**Assignment 4 ( Implementation of RSA Algorithm)**

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

'''

Tests to see if a number is prime.

'''

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 mod 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(" ")

**Output:**

- Enter a prime number (17, 19, 23, etc):

- Enter another prime number (Not one you entered above):

- Generating your public / private key-pairs now . . .

- Your public key is \_\_\_\_ and your private key is \_\_\_\_\_\_\_

- Enter a message to encrypt with your public key: