NAME: PRERNA SUNIL JADHAV
SAP ID: 60004220127
BATCH: C2-2
BRANCH: COMPUTER ENGINEERING
COURSE: INFORMATION SECURITY LABORATORY
COURSE CODE: DJ19CEL603.

EXPERIMENT 07

AIM: Study and Implement MD5 Hashing Algorithm

THEORY: The MD5 (message-digest algorithm) hashing algorithm is a one-way cryptographic function that accepts a message of any length as i/p and returns as o/p a fixed-length digest value to be used for authenticating the original message.
MD5 was developed as an improvement of MD4 with the advanced security purposes.
The output of MD5 (Digest size) is always 128 bits.
MD5 was developed in 1991 by Ronald Rivest.
Use of MD5 Algorithm:
- It is used for file authentication.
- In a web application, it is used for security purposes. Eg: Secure password for user
- Using this algorithm, we can store our password in 128 bits format.

Algorithm:

1) Append Padding Bits: Add padding such a way that the total length of the message is 64 bit less than the the exact multiple of 512.
   ∴length (original msg + padding bits)
   $= 512 * i - 64$ ...where $i = 1, 2, 3, \ldots$
2) Append length bits: add the length bit in the output of the first step in such a way that the total number of bits is the perfect multiple of 512.
3) Initialize MD buffer: we use 4 buffers i.e., J, K, L and M. The size of each buffer is 32 bits
4) Process each 512-bit block: A total of 64 operations are performed in 4 rounds. Each round has 16 operations. We apply a different function on each round.
   We perform OR, AND, XOR and NOT for calculating functions. After applying the function we perform an operation on each block.
   for performing we need:
   - add modulo 2^{32}
   - M[i] - 32 bit message
   - K[i] - 32 bit constant
   - <<<n - left shift by n bits.

a) In the first step, o/p of K, L, and M are taken & then function f is applied to them.
We will add modulo 2^{32} bits for o/p with J
b) In second step, we add M[j] bit message with the output of the first step.
c) Then add 32 bits constant i.e. K[i] to the output of the second step.
d) At last we do left shift operation by n (can be any value of n) and addition modulo by 2^{32}.
After all steps, the result of J will be fed to K.
Now same steps will be used for all functions G, H, I.
After performing all 64 operations we will get our message digest.
Output: After all rounds, the buffer J, K, L and M contains the MD5 output starting with the lower bit J and ending with higher bits M.

CONCLUSION: MD5 is fastest and simple to understand. It generates a strong password in 16 bytes format. But MD5 would generate the same hash function for diffuent i/ps. MD5 is neither symmetric nor asymmetric. algorithm.
Hence we studied and implement MD5.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Information Security Laboratory
Course Code:	DJ19CEL603
Experiment No.:	07

AIM: Study and Implement MD5 Hashing Algorithm.

