MD5 Algorithm
Cryptography – Lab 6

Name: Ojas Patil
Reg No: 21BAI1106

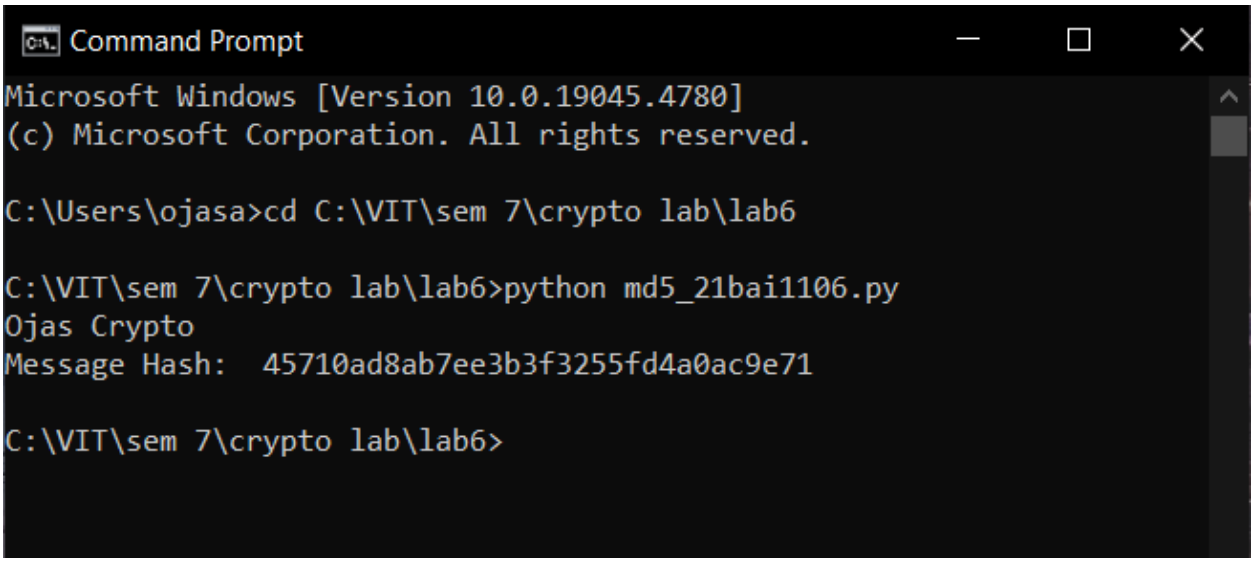
Task

To Develop a Python-based MD5 Algorithm implementation.

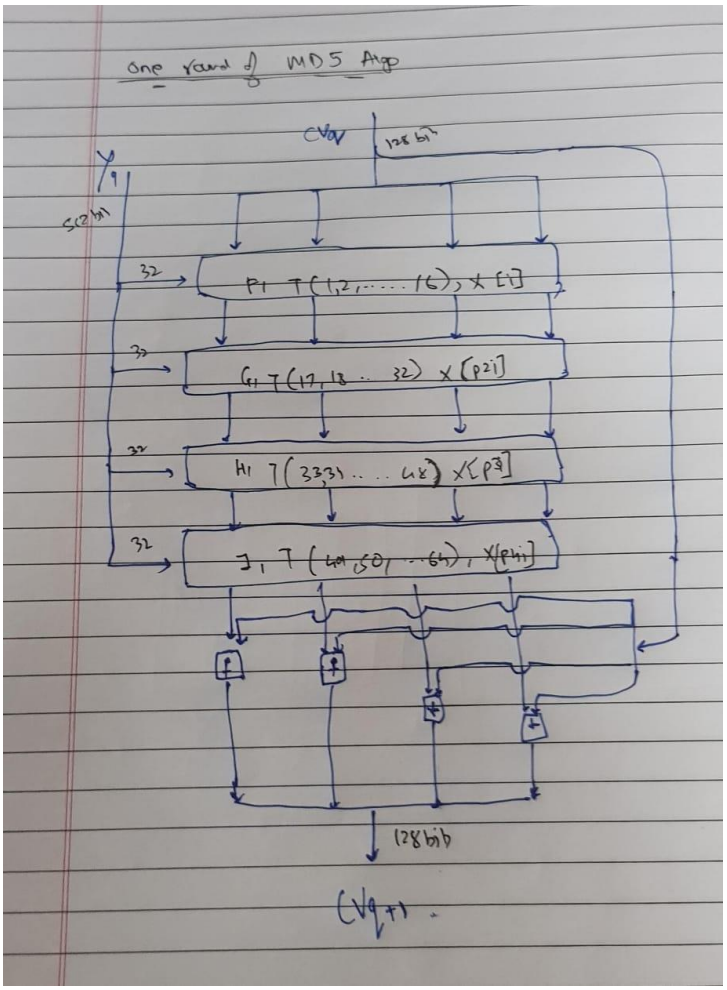
MD5 Algorithm - Definition

MD5 hashing algorithm generates a 128-bit hash (digest) from an input string. It processes the input in 512-bit blocks, applying non-linear functions and bitwise operations to produce a unique fixed-length hash, typically used for verifying data integrity.

MD5 Algorithm - Output Snapshot



MD5 - Diagram



Source Code

```
import math

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]

constants = [int(abs(math.sin(i+1)) * 4294967296) & 0xFFFFFFFF for i in
range(64)]

def pad(msg):
    msg_len_in_bits = (8*len(msg)) & 0xffffffffffffffff
    msg.append(0x80)
```

```
    while len(msg)%64 != 56:
        msg.append(0)
    msg += msg_len_in_bits.to_bytes(8, byteorder='little') # little endian
convention
    return msg


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

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

def processMessage(msg):
    init_temp = init_MDBuffer[:]

    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):
            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)

            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

        for i, val in enumerate([A, B, C, D]):
            init_temp[i] += val
            init_temp[i] &= 0xFFFFFFFF


    return sum(buffer_content<<(32*i) for i, buffer_content in
enumerate(init_temp))


def MD_to_hex(digest):
    raw = digest.to_bytes(16, byteorder='little')
    return '{:032x}'.format(int.from_bytes(raw, byteorder='big'))


def md5(msg):
    msg = bytearray(msg, 'ascii')
    msg = pad(msg)
    processed_msg = processMessage(msg)
    message_hash = MD_to_hex(processed_msg)
    print("Message Hash: ", message_hash)


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

Conclusion

The implementation of the MD5 algorithm in Python effectively demonstrates its key components, offering a solid understanding of how MD5 generates fixed-length hashes and its role in ensuring data integrity and authentication.