

EXP NO: 2a

DATE:

### RSA ALGORITHM

AIM:

To write a python program to implement RSA Algorithm.

ALGORITHM:

1. Select two large prime numbers,  $p$  and  $q$ .
2. Multiply these numbers to find  $n = p \times q$ , where  $n$  is called the modulus for encryption and decryption.
3. Choose a number  $e$  less than  $n$ , such that  $n$  is relatively prime to  $(p - 1) \times (q - 1)$ . It means that  $e$  and  $(p - 1) \times (q - 1)$  have no common factor except 1. Choose " $e$ " such that  $1 < e < \phi(n)$ ,  $e$  is prime to  $\phi(n)$ ,  $\gcd(e, \phi(n)) = 1$
4. If  $n = p \times q$ , then the public key is  $\langle e, n \rangle$ . A plaintext message  $m$  is encrypted using public key  $\langle e, n \rangle$ . To find ciphertext from the plain text following formula is used to get ciphertext  $C$ .  
$$C = m^e \bmod n$$
5. Here,  $m$  must be less than  $n$ . A larger message ( $>n$ ) is treated as a concatenation of messages, each of which is encrypted separately.
6. To determine the private key, we use the following formula to calculate the  $d$  such that:  
$$De \bmod \{(p - 1) \times (q - 1)\} = 1$$

Or

$$De \bmod \phi(n) = 1$$
7. The private key is  $\langle d, n \rangle$ . A ciphertext message  $c$  is decrypted using private key  $\langle d, n \rangle$ . To calculate plain text  $m$  from the ciphertext  $c$  following formula is used to get plain text  $m$ .

## PROGRAM:

```
from math import gcd

# defining a function to perform RSA approach
def RSA(p: int, q: int, message: int):
    # calculating n
    n = p * q

    # calculating totient, t
    t = (p - 1) * (q - 1)

    # selecting public key, e
    for i in range(2, t):
        if gcd(i, t) == 1:
            e = i
            break

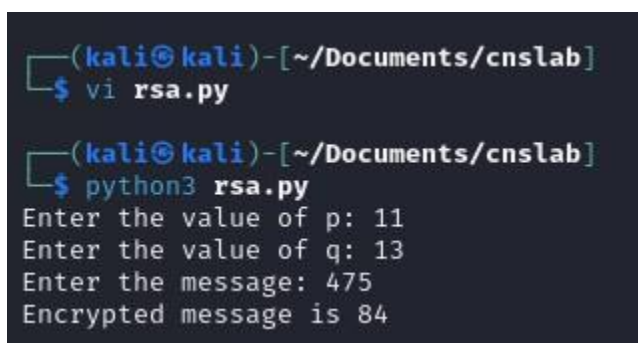
    # selecting private key, d
    j = 0
    while True:
        if (j * e) % t == 1:
            d = j
            break
        j += 1

    # performing encryption
    ct = (message ** e) % n
    print(f"Encrypted message is {ct}")
```

```
# performing decryption
mes = (ct ** d) % n
print(f"Decrypted message is {mes}")

p=int(input("Enter the value of p: "))
q=int(input("Enter the value of q: "))
msg=int(input("Enter the message: "))
RSA(p,q,msg)
```

OUTPUT:

A terminal window with a dark background and light blue text. The prompt is '(kali@kali)-[~/Documents/cnslab]'. The user enters '\$ vi rsa.py'. The prompt is again '(kali@kali)-[~/Documents/cnslab]'. The user enters '\$ python3 rsa.py'. The program then prompts for 'Enter the value of p: 11', 'Enter the value of q: 13', and 'Enter the message: 475'. Finally, it outputs 'Encrypted message is 84'.

```
(kali@kali)-[~/Documents/cnslab]
$ vi rsa.py

(kali@kali)-[~/Documents/cnslab]
$ python3 rsa.py
Enter the value of p: 11
Enter the value of q: 13
Enter the message: 475
Encrypted message is 84
```

RESULT:

Thus, a python program is implemented to demonstrate RSA Algorithm.

EXP NO: 2b

DATE:

## DIFFIE HELMAN KEY EXCHANGE

AIM:

To write a python program to Diffie Helman Key Exchange.

ALGORITHM:

1. Agree on Public Parameters: Both parties agree on two publicly known numbers: a large prime number  $(p)$  (the modulus) and a primitive root modulo  $(p)$ , denoted as  $(g)$ .
2. Generate Private Keys:
  - Each party independently selects a private key.
  - Party A selects a private key  $(a)$ , which is a randomly chosen integer within the range  $(1 < a < p-1)$ .
  - Party B selects a private key  $(b)$ , which is a randomly chosen integer within the range  $(1 < b < p-1)$ .
3. Compute Public Keys:
  - Party A computes its public key  $(A)$  using the formula  $(A = g^a \mod p)$ .
  - Party B computes its public key  $(B)$  using the formula  $(B = g^b \mod p)$ .
4. Exchange Public Keys:
  - Party A sends its public key  $(A)$  to Party B.
  - Party B sends its public key  $(B)$  to Party A.
5. Compute Shared Secret:
  - Party A uses Party B's public key  $(B)$  and its own private key  $(a)$  to compute the shared secret  $(s)$  using the formula  $(s = B^a \mod p)$ .
  - Party B uses Party A's public key  $(A)$  and its own private key  $(b)$  to compute the shared secret  $(s)$  using the formula  $(s = A^b \mod p)$ .
6. Use Shared Secret: The shared secret  $(s)$  can be used as a key for symmetric encryption, allowing Party A and Party B to securely communicate.
7. Ensure Security\*\*: The security of the Diffie-Hellman key exchange relies on the difficulty of solving the discrete logarithm problem. Therefore,  $(p)$  and  $(g)$  should be chosen such that they provide a high level of security against potential attacks.

