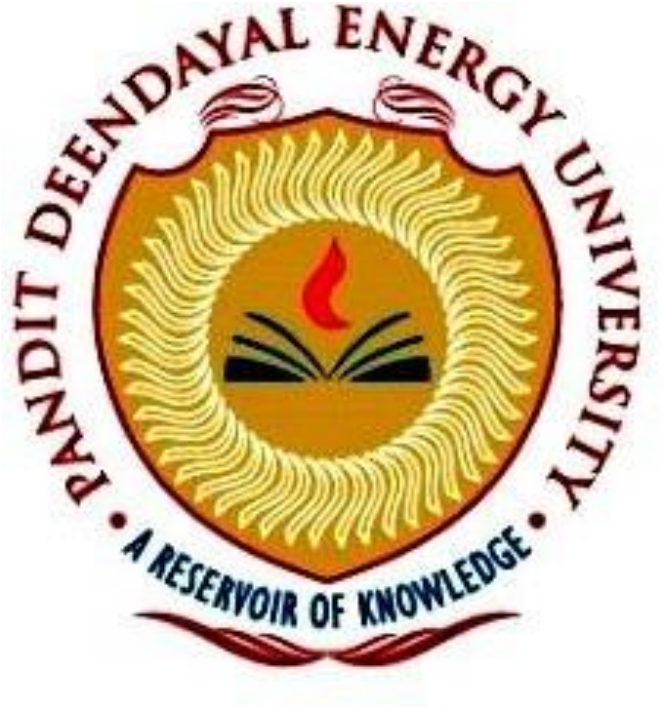
Information Security Lab

(20CP304P)

Smit Sutariya

Roll no. 21BCP142

Div:3 G:5



eFaculty Name: Hargeet Kaur

COMPUTER ENGINEERING

School of Technology,
Pandit Deendayal Energy University

January – 2023

Experiment No : 7

➔ Aim

Study and implement a program for n-gram Hill cipher.

➔ Introduction

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).
Examples:

```
Input  : Plaintext: ACT
         Key: GYBNQKURP
Output : Ciphertext: POH

Input  : Plaintext: GFG
         Key: HILLMAGIC
Output : Ciphertext: SWK
```

Encryption :

We have to encrypt the message ‘ACT’ (n=3).The key is ‘GYBNQKURP’ which can be written as the nxn matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message ‘ACT’ is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} (\text{mod } 26)$$

which corresponds to ciphertext of ‘POH’

Decryption

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters).The inverse of the matrix used in the previous example is:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

For the previous Ciphertext ‘POH’:

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

which gives us back ‘ACT’.
Assume that all the alphabets are in upper case.
Below is the implementation of the above idea for n=3

