|  |  |
| --- | --- |
| 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) mod 26 *c*2 = (*k*12*p*1 + *k*22*p*2 + *k*32*p*3) mod 26 *c*3 = (*k*13*p*1 + *k*23*p*2 + *k*33*p*3) mod 26

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

Diagram

Description automatically generated with medium confidence

or

**C** = **PK** mod 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.

A picture containing text, clock, watch

Description automatically generated

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 compute det **K** = 23, and therefore, (det **K**)-1 mod 26 = 17. We can then compute the inverse as

Text

Description automatically generated

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

**C** = E(**K**, **P**) = **PK** mod 26

**P** = D(**K**, **C**) = **CK**-1 mod 26 = **PKK**-1 = **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 = (p1j p1j…pmj) and Cj = (c1j c1j…cmj) such that Cj = PjK for 1 ≤ j ≤ m and for some unknown key matrix K. Now define two m x m matrices X = (pij) and

Y = (cij). Then we can form the matrix equation Y = XK. If X has an inverse, then we can determine K = X-1Y. 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

Text, letter

Description automatically generated

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):

                        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])):

                    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")

        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



