**Practical No : 09**

**Problem Statement:**

Implement RSA

**Program:**

import random

def extended\_gcd(a, b):

if a == 0:

return b, 0, 1

gcd, x1, y1 = extended\_gcd(b % a, a)

return gcd, y1 - (b // a) \* x1, x1

def char\_to\_number(char):

return str(ord(char) - ord('a')).zfill(2) if 'a' <= char <= 'z' else \

str(ord(char) - ord('A') + 26).zfill(2) if 'A' <= char <= 'Z' else \

'66' if char == '?' else '62'

def number\_to\_char(num\_str):

num = int(num\_str)

if 0 <= num <= 25:

return chr(num + ord('a'))

elif 26 <= num <= 51:

return chr(num - 26 + ord('A'))

return '?' if num == 66 else ' '

# Binary Exponentiation

def mod\_exp(a, e, n):

f = 1

while e > 0:

if e % 2 == 1:

f = (f \* a) % n

a = (a \* a) % n

e //= 2

return f

def crt\_decrypt(C, d, n, p, q):

dp, dq = d % (p - 1), d % (q - 1)

Vp = mod\_exp(C, dp, p)

Vq = mod\_exp(C, dq, q)

Xp = q \* (extended\_gcd(q, p)[1] % p)

Xq = p \* (extended\_gcd(p, q)[1] % q)

return (Vp \* Xp + Vq \* Xq) % n

# Function to select e

def select\_e(phi\_n):

for e in [3, 17, 65537]:

if e < phi\_n and extended\_gcd(e, phi\_n)[0] == 1:

return e

while True:

e = random.randrange(2, phi\_n)

if extended\_gcd(e, phi\_n)[0] == 1:

return e

# Calculate d

def calculate\_d(e, phi\_n):

gcd, x, \_ = extended\_gcd(e, phi\_n)

return None if gcd != 1 else x % phi\_n

# Key Generation

def rsa\_key():

p, q = 73, 151

n = p \* q

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

e = select\_e(phi\_n)

d = calculate\_d(e, phi\_n)

return [d, n], [e, n], p, q

def rsa\_encryption(plain\_text, public\_key):

cipher\_text = []

for i in range(0, len(plain\_text), 2):

p1 = char\_to\_number(plain\_text[i])

p2 = char\_to\_number(plain\_text[i + 1]) if (i + 1 < len(plain\_text)) else '66'

P = int(p1 + p2)

C = mod\_exp(P, public\_key[0], public\_key[1])

cipher\_text.append(str(C))

return cipher\_text

def rsa\_decryption(cipher\_text, private\_key, p, q):

plain\_text = ""

for C in cipher\_text:

P = crt\_decrypt(int(C), private\_key[0], private\_key[1], p, q)

plain\_text += number\_to\_char(str(P).zfill(4)[:2]) + number\_to\_char(str(P).zfill(4)[2:])

return plain\_text

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

private\_key, public\_key, p, q = rsa\_key()

# Plaintext Block consists of 4 decimal digits or 2 alphanumeric chars

plain\_text = "How are you?"

cipher\_text = rsa\_encryption(plain\_text, public\_key)

decrypted\_text = rsa\_decryption(cipher\_text, private\_key, p, q)

# Print results

print(f"p = {p}\nq = {q}")

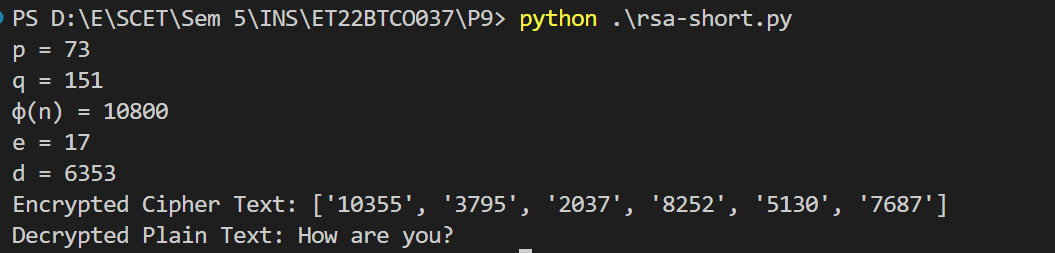
print(f"φ(n) = {(p - 1) \* (q - 1)}")

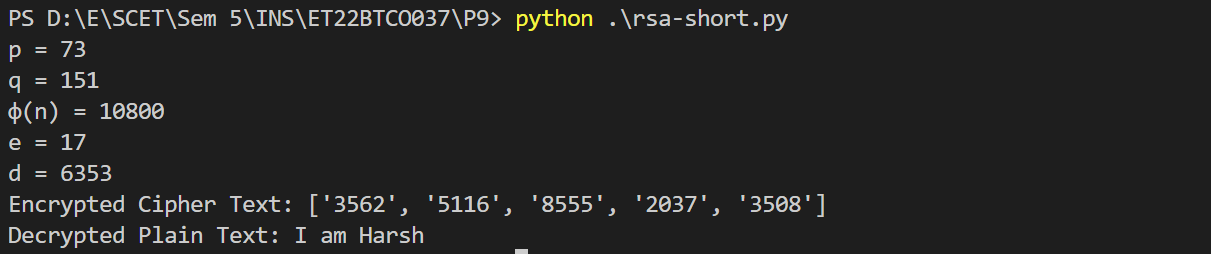
print(f"e = {public\_key[0]}\nd = {private\_key[0]}")

print(f"Encrypted Cipher Text: {cipher\_text}")

print(f"Decrypted Plain Text: {decrypted\_text}")

**Output:**

****

****

**Conclusion:**

I implemented the RSA algorithm in Python. It secures data by using a public-private key pair based on large prime numbers, making decryption without the private key difficult. However, if the private key is exposed or if the keys are too small, it can be compromised. This shows the need to combine RSA with other cryptographic techniques for better security.