➔ Program

```
import java.util.Scanner;

public class hill_cipher {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("|Enter 1 to Encrypt and 0 to Decrypt|");
        System.out.print("Enter a number : ");
        int choice = in.nextInt();
        in.nextLine();


        if(choice==1){
            System.out.print("enter String to encrypt: ");
            String str = in.nextLine();
            System.out.print("enter key: ");
            String key = in.nextLine();
            System.out.print("String after encryption : ");
            System.out.println(encrypt(key.toLowerCase(),str.toLowerCase()));
        }
        else if(choice==0){
            System.out.println("enter String to decrypt: ");
            String str = in.nextLine();
            System.out.print("enter key: ");
            String key = in.nextLine();

            decrypt(str.toLowerCase(),key.toLowerCase());
        }
        else{
            System.out.println("enter a valid choice!");
        }
        System.out.println("==================================================
======");
    }
    public static String encrypt(String key,String plainText){
        if(!isPerfectSquare(key.length())){
            return "enter a key whose length is a perfect square root.";
        }
        String ans = "";
        int root = (int)Math.sqrt(key.length());
        int k = 0;
        int[][] keyMat = new int[root][root];
```

```
        for (int i = 0; i < root; i++) {
            for (int j = 0; j < root; j++) {
                keyMat[i][j] = (key.charAt(k)-'a') % 26;
                k++;
            }
        }
        int lengthPlain = plainText.length();
        int[][] plainMat;
        if(root>=lengthPlain){
            for (int i = 0; i < root-lengthPlain; i++) {
                plainText += 'x';
                lengthPlain++;
            }
            plainMat = new int[root][1];
        } else{
            while(lengthPlain%root!=0){
                plainText += 'x';
                lengthPlain++;
            }
            plainMat = new int[root][lengthPlain/root];
        }
        int temp = 0;
        for (int i = 0; i < lengthPlain/root; i++) {
            for (int j = 0; j < root; j++) {
                plainMat[j][i] = (plainText.charAt(temp) -'a') % 26;
                temp++;
            }
        }

        char[][] finalMat = matrixMul(keyMat,plainMat);

        for (int i = 0; i < finalMat[0].length; i++) {
            for (int j = 0; j < finalMat.length; j++) {
                ans += finalMat[j][i];
            }
        }

//        for (int i = 0; i < root; i++) {
//            System.out.println(Arrays.toString(keyMat[i]));
//        }
//        System.out.println("============================");
//        for (int i = 0; i < root; i++) {
//            System.out.println(Arrays.toString(plainMat[i]));
//        }
```

```
        return ans;
    }

    public static char[][] matrixMul(int[][] keyMat, int[][] plainMat) {
        char finalMat[][] = new char[keyMat.length][plainMat[0].length];
        for (int i = 0; i < finalMat.length; i++) {
            for (int j = 0; j < finalMat[i].length; j++) {
                for (int k = 0; k < keyMat[0].length; k++) {
                    finalMat[i][j] += (keyMat[i][k] * plainMat[k][j]) ;
                }
                finalMat[i][j] = (char)((finalMat[i][j] %26) + 97) ;
            }
        }
//        for (int i = 0; i < finalMat.length; i++) {
//            System.out.println(Arrays.toString(finalMat[i]));
//        }
//        System.out.println("==============================");
        return finalMat;
    }
    public static boolean isPerfectSquare(int a){
        int root = (int)Math.sqrt(a);
        return root*root == a;
    }
    public static int[][] calculateAdjoint(int[][] matrix) {
        int size = matrix.length;
        int[][] adjoint = new int[size][size];

        if (size == 1) {
            adjoint[0][0] = 1;
        } else if (size == 2) {
            adjoint[0][0] = matrix[1][1];
            adjoint[0][1] = -matrix[0][1];
            adjoint[1][0] = -matrix[1][0];
            adjoint[1][1] = matrix[0][0];
        } else {
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    int[][] submatrix = getSubmatrix(matrix, i, j);
                    adjoint[i][j] = (int) Math.pow(-1, i + j) *
calculateDeterminant(submatrix);
                }
            }
            adjoint = transposeMatrix(adjoint);
        }
```

```
        return adjoint;
    }

    public static int[][] getSubmatrix(int[][] matrix, int rowToRemove, int
colToRemove) {
        int size = matrix.length;
        int[][] submatrix = new int[size - 1][size - 1];
        int rowIndex = 0;
        int colIndex;

        for (int i = 0; i < size; i++) {
            if (i == rowToRemove) {
                continue;
            }

            colIndex = 0;
            for (int j = 0; j < size; j++) {
                if (j == colToRemove) {
                    continue;
                }

                submatrix[rowIndex][colIndex] = matrix[i][j];
                colIndex++;
            }
            rowIndex++;
        }

        return submatrix;
    }

    public static int calculateDeterminant(int[][] matrix) {
        int size = matrix.length;

        if (size == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
        }

        int determinant = 0;
        for (int i = 0; i < size; i++) {
            determinant += matrix[0][i] * (int) Math.pow(-1, i) *
calculateDeterminant(getSubmatrix(matrix, 0, i));
        }
```

