Indian Institute of Information Technology Surat

Lab Report on
Network Security (CS 702) Practical

Submitted by

[RAHUL KUMAR SINGH] (UI21CS44)

Course Faculty

Dr. Reema Patel

Department of Computer Science and Engineering
Indian Institute of Information Technology Surat
Gujarat-394190, India

Aug-2024

Lab No: 5

Aim:

To implement a Hill Cipher for secure file encryption and decryption using Java.

Description:

Write the menu driven program for Hill Climbing Cipher with a key string converted into a matrix in java

1. Encryption and Decryption of large input text. (Input: File can contain large text (including characters, symbols, spaces, etc.)

Input Key: Key may be a statement which can contains spaces, symbols, etc.
Input File Name: plaintext.txt
Encrypted File Name: cipher.txt
Decrypted Filename: recover.txt

Code: (To be noted that MOD = 29 (for primality satisfaction))

```
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class HillCipherGUI {
    public static int mod = 29;
    public static void main(String[] args) {
        JFrame frame = new JFrame("Hill Cipher Encryption and Decryption");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton encryptButton = new JButton("Encrypt (Hill Cipher)");
        JButton decryptButton = new JButton("Decrypt (Hill Cipher)");

        encryptButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                performEncryption();
            }
        });

        decryptButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                performDecryption();
            }
        });
```

```
        JPanel panel = new JPanel();
        panel.add(encryptButton);
        panel.add(decryptButton);
        frame.add(panel);
        frame.setVisible(true);
    }

    public static void performEncryption() {
        try {
            String inputFileName = "plaintext.txt";
            String outputFileName = "cipher.txt";
            String content = new
String(Files.readAllBytes(Paths.get(inputFileName)));

            String key = JOptionPane.showInputDialog("Enter the Key for Hill
Cipher:");
            int[][] keyMatrix = generateKeyMatrix(key);

            String cleanedContent = cleanInput(content);
            String encrypted = encryptHillCipher(cleanedContent, keyMatrix);
            Files.write(Paths.get(outputFileName), encrypted.getBytes());

            JOptionPane.showMessageDialog(null, "Encryption Completed! Encrypted
file saved as " + outputFileName);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "Error: " + ex.getMessage());
        }
    }

    public static void performDecryption() {
        try {
            String inputFileName = "cipher.txt";
            String outputFileName = "recover.txt";
            String content = new
String(Files.readAllBytes(Paths.get(inputFileName)));

            String key = JOptionPane.showInputDialog("Enter the Key for Decryption
(Hill Cipher):");
            int[][] keyMatrix = generateKeyMatrix(key);
            int[][] inverseKeyMatrix = invertKeyMatrix(keyMatrix);
```

```
            String decrypted = decryptHillCipher(content, inverseKeyMatrix);
            Files.write(Paths.get(outputFileName), decrypted.getBytes());

            JOptionPane.showMessageDialog(null, "Decryption Completed! Decrypted
file saved as " + outputFileName);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "Error: " + ex.getMessage());
        }
    }

    public static String cleanInput(String text) {
        return text.replaceAll("[^A-Za-z0-9]", "").toUpperCase();
    }

    public static int[][] generateKeyMatrix(String key) {
        key = cleanInput(key);
        int matrixSize = (int) Math.sqrt(key.length());

        int[][] keyMatrix = new int[matrixSize][matrixSize];
        int index = 0;
        for (int i = 0; i < matrixSize; i++) {
            for (int j = 0; j < matrixSize; j++) {
                keyMatrix[i][j] = key.charAt(index) % 65;
                index++;
            }
        }
        return keyMatrix;
    }

    public static String encryptHillCipher(String text, int[][] keyMatrix) {
        int n = keyMatrix.length;
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i += n) {
            int[] vector = new int[n];
            for (int j = 0; j < n; j++) {
                if (i + j < text.length()) {
