Hash Algorithms

Hash Algorithms

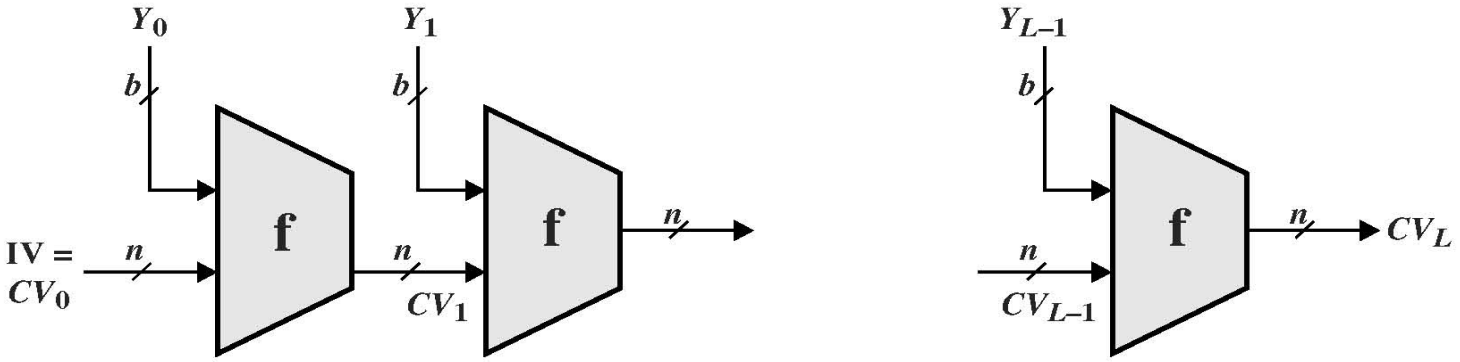
- a Hash Function produces a fingerprint of some file/message/data

  $h = H(M)$

  - Condenses a variable-length message M to a fixed-sized fingerprint

Hash Algorithms

- Simple hash functions proved unsafe
- Hence iterated hash function was proposed [Merkle] and called Secure Hash Function



IV = Initial value
CV = chaining variable
Y_i = *i*th input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

Figure 11.10 General Structure of Secure Hash Code

Hash Algorithms

- Secure Hash Function (Iterated Hash Function) is the structure of most functions in use today
  - MD5
  - SHA-1
  - RIPEMD-160

MD5

Illustrate MD 5 hash algorithm in detail

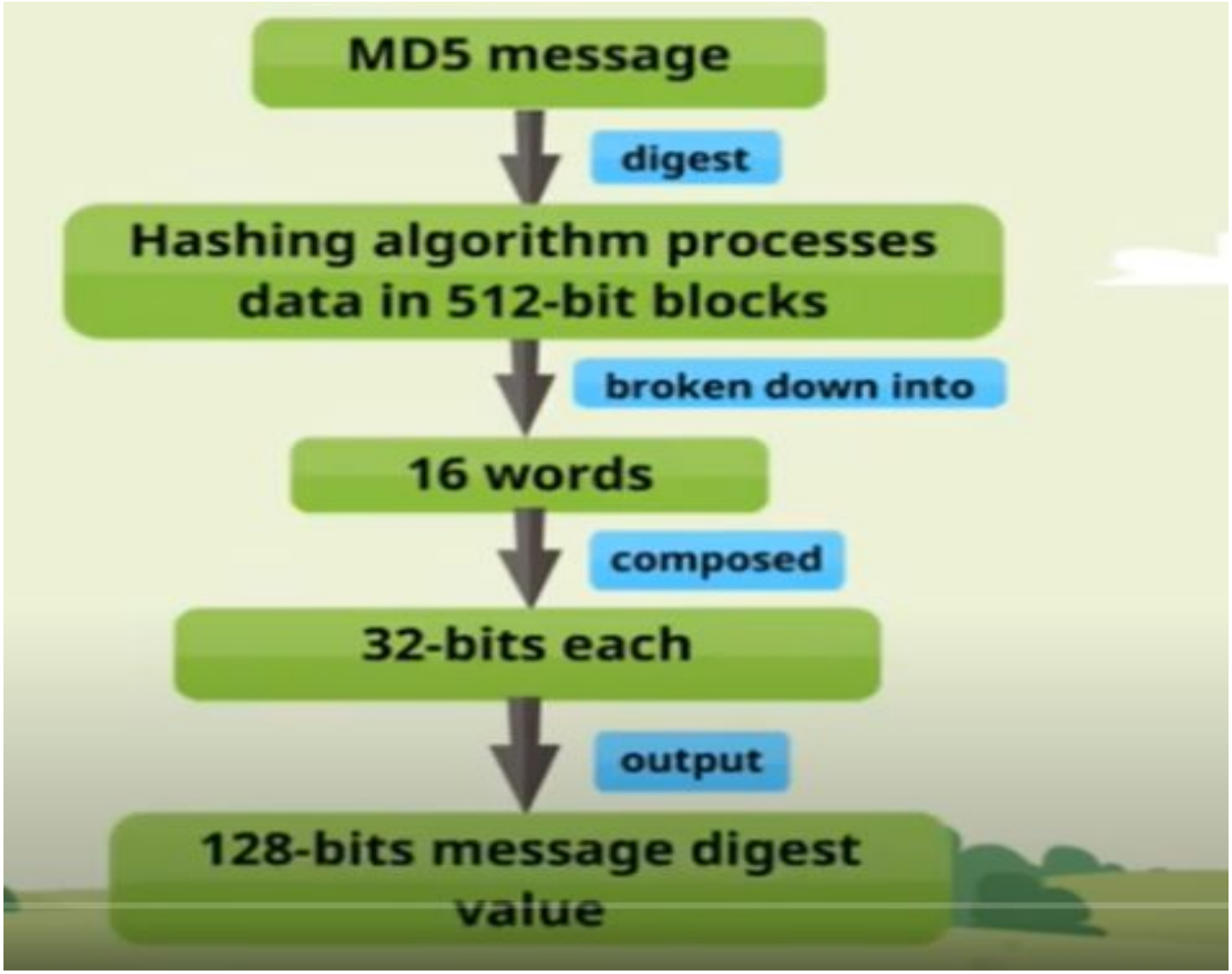
MD5

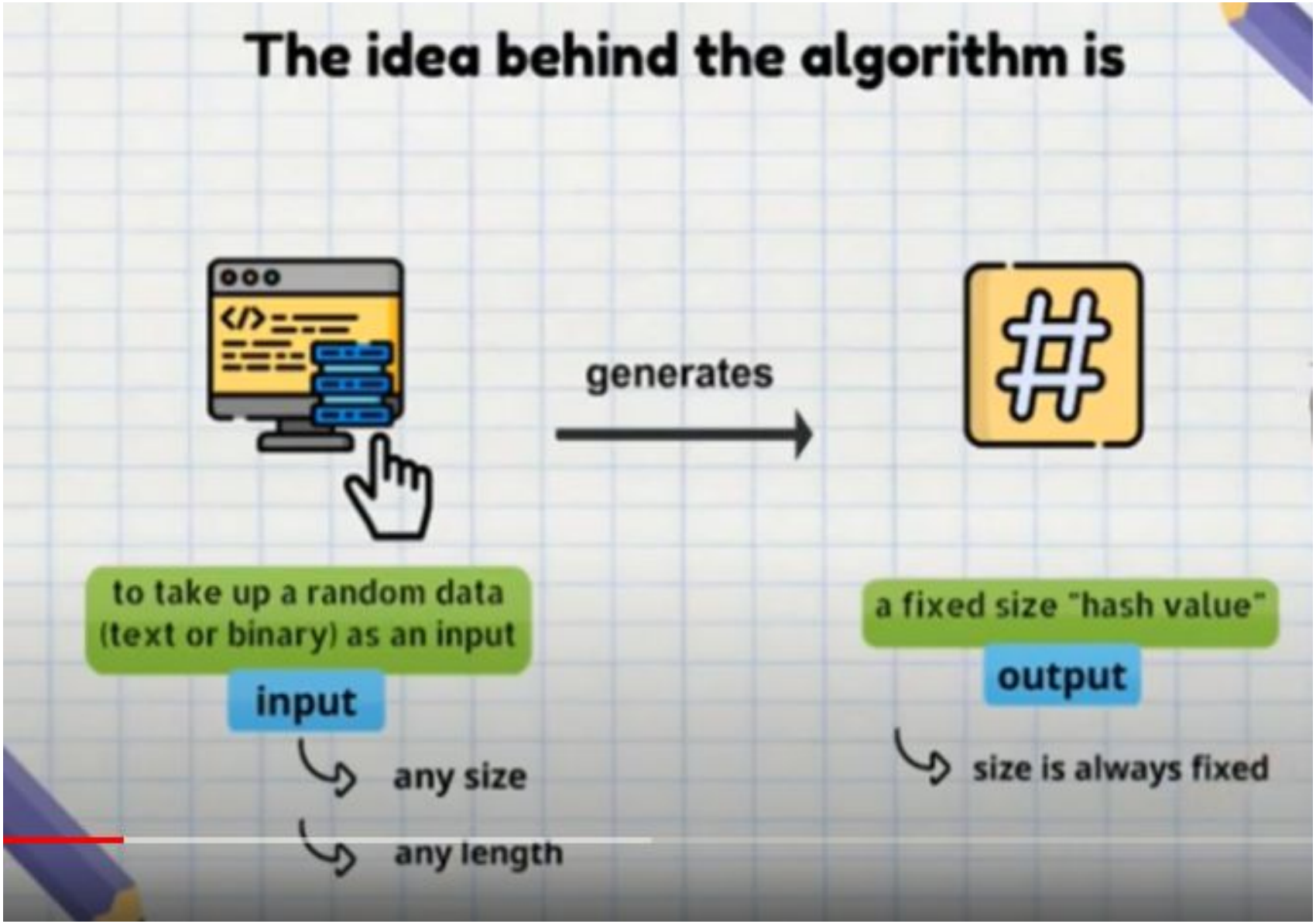
- Designed by Ronald Rivest (the R in RSA)
- Latest in a series of MD2, MD4
- Produces a 128-bit hash value
- Until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- Specified as Internet standard RFC1321

