Ex.No.3	IMPLEMENTATION AND CRYPTANALYSIS OF HILL CIPHER

AIM:

To implement

- Encryption
- Decryption
- Known Plain test – cipher text attack

in Hill cipher using Java.

THEORY:

Encryption

To encrypt a message using the Hill Cipher we must have a key. We also turn the plaintext into n-graphs and each of these into a column vector. We then perform matrix multiplication of key with a matrix formed with those column vectors then modulo the length of the alphabet on each vector. These vectors are then converted back into letters to produce the ciphertext.

Decryption

To decrypt a ciphertext encoded using the Hill Cipher, we must find the inverse matrix. Once we have the inverse matrix, the process is the same as encrypting. That is we multiply the inverse key matrix by the column vectors that the ciphertext is split into, take the results modulo the length of the alphabet, and finally convert the numbers back to letters.

Constraints for selecting Key Matrix in Hill Cipher

The key matrix must be

- a square matrix of size n x n where n denotes the size of column vector size.
- a non-singular matrix i.e. det(k) not equal to zero.
- The det(k) and m must be relative prime of each other i.e. GCD (det(k), m) = 1.

Steps for calculation of inverse of Key Matrix in Modulo arithmetic:

1. Find the determinant of key matrix.
    a. If the det(k) = 0 drop the matrix conclude that it can’t be a key.
    b. Else continue

2. Find the inverse of det(k) using Extended Euclidean algorithm.
3. Find the determinant of key matrix (k)
4. Do scalar multiplication $det(k)^{-1}$ with adj(k) this will give you the square matrix.

Euclidean and Extended Euclidean algorithm

The extended Euclidean algorithm is particularly useful when a and b are coprime (or gcd is 1). Since x is the modular multiplicative inverse of "a modulo b". In particular, the computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.

Known Plaintext and Cipher text attack with an example

A known plaintext ciphertext attack involves having knowledge of both the plaintext (the original message) and its corresponding ciphertext (the encrypted message). The goal of the attack is to deduce the encryption key or to find a method to decrypt other ciphertexts encrypted with the same key.

Example:

Ciphertext: FBRTLWUGATEPHBNXSW

Plaintext: JAMESBOND corresponds to the last 9 letters and the key matrix size is 3 x 3

By using the formula $P \cong K^{-1} C \bmod m$ and CT 'FBRTLWUGA' we can find the Plain text which is 'SENDROSESJAMESBOND' once we deduce the value of K^{-1}.

ALGORITHM:

Encryption
Decryption
Known Plaintext attack and Cipher text attack
import java.util.Scanner;
1)Hill Cipher

```
public class HillCipher {
   private final String alphaList;
   private final int m;

   public HillCipher(int m, String alphaList) {
      this.alphaList = alphaList;
      this.m = m;
   }
```

```
public int findGCD(int a, int b) {
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

public int extendedEuclids(int a, int m) {
    int t1 = 0, t2 = 1;
    int b = m;
    if (a < b) {
        int temp = a;
        a = b;
        b = temp;
    }
    while (b != 0) {
        int q = a / b;
        int temp = b;
        b = a % b;
        a = temp;
        int t = t1 - q * t2;
        t1 = t2;
        t2 = t;
    }
    if (t1 < 0) {
        return t1 + m;
    } else {
        return t1;
    }
}

public int[][] convertStrMat(String word, int syllable) {
    while (word.length() % syllable != 0) {
```

```
            word += 'X';
        }
        int rows = syllable;
        int cols = word.length() / syllable;
        int[][] result = new int[rows][cols];
        int k = 0;
        for (int i = 0; i < cols; i++) {
            for (int j = 0; j < rows; j++) {
                result[j][i] = alphaList.indexOf(word.charAt(k++));
            }
        }
        return result;
    }

    public String convertMatStr(int[][] mat) {
        StringBuilder text = new StringBuilder();
        for (int i = 0; i < mat[0].length; i++) {
            for (int j = 0; j < mat.length; j++) {
                text.append(alphaList.charAt(mat[j][i] % m));
            }
        }
        return text.toString();
    }

    public int[][] findMatMul(int[][] matA, int[][] matB) {
        int rowsA = matA.length;
        int colsA = matA[0].length;
        int colsB = matB[0].length;
        int[][] result = new int[rowsA][colsB];
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += matA[i][k] * matB[k][j];
                }
                result[i][j] %= m;
            }
        }
        return result;
    }

    public int[][] removeCopyMatrix(int[][] mat, int i, int j) {
```

```
    int size = mat.length - 1;
    int[][] result = new int[size][size];
    for (int row = 0, newRow = 0; row < mat.length; row++) {
        if (row == i) continue;
        for (int col = 0, newCol = 0; col < mat[0].length; col++) {
            if (col == j) continue;
            result[newRow][newCol++] = mat[row][col];
        }
        newRow++;
    }
    return result;
}

public boolean checkSquareMatrix(int[][] mat) {
    for (int[] row : mat) {
        if (row.length != mat.length) return false;
    }
    return true;
}

public int findDeterminant(int[][] mat) {
    if (!checkSquareMatrix(mat)) return -1;
    if (mat.length == 2) {
        return (mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]) % m;
    }
    int det = 0;
    for (int c = 0; c < mat.length; c++) {
        det += ((c % 2 == 0 ? 1 : -1) * mat[0][c] * findDeterminant(removeCopyMatrix(mat, 0, c))) % m;
    }
    return det < 0 ? det + m : det;
}

public int[][] findAdjoint(int[][] mat) {
    int size = mat.length;
    int[][] adjoint = new int[size][size];
    if (size == 2) {
        adjoint[0][0] = mat[1][1];
        adjoint[1][1] = mat[0][0];
        adjoint[0][1] = -mat[0][1];
        adjoint[1][0] = -mat[1][0];
        return adjoint;
```

```
    }
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            int[][] minor = removeCopyMatrix(mat, i, j);
            adjoint[j][i] = ((i + j) % 2 == 0 ? 1 : -1) * findDeterminant(minor) % m;
            if (adjoint[j][i] < 0) adjoint[j][i] += m;
        }
    }
    return adjoint;
}

public int[][] scalarMultiply(int scalar, int[][] mat) {
    for (int i = 0; i < mat.length; i++) {
        for (int j = 0; j < mat[i].length; j++) {
            mat[i][j] = (scalar * mat[i][j]) % m;
            if (mat[i][j] < 0) mat[i][j] += m;
        }
    }
    return mat;
}

public String encrypt(String plaintext, int[][] keyMatrix) {
    int[][] plaintextMatrix = convertStrMat(plaintext, keyMatrix.length);
    int[][] cipherMatrix = findMatMul(keyMatrix, plaintextMatrix);
    return convertMatStr(cipherMatrix);
}

public String decrypt(String ciphertext, int[][] keyMatrix) {
    int det = findDeterminant(keyMatrix);
    if (det == 0 || findGCD(det, m) != 1) {
        throw new IllegalArgumentException("Matrix is not invertible");
    }
    int detInv = extendedEuclids(det, m);  // Use m instead of 1 as the third argument
    int[][] adjointMatrix = findAdjoint(keyMatrix);
    int[][] inverseMatrix = scalarMultiply(detInv, adjointMatrix);
    int[][] ciphertextMatrix = convertStrMat(ciphertext, inverseMatrix.length);
    int[][] plaintextMatrix = findMatMul(inverseMatrix, ciphertextMatrix);
    return convertMatStr(plaintextMatrix).replaceAll("X*$", "");
}

public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
String alphaList = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
HillCipher hillCipher = new HillCipher(alphaList.length(), alphaList);

while (true) {
    System.out.println("\nHill Cipher Menu");
    System.out.println("1. Encrypt");
    System.out.println("2. Decrypt");
    System.out.println("3. Exit");
    System.out.print("Enter your choice: ");
    String choice = scanner.nextLine();

    if (choice.equals("1")) {
        System.out.print("Enter the plaintext: ");
        String plaintext = scanner.nextLine().toUpperCase();
        System.out.print("Enter the key matrix size (e.g., 2 for 2x2, 3 for 3x3): ");
        int keyMatrixSize = scanner.nextInt();
        int[][] keyMatrix = new int[keyMatrixSize][keyMatrixSize];
        System.out.println("Enter the key matrix row-wise:");
        for (int i = 0; i < keyMatrixSize; i++) {
            for (int j = 0; j < keyMatrixSize; j++) {
                keyMatrix[i][j] = scanner.nextInt();
            }
        }
        scanner.nextLine();  // Consume newline
        String ciphertext = hillCipher.encrypt(plaintext, keyMatrix);
        System.out.println("Encrypted text: " + ciphertext);

    } else if (choice.equals("2")) {
        System.out.print("Enter the ciphertext: ");
        String ciphertext = scanner.nextLine().toUpperCase();
        System.out.print("Enter the key matrix size (e.g., 2 for 2x2, 3 for 3x3): ");
        int keyMatrixSize = scanner.nextInt();
        int[][] keyMatrix = new int[keyMatrixSize][keyMatrixSize];
        System.out.println("Enter the key matrix row-wise:");
        for (int i = 0; i < keyMatrixSize; i++) {
            for (int j = 0; j < keyMatrixSize; j++) {
                keyMatrix[i][j] = scanner.nextInt();
            }
        }
        scanner.nextLine();
```

```
            try {
                String plaintext = hillCipher.decrypt(ciphertext, keyMatrix);
                System.out.println("Decrypted text: " + plaintext);
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }

        } else if (choice.equals("3")) {

            break;
        } else {
            System.out.println("Invalid choice. Please try again.");          }}
    scanner.close();    }}
```

```
Hill Cipher Menu
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter the plaintext: gokul
Enter the key matrix size (e.g., 2 for 2x2, 3 for 3x3): 2
Enter the key matrix row-wise:
5
3
1
2
Encrypted text: UIGYUF

Hill Cipher Menu
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the ciphertext: UIGYUF
Enter the key matrix size (e.g., 2 for 2x2, 3 for 3x3): 2
Enter the key matrix row-wise:
5
3
1
2
Decrypted text: GOKUL
```

