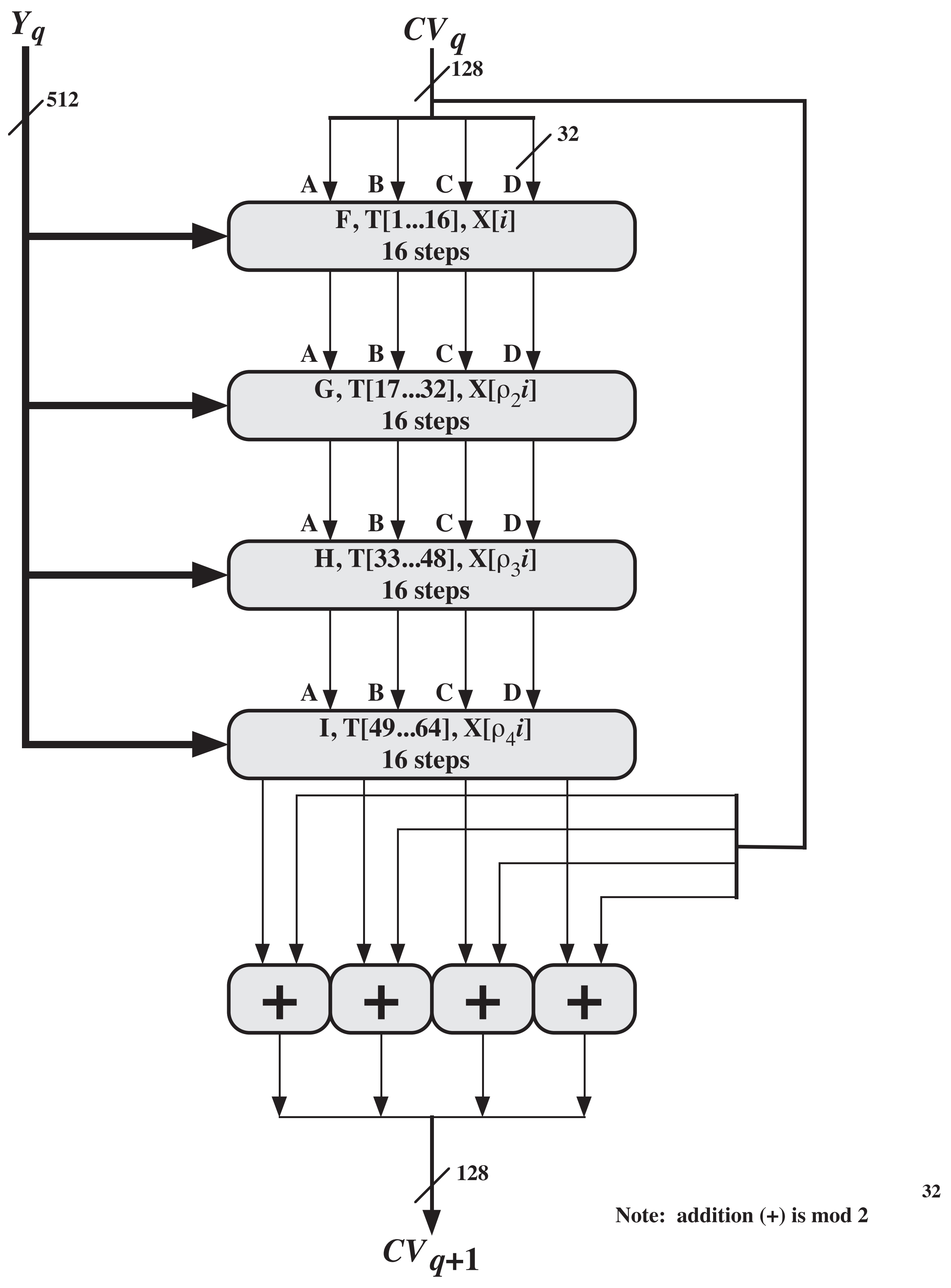
Y_q
512
CV_q
128
32
A
B
C
D
F, T[1...16], X[i]
16 steps
G, T[17...32], X[$\rho_2 i$]
16 steps
H, T[33...48], X[$\rho_3 i$]
16 steps
I, T[49...64], X[$\rho_4 i$]
16 steps
+
128
CV_{q+1}
Note: addition (+) is mod 2^{32}

Figure N.2 MD5 Processing of a Single 512-bit Block

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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 $2^{32} \times \text{abs}(\sin(i))$, 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 N.1 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}.

Table N.1 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

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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:

$$CV_0 = \text{IV}$$
$$CV_{q+1} = \text{SUM}_{32}[CV_q, \text{RF}_1 (Y_q, \text{RF}_\text{H} (Y_q, \text{RF}_\text{G} (Y_q, \text{RF}_\text{F} (Y_q, CV_q))))]$$
$$MD = CV_{L-1}$$

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

N.2 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:

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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

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 *k*th 32-bit word in the *q*th 512-bit block of the message
$T[i]$	=	the *i*th 32-bit word in matrix T
+	=	addition modulo 2^{32}

Figure N.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.

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 word output. Each function performs a set of bitwise logical operations; that is, the *n*th bit of the output is a function of the *n*th bit 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)	$c \oplus (b \vee \bar{d})$

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

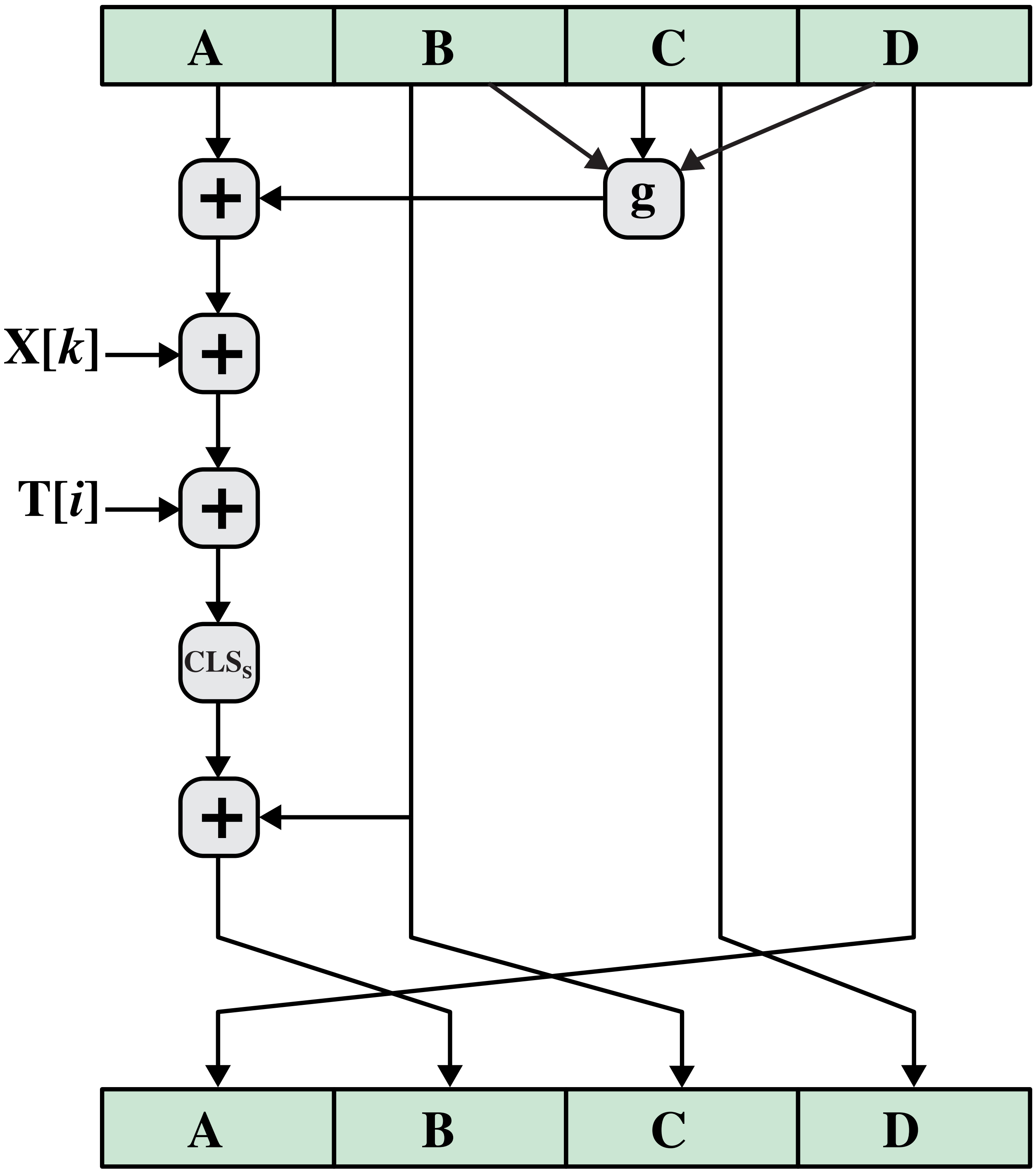

Figure N.3 Elementary MD5 Operation (single step)

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

Table N.2 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

The logical operators (AND, OR, NOT, XOR) are represented by the symbols ($\wedge$, $\vee$, $^{-}$, $\oplus$). Function F is a conditional function: If b then c else d. Similarly, G can be stated as follows: If d then b else c. Function H produces a parity bit. Table N.2 is a truth table of the four functions.

