**NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY**



**KARAIKAL – 609 609**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Roll Number :** CS22B1047 **Name :** S Karthikeyene

**Subject Code:** CS1702  **Subject Name:** Network Security

**Assignment**

**Implementation of RSA algorithm:**

import random

import sympy

def generate\_prime(bits):

    return sympy.randprime(2\*\*(bits-1), 2\*\*bits)

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

def mod\_inverse(e, phi):

    a, m = e, phi

    m0, y, x = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        y, x = x - q \* y, y

    return x + m0 if x < 0 else x

def generate\_keys(bits):

    p, q = generate\_prime(bits), generate\_prime(bits)

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randrange(2, phi)

    while gcd(e, phi) != 1:

        e = random.randrange(2, phi)

    d = mod\_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(plaintext, public\_key):

    e, n = public\_key

    return [pow(ord(char), e, n) for char in plaintext]

def decrypt(ciphertext, private\_key):

    d, n = private\_key

    return ''.join(chr(pow(char, d, n)) for char in ciphertext)

bits = int(input("Enter the key length (e.g., 512, 1024, 2048): "))

public\_key, private\_key = generate\_keys(bits)

print("\nPublic Key (e, n):", public\_key)

print("Private Key (d, n):", private\_key)

message = input("\nEnter the message to encrypt: ")

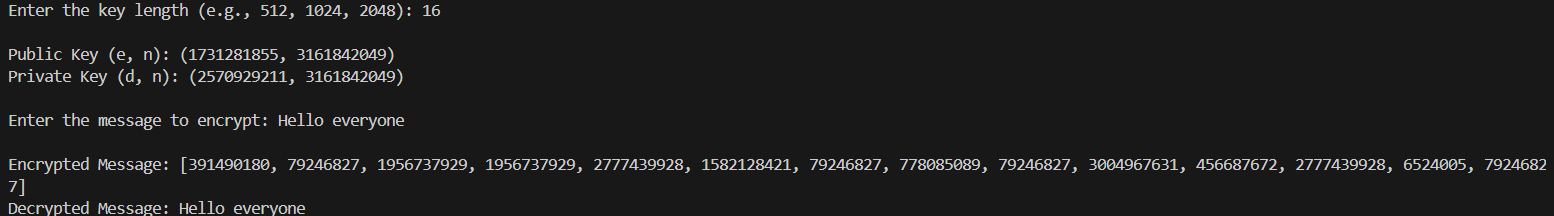
ciphertext = encrypt(message, public\_key)

decrypted\_message = decrypt(ciphertext, private\_key)

print("\nEncrypted Message:", ciphertext)

print("Decrypted Message:", decrypted\_message)

**Output:**

****

**Explanation:**

## 1. Generating Prime Numbers

The function generate\_prime(bits) generates a random prime number of the specified bit length using sympy.randprime(). Two such prime numbers p and q are generated to create the RSA key pair.

## 2. Computing RSA Components

The modulus n is computed as:  
  
 n = p × q  
  
Euler’s totient function:  
  
 ϕ(n) = (p - 1) × (q - 1)  
  
The encryption exponent e is chosen randomly such that:  
  
 1 < e < ϕ(n) and gcd(e, ϕ(n)) = 1  
  
The private exponent d is computed using the modular inverse:  
  
 d ≡ e⁻¹ mod ϕ(n)  
  
This is done using the Extended Euclidean Algorithm in mod\_inverse().

## 3. Key Generation

The function generate\_keys(bits):  
  
- Generates p and q  
- Computes n, phi(n), e, and d  
- Returns the public key (e, n) and private key (d, n)

## 4. Encryption Process

Each character in the plaintext is converted to ASCII using ord(char), then encrypted using:  
  
 c = m^e mod n  
  
where m is the ASCII value of the character.  
  
This is done using pow(ord(char), e, n) in encrypt().

## 5. Decryption Process

The ciphertext is decrypted by computing:  
  
 m = c^d mod n  
  
where c is the encrypted value.  
  
The result is converted back to characters using chr().