CODE:

```
import math

# This list maintains the amount by which to rotate the buffers during
processing stage
rotate_by = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
             5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20,
             4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
             6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]

# This list maintains the additive constant to be added in each processing
step.
constants = [int(abs(math.sin(i+1)) * 4294967296) & 0xFFFFFFFF for i in
range(64)]

# STEP 1: append padding bits s.t. the length is congruent to 448 modulo 512
# which is equivalent to saying 56 modulo 64.
# padding before adding the length of the original message is conventionally
done as:
# pad a one followed by zeros to become congruent to 448 modulo 512(or 56
modulo 64).
def pad(msg):
    msg_len_in_bits = (8*len(msg)) & 0xffffffffffffffff
    msg.append(0x80)

    while len(msg)%64 != 56:
        msg.append(0)

# STEP 2: append a 64-bit version of the length of the length of the original
message
# in the unlikely event that the length of the message is greater than 2^64,
# only the lower order 64 bits of the length are used.

# sys.byteorder -> 'little'
```

```
    msg += msg_len_in_bits.to_bytes(8, byteorder='little') # little endian
convention
    # to_bytes(8...) will return the lower order 64 bits(8 bytes) of the
length.

    return msg


# STEP 3: initialise message digest buffer.
# MD buffer is 4 words A, B, C and D each of 32-bits.

init_MDBuffer = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]

# UTILITY/HELPER FUNCTION:
def leftRotate(x, amount):
    x &= 0xFFFFFFFF
    return (x << amount | x >> (32-amount)) & 0xFFFFFFFF


# STEP 4: process the message in 16-word blocks
# Message block stored in buffers is processed in the follg general manner:
# A = B + rotate left by some amount<-(A + func(B, C, D) + additive constant +
1 of the 16 32-bit(4 byte) blocks converted to int form)

def processMessage(msg):
    init_temp = init_MDBuffer[:] # create copy of the buffer init constants to
preserve them for when message has multiple 512-bit blocks

    # message length is a multiple of 512bits, but the processing is to be
done separately for every 512-bit block.
    for offset in range(0, len(msg), 64):
        A, B, C, D = init_temp # have to initialise MD Buffer for every block
        block = msg[offset : offset+64] # create block to be processed
        # msg is processed as chunks of 16-words, hence, 16 such 32-bit chunks
        for i in range(64): # 1 pass through the loop processes some 32 bits
out of the 512-bit block.
            if i < 16:
                # Round 1
                func = lambda b, c, d: (b & c) | (~b & d)
                # if b is true then ans is c, else d.
                index_func = lambda i: i

            elif i >= 16 and i < 32:
                # Round 2
```

```
                func = lambda b, c, d: (d & b) | (~d & c)
                # if d is true then ans is b, else c.
                index_func = lambda i: (5*i + 1)%16

            elif i >= 32 and i < 48:
                # Round 3
                func = lambda b, c, d: b ^ c ^ d
                # Parity of b, c, d
                index_func = lambda i: (3*i + 5)%16

            elif i >= 48 and i < 64:
                # Round 4
                func = lambda b, c, d: c ^ (b | ~d)
                index_func = lambda i: (7*i)%16

            F = func(B, C, D) # operate on MD Buffers B, C, D
            G = index_func(i) # select one of the 32-bit words from the 512-
bit block of the original message to operate on.

            to_rotate = A + F + constants[i] + int.from_bytes(block[4*G : 4*G
+ 4], byteorder='little')
            newB = (B + leftRotate(to_rotate, rotate_by[i])) & 0xFFFFFFFF

            A, B, C, D = D, newB, B, C
            # rotate the contents of the 4 MD buffers by one every pass
through the loop

        # Add the final output of the above stage to initial buffer states
        for i, val in enumerate([A, B, C, D]):
            init_temp[i] += val
            init_temp[i] &= 0xFFFFFFFF
        # The init_temp list now holds the MD(in the form of the 4 buffers A,
B, C, D) of the 512-bit block of the message fed.


    # The same process is to be performed for every 512-bit block to get the
final MD(message digest).


    # Construct the final message from the final states of the MD Buffers
    return sum(buffer_content<<(32*i) for i, buffer_content in
enumerate(init_temp))
```

```
def MD_to_hex(digest):
    # takes MD from the processing stage, change its endian-ness and return it
as 128-bit hex hash
    raw = digest.to_bytes(16, byteorder='little')
    return '{:032x}'.format(int.from_bytes(raw, byteorder='big'))


def md5(msg):
    msg = bytearray(msg, 'ascii') # create a copy of the original message in
form of a sequence of integers [0, 256)
    msg = pad(msg)
    processed_msg = processMessage(msg)
    # processed_msg contains the integer value of the hash
    message_hash = MD_to_hex(processed_msg)
    print("Message Hash: ", message_hash)


if __name__ == '__main__':
    message = input("Enter your message: ")
    md5(message)
```

OUTPUT:

```
PS C:\Users\Jadhav\Documents\BTech\Docs\6th Sem\IS\Code> & C:/msys64/mingw64/bin/python.exe "c:/Users/Jadhav/Doc
uments/BTech/Docs/6th Sem/IS/Code/Exp7/MD5.py"
Enter your message: Hi, This is Prerna Jadhav
Message Hash:  a8c5e83e126d747b8e1b69841f588eab
PS C:\Users\Jadhav\Documents\BTech\Docs\6th Sem\IS\Code> & C:/msys64/mingw64/bin/python.exe "c:/Users/Jadhav/Doc
uments/BTech/Docs/6th Sem/IS/Code/Exp7/MD5.py"
Enter your message: Dwarkadas J. Sanghvi College of Engineering
Message Hash:  e0ef75d2c6f9c223bfd5c2ac0c5e71ae
PS C:\Users\Jadhav\Documents\BTech\Docs\6th Sem\IS\Code>
```