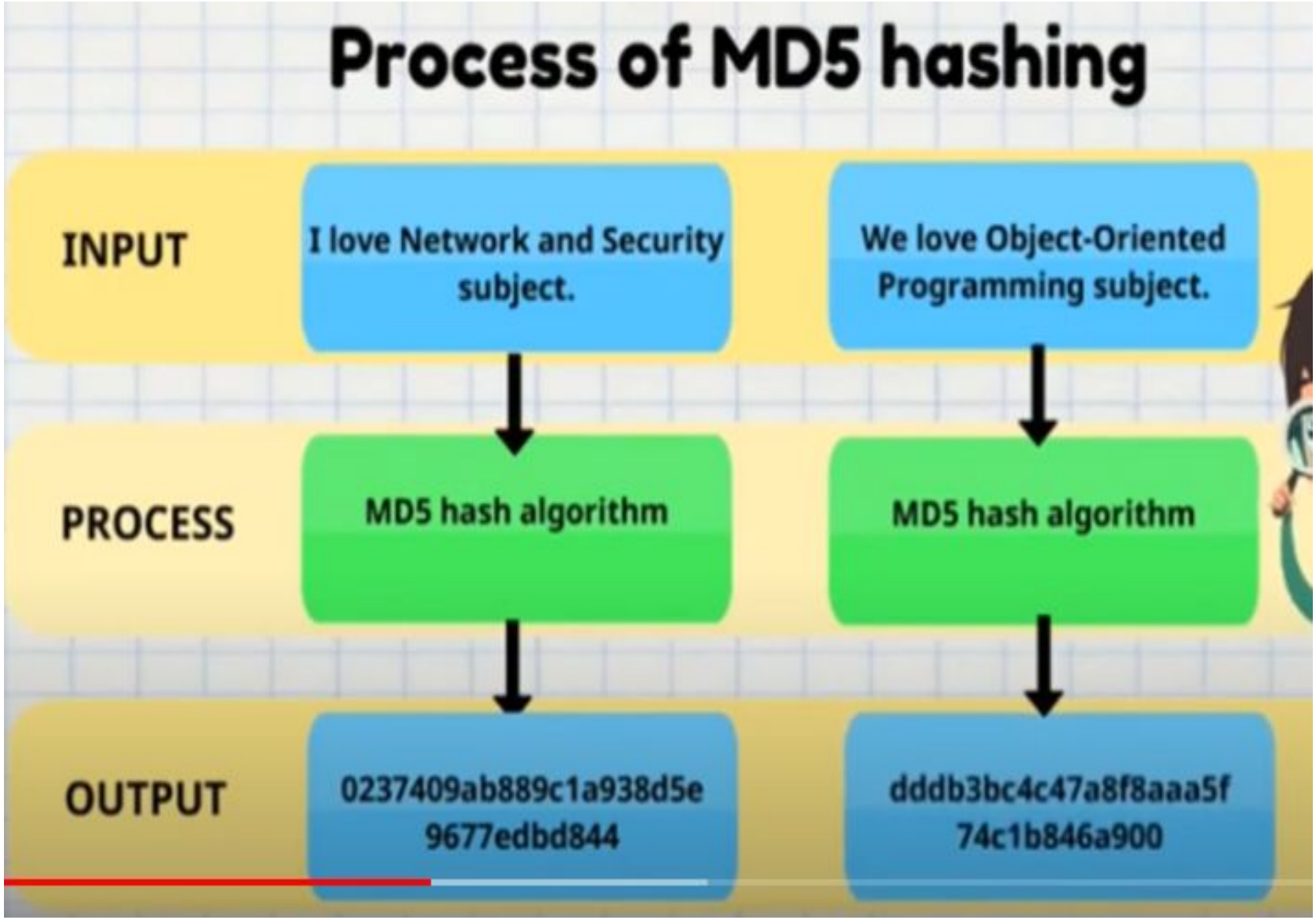
- The MD5 (message-digest algorithm) hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.
- is a cryptographic message authentication code algorithm for use on the internet,

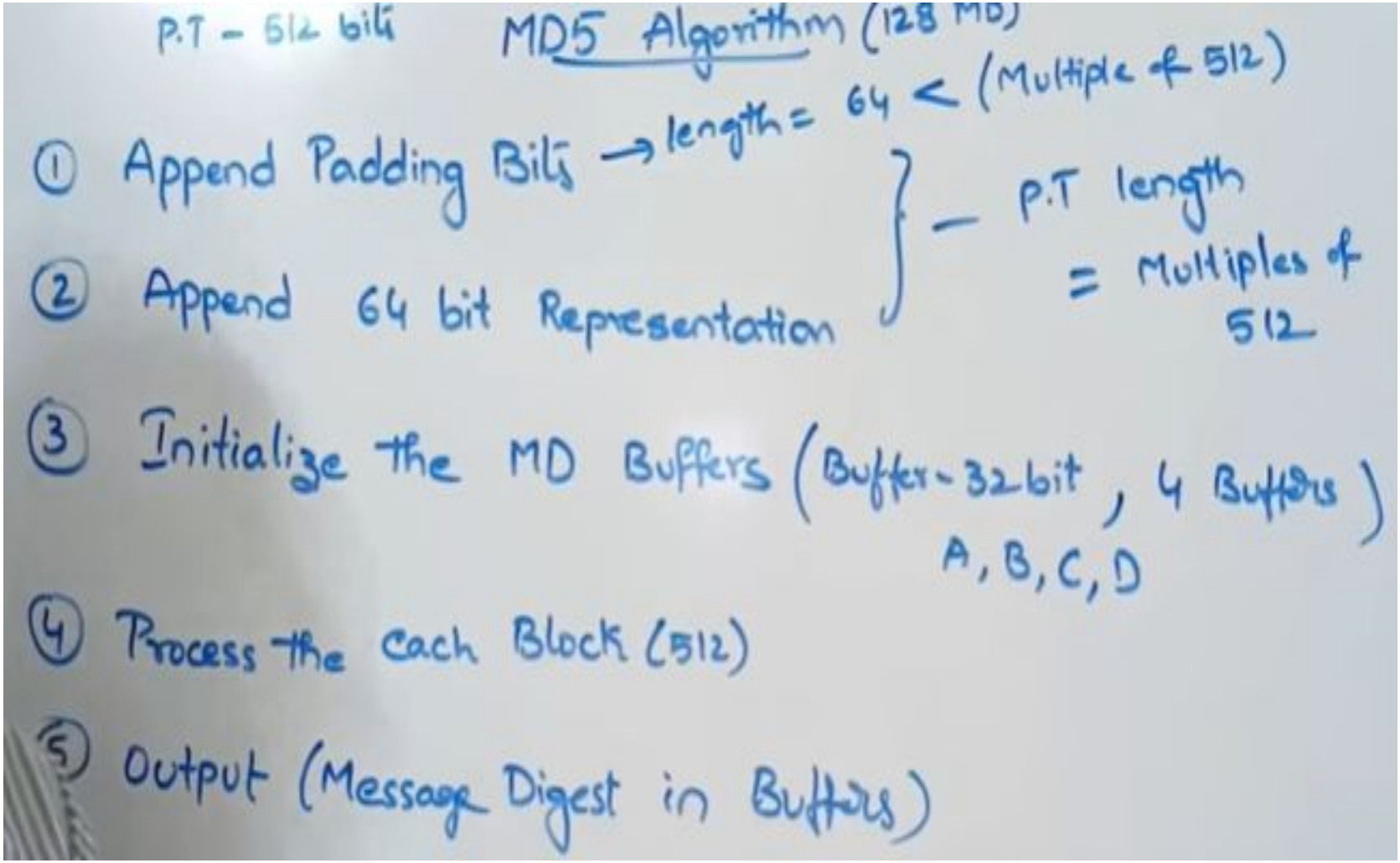
- After some initial processing, the input text is processed in 512-bit blocks, broken down into 16 words composed of 32 bits each
- The output of the algorithm is a set of four 32-bit blocks, which make up the 128-bit message digest.

