MCSE 568

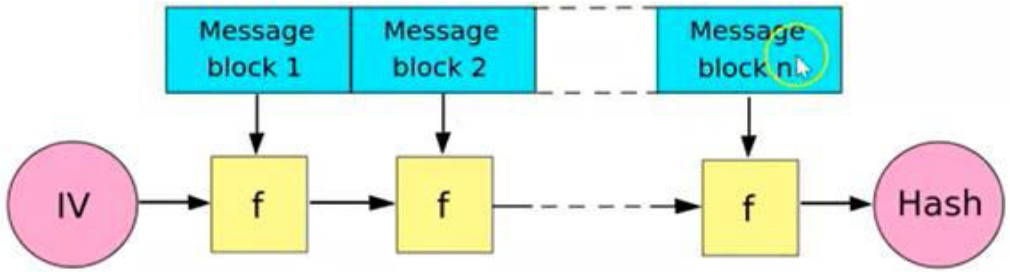
Chapter 3

Secured Hash Algorithm

Lecture 7

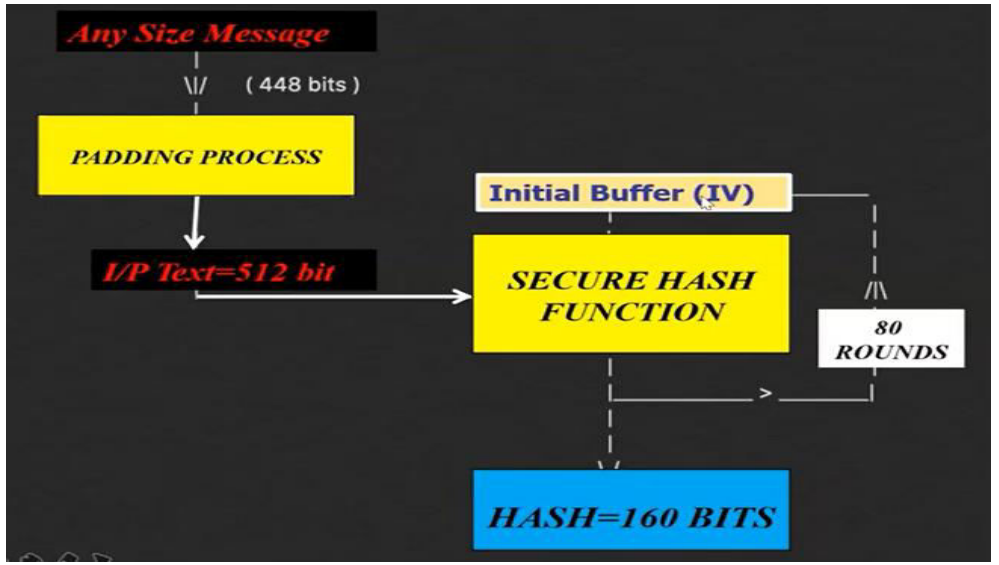
Secure Hash Algorithm (SHA-1)

A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length. The ultimate goal of this chapter is to discuss the details of the two most promising cryptographic hash algorithms—SHA-1 and MD5.

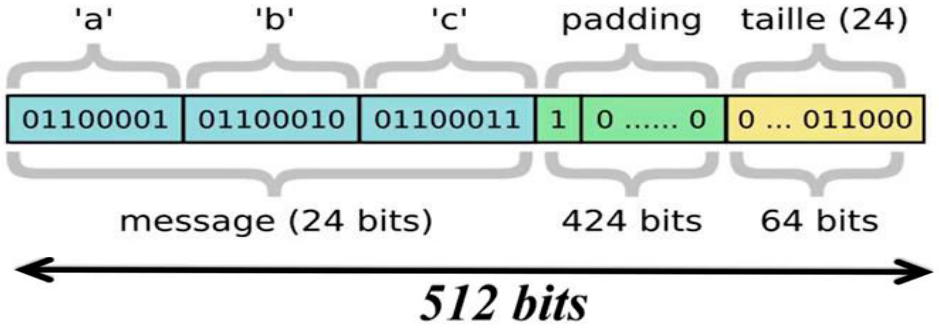


SHA-1 produces a 160-bit hash value or message digests from the input data, It uses 80 rounds of cryptographic operations to encrypt and secure a data object.

I/P Message =512 bit
Sub Block size=32 bit, 16 total Sub blocks
Initial Buffer (IV) = 5 words *32 =160 bit (A B C D E)
f= 4 Functions Rounds= 80 O/P Hash =160 bit



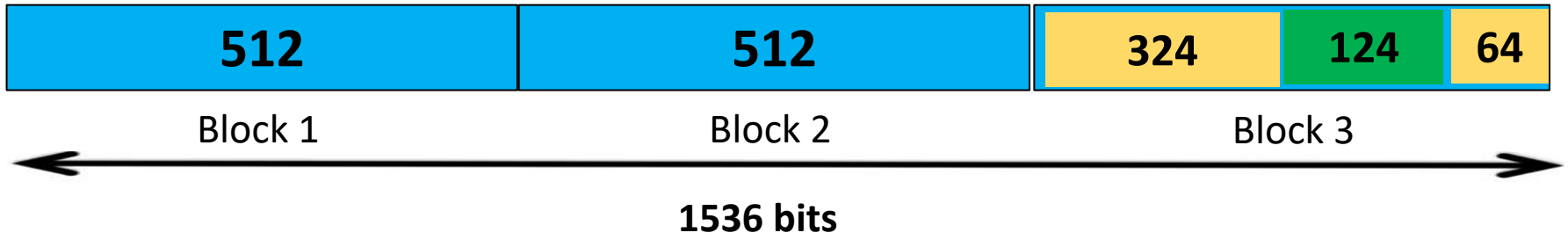
SHA message padding(512 bits) Here input='abc'



Secure Hash Algorithm (SHA-1)

Exercise on Padding

Prob: How many bits are needed to be padded with input message size=1348 bits
→Each block consists of 512 bits
→ last block needs 64 bits for store length of message
→ So, (message length) mod 512 should be (512-64)=448, Here 1348 mod 512=324
Padding required=512-324-64=124 bits
→ Total bits of message=1348+124+64=1536 bits



Secure Hash Algorithm (SHA-1)

Example Cryptography

1	C	67	01000011
2	r	114	01110010
3	y	121	01111001
4	p	112	01110000
5	t	116	01110100
6	o	111	01101111
7	g	103	01100111
8	r	114	01110010
9	a	97	01100001
10	p	112	01110000
11	h	104	01101000
12	y	121	01111001

```
01000011011100100111100101110000011101000110111101100
11101110010011000010111000001101000011110011000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000001100000
```

Msg(96)+Padding(352)+Length(64)=512 bits

Divide the 512 bits into 16 (32 bits) words

```
01000011011100100111100101110000   w[0]
01110100011011110110011101110010   w[1]
01100001011100000110100001111001   w[2]
10000000000000000000000000000000   w[3]
00000000000000000000000000000000   w[4]
00000000000000000000000000000000   w[5]
00000000000000000000000000000000   w[6]
00000000000000000000000000000000   w[7]
00000000000000000000000000000000   w[8]
00000000000000000000000000000000   w[9]
00000000000000000000000000000000   w[10]
00000000000000000000000000000000   w[11]
00000000000000000000000000000000   w[12]
00000000000000000000000000000000   w[13]
00000000000000000000000000000000   w[14]
00000000000000000000000001100000   W[15]
```

- For iterations 16 through 79, perform the following operation:

$$W(i)=W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16)$$

Secure Hash Algorithm (SHA-1)

Example Cryptography

1	C	67	01000011
2	r	114	01110010
3	y	121	01111001
4	p	112	01110000
5	t	116	01110100
6	o	111	01101111
7	g	103	01100111
8	r	114	01110010
9	a	97	01100001
10	p	112	01110000
11	h	104	01101000
12	y	121	01111001

```
01000011011100100111100101110000011101000110111101100
11101110010011000010111000001101000011110011000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000
00000000000000000000000001100000
```

Msg(96)+Padding(352)+Length(64)=512 bits

Divide the 512 bits into 16 (32 bits) words

```
01000011011100100111100101110000  w[0]
01110100011011110110011101110010  w[1]
01100001011100000110100001111001  w[2]
10000000000000000000000000000000  w[3]
00000000000000000000000000000000  w[4]
00000000000000000000000000000000  w[5]
00000000000000000000000000000000  w[6]
00000000000000000000000000000000  w[7]
00000000000000000000000000000000  w[8]
00000000000000000000000000000000  w[9]
00000000000000000000000000000000  w[10]
00000000000000000000000000000000  w[11]
00000000000000000000000000000000  w[12]
00000000000000000000000000000000  w[13]
00000000000000000000000000000000  w[14]
00000000000000000000000001100000  W[15]
```

- For iterations 16 through 79, perform the following operation:

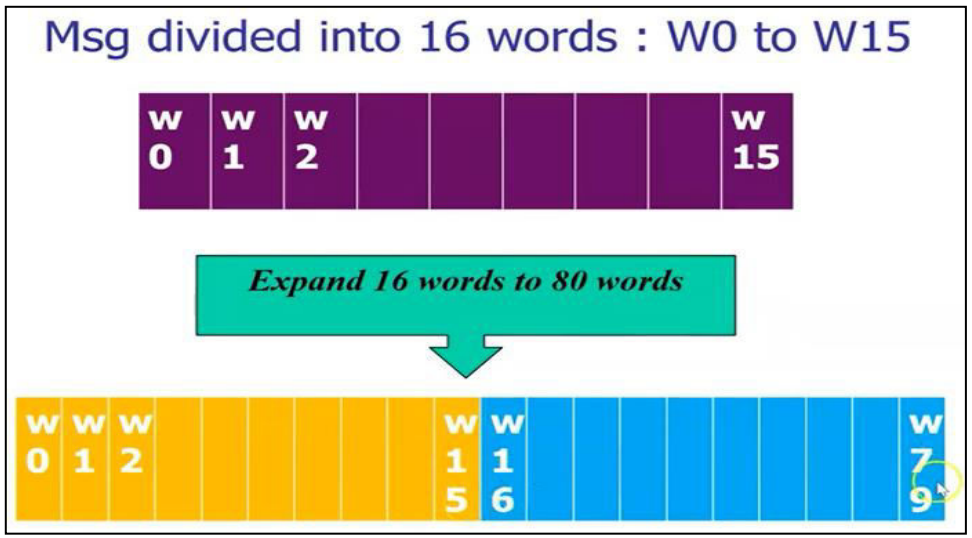
$$W(i)=W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16) \quad <<<1$$

```
w[16] = w[13] xor w[8] xor w[2] xor w[0]
```

(SHA-1: All 80 words

0: 01000011011100100111100101110000
1: 01110100011011110110011101110010
2: 01100001011100000110100001111001
3: 10000000000000000000000000000000
4: 00000000000000000000000000000000
5: 00000000000000000000000000000000
6: 00000000000000000000000000000000
7: 00000000000000000000000000000000
8: 00000000000000000000000000000000
9: 00000000000000000000000000000000
10: 00000000000000000000000000000000
11: 00000000000000000000000000000000
12: 00000000000000000000000000000000
13: 00000000000000000000000000000000
14: 00000000000000000000000000000000
15: 00000000000000000000000001100000
16: 01000100000001000010001000010010
17: 11101000110111101100111011100101
18: 11000010111000001101000000110010
19: 10001000000010000100010000100101
20: 11010001101111011001110111001011
21: 10000101110000011010000001100101
22: 00010000000100001000100001001011
23: 10100011011110110011101101010111
24: 10000011100010110000010011101111
25: 11110001100111001000110101011101
26: 11000011001101111101011011001010
27: 00010111000001101000000110010100
28: 01000000010000100010000100101100
29: 10001101111011001110110110011110
30: 10000110001001000101011110011010
31: 00010111110011111010100111111100
32: 00010001000001100011011010100011
32: 00010001000001100011011010100011
33: 00101110110111001010000000001110
34: 10001111010010100110010110011111
35: 00010111100100101010011011101110
36: 01011110011001110010100101000101
37: 01001000001110000010011001100110
38: 00000100010110101111101110100000
39: 00110110100111100110110101100110
40: 00110011000001011000010111000000
41: 10011000001110001010111011001111
42: 01110101010000111111111000111110
43: 01111100111110101001111001001001
44: 00000000011100111110001001111000
45: 01001110101100010011100001110101
46: 11011111000001000000100110100001
47: 00011111111111010000110111011000
48: 11000111111100011101110100010011
49: 11111100111001010100001100011100
50: 01110111001001110111111001111001
51: 11001001010000111000011110100101
52: 01001101010101101110011100000011
53: 10001110011000000001101000011000
54: 01000010001100011110000011001000
55: 11111000000110100101001011100101
56: 00011111101011110111100111101010
57: 10110100001011010010011010100100
58: 11110100000110100110000110110101
59: 11001001010011101011000011100111
60: 01001100000110000101010011111100
61: 01010110011011001001110000000000
62: 00100111000101010000100100111011
63: 10101110001101001001000110111010
64: 11110010001010101000110100000001
65: 01001101001111011101011001001101
66: 11000000101111101101001011101010
67: 11111000100011110100000010110110
68: 00011100100001010000101011110100
69: 11000001010100000000110000101111
70: 00000100000001001101000010101111
71: 11111101000011011101111000011111
72: 10110001100111110011001011100011
73: 01101000101011111100001010100010
74: 00001011011000100111001101111001
75: 10101100011001001011110101100101
76: 00111110010011110010101100100010
77: 01100100110101001110010111011000
78: 11111010101001010011000000000000
79: 01000000100101110110010110010100

Msg divided into 16 words : W0 to W15



Secure Hash Algorithm (SHA-1)

Initialise Buffer

5-words (160-bit) buffer (A,B,C,D,E)

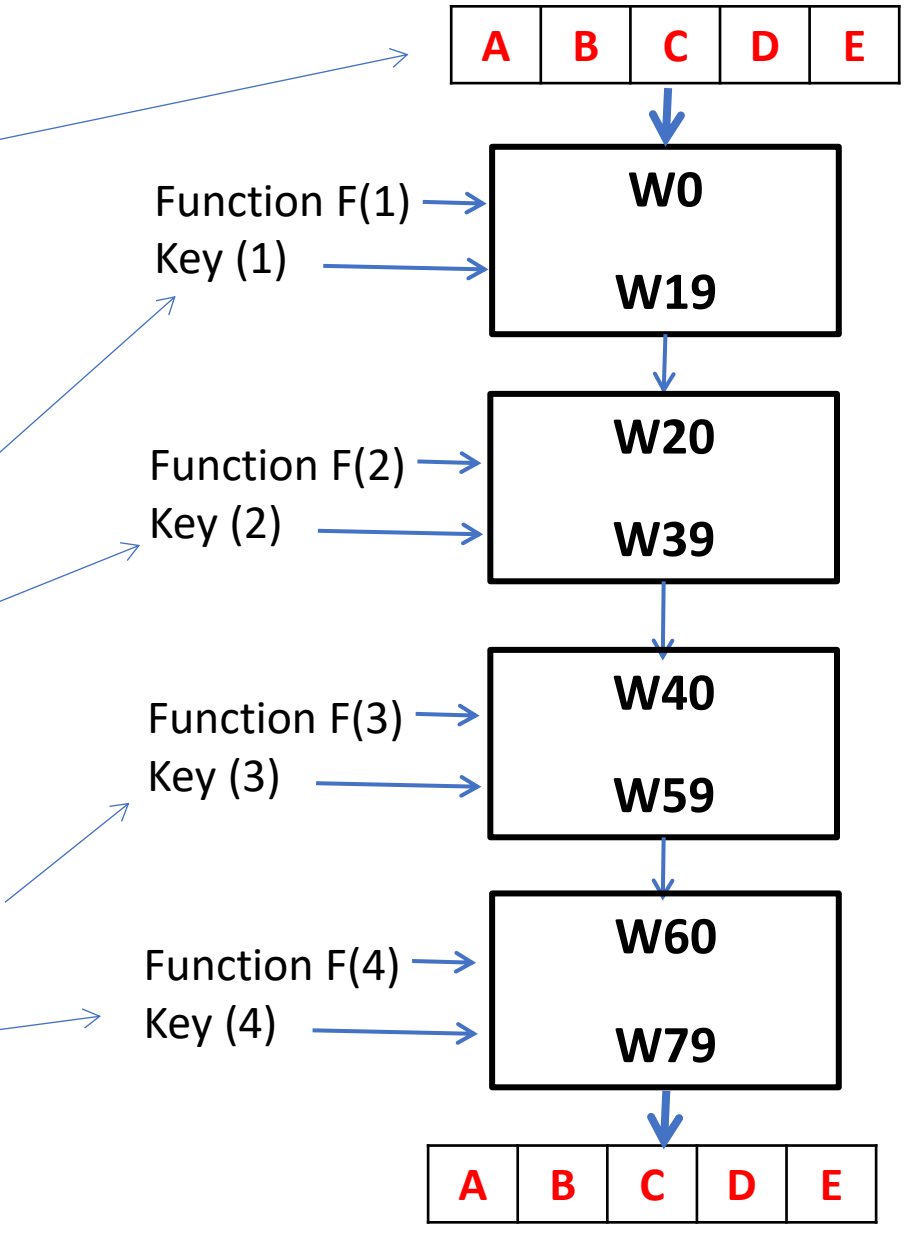
- ***A***=01 23 45 67
- ***B***=89 AB CD EF
- ***C***=FE DC BA 98
- ***D***=76 54 32 10
- ***E***=C3 D2 E1 F0

```
Rounds 1-20
f(1) = (B and C) or ((not B) and D)
k(1) = 0x5A827999
```

```
Rounds 21-40
f(2) = B xor C xor D
k(2) = 0x6ED9EBA1
```

```
Rounds 41-60
f(3) = (B and C) or (B and D) or (C and D
k(3) = 0x8F1BBCDC
```

```
Rounds 61-80
f(4) = B xor C xor D
k(4) = 0xCA62C1D6
```

