**南京航空航天大学**

**课 程 设 计 报 告**



**课 程 名 称 密码学课程设计**

**学 院 计算机科学与技术 年 级 2014**

**学 生 姓 名 闫珺 学 号 161420219**

**开 课 时 间 2016 至 2017 学年第 一 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目**  **名 称** | **实验一、古典密码** | **成绩** |  |

**一、实验目的**

通过实现简单的古典密码算法，理解密码学的相关概念如明文（plaintext）、密文（ciphertext）、加密密钥（encryption key）、解密密钥（decryption key）、加密算法(encryption algorithm)、解密算法（decryption algorithm）等。

**二、实验内容**

**1）**用Python语言实现**单表仿射**（Affine）加/解密算法；

**2）**用Python语言实现统计26个英文字母出现频率的程序；

**3）**利用**单表仿射**加/解密程序对一段较长的英文文章进行加密，再对明文和密文中字母出现的频率进行统计并作对比，观察有什么规律。

仿射变换：

**加密**：

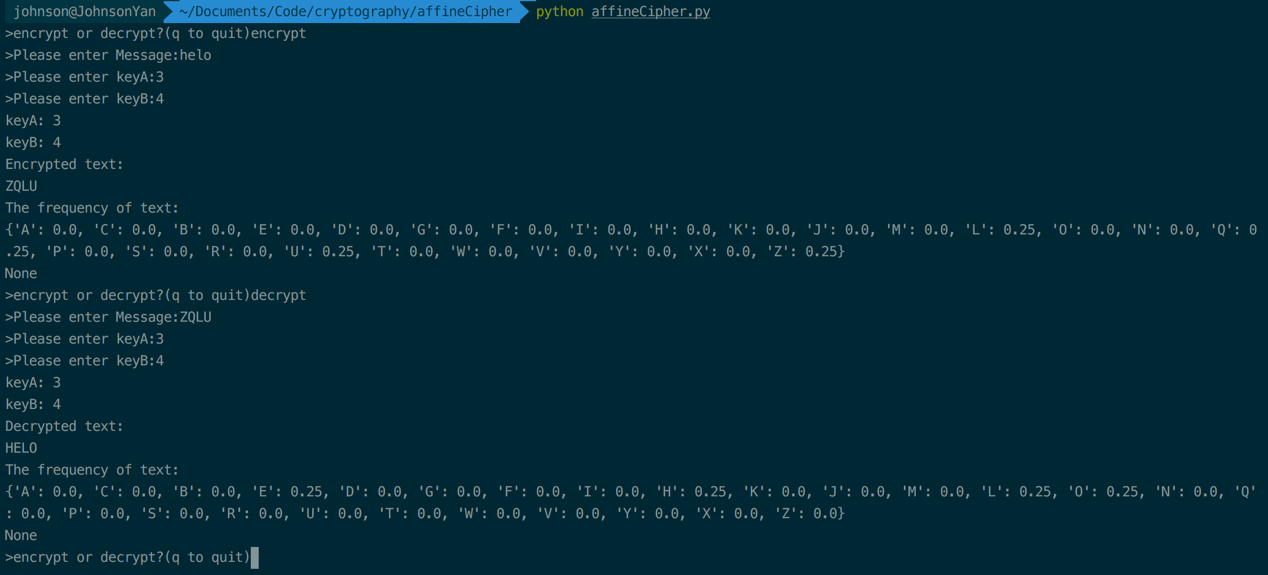
**解密**：

其中，*k*1和*k*2为密钥，*k*1∈*Zq*，*k*2∈*Zq*\*。

**实验要求：**加/解密程序对任意满足条件的*k*1和*k*2都能够处理。

**三、实验结果及分析**

成功运行



**四、实验过程**

“affineCipher.py”

import sys

import random

import cryptomath

import freqAnalysis

SYMBOLS = """ABCDEFGHIJKLMNOPQRSTUVWXYZ"""

def main():

while True:

Mode = raw\_input(">encrypt or decrypt?(q to quit)")

if Mode == 'q':

sys.exit('GoodBye')

Message = raw\_input(">Please enter Message:")

Message = Message.upper()

keyA = int(raw\_input(">Please enter keyA:"))

keyB = int(raw\_input(">Please enter keyB:"))

if Mode == 'encrypt':

translated = encryptMessage(keyA,keyB,Message)

elif Mode == 'decrypt':

translated = decryptMessage(keyA,keyB,Message)

print "keyA: %s" % keyA

print "keyB: %s" % keyB

print "%sed text:" % Mode.title()

print translated

print "The frequency of text:"

print freqAnalysis.getFrequency(translated)

def checkKeys(keyA,keyB,mode):

if keyA < 0 or keyB < 0 or keyB > 25:

sys.exit('illegal key!')

if cryptomath.gcd(keyA,26) != 1:

sys.exit('gcd(keyA,26)!=1')

def encryptMessage(keyA,keyB,Message):

checkKeys(keyA,keyB,'encrypt')

ciphertext = ''

for symbol in Message:

if symbol in SYMBOLS:

symIndex = SYMBOLS.find(symbol)

ciphertext += SYMBOLS[(symIndex \* keyA + keyB) % 26]

else:

ciphertext += symbol

return ciphertext

def decryptMessage(keyA,keyB,Message):

checkKeys(keyA,keyB,'decrypt')

plaintext = ''

modInverse = cryptomath.findModInverse(keyA,26)

for symbol in Message:

if symbol in SYMBOLS:

symIndex = SYMBOLS.find(symbol)

plaintext += SYMBOLS[(symIndex - keyB) \* modInverse % 26]

else:

plaintext += symbol

return plaintext

main()

“cryptomath.py”

def gcd(a,b):

while a != 0:

a,b = b % a,a

return b

def findModInverse(a,m):

if gcd(a,m) != 1:

return None

u1,u2,u3 = 1,0,a

v1,v2,v3 = 0,1,m

while v3 != 0:

q = u3 // v3

v1,v2,v3,u1,u2,u3 = (u1 - q \* v1),(u2 - q \* v2),(u3 - q \* v3),v1,v2,v3

return u1 % m

“freqAnalysis.py”

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def getLetterCount(message):

letterCount = {'A':0,'B':0,'C':0,'D':0,'E':0,'F':0,'F':0,'G':0,

'H':0,'I':0,'J':0,'K':0,'L':0,'M':0,'N':0,'O':0,

'P':0,'Q':0,'R':0,'S':0,'T':0,'U':0,'V':0,'W':0,

'X':0,'Y':0,'Z':0,}

for letter in message.upper():

if letter in LETTERS:

letterCount[letter] += 1

return letterCount

def getFrequency(message):

letterCount = getLetterCount(message)

total = float(sum(v for v in letterCount.values()))

for i in letterCount:

letterCount[i] = round(letterCount[i] / total,4)

print letterCount

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目**  **名 称** | **实验二、序列密码** | **成绩** |  |

**一、实验目的**

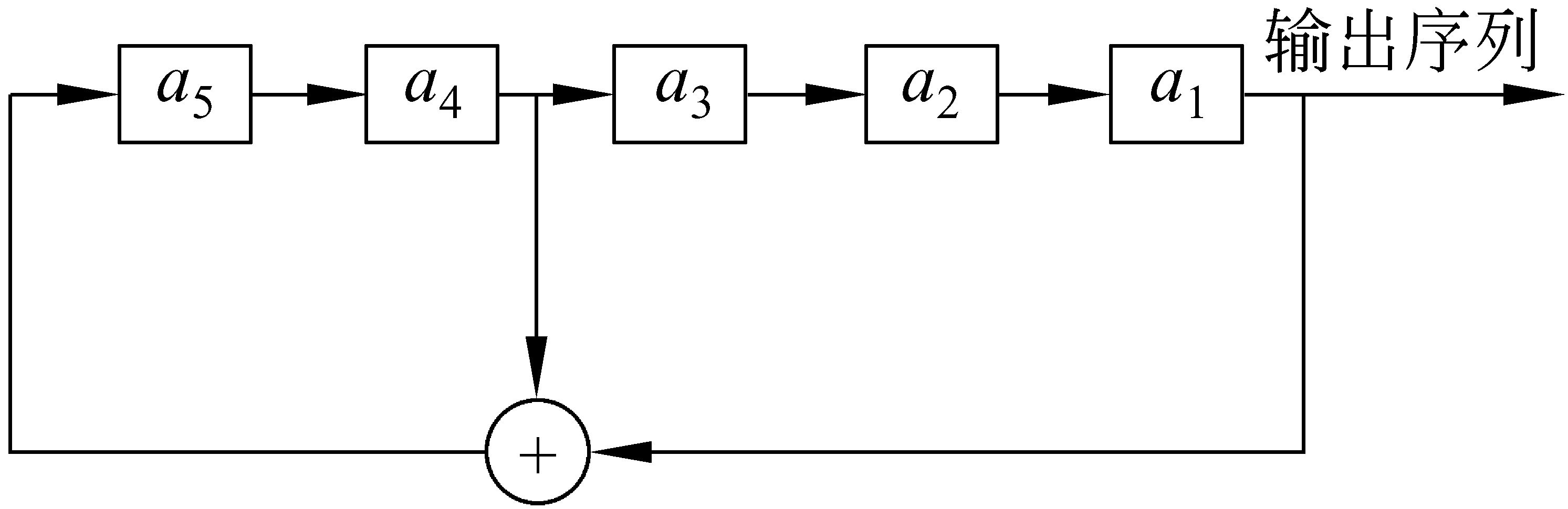
通过实现简单的线性反馈移位寄存器（LFSR），理解LFSR的工作原理、本原多项式的重要意义。

**二、实验内容**

**1）**利用Python语言实现给定的LFSR；

**2）**通过不同初始状态生成相应的序列，并观察它们的周期有什么特点；

**3）**利用生成的序列对文本进行加/解密（按对应位作模2加运算）。

给定的LFSR为：

**三、实验结果及分析**

**成功运行**



**四、实验过程**

“streamCipher.py”

#coding=utf-8

def generateCipherKey(init,length):

cipherKey = ""

for i in range(length\*8):

cipherKey += init[0] # a1移出

init = init[1:5] + str(int(init[0])^int(init[3])) # a1 异或 a4,结果移到最后，为a5,利用切片可以方便的进行快速移位

print "密钥流为：%s" % cipherKey

return cipherKey

if \_\_name\_\_ == "\_\_main\_\_":

init = raw\_input('请输入初始序列:')

clearText = raw\_input('请输入明文:')

length = len(clearText) # 明文长度

cipherKey = generateCipherKey(init,length) # 生成密钥流

cipherText = "" # 密文

revertText = "" # 解密后的明文

for i in range(length):

clearChar = ord(clearText[i])

key = int(cipherKey[i\*8:i\*8+7],2)

cipherText += chr(key^clearChar)

revertText += chr(key^ord(cipherText[i]))

print "加密后的密文为：%s" % cipherText

print "根据密文解密后的明文为：%s" % revertText

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目**  **名 称** | **实验三、DES算法的实现** | **成绩** |  |

**一、实验目的**

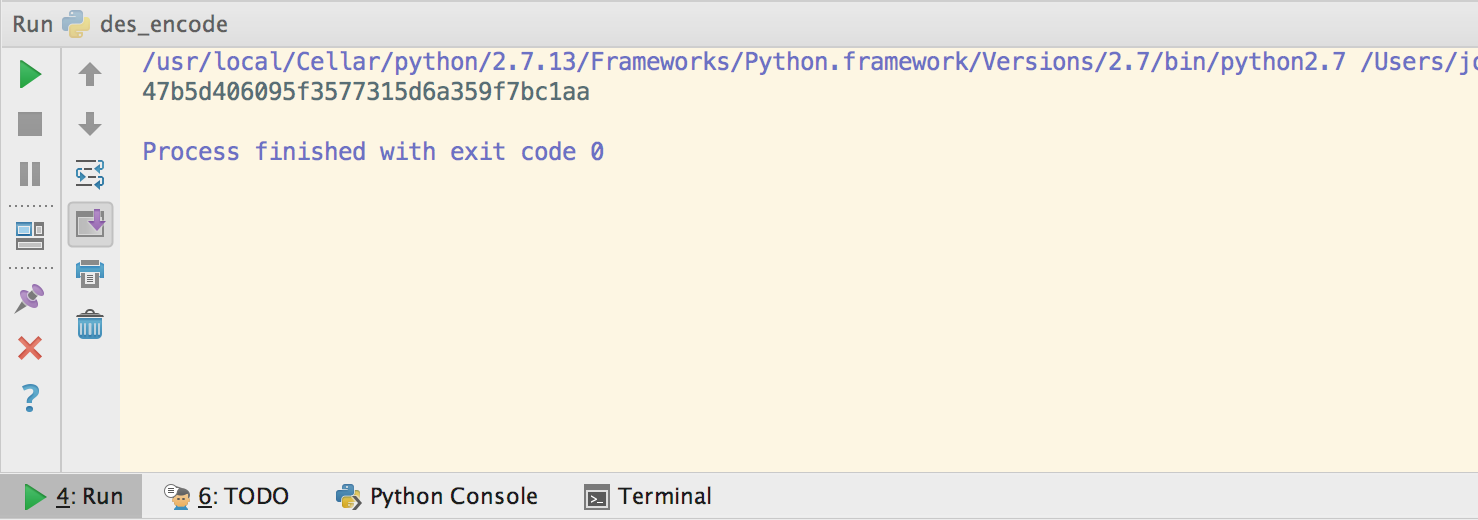
通过实现DES/AES算法，加深对DES/AES算法的理解，同时学习组合密码常用的代换、移位等运算的实现。

**二、实验内容**

**1）**利用Python实现DES/AES算法的加、解密运算。

**三、实验结果及分析**

DES加密成功，但是解密得到的结果为乱码，未调试成功。





**四、实验过程**

“destruct.py”

*#coding=utf-8  
# 初始置换 IP*ip = (58, 50, 42, 34, 26, 18, 10, 2,  
 60, 52, 44, 36, 28, 20, 12, 4,  
 62, 54, 46, 38, 30, 22, 14, 6,  
 64, 56, 48, 40, 32, 24, 16, 8,  
 57, 49, 41, 33, 25, 17, 9, 1,  
 59, 51, 43, 35, 27, 19, 11, 3,  
 61, 53, 45, 37, 29, 21, 13, 5,  
 63, 55, 47, 39, 31, 23, 15, 7)  
*# 逆初始置换 IP\_1*ip\_1 = (40, 8, 48, 16, 56, 24, 64, 32,  
 39, 7, 47, 15, 55, 23, 63, 31,  
 38, 6, 46, 14, 54, 22, 62, 30,  
 37, 5, 45, 13, 53, 21, 61, 29,  
 36, 4, 44, 12, 52, 20, 60, 28,  
 35, 3, 43, 11, 51, 19, 59, 27,  
 34, 2, 42, 10, 50, 18, 58, 26,  
 33, 1, 41, 9, 49, 17, 57, 25)  
*# 位选择函数 E*e = (32, 1, 2, 3, 4, 5,  
 4 , 5, 6, 7, 8, 9,  
 8 , 9, 10, 11, 12, 13,  
 12, 13, 14, 15, 16, 17,  
 16, 17, 18, 19, 20, 21,  
 20, 21, 22, 23, 24, 25,  
 24, 25, 26, 27, 28, 29,  
 28, 29, 30, 31, 32, 1)  
*# 置换函数 P*p=(16, 7, 20, 21,  
 29, 12, 28, 17,  
 1 , 15, 23, 26,  
 5 , 18, 31, 10,  
 2 , 8, 24, 14,  
 32, 27, 3, 9,  
 19, 13, 30, 6,  
 22, 11, 4, 25)  
*# S 盒*s=[[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],  
 [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],  
 [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],  
 [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],  
 [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],  
 [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],  
 [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],  
 [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],  
 [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],  
 [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],  
 [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],  
 [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]],  
 [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],  
 [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14,9],  
 [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],  
 [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]],  
 [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],  
 [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],  
 [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],  
 [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]],  
 [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],  
 [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],  
 [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],  
 [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]],  
 [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],  
 [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],  
 [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],  
 [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]],  
 [[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],  
 [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],  
 [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],  
 [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]]  
*# 选择置换 PC-1*pc1 = (57, 49, 41, 33, 25, 17, 9,  
 1, 58, 50, 42, 34, 26, 18,  
 10, 2, 59, 51, 43, 35, 27,  
 19, 11, 3, 60, 52, 44, 36,  
 63, 55, 47, 39, 31, 33, 15,  
 7, 62, 54, 46, 38, 30, 22,  
 14, 6, 61, 53, 45, 37, 29,  
 21, 13, 5, 28, 20, 12, 4)  
*# 选择置换 PC-2*pc2 = (14, 17, 11, 24, 1, 5, 3, 28,  
 15, 6, 21, 10, 23, 19, 12, 4,  
 26, 8, 16, 7, 27, 20, 13, 2,  
 41, 52, 31, 37, 47, 55, 30, 40,  
 51, 45, 33, 48, 44, 49, 39, 56,  
 34, 53, 46, 42, 50, 36, 29, 32)  
d = ( 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)

“des\_encode.py”

*#coding=utf-8  
#!/usr/bin/env python***from** desstruct **import** \*  
**import** re  
\_\_all\_\_ = ['desencode']  
  
**class** DES():  
 *'''des 加密'''* **def** \_\_init\_\_(self):  
 **pass** *# 加密* **def** code(self,from\_code,key,code\_len,key\_len):  
 output = ""  
 trun\_len = 0  
 *# 将密文和密钥转换为二进制* code\_string = self.\_functionCharToA(from\_code,code\_len)  
 code\_key = self.\_functionCharToA(key,key\_len)  
 *# 如果密钥长度不是16的整数倍，则以补0的方式变成16的整数倍* **if** code\_len % 16 != 0:  
 real\_len = (code\_len/16)\*16+16  
 **else**:  
 real\_len = code\_len  
 **if** key\_len % 16 != 0:  
 key\_len = (key\_len/16)\*16 + 16  
 key\_len \*= 4  
 *# 每个16进制占4位* trun\_len = 4 \* real\_len  
 *# 对每64位进行一次加密* **for** i **in** range(0,trun\_len,64):  
 run\_code = code\_string[i:i+64]  
 l = i % key\_len  
 run\_key = code\_key[l:l+64]  
 *# 64位明文、密文初始置换* run\_code = self.\_codefirstchange(run\_code)  
 run\_key = self.\_codefirstchange(run\_key)  
 *# 16次迭代* **for** j **in** range(16):  
 *# 取出明文左右32位* code\_r = run\_code[32:64]  
 code\_l = run\_code[0:32]  
 *# 左右交换* run\_code = code\_r  
 *# 右边32位扩展置换* code\_r = self.\_functionE(code\_r)  
 *# 获取本轮子密钥* key\_l = run\_key[0:28]  
 key\_r = run\_key[28:56]  
 key\_l = key\_l[d[j]:28]+key\_l[0:d[j]]  
 key\_r = key\_r[d[j]:28]+key\_r[0:d[j]]  
 run\_key = key\_l + key\_r  
 key\_y = self.\_functionKeySecondChange(run\_key)  
 *# 异或* code\_r = self.\_codeyihuo(code\_r,key\_y)  
 *# S盒代替／选择* code\_r = self.\_functionS(code\_r)  
 *# P转换* code\_r = self.\_functionP(code\_r)  
 *# 异或* code\_r = self.\_codeyihuo(code\_l,code\_r)  
 run\_code += code\_r  
  
 *#32互换* code\_r = run\_code[32:64]  
 code\_l = run\_code[0:32]  
 run\_code = code\_r + code\_l  
 *# 将二进制转换为16进制，逆初始置换* output += self.\_functionCodeChange(run\_code)  
  
 **return** output  
  
 *# 异或* **def** \_codeyihuo(self,code,key):  
 code\_len = len(key)  
 return\_list = ''  
 **for** i **in** range(code\_len):  
 **if** code[i] == key[i]:  
 return\_list += '0'  
 **else**:  
 return\_list += '1'  
  
 **return** return\_list  
  
 *# 密文或明文初始置换* **def** \_codefirstchange(self,code):  
 changed\_code = ''  
 **for** i **in** range(64):  
 changed\_code += code[ip[i]-1]  
 **return** changed\_code  
  
 *# 密钥初始置换* **def** \_keyfirstchange(self,key):  
 changed\_key = ''  
 **for** i **in** range(56):  
 changed\_key += key[pc1[i]-1]  
  
 **return** changed\_key  
  
 *# 逆初始置换* **def** \_functionCodeChange(self,code):  
 lens = len(code)/4  
 return\_list = ''  
 **for** i **in** range(lens):  
 list = ''  
 **for** j **in** range(4):  
 list += code[ip\_1[i\*4+j]-1]  
 return\_list += "%x" % int(list,2)  
  
 **return** return\_list  
  
 *# 扩展置换* **def** \_functionE(self,code):  
 return\_list = ''  
 **for** i **in** range(48):  
 return\_list += code[e[i]-1]  
  
 **return** return\_list  
  
 *# 置换 P* **def** \_functionP(self,code):  
 return\_list = ''  
 **for** i **in** range(32):  
 return\_list += code[p[i]-1]  
  
 **return** return\_list  
  
 *# S盒代替选择置换* **def** \_functionS(self,key):  
 return\_list = ''  
 **for** i **in** range(8):  
 row = int(str(key[i\*6])+str(key[i\*6+5]),2)  
 raw = int(str(key[i\*6+1])+str(key[i\*6+2]+str(key[i\*6+3])+str(key[i\*6+4])),2)  
 return\_list += self.\_functionTos(s[i][row][raw],4)  
  
 **return** return\_list  
  
 *# 密钥置换选择2* **def** \_functionKeySecondChange(self,key):  
 return\_list = ''  
 **for** i **in** range(48):  
 return\_list += key[pc2[i]-1]  
  
 **return** return\_list  
  
 *# 将十六进制转换为二进制字符串* **def** \_functionCharToA(self,code,lens):  
 return\_code = ''  
 lens = lens%16  
 **for** key **in** code:  
 code\_ord = int(key,16)  
 return\_code += self.\_functionTos(code\_ord,4)  
 **if** lens != 0:  
 return\_code += '0'\*(16-lens)\*4  
  
 **return** return\_code  
  
 *# 二进制转换* **def** \_functionTos(self,o,lens):  
 return\_code = ''  
 **for** i **in** range(lens):  
 return\_code = str(o>>i & 1)+return\_code  
  
 **return** return\_code  
  
*# 将unicode字符转换为16进制***def** tohex(string):  
 return\_string = ''  
 **for** i **in** string:  
 return\_string += "%02x" % ord(i)  
  
 **return** return\_string  
  
**def** tounicode(string):  
 return\_string = ''  
 string\_len = len(string)  
 **for** i **in** range(0,string\_len,2):  
 return\_string += chr(int(string[i:i+2],16))  
  
 **return** return\_string  
  
*# 入口函数***def** desencode(from\_code,key):  
 *# 转换位16进制* from\_code = tohex(from\_code)  
 key = tohex(key)  
 des = DES()  
 key\_len = len(key)  
 string\_len = len(from\_code)  
 **if** string\_len < 1 **or** key\_len < 1:  
 **print** 'error input'  
 **return** False  
 key\_code = des.code(from\_code,key,string\_len,key\_len)  
 **return** key\_code  
  
*# test***if** \_\_name\_\_ == '\_\_main\_\_':  
 **print** desencode(**u'HelloWorld'**,**u'yanjun'**)

“des\_decode.py”

*#coding=utf-8  
#!/usr/bin/env python***from** desstruct **import** \*  
**import** re  
\_\_all\_\_ = ['desencode']  
  
**class** DES():  
 *'''des 解密，与加密相差不大，只是在  
 解密的时候所用子密钥与加密的子密钥相反'''* **def** \_\_init\_\_(self):  
 **pass  
  
 def** decode(self, string, key, key\_len, string\_len):  
 output = ""  
 trun\_len = 0  
 num = 0  
 *# 将密文转换为二进制* code\_string = self.\_functionCharToA(string, string\_len)  
 *# 获取子密钥* code\_key = self.\_getkey(key, key\_len)  
 *# 如果密钥长度不是16的整数倍，则以补0的方式变为16的整数倍* real\_len = (key\_len / 16) + 1 **if** key\_len % 16 != 0 **else** key\_len / 16  
 trun\_len = string\_len \* 4  
 *# 对每64位进行一次加密* **for** i **in** range(0, trun\_len, 64):  
 run\_code = code\_string[i:i + 64]  
 run\_key = code\_key[num % real\_len]  
 *# 64位明文初始置换* run\_code = self.\_codefirstchange(run\_code)  
 *# 16次迭代* **for** j **in** range(16):  
 code\_r = run\_code[32:64]  
 code\_l = run\_code[0:32]  
 *# 64左右交换* run\_code = code\_r  
 *# 右边32位扩展置换* code\_r = self.\_functionE(code\_r)  
 *# 获取本轮子密钥* key\_y = run\_key[15 - j]  
 *# 异或* code\_r = self.\_codeyihuo(code\_r, key\_y)  
 *# S盒代替/选择* code\_r = self.\_functionS(code\_r)  
 *# P转换* code\_r = self.\_functionP(code\_r)  
 *# 异或* code\_r = self.\_codeyihuo(code\_l, code\_r)  
 run\_code += code\_r  
 num += 1  
 *# 32互换* code\_r = run\_code[32:64]  
 code\_l = run\_code[0:32]  
 run\_code = code\_r + code\_l  
 *# 将二进制转换为16进制、逆初始置换* output += self.\_functionCodeChange(run\_code)  
 **return** output  
  
 *# 获取子密钥* **def** \_getkey(self, key, key\_len):  
 *# 将密钥转换为二进制* code\_key = self.\_functionCharToA(key, key\_len)  
 a = [''] \* 16  
 real\_len = (key\_len / 16) \* 16 + 16 **if** key\_len % 16 != 0 **else** key\_len  
 b = [''] \* (real\_len / 16)  
 **for** i **in** range(real\_len / 16):  
 b[i] = a[:]  
 num = 0  
 trun\_len = 4 \* key\_len  
 **for** i **in** range(0, trun\_len, 64):  
 run\_key = code\_key[i:i + 64]  
 run\_key = self.\_keyfirstchange(run\_key)  
 **for** j **in** range(16):  
 key\_l = run\_key[0:28]  
 key\_r = run\_key[28:56]  
 key\_l = key\_l[d[j]:28] + key\_l[0:d[j]]  
 key\_r = key\_r[d[j]:28] + key\_r[0:d[j]]  
 run\_key = key\_l + key\_r  
 key\_y = self.\_functionKeySecondChange(run\_key)  
 b[num][j] = key\_y[:]  
 num += 1  
 **return** b  
  
 *# 异或* **def** \_codeyihuo(self, code, key):  
 code\_len = len(key)  
 return\_list = ''  
 **for** i **in** range(code\_len):  
 **if** code[i] == key[i]:  
 return\_list += '0'  
 **else**:  
 return\_list += '1'  
 **return** return\_list  
  
 *# 密文或明文初始置换* **def** \_codefirstchange(self, code):  
 changed\_code = ''  
 **for** i **in** range(64):  
 changed\_code += code[ip[i] - 1]  
 **return** changed\_code  
  
 *# 密钥初始置换* **def** \_keyfirstchange(self, key):  
 changed\_key = ''  
 **for** i **in** range(56):  
 changed\_key += key[pc1[i] - 1]  
 **return** changed\_key  
  
 *# 逆初始置换* **def** \_functionCodeChange(self, code):  
 return\_list = ''  
 **for** i **in** range(16):  
 list = ''  
 **for** j **in** range(4):  
 list += code[ip\_1[i \* 4 + j] - 1]  
 return\_list += "%x" % int(list, 2)  
 **return** return\_list  
  
 *# 扩展置换* **def** \_functionE(self, code):  
 return\_list = ''  
 **for** i **in** range(48):  
 return\_list += code[e[i] - 1]  
 **return** return\_list  
  
 *# 置换P* **def** \_functionP(self, code):  
 return\_list = ''  
 **for** i **in** range(32):  
 return\_list += code[p[i] - 1]  
 **return** return\_list  
  
 *# S盒代替选择置换* **def** \_functionS(self, key):  
 return\_list = ''  
 **for** i **in** range(8):  
 row = int(str(key[i \* 6]) + str(key[i \* 6 + 5]), 2)  
 raw = int(str(key[i \* 6 + 1]) + str(key[i \* 6 + 2]) + str(key[i \* 6 + 3]) + str(key[i \* 6 + 4]), 2)  
 return\_list += self.\_functionTos(s[i][row][raw], 4)  
 **return** return\_list  
  
 *# 密钥置换选择2* **def** \_functionKeySecondChange(self, key):  
 return\_list = ''  
 **for** i **in** range(48):  
 return\_list += key[pc2[i] - 1]  
 **return** return\_list  
  
 *# 将十六进制转换为二进制字符串* **def** \_functionCharToA(self, code, lens):  
 return\_code = ''  
 lens = lens % 16  
 **for** key **in** code:  
 code\_ord = int(key, 16)  
 return\_code += self.\_functionTos(code\_ord, 4)  
 **if** lens != 0:  
 return\_code += '0' \* (16 - lens) \* 4  
 **return** return\_code  
 *# 二进制转换* **def** \_functionTos(self, o, lens):  
 return\_code = ''  
 **for** i **in** range(lens):  
 return\_code = str(o >> i & 1) + return\_code  
 **return** return\_code  
  
*# 将unicode字符转换为16进制***def** tohex(string):  
 return\_string = ''  
 **for** i **in** string:  
 return\_string += "%x" % ord(i)  
  
 **return** return\_string  
  
**def** tounicode(string):  
 return\_string = ''  
 string\_len = len(string)  
 **for** i **in** range(0, string\_len, 2):  
 return\_string += chr(int(string[i:i + 2],16))  
 **return** return\_string  
  
*# 入口函数***def** desdecode(from\_code, key):  
 key = tohex(key)  
 des = DES()  
 key\_len = len(key)  
 string\_len = len(from\_code)  
 **if** string\_len % 16 != 0:  
 **return** False  
 **if** string\_len < 1 **or** key\_len < 1:  
 **return** False  
 key\_code = des.decode(from\_code, key, key\_len, string\_len)  
 *#key\_code.decode('utf-8')* **return** tounicode(key\_code)  
 *#return key\_code  
  
# test***if** \_\_name\_\_ == '\_\_main\_\_':  
 text = '47b5d406095f3577315d6a359f7bc1aa'  
 key = 'yanjun'  
 text = text.encode('utf-8')  
 key = key.encode('utf-8')  
 **print** desdecode(text,key)

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目**  **名 称** | **实验四、RSA算法的实现** | **成绩** |  |

**一、实验目的**

掌握并实现RSA算法。

**二、实验内容**

利用Python实现RSA算法的加、解密运算。

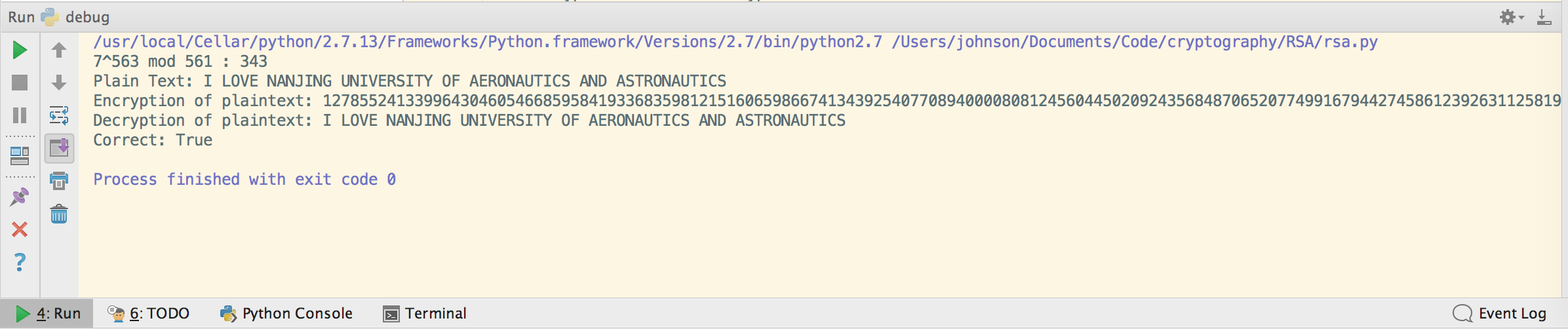
具体包括：

1. 利用扩展的Euclid计算 *a* mod *n* 的乘法逆元；
2. Miller-Rabin素性测试算法对一个给定的大数进行测试；
3. 实现的运算，并计算；
4. 利用Euler定理手工计算，并与**3）**计算的结果对比；
5. 实现RSA算法。并对"I LOVE NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS"加解密。说明：为了方便实现，分组可以小一点，比如两个字母一组。

|  |  |
| --- | --- |
| 字母及其数字编码 | 字母及其数字编码 |
| 空格 00 | N 14 |
| A 01 | O 15 |
| B 02 | P 16 |
| C 03 | Q 17 |
| D 04 | R 18 |
| E 05 | S 19 |
| F 06 | T 20 |
| G 07 | U 21 |
| H 08 | V 22 |
| I 09 | W 23 |
| J 10 | X 24 |
| K 11 | Y 25 |
| L 12 | Z 26 |
| M 13 |  |

**三、实验结果及分析**

运行成功



**四、实验过程**

“rsa.py”

**import** random  
  
**def** fastExpMod(b,e,m):  
 *# b^e mod m* result = 1  
 **while** e != 0:  
 **if** (e&1) == 1:  
 *# ei = 1, then mul* result = (result \* b) % m  
 *# e = e/2* e >>= 1  
 *# b,b^2,b^4,b^8,...,b^(2^n)* b = (b\*b) % m  
 **return** result  
  
**def** primeTest(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 a prime number* **if** fastExpMod(a,q,n) == 1:  
 **return** "Uncertain"  
 *# if there exits j satify a ^ ((2^j)\*q) mod == n-1,n maybe a prime number* **for** j **in** range(0,k):  
 **if** fastExpMod(a,(2\*\*j)\*q,n) == n - 1:  
 **return** "Uncertain"  
 *# a is not a prime number* **return** "Fail"  
  
**def** findPrime(halfkeyLength):  
 **while** True:  
 n = random.randint(0,1<<halfkeyLength)  
 **if** n % 2 != 0:  
 found = True  
 **for** i **in** range(0,10):  
 **if** primeTest(n) == "Fail":  
 found = False  
 **break  
 if** found:  
 **return** n  
  
**def** extendedGCD(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** selectE(fn,halfkeyLength):  
 **while** True:  
 e = random.randint(0,1<<halfkeyLength)  
 (x,y,r) = extendedGCD(e,fn)  
 **if** r == 1:  
 **return** e  
  
**def** computeD(fn,e):  
 (x,y,r) = extendedGCD(fn,e)  
 **if** y < 0:  
 **return** fn + y  
 **return** y  
  
**def** keyGeneration(keyLength):  
 p = findPrime(keyLength/2)  
 q = findPrime(keyLength/2)  
 n = p \* q  
 fn = (p-1)\*(q-1)  
 e = selectE(fn,keyLength/2)  
 d = computeD(fn,e)  
 **return** (n,e,d)  
  
**def** encryption(M,e,n):  
 *# RSA C = M^e mod n* **return** fastExpMod(M,e,n)  
  
**def** decryption(C,d,n):  
 *# RSA M = C^d mod n* **return** fastExpMod(C,d,n)  
  
dictionary = {' ':'00','A':'01','B':'02','C':'03','D':'04','E':'05','F':'06','G':'07',  
 'H':'08','I':'09','J':'10','K':'11','L':'12','M':'13','N':'14','O':'15',  
 'P':'16','Q':'17','R':'18','S':'19','T':'20','U':'21','V':'22','W':'23',  
 'X':'24','Y':'25','Z':'26'}  
dictionary\_1 = {'00':' ','01':'A','02':'B','03':'C','04':'D','05':'E','06':'F','07':'G',  
 '08':'H','09':'I','10':'J','11':'K','12':'L','13':'M','14':'N','15':'O',  
 '16':'P','17':'Q','18':'R','19':'S','20':'T','21':'U','22':'V','23':'W',  
 '24':'X','25':'Y','26':'Z'}  
Text = 'I LOVE NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS'  
X = ''  
**for** i **in** Text:  
 X += dictionary[i]  
X = int(X,10)  
*# Test*(n,e,d) = keyGeneration(1024)  
*# 7^563 mod 561***print** "7^563 mod 561 :",fastExpMod(7,563,561)  
C = encryption(X,e,n)  
M = decryption(C,d,n)  
  
*# int to str*decryptedText = str(M)  
**if** len(decryptedText) % 2 != 0:  
 decryptedText = '0'+decryptedText  
result = ''  
**for** i **in** range(0,len(decryptedText)/2):  
 temp = decryptedText[i\*2:i\*2+2]  
 result += dictionary\_1[temp]  
  
**print** "Plain Text:",Text  
**print** "Encryption of plaintext:",C  
**print** "Decryption of plaintext:",result  
**print** "Correct:",Text == result

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目**  **名 称** | **实验五、数字签名算法** | **成绩** |  |

**一、实验目的**

通过实现数字签名算法（DSA），加深对数字签名算法的理解，同时学习Hash算法的实现。

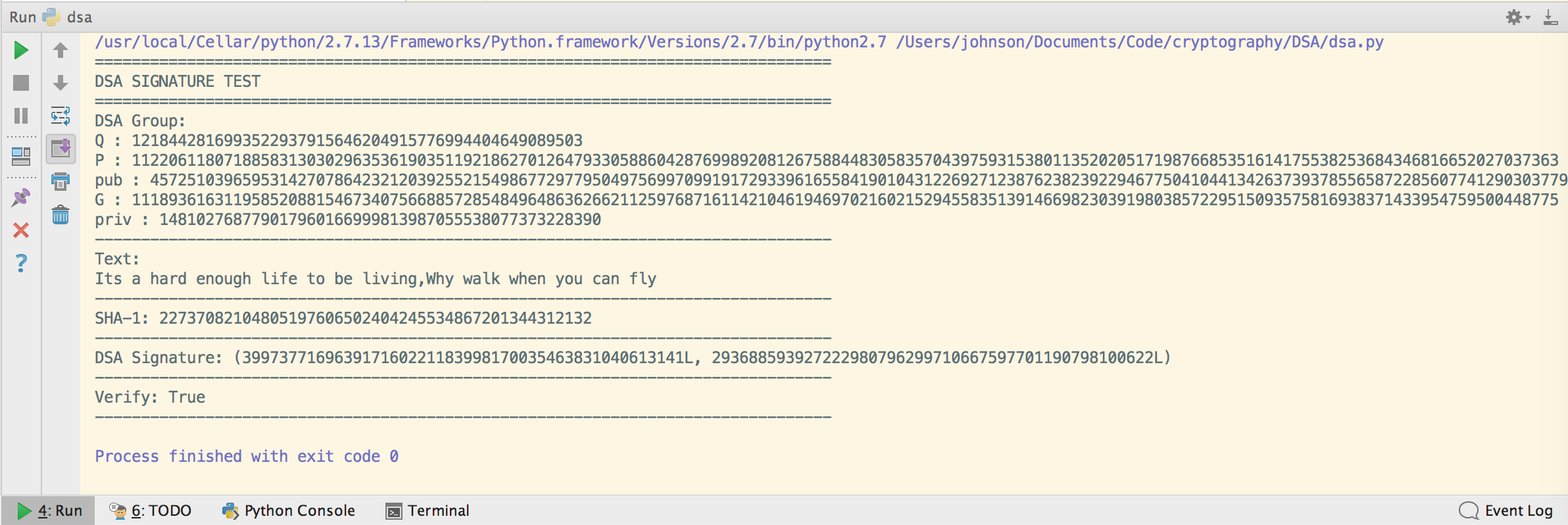
**二、实验内容**

**1）**利用Python语言实现DSA算法。

**2）**DSA中的Hash函数采用SHA算法。

**三、实验结果及分析**

运行成功，但是SHA-1算法未成功实现，调用了hashlib库。

****

**四、实验过程**

“dsa.py”

*#coding=utf-8***import** os  
**import** random  
**import** hashlib  
  
*# test dsa example*dsa\_key = {  
 'Q': 1218442816993522937915646204915776994404649089503L,  
 'P': 11220611807188583130302963536190351192186270126479330588604287699892081267588448305835704397593153801135202051719876685351614175538253684346816652027037363L,  
 'G': 11189361631195852088154673407566885728548496486362662112597687161142104619469702160215294558351391466982303919803857229515093575816938371433954759500448775L,  
 'pub': 4572510396595314270786423212039255215498677297795049756997099191729339616558419010431226927123876238239229467750410441342637393785565872285607741290303779L,  
 'priv': 148102768779017960166999813987055538077373228390L}  
text = """Its a hard enough life to be living,Why walk when you can fly"""  
  
*# 随机选择一个数***def** \_random\_s(min, max):  
 s = -1  
 digits = random.randint(len(str(min)), len(str(max)))  
 **while** True:  
 u = map(ord, os.urandom(digits))  
 **if** u == None:  
 **continue** s = int(''.join(str(x) **for** x **in** u)[:digits])  
 **if** s <= max **and** s >= min:  
 **break  
 return** s  
  
*# 计算模逆***def** \_mod\_inverse(a, b):  
 r = -1  
 B = b  
 A = a  
 eq\_set = []  
 full\_set = []  
 mod\_set = []  
  
 *# euclid 算法* **while** r != 1 **and** r != 0:  
 r = b % a  
 q = b // a  
 eq\_set = [r, b, a, q \* -1]  
 b = a  
 a = r  
 full\_set.append(eq\_set)  
  
 **for** i **in** range(0, 4):  
 mod\_set.append(full\_set[-1][i])  
  
 mod\_set.insert(2, 1)  
 counter = 0  
  
 *#extended euclid 算法* **for** i **in** range(1, len(full\_set)):  
 **if** counter % 2 == 0:  
 mod\_set[2] = full\_set[-1 \* (i + 1)][3] \* mod\_set[4] + mod\_set[2]  
 mod\_set[3] = full\_set[-1 \* (i + 1)][1]  
  
 **elif** counter % 2 != 0:  
 mod\_set[4] = full\_set[-1 \* (i + 1)][3] \* mod\_set[2] + mod\_set[4]  
 mod\_set[1] = full\_set[-1 \* (i + 1)][1]  
  
 counter += 1  
  
 **if** mod\_set[3] == B:  
 **return** mod\_set[2] % B  
 **return** mod\_set[4] % B  
  
*# a \*\* b (mod n)***def** modexp\_lr\_k\_ary(a, b, n, k=5):  
 base = 2 << (k - 1)  
 table = [1] \* base  
 **for** i **in** xrange(1, base):  
 table[i] = table[i - 1] \* a % n  
  
 r = 1  
 **for** digit **in** reversed(\_digits\_of\_n(b, base)):  
 **for** i **in** xrange(k):  
 r = r \* r % n  
 **if** digit:  
 r = r \* table[digit] % n  
 **return** r  
  
  
**def** \_digits\_of\_n(n, b):  
 digits = []  
 **while** n:  
 digits.append(int(n % b))  
 n /= b  
 **return** digits  
  
**def** dsa\_sign(q, p, g, x, message):  
  
 s = \_random\_s(1, q)  
 s1 = 0  
 s2 = 0  
 **while** True:  
 modexp = modexp\_lr\_k\_ary(g, s, p)  
 s1 = modexp % q  
 **if** s1 == 0:  
 s = \_random\_s(1, q)  
 **continue** s = \_mod\_inverse(s, q) \* (message + x \* s1)  
 s2 = s % q  
 **if** s2 == 0:  
 s = \_random\_s(1, q)  
 **continue  
 return** (int(s1), int(s2))  
  
  
**def** dsa\_verify(s1, s2, g, p, q, y, message):  
  
 **if not** s1 > 0:  
 **return** False  
 **if not** s1 < q:  
 **return** False  
 **if not** s2 > 0:  
 **return** False  
 **if not** s2 < q:  
 **return** False  
 w = \_mod\_inverse(s2, q)  
 u1 = (message \* w) % q  
 u2 = (s1 \* w) % q  
  
 u1 = pow(g, u1, p)  
 u2 = pow(y, u2, p)  
 v = u1 \* u2 % p % q  
 **if** v == s1:  
 **return** True  
 **return** False  
  
  
  
**if** \_\_name\_\_ == "\_\_main\_\_":  
 m = hashlib.sha1()  
 m.update(text)  
 message = int("0x" + m.hexdigest(), 0)  
 sig = dsa\_sign(dsa\_key["Q"], dsa\_key["P"], dsa\_key["G"], dsa\_key["priv"], message)  
 **print** "=" \* 80  
 **print** "DSA SIGNATURE TEST"  
 **print** "=" \* 80  
 **print** "DSA Group:"  
 **for** k **in** dsa\_key.keys():  
 **print** k, ':', str(dsa\_key[k])  
 **print** "-" \* 80  
 **print** "Text:"  
 **print** text  
 **print** "-" \* 80  
 **print** "SHA-1:",  
 **print** message  
 **print** "-" \* 80  
 **print** "DSA Signature:",  
 **print** sig  
 **print** "-" \* 80  
 **print** "Verify:",  
 **print** dsa\_verify(sig[0], sig[1], dsa\_key["G"], dsa\_key["P"], dsa\_key["Q"], dsa\_key["pub"], message)  
 **print** "-" \* 80