1. 致歉:

老师，我今晚手动实现了一个C+环境下的大整数类，无奈难以运行，实在是短时间内没有办法完成C环境下的RSA的全部源码

1. Python实现

我为了解决溢出的问题，采用了Python来处理，Python采用无限位数的特点，溢出的问题很好的可以解决

并且利用面向对象的方式实现并封装，对外的封装性良好，内部的实现接口也改了几遍

目前的限定的大素数的范围是30亿—40亿，Python的无限位数的特定很优良，所以在creat\_Big\_Prime函数中可以增加模块来确定用户输入的位数，从而提高加密的安全性，这一点是可以实现的

1. RSA类首部添加了\_\_doc\_\_类说明文档，执行语句RSA.\_\_doc\_\_可以查看类文档，有大量的说明，和我剩下的最后一个关于RSA算法的问题
2. RSA类内的函数都有些相应的说明文档，运行\_\_doc\_\_属性即可查看，或者查看注释也可以，注释写的还算详细
3. 增加了文档读取功能，将文档添加在Python文件的同一目录即可对文档加密并生成加密文档（和该程序在同一个目录下）

Python Source Code:

from random import \*

from time import \*

import math

import string

class RSA:

'''因为Python支持大数计算，没有对位数进行限制，只不过在处理的数据的额大小不同会采取不同的处理机制

所以，为了避免C++没有大整数的类的支持导致的快速幂的计算溢出的情况，我才用Python来实现一次RSA的全部算法流程

算法的流程包括

1.Miller-Rabin判断大素数（生成随机素数的判断）:Miller\_Rabin

2.扩展欧几里得求逆元（求解私钥）:Extends\_gcd

3.欧几里得判断是否互素（公钥的选取）:GCD

4.快速幂取模算法:quick\_mod

5.对于data的缓存区，我们暂时存储一行的解密文本并直接存储在我们的开辟的文件中

6.问题，是不是生成负数的私钥？

7.目前因为学识有限，只会对英文文本加密，中文文本加密涉及到转码问题，一直没有解决

现在的程序只能处理ASCII文本和数字文本'''

#秘钥必备元素

\_\_p=0

\_\_q=0

\_\_n=0

\_\_Euler\_n=0

\_\_private\_key=0 #私钥

\_\_public\_key=0 #公钥

#upper important

#公有类变量,这里只是保存一份拷贝，方便检查调试，实际运行我们可以删去

data=[]

#公有成员，存储解密后的字符串

saving=[] #存储上次的输入的地址信息

def \_\_init\_\_(self): #考虑的不全面，把构造函数写的太冗长，写成了main函数

self.saving=[]

self.data=[]

self.p=RSA.creat\_Big\_Prime(self)

self.q=RSA.creat\_Big\_Prime(self)

self.n=self.p\*self.q

self.Euler\_n=(self.p-1)\*(self.q-1)

print("请输入公钥，注意，为了保证算法的正确性\n请保证输入的公钥和Euler\_n(%d)互素"%self.Euler\_n)

while True:

k=eval(input("大公钥:"))

if RSA.GCD(self,k,self.Euler\_n)==1:

print("大公钥选取成功,现在开始生成私钥")

time=clock()

x=[0]

y=[0]

x[0]=0

y[0]=0

RSA.Extends\_gcd(self,k,self.Euler\_n,x,y)

if x[0]<0:

print("私钥生成失败（负数）,重新生成")

else:

print("私钥生成成功，耗时:%d"%(clock()-time))

self.private\_key=x[0]

self.public\_key=k

return

else:

print("公钥选取不满足互素的条件，请重新选择")

while RSA.GCD(self,k,self.Euler\_n)!=1: #向前探测

k+=7

print("推荐数字:%d作为公钥"%k)

return

def quick\_mod(self,a,b,c):

a=a%c

ans=1

while b!=0:

if (b&1)==1:

ans=(ans\*a)%c

b>>=1

a=(a\*a)%c

return ans

def Miller\_Rabin(self,n):

'''

这里选择测试的底数是2,3,7,61,23251在40w亿一下的数据中Miller-Rabin100%测试正确'''

t=0

p=n-1

while (p&1)==0:

t+=1

p>>=1

list\_atom=[2,3,7,61,23251]

for var in list\_atom:

x=RSA.quick\_mod(self,var,p,n)

for i in range(t):

y=RSA.quick\_mod(self,x,2,n)

if y==1 and x!=1 and x!=n-1:

return False

x=y

if x!=1:

return False

else:

return True

def GCD(self,a,b):

'''这里为了书写方便，GCD写的很简便，但是我们工程项目中最好不要用这种形式'''

if b==0:

return a

else:

return RSA.GCD(self,b,a%b)

def Extends\_gcd(self,a,b,x,y): #小心这里的x，y为了达到引用效果，我们要使用[]的可变容器作为参数传递进去

if b==0:

x[0]=1

y[0]=0

return a

t=RSA.Extends\_gcd(self,b,a%b,x,y)

w=[]

w.append(x[0])

x[0]=y[0]

y[0]=w[0]-((int)(a/b))\*y[0]

return t

#def set\_data(): #设置我们的参数

def encoding(self): #加密

filename=input("请输入你要加密的文本的地址，保证地址和Python文件同一目录下\n")

infile=open(filename,'r',encoding="UTF-8")

outfilename=input("请输入要存放密文的文本的地址\n")

self.saving.append(outfilename)

outfile=open(outfilename,'w',encoding="UTF-8")

for line in infile:

line=list(line)

for p in line:

if p not in [0,1,2,3,4,5,6,7,8,9]:

p=ord(p)

p=RSA.quick\_mod(self,p,self.public\_key,self.n)

self.data.append(p)

p=str(p) #转化成字符串写入文本文件

outfile.write(p+'\n') #一行一行，解密必须

infile.close()

outfile.close()

def decoding(self): #解密

'''我们在decoding函数中解密的文本直接打印出来,有提示信息'''

print("上次加密的文本地址:",end="")

print(self.saving)

filename=input("请输入你要解密的文件的地址\n")

infile=open(filename,'r',encoding="UTF-8")

kkk=[]

print("------------------------------------------")

print("解密内容如下:")

for line in infile:

kkk.append(line)

p=(int)(line)

p=RSA.quick\_mod(self,p,self.private\_key,self.n)

p=chr(p)

print(p,end="") #学习了不换行打印的函数技巧，添加参数end=""

infile.close()

print("\n------------------------------------------")

print("\n为了检验是不是真的加密成功，密文输出如下")

for var in kkk:

print(var,end="")

def print\_RSA(self): #打印类的相关信息

#????为什么没有定义????很神奇的在\_\_init\_\_中self制定再在这里self制定就可以了，为什么???????

print("RSA本次生成信息(以下内容只是为了方便调试，正式测试请删除)")

print("公共模数n:%d"%self.n)

print("大素数:\n%d\n%d"%(self.p,self.q))

print("Euler\_n:%d"%self.Euler\_n)

print("公钥:%d"%self.public\_key)

print("私钥:%d"%self.private\_key)

def creat\_Big\_Prime(self): #生成大素数

time=clock()

x=randint(30000000000000,40000000000000) #30w亿到40w亿，Miller-Rabin在此范围内100%正确,选择了号的2,3,7,61,23251

while 1:

if RSA.Miller\_Rabin(self,x):

print("大素数生成成功，耗时:%d"%(clock()-time))

return x

else: #往后探测

if x&1 : #避免探测偶数

x+=2

else: #转化探测奇数

x+=1

test=RSA()

test.print\_RSA() #测试打印RSA类的类成员信息，方便调试

test.encoding()

test.decoding()

运行效果示意:附图如下

test1.txt是我本地PC机上的同一目录下的文档

test2.txt利用w权限生成一个txt文档

另外，因为选择的随机生成素数的算法还不是很完善（自己思考的算法）所以在极少数非常特殊的情况下，会出现程序莫名崩掉的情况，重新运行即可

