Message Encryption & Decryption using Discrete Mathematics

Cluster Innovation Centre

University of Delhi

February 2023

Month Long Project submitted for

Discrete Mathematics

Acknowledgment

We would like to express our special thanks of gratitude to our mentor **Dr Nirmal Yadav** for guidance and supervision as well as providing necessary information regarding the project. It would be a great pleasure for us to present our 1st semester-month long project on Discrete Mathematics

Certificate of Originality

The work embodied in this report entitled “Message Encryption & Decryption using Discrete Mathematics” has been carried out by Saswat Susmoy, Aryan Sharma, Mahima Agarwal, Avinav Verma for the paper “Discrete Mathematics”. I declare that the work and language included in this project report is free from any kind of plagiarism.

Dr Nirmal Yadav
Assistant Professor
Department of Mathematics
Cluster Innovation Centre
University of Delhi

Certificate of Completion

This is to certify that the following persons: Saswat Susmoy Sahoo, Avinav Verma, Aryan Sharma, Mahima Agarwal have completed this Month Long Project for Discrete Mathematics under my guidance and supervision as per the contentment of the requirements of the first semester in the course BTech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi.

Dr Nirmal Yadav
Assistant Professor
Department of Mathematics
Cluster Innovation Centre
University of Delhi

Index

Abstract

Message Encryption and Decryption using Discrete Mathematics

by

|| Aryan Sharma || Saswat Susmoy Sahoo || Avinav Verma || Mahima Agarwal ||

Cluster Innovation Centre, 2023

In this project, We have created a model for encrypting text using Python language with the help of Combinatorics and other topics of Discrete Mathematics. We have used various mathematical concepts like Matrices, Determinants, Permutations, Functions, Relations and Modular Arithmetic in the following project. Apart from analysing all the algorithms and ciphers we have tried to implement the same through Python with some of our innovation that makes it less vulnerable and our program more efficient and applicable in real life.

Introduction

The main goal of cryptography is to make it possible for two people to communicate over an insecure channel in a way that no outsider can understand what is being communicated. It involves the use of mathematical algorithms to transform the original message, called plaintext, into an encoded form, known as ciphertext. Only those with the appropriate decryption key can convert the ciphertext back into its original form and access the information contained within it. This channel could be a computer network or a telephone line. In this project we have used various ciphers innovated to our needs to encrypt text files & images.

Why Cryptography is Important?

- Cryptography protects the confidentiality of information
- It ensures the integrity of your data
- It assures that the sender or receiver is the right one
- Both sender and receiver are held accountable through non-repudiation
- Cryptography also ensures the availability of data
- Uphold information security with powerful cryptography strategies

I.1 Background and Context

Our motive was to build a cryptographic model for encrypting files easily whether it is a text file or an image file. Our model uses multiple pre existing algorithms innovated to cater to our needs which randomizes order of the algorithms to increase the security of our model.

I.2 Scope and Objectives

In this project we have made a user-friendly interface for general audience to use the services of encryptions in real life to secure their personal data. Through our program we aim to aid people to secure their personal documents, Images, Code files etc from intruders and malwares.

I.3 Achievements

We have successfully encrypted text and image files alike to provide target audience with a quick program to secure their data. We have used old and outdated ciphers, innovated them to provide a better and more secure encryption. We have used the concepts of salting of codes to provide an additional layer of security to the ciphers.

II.1 Problem Statement

Ciphers like Caesar Shift, Permutation, Substitution, Hill etc were invented some hundreds of years ago. Although being relevant for most part of Cryptography's history, now all these ciphers are outdated and easy to decode. Vigenere Cipher which even today is still unbreakable on pen and paper has lost its value due to introduction of technology.

We took an attempt to use these ciphers to create a new program which would enhance the security level of the encryption.

Pre-Requisites

Theoretical

- Modular Arithmetic
- Combinatorics
- Determinants and Matrices
- Basic Knowledge of Modules in Python

Hardware/Software

- IDE to run Python
- Python 3 or above
- OS Module

II.2 Methodology

We have used following six different algorithms and have made few changes in each of them to increase their security and efficiency which will be listed below:-

(**11.2.1**)**Shift Cipher**:- Shift cipher is a monoalphabetic substitution cipher. It's also sometimes referred to as Caesar Shift as it was used by Julio Caesar himself.

Encryption: When a plaintext is provided to this algorithm, we use the pre-defined ASCII values to represent every character. A random key is generated by the system from the ASCII table. To generate the encrypted text the original plain text after being encoded with the ASCII counterparts is shifted using an algorithm where the random key is added to the encoded plaintext and then operated with the modulus function as shown below:

$$\mathbf{E(x) = [f(x) + k] \bmod 127}$$

where, E(x) is encrypted text

f(x) is the encoded plaintext

k is the generated random key

Decrypting: When a ciphertext is provided to this algorithm, we use the previously generated key (the one used to encrypt the plaintext) and reverse the encryption process. We start by backward shifting (subtracting) the ciphertext with the key and then applying the modulus function with 127 as shown below:

$$\mathbf{D(x) = [E(x) - k] \bmod 127}$$

where, D(x) is decrypted plain text

Note: As the ASCII Table from 1 to 32 contains characters like space, enter etc, we have made an attempt to salt the code by adding 33 to the plaintext and then applied the Encryption technique.

(11.2.2)Hill Cipher: It is an encryption technique that uses linear equations to convert a plaintext into ciphertext.
The plaintext is represented as a matrix , and the encryption process involves performing matrix multiplication with a key matrix to produce the ciphertext.

Encrypt: In Hill cipher algo every alphabet is represented by its decimal value according to ASCII.
To encrypt the text using hill cipher, we need to perform the following operation:

$$E(K, P) = (K * P) \bmod 127$$

where, K is the key matrix
P is plain text matrix
Matrix multiplication of K and P generates the encrypted ciphertext.

Decrypt: To decrypt the text using hill cipher, we need to perform the following operation:

$$D(K, C) = (K^{-1} * C) \bmod 127.$$

(11.2.3) Vignere Cipher: What is today known as the Vigenère Cipher was actually first described by Giovan Battista Bellaso in his 1553 book La cifra del. Sig. Giovan Battista Bellaso

Encryption: In Vignere Cipher we generate a list of random numbers and the length of list is also random. This list of numbers is called the random number list. And every alphabet of our text is represented by its decimal value according to ASCII. And thus we get our message list.
Then we simply add the random number list and the message list . To get a encrypted list containing numbers and then each number is converted to its character value according to ASCII. In this way we get our encrypted characters.

Decryption:
In order to decrypt the encrypted matrix , we first convert the encrypted characters list into encrypted numbers list.
Then we subtract the random number list from this encrypted number list to get our message list containing numbers which is then converted into characters to get our final message text.

(11.2.4) Substitution Cipher:

Encryption: In this algorithm we have two Lists.
List 1 which stores the integer values from 32 to 126.
And List 2 of length equal to list 1 which stores random integers from 32 to 127 .

When a plain text is provided to this algorithm, we use pre-defined ASCII values to represent every character. We check the index from list 1 at which this ASCII value is present. Then we checks what value is present at that index in list 2 and stores them in a new list. We get our encrypted text by using the character values of the ASCII values in this new list.

Decryption: In order to decrypt the encrypted text , we use previously generated List2 and the list 1. We first convert the characters to its corresponding ASCII values. Then we check the index of each of these values in List 2 and checks the values at these indexes in List 1.
We then convert those ASCII values to get Decrypted characters.

(11.2.5) Permutation Cipher:

In permutation cipher ,we generate a random list of integers of length equal to the length of our plain text containing integers of range of (0 to length).

In this way we get our list according to which we will rearrange our characters. We find the index at which every number is present and then replaces the character at that number with character at that index.

For example:
If our plain text is : HELLO
And suppose the random list generated is : [1,3,2,0,4]

Then we first check the index of 0 which is 3, so we keep the fourth character (3rd index) of HELLO at first place in our encrypted character. Then we check the index of 1 which is 0, so we keep the first character[0 index] at second place.

In this way we get our encrypted text: LHLEO

To decrypt the above encrypted text , we can just reverse the process.

(11.2.6) Affine cipher :- The affine cipher is a type of monoalphabetic substitution cipher. it is modified version of shift cipher where there are two parameters as compared to one in shift cipher.

Encryption :- When a plaintext is provided to this algorithm , each character is first converted to their respective ASCII values. After which following function is applied to every value to get the cipher text.

$$E(k)=(a*p+k) \bmod 127$$

where, a represents affine key

p= plaintext ascii value

k= shift key

a can be any value from 2 to 126 as for a=1 it becomes shift cipher k can be any value from 1 to 126 A condition for choosing a is it should be co-prime with the number we are taking the modulus of i.e, in this example it is 127 which itself is a prime number. so every number from 2 to 126 is coprime with 127. so we can take any integer from 2 to 126. The numerical value represents the corresponding ASCII character.

Decryption :- for decrypting the cipher text we use the following formula :

$$1/a(y-k) \bmod 127$$

where, 1/a is the multiplicative inverse of a in modular function which is not same in normal arithmetic. so we calculate it by adding 1 to multiples of 127 and checking if a divides that. number and gives us integer quotient. this quotient is then 1/a.

(11.3) Randomization of Ciphers :- We have written six algorithms for encryption which we apply randomly one after the other to get the encrypted text .
The random order in which the six algorithms are applied is stored in a file .
Initially an algorithm selected randomly is applied on the plain text to get the encrypted text, and then next algorithm is applied on this encrypted text , similarly all the six algorithms are applied one by one randomly to get the final encrypted text.

To decrypt the message we simply apply the decrypting algorithms in reverse order i.e., the algorithm which was used first while encryption is used at last in decryption and the algorithm which was used at last in encryption is used first while decryption .
In this way we get our encrypted and decrypted characters by randomising the algorithms which makes our method more secure.

(II.2.I) Shift Cipher Encryption Implementation Source Code

```python

def encryption(message):
    import random
    k= random.randint(0,1000000)
    with open('Keys/ShiftCipherKeysUsed.bin','wb') as file:
        file.write(str.encode((str(k))))

    prob = []
    value_y = []
    for i in range(len(message)):
        p = ord(message[i])
        y = (p+k)%127

        if y < 33:
            y = y + 33
            prob.append(i)

        value_y.append((y))
    # print(value_y)



    string = ""
    for y in value_y:
        string = string +(chr(y))




    file = open('Keys/ShiftCipherProbKeys.bin','wb')
    for p in prob:
        file.write(str.encode((f"{str(p)}\n")))


    file = open('Keys/ShiftCipherValue_y.bin','wb')
    for y in value_y:
        file.write(str.encode((f"{str(y)}\n")))
    file.close()

    return string

```

Shift Cipher Decryption Implementation Source Code

```
def decryption(Encrypted_message):
    # with open('SubstitutionCipher/DecryptedText.txt','r') as file:
    #     Encrypted_message = file.read()
    with open('Keys/ShiftCipherKeysUsed.bin','rb') as file:
        KeyUsed = file.read()


    Key = int(KeyUsed)


    problist= []
    with open('Keys/ShiftCipherProbKeys.bin','rb') as file:
        for line in file:
            curr_place = line[:-1]
            problist.append(int(curr_place))

    value_y = []
    with open('Keys/ShiftCipherValue_y.bin','rb') as file:
        for line in file:
            curr_place = line[:-1]
            value_y.append(int(curr_place))

    # print(problist)
    # print(Key)
    # print(value_y)

    DecryptedText=[]
    for i in range(len(Encrypted_message)):
        if i in problist:
            value_j = value_y[i] - 33
            x = (value_j - Key)%127
            DecryptedText.append(chr(x))
        else:
            x = (value_y[i] - Key)%127
            DecryptedText.append(chr(x))


    string = ""
    for d in DecryptedText:
        string = string +(d)
    return string
```