https://www.comparitech.com/blog/information-security/md5-algorithm-with-examples/


MD5 message
digest
Hashing algorithm processes data in 512-bit blocks
broken down into
16 words
composed
32-bits each
output
128-bits message digest value


The idea behind the algorithm is
generates
to take up a random data
(text or binary) as an input
input
any size
any length
a fixed size "hash value"
output
size is always fixed


Process of MD5 hashing
INPUT
I love Network and Security subject.
We love Object-Oriented Programming subject.
PROCESS
MD5 hash algorithm
MD5 hash algorithm
OUTPUT
0237409ab889c1a938d5e9677edbd844
dddb3bc4c47a8f8aaa5f74c1b846a900

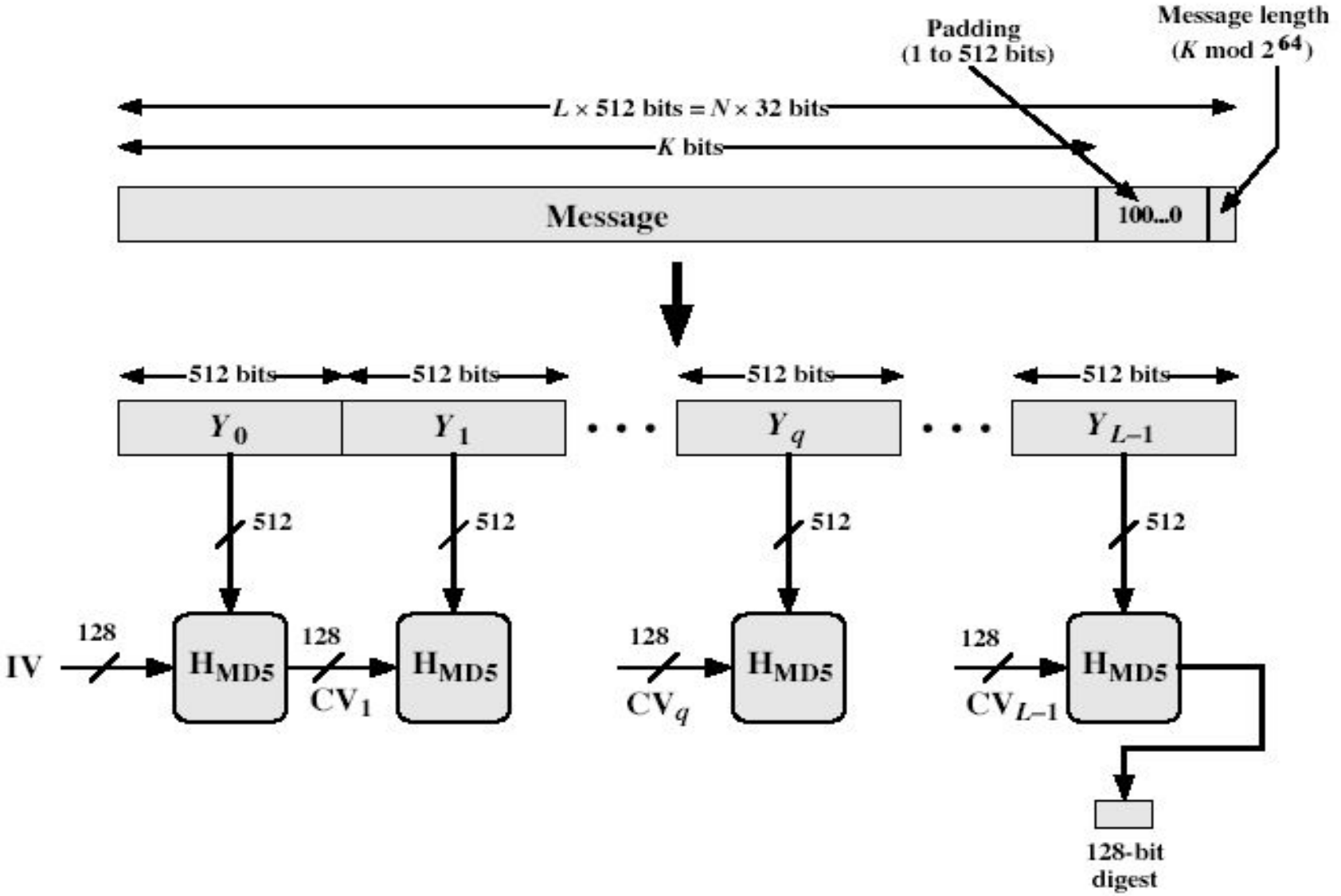

P.T - 512 bits
MD5 Algorithm (128 MD)
① Append Padding Bits → length = 64 < (Multiple of 512)
② Append 64 bit Representation
P.T length = Multiples of 512
③ Initialize the MD Buffers (Buffer - 32 bit, 4 Buffers A, B, C, D)
④ Process the each Block (512)
⑤ Output (Message Digest in Buffers)

- Step 1: Padding
- The First step in MD5 is to add padding bits to the original message. The aim of this step is to make the length of the original message equal to a value, which is 64-bit less than an exact multiple of 512
- The padding consists of a single 1-bit followed by as many 0-bits as required. Padding is always added even if the message length is already 64-bits less than a multiple of 512. The padding process is shown in the figure below

MD5 Logic

1. Append padding bits
   - Pad message so its length is 448 mod 512
   - Padding of 1-512 bits is always used.
   - Padding: 1000……0

MD5 Overview



- Step 2: Append Length
- After padding bits are added, the next step is to calculate the original length of the message and add it to the end of the message, after padding. The length of the message is calculated, excluding the padding bits. This length of the original message is expressed as a 64-bit value and these 64-bits are appended to the end of the original message along with padding.
- If the length of the message exceeds 264 bits, that is 64-bits are not enough to represent the length, only the low-order 64-bits of the length are used. So length is calculated as the actual length mod 264

- Step 3: Divide the input into 512-bit blocks
- Divide the input message into blocks, each of length 512 bits.

- **Extend into 64 words**

1. This is the first sub-step. Each chunk will be put through a little function that will create 80 words from the 16 current ones.
2. This step is a loop. What that means is that every step after this will be repeated until a certain condition is true.
3. In this case we will start by setting the variable 'i' equal to 16. After each run through of the loop we will add 1 to 'i' until 'i' is equal to 63.

- Step 4: Initialize MD buffer
  - A four-word buffer (A,B,C,D) is used to compute the message digest.
- Here each of A,B,C,D, is a 32 bit register.
  - These registers are initialized to the following values in hexadecimal:

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 5: Process Blocks

Four auxiliary functions that take as input three 32-bit words and produce as output one 32-bit word.

- **F(X,Y,Z) = XY v not(X) Z**
- **G(X,Y,Z) = XZ v Y not(Z)**
- **H(X,Y,Z) = X xor Y xor Z**
- **I(X,Y,Z) = Y xor (X v not(Z))**
- For each rounds we have the 16 input sub blocks, M[i] where $0 \leq i \leq 15$ and each sub-blocks consists of 32 bits. 't' is an array of constants and contains 64 elements with each element consisting of 32 bits. The elements of this array are represented as t[k] where k varies from 1 to 64

- So each round uses 16 out of 64 values of t. In each case the output of the intermediate as well as the final iteration is copied into the register abcd. Totally there are 16 iterations in each round

- **Converting the data to binary**
- When we put “They are deterministic” into an MD5 hash function, the first thing that happens is that it is converted to binary.
- This is done according to the **American Standard Code for Information Interchange (ASCII)**, which is basically a **standard that we use to convert human readable text into the binary code that computers can read**.
- Using an ASCII table, we see that a capital letter “T” is written as “01010100” in binary. A lowercase “h” is “01101000”, a lowercase “e” is “01100101”, and a lowercase “y” is “01111001”. The binary code for a space (SP) is “00100000”. You can see it in the table at the top of the second column, in line with the decimal number 32.