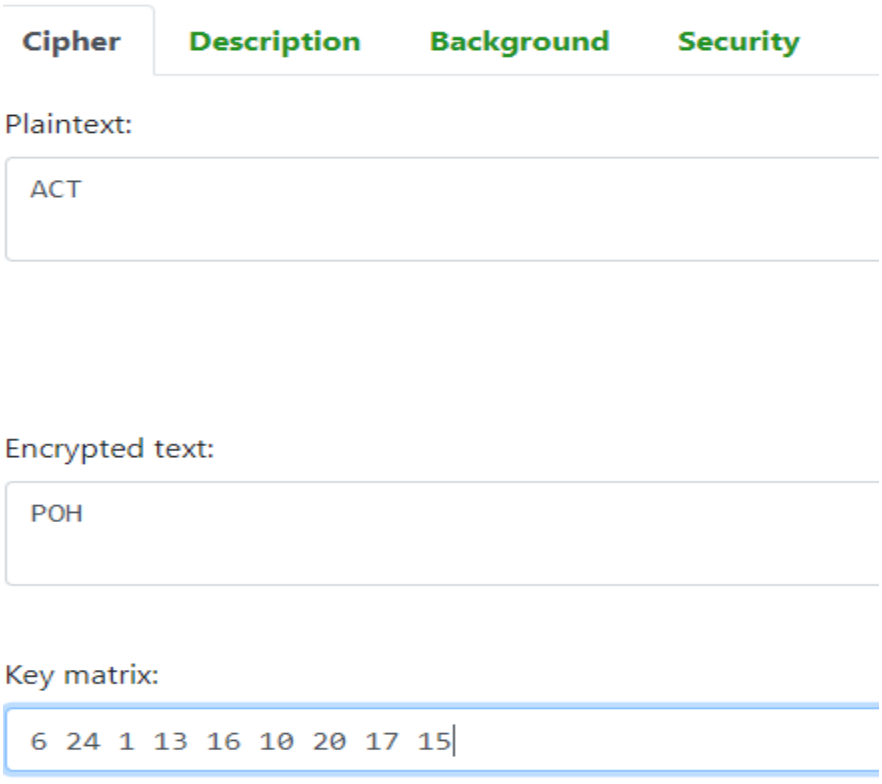
```
        return determinant;
    }

    public static int[][] transposeMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] transposed = new int[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                transposed[j][i] = matrix[i][j];
            }
        }

        return transposed;
    }
    /*-------------------------------------------------------------------------
-------*/

    // finding the modulo inverse of number A under modulo 'M'
    public static int modInverse(int A, int M) {
        for (int i = 1; i < M; i++) {
            if (((A % M) * (i % M)) % M == 1) {
                return i;
            }
        }
        return -1;
    }

    public static int[][] inverseOfKeyMatrix(int[][] keyMatrix) {
        int determinant = calculateDeterminant(keyMatrix);
        if (determinant < 0) {
            determinant = 26 - (Math.abs(determinant) % 26);
        }
        determinant = determinant % 26;
        int modularInverse = modInverse(determinant, 26);

        int[][] inverseMatrix = calculateAdjoint(keyMatrix);

        for (int i = 0; i < inverseMatrix.length; i++) {
            for (int j = 0; j < inverseMatrix[0].length; j++) {
                inverseMatrix[i][j] = (inverseMatrix[i][j]) % 26;
                if (inverseMatrix[i][j] < 0) {
                    inverseMatrix[i][j] = 26 - Math.abs(inverseMatrix[i][j]);
```

```
                }
                inverseMatrix[i][j] = (inverseMatrix[i][j] * modularInverse) %
26;
            }
        }
        return inverseMatrix;
    }
    public static void decrypt(String str,String key){
        if(!isPerfectSquare(key.length())){
            System.out.println("enter a key whose length is a perfect square
root.");
        }
        int root = (int)Math.sqrt(key.length());
        int[][] keyMat = new int[root][root];
        int k=0;
        for (int i = 0; i < root; i++) {
            for (int j = 0; j < root; j++) {
                keyMat[i][j] = (key.charAt(k)-'a') % 26;
                k++;
            }
        }
        int temp =0;
        int[][] inverseKey = inverseOfKeyMatrix(keyMat);
        int[][] strMat = new int[root][str.length()/root];
        for (int i = 0; i < strMat[0].length; i++) {
            for (int j = 0; j < strMat.length; j++) {
                strMat[j][i] = (str.charAt(temp)-'a') % 26;
                temp++;
            }
        }

        char[][] finalText = matrixMul(inverseKey,strMat);

        String ans = "";
        for (int i = 0; i < finalText[0].length; i++) {
            for (int j = 0; j < finalText.length; j++) {
                ans += finalText[j][i];
            }
        }
        System.out.println("Plaintext : " + ans);
    }
}
```