                    vector[j] = text.charAt(i + j) % 65;
                } else {
                    vector[j] = 'X' % 65;
                }
            }
```

```
            int[] encryptedVector = multiplyMatrixVector(keyMatrix, vector);
            for (int val : encryptedVector) {
                result.append((char) ((val % mod) + 65));
            }
        }
        return result.toString();
    }

    public static String decryptHillCipher(String text, int[][] inverseKeyMatrix) {
        int n = inverseKeyMatrix.length;
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < text.length(); i += n) {
            int[] vector = new int[n];
            for (int j = 0; j < n; j++) {
                vector[j] = text.charAt(i + j) % 65;
            }
            int[] decryptedVector = multiplyMatrixVector(inverseKeyMatrix, vector);
            for (int val : decryptedVector) {
                result.append((char) ((val % mod) + 65));
            }
        }
        return result.toString();
    }

    public static int[] multiplyMatrixVector(int[][] matrix, int[] vector) {
        int n = matrix.length;
        int[] result = new int[n];
        for (int i = 0; i < n; i++) {
            result[i] = 0;
            for (int j = 0; j < n; j++) {
                result[i] += matrix[i][j] * vector[j];
            }
        }
        return result;
    }

    public static int powMod(int a, int p, int m){
        int resu=1;
        int base=a;
        while(p>0)
        {
```

```
            if (p%2==1)
            {
                resu=(resu*base)%m;
            }
            p/=2;
            base=(base*base)%m;
        }
        return resu;
    }
    public static int modInverse(int a, int m) {
        a = a % m;
        for (int x = 1; x < m; x++) {
            if ((a * x) % m == 1) {
                return x;
            }
        }
        return 1;
    }


    public static int determinant(int[][] matrix) {
        int det = matrix[0][0] * (matrix[1][1] * matrix[2][2] - matrix[1][2] *
matrix[2][1]) -
                  matrix[0][1] * (matrix[1][0] * matrix[2][2] - matrix[1][2] *
matrix[2][0]) +
                  matrix[0][2] * (matrix[1][0] * matrix[2][1] - matrix[1][1] *
matrix[2][0]);
        return det;
    }

    public static int[][] cofactorMatrix(int[][] matrix) {
        int[][] cofactor = new int[3][3];
        cofactor[0][0] = matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1];
        cofactor[0][1] = -(matrix[1][0] * matrix[2][2] - matrix[1][2] *
matrix[2][0]);
        cofactor[0][2] = matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0];
        cofactor[1][0] = -(matrix[0][1] * matrix[2][2] - matrix[0][2] *
matrix[2][1]);
        cofactor[1][1] = matrix[0][0] * matrix[2][2] - matrix[0][2] * matrix[2][0];
        cofactor[1][2] = -(matrix[0][0] * matrix[2][1] - matrix[0][1] *
matrix[2][0]);
        cofactor[2][0] = matrix[0][1] * matrix[1][2] - matrix[0][2] * matrix[1][1];
```

```
        cofactor[2][1] = -(matrix[0][0] * matrix[1][2] - matrix[0][2] *
matrix[1][0]);
        cofactor[2][2] = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
        return cofactor;
    }


    public static int[][] transpose(int[][] matrix) {
        int[][] transposed = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                transposed[i][j] = matrix[j][i];
            }
        }
        return transposed;
    }


    public static int[][] invertMatrix(int[][] matrix) {
        int determinant = determinant(matrix) % mod;
        if (determinant < 0) {
            determinant += mod;
        }

        int detInverse = powMod(determinant, mod-2, mod); // Base Mod value is 29 as
the mod needs to prime to have inverse of the determinant
        int[][] cofactorMatrix = cofactorMatrix(matrix);
        int[][] adjugateMatrix = transpose(cofactorMatrix);
        int[][] inverseMatrix = new int[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                inverseMatrix[i][j] = (adjugateMatrix[i][j] * detInverse) % mod;
                if (inverseMatrix[i][j] < 0) {
                    inverseMatrix[i][j] += mod;
                }
            }
        }
        return inverseMatrix;
    }
    public static int[][] invertKeyMatrix(int[][] matrix) {
        int[][] inverseMatrix = invertMatrix(matrix);
        return inverseMatrix;
```

```
    }
}
```

Output:

Plaintext.txt

```
In the vast and ancient world, where time flows like a river and the stars paint the sky, there existed a land of great beauty and mystery. This land, rich with rolling hills, deep forests, and winding rivers, was home to a people whose lives were intertwined with the natural rhythms of the earth. They cultivated the land, worshipped the sun and the moon, and passed down stories from one generation to the next. These stories spoke of heroes and gods, of love and betrayal, of battles won and lost. But among all these tales, one legend stood out. It was the tale of a hidden treasure, buried deep within the heart of the land. A treasure so vast and powerful that it could change the course of history. Many had searched for it, but none had found it. Some said it was guarded by a fierce dragon, others believed it was cursed. But despite the dangers, the lure of the treasure was too great. And so, the search continued, driven by hope, greed, and the desire for glory. As the years passed, the legend grew, taking on a life of its own. But in the end, it was not the treasure that mattered, but the journey itself.
```

Cipher.txt [Key: HELLOWORD] & [Mod: 29]

```
\VNGP[UDP\Q\QXYMYQ\NPWB]XWEZXPDZJBYJ]LJY[ELIXCXH\Q\CSU]AYIPJBVBZUBHXPCRNYWJFUHOBH[OR\Q\KIJTLQHVF]CUV]\OE
]OBHSNHSF\RBTDMHXYKGPVGQQU[KNVYAESXDGSVSOBMC[V[CJZRXAZDFHCELWEAOFPY[JPRIGS[UTNQLBVV]FX[WYWJ\VNPEZKCTVT]G
PVCSUKKHFXSTUACRNUZXOLIPNIL\CCSUTHYJMIILDRLTM\Z\Q\UARKISG]NASJBLQCXSASJ\\MFRLHWRJKMPHAKXAISIVKJOELKVH]MH
IY]JZFXTMVLKFNCKBLCSUVM[]ZEJSCVU\IKSUJ\QHYIUBZTMPNNBYJS[O\EEBRYJRCOVBNXNSQBUCBCXSQCKKW\NWXWFKSSB\TF]H[WQ
JWY\ZGWJNZLB[YRYHCV[OYEAOCSUWQJXCTAN\TSSGAFXFXUYBN\]SHPXDGUL]RTDMMEN\NQS\IETM\Z\Q\CC[XFXUYBHPCUDP\Q\[EVG
PMQJC[CKJVGJJSTMKCYHHAXQDQANOXCTSF\]ZEN[SDYCHDSMMU]FOVFYMVZGNFAQY\[BVFYWERLZZQEGL[PBMDEAOG\SB\L]YPVOWCHD
YZNEP]IDNUOOAKUICMWFX[[IHGWANOWVXRBDERBCHMCSUTAXNDASJQCAN]AAOLIBRYXFXUYBEAOYRYJPISFELB[TXS]H[DVFIHAXFHMS
MRO]KGRLSKJXEHJGTVR\Q\CSUDVTXIHRQLIGXMBISJQXEMFZYNKDBBKCSUOTI[CZJPI\PWNPZJ[NQLICNGPFCSIFZR\S\GCSU[CZ[[IM
Q[G[ZNAUTLQQLFHAXFILYRDYWJNXDZUBPOTDT]JFYR\ECBS
```

Recover.txt

```
INTHEVASTANDANCIENTWORLDWHERETIMEFLOWSLIKEARIVERANDTHESTARSPAINTTHESKYTHEREEXISTEDALANDOFGREATBEAUTYANDM
YSTERYTHISLANDRICHWITHROLLINGHILLSDEEPFORESTSANDWINDINGRIVERSWASHOMETOAPEOPLEWHOSELIVESWEREINTERTWINEDWI
THTHENATURALRHYTHMSOFTHEEARTHTHEYCULTIVATEDTHELANDWORSHIPPEDTHESUNANDTHEMOONANDPASSEDDOWNSTORIESFROMONEG
ENERATIONTOTHENEXTTHESESTORIESSPOKEOFHEROESANDGODSOFLOVEANDBETRAYALOFBATTLESWONANDLOSTBUTAMONGALLTHESETA
LESONELEGENDSTOODOUTITWASTHETALEOFAHIDDENTREASUREBURIEDDEEPWITHINTHEHEARTOFTHELANDATREASURESOVASTANDPOWE
RFULTHATITCOULDCHANGETHECOURSEOFHISTORYMANYHADSEARCHEDFORITBUTNONEHADFOUNDITSOMESAIDITWASGUARDEDBYAFIERC
EDRAGONOTHERSBELIEVEDITWASCURSEDBUTDESPITETHEDANGERSTHELUREOFTHETREASUREWASTOOGREATANDSOTHESEARCHCONTINU
EDDRIVENBYHOPEGREEDANDTHEDESIREFORGLORYASTHEYEARSPASSEDTHELEGENDGREWTAKINGONALIFEOFITSOWNBUTINTHEENDITWA
SNOTTHETREASURETHATMATTEREDBUTTHEJOURNEYITSELFX
```

Conclusion:

- The Hill Cipher provides moderate security by using matrix-based encryption, but it is vulnerable if the key matrix is weak or improperly selected.
- It is well-suited for encrypting large blocks of text but can be computationally expensive due to matrix inversion over mod 29.
- It requires mathematical constraints such as the mod of matrix needs to prime value.
- Modern ciphers like AES are far more secure and efficient for large-scale digital encryption.
- Hill Cipher is primarily useful for educational purposes.