ASSIGNMENT – 5

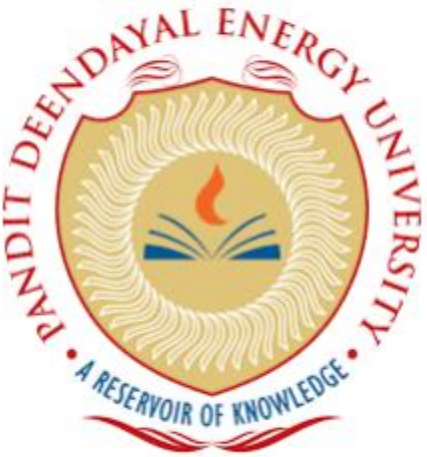
Information Security – Lab

Code:20CP304P

Name: Smit Sutariya

Roll no: 21BCP142

DIVISION 3 (Group 5)



COMPUTER ENGINEERING

School of Technology, Pandit Deendayal Energy University

Experiment No : 5

Aim : To understand Vigenère Cipher

Introduction :

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.

The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.

At different points in the encryption process, the cipher uses a different alphabet from one of the rows.

The alphabet used at each point depends on a repeating keyword.

Encryption

The plaintext(P) and key(K) are added modulo 26.

$E_i = (P_i + K_i) \bmod 26$

Decryption

$D_i = (E_i - K_i) \bmod 26$

Program (Source Code):

```
#include <iostream>
using namespace std;
string Encryption(string plain_text, string key)
{
  string result = "";
  int count = 0;
  for (int i = 0; i < plain_text.length(); i++)
  {
    if (count >= key.length())
    {
      count = 0;
    }
    if (isalpha(plain_text[i]))
    {
      if (islower(plain_text[i]))

        result +=(((plain_text[i] - 97)+((tolower(key[count])) - 97))%26)+97;
      if (isupper(plain_text[i]))

        result +=(((plain_text[i] - 65)+((toupper(key[count])) - 65))%26)+65;
      count++;
    }
    else
    {
      result += plain_text[i];
    }
  }
  return result;
}
string Decryption(string cipher_text, string key)
{
  string result = "";
  int count = 0;
  for (int i = 0; i < cipher_text.length(); i++)
  {
    if (count >= key.length())
    {
      count = 0;
    }
    if (isalpha(cipher_text[i]))
    {
      if (islower(cipher_text[i]))
```

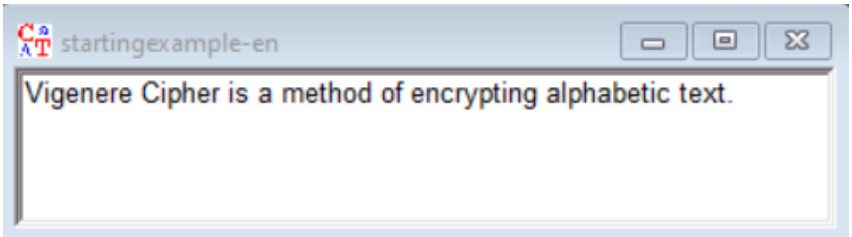
```
        result +=((((cipher_text[i] - 97)-((tolower(key[count])) -
97))+26)%26)+97;
      if (isupper(cipher_text[i]))

        result +=((((cipher_text[i] - 65)-((toupper(key[count])) -
65))+26)%26)+65;
      count++;
    }
    else
    {
      result += cipher_text[i];
    }
  }
  return result;
}
int main()
{
  string plain_text;
  string cipher_text,output_text;
  string key;
  cout << "Enter Plain Text: ";
  getline(cin, plain_text);
  cout << "Enter Key: ";
  cin >> key;
  cipher_text= Encryption(plain_text, key);
  cout<<"Encryption: "<<cipher_text<<endl;
  output_text=Decryption(cipher_text,key);
  cout<<"Decryption: "<<output_text;
}
```

Output(Program):

