ASSIGNMENT 1

AIM: Implement any 4 substitution and 2 transposition classical encryption technique in any programming language.

SUBSTITUTION ALGORITHMS

● CAESAR CIPHER

INPUT:

class Main {

public static StringBuffer encrypt(String text, int s)

{

StringBuffer result= new StringBuffer();

for (int i = 0; i < text.length(); i++)

{

if (Character.isUpperCase(text.charAt(i)))

{

char ch = (char)(((int)text.charAt(i) +

s - 65) % 26 + 65);

result.append(ch);

}

else

{

char ch = (char)(((int)text.charAt(i) +

s - 97) % 26 + 97);

result.append(ch);

}

}

return result;

}

static StringBuffer decrypt(StringBuffer encryptedText, int s) {

StringBuffer result = new StringBuffer();

for (int i = 0; i < encryptedText.length(); i++) {

if (Character.isUpperCase(encryptedText.charAt(i))) {

char ch = (char) (((int) encryptedText.charAt(i) -

s - 65 + 26) % 26 + 65);

result.append(ch);

} else {

char ch = (char) (((int) encryptedText.charAt(i) -

s - 97 + 26) % 26 + 97);

result.append(ch);

}

}

return result;

}

public static void main(String[] args)

{

String text = "YOURNAME";

int s = 3;

System.out.println("Text : " + text);

System.out.println("Key : " + s);

StringBuffer encryptedText = encrypt(text, s);

System.out.println("Cipher: " + encryptedText);

int s1 = 3;

System.out.println("Encrypted Text: " + encryptedText);

System.out.println("Key : " + s1);

System.out.println("Decrypted Text: " + decrypt(encryptedText, s1));

}

}

OUTPUT:

● POLYALPHABETIC CIPHER

INPUT

class HelloWorld {

static String generateKey(String str, String key)

{

int x = str.length();

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

{

if (x == i)

i = 0;

if (key.length() == str.length())

break;

key+=(key.charAt(i));

}

return key;

}

static String cipherText(String str, String key)

{

String cipher\_text="";

for (int i = 0; i < str.length(); i++)

{

// converting in range 0-25

int x = (str.charAt(i) + key.charAt(i)) %26;

// convert into alphabets(ASCII)

x += 'A';

cipher\_text+=(char)(x);

}

return cipher\_text;

}

static String originalText(String cipher\_text, String key)

{

String orig\_text="";

for (int i = 0 ; i < cipher\_text.length() &&

i < key.length(); i++)

{

// converting in range 0-25

int x = (cipher\_text.charAt(i) -

key.charAt(i) + 26) %26;

// convert into alphabets(ASCII)

x += 'A';

orig\_text+=(char)(x);

}

return orig\_text;

}

static String LowerToUpper(String s)

{

StringBuffer str =new StringBuffer(s);

for(int i = 0; i < s.length(); i++)

{

if(Character.isLowerCase(s.charAt(i)))

{

str.setCharAt(i, Character.toUpperCase(s.charAt(i)));

}

}

s = str.toString();

return s;

}

public static void main(String[] args)

{

String Str = "YOURNAME";

String Keyword = "SEA";

String str = LowerToUpper(Str);

String keyword = LowerToUpper(Keyword);

String key = generateKey(str, keyword);

String cipher\_text = cipherText(str, key);

System.out.println("Ciphertext : "

+ cipher\_text + "\n");

System.out.println("Original Text : "

+ originalText(cipher\_text, key));

}

}

OUTPUT:

● VERNAM CIPHER

INPUT

import java.io.\*;

public class Main

{

public static String stringEncryption(String text,

String key)

{

String cipherText = "";

int cipher[] = new int[key.length()];

for (int i = 0; i < key.length(); i++) {

cipher[i] = text.charAt(i) - 'A'

+ key.charAt(i)

- 'A';

}

for (int i = 0; i < key.length(); i++) {

if (cipher[i] > 25) {

cipher[i] = cipher[i] - 26;

}

}

for (int i = 0; i < key.length(); i++) {

int x = cipher[i] + 'A';

cipherText += (char)x;

}

return cipherText;

}

public static String stringDecryption(String s,

String key)

{

String plainText = "";

int plain[] = new int[key.length()];

for (int i = 0; i < key.length(); i++) {

plain[i]

= s.charAt(i) - 'A'

- (key.charAt(i) - 'A');

}

for (int i = 0; i < key.length(); i++) {

if (plain[i] < 0) {

plain[i] = plain[i] + 26;

}

}

for (int i = 0; i < key.length(); i++) {

int x = plain[i] + 'A';

plainText += (char)x;

}

return plainText;

}

public static void main(String[] args)

{

String plainText = "YOURNAME";

String key = "MONEY";

String encryptedText = stringEncryption(

plainText.toUpperCase(), key.toUpperCase());

System.out.println("Cipher Text - "

+ encryptedText);

System.out.println(

"Message - "

+ stringDecryption(encryptedText,

key.toUpperCase()));

}

}

OUTPUT:

TRANSPOSITION ALGORITHMS

● COLUMNAR TRANSPOSITION

INPUT

import java.util.Scanner;

public class SimpleColumnarTransposition {

// Encryption function

public static String encrypt(String plaintext, String key) {

int keyLength = key.length();

int textLength = plaintext.length();

// Calculate the number of rows required in the matrix

int numRows = (int) Math.ceil((double) textLength / keyLength);

// Create a 2D array to hold the characters

char[][] matrix = new char[numRows][keyLength];

// Fill the matrix with the plaintext characters

int textIndex = 0;

for (int i = 0; i < numRows; i++) {

for (int j = 0; j < keyLength; j++) {

if (textIndex < textLength) {

matrix[i][j] = plaintext.charAt(textIndex);

textIndex++;

} else {

matrix[i][j] = ' ';

}

}

}

// Encrypt the message by reading columns according to the key

StringBuilder ciphertext = new StringBuilder();

for (int j = 0; j < keyLength; j++) {

int col = key.indexOf(key.charAt(j));

for (int i = 0; i < numRows; i++) {

ciphertext.append(matrix[i][col]);

}

}

return ciphertext.toString();

}

// Decryption function

public static String decrypt(String ciphertext, String key) {

int keyLength = key.length();

int textLength = ciphertext.length();

// Calculate the number of rows required in the matrix

int numRows = (int) Math.ceil((double) textLength / keyLength);

// Calculate the number of characters in the last row

int lastRowLength = textLength % keyLength;

if (lastRowLength == 0) {

lastRowLength = keyLength;

}

// Create a 2D array to hold the characters

char[][] matrix = new char[numRows][keyLength];

// Fill the matrix with the ciphertext characters

int textIndex = 0;

for (int j = 0; j < keyLength; j++) {

int col = key.indexOf(key.charAt(j));

for (int i = 0; i < numRows; i++) {

if (i == numRows - 1 && j >= lastRowLength) {

matrix[i][col] = ' ';

} else {

matrix[i][col] = ciphertext.charAt(textIndex);

textIndex++;

}

}

}

// Decrypt the message by reading rows

StringBuilder plaintext = new StringBuilder();

for (int i = 0; i < numRows; i++) {

for (int j = 0; j < keyLength; j++) {

plaintext.append(matrix[i][j]);

}

}

return plaintext.toString().trim();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the plaintext: ");

String plaintext = scanner.nextLine();

System.out.print("Enter the encryption key: ");

String key = scanner.nextLine();

// Encryption

String ciphertext = encrypt(plaintext, key);

System.out.println("Encrypted Message: " + ciphertext);

// Decryption

String decryptedText = decrypt(ciphertext, key);

System.out.println("Decrypted Message: " + decryptedText);

}

}

OUTPUT:

● RAIL FENCE TRANSPOSITION

INPUT:

public class RailFenceCipher {

// Function to encrypt a message using Rail Fence Transposition

static String encrypt(String message, int rails) {

// Create a 2D array to represent the rail fence structure

char[][] railFence = new char[rails][message.length()];

// Initialize the array with space characters

for (int i = 0; i < rails; i++) {

for (int j = 0; j < message.length(); j++) {

railFence[i][j] = ' ';

}

}

// Fill in the rail fence with the message characters

int row = 0;

boolean down = false;

for (int i = 0; i < message.length(); i++) {

railFence[row][i] = message.charAt(i);

// Change direction when reaching the top or bottom rail

if (row == 0 || row == rails - 1) {

down = !down;

}

// Move to the next row in the appropriate direction

if (down) {

row++;

} else {

row--;

}

}

// Read the encrypted message row by row

StringBuilder encryptedMessage = new StringBuilder();

for (int i = 0; i < rails; i++) {

for (int j = 0; j < message.length(); j++) {

if (railFence[i][j] != ' ') {

encryptedMessage.append(railFence[i][j]);

}

}

}

return encryptedMessage.toString();

}

public static String decryptRailFence(String cipherText, int rails) {

int textLength = cipherText.length();

char[][] railMatrix = new char[rails][textLength];

boolean down = false;

int row = 0, col = 0;

// Initialize the rail matrix with placeholders

for (int i = 0; i < rails; i++) {

for (int j = 0; j < textLength; j++) {

railMatrix[i][j] = ' ';

}

}

// Fill the rail matrix with the cipherText

for (int i = 0; i < textLength; i++) {

if (row == 0 || row == rails - 1) {

down = !down;

}

railMatrix[row][col] = '\*';

col++;

if (down) {

row++;

} else {

row--;

}

}

// Reconstruct the plainText

int index = 0;

char[] plainText = new char[textLength];

for (int i = 0; i < rails; i++) {

for (int j = 0; j < textLength; j++) {

if (railMatrix[i][j] == '\*' && index < textLength) {

plainText[j] = cipherText.charAt(index);

index++;

}

}

}

return new String(plainText);

}

public static void main(String[] args) {

String message = "HELLOWORLD";

int rails = 3;

String encryptedMessage = encrypt(message, rails);

System.out.println("Encrypted Message: " + encryptedMessage);

String decryptedText = decryptRailFence(cipherText, rails);

System.out.println("Decrypted Message: " + decryptedText);

}

}

OUTPUT :

CONCLUSION: We successfully implemented substitution and transposition encryption techniques using Java programming language.