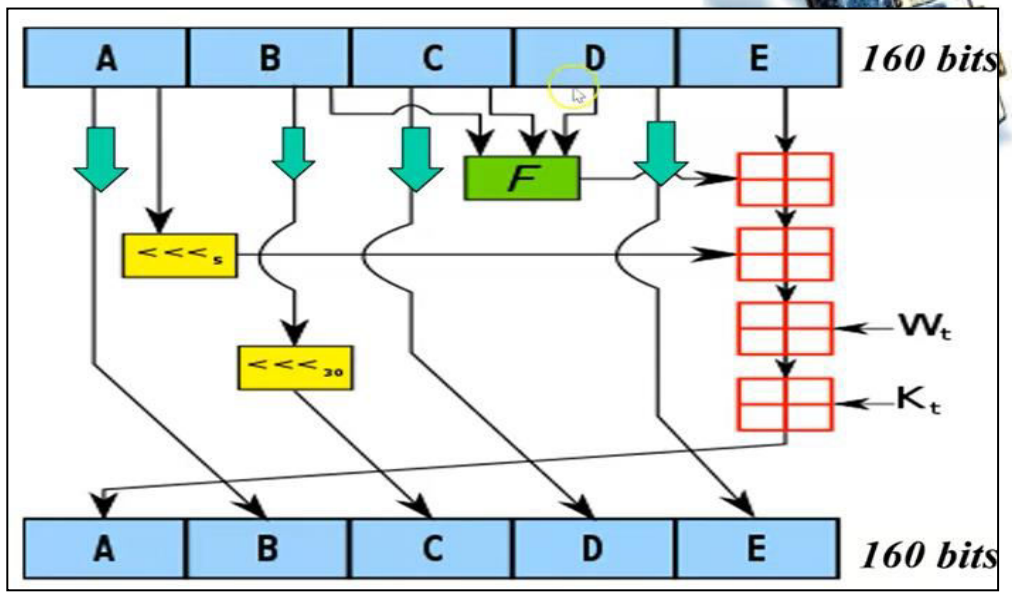


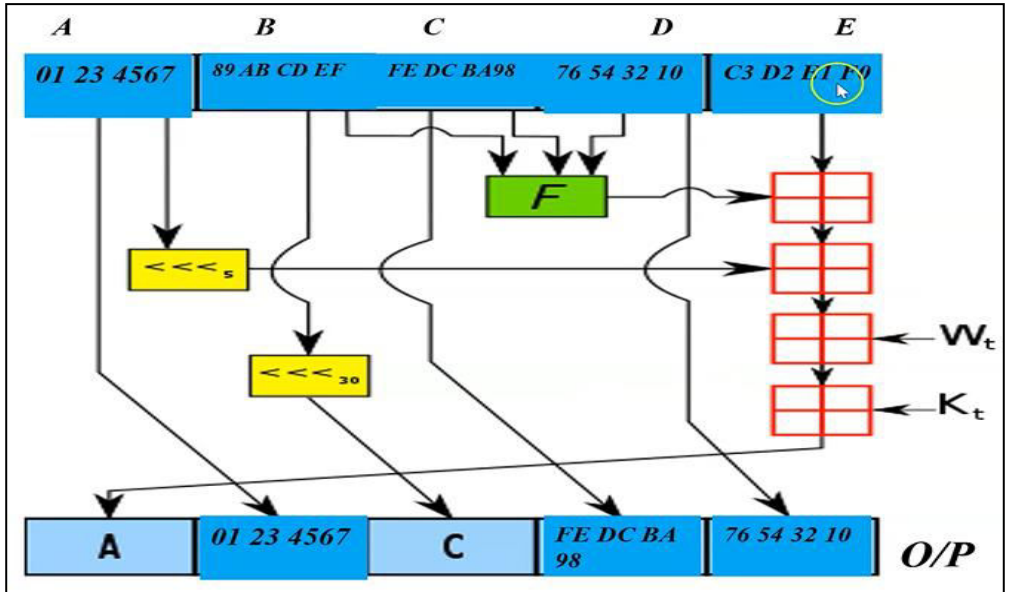
Message Digest

Secure Hash Algorithm (SHA-1)

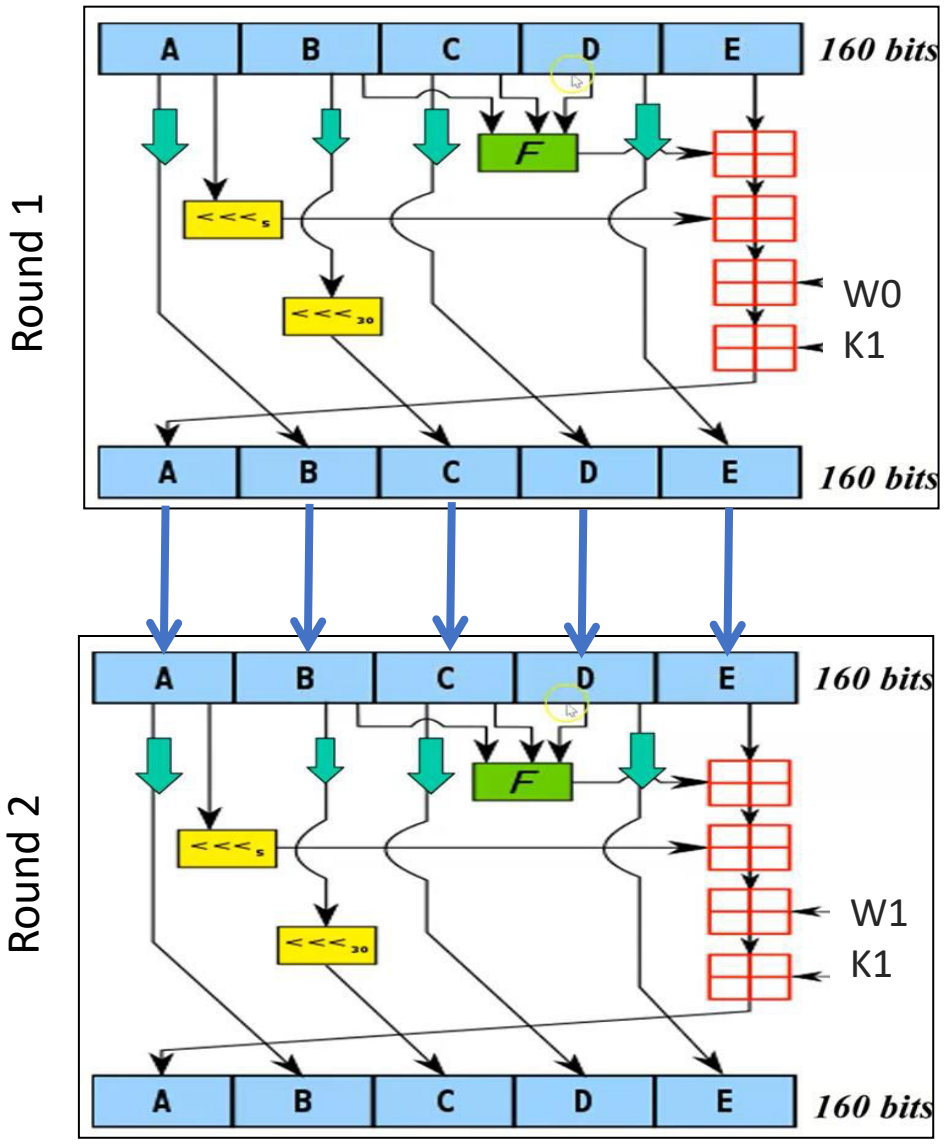
Pseudo Code….