PROGRAM:

```
def prime_checker(p):
    # Checks If the number entered is a Prime Number or not
    if p < 1:
        return -1
    elif p > 1:
        if p == 2:
            return 1
        for i in range(2, p):
            if p % i == 0:
                return -1
        return 1

def primitive_check(g, p, L):
    # Checks If The Entered Number Is A Primitive Root Or Not
    for i in range(1, p):
        L.append(pow(g, i) % p)
    for i in range(1, p):
        if L.count(i) > 1:
            L.clear()
            return -1
    return 1

l = []
while 1:
    P = int(input("Enter P : "))
    if prime_checker(P) == -1:
        print("Number Is Not Prime, Please Enter Again!")
        continue
    break

while 1:
    G = int(input(f"Enter The Primitive Root Of {P} : "))
    if primitive_check(G, P, l) == -1:
        print(f"Number Is Not A Primitive Root Of {P}, Please Try Again!")
        continue
    break

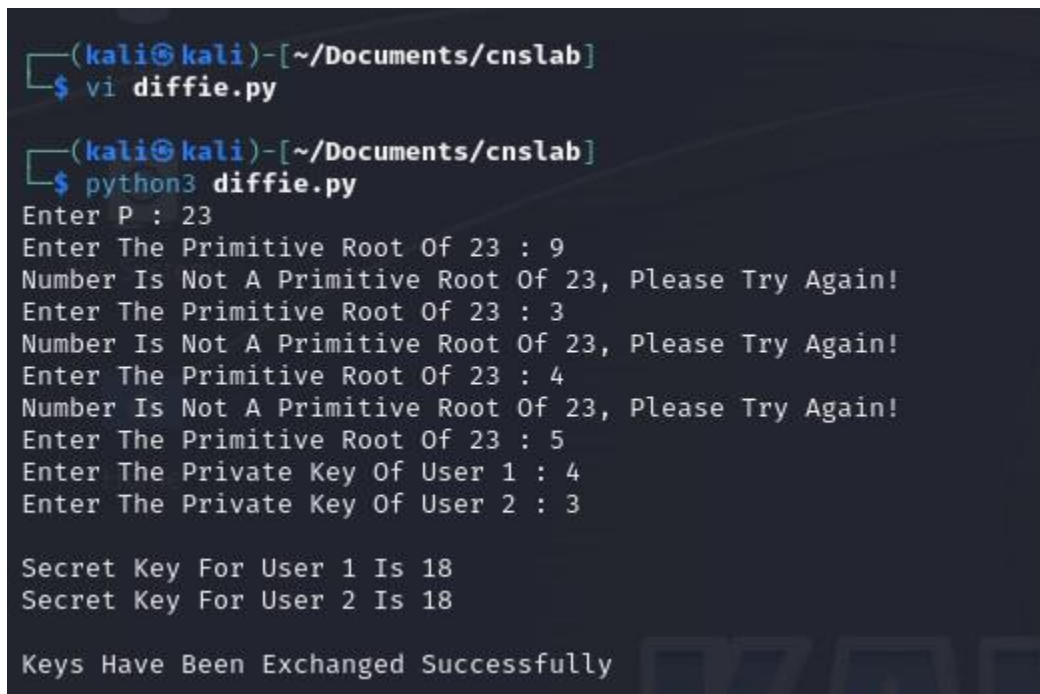
# Private Keys
x1, x2 = int(input("Enter The Private Key Of User 1 : ")), int(
    input("Enter The Private Key Of User 2 : "))
```

```

while 1:
    if x1 >= P or x2 >= P:
        print(f"Private Key Of Both The Users Should Be Less Than {P}!")
        continue
    break
# Calculate Public Keys
y1, y2 = pow(G, x1) % P, pow(G, x2) % P
# Generate Secret Keys
k1, k2 = pow(y2, x1) % P, pow(y1, x2) % P
print(f"\nSecret Key For User 1 Is {k1}\nSecret Key For User 2 Is {k2}\n")
if k1 == k2:
    print("Keys Have Been Exchanged Successfully")
else:
    print("Keys Have Not Been Exchanged Successfully")

```

OUTPUT:



```

(kali@kali)-[~/Documents/cnslab]
$ vi diffie.py

(kali@kali)-[~/Documents/cnslab]
$ python3 diffie.py
Enter P : 23
Enter The Primitive Root Of 23 : 9
Number Is Not A Primitive Root Of 23, Please Try Again!
Enter The Primitive Root Of 23 : 3
Number Is Not A Primitive Root Of 23, Please Try Again!
Enter The Primitive Root Of 23 : 4
Number Is Not A Primitive Root Of 23, Please Try Again!
Enter The Primitive Root Of 23 : 5
Enter The Private Key Of User 1 : 4
Enter The Private Key Of User 2 : 3

Secret Key For User 1 Is 18
Secret Key For User 2 Is 18

Keys Have Been Exchanged Successfully

```

RESULT:

Thus, a python program has been implemented to demonstrate Diffie Hellman Key Exchange Algorithm.