**Rail-Fence Cipher Implementation**

1. **Using any programming language of your choice implement the Rail Fence Cipher encryption algorithm ?**

Java Implementation of Rail Fence Cipher Encryption Algorithm:

import java.util.Scanner;

public class EncryptionCode {

static String pText="";

static int r, depth;

static char[][] matrix = new char[depth][];

public static void main(String[] args) {

String cText = "";

Scanner input = new Scanner(System.in);

System.out.println("Please enter the Plain text:");

pText = input.nextLine().toLowerCase();

System.out.println("Please enter the Key: ");

System.out.println("Key - Depth: ");

depth = input.nextInt();

System.out.println("Key - r (Number of times to do encryption): ");

r = input.nextInt();

matrix = new char[depth][pText.length()];

initialise\_the\_matrix(depth, matrix);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Rail Fence Cipher Encryption Method");

System.out.println();

System.out.println("Original Plain Text: " + pText);

System.out.println();

//Number of rounds(times) for which encryption will be done.

for(int i=0; i< r; i++)

{

populate\_the\_matrix(depth, matrix);

cText = traverse\_the\_matrix(depth,matrix);

System.out.println("Cipher text after round " + (i+1) + ": " + cText);

pText = cText;

}

}

//This function initializes the all the cells of the matrix with the flag "#".

private static void initialise\_the\_matrix(int d, char[][] m)

{

int row, col;

for(row=0;row<depth;row++)

{

for(col=0;col<pText.length();col++)

{

m[row][col]='#';

}

}

}

/\* This function populates the matrix with the letters of the plain text.\*/

private static void populate\_the\_matrix(int d, char[][] m)

{

String result="";

int itr=0;

int direction = 0, value = 1, row, col;

for(row=0,col=0; col<pText.length(); col++)

{

if(direction==0 && row==d)

{

direction=1;

value=-1;

row=row+value-1;

}

if(direction==1 && row==-1)

{

direction=0;

value=1;

row=row+value+1;

}

m[row][col]=pText.charAt(itr);

itr++;

row=row+value;

}

}

/\*This function traverses the matrix in row-wise manner to generate and return the cipher text.\*/

private static String traverse\_the\_matrix(int d, char[][] m)

{

String res="";

int itr=0, row,col;

for(row=0;row<depth;row++)

{

for(col=0;col<m[row].length;col++)

{

if(m[row][col]!='#')

{

res=res + m[row][col];

}

}

}

return res;

}

}

1. **What will be the decryption algorithm for a Rail-Fence cipher? Write down the pseudocode for the algorithm?**

Algorithm – Rail-Fence Cipher Encryption

In the Rail-fence Cipher Encryption Algorithm, we maintain a 2-D matrix which is used to generate the cipher text.

Step 1: Firstly, the plain-text is written down as a sequence of diagonals in the matrix (basically, in the zig-zag fashion).

Step 2: The matrix is traversed row-wise (one row at a time) to generate the cipher text.

Step 3: After each row is traversed, all the rows are combined to obtain the cipher-text.

For Example:

Original Message - We are group 1

Depth – 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| w |  | a |  | e |  | r |  | u |  | 1 |
|  | e |  | r |  | g |  | o |  | p |  |

So, the cipher-text generated will be - **waeru1ergop**

Algorithm – Rail-Fence Cipher Decryption

In the Rail-fence Cipher Decryption Algorithm, we maintain a 2-D matrix to generate the corresponding plain text.

Step 1: Firstly, we will find the location (in the zig-zag fashion that is as a sequence of diagonals) to be filled in with the letters of the cipher text in the matrix. (Moving diagonally down and then diagonally upwards, alternatively).

Step 2: Then fill in the cipher-text letters row wise on the locations found in step 1.

Step 3: After filling the text in the matrix, we traverse the matrix again in a zig-zag manner (Moving diagonally down and then diagonally upwards, alternatively) to obtain the original message.

For Example:

Cipher Text- **waeru1ergop**

Depth- 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **w** |  | **a** |  | **e** |  | **r** |  | **u** |  | **1** |
|  | e |  | r |  | g |  | o |  | p |  |

Therefore, the original text will be,

**We are group 1**

Pseudocode for Rail Fence Cipher Decryption Algorithm

Function Main

*ctext* corresponds to the given cipher text

*r* is the number of times the algorithm will execute for a single test case

*depth* is the number of rows in the *matrix*

for each i = 0 to r

result ← decrypt(depth, matrix)

ctext ← result

End Function

/\*This function returns the decrypted text that is the plain text.\*/

Function decrypt (depth, matrix)

call the function find\_the\_location\_in\_the\_matrix(depth,matrix)

call the function populate\_the\_matrix(depth,matrix)

decrypted\_text ← traverse\_the\_matrix(depth,matrix)

return decrypted\_text

End Function

/\*This function finds the locations in the matrix where the letters of the cipher text should be placed and flags them with the character ‘#’. \*/

Function find\_the\_location\_in\_the\_matrix(depth,matrix)

Initialize direction ← 0 and value ← 1

for row = 0, col = 0 to col < ctext.length

if direction == 0 and row == d then

direction ← 1

value ← -1

row ← row + value - 1

if direction == 1 and row == -1 then

direction ← 0

value ← 1

row ← row + value + 1

matrix[row][col] ← ‘#’

row ← row + value

End Function

/\*This function puts the elements of the cipher text in the matrix row-wise (that is each row at a time). \*/

Function populate\_the\_matrix(depth,matrix)

Initialize itr ← 0

for row = 0 to row < depth

for col = 0 to matrix[row].length

if matrix[row][col] == ‘#’ then

matrix[row][col] ← ctext.charAt(itr)

itr ← itr + 1

End Function

/\*This function traverses the matrix to generate the corresponding plain text in the populated matrix. \*/

Function traverse\_the\_matrix(depth,matrix)

Initialize direction ← 0 and value ← 1

for row = 0, col = 0 to col < ctext.length

if direction == 0 and row == d then

direction ← 1

value ← -1

row ← row + value - 1

if direction == 1 and row == -1 then

direction ← 0

value ← 1

row ← row + value + 1

result ← result + matrix[row][col]

row ← row + value

return result

End Function

1. **Using any programming language of your choice implement the Rail Fence Cipher**

**decryption algorithm.**

Java Implementation of Rail Fence Cipher Decryption Algorithm:

import java.util.Scanner;

public class DecryptionCode {

static int depth, r;

static String ctext;

static char[][] matrix = new char[depth][];

public static void main(String[] args) {

int i;

String res;

Scanner input = new Scanner(System.in);

System.out.println("Please enter the Cipher text: ");

ctext = input.nextLine().toLowerCase();

System.out.println("Please enter the Key: ");

System.out.println("Key - Depth: ");

depth = input.nextInt();

System.out.println("Key - r (Number of times to do decryption): ");

r = input.nextInt();

matrix = new char[depth][ctext.length()];

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Rail Fence Cipher Decryption Method");

System.out.println();

System.out.println("Original Cipher Text: " + ctext);

System.out.println();

System.out.println("Decryption of the Cipher Text in various rounds is shown below:");

System.out.println();

for(i=0;i<r;i++)

{

res = decrypt(depth, matrix);

System.out.println("Decrypted text after round " + (i+1) + ": " + res);

ctext=res;

System.out.println();

}

}

/\*This function returns the decrypted text.\*/

public static String decrypt(int d, char[][] m)

{

String decrypted\_text;

find\_the\_location\_in\_the\_matrix(d, m);

populate\_the\_matrix(d, m);

decrypted\_text = traverse\_the\_matrix(d, m);

return decrypted\_text;

}

/\*This function finds the locations where we will place the elements of the cipher text and mark them by the flag '#'.\*/

public static void find\_the\_location\_in\_the\_matrix(int d, char[][] m)

{

int direction = 0, value = 1, row, col;

for(row=0,col=0; col<ctext.length(); col++)

{

if(direction==0 && row==d)

{

direction=1;

value=-1;

row=row+value-1;

}

if(direction==1 && row==-1)

{

direction=0;

value=1;

row=row+value+1;

}

m[row][col]= '#';

row=row+value;

}

}

/\*This function populates the matrix with the letters of the cipher text\*/

public static void populate\_the\_matrix(int d, char[][] m)

{

int itr=0, row,col;

for(row=0;row<depth;row++)

{

for(col=0;col<m[row].length;col++)

{

if(m[row][col]=='#')

{

m[row][col]=ctext.charAt(itr);

itr++;

}

}

}

}

/\*This function traverses the matrix in the zig-zag manner to find the decrypted text.\*/

public static String traverse\_the\_matrix(int d, char[][] m)

{

int direction = 0, value = 1, row, col;

String result="";

for(row=0,col=0; col<ctext.length(); col++)

{

if(direction==0 && row==d)

{

direction=1;

value=-1;

row=row+value-1;

}

if(direction==1 && row==-1)

{

direction=0;

value=1;

row=row+value+1;

}

result = result + m[row][col];

row=row+value;

}

return result;

}

}

1. **Test 1 (Encryption)**

Input:

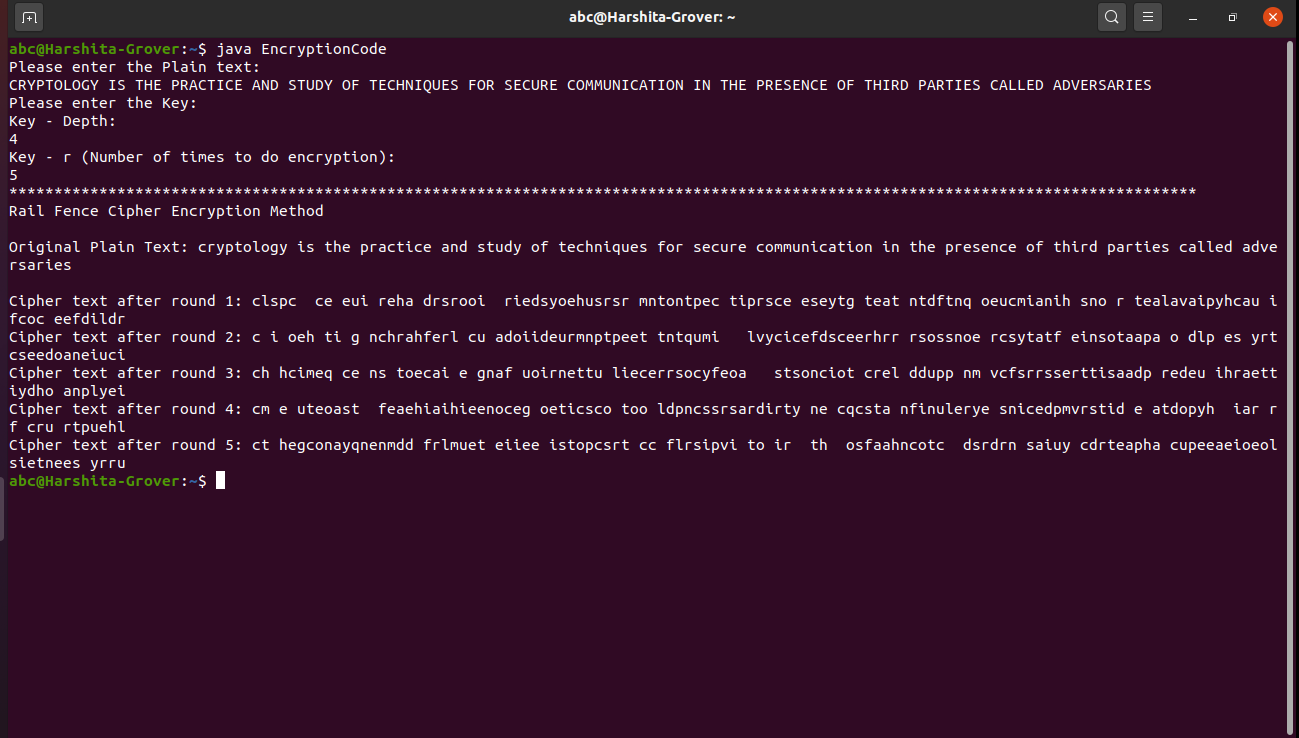
Plain text =

CRYPTOLOGY IS THE PRACTICE AND STUDY OF TECHNIQUES FOR SECURE COMMUNICATION IN THE PRESENCE OF THIRD PARTIES CALLED ADVERSARIES

Depth = 4

R (Number of times to execute the algorithm) = 5

Output:



Final Answer (The cipher text generated after round 5):

ct hegconayqnenmdd frlmuet eiiee istopcsrt cc flrsipvi to ir th osfaahncotc dsrdrn saiuy cdrteapha cupeeaeioeolsietnees yrru

1. **Test 2 (Decryption)**

Input:

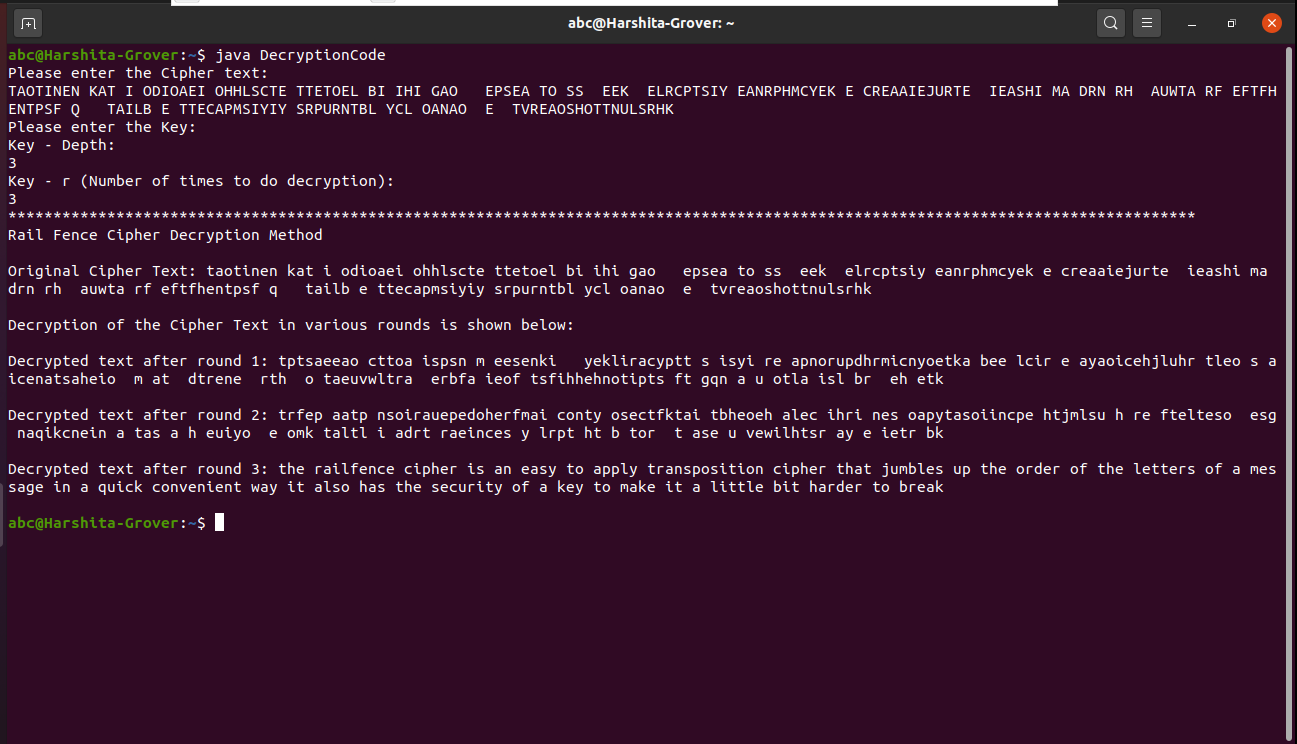
Cipher Text =

TAOTINEN KAT I ODIOAEI OHHLSCTE TTETOEL BI IHI GAO   EPSEA TO SS  EEK  ELRCPTSIY EANRPHMCYEK E CREAAIEJURTE  IEASHI MA DRN RH  AUWTA RF EFTFHENTPSF Q   TAILB E TTECAPMSIYIY SRPURNTBL YCL OANAO  E  TVREAOSHOTTNULSRHK

Depth = 3

R (Number of times to execute the algorithm) = 3

Output:



Final Answer (The plain text obtained after round 3 of decryption):

the railfence cipher is an easy to apply transposition cipher that jumbles up the order of the letters of a message in a quick convenient way it also has the security of a key to make it a little bit harder to break