**SYSTEM SECURITY**

**LAB 8**

**RSA ENCRYPTOR/DECRYPTOR**

**P.R.N:** 20190802058

**Name:** Krushal Milan Shah

**Aim:**

Write a C/C++/Python/JAVA program to implement RSA algorithm.

**Steps:**

1. Select two co-prime numbers as p and q
2. Compute n as the product of p and q
3. Compute (p1)\*(q-1) and store it in z
4. Select a random prime number e that is less than z
5. Compute the prime key, d as e \* mod -1 (z)
6. The cipher text is computed as message e \* mod n
7. Decryption is done as cipher d mod n.

**Code:**

import math

print("RSA Algo: written by Krushal Shah")

print("The following code is written in google colab.\n\n")

#Input Prime Numbers

print("PLEASE ENTER THE 'p' AND 'q' VALUES BELOW:")

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

q = int(input("Enter a prime number for q: "))

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

#Check if Input's are Prime

'''THIS FUNCTION AND THE CODE IMMEDIATELY BELOW THE FUNCTION CHECKS WHETHER THE INPUTS ARE PRIME OR NOT.'''

def prime\_check(a):

if(a==2):

return True

elif((a<2) or ((a%2)==0)):

return False

elif(a>2):

for i in range(2,a):

if not(a%i):

return false

return True

check\_p = prime\_check(p)

check\_q = prime\_check(q)

while(((check\_p==False)or(check\_q==False))):

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

q = int(input("Enter a prime number for q: "))

check\_p = prime\_check(p)

check\_q = prime\_check(q)

#RSA Modulus

'''CALCULATION OF RSA MODULUS 'n'.'''

n = p \* q

print("RSA Modulus(n) is:",n)

#Eulers Toitent

'''CALCULATION OF EULERS TOITENT 'r'.'''

r= (p-1)\*(q-1)

print("Eulers Toitent(r) is:",r)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

#GCD

'''CALCULATION OF GCD FOR 'e' CALCULATION.'''

def egcd(e,r):

while(r!=0):

e,r=r,e%r

return e

#Euclid's Algorithm

def eugcd(e,r):

for i in range(1,r):

while(e!=0):

a,b=r//e,r%e

if(b!=0):

print("%d = %d\*(%d) + %d"%(r,a,e,b))

r=e

e=b

#Extended Euclidean Algorithm

def eea(a,b):

if(a%b==0):

return(b,0,1)

else:

gcd,s,t = eea(b,a%b)

s = s-((a//b) \* t)

print("%d = %d\*(%d) + (%d)\*(%d)"%(gcd,a,t,s,b))

return(gcd,t,s)

#Multiplicative Inverse

def mult\_inv(e,r):

gcd,s,\_=eea(e,r)

if(gcd!=1):

return None

else:

if(s<0):

print("s=%d. Since %d is less than 0, s = s(modr), i.e., s=%d."%(s,s,s%r))

elif(s>0):

print("s=%d."%(s))

return s%r

#e Value Calculation

'''FINDS THE HIGHEST POSSIBLE VALUE OF 'e' BETWEEN 1 and 1000 THAT MAKES (e,r) COPRIME.'''

for i in range(1,1000):

if(egcd(i,r)==1):

e=i

print("The value of e is:",e)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

#d, Private and Public Keys

'''CALCULATION OF 'd', PRIVATE KEY, AND PUBLIC KEY.'''

print("EUCLID'S ALGORITHM:")

eugcd(e,r)

print("END OF THE STEPS USED TO ACHIEVE EUCLID'S ALGORITHM.")

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print("EUCLID'S EXTENDED ALGORITHM:")

d = mult\_inv(e,r)

print("END OF THE STEPS USED TO ACHIEVE THE VALUE OF 'd'.")

print("The value of d is:",d)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

public = (e,n)

private = (d,n)

print("Private Key is:",private)

print("Public Key is:",public)

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

#Encryption

'''ENCRYPTION ALGORITHM.'''

def encrypt(pub\_key,n\_text):

e,n=pub\_key

x=[]

m=0

for i in n\_text:

if(i.isupper()):

m = ord(i)-65

c=(m\*\*e)%n

x.append(c)

elif(i.islower()):

m= ord(i)-97

c=(m\*\*e)%n

x.append(c)

elif(i.isspace()):

spc=400

x.append(400)

return x

#Decryption

'''DECRYPTION ALGORITHM'''

def decrypt(priv\_key,c\_text):

d,n=priv\_key

txt=c\_text.split(',')

x=''

m=0

for i in txt:

if(i=='400'):

x+=' '

else:

m=(int(i)\*\*d)%n

m+=65

c=chr(m)

x+=c

return x

#Message

message = input("What would you like encrypted or decrypted?(Separate numbers with ',' for decryption):")

print("Your message is:",message)

#Choose Encrypt or Decrypt and Print

choose = input("Type '1' for encryption and '2' for decrytion.")

if(choose=='1'):

enc\_msg=encrypt(public,message)

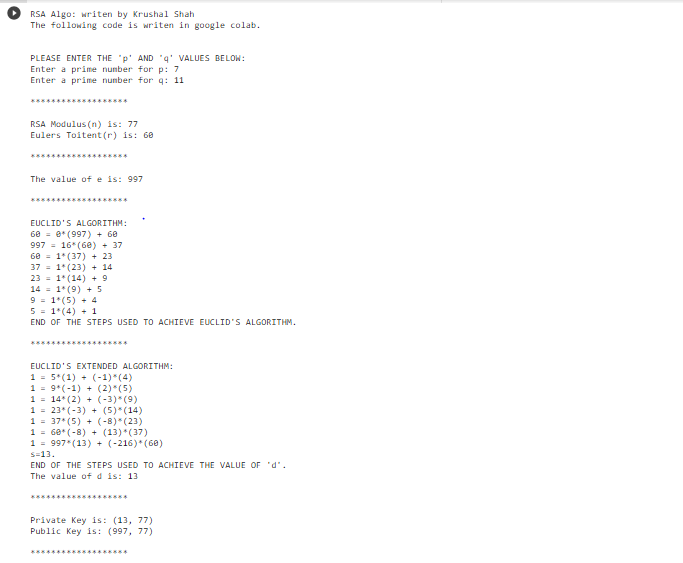
print("Your encrypted message is:",enc\_msg)

elif(choose=='2'):

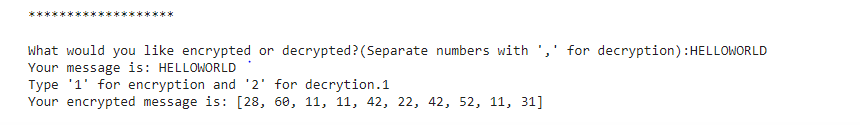
print("Your decrypted message is:",decrypt(private,message))

else:

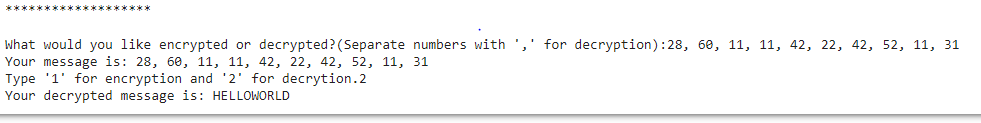
print("You entered the wrong option.")

**Results:**

*Encryption*

****

*Decryption*

****

**Conclusion:**

Successfully implemented RSA.