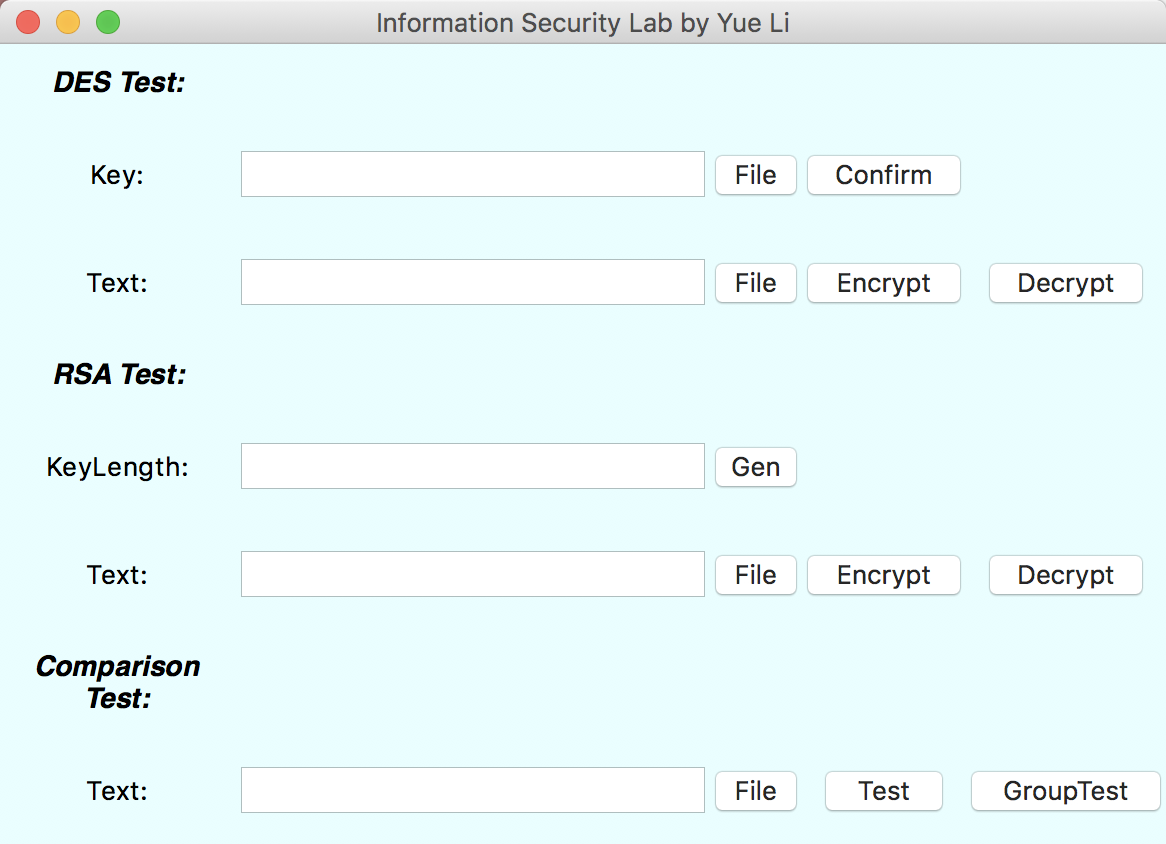
**Project 2 Lab Report**

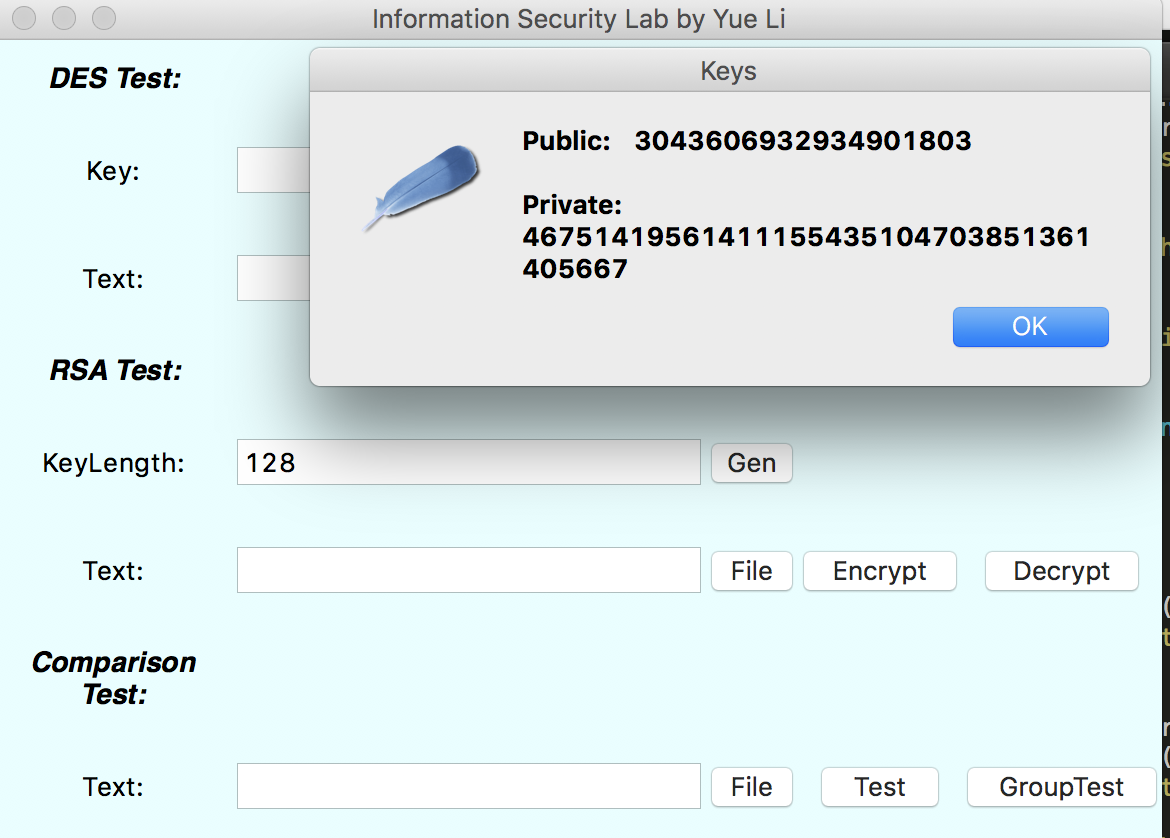
李悦 14080214

1. **Demonstration**

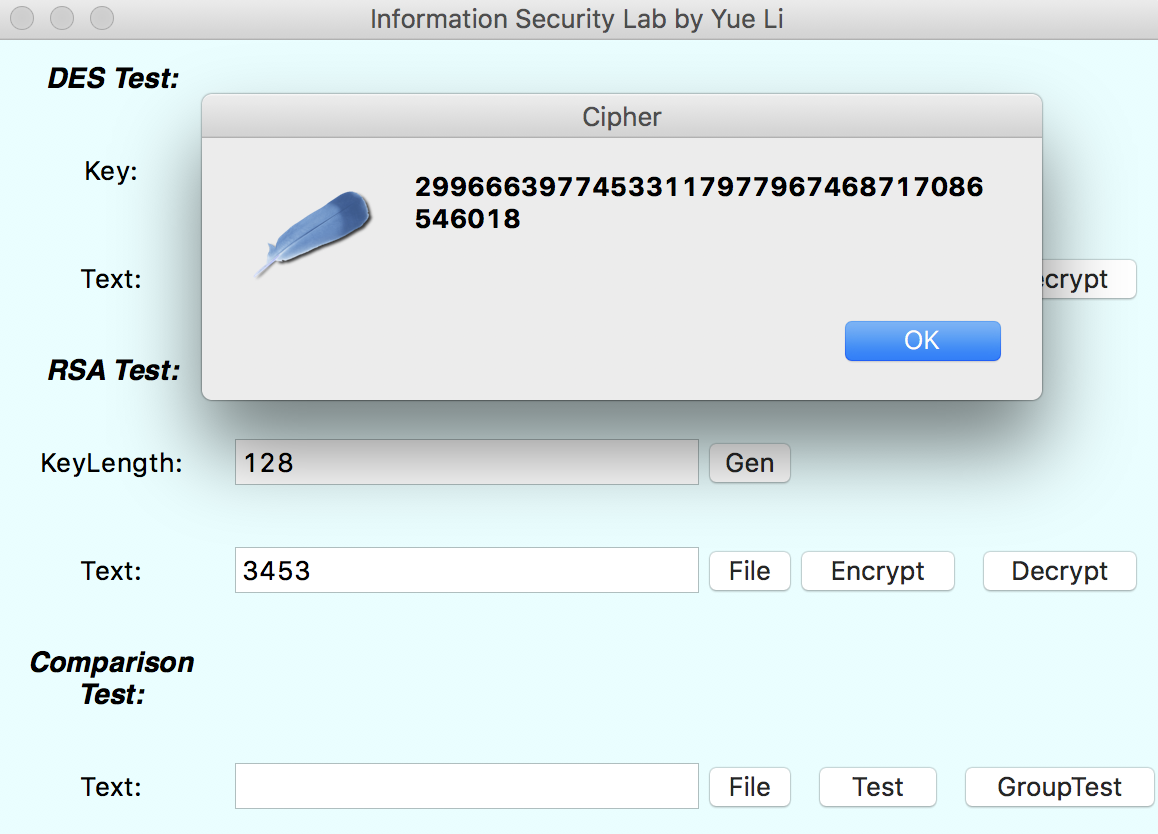
**Start state**

****

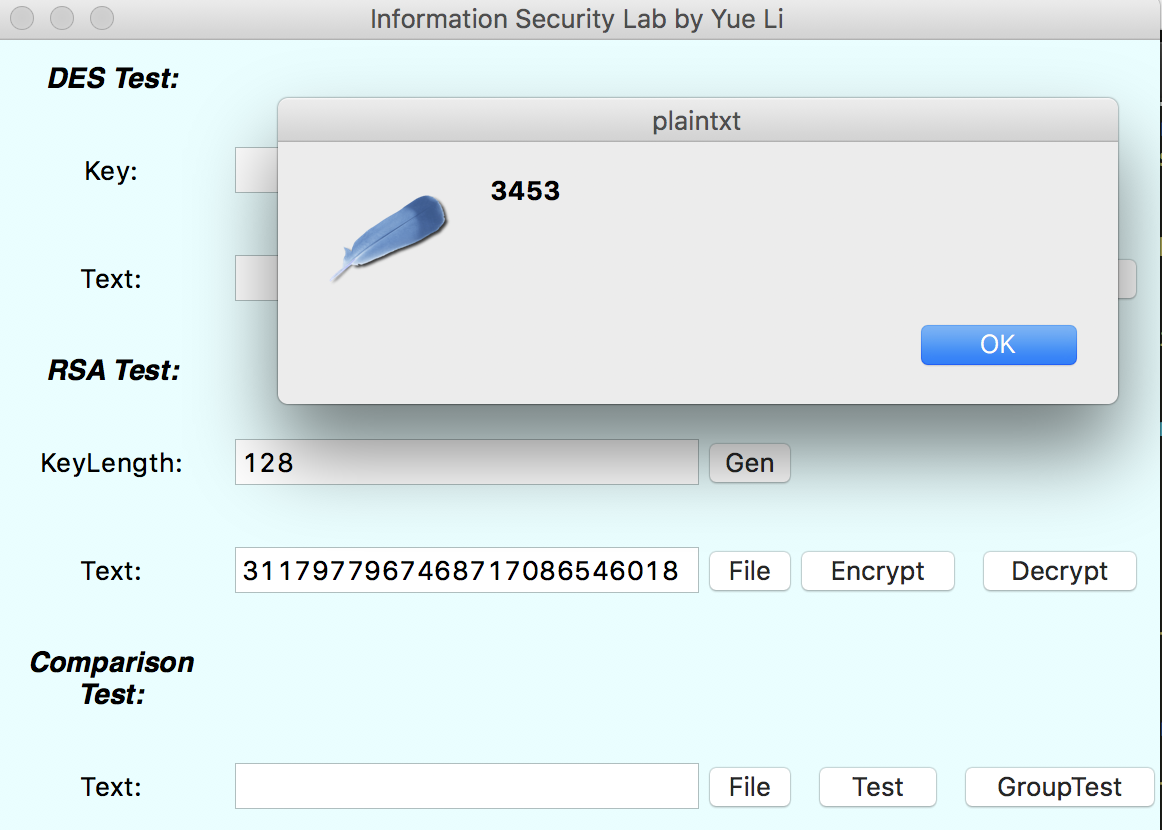