Hill Cipher Encryption Implementation Source Code

```python
import numpy
import random

def a_inverse(a):
    c=1
    if a==1:
        return 1
    else:
        for i in range(127*127):
            c=c+127
            f=c%a
            if f==0:
                ans=c/a
                ans=int(ans)
                break
    return ans

def encryption(message):
    message1=message
    #This will store the original mesaage
    with open('Keys/HillCiphermessage.bin','wb') as file:
        file.write(str.encode((message1)))


    #Checks if the message is of odd length and adds x at the end for odd length message
    if len(message1)%2 ==1:
        message = message1 +"x"
    else:
        message = message1

    #Stores length of message in file
    with open('Keys/HillCipherlength.bin','wb') as file:
        file.write(str.encode((str(len(message)) ))  )


    #Generates a random key matrix  ensuring that its determinant should not be zero
    while True:
        Key_matrix = numpy.matrix([[random.randint(1,127),random.randint(1,127)],[random.randint(1,127),random.randint(1,127)]])
        determinantKeyMatrix = Key_matrix[0,0]*Key_matrix[1,1] -Key_matrix[0,1]*Key_matrix[1,0]
        if determinantKeyMatrix==0:
            continue
        else:
            break


    #Stores key matrix in a file
    file = open('Keys/HillCipherKeyMatrix.bin','wb')
    for i in range(0,2):
        for j in range(0,2):
            file.write(str.encode((f"{str(Key_matrix[i,j])}\n")))
    file.close()

    #Generates  cipher matrix  for every two characters and adds it in the cipher matrices list
    cipher_matrices =[]
    i =0
    while i<=(len(message)-2):
        message_matrix = numpy.array([[ord(message[i]),ord(message[i+1])]])
        # print(f"Message Matrix:{message_matrix}")
        i=i+2
        cipher_matrix = (message_matrix*Key_matrix)%127
        # print(f"Cipher matrix:{cipher_matrix}")

        cipher_matrices.append(cipher_matrix)
    # print(cipher_matrices)
    # print(len(cipher_matrices))


    #Generates cipherkeys list from cipher matrices , cipher keys list stores ord value of each encrypted character
    CipherKeys =[]
    for i in range(len(cipher_matrices)):
        for j in range(0,1):
            for k in range(0,2):
                CipherKeys.append(int(cipher_matrices[i][j,k]))
    # print(CipherKeys)
    # print(len(CipherKeys))

    #Convert the ord values of encrypted charcter into its corresponding character values and store in string to return it.
    string =""
    probKeys=[]
    for i in range(len(CipherKeys)):
        if CipherKeys[i]<33:
            CipherKeys[i]=CipherKeys[i]+33
            probKeys.append(i)
            string = string +(chr(CipherKeys[i]))
        else:
            string = string +(chr(CipherKeys[i]))


    #This store the prob keys in a file
    file = open('Keys/HillCipherProbKeys.bin','wb')
    for p in probKeys:
        file.write(str.encode((f"{str(p)}\n")))


    return string
```

(II.2.2) Hill Cipher Decryption Implementation Source Code

```python
def a_inverse(a):
    c=1
    if a==1:
        return 1
    else:
        for i in range(127*127):
            c=c+127
            f=c%a
            if f==0:
                ans=c/a
                ans=int(ans)
                break
    return ans


def decryption(message):


    #Reads the keymatrix and stores it in Key_matrix
    with open('Keys/HillCipherKeyMatrix.bin','rb') as file:
        a = file.readline()
        b = file.readline()
        c = file.readline()
        d = file.readline()
    Key_matrix = numpy.matrix([[int(a),int(b)],[int(c),int(d)]])
    # print(Key_matrix)

    #reads length of message stored in file
    with open("Keys/HillCipherlength.bin",'rb') as file:
        length = int((int(file.read())))

    #reads problist from file
    problist= []
    file = open('Keys/HillCipherProbKeys.bin','rb')
    for line in file:
        problist.append(int(line))


    #reads the cipher matrices and stores it in cipher_matrices
    cipher_matrices =[]
    value=[]
    # file=open('Encrypted.txt','r')
    # message = file.read()
    for i in range(length):
        if i in problist:
            value_j =(ord(message[i]))-33
            value.append(value_j)
        else:
            value.append(ord(message[i]))
    file.close()
    i=0
    while i<length :
        cipher_matrix = numpy.matrix([value[i],value[i+1]])
        cipher_matrices.append(cipher_matrix)
        i=i+2
    # print(cipher_matrices)




    determinantKeyMatrix = Key_matrix[0,0]*Key_matrix[1,1] -Key_matrix[0,1]*Key_matrix[1,0]
    # print(determinantKeyMatrix)


    adjointOfKeyMatrix =  (numpy.matrix([[Key_matrix[1,1],(-1*Key_matrix[0,1])],[(-1*Key_matrix[1,0]),Key_matrix[0,0]]]))
    # print(adjointOfKeyMatrix)

    InverseOfDeterminant = a_inverse(determinantKeyMatrix)
    # print(InverseOfDeterminant)


    FinalInverseMatrix = (InverseOfDeterminant * adjointOfKeyMatrix)
    # print(f"Inverse Matrix : {FinalInverseMatrix}")
    file = open("Keys/HillCiphermessage.bin","rb")
    message1 = file.read()
    file.close()

    if len(message)==len(message1):
        List=[]
        for i in range(len(cipher_matrices)):
            Decrypted_matrix = (cipher_matrices[i] * FinalInverseMatrix)%127
        #   print(f"DEcryptedMatrix : {Decrypted_matrix}")
            DecryptedCharactersMatrix = numpy.matrix([[chr(Decrypted_matrix[0,0]),chr(Decrypted_matrix[0,1])]])

        #   print(DecryptedCharactersMatrix)
            List.append(DecryptedCharactersMatrix[0,0])
            List.append(DecryptedCharactersMatrix[0,1])
        string = ""
        for i in range(len(List)):
            string = string +(List[i])

        return string
    else:

        List=[]
        for i in range(len(cipher_matrices)):
            Decrypted_matrix = (cipher_matrices[i] * FinalInverseMatrix)%127
        #   print(f"DEcryptedMatrix : {Decrypted_matrix}")
            DecryptedCharactersMatrix = numpy.matrix([[chr(Decrypted_matrix[0,0]),chr(Decrypted_matrix[0,1])]])

        # #   print(DecryptedCharactersMatrix)
        #     file.write(DecryptedCharactersMatrix[0,0])
        #     file.write(DecryptedCharactersMatrix[0,1])
            List.append(DecryptedCharactersMatrix[0,0])
            List.append(DecryptedCharactersMatrix[0,1])

        List.pop()
        # print(List)
        string = ""
        for i in range(len(List)):
            string = string +(List[i])

        return string
```

