In [7]:

```
import random


'''
Euclid's algorithm for determining the greatest common divisor
Use iteration to make it faster for larger integers
'''


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


'''
Euclid's extended algorithm for finding the multiplicative inverse of two numbers
'''


def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi

    while e > 0:
        temp1 = temp_phi//e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2

        x = x2 - temp1 * x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y
        if temp_phi == 1:
            return d + phi



def is_prime(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 generate_key_pair(p, q):
    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 = pq
    n = p * q

    # Phi is the totient of n
    phi = (p-1) * (q-1)

    # Choose an integer e such that e and phi(n) are coprime
    e = random.randrange(1, phi)

    # Use Euclid's Algorithm to verify that e and phi(n) are coprime
    g = gcd(e, phi)
    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)

    # Use Extended Euclid's Algorithm to generate the private key
    d = multiplicative_inverse(e, phi)

    # Return public and private key_pair
    # Public key is (e, n) and private key is (d, n)
    return ((e, n), (d, n))

def encrypt(pk, plaintext):
    # Unpack the key into it's components
    key, n = pk
    # Convert each letter in the plaintext to numbers based on the character using a^b m
    cipher = [pow(ord(char), key, n) for char in plaintext]
    # Return the array of bytes
    return cipher


def decrypt(pk, ciphertext):
    # Unpack the key into its components
    key, n = pk
    # Generate the plaintext based on the ciphertext and key using a^b mod m
    aux = [str(pow(char, key, n)) for char in ciphertext]
    # Return the array of bytes as a string
    plain = [chr(int(char2)) for char2 in aux]
    return ''.join(plain)

if __name__ == '__main__':
    '''
    Detect if the script is being run directly by the user
    '''
    print("=========================================================================
    print("================================= RSA Encryptor / Decrypter ==============
    print(" ")

    p = int(input(" - Enter a prime number (17, 19, 23, etc): "))
    q = int(input(" - Enter another prime number (Not one you entered above): "))

    print(" - Generating your public / private key-pairs now . . .")

    public, private = generate_key_pair(p, q)

    print(" - Your public key is ", public, " and your private key is ", private)

    message = input(" - Enter a message to encrypt with your public key: ")
    encrypted_msg = encrypt(public, message)
```

```
    print(" - Your encrypted message is: ", ''.join(map(lambda x: str(x), encrypted_msg)
    print(" - Decrypting message with private key ", private, " . . .")
    print(" - Your message is: ", decrypt(private, encrypted_msg))

    print(" ")
    print("============================================ END ===========================
    print("============================================================================
```

```
==========================================================================
=================================
================================== RSA Encryptor / Decrypter =============
=================================

 - Enter a prime number (17, 19, 23, etc): 17
 - Enter another prime number (Not one you entered above): 23
 - Generating your public / private key-pairs now . . .
 - Your public key is  (15, 391)  and your private key is  (399, 391)
 - Enter a message to encrypt with your public key: harshda
 - Your encrypted message is:  12818036727612825180
 - Decrypting message with private key  (399, 391)  . . .
 - Your message is:  harshda

============================================ END =========================
=================================
==========================================================================
=================================
```

In [ ]: