**Hugh Coleman - 04/22/2024**

**FINAL EXERCISE: 04-16-24**

**Prof. Myers**

**Analysis of Algorithms SP24**

**Trinity University**

**#main.py**

from ascii import encodeAscii, decodeAscii

from rsa import totient, public\_exponent, modinv, rsa\_encrypt, rsa\_decrypt

print(encodeAscii("k"))

p = 923978444369

q = 496509772799

n = p \* q

phi = totient(p, q)

e = public\_exponent(phi)

d = modinv(e, phi)

print("enter message to be sent: ")

P = input()

print("p : " + str(p))

print("q : " + str(p))

print("n : " + str(n))

print("phi : " + str(phi))

print("e : " + str(e))

print("d : " + str(d))

print("")

print("Public key (E, n):", (e, n))

print("Private key (d, n):", (d, n))

encodedText = int(encodeAscii(P))

print("The message to be sent: " + P)

print("text after ascii encoding as integer : " + str(encodedText))

ciphertext = rsa\_encrypt(encodedText, e, n)

print("Ciphertext: " + str(ciphertext))

decrypted\_text = rsa\_decrypt(ciphertext, d, n)

decodedText = decodeAscii(decrypted\_text)

print("Decrypted text: " + str(decodedText))

**#rsa.py**

def totient(p, q):

return (p - 1) \* (q - 1)

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

def egcd(a, b):

if a == 0:

return (b, 0, 1)

g, y, x = egcd(b%a,a)

return (g, x - (b//a) \* y, y)

def modinv(a, m):

g, x, y = egcd(a, m)

if g != 1:

raise Exception('No modular inverse')

return x%m

def public\_exponent(phi):

E = 2

while gcd(E, phi) != 1:

E += 1

return E

def rsa\_encrypt(plaintext, E, n):

return pow(plaintext, E, n)

def rsa\_decrypt(ciphertext, d, n):

return pow(ciphertext, d, n)

**#ascii.py**

def encodeAscii(m):

text = ''

for x in m:

organ = str(ord(x))

if(len(organ) == 1):

organ = "90" + organ

if(len(organ) == 2):

organ = "9" + organ

text = text + organ

return(text)

def decodeAscii(m):

# Iterate over the string in steps of 3

number\_str = str(m)

text = ''

for i in range(0, len(number\_str), 3):

text = text + chr(int(number\_str[i:i+3]) % 900)

return(text)

# These functions turn the ascii into a sequence of ascii.

# I use 9’s for placeholders for leading 0’s.

**# Running on “Hello!”**

$ python main.py

107

enter message to be sent:

Hello!

p : 923978444369

q : 923978444369

n : 458764327484825650918831

phi : 458764327483405162701664

e : 3

d : 305842884988936775134443

Public key (E, n): (3, 458764327484825650918831)

Private key (d, n): (305842884988936775134443, 458764327484825650918831)

The message to be sent: Hello!

text after ascii encoding as integer : 972101108108111933

Ciphertext: 322776738291196976750205

Decrypted text: Hello!

**# Running on “a”**

$ python main.py

107

enter message to be sent:

a

p : 923978444369

q : 923978444369

n : 458764327484825650918831

phi : 458764327483405162701664

e : 3

d : 305842884988936775134443

Public key (E, n): (3, 458764327484825650918831)

Private key (d, n): (305842884988936775134443, 458764327484825650918831)

The message to be sent: a

text after ascii encoding as integer : 997

Ciphertext: 991026973

Decrypted text: a