- If we continue on in this fashion, we see that our input, “They are deterministic” is written in binary as:

01010100 01101000 01100101 01111001 00100000 01100001 01110010 01100101
00100000 01100100 01100101 01110100 01100101 01110010 01101101 01101001
01101110 01101001 01110011 01110100 01101001 01100011

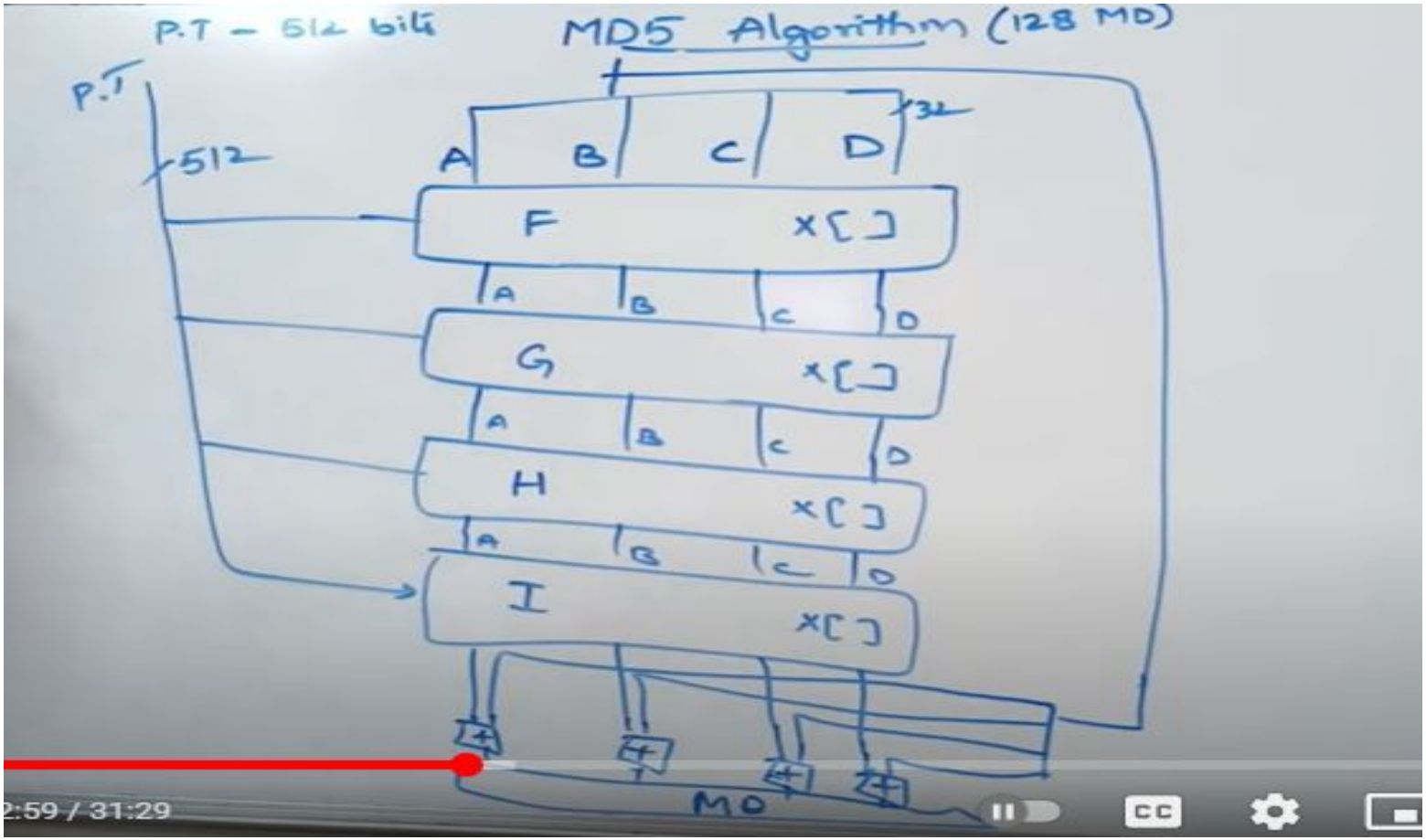
•Padding in the MD5 algorithm

- The next step in MD5 is to **add padding**. **Inputs in MD5 are broken up into 512-bit blocks, with padding added to fill up the rest of the space in the block**.
- Our input is 22 characters long including spaces, and each character is 8 bits long. This means that the input totals 176 bits.
- With an input of only 176 bits and a 512-bit block that needs to be filled, we need 336 bits of padding to complete the block.(336+176=512)

- MD5's padding scheme seems quite strange. After laying out the initial 176 bits of binary that represent our input, the rest of the block is padded with a single one, then enough zeros to bring it up to a length of 448 bits. So:
- 448 – 1 – 176 = 271
- Therefore **the padding for this block will include a one, then an extra 271 zeros**

Once the padding scheme is complete, we end up with the following 512-bit string:

01010100 01101000 01100101 01111001 00100000 01100001 01110010 01100101
00100000 01100100 01100101 01110100 01100101 01110010 01101101 01101001
01101110 01101001 01110011 01110100 01101001 01100011 10000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 10110000


P.T – 512 bits
MD5 Algorithm (128 MD)
P.T
512
A
B
C
D
32
F
X[ ]
G
H
I
MO
2:59 / 31:29

