Msg Digest Algorithm

MD5 logic

- The algo takes variable length msg as i/p, & produces 128-bit msg digest.
- The i/p is processed in 512-bit block.

step:1 Append padding bits :→

- The msg is padded so that its length in bits is congruent to

length = 448 modulo 512

- That is length of the padded msg is 64 bits less than an integer multiple of 512 bits.
- Padding is always added even if the msg is already of desired length.

eg. if msg is 448 bits long

it is padded by 512 bits

So length is 960 bits.

- padding bits range is 1 to 512
- padding contains 1 followed by 0's

1000 . . . . 00 . . . 0000

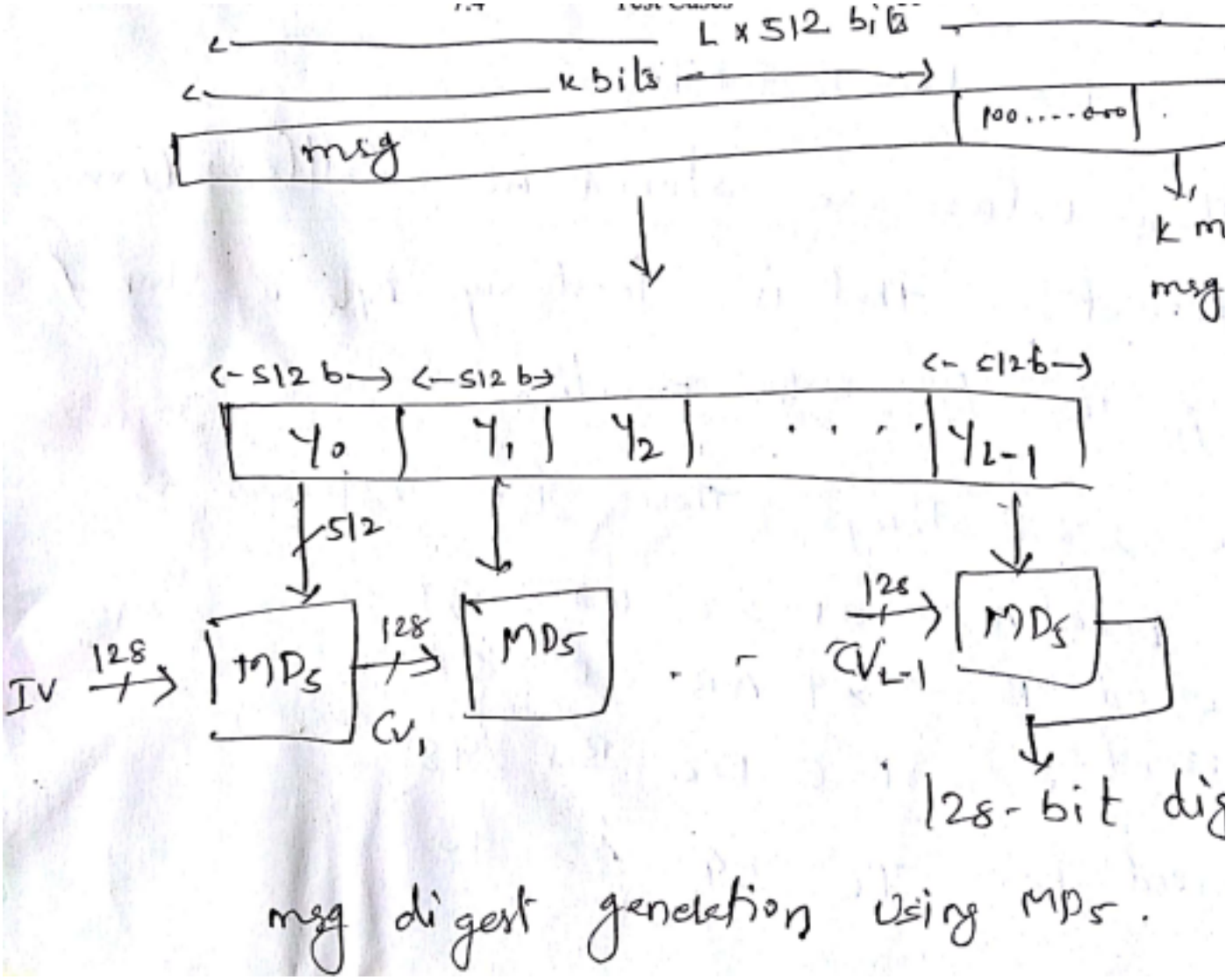
Step :2 Append length

- A 64 bit representation of the length bits of the original msg (before pad is appended to the result of step 1. (LSB first)
- If original length is greater than 2^{64} then only low order 64-bits of length used.
- Thus field contains length of original msg modulo 2^{64},

original msg length mod 2^{64}

⇒ The outcome of first 2-step yield that is an integer multiple of 512 bi in length.

⇒ The expanded msg is represented as th sequence of 512 bit blocks $Y_0, Y_1, \ldots$ so total length of $L \times 512$ bits. (L: expanded msg)



msg digest generation using MD5.

Step 3 - Initialize MD Buffer

- A 128 bit buffer is used to hold intermediate & final result of hash functio
- The buffer can be represented as 4 - 32 bits regi. A, B, C & D, 32*4 = 128 bit

→ These Registers are initiged to the following 32 bit integers (hex values)

A = 67452301

B = EFCDAB89

C = 98BADCFE

D = 10325476

These values are stored in little endian format that is least sig. Byte is stored in the low adr. position

- As 32 bit strings these IV appears as

Word A = 01 23 45 67

Word B = 89 AB CD EF

Word C = FE DC BA 98

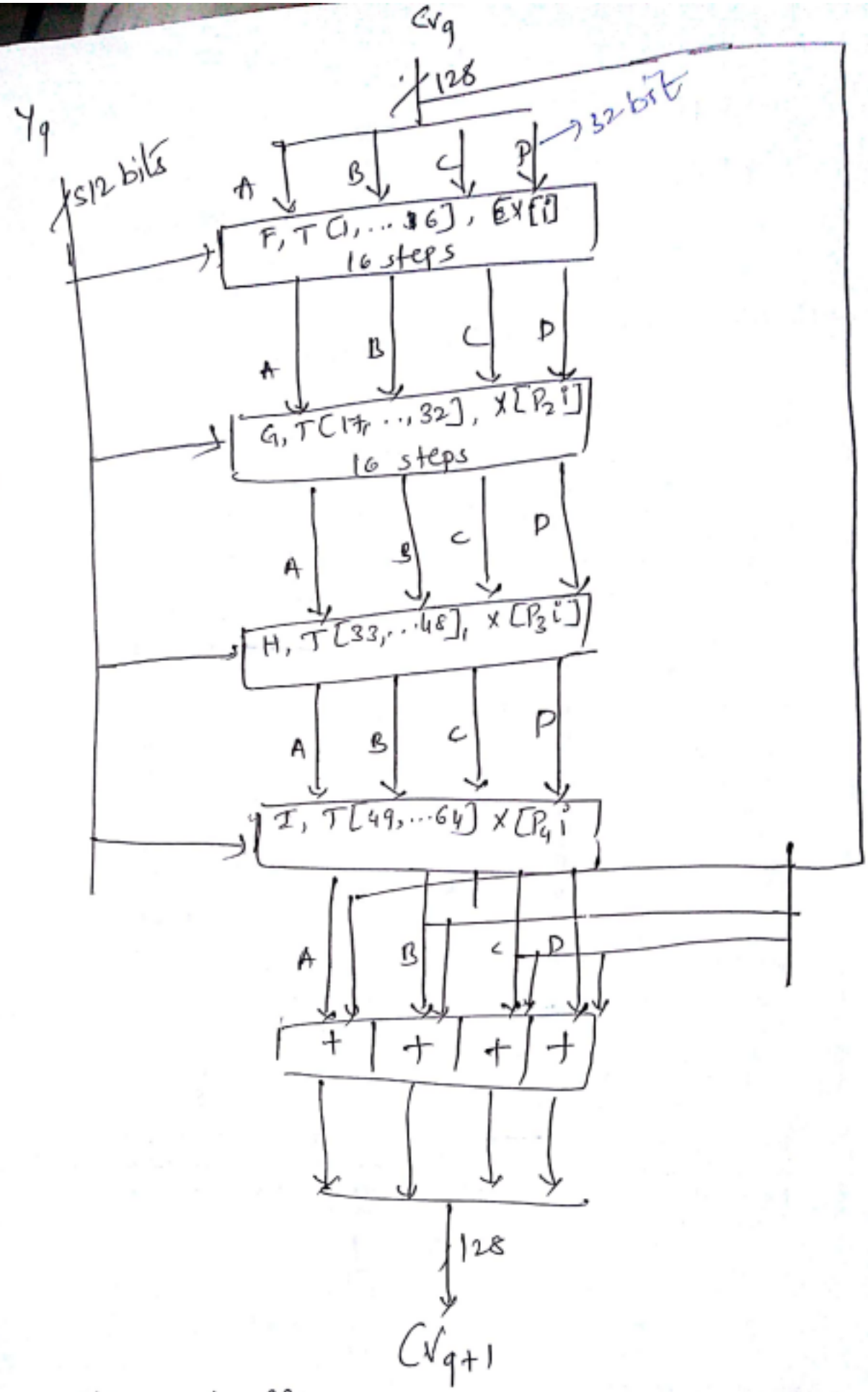
Word D = 76 54 32 10

Step 4: Process msg in 512 bits or 16 word blocks

- The heart of the algo is compression fun. that consists of 4 rounds of processing.

This module is labeled as MD5

- The four rounds has similar structure but each uses a diff. primitive logical function. referred as F, G, H, I in the specification



+ is mod 2^{32}

MD5 processing of single 512 bit block
(MD5 compression function)

IV = Initial value of ABCD buffer

Y_q = the q^{th} 512 bit block of msg.

L = no. of blocks in msg including padding & length field.

