**Experiment 5**

**Aim:** To implement MD5 hashing technique

## Languages used: Java/Python

## Theory:

## Explain MD5 in brief

## Ans: The MD5 message-digest hashing algorithm processes data in 512-bit strings, broken down into 16 words composed of 32 bits each. The output from MD5 is a 128-bit message-digest value. Computation of the MD5 digest value is performed in separate stages that process each 512-bit block of data along with the value computed in the preceding stage. The first stage begins with the message-digest values initialized using consecutive hexadecimal numerical values. Each stage includes four message-digest passes, which manipulate values in the current data block and values processed from the previous block. The final value computed from the last block becomes the MD5 digest for that block.

## Block Diagram of MD5

## Ans:

## 

## Application ofMD5

## Ans: Although it's designed as a cryptographic function, MD5 suffers from extensive vulnerabilities, which is why you want to stay away from it when it comes to protecting your CMS, web framework, and other systems that use passwords for granting access. One of the reasons this is true is that it should be computationally infeasible to find two distinct messages that hash to the same value. But MD5 fails this requirement—such collisions can potentially be found in seconds. Despite the breaches, MD5 can still be used for standard file verifications and as a checksum to verify data integrity, but only against unintentional corruption. It also remains suitable for other non-cryptographic purposes, such as determining the partition for a particular key in a partitioned database.

**Implementation:**

**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')

return msg

init\_MDBuffer = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]

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

block = msg[offset : offset+64]

for i in range(64):

if i < 16:

func = lambda b, c, d: (b & c) | (~b & d)

index\_func = lambda i: i

elif i >= 16 and i < 32:

func = lambda b, c, d: (d & b) | (~d & c)

index\_func = lambda i: (5\*i + 1)%16

elif i >= 32 and i < 48:

func = lambda b, c, d: b ^ c ^ d

index\_func = lambda i: (3\*i + 5)%16

elif i >= 48 and i < 64:

func = lambda b, c, d: c ^ (b | ~d)

index\_func = lambda i: (7\*i)%16

F = func(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\_\_':

print("MD5 hashing technique")

message = input("Enter your message: ")

md5(message)

**Output:**



**Conclusion:** The MD5 Algorithm was studied and implemented successfully.