(II.2.3) Vigenere Cipher Encryption Implementation Source Code

```python
def encryption(message):

    import random
    randomListLength = random.randint(2,len(message))
    randomList=[]
    for i in range(0,randomListLength):
        randomList.append(random.randint(33,126))

    # print(randomList)

    MessageAscii =[]
    for i in range(len(message)):
        MessageAscii.append(ord(message[i]))

    # print(f"MessageAscii={MessageAscii}")


    for i in range(len(message)-len(randomList)):
        randomList.append(randomList[i])

    # print(f"RandomList = {randomList}")


    file = open('Keys/VignereCipherRandomList.bin','wb')
    for p in randomList:
        file.write(str.encode((f"{str(p)}\n")))

    # print(len(randomList))
    # print(len(message))
    probList =[]
    Encrypted_list =[]
    for i in range(len(message)):

        encrypted = ((MessageAscii[i])+(randomList[i])) %127
        if encrypted<33:
            probList.append(i)
            encrypted = encrypted+33
            Encrypted_list.append(encrypted)
        else (variable) Encrypted_list: list
            Encrypted_list.append(encrypted)

    # print(Encrypted_list)
    # print(probList)


    string =""
    for y in Encrypted_list:
        string = string +(chr(y))



    file = open('Keys/VignereCipherProbKeys.bin','wb')
    for p in probList:
        file.write(str.encode((f"{str(p)}\n")))

    return string
```

Vigenere Cipher Decryption Implementation Source Code

```
1
2   def decryption(Encrypted_list):
3       Encrypted_listT =[]
4       # with open('HillCipher\DecryptedText.txt','r') as file:
5       #     Encrypted_list = file.read()
6       for i in range(len(Encrypted_list)):
7           Encrypted_listT.append(ord(Encrypted_list[i]))
8       # print(Encrypted_list)
9       probList= []
10      with open('Keys/VignereCipherProbKeys.bin','rb') as file:
11          for line in file:
12              probList.append(int(line))
13
14      randomList = []
15      with open('Keys/VignereCipherRandomList.bin','rb') as file:
16          for line in file:
17              randomList.append(int(line))
18
19      # print(Encrypted_listT)
20      # print(randomList)
21
22      DecryptedList=[]
23      for i in range(len(Encrypted_list)):
24          if i in probList:
25              Encrypted_listT[i]=Encrypted_listT[i] -33
26              x=( Encrypted_listT[i]-randomList[i]) %127
27              DecryptedList.append(chr(x))
28          else:
29              x=( Encrypted_listT[i]-randomList[i]) %127
30              DecryptedList.append(chr(x))
31      string =""
32      for i in range(len(DecryptedList)):
33          string = string +(DecryptedList[i])
34      return string
```

```
0
```

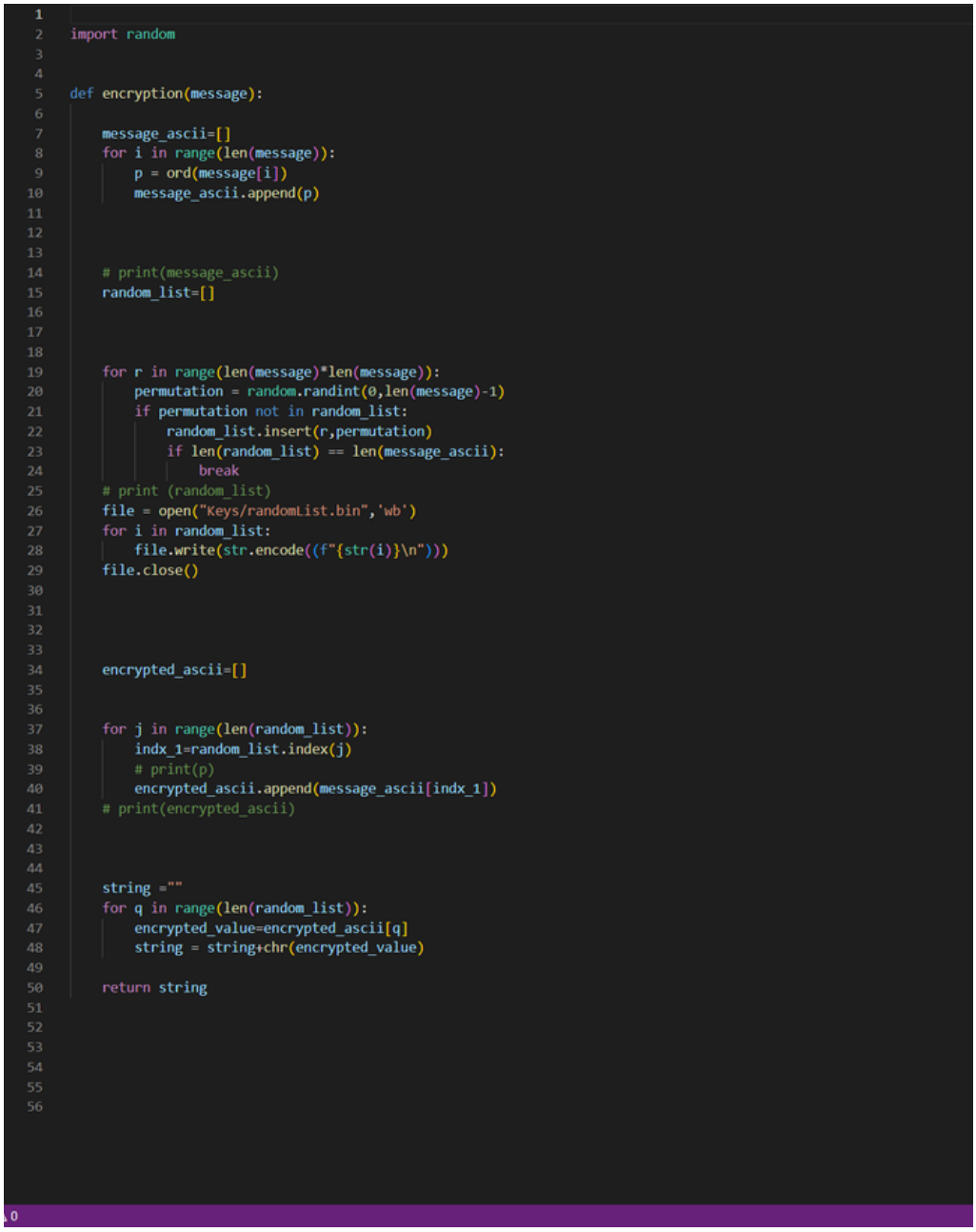
(II.2.4) Substitution Cipher Encryption Implementation Source Code

```python

def encryption(message):
    import random
    list_1 = list(range(32,127))
    list_2 = []

    for i in range(1000000):
        r = random.randint(32,126)
        if r not in list_2:
            list_2.append(r)
            if len(list_1)==len(list_2):
                break

    file = open('Keys/SubstitutionCipherList1.bin' ,'wb')
    for i in range(len(list_1)):
        file.write(str.encode((f"{str(list_1[i])}\n")))
    file.close()

    file = open('Keys/SubstitutionCipherList2.bin' ,'wb')
    for i in range(len(list_2)):
        file.write(str.encode((f"{str(list_2[i])}\n")))
    file.close()

    # print(f"List 1: {list_1}\n")
    # print(f"list_2:{list_2}\n")

    AsciiValueOfMessage =[]
    for i in range(len(message)):
        AsciiValueOfMessage.append(ord(message[i]))
    # print(AsciiValueOfMessage)

    FindeIndexOfAsciiVAlue =[]
    for i in range(len(AsciiValueOfMessage)):
        FindeIndexOfAsciiVAlue.append(list_1.index(AsciiValueOfMessage[i]))

    # print(FindeIndexOfAsciiVAlue)

    #us index pe list 2 ki value ckeck karo

    list_2Value =[]
    for i in range(len(FindeIndexOfAsciiVAlue)):
        list_2Value.append(list_2[FindeIndexOfAsciiVAlue[i]])

    # print(list_2Value)

    # list 2 ki in values se charcater banao
    EncryptedChar =[]
    for i in range(len(list_2Value)):
        EncryptedChar.append(chr(list_2Value[i]))
    # print(EncryptedChar)
    string =""
    for i in range(len(EncryptedChar)):
        string = string+(EncryptedChar[i])
    return string
```

Substitution Cipher Decryption Implementation Source Code

```python
def decryption(Encrypted_message):
    # with open("VignereCipher\Decrypted.txt",'r') as file:
    #     Encrypted_message = file.read()



    AsciiValueOfEncrypted = []
    for i in range(len(Encrypted_message)):
        AsciiValueOfEncrypted.append(ord(Encrypted_message[i]))

    # print(AsciiValueOfEncrypted)


    list_2 =[]
    with open('Keys/SubstitutionCipherList2.bin','rb') as file:
        for line in file:
            list_2.append(int(line))
    # print(list_2)

    list_1 =[]
    with open('Keys/SubstitutionCipherList1.bin','rb') as file:
        for line in file:
            list_1.append(int(line))
    # print(list_1)


    IndexofThisAscii = []
    for i in range(len(AsciiValueOfEncrypted)):
        IndexofThisAscii.append(list_2.index(AsciiValueOfEncrypted[i]))

    # print(IndexofThisAscii)



    #Ab is index pe list 1 ki value check karo
    ValueAtThisIndexInList1 =[]
    for i in range(len(IndexofThisAscii)):
        ValueAtThisIndexInList1.append(list_1[IndexofThisAscii[i]])


    #ab use decrypt karo

    decrypted = []
    for i in range(len(ValueAtThisIndexInList1)):
        decrypted.append(chr(ValueAtThisIndexInList1[i]))

    string =""
    for d in decrypted:
        string = string +(d)
    return string
```

