```
import random
```

```
class RSA:
    def __init__(self):
        pass


    def isPrime(num):
        if num == 2:
            return True
        if num < 2 or num % 2 == 0:
            return False
        for n in range(3, int(num ** 0.5) + 2, 2):
            if num % n == 0:
                return False
        return True


    def gcd(a, b):
        while b != 0:
            a, b = b, a % b
        return a


    def modInv(e, phi):
        # Initialize extended variables
        x1, x2 = 0, 1
        d, y1 = 0, 1
        temp_phi = phi
        # Modular multiplicative inverse calculation
        while e > 0:
            # Calculate quotient and remainder
            temp1 = temp_phi // e
            temp2 = temp_phi - temp1 * e
            temp_phi = e
            e = temp2
            # Update values for the next iteration
            x = x2 - temp1 * x1
            y = d - temp1 * y1
            x2, x1 = x1, x
            d, y1 = y1, y
        # Ensure that 'phi' is positive
        if temp_phi == 1:
            return d + phi


    def generateKey(p, q):
        # Check if user-entered numbers are prime
        if not (RSA.isPrime(p) and RSA.isPrime(q)):
            print('non-Prime Numbers. Run again!')
        elif p == q:
            print('Equal Numbers. Run again!')
        # Calculate 'n' and Euler's totient 'phi'
        n = p * q
        phi = (p - 1) * (q - 1)
        # Choose a public exponent 'e' that is coprime with 'phi'
        e = random.randrange(1, phi)
        g = RSA.gcd(e, phi)
        while g != 1:
            e = random.randrange(1, phi)
            g = RSA.gcd(e, phi)
        # Calculate the private exponent 'd' using extended Euclidean algorithm
        d = RSA.modInv(e, phi)
        # Pack the public and private keys
        public = (e, n)
        private = (d, n)
        return public, private


    def encrypt(pk, message):
        # Unpack the key
        k, n = pk
        # Encrypt the message character by character
        encrypted = [pow(ord(char), k, n) for char in message]
        return encrypted


    def decrypt(pk, encrypted):
        # Unpack the key
        k, n = pk
        # Decrypt the ciphertext character by character
```

```python
        # Decrypt the ciphertext character by character
        decrypting = [str(pow(char, k, n)) for char in encrypted]
        # Convert bytes into a string
        decrypted = [chr(int(ch)) for ch in decrypting]
        return ''.join(decrypted)


    def display():
        message = input("Enter the message: ")
        p = int(input("Enter a prime number: "))
        q = int(input("Enter a different prime number: "))
        public, private = RSA.generateKey(p, q)
        print("Public Key: ", public)
        print("Private Key: ", private)
        encrypted = RSA.encrypt(public, message)
        print("Encrypted Message: ",''.join(map(lambda x: str(x), encrypted)))
        decrypted = RSA.decrypt(private, encrypted)
        print("Decrypted Message: ", decrypted)



if __name__ == '__main__':
    RSA.display()
```

```
Enter the message: usaid
Enter a prime number: 797
Enter a different prime number: 859
Public Key:  (529645, 684623)
Private Key:  (759277, 684623)
Encrypted Message:  3226541968027284335563555438
Decrypted Message:  usaid
```