**RSA Key Inputting**

****

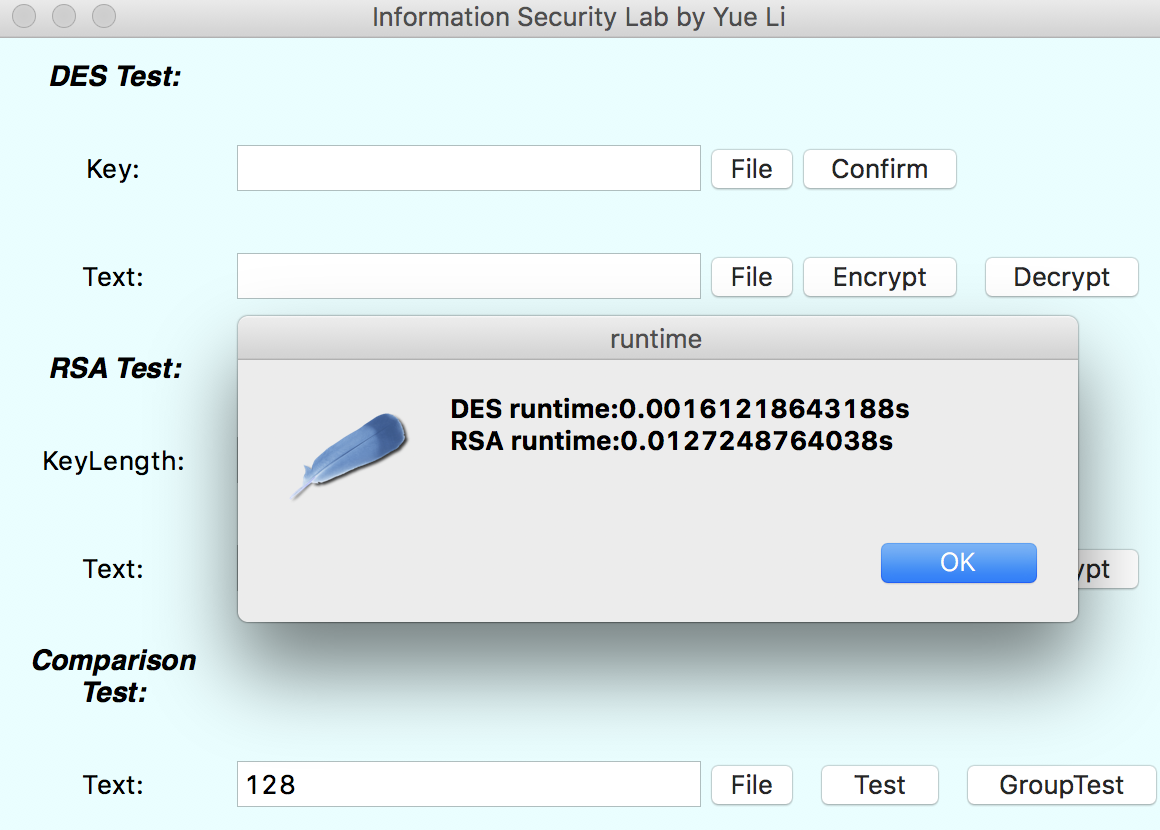
**Encryption**

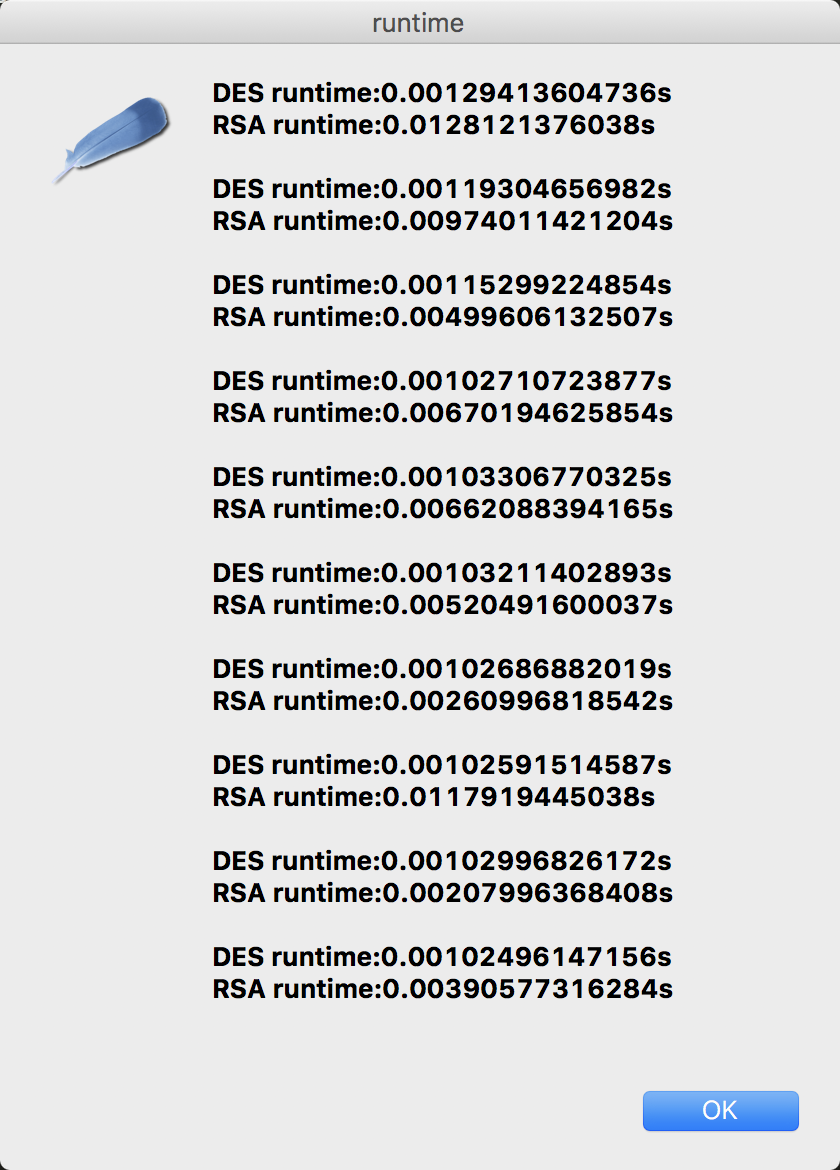
****

**Decryption**

****

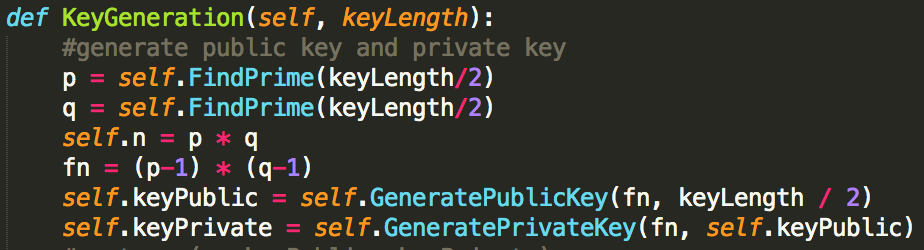