(II.2.5) Permutation Cipher Encryption Implementation Source Code



```python
import random


def encryption(message):

    message_ascii=[]
    for i in range(len(message)):
        p = ord(message[i])
        message_ascii.append(p)



    # print(message_ascii)
    random_list=[]



    for r in range(len(message)*len(message)):
        permutation = random.randint(0,len(message)-1)
        if permutation not in random_list:
            random_list.insert(r,permutation)
            if len(random_list) == len(message_ascii):
                break
    # print (random_list)
    file = open("Keys/randomList.bin",'wb')
    for i in random_list:
        file.write(str.encode((f"{str(i)}\n")))
    file.close()




    encrypted_ascii=[]


    for j in range(len(random_list)):
        indx_1=random_list.index(j)
        # print(p)
        encrypted_ascii.append(message_ascii[indx_1])
    # print(encrypted_ascii)



    string =""
    for q in range(len(random_list)):
        encrypted_value=encrypted_ascii[q]
        string = string+chr(encrypted_value)

    return string
```

Permutation Cipher Decryption Implementation Source Code

```python

def decryption(message):
    random_list=[]
    file = open("Keys/randomList.bin","rb")
    for line in file:
        # print(line)
        random_list.append(int(line))


    encrypted_ascii=[]
    for i in range(len(message)):
        encrypted_ascii.append(ord(message[i]))


    decrypted_ascii=[]







    for y in range (len(random_list)):
            indx=random_list[y]
            # print(indx)
            # print(encrypted_ascii[indx])
            decrypted_ascii.append(encrypted_ascii[indx])
        # print(decrypted_ascii)



    string =""
    for h in range(len(decrypted_ascii)):
            decrypted_value=decrypted_ascii[h]

            string = string + chr(decrypted_value)

    return string


0
```

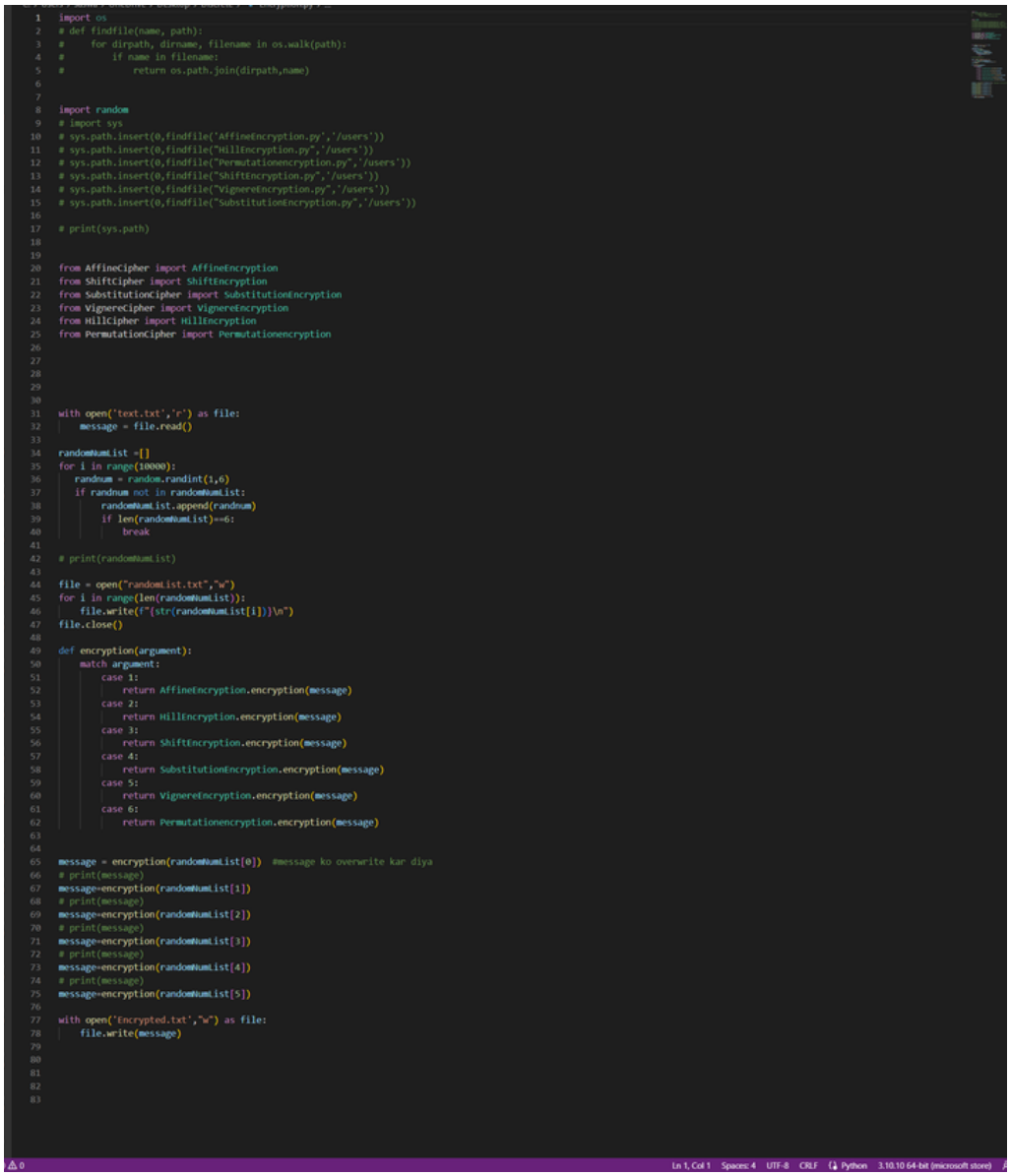
(II.2.6) Affine Cipher Encryption Implementation Source Code

```python

import random
def a_inverse(a):
        c=1
        if a==1:
            return 1
        else:
            for i in range(127*127):
                c=c+127
                f=c%a
                if f==0:
                    ans=c/a
                    ans=int(ans)
                    break
        return ans



def encryption(message):

    #Generates two random keys key1 and key2
    key1 = random.randint(0,126)
    key2= random.randint(2,126)


    #stores key1
    with open('Keys/AffineCipherkey1.bin','wb') as file:
        file.write(str.encode(str(key1)))

    #stores key2
    with open('Keys/AffineCipherkey2.bin','wb') as file:
        file.write(str.encode(str(key2)))

    #prob stores the index at which the value present is less than 33
    prob=[]

    #value_y stores the ord value of encrypted characters
    value_y=[]
    for i in range(len(message)):
        p = ord(message[i])
        y= (key2*p+key1)%127

        # Add 33 to those values which are less than 33
        if y<33:
            y=y+33
            prob.append(i)
        value_y.append(y)

    #This is a string to store the encrypted string
    string=""
    for y in value_y:
        string = string + chr(y)


    #This will store the prob keys in a file
    file = open('Keys/AffineCipherProbKeys.bin','wb')
    for p in prob:
        file.write(str.encode((f"{str(p)}\n")))

    #This will store the value_y in a file
    file = open('Keys/AffineCipherValue_y.bin','wb')
    for y in value_y:
        file.write(str.encode((f"{str(y)}\n")))
    file.close()


    return string
```

Affine Cipher Decryption Implementation Source Code

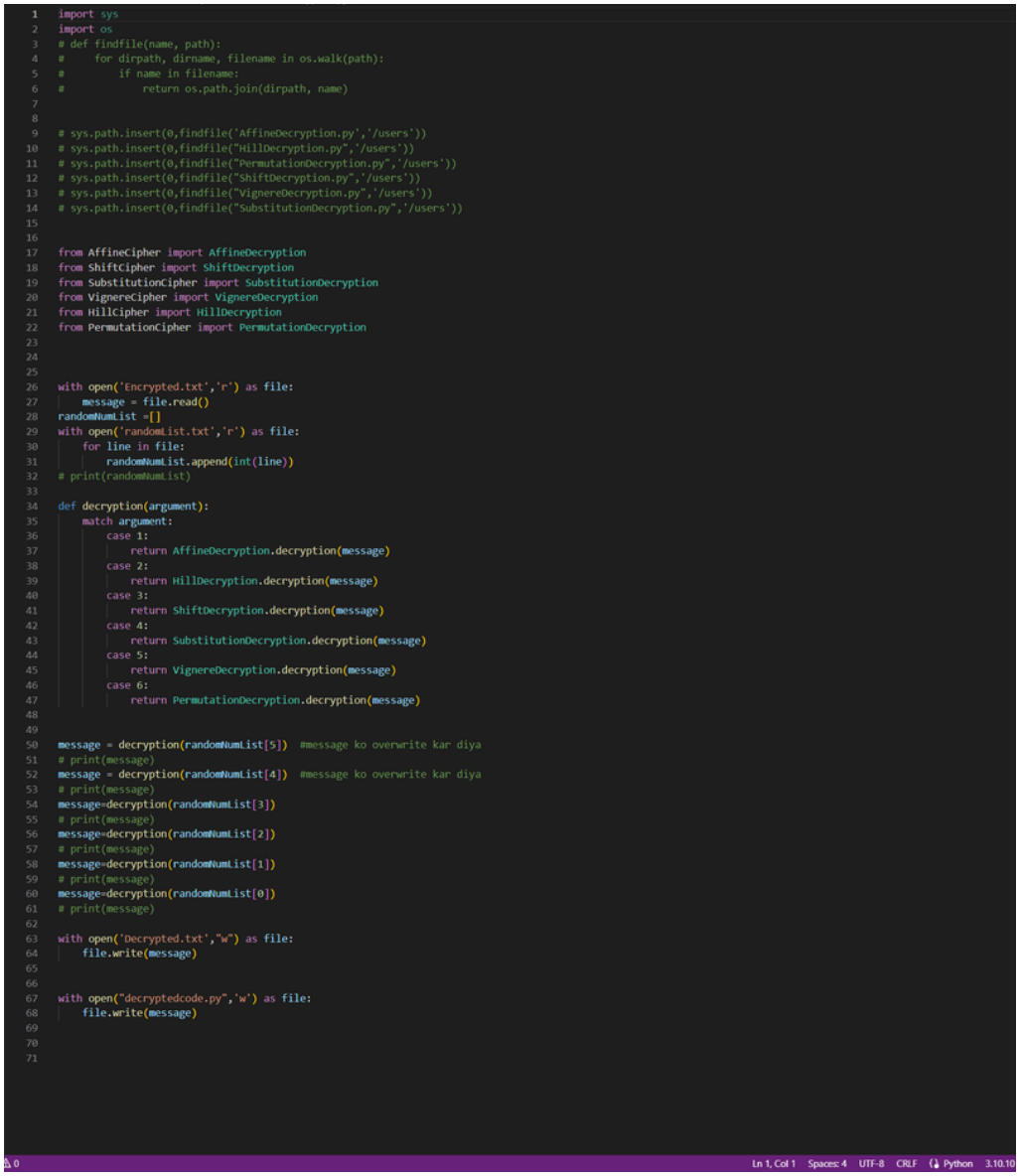
```python
def a_inverse(a):
    c=1
    if a==1:
        return 1
    else:
        for i in range(127*127):
            c=c+127
            f=c%a
            if f==0:
                ans=c/a
                ans=int(ans)
                break
    return ans

def decryption(Encrypted_message):

    #Reads the key1
    with open('Keys/AffineCipherkey1.bin','rb') as file:
        Key1Used = file.read()
    #Reads the key2
    with open('Keys/AffineCipherkey2.bin','rb') as file:
        Key2Used = file.read()

    key1 = int(Key1Used)
    key2= int(Key2Used)




    #Reads the prob Keys and stores them in problist
    problist= []
    with open('Keys/AffineCipherProbKeys.bin','rb') as file:
        for line in file:
            problist.append(int(line))


    #reads the ord value of encrypted character and stores it in value_y
    value_y = []
    with open('Keys/AffineCipherValue_y.bin','rb') as file:
        for line in file:
            value_y.append(int(line))

    DecryptedText = []
    for i in range(len(Encrypted_message)):
        if i in problist:
            #subtract 33 from those values to which we added 33 while encrypting
            value_j=value_y[i]-33
            a_inv= a_inverse(key2)
            x= a_inv*(value_j-key1)%127
            DecryptedText.append(chr(x))
        else:
            a_inv= a_inverse(key2)
            x= a_inv*(value_y[i]-key1)%127
            DecryptedText.append(chr(x))



    #returns the final decrypted string
    string =""
    for d in DecryptedText:
        string = string +(d)
    return string
```

(II.3) Randomization Encryption Implementation Source Code



```python
import os
# def findfile(name, path):
#     for dirpath, dirname, filename in os.walk(path):
#         if name in filename:
#             return os.path.join(dirpath,name)


import random
# import sys
# sys.path.insert(0,findfile('AffineEncryption.py','/users'))
# sys.path.insert(0,findfile("HillEncryption.py",'/users'))
# sys.path.insert(0,findfile("Permutationencryption.py",'/users'))
# sys.path.insert(0,findfile("ShiftEncryption.py",'/users'))
# sys.path.insert(0,findfile("VignereEncryption.py",'/users'))
# sys.path.insert(0,findfile("SubstitutionEncryption.py",'/users'))

# print(sys.path)


from AffineCipher import AffineEncryption
from ShiftCipher import ShiftEncryption
from SubstitutionCipher import SubstitutionEncryption
from VignereCipher import VignereEncryption
from HillCipher import HillEncryption
from PermutationCipher import Permutationencryption




with open('text.txt','r') as file:
    message = file.read()

randomNumList =[]
for i in range(10000):
  randnum = random.randint(1,6)
  if randnum not in randomNumList:
      randomNumList.append(randnum)
      if len(randomNumList)==6:
          break

# print(randomNumList)

file = open("randomList.txt","w")
for i in range(len(randomNumList)):
    file.write(f"{str(randomNumList[i])}\n")
file.close()

def encryption(argument):
    match argument:
        case 1:
            return AffineEncryption.encryption(message)
        case 2:
            return HillEncryption.encryption(message)
        case 3:
            return ShiftEncryption.encryption(message)
        case 4:
            return SubstitutionEncryption.encryption(message)
        case 5:
            return VignereEncryption.encryption(message)
        case 6:
            return Permutationencryption.encryption(message)


message = encryption(randomNumList[0])  #message ko overwrite kar diya
# print(message)
message=encryption(randomNumList[1])
# print(message)
message=encryption(randomNumList[2])
# print(message)
message=encryption(randomNumList[3])
# print(message)
message=encryption(randomNumList[4])
# print(message)
message=encryption(randomNumList[5])

with open('Encrypted.txt',"w") as file:
    file.write(message)
```

Randomization Decryption Implementation Source Code



```python
import sys
import os
# def findfile(name, path):
#     for dirpath, dirname, filename in os.walk(path):
#         if name in filename:
#             return os.path.join(dirpath, name)


# sys.path.insert(0,findfile('AffineDecryption.py','/users'))
# sys.path.insert(0,findfile("HillDecryption.py",'/users'))
# sys.path.insert(0,findfile("PermutationDecryption.py",'/users'))
# sys.path.insert(0,findfile("ShiftDecryption.py",'/users'))
# sys.path.insert(0,findfile("VignereDecryption.py",'/users'))
# sys.path.insert(0,findfile("SubstitutionDecryption.py",'/users'))


from AffineCipher import AffineDecryption
from ShiftCipher import ShiftDecryption
from SubstitutionCipher import SubstitutionDecryption
from VignereCipher import VignereDecryption
from HillCipher import HillDecryption
from PermutationCipher import PermutationDecryption



with open('Encrypted.txt','r') as file:
    message = file.read()
randomNumList =[]
with open('randomList.txt','r') as file:
    for line in file:
        randomNumList.append(int(line))
# print(randomNumList)

def decryption(argument):
    match argument:
        case 1:
            return AffineDecryption.decryption(message)
        case 2:
            return HillDecryption.decryption(message)
        case 3:
            return ShiftDecryption.decryption(message)
        case 4:
            return SubstitutionDecryption.decryption(message)
        case 5:
            return VignereDecryption.decryption(message)
        case 6:
            return PermutationDecryption.decryption(message)


message = decryption(randomNumList[5])  #message ko overwrite kar diya
# print(message)
message = decryption(randomNumList[4])  #message ko overwrite kar diya
# print(message)
message=decryption(randomNumList[3])
# print(message)
message=decryption(randomNumList[2])
# print(message)
message=decryption(randomNumList[1])
# print(message)
message=decryption(randomNumList[0])
# print(message)

with open('Decrypted.txt","w") as file:
    file.write(message)


with open("decryptedcode.py",'w') as file:
    file.write(message)
```

Future Work

We have developed some concepts that we intend to put into action in the future based on the progress we have made so far in this work:

- **Library** - We would like to contribute to the Python Community by building a library of this in the future which would enable anyone using python to import these functions and ciphers directly and use it on the go.

- **Image Encryption** - Using a secret key, an encrypted image is created through the process of image encryption. Using the secret key, the decryption process converts the cipher image into the original image.

References

- https://www.encryptionconsulting.com/education-center/what-is-cryptography/
- https://crypto.interactive-maths.com
- Cryptography theory and practice by-Douglar R Stinson