Experiment No. 5

Aim: To implement RSA algorithm.

Theory:

The **RSA algorithm** is an asymmetric cryptography algorithm; this means that it uses a *public* key and a *private* key (i.e. two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.

The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.

The following illustration highlights how asymmetric cryptography works:

How it works

The RSA algorithm ensures that the keys, in the above illustration, are as secure as possible. The following steps highlight how it works:

1. Generating the keys

1. Select two large prime numbers, p and q. The prime numbers need to be large so that they will be difficult for someone to figure out.
2. Calculate $n = p \times q$.
3. Calculate the ***totient*** function; $\varphi(n) = (p-1)(q-1)$.
4. Select an integer e, such that e is ***co-prime*** to $\varphi(n)$ and $1 < e < \varphi(n)$. The pair of numbers (n, e) makes up the public key.

Note: Two integers are co-prime if the only positive integer that divides them is 1.

Calculate $d => e.d \equiv 1 mod \varphi(n)$

d can be found using the ***Extended Euclidean algorithm***. The pair (n, d) makes up the private key.

2. Encryption

Given a plaintext P, represented as a number, the cipher text C is calculated as:

$C = P^e \equiv n.$

3. Decryption

Using the private key (n, d), the plaintext can be found using:

$$P = C^d \equiv n$$

<u>Example</u>:

Assuming receiver selects $p = 193\ and\ q = 103$

$n = p \times q = 193 \times 103 = 19879$

$\varphi(n) = (p - 1) \times (q - 1) = 192 \times 102 = 19584$

Considering, e = 1901 which is indeed coprime to $\varphi(n)$

$d = e^{-1} \equiv \varphi(n) = 3173$

If sender wants to send the Plain text $P = 997$ using public key e it will encrypted as follows:

$C = P^e \equiv n = 997^{1901} \equiv 19879 = 7915$

This Cipher text $C = 7915$ will be received at receiver side and it will use its own private key d to decrypt it:

$P = C^d \equiv n = 7915^{3173} \equiv 19879 = 997$

Implementation:

```
import math
import random

#implementation of RSA key generation algorithm and encryption, decryption using same key
def gcd(a,b):
    if not b:
        return a
    return gcd(b,a%b)

class RSA:
    def __init__(self):
        self.primes = []
        self.generate_totient()
        self.generate_keys()



    def generate_random_prime(self):
        self.primes.append(2)
        for i in range(3,200):
            if not i%2:
                continue
            Flag = True
            for j in range(2,i):
                if not i%j:
                    Flag = False
                    break
            if Flag:
                self.primes.append(i)

    def select_primes(self):
        self.generate_random_prime()
```

```
        p = random.sample(self.primes, 1)[0]
        q = p
        while(q == p):
            q = random.sample(self.primes, 1)[0]
        self.p = p
        self.q = q

    def generate_totient(self):
        self.select_primes()
        self.n = self.p * self.q
        self.totient_n = (self.p-1)*(self.q-1)

    def multiplicative_inverse(self):

        a,m,x,y = self.e, self.totient_n,1,0
        while (a > 1):

            q = a // m
            t = m
            m = a % m
            a = t
            t = y
            y = x - q * y
            x = t

        self.e_inverse = x

    def generate_keys(self):
        ls = [x for x in range(2,self.totient_n)]
        e = None
        ls = random.sample(ls,len(ls))
        for x in ls:
```

```
            k = gcd(self.totient_n,x)
            if k == 1:
                e = x
                break
        self.e = e
        self.multiplicative_inverse()
        self.generate_private_key()

    def generate_private_key(self):
        self.d = self.e_inverse % self.totient_n

    def fast_expo(self,txt,key):
        return (txt**key)%self.n

    def encryption(self,plain_txt):
        return self.fast_expo(plain_txt,self.e)

    def decryption(self, cipher_txt):
        return self.fast_expo(cipher_txt, self.d)

g = RSA()
print(f"p = {g.p}, q = {g.q}, n = {g.n}, totient_n = {g.totient_n} \
public_key = {g.e} public_key_inverse = {g.e_inverse} private_key = {g.d}\n")

encrypted = g.encryption(int(input(f"Enter value in [1,{g.n}] to encrypt it: ")))
print(f"Encrypted value = {encrypted}\n")
decrypted = g.decryption(int(input(f"Enter value for n = {g.n} to decrypt it: ")))
print(f"Decrypted value = {decrypted}")
```

Output:

```
p = 149, q = 151, n = 22499, totient_n = 22200 public_key = 911 public_key_inverse = -6409 private_key = 15791

Enter value in [1,22498] to encrypt it: 22495
Encrypted value = 476

Enter value for n = 22499 to decrypt it: 476
Decrypted value = 22495
```

```
p = 67, q = 137, n = 9179, totient_n = 8976 public_key = 7835 public_key_inverse = -4429 private_key = 4547

Enter value in [1,9178] to encrypt it: 9000
Encrypted value = 8437

Enter value for n = 9179 to decrypt it: 8437
Decrypted value = 9000
```

```
p = 179, q = 113, n = 20227, totient_n = 19936 public_key = 15445 public_key_inverse = -7107 private_key = 12829

Enter value in [1,20226] to encrypt it: 19999
Encrypted value = 13568

Enter value for n = 20227 to decrypt it: 13568
Decrypted value = 19999
```

```
p = 53, q = 139, n = 7367, totient_n = 7176 public_key = 509 public_key_inverse = -1579 private_key = 5597

Enter value in [1,7366] to encrypt it: 1
Encrypted value = 1

Enter value for n = 7367 to decrypt it: 1
Decrypted value = 1
```

```
p = 101, q = 173, n = 17473, totient_n = 17200 public_key = 1739 public_key_inverse = -5341 private_key = 11859

Enter value in [1,17472] to encrypt it: 6942
Encrypted value = 3891

Enter value for n = 17473 to decrypt it: 3891
Decrypted value = 6942
```