EXP NO: 2a
DATE:

RSA ALGORITHM

AIM:

To write a python program implementing the RSA algorithm.

ALGORITHM:

- Input: Read the plaintext message and the keyword.
- Generate Key Order: Determine the numerical order of the keyword letters (e.g., "CIPHER" → [3, 1, 4, 5, 2, 6]).
- Pad Message: Pad the plaintext with spaces to fit into a rectangle where the number of columns is the length of the keyword.
- Fill Grid: Write the plaintext into a grid, row by row, with columns equal to the length of the keyword.
- Read Columns: Read the columns in the order specified by the keyword.
- Construct Ciphertext: Concatenate the characters read from each column to form the ciphertext.
- Output: Print or return the final ciphertext.

PROGRAM:

```
from math import gcd

# defining a function to perform RSA approch
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:

```
┌──(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 implementing the Diffie Hellman algorithm.

ALGORITHM:

- Input: Choose a large prime number p and a base g (both public).
- Private Keys: Each party (Alice and Bob) selects a private key (a for Alice and b for Bob), which are secret integers.
- Compute Public Keys:
- Alice computes A = g^a mod p
- Bob computes B = g^b mod p
- Exchange Public Keys: Alice sends A to Bob, and Bob sends B to Alice.
- Compute Shared Secret:
- Alice computes the shared secret s = B^a mod p
- Bob computes the shared secret s = A^b mod p
- Verify Shared Secret: Both Alice and Bob now have the same shared secret s.
- Use Shared Secret: The shared secret s can be used as a key for symmetric encryption to securely communicate.

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.