Figure N.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, the words are used in their original order. The following permutations are defined for rounds 2 through 4:

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

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

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

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

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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 for 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).

```
/* 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  1210]
  [CDAB 10  1711]
  [BCDA 11  2212]
  [ABCD 12   713]
  [DABC 13  1214]
  [CDAB 14  1715]
  [BCDA 15  2216]
```

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
/* 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   517]
[DABC  6   918]
[CDAB 11  1419]
[BCDA  0  2020]
[ABCD  5   521]
[DABC 10   922]
[CDAB 15  1423]
[BCDA  4  2024]
[ABCD  9   525]
[DABC 14   926]
[CDAB  3  1427]
[BCDA  8  2028]
[ABCD 13   529]
[DABC  2   930]
[CDAB  7  1431]
[BCDA 12  2032]

/* 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   433]
[DABC  8  1134]
[CDAB 11  1635]
[BCDA 14  2336]
[ABCD  1   437]
[DABC  4  1138]
[CDAB  7  1639]
[BCDA 10  2340]
[ABCD 13   441]
[DABC  0  1142]
[CDAB  3  1643]
[BCDA  6  2344]
[ABCD  9   445]
[DABC 12  1146]
[CDAB 15  1647]
[BCDA  2  2348]
```

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

```
  /* 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   649]
  [DABC  7  1050]
  [CDAB 14  1551]
  [BCDA  5  2152]
  [ABCD 12   653]
  [DABC  3  1054]
  [CDAB 10  1555]
  [BCDA  1  2156]
  [ABCD  8   657]
  [DABC 15  1058]
  [CDAB  6  1559]
  [BCDA 13  2160]
  [ABCD  4   661]
  [DABC 11  1062]
  [CDAB  2  1563]
   [BCDA 9  2164]

  /* 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)

N.3 MD4

MD4 is a precursor to MD5 developed by the same designer, Ron Rivest. It was originally published as an RFC in October 1990, and a slightly revised version was published as RFC 1320 in April 1992, the same date as MD5. It is worth briefly discussing MD4 because MD5 shares the design goals of MD4, which were documented in a paper by Rivest [RIVE90]. The following goals were listed:

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

- **Security:** There is the usual requirement for a hash code, namely, that it be computationally infeasible to find two messages that have the same message digest. At the time of its publication, MD4 was secure against brute-force attacks because of the length of the digest. Rivest also felt it to be secure against cryptanalytic attacks based on the state of the art and on the complexity of MD4.
- **Speed:** The algorithm should lend itself to implementations in software that executes rapidly. In particular, the algorithm is intended to be fast on 32-bit architectures. Thus, the algorithm is based on a simple set of primitive operations on 32-bit words.
- **Simplicity and compactness:** The algorithm should be simple to describe and simple to program, without requiring large programs or substitution tables. These characteristics not only have obvious programming advantages but also are desirable from a security point of view, because a simple algorithm is more likely to receive the necessary critical review.
- **Favor little-endian architecture:** Some processor architectures (such as the Intel 80xxx and Pentium line) store the least significant byte of a word in the low-address byte position (little endian). Others (such as a SUN Sparcstation) store the most significant byte of a word in the low-address byte position (big endian). This distinction is significant when treating a message as a sequence of 32-bit words, because one of the two architectures will have to byte-reverse each word for processing. Rivest chose to use a little-endian scheme for interpreting a message as a sequence of 32-bit words. This choice was made on the basis of Rivest's observation that big-endian processors are generally faster and can therefore better afford the processing penalty.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

These design goals carried over to MD5. MD5 is somewhat more complex and hence somewhat slower to execute than MD4. Rivest felt that the added complexity was justified by the increased level of security afforded. The following are the main differences between the two:

1. MD4 uses three rounds of 16 steps each, whereas MD5 uses four rounds of 16 steps each.
2. In MD4, no additive constant is used in the first round. The same additive constant is used for each of the steps of the second round. Another additive constant is used for each of the steps of the third round. In MD5, a different additive constant, T[*i*], is used for each of the 64 steps.
3. MD5 uses four primitive logical functions, one for each round, compared to three for MD4, again one for each round.
4. In MD5, each step adds in the result of the preceding step. For example, the step 1 result updates word A. The step 2 result, which is stored in D, is formed by adding A to the circular left shift result. Similarly, the step 3 result is stored in C and is formed by adding D to the circular left shift result. MD4 did not include this final addition. Rivest feels that the inclusion of the previous step's result promotes a greater avalanche effect.

N.4 STRENGTH OF MD5

The MD5 algorithm has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic functions (F, G, H, I) 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. Rivest conjectures in the RFC

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

that MD5 is as strong as possible for a 128-bit hash code; namely, the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, whereas the difficulty of finding a message with a given digest is on the order of 2^{128} operations.

As of this writing, no analysis has been done to disprove these conjectures. 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 [DOBB96]. 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.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

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. Accordingly, first SHA-1 and then SHA-2 have supplanted MD5.

N.5 REFERENCES

BERS92 Berson, T. "Differential Cryptanalysis Mod 2^{32} with Applications to MD5." *Proceedings, Eurocrypt '92*, May 1992.

BOER93 Boer, B., and Bosselaers, A. "Collisions for the Compression Function of MD5." *Proceedings, Eurocrypt '93*, 1993.

DOBB96 Dobbertin, H. "The Status of MD5 After a Recent Attack." *CryptoBytes*, Summer, 1996.

RIVE90 Rivest, R. "The MD4 Message Digest Algorithm." *Proceedings, Crypto '90*, August 1990.

© 2014 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved