ID :VU4F2223028 Experiment No.:07 Security Lab

**Aim:** Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA.

**Programs:**

**RSA Encryption/Decryption Example** -

def gcd(a, b):

"""Calculate the Greatest Common Divisor of a and b."""

while b != 0:

a, b = b, a % b

return a

def mod\_inverse(e, phi):

"""Find the modular inverse of e under modulo phi using the Extended Euclidean Algorithm."""

d\_old, d\_new = 0, 1

r\_old, r\_new = phi, e

while r\_new != 0:

quotient = r\_old // r\_new

r\_old, r\_new = r\_new, r\_old - quotient \* r\_new

d\_old, d\_new = d\_new, d\_old - quotient \* d\_new

if r\_old == 1:

return d\_old % phi

def rsa\_encrypt(plain\_text, e, N):

"""Encrypt plaintext using the public key (e, N)."""

return pow(plain\_text, e, N)

def rsa\_decrypt(cipher\_text, d, N):

"""Decrypt ciphertext using the private key (d, N)."""

return pow(cipher\_text, d, N)

# 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 # N = P \* Q

phi = (P - 1) \* (Q - 1) # phi = (P - 1) \* (Q - 1)

# Public key exponent e (Choose e such that 1 < e < phi and gcd(e, phi) = 1)

e = int(input("Enter a value for public key exponent e (should be coprime with φ): "))

# Ensure gcd(e, phi) = 1 (e and phi are coprime)

if gcd(e, phi) != 1:

raise ValueError("e and phi are not coprime!")

# Private key exponent d (modular inverse of e mod phi)

d = mod\_inverse(e, phi)

# Display calculated values of N, phi, and private key d

print(f"Calculated N: {N}")

print(f"Calculated φ(N): {phi}")

print(f"Calculated private key d: {d}")

# User input for plaintext

plain\_text = int(input("Enter the plaintext (numeric value) to encrypt: "))

# Encryption

cipher\_text = rsa\_encrypt(plain\_text, e, N)

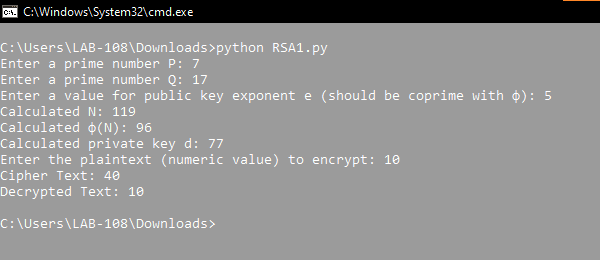
print(f"Cipher Text: {cipher\_text}")

# Decryption

decrypted\_text = rsa\_decrypt(cipher\_text, d, N)

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

**Output:**



**Digital Signature Example** –

import rsa

# Key Generation

(public\_key, private\_key) = rsa.newkeys(512)

# Sample message to be signed

message = "Verify this digital signature."

# Signing the message with the private key

signature = rsa.sign(message.encode(), private\_key, 'SHA-256')

print(f"Signature: {signature}")

# Verifying the message using the public key

try:

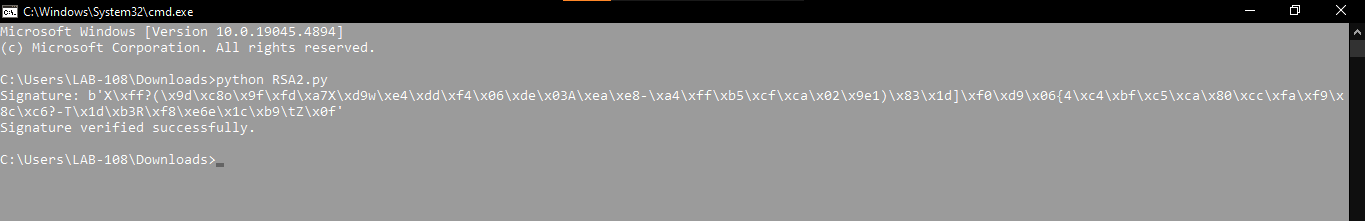
rsa.verify(message.encode(), signature, public\_key)

print("Signature verified successfully.")

except rsa.VerificationError:

print("Signature verification failed!")

**Output :**

****

**Conclusion:**

We have suscessfully Implemented and analyzed of RSA cryptosystem and Digital signature scheme using RSA.