VIT AP UNIVERSITY, ANDHRA PRADESH
Introduction to Cryptography
PRACTICAL ASSIGNMENT

Academic year: 2020-2021 Branch/ Class: B.Tech
Course: Introduction to Cryptography (CSE1007) Slot: B
Semester: Fall
Faculty Name: Dr Saroj Kumar Panigrahy School: SCOPE
Student name: Mohd Priyanshu Yakub Reg. no.: 20BCE7305

1. Find the GCD using Euclidian algorithm and multiplicative inverse modulo n using Extended-Euclidian algorithm.

   Programming Language used: Python

   Code:

```
#using simple euclidian algorithm to get GCD of two numbers

def getGCD(n,b):

      r1 =n

      r2 = b

      while(r2>0):

            q = int(r1/r2)

            r = r1-q*r2

            r1 = r2

            r2 = r

      return r1



#using extended euclidian algorithm to get inverse modulo

def getModInverse(n,b):

      r1 =n

      r2 = b

      t1 = 0

      t2 = 1

      while(r2>0):

            q = int(r1/r2)

            r = r1-q*r2
```

```
            r1 = r2
            r2 = r
            #inverse part
            t = t1- q*t2
            t1 = t2
            t2 = t
      #to maintain +ve inverse value and that it is in Zn
      if(t1<0):
            t1 = n +t1
      return t1

def main(x,y):
    n = int(x)
    b = int(y)

    gcd = getGCD(n,b)
    print("Gcd of given numbers is ", gcd)
    if gcd == 1:
        inv = getModInverse(n,b)
        print("inverse of " , b ,"in modulo ", n," is: ", inv)
    else:
        print("Gcd of given numbers is not equal to one so inverse doesn't exist")
    pass



x = input("Input n in modulo n: ")

y = input("input number to get GCD and modulo of: ")

main(x,y)
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\1 question\Python code>python GCD.py

Input n in modulo n: 7

input number to get GCD and modulo of: 3

Gcd of given numbers is  1

inverse of  3 in modulo  7  is:  5
```

2. Design a menu based modular arithmetic calculator [addition, subtraction, multiplication, division, inverse of a number (additive and multiplicative)].

Programming Language used: Python

Code:

```
#Euclidian algorithm to get GCD

def getGCD(b,n):

    r1 =n

    r2 = b

    while(r2>0):

        q = int(r1/r2)

        r = r1-q*r2

        r1 = r2

        r2 = r

    return r1



#usiing extended euclidian algorithm to get inverse modulo

def getModInverse(b,n):

    r1 =n

    r2 = b

    t1 = 0

    t2 = 1

    while(r2>0):

        q = int(r1/r2)

        r = r1-q*r2

        r1 = r2

        r2 = r

        #inverse part

        t = t1- q*t2

        t1 = t2

        t2 = t

    #to maintain +ve inverse value and that it is in Zn

    if(t1<0):

        t1 = n +t1
```

```
    return t1

#getting additive inverse
def getAddInv(a,n):
    return n-(a%n)



#addition in modulo n
def add():
    print("\nAddition: (a+b) modulo n")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    b = int(input("Enter Value of 'b': " ))

    s = (a+b)%n
    print("(",a,"+",b,")modulo",n," = ",s)

#subtraction in modulo n
def diff():
    print("\nSubtraction: (a-b) modulo n")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    b = int(input("Enter Value of 'b': " ))

    d = (a-b)%n
    print("(",a,"-",b,")modulo",n," = ",d)



#multiplication in modulo n
def multi():
    print("\nMultipliacation: (a*b) modulo n")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    b = int(input("Enter Value of 'b': " ))
```

```
    m = (a*b)%n
    print("(",a,"*",b,")modulo",n," = ",m)



#Division in modulo n
def division():
    print("\nDivision: (a/b) modulo n NOTE:only possible if b has multiplicative
inverse in modulo n")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    b = int(input("Enter Value of 'b': " ))

    #check wether b has multiplicative inverse or not if not division is not possible
    #(b * q) % n = a % n. concept we find c as inv(b)*a%n = q
    if (getGCD(b,n) == 1):
        inverse = getModInverse(b,n)
        q = (inverse*a)%n
        print("(",a,"/",b,")modulo",n," = ",q)
    else:
        print("Inverse of",b," in modulo ",n, " Doesn't exist, therefore:\n Division
not Defined")



def aInv():
    print("\nAdditive inverse: a + x ≡ 0 modulo n, we need x")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    print("Additive inverse of ",a,"modulo",n," = ", getAddInv(a,n))

def mInv():
    print("\nMultiplicative inverse: a * x ≡ 1 modulo n, we need x")
    n = int(input("Enter Value of 'n': " ))
    a = int(input("Enter Value of 'a': " ))
    if(getGCD(a,n)==1):
        print("Multiplicative inverse of ",a,"modulo",n," = ", getModInverse(a,n))
    else:
```

```
        print("inverse Doesn't exist as GCD of",a," and ",n,"is not equal to 1" )


#main menu definition

def menu():


    print("\nWhat would you like to do?")

    print("1.Addition \n2.Subtraction \n3.Multiplication \n4.Division \n5.Additive
Inverse \n6.Multiplicative Inverse")

    print("7.Quit")


    #using python dictionary to create switcher case and call respective functions

    case = {

            "1":add,

            "2":diff,

            "3":multi,

            "4":division,

            "5":aInv,

            "6":mInv,

            }

    #input

    option = input("Select your option: ")

    if(option=="7"):

        print("\nThank you for using MODULAR ARITHMETIC CALCULATOR\n By: Priyanhsu
Yakub 20BCE7305")

        return True

    else:

        case[option]()

        return False


#looping the menu till exit is pressed

exit = False;

print("\n-----MODULAR ARITHMETIC CALCULATOR--------By: Priyanhsu Yakub 20BCE7305------
-----")

while (exit==False):

    exit = menu()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\2 question\python>python main.py


-----MODULAR ARITHMETIC CALCULATOR--------By: Priyanhsu Yakub 20BCE7305-----------


What would you like to do?

1.Addition

2.Subtraction

3.Multiplication

4.Division

5.Additive Inverse

6.Multiplicative Inverse

7.Quit

Select your option: 1


Addition: (a+b) modulo n

Enter Value of 'n': 7

Enter Value of 'a': 5

Enter Value of 'b': 6

( 5 + 6 )modulo 7  =  4


What would you like to do?

1.Addition

2.Subtraction

3.Multiplication

4.Division

5.Additive Inverse

6.Multiplicative Inverse

7.Quit

Select your option: 2


Subtraction: (a-b) modulo n

Enter Value of 'n': 7
```

```
Enter Value of 'a': 9
Enter Value of 'b': 12
( 9 - 12 )modulo 7  =  4

What would you like to do?
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Additive Inverse
6.Multiplicative Inverse
7.Quit
Select your option: 3

Multipliacation: (a*b) modulo n
Enter Value of 'n': 5
Enter Value of 'a': 3
Enter Value of 'b': 3
( 3 * 3 )modulo 5  =  4

What would you like to do?
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Additive Inverse
6.Multiplicative Inverse
7.Quit
Select your option: 4

Division: (a/b) modulo n NOTE:only possible if b has multiplicative inverse in modulo n
Enter Value of 'n': 7
Enter Value of 'a': 12
Enter Value of 'b': 23
```

```
( 12 / 23 )modulo 7  =  6

What would you like to do?
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Additive Inverse
6.Multiplicative Inverse
7.Quit
Select your option: 5

Additive inverse: a + x ≡ 0 modulo n, we need x
Enter Value of 'n': 7
Enter Value of 'a': 4
Additive inverse of  4 modulo 7  =  3

What would you like to do?
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Additive Inverse
6.Multiplicative Inverse
7.Quit
Select your option: 6

Multiplicative inverse: a * x ≡ 1 modulo n, we need x
Enter Value of 'n': 11
Enter Value of 'a': 5
Multiplicative inverse of  5 modulo 11  =  9

What would you like to do?
1.Addition
2.Subtraction
```

```
3.Multiplication

4.Division

5.Additive Inverse

6.Multiplicative Inverse

7.Quit

Select your option: 7


Thank you for using MODULAR ARITHMETIC CALCULATOR

 By: Priyanhsu Yakub 20BCE7305
```

3. Implement Caesar cipher and multiplicative substitution cipher and try cryptanalysis.

Programming Language used: Python

Caesar cipher

Code:

```
 #casesar cipher or simply Additive cipher
def Encryption(PlainText, Key):
      #ensuring uniformity of plaintext using lower() function
      PlainText = PlainText.lower()
      PTno = []
      #converting plain text to numbers
      for character in PlainText:
            number = ord(character)-97
            PTno.append(number)
      #checking if given key is valid
      exists = False
      for k in range (1,27):
            if Key == k:
                  exists = True
      if exists == False:
            print("given key is not valid")
            return False
      #creating output
      output = []
      for j in PTno:
            num = (j+Key)%26 +97
            output.append(num)
      #converting from number to letters
      string_out = [chr(o) for o in output]
      out = ''.join(string_out)
      print("for key = ", Key, "Cipher text is : ",out.upper())
      return out.upper()
```

```
def Decrypt(CT,k):
    CT = CT.lower()
    CTno = []
    for character in CT:
        number = ord(character) - 97
        CTno.append(number)
    output = []
    for i in CTno:
        num = (i-k)%26 +97
        output.append(num)
    string_out = [chr(o) for o in output]
    return ''.join(string_out)




def BruteForce(CT):
    print("\nBruteForcing Xaesar Cipher for Cipher Text: ", CT)
    for k in range (1,27):
      P = Decrypt(CT,k)
      print("for Key = ",k,", Plain Text is : ", P, )



PT = input("Input PlainText: ")
#ensuring no spaces in given text
PT = PT.replace(" ","")
k = int(input("Input key: "))

e = Encryption(PT,k)
if e != False:
      #BruteForceCaesar(e)
      BruteForce(e)
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\3 Question>python Caesar.py

Input PlainText: hello

Input key: 5

for key =  5 Cipher text is :  MJQQT


BruteForcing Xaesar Cipher for Cipher Text:  MJQQT

for Key =  1 , Plain Text is :  lipps

for Key =  2 , Plain Text is :  khoor

for Key =  3 , Plain Text is :  jgnnq

for Key =  4 , Plain Text is :  ifmmp

for Key =  5 , Plain Text is :  hello

for Key =  6 , Plain Text is :  gdkkn

for Key =  7 , Plain Text is :  fcjjm

for Key =  8 , Plain Text is :  ebiil

for Key =  9 , Plain Text is :  dahhk

for Key =  10 , Plain Text is :  czggj

for Key =  11 , Plain Text is :  byffi

for Key =  12 , Plain Text is :  axeeh

for Key =  13 , Plain Text is :  zwddg

for Key =  14 , Plain Text is :  yvccf

for Key =  15 , Plain Text is :  xubbe

for Key =  16 , Plain Text is :  wtaad

for Key =  17 , Plain Text is :  vszzc

for Key =  18 , Plain Text is :  uryyb

for Key =  19 , Plain Text is :  tqxxa

for Key =  20 , Plain Text is :  spwwz

for Key =  21 , Plain Text is :  rovvy

for Key =  22 , Plain Text is :  qnuux

for Key =  23 , Plain Text is :  pmttw

for Key =  24 , Plain Text is :  olssv

for Key =  25 , Plain Text is :  nkrru

for Key =  26 , Plain Text is :  mjqqt
```

