Re-creation of the RSA algorithm

New Mexico
Supercomputing Challenge
Final Report
3/12/18

Team Number: 132
Team Members: Logan Pierson(10th), Sarah Maurice(10th), Houston Fett(9th), Evan Custer(10th).
Email Addresses:20lpierson@sandiaprep.org, 20smaurice@sandiaprep.org, 21hfett@sandiaprep.org, 20ecuster@sandiaprep.org.
Sponsoring Teacher: Neal Holtschulte (Sandia Preparatory School)

Table of Contents

Executive summary:

The project was focused on studying how to create RSA algorithm in python. We gathered information from different websites and books to understand the RSA algorithm. We gathered information on how to recreate the RSA algorithm in python, how to convert strings to numbers. Encrypting the strings using the ascii codes within the python library then running it through the RSA algorithm to decrypt the codes. We ran this through our multiple computers to test for bugs and errors within our code. The data we've gathered from our working code is very positive as we are able to encrypt and decrypt. Yet we are wanting to get to the point where we can break the encryption. As the sentences that the user wanted to encrypt, get larger, our code grew slower. Larger numbers, as with any encryption code, slow down the code just because our processors are on the slower side.

As this code is more meant for an educational outlet for our group of new programmers. Our main problems we are gonna learn about is mainly understanding the RSA algorithm

Problem Statement:

The problems that our group ran into with our RSA program was understanding what was going on with the code that we were writing. Our group solved this problem with starting with a few short programs to gain more knowledge in the individual parts of the program before we put the hole program together. To do this we made a prime number generator, a program that used the RSA algorithm that we used to encrypt and decrypt a little string of numbers, and finally a program that converted a string of letters into a string of numbers. We brought together all of these programs into one big program that one can use to encrypt or decrypt a string.

With our program now written it is time to test the program. Our first few test showed that the program took a long time to run to calculate the prime numbers that we used, and the messages that we got back from the program did not match up with the original message that we inputted into the program. We eventually fixed that problem when we found a bug in the prime number generator. The bug that we fixed, fixed the problem that we had with the incongruence with the input and output of the program. One problem that we still have is how long the program takes to run to generate the prime numbers that we use.

Methods:

We used for loops to design the basis of the program to encrypt and decrypt the code. The program python, was used to design the code for our code. Our variables followed the original RSA algorithm variables that Ron Rivest, Adi Shamir, and Leonard Adleman. We converted our numbers to ascii values instead of the original algorithms method.

Our program first calculates our prime number list to find our largest two prime numbers. P and Q are the largest two prime numbers that are used for our calculations. These two prime numbers P and Q are then multiplied together to create N. PHI = (P-1) * (Q - 1). We run PHI, P, and Q through our function tofindE. The function tests the Greatest Common Denominator or GCD for our three variables. This function returns the value of e which increases till the variable great doesn't equal one. Then we make d another variable. This variable equals the extend function within our functions file. We apply the extend function to the variables E and PHI this equals our d variable. Our program runs these calculations before we actually have the user add in their sentence they want to encrypt.

Once the program has ran through all these pre-variables and equations, we ask the user for a sentence and from that we break that sentence into three letter strings of that sentence. Then run those 3 letter strings through a function called getNextChr which takes a letter and changes it to the ascii numerical value. We then convert these 9 digit strings of numbers to integer values. Our decrypted value equals the ((encrypted message modulus n)**d)modulus n. This number is then taken and put into a list called message_encrpt. We run every item in our list through our last loop. This loop changes 3 digit ascii codes back to letters. Lastly we put our message into a final variable called res to join the message so it's readable for us.

Results and Conclusions:

The results that we found is that the program that we wrote took a long time at the beginning to calculate the prime numbers that we use. After the primes are calculated you can input the message that you want to encrypt, then the program will give you the encrypted message that you can send to someone else.

We verified that the program was by checking what the program gave us as our decrypted message compared to the message that we wanted to encrypt. We had to debug the code to make it work like how we wanted it to. One of those problems was the prime numbers generator was not working properly and needed to be modified a little.

Our team's most significant achievement was having our program run, this was due to all of the little bugs in the program that were in important places in the code that changed the data that then was passed to the next part of the program.

Future Work:

If we were to continue this project we would modify our code to create prime numbers faster. We would also make the code ask the user for the option between encrypting or decrypting their messages. Also making our code more user friendly and easier to understand would be beneficial. As well as changing our code to converting letters to 4 digit numbers instead of 3 digit numbers to make our recreation of the RSA algorithm more secure.