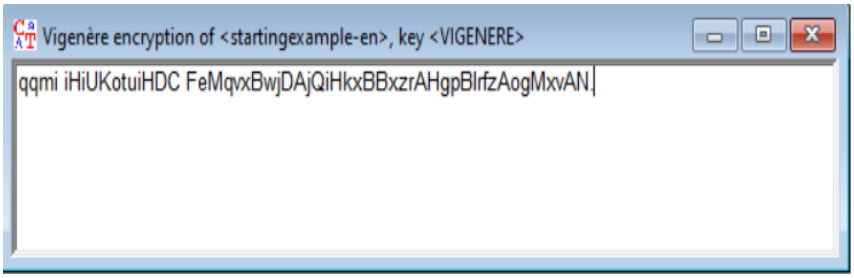
```
PS D:\Sem-5\Info.sec\Lab5> cd "d:\Sem-5\Info.sec\Lab5\" ;
Enter Plain Text: smitsutaria
Enter Key: smix
Encryption: kyqqkgbxjui
Decryption: smitsutaria
PS D:\Sem-5\Info.sec\Lab5>
```

Output (Cryptool):

Plain-Text:



Cipher-Text:



Cryptanalysis :

Frequency analysis

Once the length of the key is known, the ciphertext can be rewritten into that many columns, with each column corresponding to a single letter of the key. Each column consists of plaintext that has been encrypted by a single Caesar cipher. The Caesar key (shift) is just the letter of the Vigenère key that was used for that column. Using methods similar to those used to break the Caesar cipher, the letters in the ciphertext can be discovered.

Key elimination

The Vigenère cipher, with normal alphabets, essentially uses modulo arithmetic, which is commutative. Therefore, if the key length is known (or guessed), subtracting the cipher text from itself, offset by the key length, will produce the plain text subtracted from itself, also offset by the key length. If any "probable

word" in the plain text is known or can be guessed, its self-subtraction can be recognized, which allows recovery of the key by subtracting the known plaintext from the cipher text. Key elimination is especially useful against short messages

Applications :

The Vigenère cipher is a classical encryption technique that was invented by Blaise de Vigenère in the 16th century. It is a polyalphabetic substitution cipher, which means that it uses multiple substitution alphabets to encrypt plaintext, making it more secure than simple substitution ciphers like the Caesar cipher. Here are some applications and use cases of the Vigenère cipher:

Military and Diplomatic Communications: The Vigenère cipher was historically used for secure military and diplomatic communications. It provided a more secure way to encode messages compared to simpler ciphers.

Privacy in Correspondence: Before the advent of modern encryption techniques, the Vigenère cipher was used by individuals and organizations to protect the privacy of their correspondence. It was considered a relatively secure method for encrypting letters and messages.

Puzzles and Games: The Vigenère cipher has been used in puzzles and games, particularly in puzzle books and brain teasers. People often use it to create or solve cryptograms for entertainment and educational purposes.

Cryptanalysis Practice: In the field of cryptanalysis (the study of breaking codes and ciphers), the Vigenère cipher is often used as a practice exercise for students and enthusiasts to develop their skills in deciphering encrypted messages.

References :

1. Ref-1: https://www.geeksforgeeks.org/vigenere-cipher/

2. Ref-2: https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

EXPERIMENT NO: 6

Aim: Study and implement a program for 5*5 Playfair cipher.

Introduction: Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a **digraph** (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then **J** is replaced by **I**. It means treat I and J as the same letter, accordingly.

Algorithm to encrypt the plain text: The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter. If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).