Multiplicative substitution cipher

Code:

```
#using extended euclidian algorithm to get inverse modulo
def getModInverse(n,b):
	r1 =n
	r2 = b
	t1 = 0
	t2 = 1
	while(r2>0):
		q = int(r1/r2)
		r = r1-q*r2
		r1 = r2
		r2 = r
		#inverse part
		t = t1- q*t2
		t1 = t2
		t2 = t
	#to maintain +ve inverse value and that it is in Zn
	if(t1<0):
		t1 = n +t1
	return t1



#multiplicative encryption using given plain text
def Encryption(PlainText, Key):
	#ensuring uniformity of plaintext using lower() function
	PlainText = PlainText.lower()
	PTno = []
	#converting plain text to numbers
	for character in PlainText:
		number = ord(character)-97
		PTno.append(number)
	#all possible multiplicative keys i.e. Zn*
	keys = [1,3,5,7,9,11,15,17,19,21,23,25]
```

```
        #checking if given key is valid
        exists = False
        for k in keys:
              if Key == k:
                    exists = True
        if exists == False:
              print("given key is not valid")
              return False

        #creating output
        output = []
        for j in PTno:
              num = (j*Key)%26 +97
              output.append(num)
        string_out = [chr(o) for o in output]
        out = ''.join(string_out)
        print("for key = ", Key, "Cipher text is : ",out.upper())
        return out.upper()

def Decrypt(CT,k):
    k_inv = getModInverse(26,k)
    CT = CT.lower()
    CTno = []
    for character in CT:
        number = ord(character) - 97
        CTno.append(number)
    output = []
    for i in CTno:
        num = (i*k_inv)%26 +97
        output.append(num)
    string_out = [chr(o) for o in output]
    return ''.join(string_out)



def BruteForce(CT):
```

```
    print("\nBruteForcing Multiplicative Substitution for Cipher Text: ", CT)
    keys = [1,3,5,7,9,11,15,17,19,21,23,25]
    for k in keys:
      P = Decrypt(CT,k)
      k_inv = getModInverse(26,k)
      print("for Key = ",k,",i.e., k^-1(inverse key) = ",k_inv,", Plain Text is : ",
P, )

PT = input("Input PlainText: ")
#ensuring no spaces in given text
PT = PT.replace(" ","")
k = int(input("Input key: "))

e = Encryption(PT,k)
if e != False:
      #BruteForceMsub(e)
      BruteForce(e)
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\3 Question>python MSub.py
Input PlainText: hello
Input key: 3
for key =  3 Cipher text is :  VMHHQ

BruteForcing Multiplicative Substitution for Cipher Text:  VMHHQ
for Key =  1 ,i.e., k^-1(inverse key) =  1 , Plain Text is :  vmhhq
for Key =  3 ,i.e., k^-1(inverse key) =  9 , Plain Text is :  hello
for Key =  5 ,i.e., k^-1(inverse key) =  21 , Plain Text is :  zsrry
for Key =  7 ,i.e., k^-1(inverse key) =  15 , Plain Text is :  dybbg
for Key =  9 ,i.e., k^-1(inverse key) =  3 , Plain Text is :  lkvvw
for Key =  11 ,i.e., k^-1(inverse key) =  19 , Plain Text is :  judds
for Key =  15 ,i.e., k^-1(inverse key) =  7 , Plain Text is :  rgxxi
for Key =  17 ,i.e., k^-1(inverse key) =  23 , Plain Text is :  pqffe
for Key =  19 ,i.e., k^-1(inverse key) =  11 , Plain Text is :  xczzu
```

```
for Key =  21 ,i.e., k^-1(inverse key) =  5 , Plain Text is :  bijjc

for Key =  23 ,i.e., k^-1(inverse key) =  17 , Plain Text is :  twppm

for Key =  25 ,i.e., k^-1(inverse key) =  25 , Plain Text is :  fottk


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\3 Question>python MSub.py

Input PlainText: hello

Input key: 2

given key is not valid
```

4. Implement Affine cipher and try cryptanalysis.

Programming Language used: Python

Code:

```
def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1


def MultiEncryption(PlainText,Key):
    #ensuring uniformity of plaintext using lower() function
    PlainText = PlainText.lower()
    PTno = []
    #converting plain text to numbers
    for character in PlainText:
        number = ord(character)-97
        PTno.append(number)
    #all possible multiplicative keys i.e. Zn*
    keys = [1,3,5,7,9,11,15,17,19,21,23,25]
    #checking if given key is valid
    exists = False
```

```
    for k in keys:
        if Key == k:
            exists = True
    if exists == False:
        print("given key is not valid")
        return False

    #creating output
    output = []
    for j in PTno:
        num = (j*Key)%26 +97
        output.append(num)
    string_out = [chr(o) for o in output]
    out = ''.join(string_out)
    #print("for key = ", Key, "PLain text is : ",out.upper())
    return out.upper()



def AddEncryption(PlainText,Key):
    #ensuring uniformity of plaintext using lower() function
    PlainText = PlainText.lower()
    PTno = []
    #converting plain text to numbers
    for character in PlainText:
        number = ord(character)-97
        PTno.append(number)
    #checking if given key is valid
    exists = False
    for k in range (1,27):
        if Key == k:
            exists = True
    if exists == False:
        print("given key is not valid")
        return False
```

```
    #creating output
    output = []
    for j in PTno:
        num = (j+Key)%26 +97
        output.append(num)
    #converting from number to letters
    string_out = [chr(o) for o in output]
    out = ''.join(string_out)
    #print("for key = ", Key, "PLain text is : ",out.upper())
    return out.upper()



def Multiplicative_Decrypt(CT,k):
    k_inv = getModInverse(26,k)
    CT = CT.lower()
    CTno = []
    for character in CT:
        number = ord(character) - 97
        CTno.append(number)
    output = []
    for i in CTno:
        num = (i*k_inv)%26 +97
        output.append(num)
    string_out = [chr(o) for o in output]
    return ''.join(string_out)


def Additive_decrypt(CT,k):
    CT = CT.lower()
    CTno = []
    for character in CT:
        number = ord(character) - 97
        CTno.append(number)
    output = []
```

```
    for i in CTno:
        num = (i-k)%26 +97
        output.append(num)
    string_out = [chr(o) for o in output]
    return ''.join(string_out)




def AffineCipher(PlainText,k1,k2):
    T = MultiEncryption(PlainText,k1)
    CT = AddEncryption(T,k2)
    print("for Key Pair(",k1,", ",k2,") Ciphertext is : ",CT)
    return CT

#combination of additive and multiplicative decryptiom:
def AffineBruteForce(CT):
    print("\nBruteForcing AffineCipher for Cipher Text: ", CT)
    k = [1,3,5,7,9,11,15,17,19,21,23,25]
    for k1 in k:
        for k2 in range (1,27):
            P = Multiplicative_Decrypt(Additive_decrypt(CT,k2),k1)
            print("for Key Pair(",k1,", ",k2,") Plain Text is : ", P)



def main():
    PT = input("Input PlainText: ")
    #ensuring no spaces in given text
    PT = PT.replace(" ","")
    k1 = int(input("Input key 1 or multiplicative key: "))
    k2 = int(input("Input key 2 or Additive key: "))
    CT = AffineCipher(PT,k1,k2)
    AffineBruteForce(CT)

#calling main
main()
```

Output(correct option in cryptanalysis is in bold):

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\4 question>python affine.py

Input PlainText: hello

Input key 1 or multiplicative key: 7

Input key 2 or Additive key: 2

for Key Pair( 7 ,  2 ) Ciphertext is :  ZEBBW


BruteForcing AffineCipher for Cipher Text:  ZEBBW

for Key Pair( 1 ,  1 ) Plain Text is :  ydaav

for Key Pair( 1 ,  2 ) Plain Text is :  xczzu

for Key Pair( 1 ,  3 ) Plain Text is :  wbyyt

for Key Pair( 1 ,  4 ) Plain Text is :  vaxxs

for Key Pair( 1 ,  5 ) Plain Text is :  uzwwr

for Key Pair( 1 ,  6 ) Plain Text is :  tyvvq

for Key Pair( 1 ,  7 ) Plain Text is :  sxuup

for Key Pair( 1 ,  8 ) Plain Text is :  rwtto

for Key Pair( 1 ,  9 ) Plain Text is :  qvssn

for Key Pair( 1 ,  10 ) Plain Text is :  purrm

for Key Pair( 1 ,  11 ) Plain Text is :  otqql

for Key Pair( 1 ,  12 ) Plain Text is :  nsppk

for Key Pair( 1 ,  13 ) Plain Text is :  mrooj

for Key Pair( 1 ,  14 ) Plain Text is :  lqnni

for Key Pair( 1 ,  15 ) Plain Text is :  kpmmh

for Key Pair( 1 ,  16 ) Plain Text is :  jollg

for Key Pair( 1 ,  17 ) Plain Text is :  inkkf

for Key Pair( 1 ,  18 ) Plain Text is :  hmjje

for Key Pair( 1 ,  19 ) Plain Text is :  gliid

for Key Pair( 1 ,  20 ) Plain Text is :  fkhhc

for Key Pair( 1 ,  21 ) Plain Text is :  ejggb

for Key Pair( 1 ,  22 ) Plain Text is :  diffa

for Key Pair( 1 ,  23 ) Plain Text is :  cheez

for Key Pair( 1 ,  24 ) Plain Text is :  bgddy

for Key Pair( 1 ,  25 ) Plain Text is :  afccx
```

```
for Key Pair( 1 ,  26 ) Plain Text is :  zebbw
for Key Pair( 3 ,  1 ) Plain Text is :  ibaah
for Key Pair( 3 ,  2 ) Plain Text is :  zsrry
for Key Pair( 3 ,  3 ) Plain Text is :  qjiip
for Key Pair( 3 ,  4 ) Plain Text is :  hazzg
for Key Pair( 3 ,  5 ) Plain Text is :  yrqqx
for Key Pair( 3 ,  6 ) Plain Text is :  pihho
for Key Pair( 3 ,  7 ) Plain Text is :  gzyyf
for Key Pair( 3 ,  8 ) Plain Text is :  xqppw
for Key Pair( 3 ,  9 ) Plain Text is :  ohggn
for Key Pair( 3 ,  10 ) Plain Text is :  fyxxe
for Key Pair( 3 ,  11 ) Plain Text is :  wpoov
for Key Pair( 3 ,  12 ) Plain Text is :  ngffm
for Key Pair( 3 ,  13 ) Plain Text is :  exwwd
for Key Pair( 3 ,  14 ) Plain Text is :  vonnu
for Key Pair( 3 ,  15 ) Plain Text is :  mfeel
for Key Pair( 3 ,  16 ) Plain Text is :  dwvvc
for Key Pair( 3 ,  17 ) Plain Text is :  unmmt
for Key Pair( 3 ,  18 ) Plain Text is :  leddk
for Key Pair( 3 ,  19 ) Plain Text is :  cvuub
for Key Pair( 3 ,  20 ) Plain Text is :  tmlls
for Key Pair( 3 ,  21 ) Plain Text is :  kdccj
for Key Pair( 3 ,  22 ) Plain Text is :  butta
for Key Pair( 3 ,  23 ) Plain Text is :  slkkr
for Key Pair( 3 ,  24 ) Plain Text is :  jcbbi
for Key Pair( 3 ,  25 ) Plain Text is :  atssz
for Key Pair( 3 ,  26 ) Plain Text is :  rkjjq
for Key Pair( 5 ,  1 ) Plain Text is :  klaaz
for Key Pair( 5 ,  2 ) Plain Text is :  pqffe
for Key Pair( 5 ,  3 ) Plain Text is :  uvkkj
for Key Pair( 5 ,  4 ) Plain Text is :  zappo
for Key Pair( 5 ,  5 ) Plain Text is :  efuut
for Key Pair( 5 ,  6 ) Plain Text is :  jkzzy
for Key Pair( 5 ,  7 ) Plain Text is :  opeed
for Key Pair( 5 ,  8 ) Plain Text is :  tujji
```

```
for Key Pair( 5 ,  9 ) Plain Text is :  yzoon
for Key Pair( 5 ,  10 ) Plain Text is :  detts
for Key Pair( 5 ,  11 ) Plain Text is :  ijyyx
for Key Pair( 5 ,  12 ) Plain Text is :  noddc
for Key Pair( 5 ,  13 ) Plain Text is :  stiih
for Key Pair( 5 ,  14 ) Plain Text is :  xynnm
for Key Pair( 5 ,  15 ) Plain Text is :  cdssr
for Key Pair( 5 ,  16 ) Plain Text is :  hixxw
for Key Pair( 5 ,  17 ) Plain Text is :  mnccb
for Key Pair( 5 ,  18 ) Plain Text is :  rshhg
for Key Pair( 5 ,  19 ) Plain Text is :  wxmml
for Key Pair( 5 ,  20 ) Plain Text is :  bcrrq
for Key Pair( 5 ,  21 ) Plain Text is :  ghwwv
for Key Pair( 5 ,  22 ) Plain Text is :  lmbba
for Key Pair( 5 ,  23 ) Plain Text is :  qrggf
for Key Pair( 5 ,  24 ) Plain Text is :  vwllk
for Key Pair( 5 ,  25 ) Plain Text is :  abqqp
for Key Pair( 5 ,  26 ) Plain Text is :  fgvvu
for Key Pair( 7 ,  1 ) Plain Text is :  wtaad
```
for Key Pair(7 , 2) Plain Text is : hello
```
for Key Pair( 7 ,  3 ) Plain Text is :  spwwz
for Key Pair( 7 ,  4 ) Plain Text is :  dahhk
for Key Pair( 7 ,  5 ) Plain Text is :  olssv
for Key Pair( 7 ,  6 ) Plain Text is :  zwddg
for Key Pair( 7 ,  7 ) Plain Text is :  khoor
for Key Pair( 7 ,  8 ) Plain Text is :  vszzc
for Key Pair( 7 ,  9 ) Plain Text is :  gdkkn
for Key Pair( 7 ,  10 ) Plain Text is :  rovvy
for Key Pair( 7 ,  11 ) Plain Text is :  czggj
for Key Pair( 7 ,  12 ) Plain Text is :  nkrru
for Key Pair( 7 ,  13 ) Plain Text is :  yvccf
for Key Pair( 7 ,  14 ) Plain Text is :  jgnnq
for Key Pair( 7 ,  15 ) Plain Text is :  uryyb
for Key Pair( 7 ,  16 ) Plain Text is :  fcjjm
for Key Pair( 7 ,  17 ) Plain Text is :  qnuux
```

```
for Key Pair( 7 ,  18 ) Plain Text is :  byffi
for Key Pair( 7 ,  19 ) Plain Text is :  mjqqt
for Key Pair( 7 ,  20 ) Plain Text is :  xubbe
for Key Pair( 7 ,  21 ) Plain Text is :  ifmmp
for Key Pair( 7 ,  22 ) Plain Text is :  tqxxa
for Key Pair( 7 ,  23 ) Plain Text is :  ebiil
for Key Pair( 7 ,  24 ) Plain Text is :  pmttw
for Key Pair( 7 ,  25 ) Plain Text is :  axeeh
for Key Pair( 7 ,  26 ) Plain Text is :  lipps
for Key Pair( 9 ,  1 ) Plain Text is :  ujaal
for Key Pair( 9 ,  2 ) Plain Text is :  rgxxi
for Key Pair( 9 ,  3 ) Plain Text is :  oduuf
for Key Pair( 9 ,  4 ) Plain Text is :  larrc
for Key Pair( 9 ,  5 ) Plain Text is :  ixooz
for Key Pair( 9 ,  6 ) Plain Text is :  fullw
for Key Pair( 9 ,  7 ) Plain Text is :  criit
for Key Pair( 9 ,  8 ) Plain Text is :  zoffq
for Key Pair( 9 ,  9 ) Plain Text is :  wlccn
for Key Pair( 9 ,  10 ) Plain Text is :  tizzk
for Key Pair( 9 ,  11 ) Plain Text is :  qfwwh
for Key Pair( 9 ,  12 ) Plain Text is :  nctte
for Key Pair( 9 ,  13 ) Plain Text is :  kzqqb
for Key Pair( 9 ,  14 ) Plain Text is :  hwnny
for Key Pair( 9 ,  15 ) Plain Text is :  etkkv
for Key Pair( 9 ,  16 ) Plain Text is :  bqhhs
for Key Pair( 9 ,  17 ) Plain Text is :  yneep
for Key Pair( 9 ,  18 ) Plain Text is :  vkbbm
for Key Pair( 9 ,  19 ) Plain Text is :  shyyj
for Key Pair( 9 ,  20 ) Plain Text is :  pevvg
for Key Pair( 9 ,  21 ) Plain Text is :  mbssd
for Key Pair( 9 ,  22 ) Plain Text is :  jyppa
for Key Pair( 9 ,  23 ) Plain Text is :  gvmmx
for Key Pair( 9 ,  24 ) Plain Text is :  dsjju
for Key Pair( 9 ,  25 ) Plain Text is :  apggr
for Key Pair( 9 ,  26 ) Plain Text is :  xmddo
```

```
for Key Pair( 11 ,  1 ) Plain Text is :  ofaaj
for Key Pair( 11 ,  2 ) Plain Text is :  vmhhq
for Key Pair( 11 ,  3 ) Plain Text is :  ctoox
for Key Pair( 11 ,  4 ) Plain Text is :  javve
for Key Pair( 11 ,  5 ) Plain Text is :  qhccl
for Key Pair( 11 ,  6 ) Plain Text is :  xojjs
for Key Pair( 11 ,  7 ) Plain Text is :  evqqz
for Key Pair( 11 ,  8 ) Plain Text is :  lcxxg
for Key Pair( 11 ,  9 ) Plain Text is :  sjeen
for Key Pair( 11 ,  10 ) Plain Text is :  zqllu
for Key Pair( 11 ,  11 ) Plain Text is :  gxssb
for Key Pair( 11 ,  12 ) Plain Text is :  nezzi
for Key Pair( 11 ,  13 ) Plain Text is :  ulggp
for Key Pair( 11 ,  14 ) Plain Text is :  bsnnw
for Key Pair( 11 ,  15 ) Plain Text is :  izuud
for Key Pair( 11 ,  16 ) Plain Text is :  pgbbk
for Key Pair( 11 ,  17 ) Plain Text is :  wniir
for Key Pair( 11 ,  18 ) Plain Text is :  duppy
for Key Pair( 11 ,  19 ) Plain Text is :  kbwwf
for Key Pair( 11 ,  20 ) Plain Text is :  riddm
for Key Pair( 11 ,  21 ) Plain Text is :  ypkkt
for Key Pair( 11 ,  22 ) Plain Text is :  fwrra
for Key Pair( 11 ,  23 ) Plain Text is :  mdyyh
for Key Pair( 11 ,  24 ) Plain Text is :  tkffo
for Key Pair( 11 ,  25 ) Plain Text is :  armmv
for Key Pair( 11 ,  26 ) Plain Text is :  hyttc
for Key Pair( 15 ,  1 ) Plain Text is :  mvaar
for Key Pair( 15 ,  2 ) Plain Text is :  fottk
for Key Pair( 15 ,  3 ) Plain Text is :  yhmmd
for Key Pair( 15 ,  4 ) Plain Text is :  raffw
for Key Pair( 15 ,  5 ) Plain Text is :  ktyyp
for Key Pair( 15 ,  6 ) Plain Text is :  dmrri
for Key Pair( 15 ,  7 ) Plain Text is :  wfkkb
for Key Pair( 15 ,  8 ) Plain Text is :  pyddu
for Key Pair( 15 ,  9 ) Plain Text is :  irwwn
```

```
for Key Pair( 15 ,  10 ) Plain Text is :  bkppg
for Key Pair( 15 ,  11 ) Plain Text is :  udiiz
for Key Pair( 15 ,  12 ) Plain Text is :  nwbbs
for Key Pair( 15 ,  13 ) Plain Text is :  gpuul
for Key Pair( 15 ,  14 ) Plain Text is :  zinne
for Key Pair( 15 ,  15 ) Plain Text is :  sbggx
for Key Pair( 15 ,  16 ) Plain Text is :  luzzq
for Key Pair( 15 ,  17 ) Plain Text is :  enssj
for Key Pair( 15 ,  18 ) Plain Text is :  xgllc
for Key Pair( 15 ,  19 ) Plain Text is :  qzeev
for Key Pair( 15 ,  20 ) Plain Text is :  jsxxo
for Key Pair( 15 ,  21 ) Plain Text is :  clqqh
for Key Pair( 15 ,  22 ) Plain Text is :  vejja
for Key Pair( 15 ,  23 ) Plain Text is :  oxcct
for Key Pair( 15 ,  24 ) Plain Text is :  hqvvm
for Key Pair( 15 ,  25 ) Plain Text is :  ajoof
for Key Pair( 15 ,  26 ) Plain Text is :  tchhy
for Key Pair( 17 ,  1 ) Plain Text is :  graap
for Key Pair( 17 ,  2 ) Plain Text is :  judds
for Key Pair( 17 ,  3 ) Plain Text is :  mxggv
for Key Pair( 17 ,  4 ) Plain Text is :  pajjy
for Key Pair( 17 ,  5 ) Plain Text is :  sdmmb
for Key Pair( 17 ,  6 ) Plain Text is :  vgppe
for Key Pair( 17 ,  7 ) Plain Text is :  yjssh
for Key Pair( 17 ,  8 ) Plain Text is :  bmvvk
for Key Pair( 17 ,  9 ) Plain Text is :  epyyn
for Key Pair( 17 ,  10 ) Plain Text is :  hsbbq
for Key Pair( 17 ,  11 ) Plain Text is :  kveet
for Key Pair( 17 ,  12 ) Plain Text is :  nyhhw
for Key Pair( 17 ,  13 ) Plain Text is :  qbkkz
for Key Pair( 17 ,  14 ) Plain Text is :  tennc
for Key Pair( 17 ,  15 ) Plain Text is :  whqqf
for Key Pair( 17 ,  16 ) Plain Text is :  zktti
for Key Pair( 17 ,  17 ) Plain Text is :  cnwwl
for Key Pair( 17 ,  18 ) Plain Text is :  fqzzo
```

```
for Key Pair( 17 ,  19 ) Plain Text is :  itccr
for Key Pair( 17 ,  20 ) Plain Text is :  lwffu
for Key Pair( 17 ,  21 ) Plain Text is :  oziix
for Key Pair( 17 ,  22 ) Plain Text is :  rclla
for Key Pair( 17 ,  23 ) Plain Text is :  ufood
for Key Pair( 17 ,  24 ) Plain Text is :  xirrg
for Key Pair( 17 ,  25 ) Plain Text is :  aluuj
for Key Pair( 17 ,  26 ) Plain Text is :  doxxm
for Key Pair( 19 ,  1 ) Plain Text is :  ehaax
for Key Pair( 19 ,  2 ) Plain Text is :  twppm
for Key Pair( 19 ,  3 ) Plain Text is :  ileeb
for Key Pair( 19 ,  4 ) Plain Text is :  xattq
for Key Pair( 19 ,  5 ) Plain Text is :  mpiif
for Key Pair( 19 ,  6 ) Plain Text is :  bexxu
for Key Pair( 19 ,  7 ) Plain Text is :  qtmmj
for Key Pair( 19 ,  8 ) Plain Text is :  fibby
for Key Pair( 19 ,  9 ) Plain Text is :  uxqqn
for Key Pair( 19 ,  10 ) Plain Text is :  jmffc
for Key Pair( 19 ,  11 ) Plain Text is :  ybuur
for Key Pair( 19 ,  12 ) Plain Text is :  nqjjg
for Key Pair( 19 ,  13 ) Plain Text is :  cfyyv
for Key Pair( 19 ,  14 ) Plain Text is :  runnk
for Key Pair( 19 ,  15 ) Plain Text is :  gjccz
for Key Pair( 19 ,  16 ) Plain Text is :  vyrro
for Key Pair( 19 ,  17 ) Plain Text is :  knggd
for Key Pair( 19 ,  18 ) Plain Text is :  zcvvs
for Key Pair( 19 ,  19 ) Plain Text is :  orkkh
for Key Pair( 19 ,  20 ) Plain Text is :  dgzzw
for Key Pair( 19 ,  21 ) Plain Text is :  svool
for Key Pair( 19 ,  22 ) Plain Text is :  hkdda
for Key Pair( 19 ,  23 ) Plain Text is :  wzssp
for Key Pair( 19 ,  24 ) Plain Text is :  lohhe
for Key Pair( 19 ,  25 ) Plain Text is :  adwwt
for Key Pair( 19 ,  26 ) Plain Text is :  pslli
for Key Pair( 21 ,  1 ) Plain Text is :  qpaab
```

```
for Key Pair( 21 ,  2 ) Plain Text is :  lkvvw
for Key Pair( 21 ,  3 ) Plain Text is :  gfqqr
for Key Pair( 21 ,  4 ) Plain Text is :  ballm
for Key Pair( 21 ,  5 ) Plain Text is :  wvggh
for Key Pair( 21 ,  6 ) Plain Text is :  rqbbc
for Key Pair( 21 ,  7 ) Plain Text is :  mlwwx
for Key Pair( 21 ,  8 ) Plain Text is :  hgrrs
for Key Pair( 21 ,  9 ) Plain Text is :  cbmmn
for Key Pair( 21 ,  10 ) Plain Text is :  xwhhi
for Key Pair( 21 ,  11 ) Plain Text is :  srccd
for Key Pair( 21 ,  12 ) Plain Text is :  nmxxy
for Key Pair( 21 ,  13 ) Plain Text is :  ihsst
for Key Pair( 21 ,  14 ) Plain Text is :  dcnno
for Key Pair( 21 ,  15 ) Plain Text is :  yxiij
for Key Pair( 21 ,  16 ) Plain Text is :  tsdde
for Key Pair( 21 ,  17 ) Plain Text is :  onyyz
for Key Pair( 21 ,  18 ) Plain Text is :  jittu
for Key Pair( 21 ,  19 ) Plain Text is :  edoop
for Key Pair( 21 ,  20 ) Plain Text is :  zyjjk
for Key Pair( 21 ,  21 ) Plain Text is :  uteef
for Key Pair( 21 ,  22 ) Plain Text is :  pozza
for Key Pair( 21 ,  23 ) Plain Text is :  kjuuv
for Key Pair( 21 ,  24 ) Plain Text is :  feppq
for Key Pair( 21 ,  25 ) Plain Text is :  azkkl
for Key Pair( 21 ,  26 ) Plain Text is :  vuffg
for Key Pair( 23 ,  1 ) Plain Text is :  szaat
for Key Pair( 23 ,  2 ) Plain Text is :  bijjc
for Key Pair( 23 ,  3 ) Plain Text is :  krssl
for Key Pair( 23 ,  4 ) Plain Text is :  tabbu
for Key Pair( 23 ,  5 ) Plain Text is :  cjkkd
for Key Pair( 23 ,  6 ) Plain Text is :  lsttm
for Key Pair( 23 ,  7 ) Plain Text is :  ubccv
for Key Pair( 23 ,  8 ) Plain Text is :  dklle
for Key Pair( 23 ,  9 ) Plain Text is :  mtuun
for Key Pair( 23 ,  10 ) Plain Text is :  vcddw
```

```
for Key Pair( 23 ,  11 ) Plain Text is :  elmmf
for Key Pair( 23 ,  12 ) Plain Text is :  nuvvo
for Key Pair( 23 ,  13 ) Plain Text is :  wdeex
for Key Pair( 23 ,  14 ) Plain Text is :  fmnng
for Key Pair( 23 ,  15 ) Plain Text is :  ovwwp
for Key Pair( 23 ,  16 ) Plain Text is :  xeffy
for Key Pair( 23 ,  17 ) Plain Text is :  gnooh
for Key Pair( 23 ,  18 ) Plain Text is :  pwxxq
for Key Pair( 23 ,  19 ) Plain Text is :  yfggz
for Key Pair( 23 ,  20 ) Plain Text is :  hoppi
for Key Pair( 23 ,  21 ) Plain Text is :  qxyyr
for Key Pair( 23 ,  22 ) Plain Text is :  zghha
for Key Pair( 23 ,  23 ) Plain Text is :  ipqqj
for Key Pair( 23 ,  24 ) Plain Text is :  ryzzs
for Key Pair( 23 ,  25 ) Plain Text is :  ahiib
for Key Pair( 23 ,  26 ) Plain Text is :  jqrrk
for Key Pair( 25 ,  1 ) Plain Text is :  cxaaf
for Key Pair( 25 ,  2 ) Plain Text is :  dybbg
for Key Pair( 25 ,  3 ) Plain Text is :  ezcch
for Key Pair( 25 ,  4 ) Plain Text is :  faddi
for Key Pair( 25 ,  5 ) Plain Text is :  gbeej
for Key Pair( 25 ,  6 ) Plain Text is :  hcffk
for Key Pair( 25 ,  7 ) Plain Text is :  idggl
for Key Pair( 25 ,  8 ) Plain Text is :  jehhm
for Key Pair( 25 ,  9 ) Plain Text is :  kfiin
for Key Pair( 25 ,  10 ) Plain Text is :  lgjjo
for Key Pair( 25 ,  11 ) Plain Text is :  mhkkp
for Key Pair( 25 ,  12 ) Plain Text is :  nillq
for Key Pair( 25 ,  13 ) Plain Text is :  ojmmr
for Key Pair( 25 ,  14 ) Plain Text is :  pknns
for Key Pair( 25 ,  15 ) Plain Text is :  qloot
for Key Pair( 25 ,  16 ) Plain Text is :  rmppu
for Key Pair( 25 ,  17 ) Plain Text is :  snqqv
for Key Pair( 25 ,  18 ) Plain Text is :  torrw
for Key Pair( 25 ,  19 ) Plain Text is :  upssx
```

```
for Key Pair( 25 ,  20 ) Plain Text is :  vqtty
for Key Pair( 25 ,  21 ) Plain Text is :  wruuz
for Key Pair( 25 ,  22 ) Plain Text is :  xsvva
for Key Pair( 25 ,  23 ) Plain Text is :  ytwwb
for Key Pair( 25 ,  24 ) Plain Text is :  zuxxc
for Key Pair( 25 ,  25 ) Plain Text is :  avyyd
for Key Pair( 25 ,  26 ) Plain Text is :  bwzze
```

5. Implement Autokey and Playfair ciphers.

Programming Language used: Python

Autokey Cipher

Code:

```
def Encryption(PlainText,k):
    #removing blank spaces in given string
    PT = PlainText.replace(" ","")
    #converting plain text to numbers
    PT = PT.lower()

    PTno = []
    Key = []
    Key.append(k)
    #adding all charcter numbers to Plaintext and key list
    for char in PT:
        num = ord(char) - 97
        PTno.append(num)
        Key.append(num)
    #removing last element from list as it is not needed
    Key.pop()
    CTno = []
    for(pi,ki)in zip(PTno,Key):
        CTno.append((pi + ki)%26 + 97)

    CT_out = [chr(o) for o in CTno]
    CT =''.join(CT_out)
    return CT.upper()




def Decryption(CT,key):
    #once we decrypt the first letter we have to use the same letter to decrypt next
letter
    CT = CT.lower()
    CTno = []
```

```
    for char in CT:
        num = ord(char) - 97
        CTno.append(num)
    ki = key

    P = []
    for c in CTno:
        P.append((c-ki)%26 +97)
        ki = (c-ki)%26
    PT_out = [chr(o) for o in P]
    PT = ''.join(PT_out)
    return PT

def main():
    PT = input("Enter Text: ")
    PT = PT.replace(" ", "") #removing spaces in plain text
    Key = int(input("Input key in range 0-25: "))
    CT = Encryption(PT,12)
    print("Encrypted message: ",CT)
    print("\n Decryting message gives: ",Decryption(CT,12))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\5 question>python autoKey.py

Enter Text: Attack Is Today

Input key in range 0-25: 12

Encrypted message:  MTMTCMSALHRDY


 Decryting message gives:  attackistoday
```

Playfair Cipher

Code:

```
#note dummy variable is X
import string



# create matrix without duplicates

#find index of given element in matrix as i,j
def get_index(matrix, element):
	for i in matrix:
		if element == 'I':
			element = ('I','J')

		if element == 'J':
			element = ('I','J')
		if(element in i):
			return (matrix.index(i),i.index(element))



def k_exists(matrix, k):
	#here i is each sub list which as a whole form the matrix
	for i in matrix:
		exists = k in i
		if(exists):
			return True
	return False



def handle_IJ(matrix): #for handling I,j
#iterating throught each sub list to
#find existence of I as we only allowe I to be entered
	for i in matrix:
		exists = 'I' in i
		if(exists):
```

```
                matrix[matrix.index(i)][i.index('I')] = ('I','J')
        return matrix
        # return False



def key_matrix(Key): #key is a string
        Key = Key.upper()
        #initialising matrix with dummy values to be edited later when filling
        K_matrix = [[0,0,0,0,0],
                                [0,0,0,0,0],
                                [0,0,0,0,0],
                                [0,0,0,0,0],
                                [0,0,0,0,0]]
        i = 0
        j = 0
        for k in Key:
                if(k_exists(K_matrix,k)!= True):
                        if k == 'J': #replacing J with I it will be replaced later with
(I,J)
                                k = 'I'
                        K_matrix[i][j] = k
                        j +=1
                if(j>=5): #resetting value of j to go to next row
                        j = 0
                        i +=1
        #getting string of alphabets to enter
        alphabet_string = string.ascii_uppercase
        allowed_alphabet = []
        for char in alphabet_string:
                if(k_exists(K_matrix,char)!= True):
                        if char != 'J':
                                allowed_alphabet.append(char)
        for c in allowed_alphabet:
                K_matrix[i][j] = c
                j+=1
```

```
        if j>=5:
            j =0
            i +=1
    # print(K_matrix)
    final_key_matrix = handle_IJ(K_matrix)



    # print(final_key_matrix)
    return final_key_matrix



#creating pairs  using list and
#tuples we use tuple as it is ordered and unchangable
def get_pairs(PlainText): #creating letter pairs of given plain text
    PlainText = PlainText.lower()
    string_len = len(PlainText)
    #gonna take values and then we convert to tuple when pair filled
    temp = []
    i = 0
    j = 0
    pair_list = []
    for char in PlainText:
        if(char in temp): #checking possibility of duplicate in pai
            temp.append('x')
            pair_list.append(tuple(temp))
            i = 0
            temp = []
        temp.append(char)
        i += 1
        if i == 2:
            i = 0
            pair_list.append(tuple(temp))
            temp = [] #resetting
    #if last pair not made
    if temp != []:
```

```
            temp.append('x') #dummy val
            pair_list.append(tuple(temp))


      return pair_list




def Encrypt_pair(pair, key_matrix):
      a = pair[0].upper()
      b = pair[1].upper()
      i1 = get_index(key_matrix,a)
      i2 = get_index(key_matrix,b)


      #if in same row
      if(i1[0] == i2[0]):
            if i1[1] >= 4:
                  a_out = key_matrix[i1[0]][0]
            else:
                  a_out = key_matrix[i1[0]][i1[1]+1]
            if i2[1] >= 4:
                  b_out = key_matrix[i2[0]][0]
            else:
                  b_out = key_matrix[i2[0]][i2[1]+1]

      #if in same column
      elif(i1[1] == i2[1]):
            if i1[0] >= 4:
                  a_out = key_matrix[0][i1[1]]
            else:
                  a_out = key_matrix[i1[0]+1][i1[1]]
            if i2[0] >= 4:
                  b_out = key_matrix[0][i2[1]]
```

```
        else:
            b_out = key_matrix[i2[0]+1][i2[1]]


    #otherwise
    else:
        a_out = key_matrix[i1[0]][i2[1]]
        b_out = key_matrix[i2[0]][i1[1]]
    return (a_out,b_out)


def Decrypt_pair(pair, key_matrix):
    a = pair[0].upper()
    b = pair[1].upper()
    i1 = get_index(key_matrix,a)
    i2 = get_index(key_matrix,b)


    #if in same row
    if(i1[0] == i2[0]):
        if i1[1] <= 0:
            a_out = key_matrix[i1[0]][4]
        else:
            a_out = key_matrix[i1[0]][i1[1]-1]
        if i2[1] <= 0:
            b_out = key_matrix[i2[0]][4]
        else:
            b_out = key_matrix[i2[0]][i2[1]-1]

    #if in same column
    elif(i1[1] == i2[1]):
        if i1[0] <= 0:
            a_out = key_matrix[4][i1[1]]
        else:
            a_out = key_matrix[i1[0]-1][i1[1]]
```

```
        if i2[0] <= 0:
            b_out = key_matrix[4][i2[1]]
        else:
            b_out = key_matrix[i2[0]-1][i2[1]]


    #otherwise
    else:
        a_out = key_matrix[i1[0]][i2[1]]
        b_out = key_matrix[i2[0]][i1[1]]
    return (a_out,b_out)


def Encryption(PlainText,key):
    pair_list = get_pairs(PlainText)
    key_m = key_matrix(key)
    C_pair_list = []
    for pair in pair_list:
        c = Encrypt_pair(pair,key_m)
        C_pair_list.append(c)
    CT_list = []
    for pair in C_pair_list:
        for element in pair:
            if element == ('I','J'):
                CT_list.append('I')
            else:
                CT_list.append(element)
    CipherText = ''.join(CT_list)
    return CipherText


def Decryption(CipherText,key):
    pair_list = get_pairs(CipherText)
    key_m = key_matrix(key)
    P_pair_list = []
```

```
    for pair in pair_list:
        p = Decrypt_pair(pair,key_m)
        P_pair_list.append(p)
    PT_list = []
    for pair in P_pair_list:
        for element in pair:
            if(element != 'X'):
                if element == ('I','J'):
                    PT_list.append('I')
                else:
                    PT_list.append(element)
    PlainText = ''.join(PT_list)
    return PlainText.lower()




def main():
    PT  = input("Input plain Text: ")
    Key = input ("Input Key: ")

    PT = PT.replace(" ", "") #removing spaces in plain text

    CT  = Encryption(PT,Key)
    print("Cipher text is: ", CT, "\n")

    print("Decryption of above given ciphertext: ",Decryption(CT,Key))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\5 question>python playFair.py

Input plain Text: have a good day

Input Key: crypt

Cipher text is:  GBXBGMQWWIEBPW


Decryption of above given ciphertext:  haveagoodday
```

6. Implement Vigenère cipher and try cryptanalysis.

Programming Language used: Python

Code:

```
import math
import itertools
import string
import time

def Encryption(PlainText, Key):
    #convert plaintext and key into list of numbers
    #ensuring no spaces in given text
    PlainText = PlainText.replace(" ","")

    PlainText = PlainText.lower()
    PTno = []
    #converting plain text to numbers
    for character in PlainText:
        number = ord(character)-97
        PTno.append(number)

    Key = Key.lower()
    Kno = []
    #converting Key stream to numbers
    for character in Key:
        number = ord(character)-97
        Kno.append(number)

    Kno_len = len(Kno) #we will use this with mod and iterator and add to plain text
no
    k_iterator = 0

    output = []
    for i in PTno:
        num = (i+Kno[k_iterator%Kno_len])%26 +97
        output.append(num)
```

```
        k_iterator += 1
    string_out = [chr(o) for o in output]
    CT = ''.join(string_out)

    return CT.upper()

def Decryption(CipherText, Key):
    #convert plaintext and key into list of numbers

    CT = CipherText.lower()
    CTno = []
    #converting plain text to numbers
    for character in CT:
        number = ord(character)-97
        CTno.append(number)

    Key = Key.lower()
    Kno = []
    #converting Key stream to numbers
    for character in Key:
        number = ord(character)-97
        Kno.append(number)

    Kno_len = len(Kno) #we will use this with mod and iterator and add to plain text
no
    k_iterator = 0

    output = []
    for i in CTno:
        num = (i-Kno[k_iterator%Kno_len])%26 +97
        output.append(num)
        k_iterator += 1
    string_out = [chr(o) for o in output]
    PT = ''.join(string_out)
```

```
    return PT


#cryptanalysis
def Kasiski_test(CT):


    print("\n PERFORMING KASISKI TEST")
    #converting Ciphertext into list as list is easy to iterate through
    CT_list = []
    for char in CT:
        CT_list.append(char)
    CT_len = len(CT)
    Difference_list = [] #need to get gcd of numbers in this lis


    for i in range(0,CT_len):


        for j in range(i+5,CT_len):
            if(CT_list[i]==CT_list[j-2] and CT_list[i+1]==CT_list[j-1] and
CT_list[i+2]==CT_list[j]):
                first_index = i
                second_index = j-2
                Difference = j-2-i
                Difference_list.append(Difference)
                break
    GCD = Difference_list[0]
    for i in range (1,len(Difference_list)):
        GCD = math.gcd(GCD,Difference_list[i])
    print("Key Length is multiple of: ", GCD)
    m = GCD
    #now we brute force using this information:
    #for experiment sake we will limit to m letter words storgae can go upto 100 mb so
here we only go till code



    for key_tuple in itertools.product(string.ascii_lowercase, repeat=m):
        key = ''.join(key_tuple)
        print("With Key = ",key," Decrypted message: ", Decryption(CT,key))
```

```
def main():
    #hard coding input for example can change key and plaintext
    #as per requirements.
    PT = "she is listening"
    K = "PASCAL"
    print("Given Plain text: ", PT)
    print("Given Key: ", K)
    CT = Encryption(PT,K)
    print("Cipher text when Encrypted: ", CT)
    print("Decryption of Cipher text: ",Decryption(CT,K))
    CT_test =
"LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEVLWPCZVGTHVTSGXQOVGCSVETQLTJSUMVWVEUVLXEWSLGFZMVVWLGYHCUSWXQHKVGSHEEVFLCFDGVSUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUEEHHVUCFVGOWICQLTJSUXGLW"
    print("\n sample cipher text for using kasiskit on: ", CT_test)
    time.sleep(5)
    Kasiski_test(CT_test)

main()
```

Output(Cryptanalysis correct option in bold):

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\6 question>python Vigenere.py
Given Plain text:  she is listening
Given Key:  PASCAL
Cipher text when Encrypted:  HHWKSWXSLGNTCG
Decryption of Cipher text:  sheislistening
```

sample cipher text for using kasiskit on:
LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETKKMEVLWPCZVGTHVTSGXQOVGCSVETQLTJSUMVWVEUVLXEWSLGF
ZMVVWLGYHCUSWXQHKVGSHEEVFLCFDGVSUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUEEHHVUCFVGOWICQLTJSUX
GLW

PERFORMING KASISKI TEST

Key Length is multiple of: 4

With Key = aaaa Decrypted message:
liomwgfeggdvwghhcqucrhrwagwiowqlkgzetkkmevlwpczvgthvtsgxqovgcsvetqltjsumvwveuvlxewslgf
zmvvwlgyhcuswxqhkvgsheevflcfdgvsumphkirzdmphhbvwvwjwixgfwltshgjoueehhvucfvgowicqltjsux
glw

With Key = aaab Decrypted message:
liolwgfdggduwghgcqubrhrvagwhowqkkgzdtkklevlvpczugthutsgwqovfcsvdtqlsjsulvwvduvlwewskgf
zlvvwkgyhbuswwqhkugshdevfkcfdfvsulphkhrzdlphhavwvvjwiwgfwktshfjoudehhuucfugowhcqlsjsuw
glw

With Key = aaac Decrypted message:
liokwgfcggdtwghfcquarhruagwgowqjkgzctkkkevlupcztgthttsgvqovecsvctqlrjsukvwvcuvlvewsjgf
zkvvwjgyhauswvqhktgshcevfjcfdevsukphkgrzdkphhzvwvujwivgfwjtshejoucehhtucftgowgcqlrjsuv
glw

With Key = aaad Decrypted message:
liojwgfbggdswghecquzrhrtagwfowqikgzbtkkjevltpczsgthstsguqovdcsvbtqlqjsujvwvbuvluewsigf
zjvvwigyhzuswuqhksgshbevficfddvsujphkfrzdjphhyvwvtjwiugfwitshdjoubehhsucfsgowfcqlqjsuu
glw

With Key = aaae Decrypted message:
lioiwgfaggdrwghdcquyrhrsagweowqhkgzatkkievlspczrgthrtsgtqovccsvatqlpjsuivwvauvltewshgf
zivvwhgyhyuswtqhkrgshaevfhcfdcvsuiphkerzdiphhxvwvsjwitgfwhtshcjouaehhrucfrgowecqlpjsut
glw

With Key = aaaf Decrypted message:
liohwgfzggdqwghccquxrhrragwdowqgkgzztkkhevlrpczqgthqtsgsqovbcsvztqlojsuhvwvzuvlsewsggf
zhvvwggyhxuswsqhkqgshzevfgcfdbvsuhphkdrzdhphhwvwvrjwisgfwgtshbjouzehhqucfqgowdcqlojsus
glw

With Key = aaag Decrypted message:
liogwgfyggdpwghbcquwrhrqagwcowqfkgzytkkgevlqpczpgthptsgrqovacsvytqlnjsugvwvyuvlrewsfgf
zgvvwfgyhwuswrqhkpgshyevffcfdavsugphkcrzdgphhvvwvqjwirgfwftshajouyehhpucfpgowccqlnjsur
glw

With Key = aaah Decrypted message:
liofwgfxggdowghacquvrhrpagwbowqekgzxtkkfevlppczogthotsgqqovzcsvxtqlmjsufvwvxuvlqewsegf
zfvvwegyhvuswqqhkogshxevfecfdzvsufphkbrzdfphhuvwvpjwiqgfwetshzjouxehhoucfogowbcqlmjsuq
glw

With Key = aaai Decrypted message:
lioewgfwggdnwghzcquurhroagwaowqdkgzwtkkeevlopczngthntsgpqovycsvwtqlljsuevwvwuvlpewsdgf
zevvwdgyhuuswpqhkngshwevfdcfdyvsuephkarzdephhtvwvojwipgfwdtshyjouwehhnucfngowacqlljsup
glw

With Key = aaaj Decrypted message:
liodwgfvggdmwghycqutrhrnagwzowqckgzvtkkdevlnpczmgthmtsgoqovxcsvvtqlkjsudvwvvuvloewscgf

zdvvwcgyhtuswoqhkmgshvevfccfdxvsudphkzrzddphhsvwvnjwiogfwctshxjouvehhmucfmgowzcqlkjsuo
glw

With Key = aaak Decrypted message:
liocwgfuggdlwghxcqusrhrmagwyowqbkgzutkkcevlmpczlgthltsgnqovwcsvutqljjsucvwvuuvlnewsbgf
zcvvwbgyhsuswnqhklgshuevfbcfdwvsucphkyrzdcphhrvwvmjwingfwbtshwjouuehhlucflgowycqljjsun
glw

With Key = aaal Decrypted message:
liobwgftggdkwghwcqurrhrlagwxowqakgzttkkbevllpczkgthktsgmqovvcsvttqlijsubvwvtuvlmewsagf
zbvvwagyhruswmqhkkgshtevfacfdvvsubphkxrzdbphhqvwvljwimgfwatshvjoutehhkucfkgowxcqlijsum
glw

With Key = aaam Decrypted message:
lioawgfsggdjwghvcquqrhrkagwwowqzkgzstkkaevlkpczjgthjtsglqovucsvstqlhjsuavwvsuvllewszgf
zavvwzgyhquswlqhkjgshsevfzcfduvsuaphkwrzdaphhpvwvkjwilgfwztshujousehhjucfjgowwcqlhjsul
glw

With Key = aaan Decrypted message:
liozwgfrggdiwghucquprhrjagwvowqykgzrtkkzevljpczigthitsgkqovtcsvrtqlgjsuzvwvruvlkewsygf
zzvvwygyhpuswkqhkigshrevfycfdtvsuzphkvrzdzphhovwvjjwikgfwytshtjourehhiucfigowvcqlgjsuk
glw

…………

…………

…………

With Key = cocx Decrypted message:
jumpusdhesbyusfkacsfptpzysulmiooisxhrwipchjznoxyeffyreeaoatjaethrcjwhesptithshjaciqoer
xpthuoekffseuaotiyeefhchdoarbjtespntilplbpntfetitzhigaeruorefjhashctfysodyeaulacjwhesa
exu

With Key = cocy Decrypted message:
jumousdgesbxusfjacseptpyysukmionisxgrwiochjynoxxeffxreezoatiaetgrcjvhesotitgshjzciqner
xothunekfeseuzotixeefgchdnarbitesontikplbontfdtityhigzerunrefihasgctfxsodxeaukacjvhesz
exu

With Key = cocz Decrypted message:
jumnusdfesbwusfiacsdptpxysujmiomisxfrwinchjxnoxweffwreeyoathaetfrcjuhesntitfshjyciqmer
xnthumekfdseuyotiweeffchdmarbhtesnntijplbnntfctitxhigyerumrefhhasfctfwsodweaujacjuhesy
exu

With Key = coda Decrypted message:
julmusceesavusehacrcptowystiminliswerwhmchiwnowvefevredxoasgaesercithermtiseshixcipler
wmthtlekecsetxothveeeechclaragtermnthiplamntebtiswhifxertlreegharectevsocveatiacitherx
ext

With Key = codb Decrypted message:
julluscdesauusegacrbptovysthminkiswdrwhlchivnowuefeuredwoasfaesdrcisherltisdshiwcipker
wlthtkekebsetwothueeedchckarafterlnthhplalnteatisvhifwertkreefhardcteusocueathacisherw
ext

With Key = codc Decrypted message:
julkusccesatusefacraptouystgminjiswcrwhkchiunowtefetredvoaseaescrcirherktiscshivcipjer
wkthtjekeasetvothteeecchcjaraeterknthgplaknteztisuhifvertjreeeharcctetsocteatgacirherv
ext

With Key = codd Decrypted message:
juljuscbesasuseeacrzptotystfminiiswbrwhjchitnowsefesreduoasdaesbrciqherjtisbshiucipier
wjthtiekezsetuothseeebchciaradterjnthfplajnteytisthifuertireedharbctessocseatfaciqheru
ext

With Key = code Decrypted message:
juliuscaesarusedacryptosysteminhiswarwhichisnowreferredtoascaesarcipheritisashitcipher
withthekeysettothreeeachcharacterintheplaintextisshifterthreecharactersocreateaciphert
ext

With Key = codf Decrypted message:
julhusczesaqusecacrxptorystdmingiswzrwhhchirnowqefeqredsoasbaeszrcioherhtiszshiscipger
whthtgekexsetsothqeeezchcgarabterhnthdplahntewtisrhifsertgreebharzcteqsocqeatdaciohers
ext

With Key = codg Decrypted message:
julguscyesapusebacrwptoqystcminfiswyrwhgchiqnowpefepredroasaaesyrcinhergtisyshircipfer
wgthtfekewsetrothpeeeychcfaraatergnthcplagntevtisqhifrertfreeaharyctepsocpeatcacinherr
ext

With Key = codh Decrypted message:
julfuscxesaouseaacrvptopystbmineiswxrwhfchipnowoefeoredqoaszaesxrcimherftisxshiqcipeer
wfthteekevsetqothoeeexchcearazterfnthbplafnteutisphifqertereezharxcteosocoeatbacimherq
ext

With Key = codi Decrypted message:
juleuscwesanusezacruptooystamindiswwrwhechionownefenredpoasyaeswrcilheretiswshipcipder
wethtdekeusetpothneeewchcdarayterenthaplaentettisohifpertdreeyharwctensocneataacilherp
ext

With Key = codj Decrypted message:
julduscvesamuseyacrtptonystzminciswvrwhdchinnowmefemredooasxaesvrcikherdtisvshiocipcer
wdthtceketsetoothmeeevchccaraxterdnthzpladntestisnhifoertcreexharvctemsocmeatzacikhero
ext

…………

…………

…………

With Key = zzzm Decrypted message:
mjpaxhgshhejxhivdrvqsiskbhxwpxrzlhasullafwmkqdajhuijuthlrpwudtwsurmhktvawxwsvwmlfxtzhg
aawwxzhziqvtxlriljhtisfwgzdgeuwtvaqilwsaeaqiipwxwkkxjlhgxzutiukpvsfiijvdgjhpxwdrmhktvl
hmx

With Key = zzzn Decrypted message:
mjpzxhgrhheixhiudrvpsisjbhxvpxrylharullzfwmjqdaihuiiuthkrpwtdtwrurmgktvzwxwrvwmkfxtyhg
azwwxyhzipvtxkrilihtirfwgydgetwtvzqilvsaezqiiowxwjkxjkhgxyutitkpvrfiiivdgihpxvdrmgktvk
hmx

With Key = zzzo Decrypted message:
mjpyxhgqhhehxhitdrvosisibhxupxrxlhaqullyfwmiqdahhuihuthjrpwsdtwqurmfktvywxwqvwmjfxtxhg
aywwxxhziovtxjrilhhtiqfwgxdgeswtvyqilusaeyqiinwxwikxjjhgxxutiskpvqfiihvdghhpxudrmfktvj
hmx

With Key = zzzp Decrypted message:
mjpxxhgphhegxhisdrvnsishbhxtpxrwlhapullxfwmhqdaghuiguthirpwrdtwpurmektvxwxwpvwmifxtwhg
axwwxwhzinvtxirilghtipfwgwdgerwtvxqiltsaexqiimwxwhkxjihgxwutirkpvpfiigvdgghpxtdrmektvi
hmx

With Key = zzzq Decrypted message:
mjpwxhgohhefxhirdrvmsisgbhxspxrvlhaoullwfwmgqdafhuifuthhrpwqdtwourmdktvwwxwovwmhfxtvhg
awwwxvhzimvtxhrilfhtiofwgvdgeqwtvwqilssaewqiilwxwgkxjhhgxvutiqkpvofiifvdgfhpxsdrmdktvh
hmx

With Key = zzzr Decrypted message:
mjpvxhgnhheexhiqdrvlsisfbhxrpxrulhanullvfwmfqdaehuieuthgrpwpdtwnurmcktvvwxwnvwmgfxtuhg
avwwxuhzilvtxgrilehtinfwgudgepwtvvqilrsaevqiikwxwfkxjghgxuutipkpvnfiievdgehpxrdrmcktvg
hmx

With Key = zzzs Decrypted message:
mjpuxhgmhhedxhipdrvksisebhxqpxrtlhamullufwmeqdadhuiduthfrpwodtwmurmbktvuwxwmvwmffxtthg
auwwxthzikvtxfrildhtimfwgtdgeowtvuqilqsaeuqiijwxwekxjfhgxtutiokpvmfiidvdgdhpxqdrmbktvf
hmx

With Key = zzzt Decrypted message:
mjptxhglhhecxhiodrvjsisdbhxppxrslhalulltfwmdqdachuicutherpwndtwlurmaktvtwxwlvwmefxtshg
atwwxshzijvtxerilchtilfwgsdgenwtvtqilpsaetqiiiwxwdkxjehgxsutinkpvlfiicvdgchpxpdrmaktve
hmx

With Key = zzzu Decrypted message:
mjpsxhgkhhebxhindrvisiscbhxopxrrlhakullsfwmcqdabhuibuthdrpwmdtwkurmzktvswxwkvwmdfxtrhg
aswwxrhziivtxdrilbhtikfwgrdgemwtvsqilosaesqiihwxwckxjdhgxrutimkpvkfiibvdgbhpxodrmzktvd
hmx

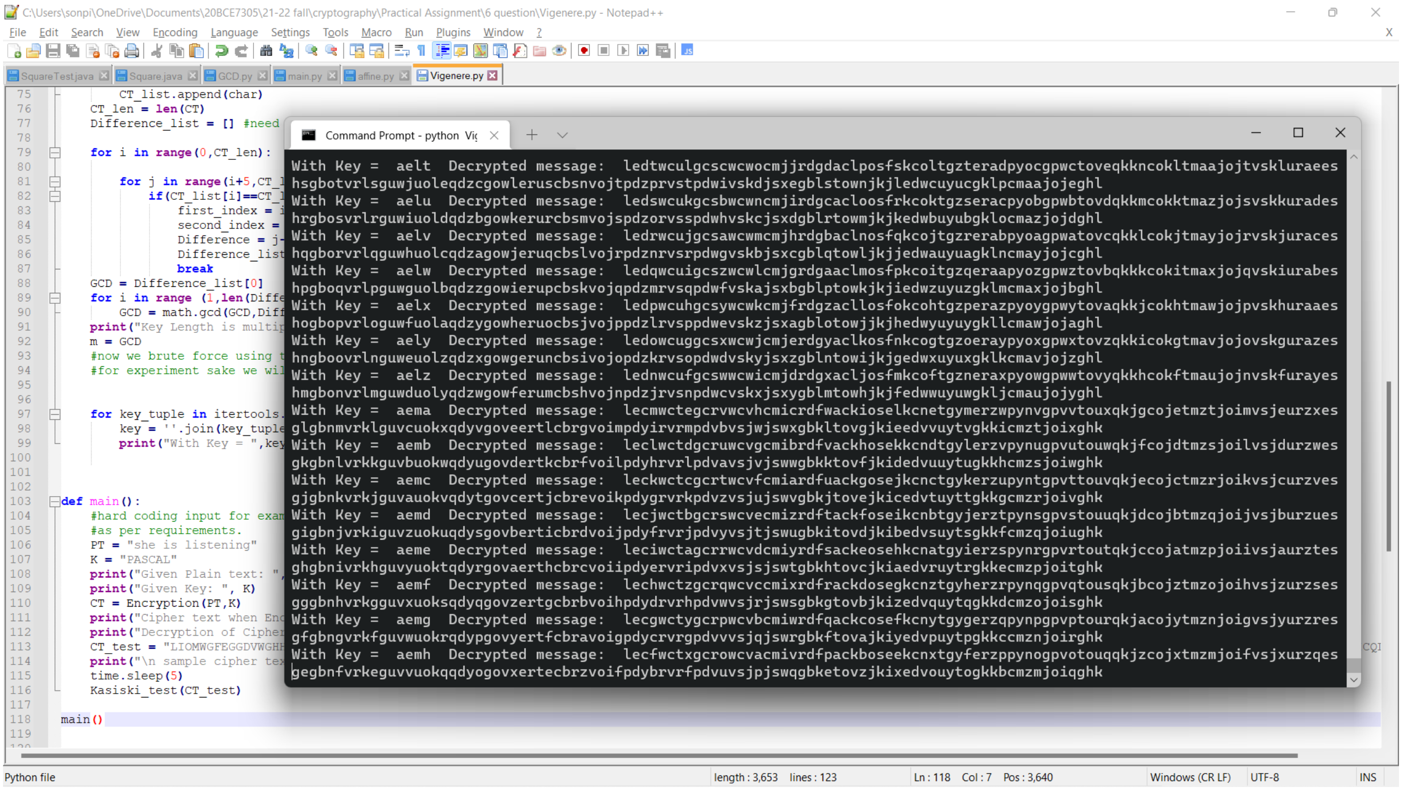
With Key = zzzv Decrypted message:
mjprxhgjhheaxhimdrvhsisbbhxnpxrqlhajullrfwmbqdaahuiauthcrpwldtwjurmyktvrwxwjvwmcfxtqhg
arwwxqhzihvtxcrilahtijfwgqdgelwtvrqilnsaerqiigwxwbkxjchgxqutilkpvjfiiavdgahpxndrmyktvc
hmx

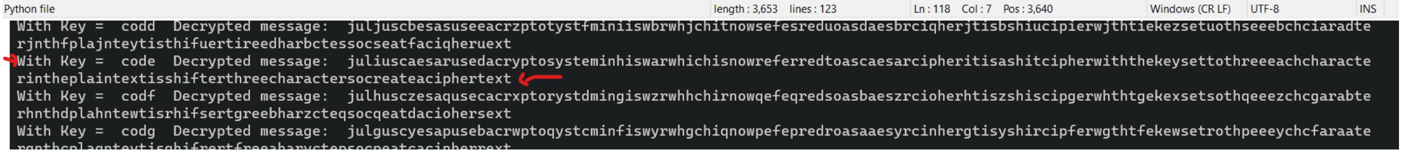
With Key = zzzw Decrypted message:
mjpqxhgihhezxhildrvgsisabhxmpxrplhaiullqfwmaqdazhuizuthbrpwkdtwiurmxktvqwxwivwmbfxtphg
aqwwxphzigvtxbrilzhtiifwgpdgekwtvqqilmsaeqqiifwxwakxjbhgxputikkpvifiizvdgzhpxmdrmxktvb
hmx

With Key = zzzx Decrypted message:
mjppxhghhheyxhikdrvfsiszbhxlpxrolhahullpfwmzqdayhuiyutharpwjdtwhurmwktvpwxwhvwmafxtohg
apwwxohzifvtxarilyhtihfwgodgejwtvpqillsaepqiiewxwzkxjahgxoutijkpvhfiiyvdgyhpxldrmwktva
hmx

With Key = zzzy Decrypted message:
mjpoxhgghhexxhijdrvesisybhxkpxrnlhagullofwmyqdaxhuixuthzrpwidtwgurmvktvowxwgvwmzfxtnhg
aowwxnhzievtxzrilxhtigfwgndgeiwtvoqilksaeoqiidwxwykxjzhgxnutiikpvgfiixvdgxhpxkdrmvktvz
hmx

With Key = zzzz Decrypted message:
mjpnxhgfhhewxhiidrvdsisxbhxjpxrmlhafullnfwmxqdawhuiwuthyrpwhdtwfurmuktvnwxwfvwmyfxtmhg
anwwxmhzidvtxyrilwhtiffwgmdgehwtvnqiljsaenqiicwxwxkxjyhgxmutihkpvffiiwvdgwhpxjdrmuktvy
hmx





At key = “code” we can see the plain text:

Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar cipher. It is an additive cipher with the key set to three. Each character in the plaintext is shifted three characters to create ciphertext.

7. Implement Hill cipher and One-time-pad cipher.

Programming Language used: Python

Hill Cipher

Code:

```
#dummy char will be Z i.e., its value 25

import numpy as np

import math

#get inverse modulo n of a number b

def getModInverse(n,b):

	r1 =n

	r2 = b

	t1 = 0

	t2 = 1

	while(r2>0):

		q = int(r1/r2)

		r = r1-q*r2

		r1 = r2

		r2 = r

		#inverse part

		t = t1- q*t2

		t1 = t2

		t2 = t

	#to maintain +ve inverse value and that it is in Zn

	if(t1<0):

		t1 = n +t1

	return t1



def Encrypt(p,k):

	#matrix multiplication:

	out = p@k

	out %= 26

	return out
```

```
def Decrypt(CT,K):
      D = round(np.linalg.det(K))
      D_inv = getModInverse(26,D%26)

      #as inverse of matrix is adj/det => inv*det = adj
      Adj_K = np.linalg.inv(K)*np.linalg.det(K)
      #using round because linalg library uses linear algebra and doesn't give exact
integer output but a very close decimal value
      Adj_K = np.round(Adj_K)
      Adj_K = Adj_K.astype(int)
      Adj_K %=  26

      K_inv = D_inv*Adj_K
      K_inv %=26
      out = CT@K_inv
      out %= 26

      return out



#decoding function to decode cipher text
def decoding(matrix,n):
      out = []
      for m in matrix:
            for i in m:
                  out.append(chr(i+97))
      return ''.join(out)



def main():
      #note the matrix condition must be met
      #if the number of elements is less than that required dummy variables will be
introduced
      #dummy char will be Z i.e., its value 25
```

```
	#if number of elements is greater than necessary matrix will need to be redefined
	#therefore must reneter key
	n = int(input("The Key matrix is a square matrix input n for nxn matrix: "))
	PT = input("Enter plain text: ")
	PT = PT.replace(" ", "") #removing spaces in plain text
	while(True):
		Key = input("Enter key string: ")
		Key = Key.lower()
		if(n**2<len(Key)):
			print("Key size is more than matrix re-enter key \n")
		else:
			break
	#converting key to list of required numbers:
	Kno = []
	#converting plain text to numbers
	for character in Key:
		number = ord(character)-97
		Kno.append(number)
	#adding dummy characters
	if(len(Kno)<n**2):
		m = len(Kno)
		for i in range(0,n**2-m):
			Kno.append(25)

	temp = []
	matrix_k = []
	j = 0
	for i in Kno:
		temp.append(i)
		j +=1
		if j % n == 0:
			matrix_k.append(temp)
			temp = []
			j = 0
```

```
# plaintext matrix can only have n columns
#for plain text conver it to a list append required dummy variables
#for char in list, inner loof for i in n
PT = PT.lower()
PTno = []
#converting plain text to numbers
for character in PT:
      number = ord(character)-97
      PTno.append(number)
if(len(PTno)%n!=0):
      m = len(PTno)%n
      for i in range(0,n-m):
            PTno.append(25)
print("PlainText in encoded into numbers: ",PTno)
print("Key in encoded into numbers: ",Kno)
temp = []
matrix_PT = []
j = 0
for i in PTno:
      temp.append(i)
      j +=1
      if j % n == 0:
            matrix_PT.append(temp)
            temp = []
            j = 0



k = np.array(matrix_k)
p = np.array(matrix_PT)

#determinant
D = round(np.linalg.det(k))%26
if(math.gcd(D,26)==1):
```

```
            CT = Encrypt(p,k)
            print("Cipher text: ",CT," ==> ",decoding(CT,n).upper())
            t = Decrypt(CT,k)
      #converting cipher text to letters
            print("Decrypted output",t,"==>",decoding(t,n))

      else:
            print("Given key's determinant doen't have multiplicative inverse in Zn26")

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\7 question>python hill.py

The Key matrix is a square matrix input n for nxn matrix: 2

Enter plain text: hi

Enter key string: hill

PlainText in encoded into numbers:  [7, 8]

Key in encoded into numbers:  [7, 8, 11, 11]

Cipher text:  [[ 7 14]]  ==>  HO

Decrypted output [[7 8]] ==> hi

C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\7 question>python hill.py

The Key matrix is a square matrix input n for nxn matrix: 3

Enter plain text: hi

Enter key string: crypto

PlainText in encoded into numbers:  [7, 8, 25]

Key in encoded into numbers:  [2, 17, 24, 15, 19, 14, 25, 25, 25]

Cipher text:  [[ 5 12 21]]  ==>  FMV

Decrypted output [[ 7  8 25]] ==> hiz
```

One Time Pad

Code:

```
#in one time pad encryption and decryption are done by same function
#we are using the ascii values to do encryptions and decryptions

import random
import string

def randKey(chars = string.ascii_uppercase + string.digits, N=10):
	return ''.join(random.choice(chars) for _ in range(N))


def EncAndDec(text,Key):

	text = text.lower()
	#converting string to list
	text1 =[]
	text1[:0]=text
	key1 = []
	key1[:0] = Key
	T = []
	for (c,k) in zip(text1,key1):
		c_num = ord(c)
		k_num = ord(k)
		#using bitwise xor operator on each ascii value of text and key
		T.append(c_num^k_num)
	String_out = [chr(o) for o in T]
	out = ''.join(String_out)
	return out.upper()


def main():
	PT  = input("Input plain Text: ")
	PT = PT.replace(" ", "") #removing spaces in plain text
	Key = randKey(N = len(PT))
```

```
    print("Randomly generated Key: ", Key)
    CT = EncAndDec(PT,Key)
    print("\n Cipher Text:",CT)
    print("Decrypted Plain Text : ", EncAndDec(CT,Key).lower())

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\7 question>python otPad.py
Input plain Text: hello
Randomly generated Key:  IBC7F

 Cipher Text: !'/[)
Decrypted Plain Text :  hello

C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\7 question>python otPad.py
Input plain Text: hello
Randomly generated Key:  XRMVA

 Cipher Text: 07!:.
Decrypted Plain Text :  hello

C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\7 question>python otPad.py
Input plain Text: hello
Randomly generated Key:  P533A

 Cipher Text: 8P__.
Decrypted Plain Text :  hello
```

8. Implement deterministic (Divisibility-test) and probabilistic Primality testing algorithms (Miller-Rabin).

Programming Language used: Python

Divisibility-test

Code:

```
import math

from sympy import symbols, Eq, solve


def Divisibility_test(n):

    r = 2

    while(r<=math.sqrt(n)):

        if(n%r==0):

            return "A composite number"

        r += 1

    return " a prime number "




def main():

    n = int(input("Input number to check if it is prime: "))

    print("Using divisibility test we find that the given number is: ",Divisibility_test(n))


main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\8 question>python Divisibility.py

Input number to check if it is prime: 17

Using divisibility test we find that the given number is:   a prime number


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\8 question>python Divisibility.py

Input number to check if it is prime: 32

Using divisibility test we find that the given number is:  A composite number
```

Miller-Rabin Test

Code:

```
#brute forcing to get k, generallly higher the k value more is the acurracy
#so we find the highest value possible

def get_mk(n):
    m = 1
    k = 1
    while (n-1)%2**k ==0 :
        m = (n-1)/pow(2,k)
        k += 1
    return (int(m),int(k-1))
#k-1 because there will be an extra increment from above loop
#also because we used multiplication
#and division operaton they are floats so we convert to int



def Miller_Rabin(n,a=2): #for prime test base is generally 2
    mk = get_mk(n)
    #opening tuple to get m and k
    m = mk[0]
    k = mk[1]
    T = pow(a,m)%n #a^m mod n
    #print(T) used for debugging
    if T == +1%n | T== -1%n :
        return "A Prime"
    for i in range (1,k):
        T = pow(T,2)%n
        #print(i,T,n) used for debugging
        #we are using (+ or -)1%n because inherently python
        #doesn't know if given number is equal to -1
        if T == 1%n: #is T = 1 mod n
            return "A Composite"
        if T == -1%n :#is T = -1 mod n
```

```
            return "A Prime"
    return "A Composite"




n = int(input("Input a number for Miller-Rabin test: "))


print("Given number is : ",Miller_Rabin(n))
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\8 question>python millerRabin.py

Input a number for Miller-Rabin test: 14

Given number is :  A Composite


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\8 question>python millerRabin.py

Input a number for Miller-Rabin test: 17

Given number is :  A Prime


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\8 question>python millerRabin.py

Input a number for Miller-Rabin test: 32

Given number is :  A Composite
```

9. Implement Chinese Remainder Theorem.

Programming Language used: Python

Code:

```
def getModInverse(n,b):
      r1 =n
      r2 = b
      t1 = 0
      t2 = 1
      while(r2>0):
            q = int(r1/r2)
            r = r1-q*r2
            r1 = r2
            r2 = r
            #inverse part
            t = t1- q*t2
            t1 = t2
            t2 = t
      #to maintain +ve inverse value and that it is in Zn
      if(t1<0):
            t1 = n +t1
      return t1

def crt(a_list,m_list):
      #initialising M
      M = 1
      for m in m_list:
            M *= m
      Mi_list= []
      for m in m_list:
            Mi_list.append(M/m)
      invMi_list = []
      for(m,Mi) in zip(m_list,Mi_list):
            invMi_list.append(getModInverse(m,Mi))

      x = 0#initialising solution
```

```
    list_len = len(m_list)
    for i in range(0,list_len):
        x += a_list[i]*Mi_list[i]*invMi_list[i]
    x = x%M

    return int(x) #not necessary but just to remove decimal
                #point which occurs as we used multiplication

def main():
    m_list = []
    a_list = []

    k = int(input("for equations of form - a modulo m \nPlease enter the number of equations: "))
    for i in range (0,k):
        a = int(input("Input a : "))
        m = int(input("Input its coressponding m  : "))
        a_list.append(a)
        m_list.append(m)
    print("using chinese remainder theorem, the value of x for which it is congruent to all given equations is:\n x = ",crt(a_list,m_list))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\9 question>python CRT.py

for equations of form - a modulo m

Please enter the number of equations: 3

Input a : 2

Input its coressponding m  : 3

Input a : 3

Input its coressponding m  : 5

Input a : 2

Input its coressponding m  : 7

using chinese remainder theorem, the value of x for which it is congruent to all given
equations is:

 x =  23
```

10. Implement RSA cryptosystem.

Programming Language used: Python

Code:

```
import random
import math

def Divisibility_test(n):
    r = 2
    while(r<= math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
    return True

#simple prime gen functiones mixed
def PrimeGen():
      while True:
            n = random.randint(1,100)
            fn = 2*n + 3
            gn = n**2 + 1
            hn  = 2**n + 1
            if(Divisibility_test(hn)):
                  return hn
            elif(Divisibility_test(gn)):
                  return gn
            elif(Divisibility_test(fn)):
                  return fn
            else:
                  print("prime not found repeat loop")

def getGCD(n,b):
      r1 =n
      r2 = b
      while(r2>0):
```

```
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
    return r1
def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1
# #test :
# n = 159197
# totient_n = totient(n)
# print(totient_n)
#print(random.randrange(100, 1000, 3))

def RSA_KeyGen():
    #add generating function for p and q
    p = PrimeGen()
    while(True):
        q = PrimeGen()
        if p!=q :
```

```
                break
        #make sure p!=q

        n = p*q
        #as both are primes the value of totient(n) is given by this equation
        totient_n = (p-1)*(q-1)

        cond = True
        while(cond):
                e = random.randrange(1, totient_n)
                if(getGCD(totient_n,e)==1):
                        cond = False

        d = getModInverse(totient_n,e)

        Public_Key = (e,n)
        Private_key = d
        return (Public_Key,Private_key)

def RSA_encryption(P,e,n):
        C = pow(P,e)%n
        return C
def RSA_Decryption(C,d,n):
        P = pow(C,d,n)
        return P

def main():
        key_pair = RSA_KeyGen()
        Public_Key = key_pair[0]
        Private_key = key_pair[1]
        print("Public Keys generated: Public key e = ",Public_Key[0]," Public_Key n = ",
Public_Key[1])
        print("Private key generated: Private key d = ",Private_key)
        P = int(input("Enter Plain text in Zn : "))
        C = RSA_encryption(P,Public_Key[0],Public_Key[1])
```

```
    print("RSA encrypted Cipher text: ",C,"\n")
    print("RSA Decrypted Plain text: ", RSA_Decryption(C,Private_key,Public_Key[1]))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\10 question>python RSA.py

Public Keys generated: Public key e =  22585  Public_Key n =  26989

Private key generated: Private key d =  6921

Enter Plain text in Zn : 123

RSA encrypted Cipher text:  14595


RSA Decrypted Plain text:  123


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\10 question>python RSA.py

prime not found repeat loop

Public Keys generated: Public key e =  1113813  Public_Key n =  1581823

Private key generated: Private key d =  739149

Enter Plain text in Zn : 3421

RSA encrypted Cipher text:  995843


RSA Decrypted Plain text:  3421
```

11. Implement Rabin cryptosystem.

Programming Language used: Python

Code:

```
import random
import math

#gcd to check if given Plain text is valid
def getGCD(n,b):
    r1 =n
    r2 = b
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
    return r1



def Divisibility_test(n):
    r = 2
    while(r<=math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
    return True



#mersenne
def PrimeGen():
    # can use mersenne primes too
    # prime_list = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
    # while True:
    #     p = prime_list[random.randint(1,len(prime_list)-1)]
    #     Mi = pow(2,p)-1
    #     if(Divisibility_test(Mi)):
```

```
    #          return Mi
    while True:
        n = random.randint(1,100)
        fn = 2*n + 3
        gn = n**2 + 1
        hn  = 2**n + 1
        if(Divisibility_test(hn)):
            return hn
        elif(Divisibility_test(gn)):
            return gn
        elif(Divisibility_test(fn)):
            return fn
        else:
            print("prime not found repeat loop")



def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1
```

```
#modified CRT for Rabin system which has only two equations
def crt(a,b,p,q):
    #initialising M
    M = p*q
    M1 = M/p
    M2 = M/q

    inv_M1 = getModInverse(p,M1)
    inv_M2 = getModInverse(q,M2)
    x = (a*M1*inv_M1 + b*M2*inv_M2)%M

    return int(x) #not necessary but just to remove decimal point which occurs as we 
used multiplication

#check prime if of form 4k + 3
def prime_check(n):
    if (n-3)%4 == 0:
        return True
    return False



def Rabin_KeyGen():
    while True:
        p = PrimeGen()
        if prime_check(p):
            break

    while True:
        q = PrimeGen() #use generator here
        if prime_check(q) and p!=q:
            break

    n = p*q
```

```
    Public_key = n
    Private_key = (p,q)

    return (Public_key, Private_key)



def Rabin_Encryption(n,P): #n is public key P is from Zn*
    C = pow(P,2)%n
    return C



def Rabin_Decryption(p,q,C):
    a1 = (C**((p+1)//4))%p
    a2 = (-(C**((p+1)//4)))%p
    b1 = (C**((q+1)//4))%q
    b2 = (-(C**((q+1)//4)))%q
    #using crt:
    P1 = crt(a1,b1,p,q)
    P2 = crt(a1,b2,p,q)
    P3 = crt(a2,b1,p,q)
    P4 = crt(a2,b2,p,q)
    return (P1,P2,P3,P4)


Keys = Rabin_KeyGen()
Public_key = Keys[0]
Private_key = Keys[1]
print("Public Key generated: n = ",Public_key)
print("Private keys generated: p = ",Private_key[0], " q = ",Private_key[1])
#encryption call by alice
```

```
while True:
    PT  = int(input("Input Plaintext which is a part of Zn* where n is the public key given:  "))
    if getGCD(Public_key,PT) == 1:
        break
    else:
        print("Invalid input, please try again")

CT = Rabin_Encryption(Public_key,PT)
print("Ciphert text: ",CT)

#decryption call
p = Private_key[0]
q = Private_key[1]
print("Decryption of Ciphertext generates 4 different text and the plain text is one of them:\n ",Rabin_Decryption(p,q,CT))
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\11 Question>python Rabin.py
Public Key generated: n =  217
Private keys generated: p =  7  q =  31
Input Plaintext which is a part of Zn* where n is the public key given:  24
Ciphert text:  142
Decryption of Ciphertext generates 4 different text and the plain text is one of them:
  (193, 179, 38, 24)
```

Output using Mersenne prime generator which is commented in code:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\11 Question>python Rabin.py
Public Key generated: n =  1073602561
Private keys generated: p =  8191  q =  131071
Input Plaintext which is a part of Zn* where n is the public key given:  123
Ciphert text:  15129
Decryption of Ciphertext generates 4 different text and the plain text is one of them:
  (427291583, 1073602438, 123, 646310978)
```

12. Implement ElGamal cryptosystem.

Programming Language used: Python

Code:

```
import random
import math
from math import gcd

def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1


def Divisibility_test(n):
    r = 2
    while(r<=math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
```

```
    return True


#Fermats prime gen and other functiones mixed
def PrimeGen():
      while True:
            n = random.randint(1,100)
            fn = 2*n + 3
            gn = n**2 + 1
            hn  = 2**n + 1
            if(Divisibility_test(hn)):
                  return hn
            elif(Divisibility_test(gn)):
                  return gn
            elif(Divisibility_test(fn)):
                  return fn
            else:
                  print("prime not found repeat loop")



def primRoots(modulo):
    required_set = {num for num in range(1, modulo) if gcd(num, modulo) }
    return [g for g in range(1, modulo) if required_set == {pow(g, powers, modulo)
            for powers in range(1, modulo)}]


def Elgamal_KeyGen():
      p = PrimeGen() #use generator here


      prim_root_list = []
      while(True):
            prim_root_list = primRoots(p)
            if prim_root_list != [] :
                  break
            else:
                  p = PrimeGen()
```

```
	d = random.randint(1,p-2) #any number from 1 to p-2 as in Zp* all values from 1 to p-1 are present and it is a cyclic group


	e1 = random.choice(prim_root_list)


	e2 = pow(e1,d)%p


	PublicKey = (e1,e2,p)
	PrivateKey = d

	return (PublicKey,PrivateKey)

def Elgamal_Encryption(e1,e2,p,P):
	#as Zp* forms a group by itself excluding p :
	#we can choose a random integer from zp*
	r = random.randint(1,p-1)
	print("random number chosen, r = ", r)
	C1 = pow(e1,r)%p
	C2 = (P*pow(e2,r))%p

	return (C1,C2)

def Elgamal_DEcryption(d,p,C1,C2):
	P = (C2*(getModInverse(p,pow(C1,d)%p)))%p
	return P

def main():
	key = Elgamal_KeyGen()
	PublicKey = key[0]
	PrivateKey = key[1]
	print("Public keys e1,e2,p are : ", PublicKey)
	print("Private key d: ", PrivateKey)
	P = int(input("Input Plain Text: "))
	e1 = PublicKey[0]
	e2 = PublicKey[1]
```

```
    p = PublicKey[2]
    CT = Elgamal_Encryption(e1,e2,p,P)
    print("Encrypted Message(Cipher text):  ",CT)
    print("Decrypted Message: ", Elgamal_DEcryption(PrivateKey,p,CT[0],CT[1]))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\12 question>python elgamal.py
Public keys e1,e2,p are :  (564, 561, 577)
Private key d:  272
Input Plain Text: 71
random number chosen, r =  431
Encrypted Message(Cipher text):   (135, 248)
Decrypted Message:  71

C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\12 question>python elgamal.py
Public keys e1,e2,p are :  (2327, 2367, 4357)
Private key d:  2504
Input Plain Text: 234
random number chosen, r =  4011
Encrypted Message(Cipher text):   (3553, 43)
Decrypted Message:  234
```

13. Implement RSA Digital Signature Scheme.

Programming Language used: Python

Code:

```
import random
import math


def Divisibility_test(n):
    r = 2
    while(r<=math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
    return True


#simple prime gen functiones mixed
def PrimeGen():
      while True:
            n = random.randint(1,100)
            fn = 2*n + 3
            gn = n**2 + 1
            hn  = 2**n + 1
            if(Divisibility_test(hn)):
                  return hn
            elif(Divisibility_test(gn)):
                  return gn
            elif(Divisibility_test(fn)):
                  return fn
            else:
                  print("prime not found repeat loop")



def getModInverse(n,b):
      r1 =n
      r2 = b
      t1 = 0
```

```
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1

#Signing
def Private_key_encryption(M,d,n):
    #S is signature
    S = pow(M,d)%n
    return S




def Signature_Decryption(S,e,n):
    M = pow(S,e)%n
    return M

#verifying
def Signature_confirmation(M1,M2):
    if M1 == M2:
        return True
    else:
        return False
```

```
def RSA_KeyGen():
      #add generating function for p and q
      p = PrimeGen()
      while(True):
            q = PrimeGen()
            #make sure p!=q
            if p!=q :
                  break
      n = p*q
      #as both are primes the value of totient(n) is given by this equation
      totient_n = (p-1)*(q-1)

      cond = True
      while(cond):
            e = random.randrange(2, totient_n-1)
            if math.gcd(e,totient_n)==1:
                  cond = False

      d = getModInverse(totient_n,e)

      Public_Key = (e,n)
      Private_key = d
      return (Public_Key,Private_key)




def main():
      #key generation
      key_pair = RSA_KeyGen()
      Public_Key = key_pair[0]
      Private_key = key_pair[1]
      print("\nPublic key e = ",Public_Key[0]," Public_Key n = ", Public_Key[1])
      print("Private key generated: Private key d = ",Private_key)
```

```
    M = int(input("Enter Message in Zn : "))


    #Signing
    S = Private_key_encryption(M,Private_key,Public_Key[1])


    print("\nRSA Private key encrypted Signature is: ",S)
    print("message transmitted: ", (M,S))


    #verification
    M1 = Signature_Decryption(S,Public_Key[0],Public_Key[1])
    print("\nDecrypting Signature with public key gives: ", M1)
    if Signature_confirmation(M,M1):
        print("Given Message and Message decryped from signature are same Digital signature,origin verified, message not tampered with\n Message ACCEPTED")
    else:
        print("as signature is not giving original message when decrypted, user unverified\n Message Rejected")



main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\13 question>python rsa_digitalSign.py


Public key e =  14461  Public_Key n =  31459

Private key generated: Private key d =  18517

Enter Message in Zn : 1321


RSA Private key encrypted Signature is:  8888

message transmitted:  (1321, 8888)


Decrypting Signature with public key gives:  1321

Given Message and Message decryped from signature are same Digital signature,origin verified, message not tampered with

 Message ACCEPTED


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\13 question>python rsa_digitalSign.py

prime not found repeat loop


Public key e =  641445  Public_Key n =  914659

Private key generated: Private key d =  384709

Enter Message in Zn : 5312


RSA Private key encrypted Signature is:  475967

message transmitted:  (5312, 475967)


Decrypting Signature with public key gives:  5312

Given Message and Message decryped from signature are same Digital signature,origin verified, message not tampered with

 Message ACCEPTED
```

14. Implement ElGamal Digital Signature Scheme.

Programming Language used: Python

Code:

```
import random
import math
from math import gcd

def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1



def Divisibility_test(n):
    r = 2
    while(r<=math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
    return True
```

```
#Fermats prime gen and other functiones mixed
def PrimeGen():
      while True:
            n = random.randint(1,100)
            fn = 2*n + 3
            gn = n**2 + 1
            hn  = 2**n + 1
            if(Divisibility_test(hn)):
                  return hn
            elif(Divisibility_test(gn)):
                  return gn
            elif(Divisibility_test(fn)):
                  return fn
            else:
                  print("prime not found repeat loop")


def primRoots(modulo):
    required_set = {num for num in range(1, modulo) if gcd(num, modulo) }
    return [g for g in range(1, modulo) if required_set == {pow(g, powers, modulo)
            for powers in range(1, modulo)}]


def Elgamal_KeyGen():
      p = PrimeGen() #use generator here


      prim_root_list = []
      while(True):
            prim_root_list = primRoots(p)
            if prim_root_list != [] :
                  break
            else:
                  p = PrimeGen()


      #any number from 1 to p-2 as in Zp* all values from 1 to p-1 are present and it
is a cyclic group
      d = random.randint(1,p-2)
```

```python
    e1 = random.choice(prim_root_list)
    e2 = (e1**d)%p

    PublicKey = (e1,e2,p)
    PrivateKey = d

    return (PublicKey,PrivateKey)

def Elgamal_Signature(e1,p,d,M):
    #as Zp* forms a group by itself excluding p :
    #we can choose a random integer(secret) from zp*
    while(True):
        r = random.randint(1,p-1)
        if(math.gcd(r,p-1)==1):
            break
    #r = 107 #used for testing
    #print("r = ", r)
    S1 = (e1**r)%p
    r1 = getModInverse(p-1,r)
    temp = (M-(d*S1))%(p-1)
    S2 = ((M-(d*S1))*r1)%(p-1)

    return (S1,S2)

def Elgamal_Verifiying(S1,S2,M,e1,e2,p):
    V1 = (pow(e2,S1)*pow(S1,S2))%p
    V2 = (e1**M)%p
    if(V1 == V2):
        return " Verified"
    return "Not Verified"

def main():
    key = Elgamal_KeyGen()
    PublicKey = key[0]
    PrivateKey = key[1]
```

```
    print("Sender:")
    print("Public keys generated e1,e2,p are : ", PublicKey)
    print("Private key generated d: ", PrivateKey)
    M = int(input("Input Message: "))
    e1 = PublicKey[0]
    e2 = PublicKey[1]
    p = PublicKey[2]
    print("\nReciever:")
    S = Elgamal_Signature(e1,p,PrivateKey,M)
    print("Signatures and Message:  ",S,",  ", M)
    print("Verification: ", Elgamal_Verifiying(S[0],S[1],M,e1,e2,p))

main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\14 question>python elgamal_sign.py

Sender:

Public keys generated e1,e2,p are :  (115, 110, 139)

Private key generated d:  137

Input Message: 21


Reciever:

Signatures and Message:   (70, 67) ,   21

Verification:   Verified


C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\14 question>python elgamal_sign.py

Sender:

Public keys generated e1,e2,p are :  (85, 84, 127)

Private key generated d:  118

Input Message: 321


Reciever:

Signatures and Message:   (3, 33) ,   321

Verification:   Verified
```

15. Implement Diffie-Hellman Key-Exchange Algorithm.

Programming Language used: Python

Code:

```
import random
import math
from math import gcd

def getModInverse(n,b):
    r1 =n
    r2 = b
    t1 = 0
    t2 = 1
    while(r2>0):
        q = int(r1/r2)
        r = r1-q*r2
        r1 = r2
        r2 = r
        #inverse part
        t = t1- q*t2
        t1 = t2
        t2 = t
    #to maintain +ve inverse value and that it is in Zn
    if(t1<0):
        t1 = n +t1
    return t1



def Divisibility_test(n):
    r = 2
    while(r<=math.sqrt(n)):
        if(n%r==0):
            return False
        r += 1
    return True
```

```
#Fermats prime gen and other functiones mixed
def PrimeGen():
    while True:
        n = random.randint(1,100)
        fn = 2*n + 3
        gn = n**2 + 1
        hn  = 2**n + 1
        if(Divisibility_test(hn)):
            return hn
        elif(Divisibility_test(gn)):
            return gn
        elif(Divisibility_test(fn)):
            return fn
        else:
            print("prime not found repeat loop")




#function to generate primitive roots of given prime number
def primRoots(modulo):
    required_set = {num for num in range(1, modulo) if gcd(num, modulo) }
    return [g for g in range(1, modulo) if required_set == {pow(g, powers, modulo)
            for powers in range(1, modulo)}]


def Calculate_R(n,g,p):
    return pow(g,n)%p



def Calculate_SKey(R,n,p):
    return pow(R,n)%p
```

```
def value_K(g,x,y,p):
    return pow(g,x*y)%p



def main():


    p = PrimeGen();
    prim_root_list = []
    while(True):
        prim_root_list = primRoots(p)
        if prim_root_list != [] :
            break
        else:
            p = PrimeGen()

    g = random.choice(prim_root_list)
    print("Public: value of p = ",p," and value of g = ",g)



    Alice_x = random.randint(1,p-1)
    print("Random number x chosen by Alice: ", Alice_x)



    Bob_y = random.randint(1,p-1)
    print("Random number y chosen by Bob: ", Bob_y)



    R1 = Calculate_R(Alice_x, g, p)
    R2 = Calculate_R(Bob_y, g, p)

    print("Alice sends ", R1, "to Bob.")
    print("Bob sends ", R2, "to Alice.")

    #Alice calculates symmetric key
    Alice_key = Calculate_SKey(R2,Alice_x,p)
```

```
    print("Symmetric key Alice generated by using Bob's number: ", Alice_key)

    #Bob calculates symmetric key
    Bob_key = Calculate_SKey(R1,Bob_y,p)
    print("Symmetric key Bob generated by using Alice's number: ", Bob_key)

    K = value_K(g,Alice_x,Bob_y,p)

    #check if all the key vlues are same
    if K == Alice_key and K == Bob_key:
        print("Diffie-Hellman Key Agreement -- Successful")
    else:

        print("Diffie-Hellman Key Agreement -- Unsuccessful")



main()
```

Output:

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical
Assignment\15 question>python diffie_hellman.py
Public: value of p =  173  and value of g =  102
Random number x chosen by Alice:  34
Random number y chosen by Bob:  14
Alice sends  73 to Bob.
Bob sends  89 to Alice.
Symmetric key Alice generated by using Bob's number:  169
Symmetric key Bob generated by using Alice's number:  169
Diffie-Hellman Key Agreement -- Successful
```

```
C:\Users\sonpi\OneDrive\Documents\20BCE7305\21-22 fall\cryptography\Practical Assignment\15 question>python diffie_hellman.py

Public: value of p =  5477  and value of g =  2377

Random number x chosen by Alice:  5067

Random number y chosen by Bob:  4882

Alice sends  4016 to Bob.

Bob sends  543 to Alice.

Symmetric key Alice generated by using Bob's number:  3039

Symmetric key Bob generated by using Alice's number:  3039

Diffie-Hellman Key Agreement -- Successful
```