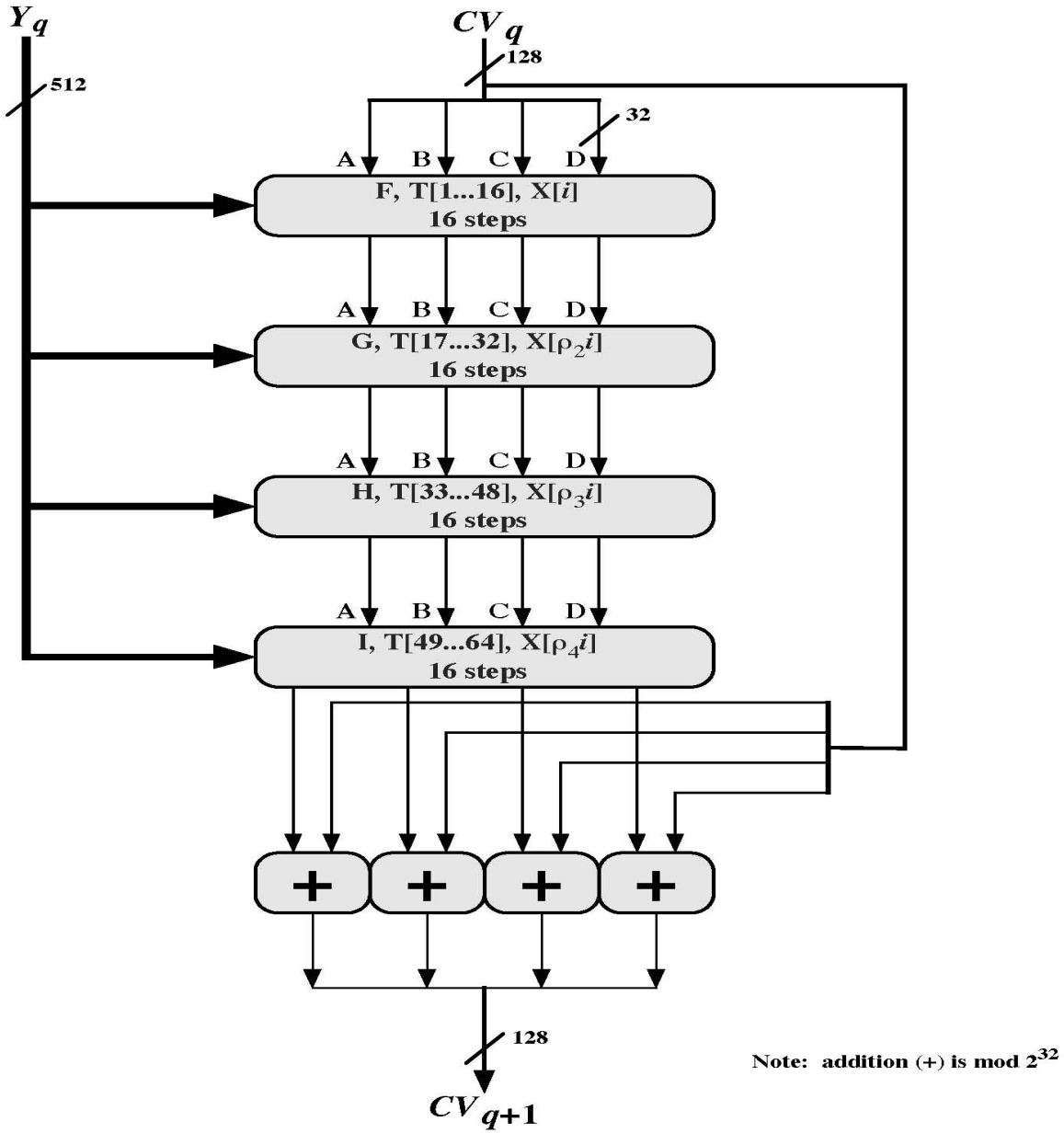
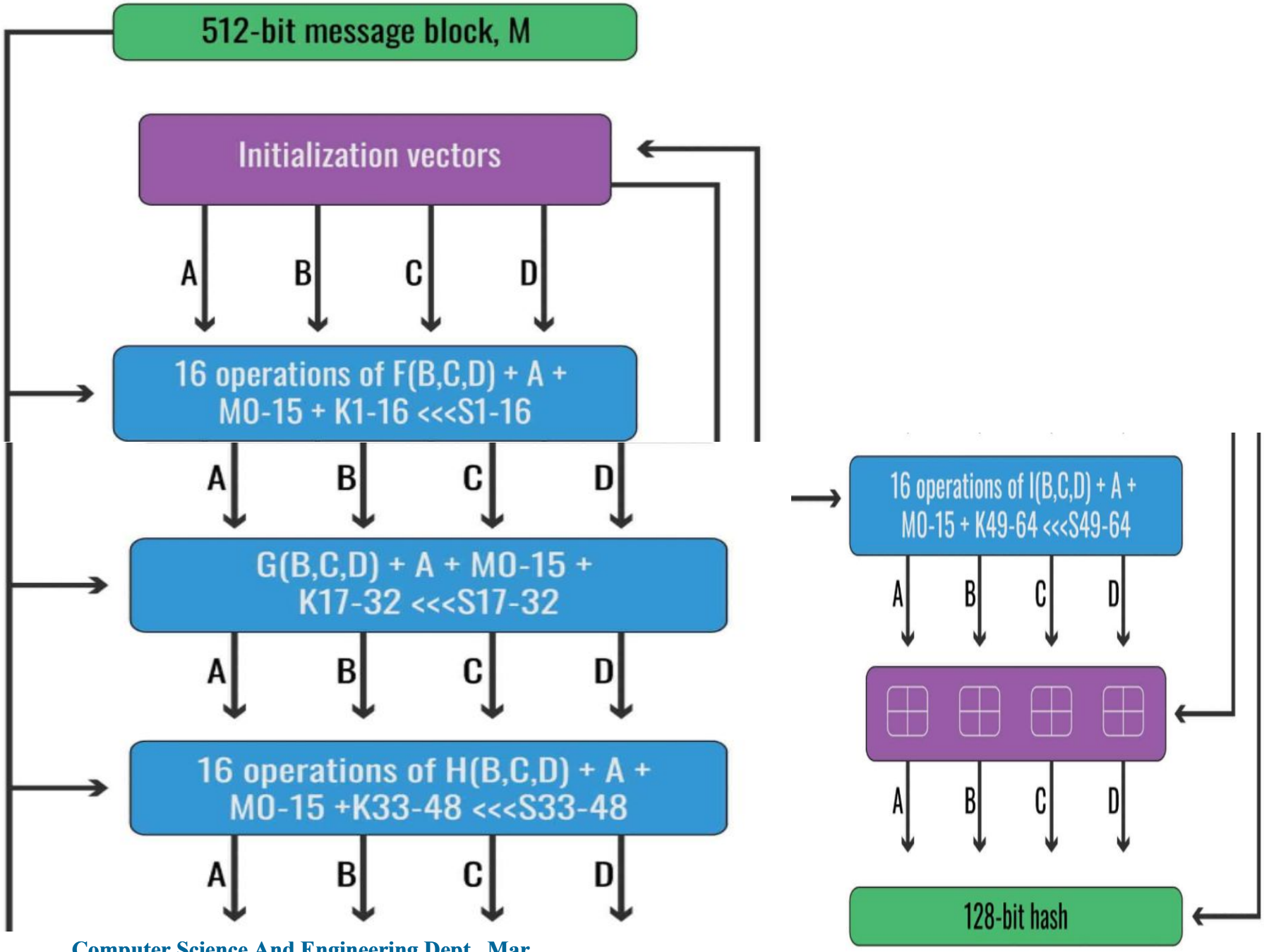


Figure 12.2 MD5 Processing of a Single 512-bit Block

- https://www.youtube.com/watch?v=53O9J2J5i14


512-bit message block, M
Initialization vectors
A
B
C
D
16 operations of F(B,C,D) + A + M0-15 + K1-16 <<<S1-16
A
B
C
D
G(B,C,D) + A + M0-15 + K17-32 <<<S17-32
A
B
C
D
16 operations of H(B,C,D) + A + M0-15 +K33-48 <<<S33-48
A
B
C
D
16 operations of I(B,C,D) + A + M0-15 + K49-64 <<<S49-64
A
B
C
D
A
B
C
D
128-bit hash

- **The input M**
- At the top, we have our input, which says **512-bit message block, M**. At this stage of the diagram, it already includes all of the padding that we added in the last step. If you follow the arrow down, you will see that it enters each of the four "16 operations of…" rectangles. **Each of these four rectangles are called rounds**, and each of them are composed of a series of sixteen operations
- This means that **our input, M, is an input in each of these four stages**. However, before it can be used as an input, **our 512-bit M needs to be split into sixteen 32-bit "words".** Each of these words is assigned its own number, ranging from M0 to M15. In our example, these 16 words are

Each of these sixteen values act as inputs to the complex set of operations that are represented by each "16 operations of…" rectangle. **Once again, these four "16 operations of…" rectangles represent the four different rounds, with the one at the top representing the first round, while the lowest one is the fourth round**. While each of these M inputs are used in every single round, they are added in different orders.

In the first round, the M inputs are added into the algorithm sequentially, e.g. M0, M1, M2… M15.

In the second round, the M inputs are added in the following order:

M1, M6, M11, M0, M5, M10, M15, M4, M9, M14, M3, M8, M13, M2, M7, M12

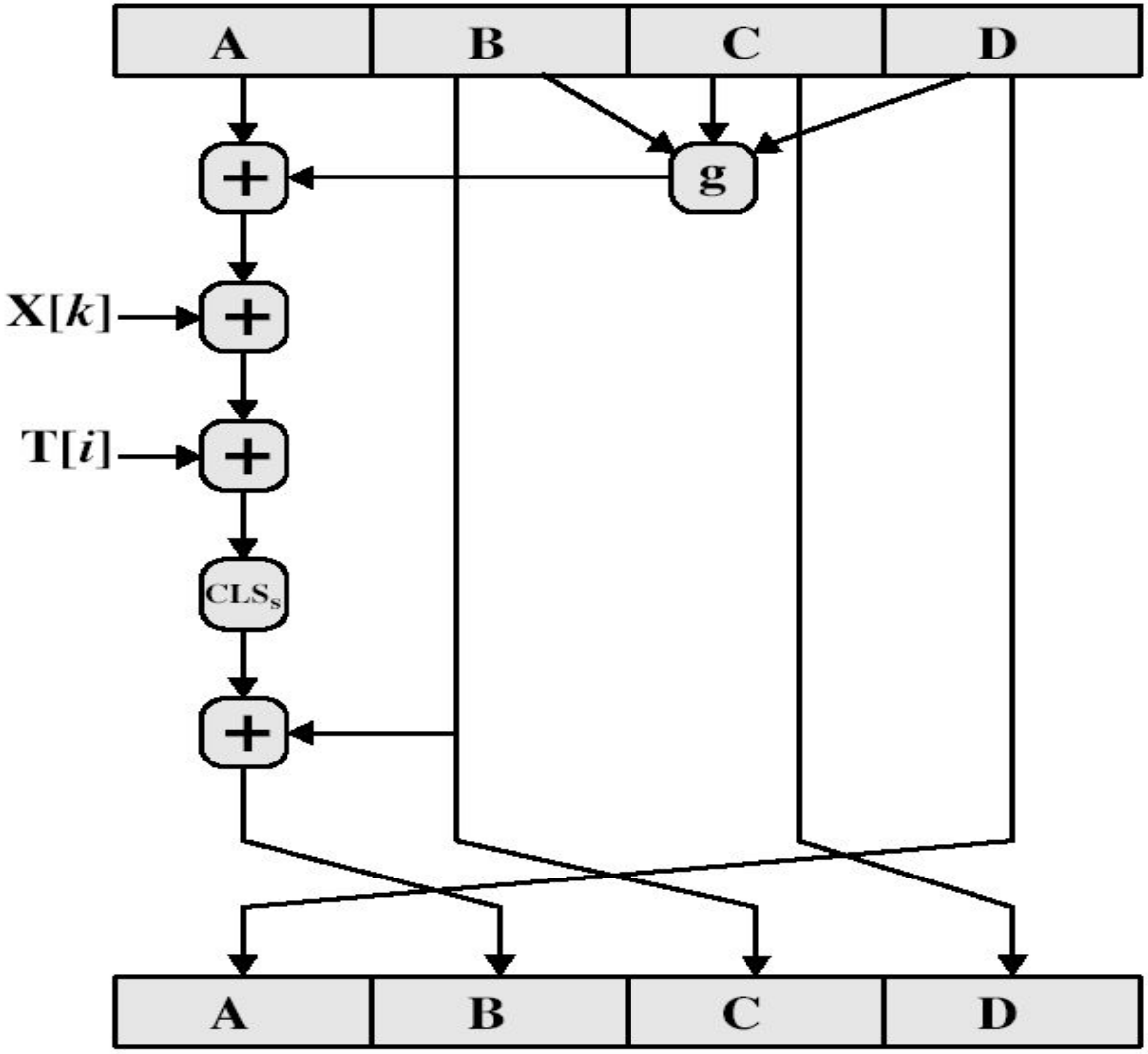
In the third round, the M inputs are added in this sequence:

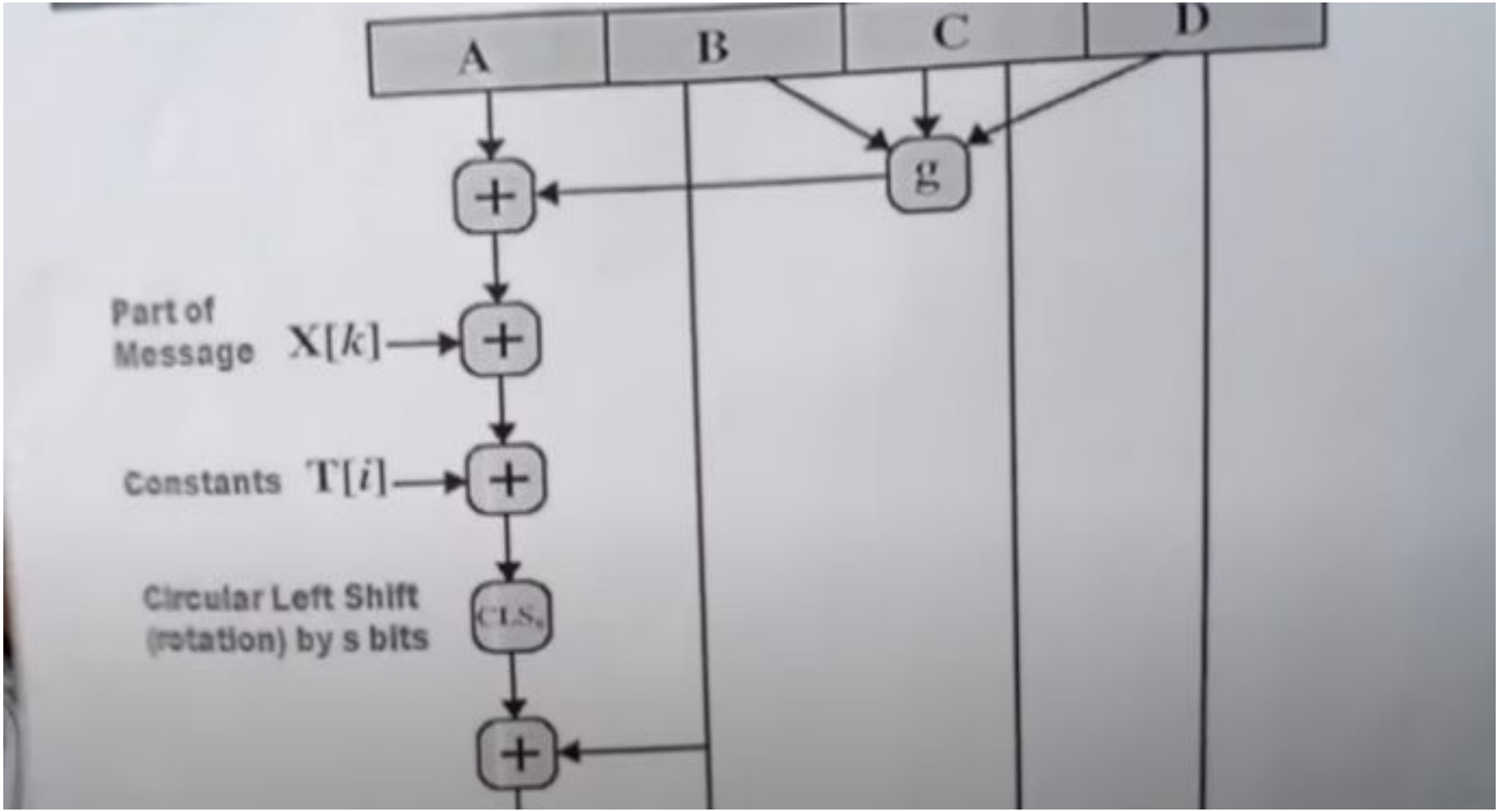
M5, M8, M11, M14, M1, M4, M7, M10, M13, M0, M3, M6, M9, M12, M15, M2

In the fourth round, the M inputs are added in the following order:

M0, M7, M14, M5, M12, M3, M10, M1, M8, M15, M6, M13, M4, M11, M2, M9

MD5 Compression Function




A
B
C
D
g
+
+
+
+
Part of Message X[k]
Constants T[i]
Circular Left Shift (rotation) by s bits
CLS

- When we zoom in on each “16 operations of…” rectangle, you can see that the arrows from B, C and D point to a box labelled g. This represents function g(B, C, D) – note that in the other three rounds, the g function is replaced by the G, H and I functions, respectively.
- If you follow the arrow out of it into the next box, this indicates that **the output of g(B, C, D) is added to the initialization vector A**, with a special type of addition. In the first operation, initialization vector A’s value is 01234567, but it changes in subsequent operations.

MD5 Compression Function

- 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}

Primitive Function g

- The function

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

X[k]

Round	X[k]
1	
2	
3	
4	

T[i]

(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] = F57COFAF	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

```
/* 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 */
```

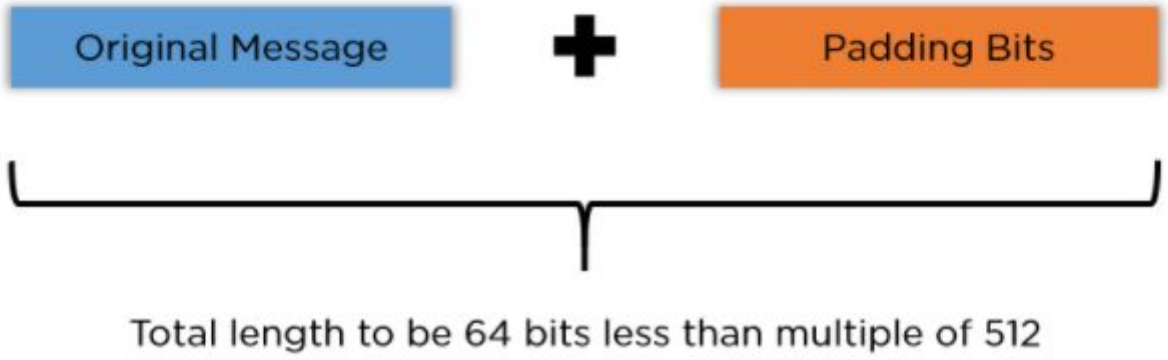

Figure 12.4 Basic MD5 Update Algorithm (RFC 1321)

Steps in MD5 Algorithm

There are four major sections of the algorithm:

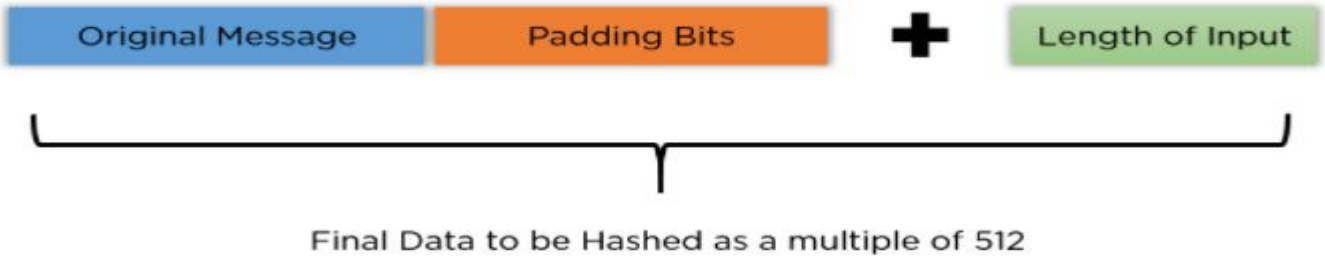
Padding Bits

When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one(1) first, followed by zeroes to round out the extra characters.