If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Decrypting the Playfair cipher is as simple as doing the same process in reverse. The receiver has the same key and can create the same key table, and then decrypt any messages made using that key

Program Code:

```
import java.awt.Point;
import java.util.Scanner;

public class PlayfairCipher {
    private int length = 0;
    private String[][] table;

    public static void main(String args[]) {
        new PlayfairCipher().start();
    }

    private void start() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the key for Playfair cipher: ");
        String key = parseString(sc);
        while (key.equals("")) {
            System.out.println("Key cannot be empty. Please enter a valid key.");

        }
        table = this.cipherTable(key);

        System.out.print("Enter the plaintext to be encipher: ");
        String input = parseString(sc);
        while (input.equals("")) {
            System.out.println("Plaintext cannot be empty. Please enter a valid
plaintext.");

        }

        String output = cipher(input);
        String decodedOutput = decode(output);

        keyTable(table);
        printResults(output, decodedOutput);

        sc.close();
```

```
    }

    private String parseString(Scanner sc) {
        String parse = sc.nextLine();
        parse = parse.toUpperCase();
        parse = parse.replaceAll("[^A-Z]", "");
        parse = parse.replace("J", "I");
        return parse;
    }

    private String[][] cipherTable(String key) {
        String[][] playfairTable = new String[5][5];
        String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

        for (int i = 0; i < 5; i++)
            for (int j = 0; j < 5; j++)
                playfairTable[i][j] = "";

        for (int k = 0; k < keyString.length(); k++) {
            boolean repeat = false;
            boolean used = false;
            for (int i = 0; i < 5; i++) {
                for (int j = 0; j < 5; j++) {
                    if (playfairTable[i][j].equals("" + keyString.charAt(k))) {
                        repeat = true;
                    } else if (playfairTable[i][j].equals("") && !repeat &&
!used) {
                        playfairTable[i][j] = "" + keyString.charAt(k);
                        used = true;
                    }
                }
            }
        }
        return playfairTable;
    }

    private String cipher(String in) {
        length = (int) in.length() / 2 + in.length() % 2;

        for (int i = 0; i < (length - 1); i++) {
            if (in.charAt(2 * i) == in.charAt(2 * i + 1)) {
                in = new StringBuffer(in).insert(2 * i + 1, 'X').toString();
                length = (int) in.length() / 2 + in.length() % 2;
            }
        }
```

```
        String[] digraph = new String[length];

        for (int j = 0; j < length; j++) {
            if (j == (length - 1) && in.length() / 2 == (length - 1))
                in = in + "X";
            digraph[j] = in.charAt(2 * j) + "" + in.charAt(2 * j + 1);
        }

        String out = "";
        String[] encDigraphs = new String[length];
        encDigraphs = encodeDigraph(digraph);

        for (int k = 0; k < length; k++)
            out = out + encDigraphs[k];

        return out;
    }

    private String[] encodeDigraph(String di[]) {
        String[] encipher = new String[length];

        for (int i = 0; i < length; i++) {
            char a = di[i].charAt(0);
            char b = di[i].charAt(1);
            int r1 = (int) getPoint(a).getX();
            int r2 = (int) getPoint(b).getX();
            int c1 = (int) getPoint(a).getY();
            int c2 = (int) getPoint(b).getY();

            if (r1 == r2) {
                c1 = (c1 + 1) % 5;
                c2 = (c2 + 1) % 5;
            } else if (c1 == c2) {
                r1 = (r1 + 1) % 5;
                r2 = (r2 + 1) % 5;
            } else {
                int temp = c1;
                c1 = c2;
                c2 = temp;
            }

            encipher[i] = table[r1][c1] + "" + table[r2][c2];
        }
```

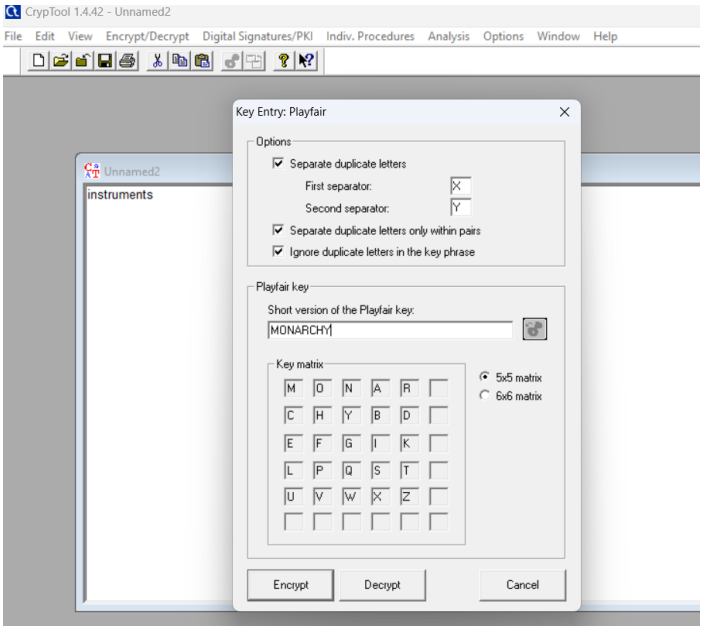
```
        return encipher;
    }

    private String decode(String out) {
        String decoded = "";

        for (int i = 0; i < out.length() / 2; i++) {
            char a = out.charAt(2 * i);
            char b = out.charAt(2 * i + 1);
            int r1 = (int) getPoint(a).getX();
            int r2 = (int) getPoint(b).getX();
            int c1 = (int) getPoint(a).getY();
            int c2 = (int) getPoint(b).getY();
            if (r1 == r2) {
                c1 = (c1 + 4) % 5;
                c2 = (c2 + 4) % 5;
            } else if (c1 == c2) {
                r1 = (r1 + 4) % 5;
                r2 = (r2 + 4) % 5;
            } else {
                int temp = c1;
                c1 = c2;
                c2 = temp;
            }

            decoded = decoded + table[r1][c1] + table[r2][c2];
        }
        return decoded;
    }

    private Point getPoint(char c) {
        Point pt = new Point(0, 0);

        for (int i = 0; i < 5; i++)
            for (int j = 0; j < 5; j++)
                if (c == table[i][j].charAt(0))
                    pt = new Point(i, j);

        return pt;
    }

    private void keyTable(String[][] printTable) {
        System.out.println("Playfair Cipher Key Matrix:");
        System.out.println();
```

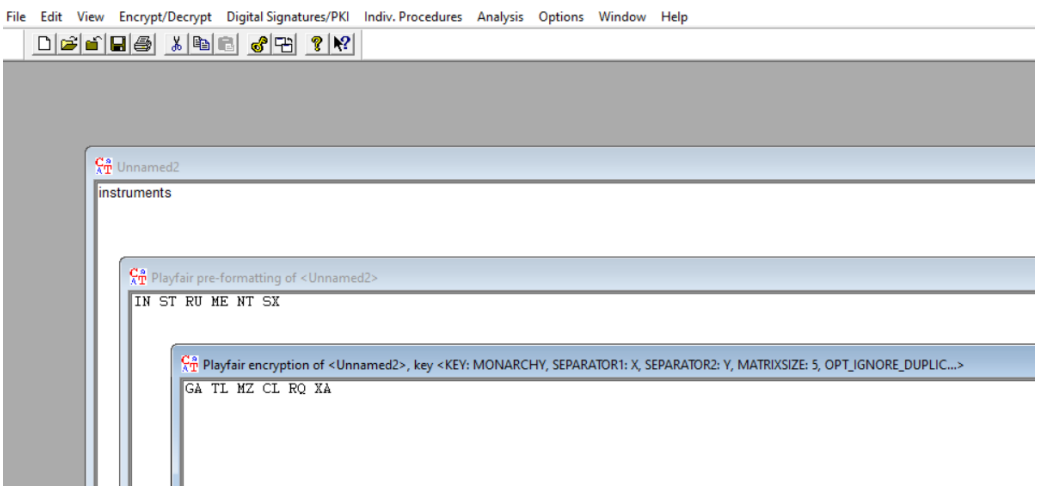
```
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                System.out.print(printTable[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println();
    }

    private void printResults(String encipher, String dec) {
        System.out.print("Encrypted Message: ");
        System.out.println(encipher);
        System.out.println();
        System.out.print("Decrypted Message: ");
        System.out.println(dec);
    }
}
```

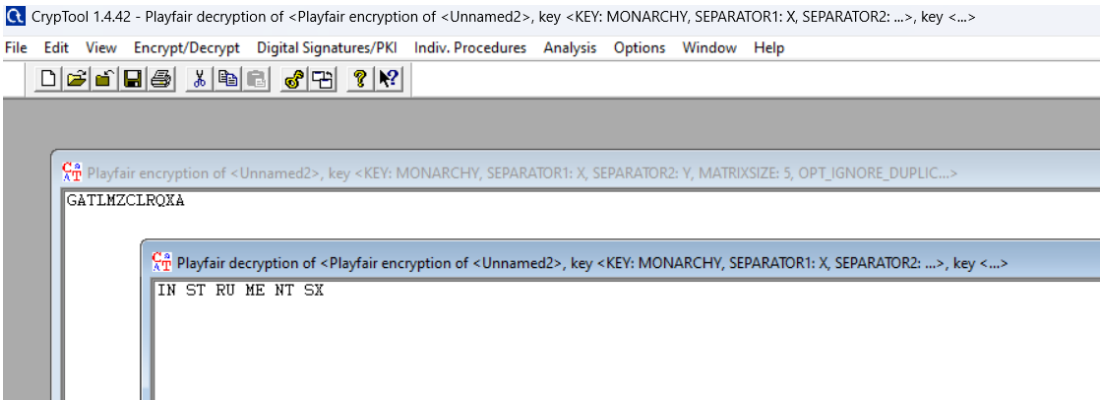
Program Output(IDE):

```
PS D:\infoseclab> cd "d:\infoseclab\" ; if ($?) { javac PlayfairCipher.java } ; if ($?) { java PlayfairCipher }
Enter the key for Playfair cipher: monarchy
Enter the plaintext to be encipher: instruments
Playfair Cipher Key Matrix:

M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Encrypted Message: GATLMZCLRQXA

Decrypted Message: INSTRUMENTSX
```

Program Output (Cryptool):







Cryptanalysis:

Advantages of Playfair Cipher:

- **Polygram Substitution:** Encrypts pairs of letters together, increasing complexity.
- **Enhanced Security:** Offers more security compared to simple substitution ciphers with moderate complexity.
- **Key Management:** The 5x5 matrix provides a clear visual representation of the key.
- **Suitable for Pen-and-Paper:** Can be implemented manually without special tools.

Disadvantages of Playfair Cipher:

- **Limited Character Set:** Cannot encrypt numbers, symbols, or non-alphabetic characters.
- **Small Key Space:** Vulnerable to brute-force attacks due to the limited key space.
- **Lack of Perfect Secrecy:** Doesn't achieve perfect secrecy, making it less secure.
- **Unsuitable for Modern Cryptography:** Not recommended for highly sensitive data.

Time Complexity:

Encryption: The time complexity for encrypting a message using the Playfair cipher is typically O(n), where n is the length of the message. This is because you need to process each pair of letters in the message, look up their positions in the key matrix, and apply the encryption rules.

Decryption: Decryption using the Playfair cipher also has a time complexity of O(n), as you need to process each pair of letters in the ciphertext, find their positions in the key matrix, and apply the decryption rules.

Applications:

- Applications of the Playfair cipher include military communications, cryptography puzzles, and cryptanalysis studies.
- It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II.
- It has also been used in digital encryption systems, such as the CAST-128 and A5/1 encryption algorithms.

References:

- Wikipedia()
- GeeksforGeeks(https://www.geeksforgeeks.org/playfair-cipher-with-examples/)