Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 13-08-2021	Subject: Cryptology

Theory

This encryption algorithm takes successive plaintext letters and substitutes for them ciphertext letters. The substitution is determined by linear equations in which each character is assigned a numerical value(a = 0, b = 1,……., z = 25). For m = 3, the system can be described as

$$c_1 = (k_{11}p_1 + k_{21}p_2 + k_{31}p_3) \bmod 26$$
$$c_2 = (k_{12}p_1 + k_{22}p_2 + k_{32}p_3) \bmod 26$$
$$c_3 = (k_{13}p_1 + k_{23}p_2 + k_{33}p_3) \bmod 26$$

This can be expressed in terms of row vectors and matrices:

$$(c_1\ c_2\ c_3) = (p_1\ p_2\ p_3)\begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \bmod 26$$

or

$$\mathbf{C} = \mathbf{PK} \bmod 26$$

where **C** and **P** are row vectors of length 3 representing the plaintext and ciphertext, and K is a 3 x 3 matrix representing the encryption key. Operations are performed mod 26.

For example, consider the plaintext "paymoremoney" and use the encryption key.

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

The first three letters of the plaintext are represented by the vector (15 0 24).

Then (15 0 24)**K** = (303 303 531) mod 26 = (17 17 11) = RRL. Continuing in this fashion, the ciphertext for the entire plaintext is RRLMWBKASPDH.

Decryption requires using the inverse of the matrix **K**. We can computedet **K** = 23, and therefore, (det **K**)$^{-1}$ mod 26 = 17. We can then compute the inverse as

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}\begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix K-1 is applied to the ciphertext, then the plaintext is recovered.

In general terms, the Hill system can be expressed as

$$\mathbf{C} = \mathrm{E}(\mathbf{K}, \mathbf{P}) = \mathbf{PK} \bmod 26$$
$$\mathbf{P} = \mathrm{D}(\mathbf{K}, \mathbf{C}) = \mathbf{CK}^{-1} \bmod 26 = \mathbf{PKK}^{-1} = \mathbf{P}$$

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger matrix hides more frequency information. Thus, a 3 * 3 Hill cipher hides not only single-letter but also two-letter frequency information.

Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack. For an m x m Hill cipher, sup- pose we have plaintext–ciphertext pairs, each of length . We label the pairs Pj = $(p_{1j}\ p_{1j} \ldots p_{mj})$ and $C_j = (c_{1j}\ c_{1j} \ldots c_{mj})$ such that $C_j = P_{jK}$ for $1 \leq j \leq m$ and for some unknown key matrix K. Now define two m x m matrices X = (p_{ij}) and Y = (c_{ij}). Then we can form the matrix equation Y = XK. If X has an inverse, then we can determine K = $X^{-1}Y$. If X is not invertible, then a new version of X can be formed with additional plaintext–ciphertext pairs until an invertible X is obtained.

Consider this example. Suppose that the plaintext "hillcipher" is encrypted using a 2 * 2 Hill cipher to yield the ciphertext HCRZSSXNSP. Thus, we know that (7 8)**K** mod 26 = (7 2); (11 11)**K** mod 26 = (17 25); and so on. Using the first two plaintext–ciphertext pairs, we have

$$\begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix} \mathbf{K} \bmod 26$$

The inverse of **X** can be computed:

$$\begin{pmatrix} 7 & 8 \\ 11 & 11 \end{pmatrix}^{-1} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix}$$

so

$$\mathbf{K} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix} \begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 549 & 600 \\ 398 & 577 \end{pmatrix} \bmod 26 = \begin{pmatrix} 3 & 2 \\ 8 & 5 \end{pmatrix}$$

This result is verified by testing the remaining plaintext–ciphertext pairs.

Code

```
import string
import numpy as np

while True:
    ch = int(input('Welcome to Hill Cipher Encryption and Decryption Program 
Made by Varun Khadayate..\n [*] Press 1 for Encryption \n [*] Press 2 for 
Decryption \n [*] Press 0 to exit..\n \nYour Choice:: '))

    if ch == 1:
        print("============================================")
        print("             !!!!Encryption!!!!             ")
        main=string.ascii_lowercase

        def generate_key(n,s):
            s=s.replace(" ","")
            s=s.lower()

            key_matrix=['' for i in range(n)]
            i=0;j=0
            for c in s:
                if c in main:
                    key_matrix[i]+=c
                    j+=1
                    if(j>n-1):
```

```python
                i+=1
                j=0
    print("The key matrix "+"("+str(n)+'x'+str(n)+") is:")
    print(key_matrix)

    key_num_matrix=[]
    for i in key_matrix:
        sub_array=[]
        for j in range(n):
            sub_array.append(ord(i[j])-ord('a'))
        key_num_matrix.append(sub_array)

    for i in key_num_matrix:
        print(i)
    return(key_num_matrix)


def message_matrix(s,n):
    s=s.replace(" ","")
    s=s.lower()
    final_matrix=[]
    if(len(s)%n!=0):
        while(len(s)%n!=0):
            s=s+'z'
    print("\n         !!!Encrypted Successfully!!!         ")
    print("Converted plain_text for encryption: ",s)
    for k in range(len(s)//n):
        message_matrix=[]
        for i in range(n):
            sub=[]
            for j in range(1):
                sub.append(ord(s[i+(n*k)])-ord('a'))
            message_matrix.append(sub)
        final_matrix.append(message_matrix)
    print("The column matrices of plain text in numbers are:  ")
    for i in final_matrix:
        print(i)
    return(final_matrix)

def getCofactor(mat, temp, p, q, n):
    i = 0
    j = 0

    for row in range(n):
        for col in range(n):
            if (row != p and col != q) :
                temp[i][j] = mat[row][col]
                j += 1
```

```
                        if (j == n - 1):
                            j = 0
                            i += 1
        def determinantOfMatrix(mat, n):
            D = 0
            if (n == 1):
                return mat[0][0]
            temp = [[0 for x in range(n)]
                    for y in range(n)]

            sign = 1
            for f in range(n):
                getCofactor(mat, temp, 0, f, n)
                D += (sign * mat[0][f] *
                    determinantOfMatrix(temp, n - 1))
                sign = -sign
            return D

        def isInvertible(mat, n):
            if (determinantOfMatrix(mat, n) != 0):
                return True
            else:
                return False

        def multiply_and_convert(key,message):
            res_num = [[0 for x in range(len(message[0]))] for y in
range(len(key))]

            for i in range(len(key)):
                for j in range(len(message[0])):
                    for k in range(len(message)):
                        res_num[i][j]+=key[i][k] * message[k][j]

            res_alpha = [['' for x in range(len(message[0]))] for y in
range(len(key))]
            for i in range(len(key)):
                for j in range(len(message[0])):
                    res_alpha[i][j]+=chr((res_num[i][j]%26)+97)

            return(res_alpha)

        n=int(input("What will be the order of square matrix: "))
        s=input("Enter the key: ")
        key=generate_key(n,s)

        if (isInvertible(key, len(key))):
            print("Yes it is invertable and can be decrypted")
        else:
```

```
            print("No it is not invertable and cannot be decrypted")

        plain_text=input("Enter the message: ")
        message=message_matrix(plain_text,n)
        final_message=''
        for i in message:
            sub=multiply_and_convert(key,i)
            for j in sub:
                for k in j:
                    final_message+=k
        print("plain message: ",plain_text)
        print("final encrypted message: ",final_message)
        print("\n============================================\n\n")

    elif ch == 2:
        print("\n============================================")
        print("               !!!Decryption!!!               ")
        main=string.ascii_lowercase

        def generate_key(n,s):
            s=s.replace(" ","")
            s=s.lower()

            key_matrix=['' for i in range(n)]
            i=0;j=0
            for c in s:
                if c in main:
                    key_matrix[i]+=c
                    j+=1
                    if(j>n-1):
                        i+=1
                        j=0
            print("The key matrix "+"("+str(n)+'x'+str(n)+") is:")
            print(key_matrix)

            key_num_matrix=[]
            for i in key_matrix:
                sub_array=[]
                for j in range(n):
                    sub_array.append(ord(i[j])-ord('a'))
                key_num_matrix.append(sub_array)

            for i in key_num_matrix:
                print(i)
            return(key_num_matrix)


        def modInverse(a, m) :
            a = a % m;
```

```
            for x in range(1, m) :
                if ((a * x) % m == 1) :
                    return x
            return 1

        def method(a, m) :
            if(a>0):
                return (a%m)
            else:
                k=(abs(a)//m)+1
            return method(a+k*m,m)


        def message_matrix(s,n):
            s=s.replace(" ","")
            s=s.lower()
            final_matrix=[]
            if(len(s)%n!=0):
                for i in range(abs(len(s)%n)):
                    s=s+'z'
            print("\n         !!!Decrypted Successfully!!!         ")
            print("Converted cipher_text for decryption: ",s)
            for k in range(len(s)//n):
                message_matrix=[]
                for i in range(n):
                    sub=[]
                    for j in range(1):
                        sub.append(ord(s[i+(n*k)])-ord('a'))
                    message_matrix.append(sub)
                final_matrix.append(message_matrix)
            print("The column matrices of plain text in numbers are:  ")
            for i in final_matrix:
                print(i)
            return(final_matrix)


        def multiply_and_convert(key,message):
            res_num = [[0 for x in range(len(message[0]))] for y in
range(len(key))]

            for i in range(len(key)):
                for j in range(len(message[0])):
                    for k in range(len(message)):
                        res_num[i][j]+=key[i][k] * message[k][j]

            res_alpha = [['' for x in range(len(message[0]))] for y in
range(len(key))]
            for i in range(len(key)):
                for j in range(len(message[0])):
```

```python
                res_alpha[i][j]+=chr((res_num[i][j]%26)+97)

        return(res_alpha)

    n=int(input("What will be the order of square matrix: "))
    s=input("Enter the key: ")
    key_matrix=generate_key(n,s)
    A = np.array(key_matrix)
    det=np.linalg.det(A)
    adjoint=det*np.linalg.inv(A)

    if(det!=0):
        convert_det=modInverse(int(det),26)
        adjoint=adjoint.tolist()
        print("Adjoint Matrix before modulo26 operation: ")
        for i in adjoint:
            print(i)
        print(convert_det)

        for i in range(len(adjoint)):
            for j in range(len(adjoint[i])):
                adjoint[i][j]=round(adjoint[i][j])
                adjoint[i][j]=method(adjoint[i][j],26)
        print("Adjoint Matrix after modulo26 operation: ")
        for i in adjoint:
            print(i)
        adjoint=np.array(adjoint)
        inverse=convert_det*adjoint

        inverse=inverse.tolist()
        for i in range(len(inverse)):
            for j in range(len(inverse[i])):
                inverse[i][j]=inverse[i][j]%26

        print("Inverse matrix after applying modulo26 operation: ")
        for i in inverse:
            print(i)

        cipher_text=input("Enter the cipher text: ")
        message=message_matrix(cipher_text,n)
        plain_text=''
        for i in message:
            sub=multiply_and_convert(inverse,i)
            for j in sub:
                for k in j:
                    plain_text+=k
        print("plain message: ",plain_text)
        print("\n============================================\n\n")
```

```python
        else:
            print("\n          !!!Decrypted Unsuccessfully!!!          ")
            print("Matrix cannot be inverted")
            print("\n===========================================\n\n")

    elif ch == 0:
        print("\n===========================================")
        print("     Thank You for using the Software ;)      ")
        print("                 Exiting Now.                 ")
        print("=============================================")
        exit()
```