**Runtime Test**

****

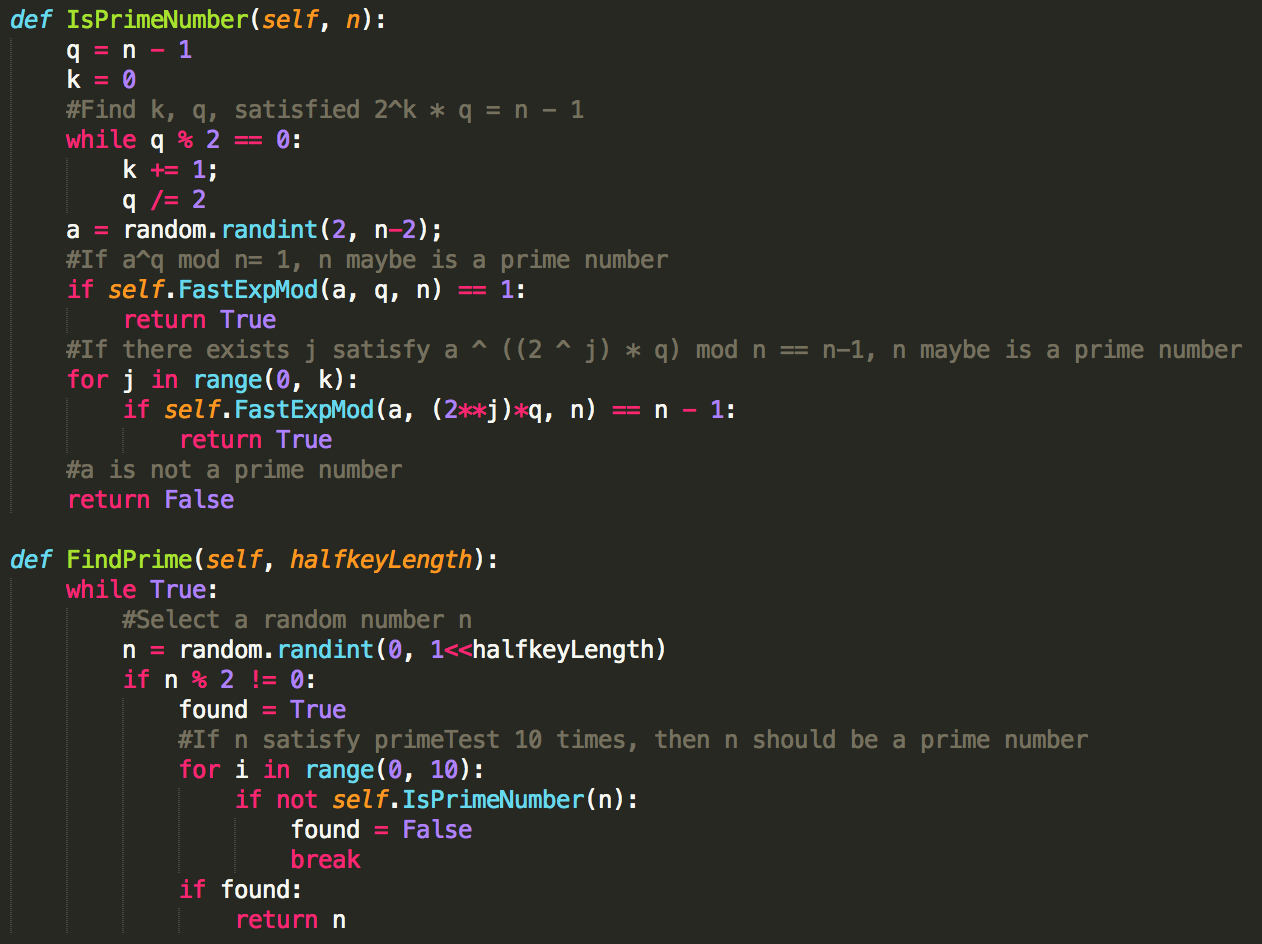
****

1. **RSA Encryption algorithm and the implementation in Python**

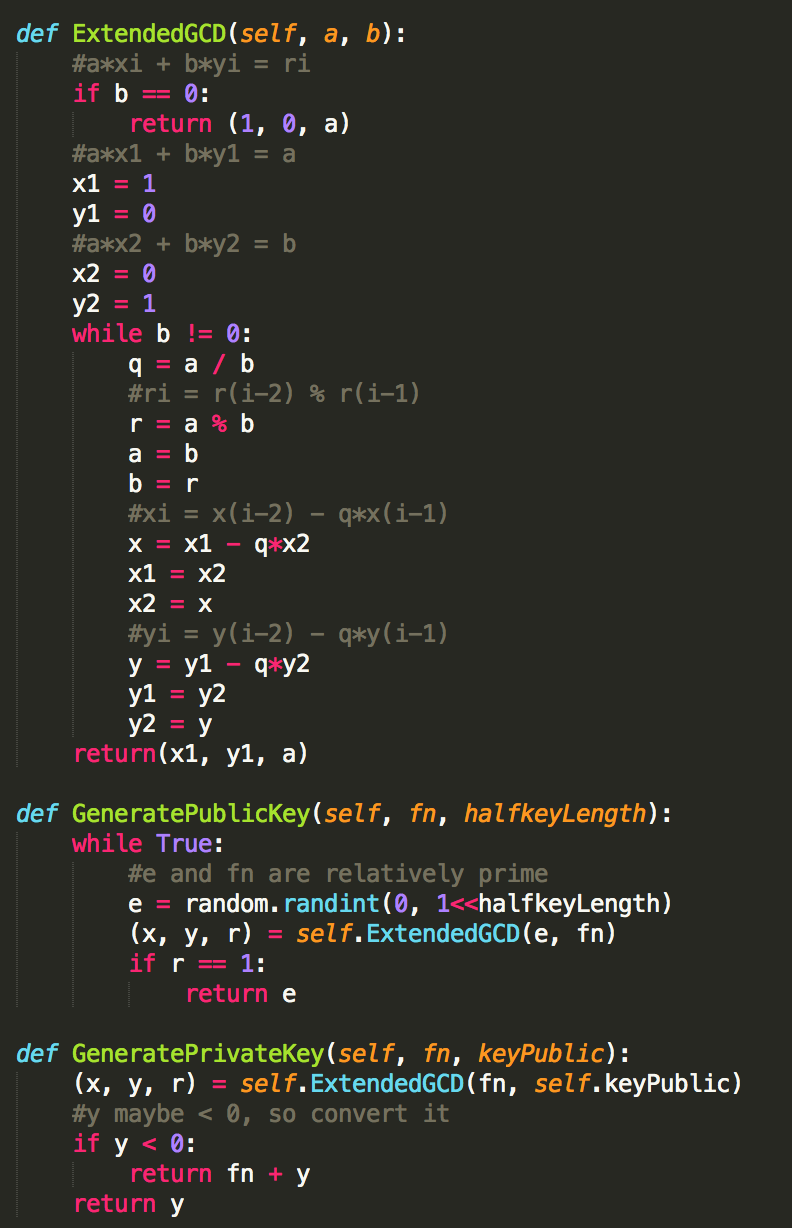
**Step 1: Key Generation.**

****

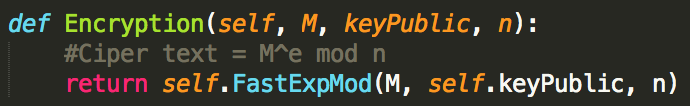
* 1. Choose two distinct prime numbers p and q.



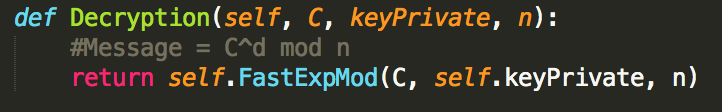
* 1. Compute n = pq.
  2. Compute the number of such numbers that are smaller than n and are relatively prime to n.
  3. Choose an integer p such that p<n and is relatively prime to fi of n.
  4. Choose an integer s such that (p\*s mod fi(n))) = 1.
  5. s is the private key, p is the public key



**Step 2: Encryption**

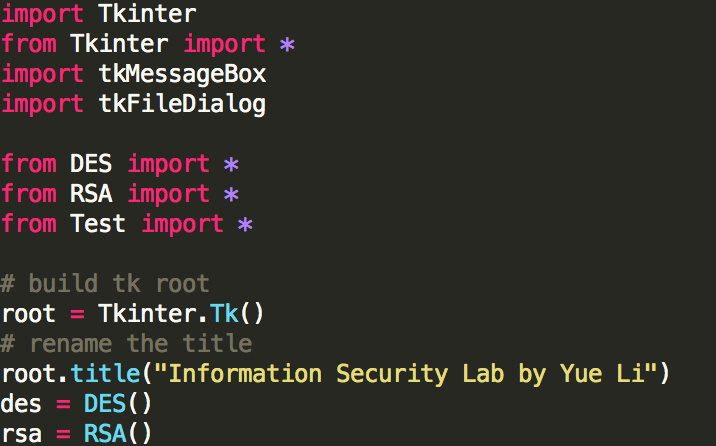


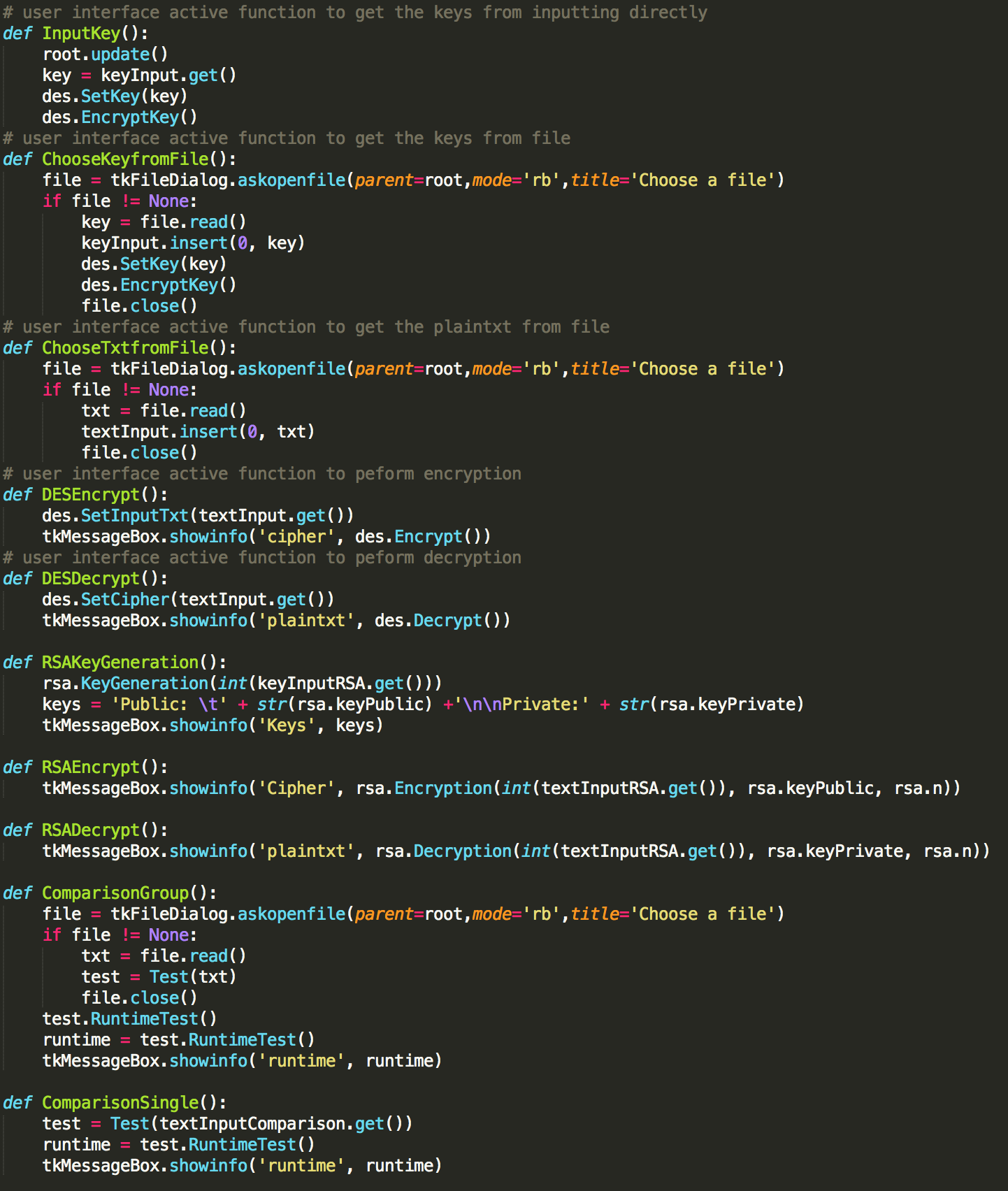
1. **RSA Decryption algorithm and the implementation in Python**

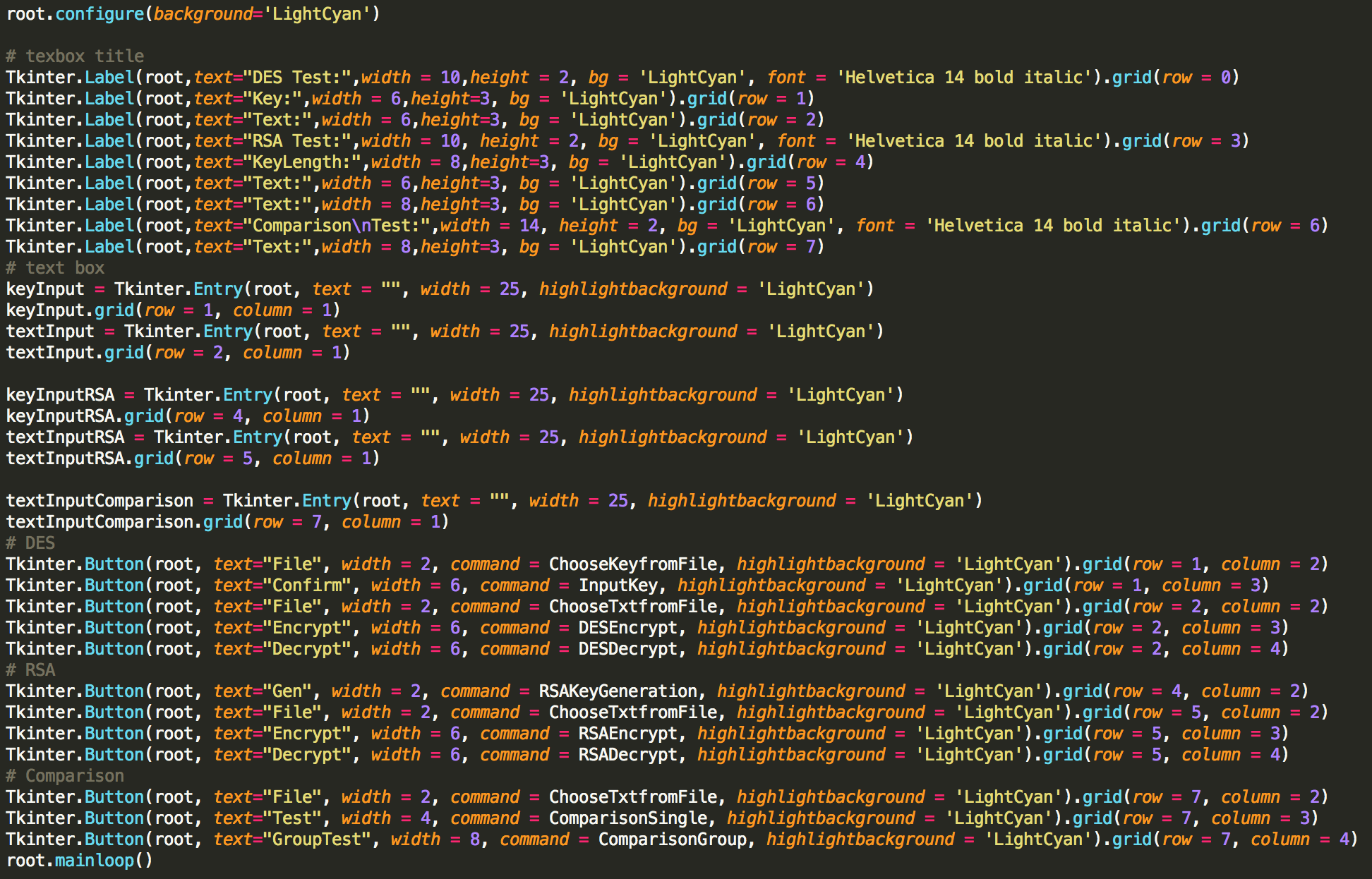


1. **User Interface and the implementation in Python**

User interface is created via python Tkinter library.

****

****

****

1. **Other function and the implementation in Python**

For each line in the input text file compute it’s runtime when using DES algorithm and when using RSA algorithm. Store this runtime as string to return.

****

1. **Complete source codes in Python**
2. Ui.py

import Tkinter

from Tkinter import \*

from Tkinter import \*

import tkMessageBox

import tkFileDialog

from DES import \*

from RSA import \*

from Test import \*

# build tk root

root = Tkinter.Tk()

# rename the title

root.title("Information Security Lab by Yue Li")

des = DES()

rsa = RSA()

# user interface active function to get the keys from inputting directly

def InputKey():

root.update()

key = keyInput.get()

des.SetKey(key)

des.EncryptKey()

# user interface active function to get the keys from file

def ChooseKeyfromFile():

file = tkFileDialog.askopenfile(parent=root,mode='rb',title='Choose a file')

if file != None:

key = file.read()

keyInput.insert(0, key)

des.SetKey(key)

des.EncryptKey()

file.close()

# user interface active function to get the plaintxt from file

def ChooseTxtfromFile():

file = tkFileDialog.askopenfile(parent=root,mode='rb',title='Choose a file')

if file != None:

txt = file.read()

textInput.insert(0, txt)

file.close()

# user interface active function to peform encryption

def DESEncrypt():

des.SetInputTxt(textInput.get())

tkMessageBox.showinfo('cipher', des.Encrypt())

# user interface active function to peform decryption

def DESDecrypt():

des.SetCipher(textInput.get())

tkMessageBox.showinfo('plaintxt', des.Decrypt())

def RSAKeyGeneration():

rsa.KeyGeneration(int(keyInputRSA.get()))

keys = 'Public: \t' + str(rsa.keyPublic) +'\n\nPrivate:' + str(rsa.keyPrivate)

tkMessageBox.showinfo('Keys', keys)

def RSAEncrypt():

tkMessageBox.showinfo('Cipher', rsa.Encryption(int(textInputRSA.get()), rsa.keyPublic, rsa.n))

def RSADecrypt():

tkMessageBox.showinfo('plaintxt', rsa.Decryption(int(textInputRSA.get()), rsa.keyPrivate, rsa.n))

def ComparisonGroup():

file = tkFileDialog.askopenfile(parent=root,mode='rb',title='Choose a file')

if file != None:

txt = file.read()

test = Test(txt)

file.close()

test.RuntimeTest()

runtime = test.RuntimeTest()

tkMessageBox.showinfo('runtime', runtime)

def ComparisonSingle():

test = Test(textInputComparison.get())

runtime = test.RuntimeTest()

tkMessageBox.showinfo('runtime', runtime)

root.configure(background='LightCyan')

# texbox title

Tkinter.Label(root,text="DES Test:",width = 10,height = 2, bg = 'LightCyan', font = 'Helvetica 14 bold italic').grid(row = 0)

Tkinter.Label(root,text="Key:",width = 6,height=3, bg = 'LightCyan').grid(row = 1)

Tkinter.Label(root,text="Text:",width = 6,height=3, bg = 'LightCyan').grid(row = 2)

Tkinter.Label(root,text="RSA Test:",width = 10, height = 2, bg = 'LightCyan', font = 'Helvetica 14 bold italic').grid(row = 3)

Tkinter.Label(root,text="KeyLength:",width = 8,height=3, bg = 'LightCyan').grid(row = 4)

Tkinter.Label(root,text="Text:",width = 6,height=3, bg = 'LightCyan').grid(row = 5)

Tkinter.Label(root,text="Text:",width = 8,height=3, bg = 'LightCyan').grid(row = 6)

Tkinter.Label(root,text="Comparison\nTest:",width = 14, height = 2, bg = 'LightCyan', font = 'Helvetica 14 bold italic').grid(row = 6)

Tkinter.Label(root,text="Text:",width = 8,height=3, bg = 'LightCyan').grid(row = 7)

# text box

keyInput = Tkinter.Entry(root, text = "", width = 25, highlightbackground = 'LightCyan')

keyInput.grid(row = 1, column = 1)

textInput = Tkinter.Entry(root, text = "", width = 25, highlightbackground = 'LightCyan')

textInput.grid(row = 2, column = 1)

keyInputRSA = Tkinter.Entry(root, text = "", width = 25, highlightbackground = 'LightCyan')

keyInputRSA.grid(row = 4, column = 1)

textInputRSA = Tkinter.Entry(root, text = "", width = 25, highlightbackground = 'LightCyan')

textInputRSA.grid(row = 5, column = 1)

textInputComparison = Tkinter.Entry(root, text = "", width = 25, highlightbackground = 'LightCyan')

textInputComparison.grid(row = 7, column = 1)

# DES

Tkinter.Button(root, text="File", width = 2, command = ChooseKeyfromFile, highlightbackground = 'LightCyan').grid(row = 1, column = 2)

Tkinter.Button(root, text="Confirm", width = 6, command = InputKey, highlightbackground = 'LightCyan').grid(row = 1, column = 3)

Tkinter.Button(root, text="File", width = 2, command = ChooseTxtfromFile, highlightbackground = 'LightCyan').grid(row = 2, column = 2)

Tkinter.Button(root, text="Encrypt", width = 6, command = DESEncrypt, highlightbackground = 'LightCyan').grid(row = 2, column = 3)

Tkinter.Button(root, text="Decrypt", width = 6, command = DESDecrypt, highlightbackground = 'LightCyan').grid(row = 2, column = 4)

# RSA

Tkinter.Button(root, text="Gen", width = 2, command = RSAKeyGeneration, highlightbackground = 'LightCyan').grid(row = 4, column = 2)

Tkinter.Button(root, text="File", width = 2, command = ChooseTxtfromFile, highlightbackground = 'LightCyan').grid(row = 5, column = 2)

Tkinter.Button(root, text="Encrypt", width = 6, command = RSAEncrypt, highlightbackground = 'LightCyan').grid(row = 5, column = 3)

Tkinter.Button(root, text="Decrypt", width = 6, command = RSADecrypt, highlightbackground = 'LightCyan').grid(row = 5, column = 4)

# Comparison

Tkinter.Button(root, text="File", width = 2, command = ChooseTxtfromFile, highlightbackground = 'LightCyan').grid(row = 7, column = 2)

Tkinter.Button(root, text="Test", width = 4, command = ComparisonSingle, highlightbackground = 'LightCyan').grid(row = 7, column = 3)

Tkinter.Button(root, text="GroupTest", width = 8, command = ComparisonGroup, highlightbackground = 'LightCyan').grid(row = 7, column = 4)

root.mainloop()

1. Rsa.py

import random

class RSA(object):

n = 0;

keyPublic = 0

keyPrivate = 0

"""docstring for RSA"""

def \_\_init\_\_(self):

pass

def FastExpMod(self, b, e, m):

"""

e = e0\*(2^0) + e1\*(2^1) + e2\*(2^2) + ... + en \* (2^n)

b^e = b^(e0\*(2^0) + e1\*(2^1) + e2\*(2^2) + ... + en \* (2^n))

= b^(e0\*(2^0)) \* b^(e1\*(2^1)) \* b^(e2\*(2^2)) \* ... \* b^(en\*(2^n))

b^e mod m = ((b^(e0\*(2^0)) mod m) \* (b^(e1\*(2^1)) mod m) \* (b^(e2\*(2^2)) mod m) \* ... \* (b^(en\*(2^n)) mod m) mod m

"""

result = 1

while e != 0:

if (e&1) == 1:

# ei = 1, then mul

result = (result \* b) % m

e >>= 1

# b, b^2, b^4, b^8, ... , b^(2^n)

b = (b\*b) % m

return result

def IsPrimeNumber(self, n):

q = n - 1

k = 0

#Find k, q, satisfied 2^k \* q = n - 1

while q % 2 == 0:

k += 1;

q /= 2

a = random.randint(2, n-2);

#If a^q mod n= 1, n maybe is a prime number

if self.FastExpMod(a, q, n) == 1:

return True

#If there exists j satisfy a ^ ((2 ^ j) \* q) mod n == n-1, n maybe is a prime number

for j in range(0, k):

if self.FastExpMod(a, (2\*\*j)\*q, n) == n - 1:

return True

#a is not a prime number

return False

def FindPrime(self, halfkeyLength):

while True:

#Select a random number n

n = random.randint(0, 1<<halfkeyLength)

if n % 2 != 0:

found = True

#If n satisfy primeTest 10 times, then n should be a prime number

for i in range(0, 10):

if not self.IsPrimeNumber(n):

found = False

break

if found:

return n

def ExtendedGCD(self, a, b):

#a\*xi + b\*yi = ri

if b == 0:

return (1, 0, a)

#a\*x1 + b\*y1 = a

x1 = 1

y1 = 0

#a\*x2 + b\*y2 = b

x2 = 0

y2 = 1

while b != 0:

q = a / b

#ri = r(i-2) % r(i-1)

r = a % b

a = b

b = r

#xi = x(i-2) - q\*x(i-1)

x = x1 - q\*x2

x1 = x2

x2 = x

#yi = y(i-2) - q\*y(i-1)

y = y1 - q\*y2

y1 = y2

y2 = y

return(x1, y1, a)

def GeneratePublicKey(self, fn, halfkeyLength):

while True:

#e and fn are relatively prime

e = random.randint(0, 1<<halfkeyLength)

(x, y, r) = self.ExtendedGCD(e, fn)

if r == 1:

return e

def GeneratePrivateKey(self, fn, keyPublic):

(x, y, r) = self.ExtendedGCD(fn, self.keyPublic)

#y maybe < 0, so convert it

if y < 0:

return fn + y

return y

def KeyGeneration(self, keyLength):

#generate public key and private key

p = self.FindPrime(keyLength/2)

q = self.FindPrime(keyLength/2)

self.n = p \* q

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

self.keyPublic = self.GeneratePublicKey(fn, keyLength / 2)

self.keyPrivate = self.GeneratePrivateKey(fn, self.keyPublic)

# return (n, keyPublic, keyPrivate)

def Encryption(self, M, keyPublic, n):

#Ciper text = M^e mod n

return self.FastExpMod(M, self.keyPublic, n)

def Decryption(self, C, keyPrivate, n):

#Message = C^d mod n

return self.FastExpMod(C, self.keyPrivate, n)

1. Test.py

from DES import \*

from RSA import \*

from time import time

class Test(object):

"""docstring for Test"""

def \_\_init\_\_(self, testData):

self.testData = testData

self.desRuntime = []

self.rsaRuntime = []

self.runtime = ''

def RuntimeTest(self):

data = self.testData.strip().split('\n')

for i in range(len(data)):

start = time()

des = DES()

key = '0001110111001011011010000011101011011001010000001111110010110011'

des.SetKey(key)

des.EncryptKey()

des.SetInputTxt(data[i])

des.SetCipher(des.Encrypt())

desResult = des.Decrypt()

stop = time()

self.runtime += ('DES runtime:' + str(stop-start) + "s\n")

start = time()

rsa = RSA()

rsa.KeyGeneration(128)

rsaResult = rsa.Decryption(rsa.Encryption(int(data[i]), rsa.keyPublic, rsa.n), rsa.keyPrivate, rsa.n)

stop = time()

self.runtime += ('RSA runtime:' + str(stop-start) + "s\n\n")

return self.runtime