Padding Length

You need to add a few more characters to make your final string a multiple of 512. To do so, take the length of the initial input and express it in the form of 64 bits. On combining the two, the final string is ready to be hashed.



Initialize MD Buffer

The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each and are initialized as follows:

A = 01 23 45 67

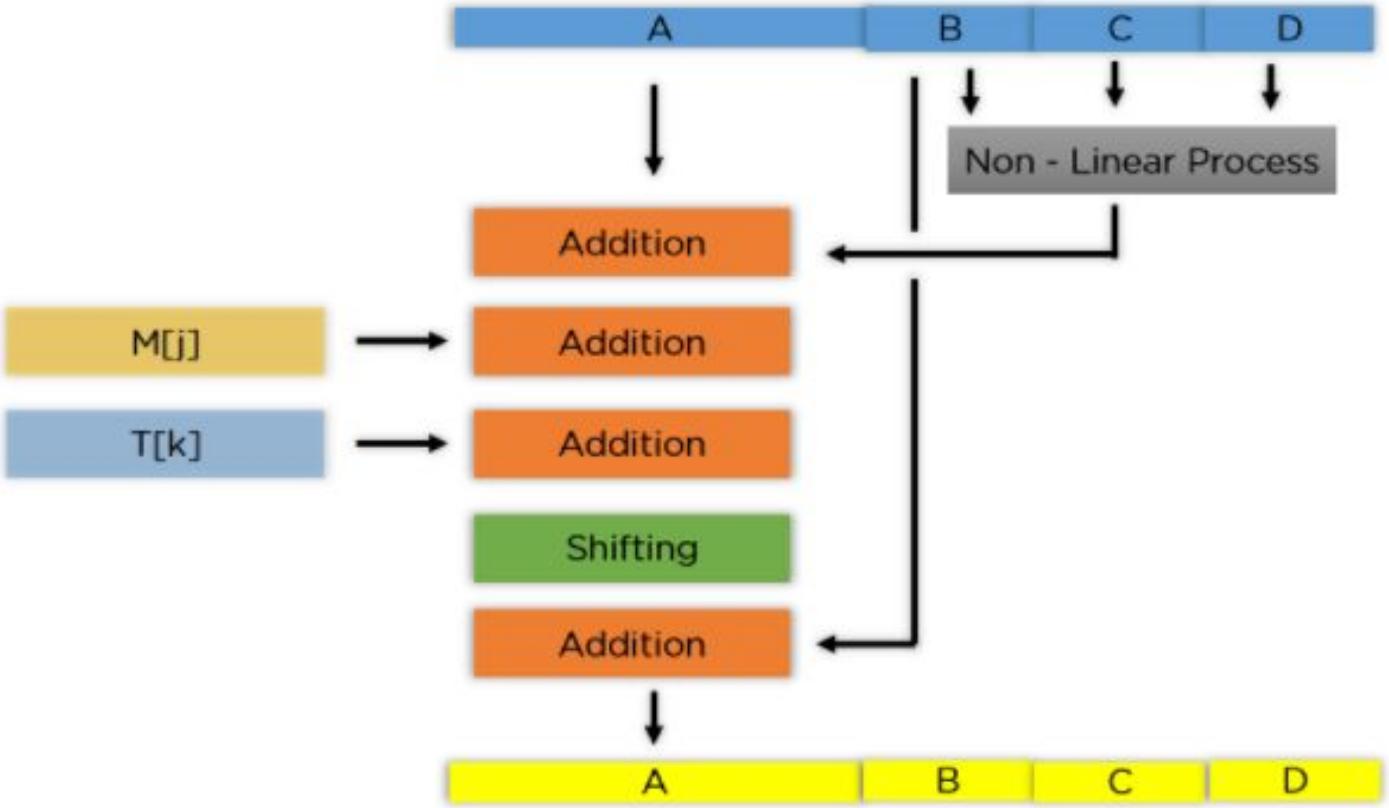
B = 89 ab cd ef

C = fe dc ba 98

D = 76 54 32 10

This constant array can be denoted as T[1] -> T[64].

Each of the sub-blocks are denoted as M[0] -> M[15].



Strength of MD5

- Every hash bit is dependent on all message bits
- The complex repetition of the basic functions(F,G,H,I) produces results that are well mixed.
- Security is as good as possible for a 128 bit hash
  - Given a hash, find a message: $O(2^{128})$ operations

- However MD5 is now vulnerable
  - Brute-force search now considered possible

Secure Hash Algorithm (SHA-1)

Illustrate the working of SHA-1 with diagrams. (9)

Secure Hash Algorithm (SHA-1)

- SHA was designed by NIST, and published as FIPS 180 in 1993
- A revised version was issued in 1995 called SHA-1
- SHA-1 Produces 160-bit hash values
- Based on design of MD4 with key differences

- SHA stands for secure hashing algorithm. SHA is a modified version of MD5 and used for hashing data and **certificates**.
- A hashing algorithm shortens the input data into a smaller form that cannot be understood by using bitwise operations, modular additions, and compression functions

SHA Overview

1. Pad message so its length is 448 mod 512
2. Append a 64-bit length value to message
3. Initialise 5-word (160-bit) buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. Process message in 512-bit (16-word) chunks:
   - expand 16 words into 80 words by mixing & shifting
   - use 4 rounds of 20 step operations on message block & buffer
   - add output to input to form new buffer value
5. output hash value is the final buffer value