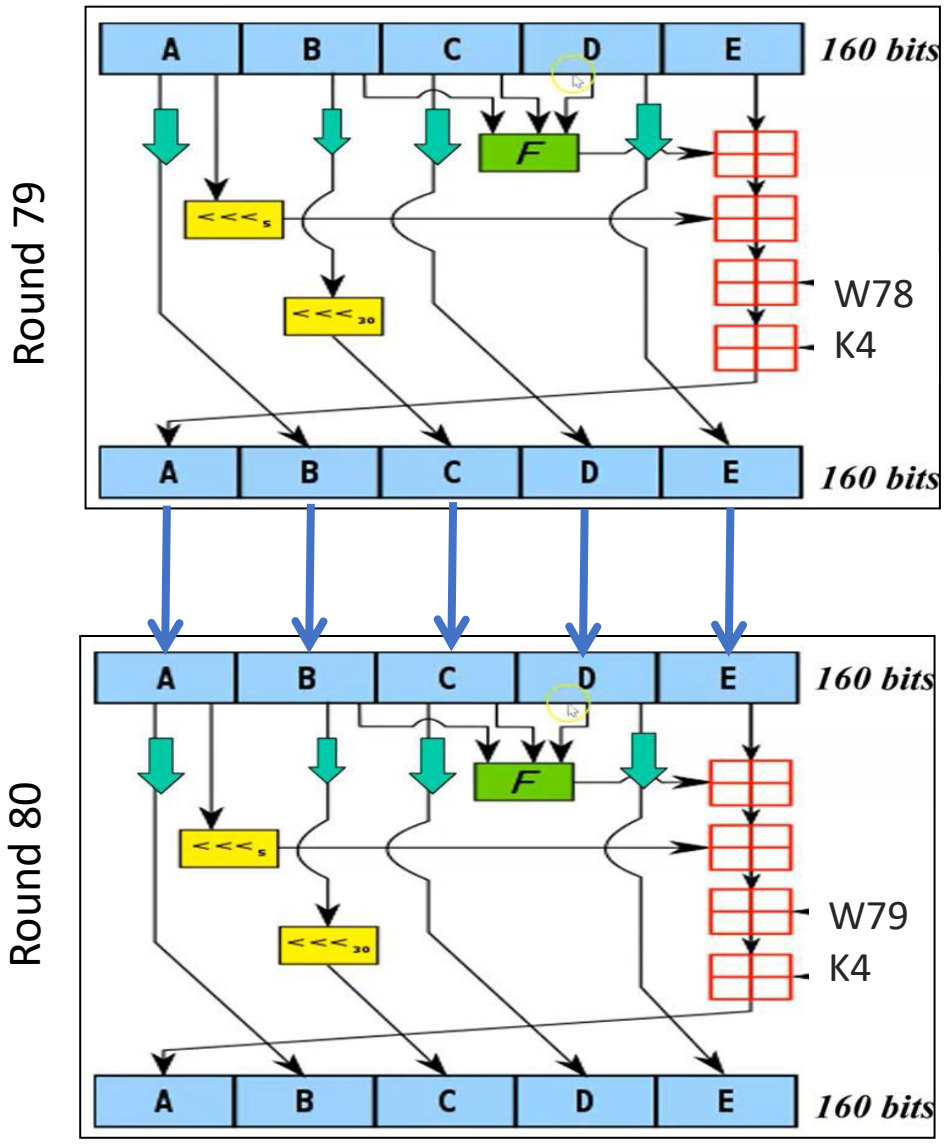
```
/* Divide the message into 16 words */
    (W(0),W(1),...,W(15))
For t = 16 to 79 do:
    W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1
End of for loop
    A = H0, B = H1, C = H2, D = H3, E = H4
For t = 0 to 79 do:
    if 0 ≤ t≤ 19 then f = (b and c) or ((not b) and d) k = 0x5A827999
    else if 20 ≤ t ≤ 39 f = b xor c xor d k = 0x6ED9EBA1
    else if 40 ≤ t ≤ 59 f = (b and c) or (b and d) or (c and d) k = 0x8F1BBCDC
    else if 60 ≤ i ≤ 79 f = b xor c xor d k = 0xCA62C1D6
     TEMP = A<<<5 + f(t;B,C,D) + E + W(t) + k
      E = D,  D = C,  C = B<<<30, B = A, A = TEMP
  End of for loop
//for next block
 H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E
```





Secure Hash Algorithm (SHA-1)

