**Project 7**

**Topic**

* RSA

**Objective**

* To develop a deeper understanding of RSA

**Constraint**

* Work must run using Sage under Linux

**Problem Specification**

Write a functioning RSA cipher.  The cipher will be a Sage program with these functions, callable from the Sage prompt, and other functions as described below:

Pre: *size* is an exponent, as in *2^size*, where size >= 100

Post: program returns the RSA public key,(n,e)  and the private key, d.

Comments: e is prime and of the form 2^r + 1, where *r* is an integer.  It does not have to be large. 17 is a reasonable choice, though you might impress me by generating one in the proper form. n, d are generated according to the RSA algorithm.

key\_gen(size)

Pre: plain\_text is a text string, chosen by the user.  “Than you can understand,” is an example.  There are no limits on its size.  *e, n* are returned by key\_gen

Post: returns the encryption of plain\_text, using the RSA algorithm.

encrypt(plain\_text, e, n)

Pre: d, n are returned by key\_gen.  c is returned by encrypt.

Post: returns the decryption of the cipher\_text, using the RSA algorithm

decrypt(c,d,n):

**Ancillary Functions**

The project requires several ancillary functions:

* RSA works with integers.  Include the functions in my txt\_num\_conv.sage (described in Project 6), and call them where needed.
* For RSA to work, the message m must be less than n, the modulus.   Your cipher must contain a function or functions that decompose m and c into blocks of the correct size. You’ll find the process described on p. 97 of McAndrew.

Below is the code I produced for this assignment:

def gen\_key(size, e):

p = random\_prime((2^size)-1, 2^(size-1))

q = random\_prime((2^size)-1,2^(size-1))

n = p \* q

d = inverse\_mod(e,(p-1)\*(q-1))

return n,e,d

def txt\_to\_num(msg\_in):

msg\_idx = list(map(ord, msg\_in))

print(msg\_idx)

num = ZZ(msg\_idx,256)

print('this is after ZZ function: ', num)

return num

def num\_to\_txt(num\_in):

msg\_idx = list(num\_in.digits(256))

print('this is after num.digits: ', msg\_idx)

m = list(map(chr,msg\_idx))

print('this is m in num 2 txt: ', m)

m = ''.join(m)

return m

def encrypt(plain\_txt,e,n):

m = txt\_to\_num(plain\_txt)

print('Encrypt, this is n: ', n)

print('Encrypt, this is e: ', e)

c = power\_mod(m, e ,n)

print('Encrypt, this is m: ', m)

return c

def decrypt(c,d,n):

m = power\_mod(c,d,n)

print('Decrypt, This is m : ', m)

plain\_txt = num\_to\_txt(m)

print('Decrypt, after num to text: ', plain\_txt)

return plain\_txt

def main():

size = input('Enter in values for size ')

size = int(size)

e = input('Enter in values for e ')

e = int(e)

plain\_txt = input('Enter in a sentence ')

n, e, d = gen\_key(size, e)

c = encrypt(plain\_txt,e,n)

p = decrypt(c,d,n)

Note: The code below was provided by the professor for the functions:

num\_to\_txt(msg\_in) and txt\_to\_num(num\_in)

I modified these functions a bit in order for my program to run successfully.

Text

Description automatically generated