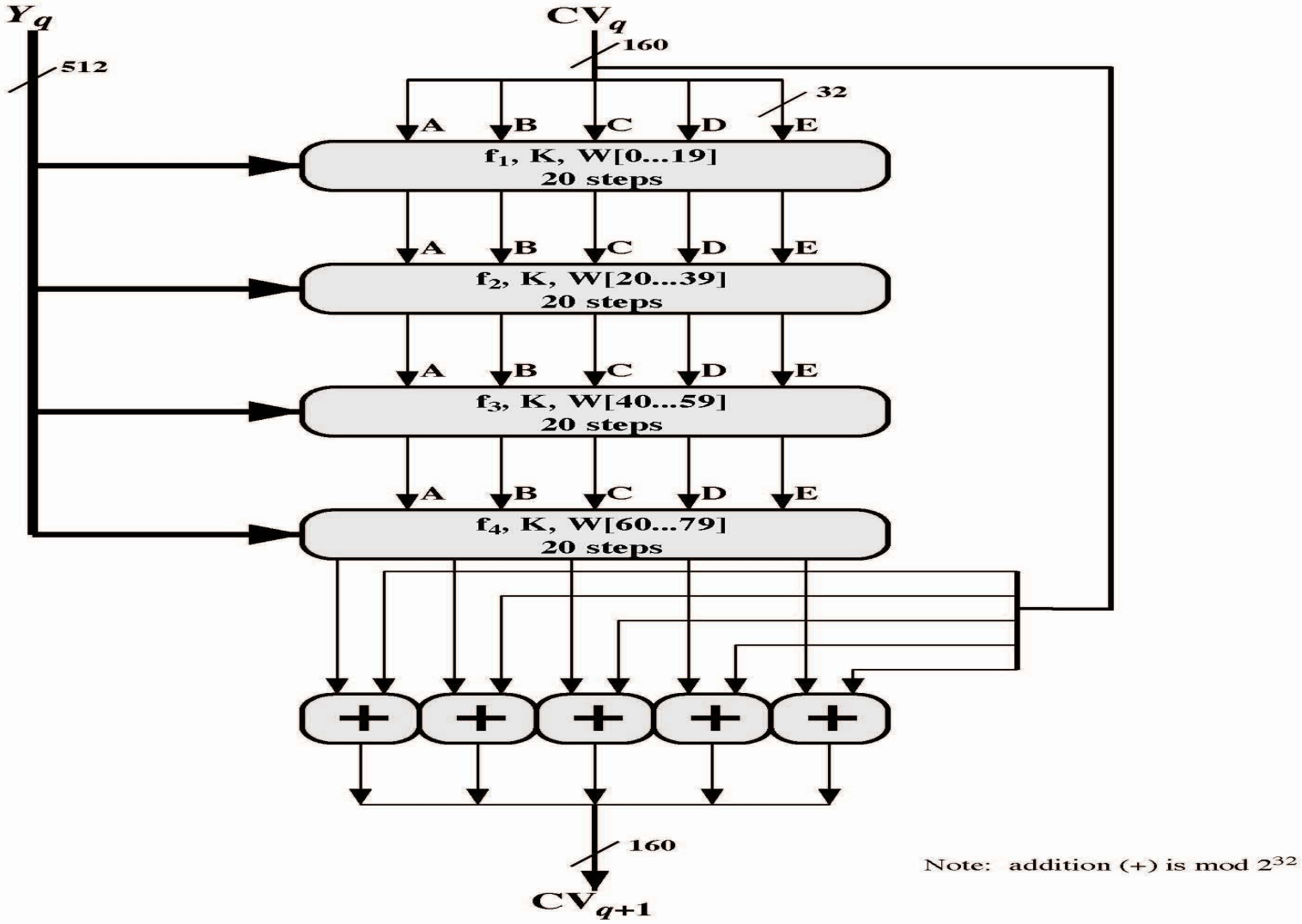
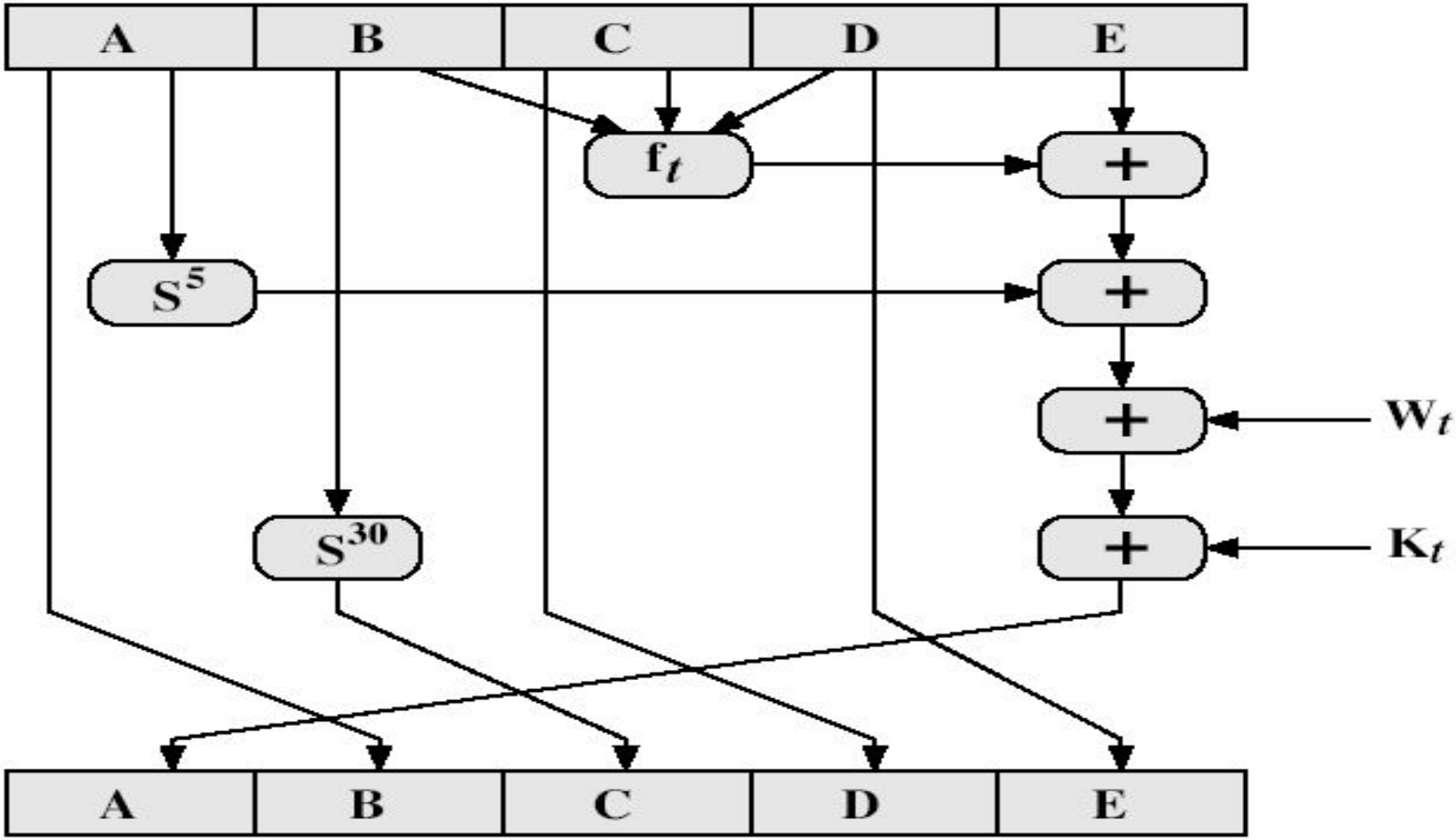


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

SHA-1 Compression Function



Logical functions for SHA-1

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

W_t

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

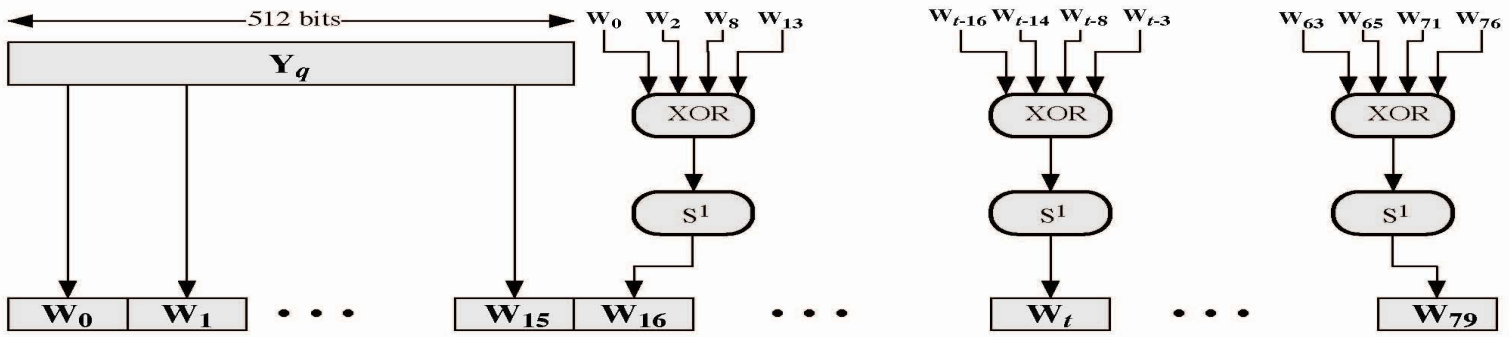


Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

K_t

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

SHA-1 Compression Function

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

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}

- SHAs are also used to hash passwords so that the server only needs to remember hashes rather than passwords.
- In this way, if an attacker steals the database containing all the hashes, they would not have direct access to all of the plaintext passwords, they would also need to find a way to crack the hashes to be able to use the passwords.
- SHAs can also work as indicators of a file's integrity. If a file has been changed in transit, the resulting hash digest created from the hash function will not match the hash digest originally created and sent by the file's owner.

SHA-1 verses MD5

- Brute force attack is harder (160 vs 128 bits for MD5)
- Not vulnerable to any known attacks (compared to MD4/5)
- A little slower than MD5 (80 vs 64 steps)
- Both designed as simple and compact
- Optimised for big endian cpu's (SUN) vs MD5 for little endian cpu's (PC)

Revised Secure Hash Standard

- NIST have issued a revision FIPS 180-2
- adds 3 additional hash algorithms
- SHA-256, SHA-384, SHA-512
  - Different lengths of hash bits
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1

Security of HASH FUNCTIONS AND MACs

Brute-Force Attack

- Hash Functions
  - **One–way** : For any given code *h*, it is computationally infeasible to find *x* such that H(*x*) = *h*.
  - **Weak Collision resistance** : For any given block x, it is computationally infeasible to find $y \neq x$ with H(*y*) = H(*x*).
  - **Strong Collision resistance** : It is computationally infeasible to find any pair (*x, y*) such that H(*x*) = H(*y*)

Brute-Force Attack

- **Message Authentication Codes**
  - **Computation resistance**: Given one or more text-MAC pairs (x_i, $C_k(x_i)$), it is computationally infeasible to compute any text-MAC pair (x, $C_k(x)$) for any new input $x \neq x_i$

Cryptanalysis

- Hash Function
  - Depends on the strength of the compression function
  - Cryptanalysis in MD5, SHA1 take more time than brute-force attack.
- MAC
  - An ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort