Introduction

Hill cipher Program in JAVA (Z_{256})

The Hill cipher is a classic encryption technique that operates on blocks of plaintext and transforms them into ciphertext using matrix multiplication. In this document, we present a Java implementation of the Hill cipher with a key matrix size of 2x2. This program offers both encryption and decryption functionalities, allowing users to securely encode and decode messages.

The Hill cipher's strength lies in its use of linear algebra principles to manipulate plaintext, making it resistant to traditional frequency analysis attacks. By leveraging a 2x2 key matrix, our implementation offers a balance between simplicity and security, making it suitable for educational purposes and small-scale encryption tasks.

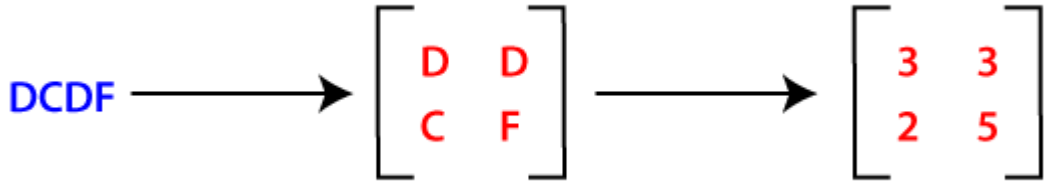
In the sections that follow, we provide a detailed overview of the functionality, usage, and implementation of our Hill cipher program in Java, along with examples to demonstrate its effectiveness in securing communication

Program Functionality

1. **Input Text File Handling:**
    - The program starts by taking an input text file, typically named `file.txt`, containing the plaintext message that the user wishes to encrypt. This file is read and processed by the program to retrieve the plaintext.
2. **Encryption Process:**
    - Upon reading the input text file, the program applies the encryption function of the Hill cipher algorithm to transform the plaintext into ciphertext.
    - The encryption function involves breaking the plaintext into blocks of appropriate size (in this case, 2 characters per block for a 2x2 key matrix), and then performing matrix multiplication with the key matrix to produce the corresponding ciphertext blocks.
3. **Output Result:**
    - Once the encryption process is complete, the program returns the resulting ciphertext. This ciphertext typically comprises alphanumeric characters, representing the encrypted version of the original plaintext message.

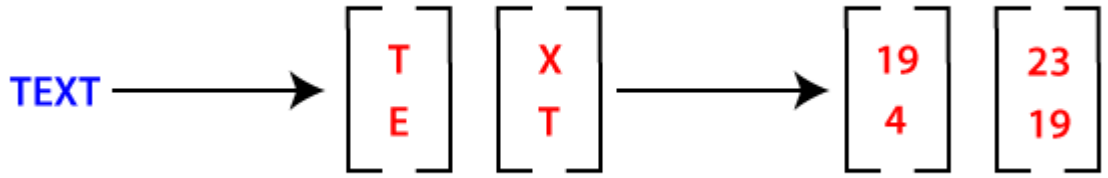
Steps For Encryption

Step 1: Let's say our key text (2x2) is **DCDF**. Convert this key using a substitution scheme into a 2x2 key matrix as shown below:



Step 2: Now, we will convert our plain text into vector form. Since the key matrix is 2x2, the vector must be 2x1 for matrix multiplication. (Suppose the key matrix is 3x3, a vector will be a 3x1 matrix.)

In our case, plain text is **TEXT** that is four letters long word; thus we can put in a 2x1 vector and then substitute as:



Step 3: Multiply the key matrix with each 2x1 plain text vector, and take the modulo of result (2x1 vectors) by 256. Then concatenate the results, and we get the encrypted or ciphertext as **RGWL**.

The algorithme is : E(K, P) = (K * P) mod 256

Steps For Decryption

Step 1: Calculate the inverse of the key matrix. First, we need to find the determinant of the key matrix (must be between 0-255). Here the Extended Euclidean algorithm is used to get modulo multiplicative inverse of key matrix determinant

Step 2: Now, we multiply the 2x1 blocks of ciphertext and the inverse of the key matrix. The resultant block after concatenation is the plain text that we have encrypted i.e., **TEXT**.

The Code

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class Main {
	//------------------------------------------------------------------------
	static String alphabet
="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
!#$%&'()*+,-./:;<=>?@[\\]^_`{|}~√↠∞▶▲◀●☀☢☠☞☜☝☑☐⊙☆★☄☃☂☘☣☪☮
☯☸☹☺☻☼☽☾♚♛♜♝♞♟♠♥♦♣⌨\uFE0E✏\uFE0E✒\uFE0E✉\uFE0E✂\uFE0
E⛏\uFE0E⚒\uFE0E⚔\uFE0E⚙\uFE0E⚖\uFE0E□□□□□□□□↴↵↶↼⇇⇠↞↩⇚⇛↕↖↗↘↙↮↠
↖↝↫↟↠↡↚↛↥↦↧↕↩↪↫↬↭↯↰↱↲↳↴↵↶↷↸↹↺↻↼↽↾↿⇀⇁⇂⇃⇄⇅⇆⇇⇈⇉⇊⇋⇌⇍⇎⇏⇕⇖⇗⇘⇙⇚⇛⇜⇝↭↕↞↠"
;
	//------------------------------------------------------------------------
	static String readFile(String fileName) {
	StringBuilder content = new StringBuilder();
	try {
	File file = new File(fileName);
	Scanner scanner = new Scanner(file);

	// Read the file line by line
	while (scanner.hasNextLine()) {
	String line = scanner.nextLine();
	content.append(line);

	}
	scanner.close();
	} catch (FileNotFoundException e) {
	System.out.println("File not found: " + fileName);
	e.printStackTrace();
	}
	return content.toString();
	}
	//------------------------------------------------------------------------
	static int [][] get_key_matrix(String key){
	int [][] key_matrix = new int[2][2];
	int index = 0;
	for (int i=0;i<2;i++){
	for (int j=0;j<2;j++){
	key_matrix[j][i] = (alphabet.indexOf(key.charAt(index++))+1)%256;
	}
	}
	return key_matrix;
	}
	//------------------------------------------------------------------------
	static int det_matrix(int [][] key_matrix ){
```

```
        int det = ((key_matrix[0][0]*key_matrix[1][1])-(key_matrix[1][0]*key_matrix[0][1]))%256;
        if (det<0){
        det+=256;
        }
        return det;
        }
        //------------------------------------------------------------------------
        static String hill_cipher_encrypt(String plainText, String key){
        String cipher ="";
        int [][] key_matrix= get_key_matrix(key);
        System.out.println("The index of character of the cipher Text : ");
        for (int i=0;i<plainText.length();i=i+2){
        int c1 = (key_matrix[0][0]*(alphabet.indexOf(plainText.charAt(i))+1) +
key_matrix[0][1]*(alphabet.indexOf(plainText.charAt(i+1))+1)) % 256;
        int c2 = (key_matrix[1][0]*(alphabet.indexOf(plainText.charAt(i))+1) +
key_matrix[1][1]*(alphabet.indexOf(plainText.charAt(i+1))+1)) % 256;
        System.out.println("C"+(i+1)+": "+c1+"  C"+(i+2)+": "+c2);
        cipher += (char) alphabet.charAt(c1-1);
        cipher += (char) alphabet.charAt(c2-1);
        }
        return cipher;
        }
        //------------------------------------------------------------------------
        static int gcd(int a, int b) {
        while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
        }
        return a;
        }
        //------------------------------------------------------------------------
        static int findMultiplicativeInverse(int a) {
        if (a == 0) {
        throw new IllegalArgumentException("Cannot compute multiplicative inverse of zero.");
        }

        int m = 256; // Modulus
        int m0 = m, t, q;
        int x0 = 0, x1 = 1;

        if (m == 1)
        return 0;
        if (gcd(a, m) != 1) {
        throw new IllegalArgumentException("Input parameter and modulus are not coprime.");
        }

        // Apply extended Euclid Algorithm
```

```
while (a > 1) {
// q is quotient
q = a / m;
t = m;

// m is remainder now, process same as
// euclid's algo
m = a % m;
a = t;
t = x0;

x0 = x1 - q * x0;
x1 = t;
}

// Make x1 positive
if (x1 < 0)
x1 += m0;

return x1;
}
//------------------------------------------------------------------------
static int [][] matrix_inverse(int [][] matrix){
int det = det_matrix(matrix);
int inverse_det = findMultiplicativeInverse(det);
int [][] inverse_matrix = new int[2][2];
inverse_matrix [0][0] = (inverse_det*matrix[1][1])%256;
inverse_matrix [0][1] = (inverse_det*(-matrix[0][1]))%256;
inverse_matrix [1][0] = (inverse_det*(-matrix[1][0]))%256;
inverse_matrix [1][1] = (inverse_det*matrix[0][0])%256;
for (int i = 0; i < 2; i++) {
for (int j = 0; j < 2; j++) {
if (inverse_matrix[i][j] < 0) {
        inverse_matrix[i][j] += 256;
}
}
}
return inverse_matrix;
}
//------------------------------------------------------------------------
static String hill_cipher_decrypt(String cipherText, String key){
int [][] inverse_matrix = matrix_inverse(get_key_matrix(key));
String plainText ="";
System.out.println("The index of character of the plain Text : ");
for (int j=0;j<cipherText.length();j=j+2){
int p1 = (inverse_matrix[0][0]*(alphabet.indexOf(cipherText.charAt(j))+1) +
inverse_matrix[0][1]*(alphabet.indexOf(cipherText.charAt(j+1))+1)) % 256;
```

```
        int p2 = (inverse_matrix[1][0]*(alphabet.indexOf(cipherText.charAt(j))+1) +
inverse_matrix[1][1]*(alphabet.indexOf(cipherText.charAt(j+1))+1)) % 256;
        System.out.println("P"+(j+1)+": "+p1+"  P"+(j+2)+": "+p2);
        plainText += (char) alphabet.charAt(p1-1);
        plainText += (char) alphabet.charAt(p2-1);
        }
        return plainText;
        }

        public static void main(String[] args) {
        String filename = "/home/kali/IdeaProjects/Hill_Cipher/file.txt";
        String PlainText = readFile(filename);
        if ((PlainText.length() % 2)!=0){
        PlainText += "Y";
        }
        //------------------------------------------------------------------------
        System.out.println("The alphabet length is : "+ alphabet.length());
        System.out.print("Entre The Key of 4 characters :");
        Scanner scan = new Scanner(System.in);
        String key = scan.next();
        //------------------------------------------------------------------------
        if (key.length()!=4){
        System.out.println("Please enter key of 4 characters");
        }
        else{
        //------------------------------------------------------------------------
        int [][] key_matrix = get_key_matrix(key);
        int det = det_matrix(key_matrix);
        //------------------------------------------------------------------------
        if ((det!=0) && (gcd(det,256)==1)){
        //------------------------------------------------------------------------
        int [][] inverse_key_matrix = matrix_inverse(key_matrix);
        //------------------------------------------------------------------------
        System.out.println("-----------------------------------------------------");
        System.out.println("The PlainText is : "+PlainText);
        System.out.println("-----------------------------------------------------");
        String cipherText = hill_cipher_encrypt(PlainText,key);
        System.out.println("The Cipher Text is : "+cipherText);
        System.out.println("-----------------------------------------------------");
        System.out.println("Information of The Attack : ");
        System.out.println("-----------------------------------------------------");
        System.out.println("the key matrix is :");
        System.out.println("|"+key.charAt(0)+" "+key.charAt(2)+"|"+"--->"+"|"+key_matrix[0][0]+"
"+key_matrix[0][1]+"|");
        System.out.println("|"+key.charAt(1)+" "+key.charAt(3)+"|"+"--->"+"|"+key_matrix[1][0]+"
"+key_matrix[1][1]+"|");
        System.out.println("-----------------------------------------------------");
        System.out.println("The determinate of the key matrix is : "+det);
```

```
        System.out.println("The key is valid :)");
        System.out.println("-----------------------------------------------------");
        System.out.println("the inverse of the key matrix is :");
        System.out.println("|"+inverse_key_matrix[0][0]+" "+inverse_key_matrix[0][1]+"|");
        System.out.println("|"+inverse_key_matrix[1][0]+" "+inverse_key_matrix[1][1]+"|");
        System.out.println("-----------------------------------------------------");
        System.out.println("The determinate of the inverse key matrix is :
"+findMultiplicativeInverse(det));
        System.out.println("-----------------------------------------------------");
        String plainTexto = hill_cipher_decrypt(cipherText,key);
        System.out.println("The Plain Text is : "+plainTexto);
        }else{
        System.out.println("The key is not valid :( ,Please Try Again");
        }
        }
        }
}
```