B. Tech – CE | Semester: VI | Subject: NIS
Lab 4

RSA Algorithm Description :

→ RSA (Rivest-Shamir-Adleman) is an asymmetric cryptographic algorithm used to encrypt and decrypt messages by modern computers. Asymmetric states that there are two different keys used in the encryption and decryption process, which also is called public-key cryptography. This is simply because one of the two keys can be given to anyone without exploiting the security of the algorithm

Key pairs and Complexity :

The RSA algorithm involves both private and public keys. The public key can be known and published to anyone, as it is used to encrypt the messages from plaintext to ciphertext. The messages that are encrypted with this specific public key can however only be decrypted with the corresponding private key. The key generation process of the RSA algorithm is what makes it so secure and reliable today, as it contains a high level of complexity compared to other cryptographic algorithms

Key generation :

→ Unlike symmetric algorithms, such as for example AES, public key algorithms require the computation of the pair (Kpublic,Kprivate). What makes RSA so special compared to other encryption algorithms is that these keys must be computed using mathematics, and are not random numbers that are generated.The key generation part of the RSA algorithm is quite central and important, and this is something that missing in most symmetric key algorithms, where the key generation part is not really complicated in terms of mathematical computations

Encryption and Decryption :

→ After computing all the necessary variables for the key generation, it is time to encrypt and decrypt a mes-sage using the algorithm. This is of course given the fact that Kpublic has been generated, and consists of N and e. The formula is very simple for both the encryption and decryption process, which states that:

Encryption: $c \equiv m^e \pmod n$, where m is the message in plaintext.

Decryption: $m \equiv c^d \pmod n$, where c is the ciphertext

Task 1 : Do Key generation
Task 2. Encryption
Task 3. Decryption, Use only Multiply and Square to do Modular Exponentiation.
Task 4.Miller Rabin for Primality Testing

All this things in one code :

Source Code: Python

```python
from collections import defaultdict
import math
from random import randrange, getrandbits
d = defaultdict(lambda: "Not Present")
for i in range(27):
    d[chr(97+i)] = i

def DoMultiplictiveInverse(n , a):
    r1, r2, t1, t2= n, a, 0, 1
    while r2 > 0:
        q = r1//r2
        r = r1-(q*r2)
        r1 = r2
        r2 = r

        t = t1-(q*t2)
        t1 = t2
        t2 = t
    if r1 == 1:
        if t1 < 0:
            return t1 + n
        else:
            return t1

    else:
        return -1

def get_key(val):
    for key, value in d.items():
         if val == value:
             return key
    return "key doesn't exist"
```

```
def multiply_and_square(bas, exp, N):
    t = 1;
    while(exp > 0):

        # for cases where exponent
        # is not an even value
        if (exp % 2 != 0):
            t = (t * bas) % N;

        bas = (bas * bas) % N;
        exp = int(exp / 2);
    return t % N;
def computeGCD(x, y):

   while(y):
       x, y = y, x % y

   return x

def keyGenration():
    p = generate_prime_number()
    q = generate_prime_number()


    print("P --> ", p , "Q -->", q)
    n = p*q
    phi_n = (p-1)*(q-1)
    e= 0
    for i in range(2 , phi_n):
        if computeGCD(phi_n , i) == 1:
            e = i
            break
    print("e--> ",e)
    e_inverse = DoMultiplictiveInverse(phi_n , e)
    d = e_inverse % phi_n
    public_key = [e , n]
    privte_key = [d , n]
    return public_key,privte_key
def is_prime(n, k=128):
    """ Test if a number is prime
        Args:
            n -- int -- the number to test
```

```python
            k -- int -- the number of tests to do
        return True if n is prime
    """
    # Test if n is not even.
    # But care, 2 is prime !
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 2 == 0:
        return False
    # find r and s
    s = 0
    r = n - 1
    while r & 1 == 0:
        s += 1
        r //= 2
    # do k tests
    for _ in range(k):
        a = randrange(2, n - 1)
        x = pow(a, r, n)
        if x != 1 and x != n - 1:
            j = 1
            while j < s and x != n - 1:
                x = pow(x, 2, n)
                if x == 1:
                    return False
                j += 1
            if x != n - 1:
                return False
    return True
def generate_prime_candidate(length):
    """ Generate an odd integer randomly
        Args:
            length -- int -- the length of the number to generate, in
bits
        return a integer
    """
    # generate random bits
    p = getrandbits(length)
    # apply a mask to set MSB and LSB to 1
    p |= (1 << length - 1) | 1
    return p
def generate_prime_number(length=50):
    """ Generate a prime
```

```
        Args:
            length -- int -- length of the prime to generate, in
bits
        return a prime
    """
    p = 4
    # keep generating while the primality test fail
    while not is_prime(p, 128):
        p = generate_prime_candidate(length)
    return p

def RSA_Encryption(m , e , n):
    return multiply_and_square(m , e, n)

def RSA_Decryption(c , d, n):
    return multiply_and_square(c, d, n)%n

if __name__ == "__main__":
    public_key , privte_key = keyGenration()

    m = 3
    print("Plain Number -->"  , m)
    encrpted_num = RSA_Encryption(m , public_key[0] , public_key[1])
    print("Encrypted Number --> ",encrpted_num)

    decrypted_num = RSA_Decryption(encrpted_num , privte_key[0] ,
privte_key[1])
    print("Decrypted Number --> ",decrypted_num)
```

Output :

```
P -->  141242197 Q --> 211582141
e-->  13
Plain Number --> 34
Encrypted Number -->  2356958783181349
Decrypted Number -->  34


...Program finished with exit code 0
Press ENTER to exit console.
```