CV_q = chaining variable processed with q^{th} of msg

RFX = round function using primitive function X.

MD = final msg digest value

SUM_{32} = Addition modulo 2^{32} perfo separately on each word of pair of i/p.

→ each round takes 512 bit block being processed, & 128-bit buffe value ABCD & updates content buffer.

- Each round also makes use of one 4th ($\frac{1}{4}^{th}$) of 64 element table

T [1 . . . 64] ,

T is constructed from Sine function

The i^{th} element of T, i.e T[i], has a value equal to the integer part of $2^{32} \times abs(sine(i))$

where i is in radians

→ because abs (sin(i)) is a number bet^n 0 & 1

- The table provides randomized set of 32 bit patterns. which should eliminates any irregularities in the i/p data

→ The o/p of 4th round is added to i/p of 1st round to produce CV_{q+1}

The addition is done independently for each of the 4 words

(+ is mod 2^{32})

Step 5

After all L 512 bits blocks have been the o/p from L^{th} stage (last stage) the 128 bit msg digest

Scanned by CamScanner

MD5 Compression function

→ detail at the logic of each 4 rounds of the processing 512-bit block.

→ Each round consist of seq of 16 steps operating on the buffer ABCD.

Each step is of the form

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

a,b,c,d = four words (1 word 32 bit) of buffer, in a specified order that varies across steps

