1. **input.py**

def power(base, expo, m):

    res = 1

    base = base % m

    while expo > 0:

        if expo & 1:

            res = (res \* base) % m

        base = (base \* base) % m

        expo //= 2

    return res

def modInverse(e, phi):

    for d in range(2, phi):

        if (e \* d) % phi == 1:

            return d

    return -1

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def is\_prime(n):

    if n < 2:

        return False

    for i in range(2, int(n \*\* 0.5) + 1):

        if n % i == 0:

            return False

    return True

def generateKeys(p, q):

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = 2

    while e < phi:

        if gcd(e, phi) == 1:

            break

        e += 1

    d = modInverse(e, phi)

    if d == -1:

        raise ValueError("No modular inverse found")

    return e, d, n

def encrypt(m, e, n):

    return power(m, e, n)

def decrypt(c, d, n):

    return power(c, d, n)

p = int(input("Enter a prime number (p): "))

if not is\_prime(p):

    print("p is not a prime number. Exiting.")

    exit()

q = int(input("Enter another prime number (q): "))

if not is\_prime(q):

    print("q is not a prime number. Exiting.")

    exit()

e, d, n = generateKeys(p, q)

print(f"Public Key (e, n): ({e}, {n})")

print(f"Private Key (d, n): ({d}, {n})")

M = int(input("Enter a message (as a number) to encrypt: "))

C = encrypt(M, e, n)

print(f"Encrypted Message: {C}")

decrypted = decrypt(C, d, n)

print(f"Decrypted Message: {decrypted}")

1. **Output**