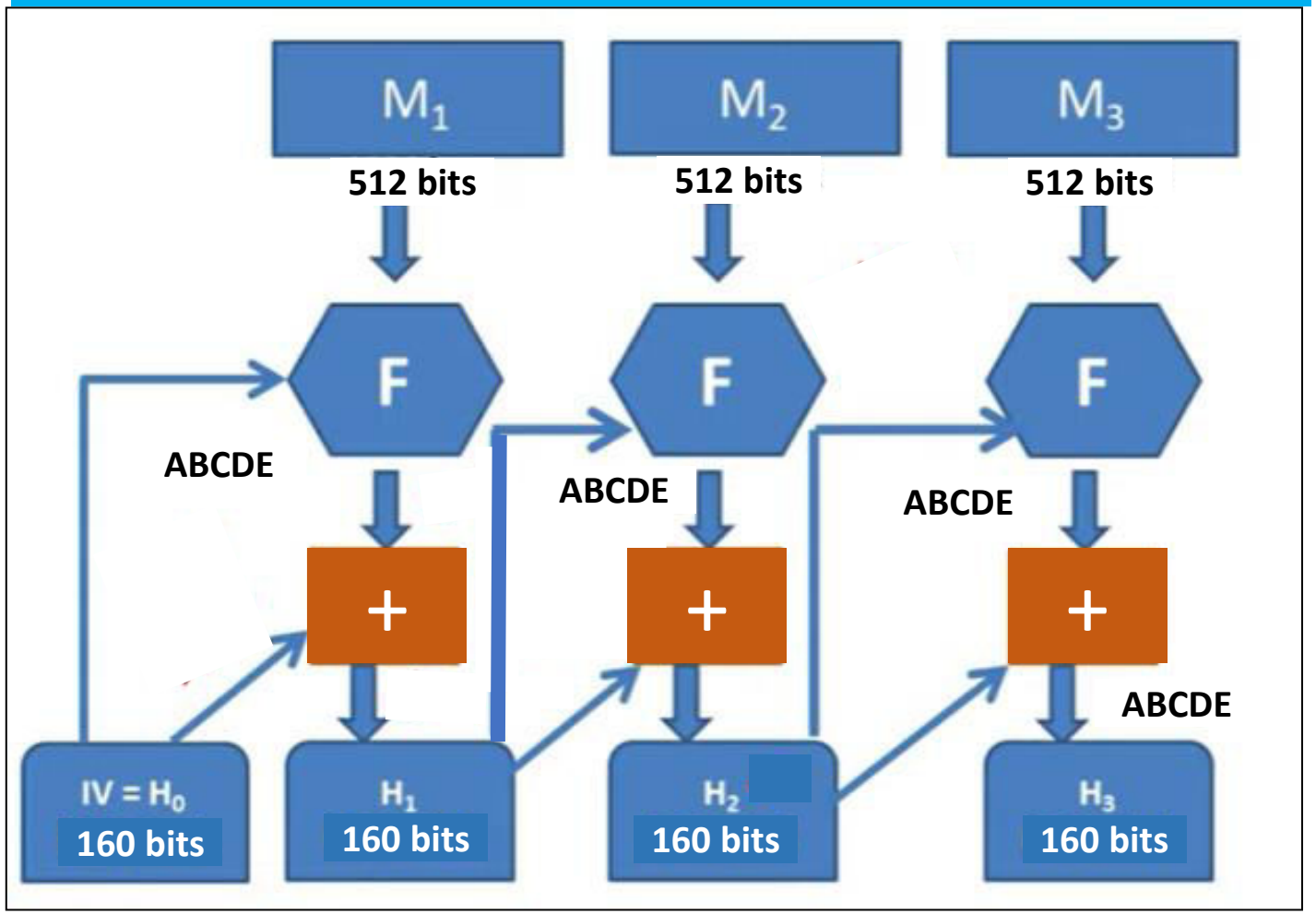




Final Hash function

Secure Hash Algorithm (SHA-1)

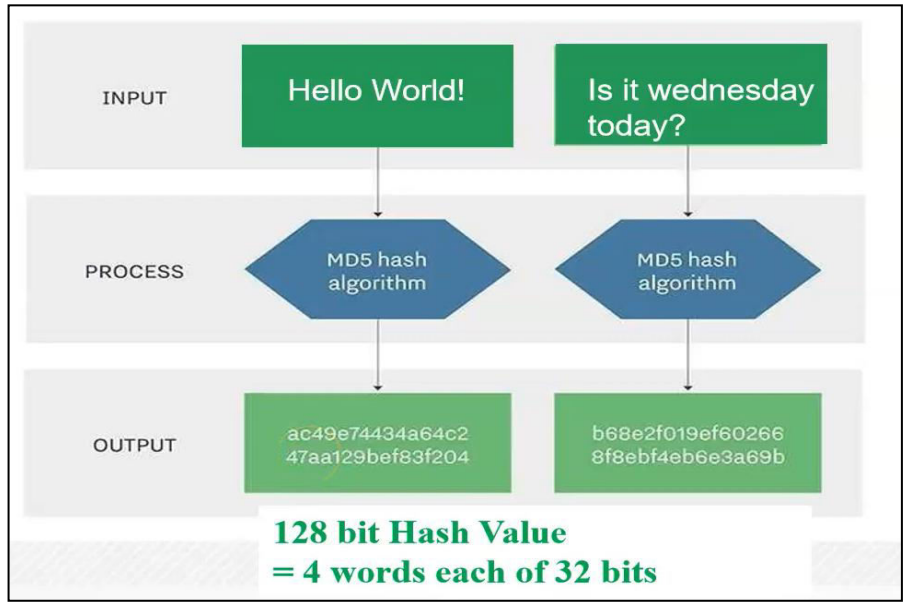
Overall Block Diagram

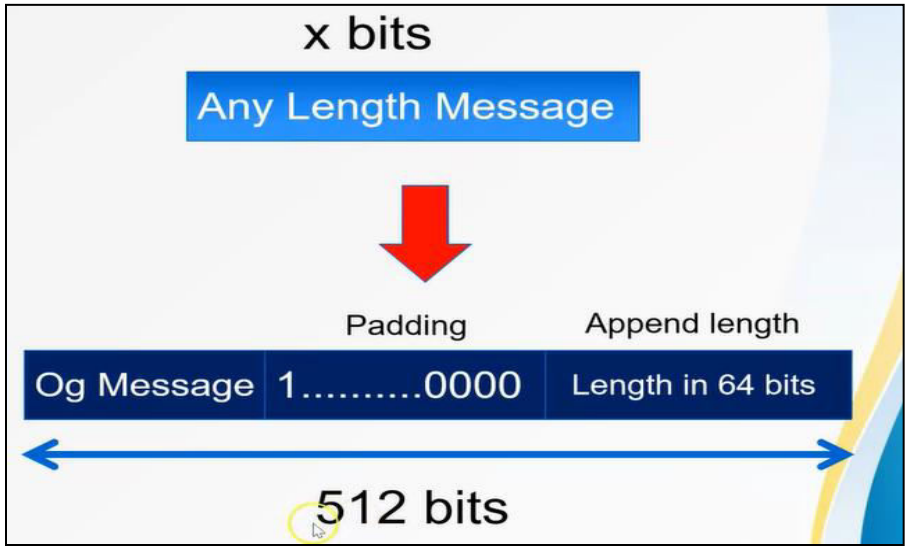


Message Diagest 5(MD-5)

What is MD5?

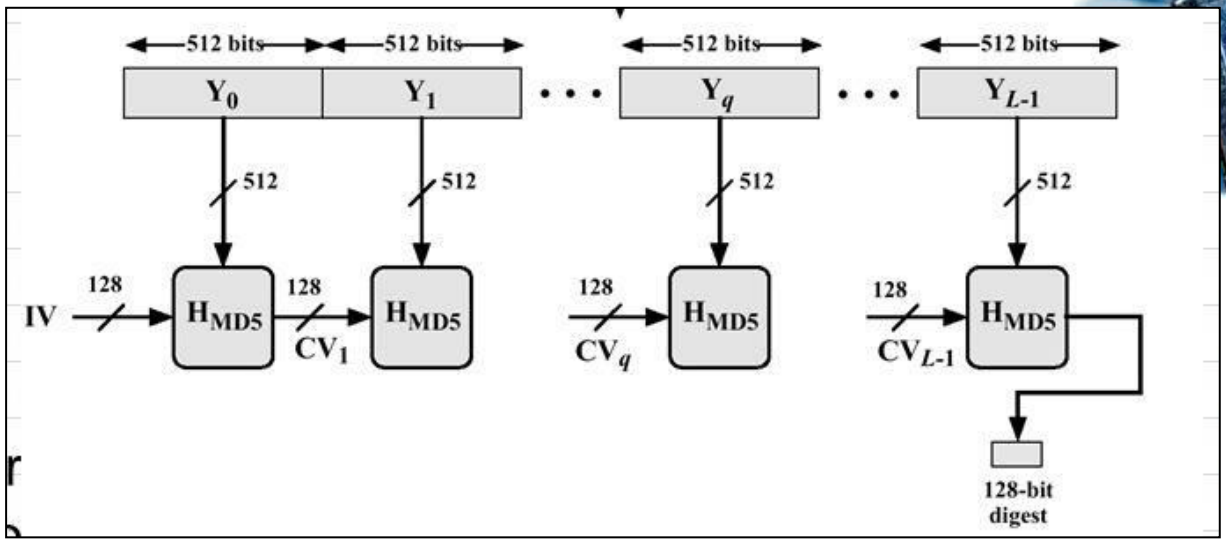
The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value (128-bit) to be used for authenticating the original message.