Known Plain test – cipher text attack

```
public class Decryptors {
    private final int m;
    private final String alphaList;
    private final HillCipher helpers;

    public Decryptors(int m, String alphaList) {
        this.m = m;
        this.alphaList = alphaList;
        this.helpers = new HillCipher(m, alphaList);
    }

    public int[][] knowPtCt(String pt, String ct, int size) {
        int[][] ptMatrix = helpers.convertStrMat(pt, size);
        System.out.println("Plain text matrix is: ");
        printMatrix(ptMatrix);
        int[][] ctMatrix = helpers.convertStrMat(ct, size);
        System.out.println("Cipher text matrix is: ");
        printMatrix(ctMatrix);
        int detCt = helpers.findDeterminant(ctMatrix);
        System.out.println("Determinant of cipher text matrix is: " + detCt);
        int detInv = helpers.extendedEuclids(detCt, m);
        System.out.println("The inverse of " + detCt + " is " + detInv);
        int[][] adjCt = helpers.findAdjoint(ctMatrix);
        System.out.println("The adjoint of cipher text matrix is: ");
        printMatrix(adjCt);
        int[][] ctInv = helpers.scalarMultiply(detInv, adjCt);
        System.out.println("The inverse of cipher text matrix is: ");
        printMatrix(ctInv);
        int[][] keyInv = helpers.findMatMul(ptMatrix, ctInv);
        System.out.println("The inverse of key matrix is: ");
        printMatrix(keyInv);
        return keyInv;
    }

    public String findPlainText(int[][] keyInv, String ct) {
        int[][] ctMatrix = helpers.convertStrMat(ct, keyInv.length);
        int[][] plainTextMatrix = helpers.findMatMul(keyInv, ctMatrix);
        return helpers.convertMatStr(plainTextMatrix);
    }

    private void printMatrix(int[][] matrix) {
```

```
        for (int[] row : matrix) {
            for (int val : row) {
                System.out.print(val + " ");
            }
            System.out.println();
        }}
    public static void main(String[] args) {
        Decryptors decryptor = new Decryptors(26, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");
        int[][] keyInv = decryptor.knowPtCt("JAMESBOND", "TEPHBNXSW", 3);
        String cipherText = "FBRTLWUGA";
        String originalText = decryptor.findPlainText(keyInv, cipherText);
        System.out.println("Decrypted text: " + originalText);
    }
}
```

```
C:\Users\gokul\.jdks\openjdk-22.0.1\bin\jav
Plain text matrix is:
9 4 14
0 18 13
12 1 3
Cipher text matrix is:
19 7 23
4 1 18
15 13 22
Determinant of cipher text matrix is: 21
The inverse of 21 is 5
The adjoint of cipher text matrix is:
22 15 25
0 21 10
11 14 17
The inverse of cipher text matrix is:
6 23 21
0 1 24
3 18 7
The inverse of key matrix is:
18 21 19
13 18 3
3 19 11
Decrypted text: SENDROSES
```

RESULT:

Thus the programs for

- Encryption
- Decryption
- Known Plain text – Cipher text attack

in Hill Cipher are implemented in Java and the results are verified.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	7	
Use of Comment lines and standard coding practices	3	
Viva	10	
Sub Total	20	
Completion of experiment on time	3	
Documentation	7	
Sub Total	10	
Signature of the faculty with Date		