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:

```
Enter the key length (e.g., 512, 1024, 2048): 16

Public Key (e, n): (1731281855, 3161842049)
Private Key (d, n): (2570929211, 3161842049)

Enter the message to encrypt: Hello everyone

Encrypted Message: [391490180, 79246827, 1956737929, 1956737929, 2777439928, 1582128421, 79246827, 778085089, 79246827, 3004967631, 456687672, 2777439928, 6524005, 7924682
7]
Decrypted Message: Hello everyone
```

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:

$$n = p \times q$$

Euler's totient function:

$$\phi(n) = (p - 1) \times (q - 1)$$

The encryption exponent e is chosen randomly such that:

$$1 < e < \phi(n) \text{ and } \gcd(e, \phi(n)) = 1$$

The private exponent d is computed using the modular inverse:

$$d \equiv e^{-1} \bmod \phi(n)$$

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:

$$c = m^e \bmod n$$

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:

$$m = c^d \bmod n$$

where c is the encrypted value.

The result is converted back to characters using chr().