Message Diagest 5(MD-5): Implementation

- 128 bit message digest developed by Ron Rivest. This algorithm takes the input length of arbitrary length and 128-bit message digest is produced.
- The input message is 512-bit blocks. Figure shows processing of message to produce message digest.



Step 1: Append Padding Bits:

- The message is padded to make the length of message is 448 mod 512. 64 bits is padded with 448 bits and convert into multiple of 512 bit.
- The padding message consists a single 1-bit followed by 0 bits. The length of padding bits is in between 1 to 512.

Step 2: Append Length:

- 64 bit of original message is appended to the result of above step 1.
- It is appended such that least significant bytes to most significant byte.
- The output of step 2 yields a message of integer multiple of 512 bits.
- As M_0, M_1,..., M_q,..., M_{L-1}. The total length of expended message is L * 512 bits.

Message Diagest 5(MD-5): Implementation

Step 3: Initialize MD Buffer:

- A 128-bit buffer is used to store the intermediate as well as final result. A buffer is represented as four 32-bit registers as four 32-bit registers as P, Q, R, S.

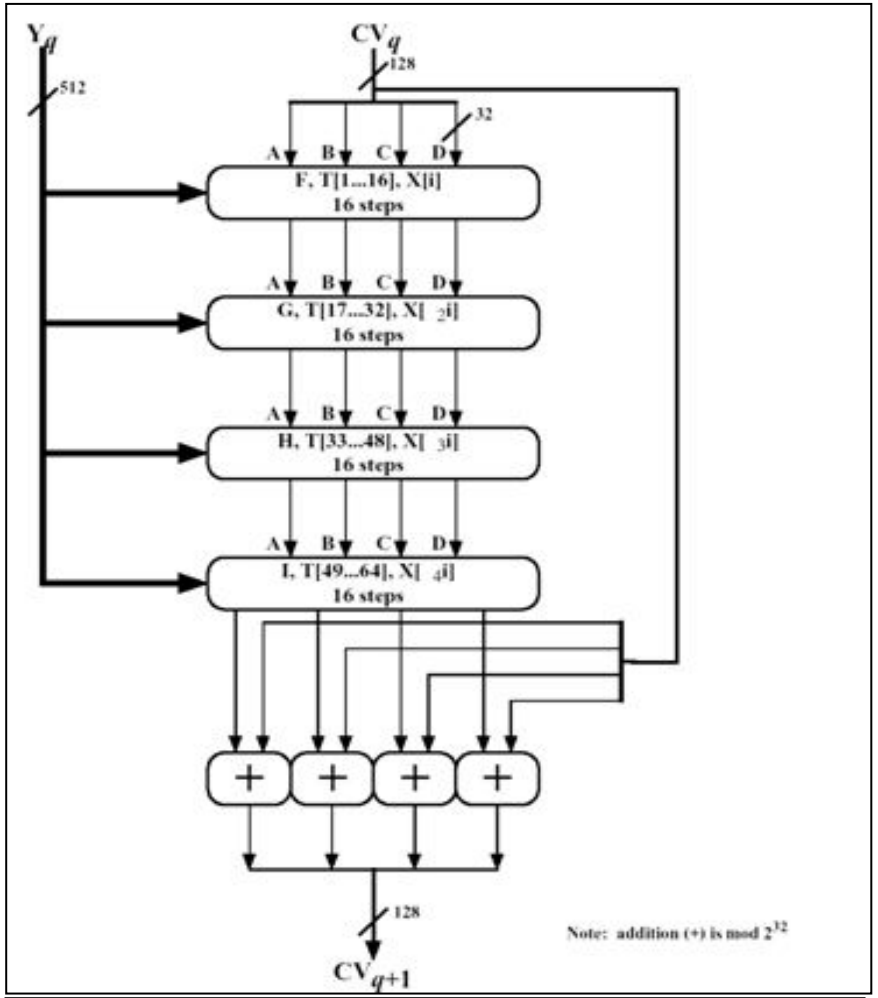
  P = 01 23 45 67
  Q = 89 AB CD EF
  R = FE DC BA 98
  S = 76 54 32 10

- It is used an initial values (IV).

Step 4: Process Message in 512-bit blocks:

- It consists of four rounds of processing as shown in figure. These four rounds have similar structure as SHA but differ in primitive logical function referred as A, B, C, D.

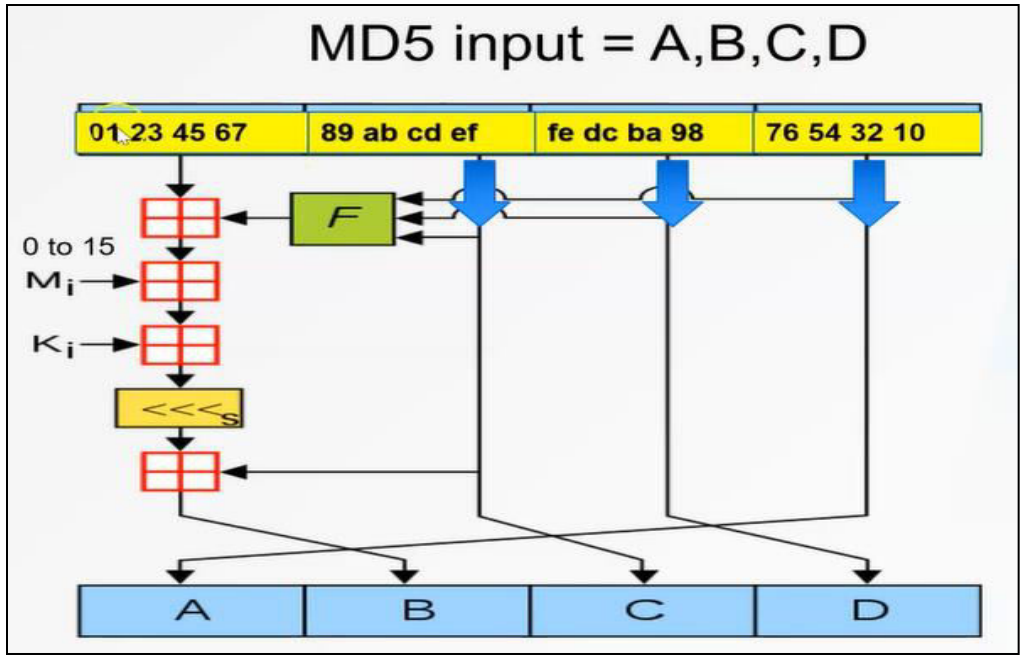
- Each round takes input 512-bit block, processed it and produces 128 bit output. The output of fourth round is added to the first round CV_q to produce CV_{q+1}.



