Name: Jigar Siddhpura **SAPID:** 60004200155

DIV: C/C2 **Branch:** Computer Engineering

IS - Experiment 7 - MD5 Algorithm

60004210155
Jigar Siddhpura
C22

IS - Exp 7 - MD5

Aim: To study & implement MD5 hashing algorithm.

Theory: MD5 is quite fast & produces a 128 bit message digest. The input text is processed in 512 bit blocks (which are divided into 16, 32 bit subblocks). The output of algo is a set of 4, 32 bit blocks, which makeup 128 bit message digest.

Working of MD5:

1. Padding — Make length of original msg equal to value which is 64 bits less than exact multiple of 512 bits padding consist of single 1 bit followed by all 0 bits.
2. Append Length — After padding, calculate length of original msg in terms of 64 bits (2^{64}) which is appended at end.
3. Divide input into 512 bit blocks.
4. Initialise chaining variables — 4 var's - A, B, C & D are initialised.
5. Process blocks — A loop that runs for as many 512 blocks are: we have 4 rounds. In each rounds, we process all 16 sub blocks. Inputs for rounds are all 16 sub blocks, variables a, b, c, d; some constands designated as k.

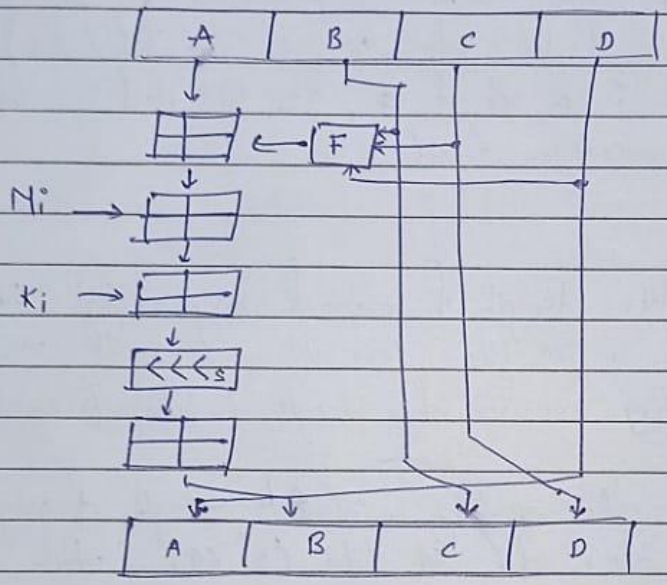
All 4 rounds vary in one major way. Step 1 of 4 rounds have diff. processing. In each round, we have 16 input subblocks named M[0], M[1], ... M[15], also T is an array of constants. It contains 64 elements with each element consisting of 32 bits, we denote it as K[0], K[1] ... K[63]

Diagramatically, where $F(x,y,z) = (x \text{ and } y) \text{ OR } (\text{not}(x) \text{ and } z)$

$G(x,y,z) = (x \text{ and } z) \text{ OR } (y \text{ and } \text{not}(z))$

$H(x,y,z) = x \text{ xor } y \text{ xor } z$

$I(x,y,z) = y \oplus (x \text{ or } \neg z)$



So, here F are given B, C, D at start of the round then output is added with A, output obtained is then added to M[i], then to K[i], which is circularly left shifted by S bits.
B is added up then at last all variable are right shifted by 1 position.

Conclusion: Thus, we studied & implemented MD5 in python for scratch for 1st round.

CODE

```
import random

def left_rotate(x, n):
    return ((x << n) | (x >> (32 - n))) & 0xFFFFFFFF

def md5(message):
    """
    A function to calculate the MD5 hash of the input message.
    Takes a message as input and returns the MD5 hash in hexadecimal format.
    """
    # Step 1 and 2: Padding and Append Length
    padding_length = 0
    if (len(message) + 64) % 512:
        message += "1"
        padding_length = 512 - ((len(message) + 64) % 512)
    message += "0" * padding_length
    print(f"Padding Length : {padding_length + 1}")

    # Step 3: Divide the input into 512-bit blocks
    message_words = [int(message[i: i + 32], 2) for i in range(0, len(message), 32)]
    original_message_length = len(message) - 64
    message_words.append(original_message_length)

    # Step 4: Initialise the chaining variables
    # Hexadecimal Constants are initialized
    K = [0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476]
    print(f"Chaining variables are: {K}")

    def F(x, y, z):
        return (x & y) | (~x & z)

    # Step 5: Process Block
    a, b, c, d = K
    for i in range(0, len(message_words), 16):
        f = F(b, c, d)
        g = (i >> 2) & 0x03
        for j in range(16):
            if i + j < len(message_words):
                temp = (a + f + message_words[i + j] + g) & 0xFFFFFFFF
                a = d
                d = c
                c = b
                b = (b + left_rotate(temp, 7)) & 0xFFFFFFFF
```

```
    digest = format(a, "08x")
    digest += format(b, "08x")
    digest += format(c, "08x")
    digest += format(d, "08x")
    return digest

random_message = "".join([random.choice(["0", "1"]) for _ in range(1000)])
print("Random 1000-bit message:", random_message)
first_round_res = md5(random_message)
print("After First Round :", first_round_res)
```

OUTPUT

```
PS D:\SEM-6\IS\EXPERIMENTS> python -u "d:\SEM-6\IS\EXPERIMENTS\md5.py"
Random 1000-bit message: 0000001010111010100100100000101000011001100011110000001111110110110010110100010100000000111010011110100011000001110111110101000000001
1101010010000000100111110001011001100111110111001110111000100100001111010101011001001100011101101010011011110111111110001010111001110010101010101101000011010
1100001100111100010111011000011010010101110111011011100110010101000101111010111110100101111111110110101110111101001110010001001010110110101100011001101100
1111000011100100101000100011010000011100011010101011100011000100001101001011010000011110011001101011010111100001110110100110100100100100101000001011011110
1010110000111001000001100110000111011100010000110000000010010100101101010001000001011110010010001001011010111000101101000110011010010011110001100011101001
1011111110100010000101010010010001100101101100011000001100010000011010111101100011110110100011000101010101011011001111001010010011110000000111010000101010
01010101000000101100011011010101010100000010001000110001011001110011010101100111110111010100111
Padding Length : 472
Chaining variables are: [1732584193, 4023233417, 2562383102, 271733878]
After First Round : ecc303e8acbdde67f2c92e224b7cd059
PS D:\SEM-6\IS\EXPERIMENTS>
```