Output

```
PS E:\College-Codes\Fourth Year\SEM VII\CT> & C:/Users/varun/AppData/Local/Programs/Python/Python310/python.exe "e:/College-Codes/Fourth Year/SEM VII/CT/Hill_Cipher.py"
Welcome to Hill Cipher Encryption and Decryption Program Made by Varun Khadayate..
 [*] Press 1 for Encryption
 [*] Press 2 for Decryption
 [*] Press 0 to exit..

Your Choice:: 1
============================================
             !!!!Encryption!!!!
What will be the order of square matrix: 3
Enter the key: monarchyk
The key matrix (3x3) is:
['mon', 'arc', 'hyk']
[12, 14, 13]
[0, 17, 2]
[7, 24, 10]
Yes it is invertable and can be decrypted
Enter the message: hi my name is varun

         !!!Encrypted Successfully!!!
Converted plain_text for encryption:  himynameisvarun
The column matrices of plain text in numbers are:
[[7], [8], [12]]
[[24], [13], [0]]
[[12], [4], [8]]
[[18], [21], [0]]
[[17], [20], [13]]
plain message:  hi my name is varun
final encrypted message:  oexcnmsgaqtgdcb

============================================


 Welcome to Hill Cipher Encryption and Decryption Program Made by Varun Khadayate..
  [*] Press 1 for Encryption
  [*] Press 2 for Decryption
  [*] Press 0 to exit..

 Your Choice:: 2

 ============================================
              !!!Decryption!!!
 What will be the order of square matrix: 3
 Enter the key: ['mon', 'arc', 'hyk']
 The key matrix (3x3) is:
 ['mon', 'arc', 'hyk']
 [12, 14, 13]
 [0, 17, 2]
 [7, 24, 10]
 Adjoint Matrix before modulo26 operation:
 [122.00000000000009, 172.00000000000014, -193.00000000000014]
 [14.000000000000009, 29.00000000000003, -24.000000000000018]
 [11, 18, 22]
 Inverse matrix after applying modulo26 operation:
 [2, 22, 19]
 [16, 9, 6]
 [7, 2, 14]
 Enter the cipher text: oexcnmsgaqtgdcb

          !!!Decrypted Successfully!!!
 Converted cipher_text for decryption:  oexcnmsgaqtgdcb
 The column matrices of plain text in numbers are:
 [[14], [4], [23]]
 [[2], [13], [12]]
 [[18], [6], [0]]
 [[16], [19], [6]]
 [[3], [2], [1]]
```

```
plain message:  himynameisvarun

============================================



Welcome to Hill Cipher Encryption and Decryption Program Made by Varun Khadayate..
 [*] Press 1 for Encryption
 [*] Press 2 for Decryption
 [*] Press 0 to exit..

Your Choice:: 0

============================================
     Thank You for using the Software ;)
                 Exiting Now.
==============================================
```