Acknowledgements:

Neal Holtschulte
Kendall Pierson

Appendix:

All of our source code is below

The Encryption code

```
"""
File name : RSA
11-15-17
CS1 Encryption w/ powmod
"""
import rsa_2, isprme, string

def getNextChr(string):
    a1 = string[0]
    o = ord(a1)
    a = str(o)
    if len(a) < 3:
        a = "0"+ a
    return a

def decrypt(x, y, z):
    #print("in decrypt: x= ",x)
    a = chr(int(x))
    b = chr(int(y))
    c = chr(int(z))
    return a + b + c

#so user knows to wait and not kill the program
print("Please Wait While We Do Our Calcuation. This Should Take About a Minute ")

#making a large list of prime numbers that goes up to a very large number
"""
primes_list = []
for i in range(9999999990000,10000000000000):
    if isprme.isPrime(i)==True:
        primes_list.append(i)
"""
#Create a variable named n and give it the value of the product of p and q
P=  49993
#primes_list[-1]
print(P)
```

```
Q=49999
#primes_list[-2]
print(Q)

n=P*Q
print("N")
print(n)

#Create a variable named phi and give it the value of the product of p-1 and q-1
PHI=(P-1)*(Q-1)
print(PHI)

#Create a variable named e and set it equal to the result of calling your tofindE function and
passing it the argument phi.
E=rsa_2.tofindE(PHI,P,Q)
print("E")
print(E)

#Create a variable named d and set it equal to the result of calling your extend function and
passing extend the arguments e and phi in that order.
d=rsa_2.extend(E,PHI)
print(d)
print("To Make Sure that the Program does what you want it to do. Please copy and paste you
choice into the input.")
#allows user to choice if they want to encrpt or decrpty


#getting the string that you want to encrypt
encrpt=[]
ecrypted_message=[]
message_org=input("What is your message that you want to encrypt?: ")

while len(message_org) % 3 != 0:
    message_org = message_org + " "

for i in range(0,len(message_org),3):
    a = getNextChr(message_org[i])
    b = getNextChr(message_org[i+1])
```

```
    c = getNextChr(message_org[i+2])
    encrpt.append (int(a + b + c))

            #encrypts the numerical message and then prints it
for i in range(0,len(encrpt)):
    ecrypted_message.append(int((encrpt[i]**E)%n))
print("This is Your Encrypted Message:")
print(ecrypted_message)

decrypted_message=[]

    #takes the numerical string and decrypting is
for i in range(len(ecrypted_message)):
    decrypted_message.append(rsa_2.powMod(ecrypted_message[i],d,n))
print("decrypted_message")
print(decrypted_message)

message=[]

for i in range(len(decrypted_message)):
    obj = str(decrypted_message[i])
    while len(obj)<9:
        obj = "0"+obj
    x = obj[0:3]
    y = obj[3:6]
    z = obj[6:9]
    r = decrypt(x, y, z)
    message.append(r)
        #convert numeric message to a string by most likely calling a function in another program
and then prints it

res="".join(message)
print("string message")
print(res)
Prime Number Calculator
“”””
File name : isPrime
11-08-2017
CS1 isPrime
```

```
“””
import math
def isPrime(x):
    if x%2!=0:
        for i in range (3,int(math.sqrt(x)+1),2):
            if x%i==0:
                return False
        return True
    else:
        return False
RSA Functions
“””
File name : Encryption
11-14-17
CS1 Encryption
"""
import math
def GCDintern(m,n):

    a = m%n
    while (a != 0):
        m = n
        n = a
        a = m%n
    return n

def GCD(m,n):

    if (m == 0):
        return n

    if (n == 0):
        return m
    if (m < 0):
        return GCD(-m,n)

    if (n < 0):
        return GCD(m,-n)
```

```
    if (m > n):
        return GCDintern(n,m)

    return GCDintern(m,n)


def tofindE(PHI,P,Q):
    great = 0
    e = 2
    while (great != 1):
        e = e + 1
        great = GCD(e,PHI)
        PHI = (P - 1) * (Q - 1)
    return e

def extend(E,PHI):
    u1 = 1
    u2 = 0
    u3 = PHI
    v1 = 0
    v2 = 1
    v3 = E

    while (v3 != 0):
        q = math.floor(u3/v3)
        t1 = u1 - q * v1
        t2 = u2 - q * v2
        t3 = u3 - q * v3

        u1 = v1
        u2 = v2
        u3 = v3

        v1 = t1
        v2 = t2
        v3 = t3
        z = 1

    uu = u1
```

```
    vv = u2

    if (vv < 0):
            inverse = vv + PHI

    else:
       inverse = vv
    return inverse

def powMod(n,e,m):
    if (m == 0 or e < 0):

       return 0

    res = 1
    pow = n
    e1  = e

    while (e1 != 0):
       d   = e1%2
       e1  = math.floor(e1/2)
    if (d == 1):
       res = (res*pow)%m
       pow = (pow*pow)%m
    if (res < 0):
       res += m
    return res
```