➔ Output (Program)

```
C:\Users\hp\Desktop\IS>java HillCipher
|Enter 1 to Encrypt and 0 to Decrypt|
Enter a number : 1
enter String to encrypt: ACT
enter key: GYBNQKURP
String after encryption : poh
================================================

C:\Users\hp\Desktop\IS>java HillCipher
|Enter 1 to Encrypt and 0 to Decrypt|
Enter a number : 0
enter String to decrypt:
POH
enter key: GYBNQKURP
Plaintext : act
================================================
```

➔ Output (Cryptool)



➔ Cryptanalysis

1. Key Size and Complexity: Hill cipher uses a matrix as the encryption key, which can be relatively large and complex, especially for longer messages. The key size increases with the length of the message, making it unwieldy for practical use.

2. Vulnerability to Known-Plaintext Attacks: Hill cipher is susceptible to known-plaintext attacks, where an attacker can decipher an encrypted message if they have knowledge of the plaintext message and the corresponding ciphertext. This makes it insecure in real-world scenarios.

3. Key Inversion: In some cases, it can be relatively easy to compute the inverse of the key matrix, especially when the matrix is not chosen carefully. Once the key matrix is known, an attacker can easily decrypt messages.

4. Lack of Confusion and Diffusion: Hill cipher does not exhibit strong confusion and diffusion properties, which are important characteristics of secure encryption algorithms. This makes it vulnerable to various cryptographic attacks, including frequency analysis.

5. Low Resistance to Brute-Force Attacks: Due to the limited key space, Hill cipher is susceptible to brute-force attacks, where an attacker systematically tries all possible keys to decrypt a message. With modern computational power, brute-force attacks on Hill cipher can be relatively quick.

➔ Application

1. Secure Communication: Hill ciphers can be used to encrypt sensitive messages and ensure secure communication between two parties. While it's

considered relatively weak by modern standards, it can still provide a level of confidentiality for basic communications.

2. Educational Purposes: Hill cipher is often used in educational settings to teach students about encryption, linear algebra, and cryptography principles. It's a simple yet instructive example of how encryption works.

3. Historical Cryptography: Hill cipher was developed in the early 20th century and was one of the first examples of polygraphic substitution ciphers. It has historical significance as a pioneering cryptographic technique.

4. Steganography: Hill cipher can be used in steganography, which is the practice of hiding messages within other seemingly innocuous data. By encrypting a message using the Hill cipher and then embedding it within an image or text, one can hide sensitive information.

5. Basic Data Protection: While not suitable for securing highly sensitive data in modern contexts, Hill cipher can provide basic data protection for less critical applications where strong encryption is not necessary.

6. Obfuscation: In some cases, Hill cipher can be used for obfuscation rather than strict security. It can make data less readable to casual observers or automated algorithms.

➔ References

1) https://www.geeksforgeeks.org/hill-cipher/
2) https://chat.openai.com/c/eb0a5e67-867f-41f9-afff-27a280a7fbab