**ASSIGNMENT NO : 7**

NAME: Viraj Patil

PRN : 21510097

Q – IMPLEMENT RSA ALGORITHM

import math

# Function to calculate modular inverse

def mod\_inverse(e, phi):

# Using Extended Euclidean Algorithm

g, x, y = extended\_gcd(e, phi)

if g != 1:

raise Exception('Modular inverse does not exist')

else:

return x % phi

# Extended Euclidean Algorithm

def extended\_gcd(a, b):

if a == 0:

return (b, 0, 1)

else:

g, x, y = extended\_gcd(b % a, a)

return (g, y - (b // a) \* x, x)

# RSA algorithm demonstration

def main():

# User input for prime numbers p and q

p = int(input("Enter a prime number p: "))

q = int(input("Enter a prime number q: "))

n = p \* q

print(f"n (p \* q) = {n}")

# Calculate the totient (phi)

phi = (p - 1) \* (q - 1)

print(f"Totient (phi) = {phi}")

# Choose e such that it is co-prime to phi and less than phi

e = 2

while e < phi:

if math.gcd(e, phi) == 1:

break

else:

e += 1

print(f"Public key exponent (e) = {e}")

# Private key (d stands for decrypt)

d = mod\_inverse(e, phi)

print(f"Private key exponent (d) = {d}")

# User input for the message to be encrypted

message = input("Enter the message data to be encrypted: ")

# Encrypt the message

encrypted = [pow(ord(char), e, n) for char in message]

print(f"Encrypted data asci = {encrypted}")

# Decrypt the message

decrypted = [(pow(char, d, n)) for char in encrypted]

print(f"Decrypted data Ascii= {decrypted}")

decrypted = [chr(char) for char in decrypted]

print(f"Decrypted data = {''.join(decrypted)}")

# Call the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

OUTPUT :