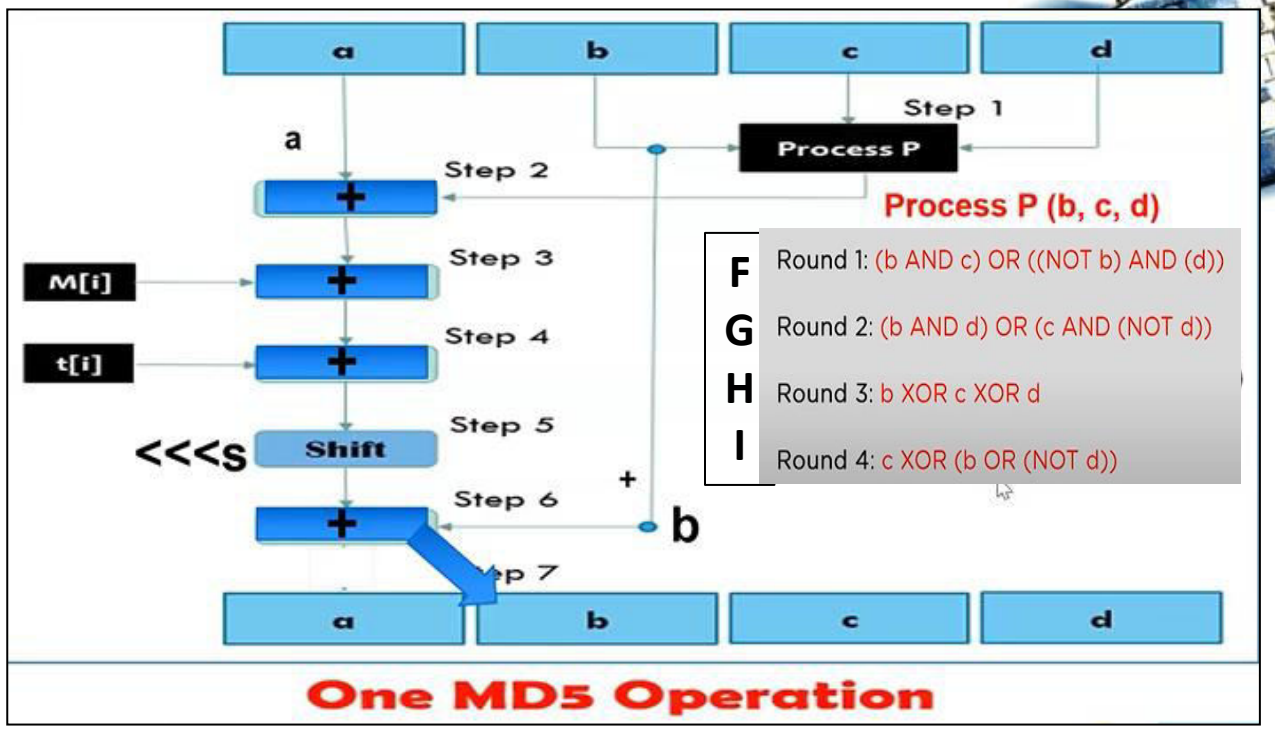
Step 5: Output:

- After processing all L 512-bit blocks, the 128 bit message digest is produced as a output.

Message Diagest 5(MD-5): Implementation



T_1 = d76aa478	T_{17} = f61e2562	T_{33} = fffa3942	T_{49} = f42922
T_2 = e8c7b756	T_{18} = c040b340	T_{34} = 8771f681	T_{50} = 432aff
T_3 = 242070db	T_{19} = 265e5a51	T_{35} = 6d9d6122	T_{51} = ab9423
T_4 = c1bdceee	T_{20} = e9b6c7aa	T_{36} = fde5380c	T_{52} = fc93a0
T_5 = f57c0faf	T_{21} = d62f105d	T_{37} = a4beea44	T_{53} = 655b59
T_6 = 4787c62a	T_{22} = 02441453	T_{38} = 4bdecfa9	T_{54} = 8f0ccc9
T_7 = a8304613	T_{23} = d8a1e681	T_{39} = f6bb4b60	T_{55} = ffeff47d
T_8 = fd469501	T_{24} = e7d3fbc8	T_{40} = bebfbc70	T_{56} = 85845dd
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



b = b + ((a + Process P (b, c, d)+ M[i] + T[k])<<<s

Total 64 operation perform in 4 rounds

1st round = 16 operation — F
2nd round = 16 operation — G
3rd = 16 — H
4th = 16 — I

SHA-1 and MD5

Difference between SHA-1 and MD5

No	SHA – 1	MD 5
1	It uses a 160-bit message digest.	It uses a 128-bit message digest
2	It is stronger against Brute-force attack compare to MD5.	It is weak against Brute-force attack compare to SHA-1.
3	SHA – 1 is not vulnerable against cryptanalysis.	MD5 is vulnerable against cryptanalysis.
4	SHA – 1 is slower than MD5.	MD5 is faster than SHA-1.
5	It uses big-endian method to represent the message.	It uses little-endian method to represent the message.
6	SHA has 80 steps	MD5 has 64 steps.

SHA-1 and MD5

Comparison based on parameter of different SHA version

- SHA – 1 was cracked in the year 2005. New hash function SHA-512 is introduced to overcom problem SHA-1.

❖**Comparison of SHA Parameters:**

Sr. No.	Parameters	SHA-1	SHA-256	SHA-384	SHA-512
1	Message digest size	160	256	384	512
2	Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
3	Block size	512	512	1024	1024
4	Word size	32	32	64	64
5	Number of steps	80	64	80	80
6	Security level	80	128	192	256