7) To write a python program to implement the RSA encryption algorithm.

PROGRAM:-

```
import math
import random
def is_prime(num):
    """Check if a number is prime"""
    if num < 2:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    sqrt_num = int(math.sqrt(num)) + 1
    for i in range(3, sqrt_num, 2):
        if num % i == 0:
            return False
    return True
def gcd(a, b):
    """Calculate greatest common divisor"""
    while b != 0:
        a, b = b, a % b
    return a
def modinv(e, t):
    """Calculate modular inverse using extended Euclidean algorithm"""
    g, x, y = extended_gcd(e, t)
    if g != 1:
        return None  # No inverse exists
    else:
        return x % t
```

```
def extended_gcd(a, b):
    """Extended Euclidean algorithm"""
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)
def generate_keys(p, q):
    """Generate RSA public and private keys"""
    if not (is_prime(p) and is_prime(q)):
        raise ValueError("Both numbers must be prime")
    elif p == q:
        raise ValueError("p and q cannot be equal")
    n = p * q
    t = (p-1) * (q-1)
    # Choose e such that 1 < e < t and gcd(e, t) == 1
    e = random.randrange(1, t)
    while gcd(e, t) != 1:
        e = random.randrange(1, t)
    d = modinv(e, t)
    return ((e, n), (d, n))
def encrypt(pk, plaintext):
    """Encrypt message using public key (e, n)"""
    e, n = pk
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher
def decrypt(pk, ciphertext):
    """Decrypt message using private key (d, n)"""
    d, n = pk
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)
```

```
if __name__ == '__main__':
    print("RSA Encryption/Decryption")
    # Get prime numbers from user
    while True:
        try:
            p = int(input("Enter first prime number: "))
            q = int(input("Enter second prime number: "))
            public, private = generate_keys(p, q)
            break
        except ValueError as e:
            print(e)
    print(f"Public key (e, n): {public}")
    print(f"Private key (d, n): {private}"
    message = input("Enter message to encrypt: ")
    encrypted_msg = encrypt(public, message)
    print("Encrypted message (as numbers):", ' '.join(map(str, encrypted_msg)))
    decrypted_msg = decrypt(private, encrypted_msg)
    print("Decrypted message:", decrypted_msg)
```

OUTPUT:-

```
Public: (17, 3233)
Private: (2753, 3233)
Encrypted: [1394, 887, 887, 3016, 3016]
Decrypted: HELLO
```