g = one of the primitive funs F, G, H, I

<<<s = Circular left shift of 32-bit argument by s bits

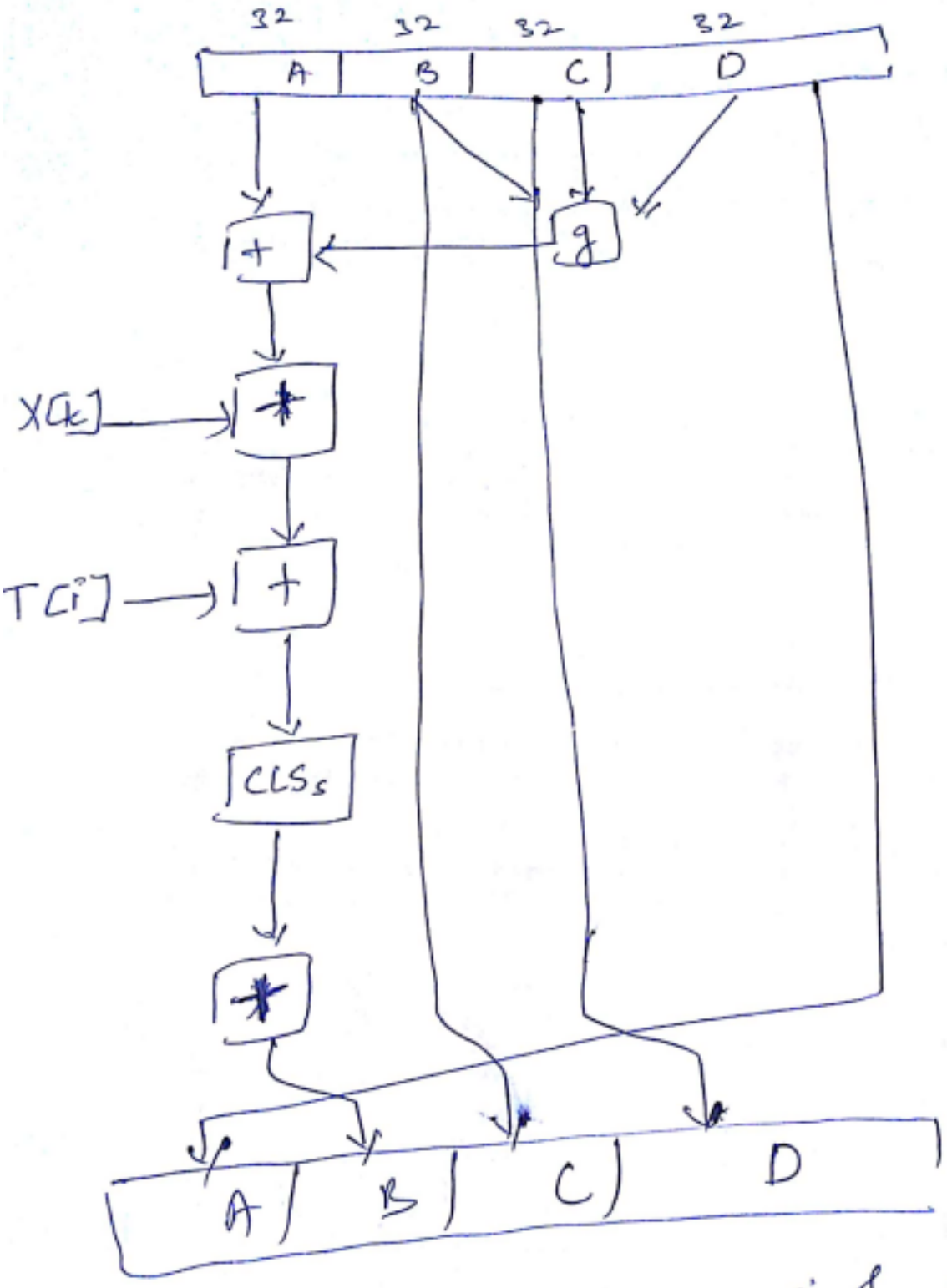
X[k] = M[q x 16 + k] ⇒ k^{th} 32 bit word in q^{th} 512 bit block of msg.

T[i] = i^{th} 32 bit word in matrix T.

+ = addition modulo 2^{32}

Steps of operation

32 x 4 = 128 bit



CLS$_s$

Circular left shift by s bits

MD5 operation single step

→ One of the four primitive logical functio
used for each ~~rounds~~ of four rounds

- each primitive function (F, H, G, I) ta
3 - 32 bit words as i/p & produces 3
- Each function performs a set of bitwise
operation;

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

$\wedge \rightarrow$ and $- \rightarrow$ not
$\vee \rightarrow$ or $\oplus \rightarrow$ XOR.

Truth table

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

16-words — 100
16 words

→ array of 32 bit words @ X.[0...15]
holds the value of the current 512 bi
block being processed.

→ within a round, each of the 16 words of X[
is used exactly once, during 1 step
the order varies from round to ro

16 bits

in first round → original word is used.
in their order.

Then the following permutations are defined
for round 2 to 4.

$P_2(i) = (1 + 5i) \bmod 16$

$P_3(i) = (5 + 3i) \bmod 16$

$P_4(i) = 7i \bmod 16$

→ element T is also used exactly once, during one step of one round.

→ 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.

The point to all these complexity is to make it very difficult to generate collisions.

(2 512-bit blocks that produces same o/p)