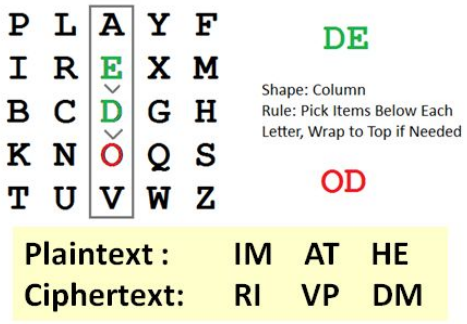
Experiment 3

Aim: Write a program to implement Play-Fair Cipher encryption-decryption.

Theory: Playfair cipher, also known as Playfair square, Wheatstone-Playfair cipher or Wheatstone cipher is a manual symmetric encryption technique and was the first literal diagram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution ciphers.
In the Playfair cipher, initially a key table is created. The key table is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I. The sender and the receiver decide on a particular key, say 'Algorithm'. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order.



Source Code:

```
import java.util.*;
import java.util.Scanner;
public class playfair {
   public static void main(String[] args) {
      final Scanner sc = new Scanner(System.in);
      System.out.println("Enter plaintext");
         final String plaintext = sc.nextLine();
         System.out.println("Enter keyword");
         final String keyword = sc.nextLine();
         final String encryptedText = encrypt(plaintext, keyword);
         final String ptext = decrypt(encryptedText, keyword);
         System.out.println("Plain text    :"+plaintext);
```

```
        System.out.println("Keyword       :"+keyword);
        System.out.println("Encrypted text:"+encryptedText);
        System.out.println("Decrypted text:"+ptext);
      sc.close();
   }

   public static String decrypt(String encryptedText, String keyword) {
      if (encryptedText.length() % 2 != 0)
         {    throw new IllegalArgumentException("Encrypted text length must be even");   }
      encryptedText = encryptedText.toLowerCase();
      keyword = keyword.toLowerCase();
      final char[][] grid = generateGrid(keyword);
      final StringBuilder plaintext = new StringBuilder();
      for (int i = 0; i < encryptedText.length(); i += 2) {
         final char ch1 = encryptedText.charAt(i);
         final char ch2 = encryptedText.charAt(i + 1);
         final Position ch1Position = findPosition(grid, ch1);
         final Position ch2Position = findPosition(grid, ch2);
         final char plaintextCh1;
         final char plaintextCh2;
         if (ch1Position.column == ch2Position.column) {
         plaintextCh1 = grid[Math.floorMod(ch1Position.row -1,
grid.length)][ch1Position.column];
            plaintextCh2 = grid[Math.floorMod(ch2Position.row - 1,
grid.length)][ch2Position.column];
         } else if (ch1Position.row == ch2Position.row) {
            plaintextCh1 = grid[ch1Position.row][Math.floorMod(ch1Position.column - 1,
grid.length)];
            plaintextCh2 = grid[ch2Position.row][Math.floorMod(ch2Position.column - 1,
grid.length)];
         } else {
            final Position topRight;
            final Position bottomLeft;
            if (ch1Position.row < ch2Position.row) {
               topRight = ch1Position;
               bottomLeft = ch2Position;
            } else {
               topRight = ch2Position;
               bottomLeft = ch1Position;}
```

```
            final int rectWidth = topRight.column - bottomLeft.column + 1;
            final Position topLeft = new Position(topRight.row, topRight.column - rectWidth + 1);
            final Position bottomRight = new Position(bottomLeft.row, bottomLeft.column +
rectWidth - 1);
            if (ch1Position == topRight) {
               plaintextCh1 = grid[topLeft.row][topLeft.column];
               plaintextCh2 = grid[bottomRight.row][bottomRight.column];
            } else {
               plaintextCh1 = grid[bottomRight.row][bottomRight.column];
               plaintextCh2 = grid[topLeft.row][topLeft.column];
            }
         }
         plaintext.append(plaintextCh1).append(plaintextCh2);
      }
      return plaintext.toString();
   }

   public static String encrypt(String plaintext, String keyword) {
      plaintext = plaintext.toLowerCase();
      keyword = keyword.toLowerCase();
      final char[][] grid = generateGrid(keyword);
      final String diagraphText = plaintext.length() % 2 == 0 ? plaintext : plaintext + "z";
      final StringBuilder encryptedText = new StringBuilder();
      for (int i = 0; i < diagraphText.length(); i += 2) {
         final char ch1 = diagraphText.charAt(i);
         final char ch2 = diagraphText.charAt(i + 1);
         final Position ch1Pos = findPosition(grid, ch1);
         final Position ch2Pos = findPosition(grid, ch2);
         final char encryptedCh1;
         final char encryptedCh2;
         if (ch1Pos.column == ch2Pos.column) {
            encryptedCh1 = grid[Math.floorMod(ch1Pos.row + 1, grid.length)][ch1Pos.column];
            encryptedCh2 = grid[Math.floorMod(ch2Pos.row + 1, grid.length)][ch2Pos.column];
         } else if (ch1Pos.row == ch2Pos.row) {
            encryptedCh1 = grid[ch1Pos.row][Math.floorMod(ch1Pos.column + 1, grid.length)];
            encryptedCh2 = grid[ch2Pos.row][Math.floorMod(ch2Pos.column + 1, grid.length)];
         } else {
            final Position topLeft;
            final Position bottomRight;
```

```
            if (ch1Pos.row < ch2Pos.row) {
               topLeft = ch1Pos;
               bottomRight = ch2Pos;
            } else {
               topLeft = ch2Pos;
               bottomRight = ch1Pos;
            }
            final int rectWidth = bottomRight.column - topLeft.column + 1;
            final Position topRight = new Position(topLeft.row, topLeft.column + rectWidth - 1);
            final Position bottomLeft = new Position(bottomRight.row, bottomRight.column -
rectWidth + 1);
            if (ch1Pos == topLeft) {
               encryptedCh1 = grid[topRight.row][topRight.column];
               encryptedCh2 = grid[bottomLeft.row][bottomLeft.column];
            } else {
               encryptedCh1 = grid[bottomLeft.row][bottomLeft.column];
               encryptedCh2 = grid[topRight.row][topRight.column];   }
         }
         encryptedText.append(encryptedCh1).append(encryptedCh2);
      }
      return encryptedText.toString();
   }

   private static Position findPosition(char[][] grid, char ch) {
      for (int i = 0; i < grid.length; i++) {
         for (int j = 0; j < grid.length; j++) {
            if (grid[i][j] == ch)
                  {        return new Position(i, j);  }
         }}
      throw new IllegalArgumentException("Character " + ch + " not found in the grid");
   }

   private static char[][] generateGrid(String keyword) {
      final char[][] grid = new char[5][5];
      final Set<Character> charset = buildCharset(keyword);
      final Iterator<Character> charIterator = charset.iterator();
      for (int i = 0; i < 5; i++) {
         for (int j = 0; j < 5; j++) {
```

```
            final char ch = charIterator.next();
            grid[i][j] = ch;  }
    }
    return grid;
}

private static Set<Character> buildCharset(String keyword) {
    final Set<Character> charset = new LinkedHashSet<>(25);
    for (final char ch : keyword.toCharArray()) {
        charset.add(ch);
    }
    final Set<Character> skipChars = new HashSet<>();
    if (charset.contains('j')) {
        if (!charset.contains('i')) {
            skipChars.add('i');
        }
    } else {
        skipChars.add('j');
    }
    for (char ch = 'a'; ch <= 'z' && charset.size() < 25; ch++) {
        if (!charset.contains(ch) && !skipChars.contains(ch)) {
            charset.add(ch);
        }
    }
    return charset;
}

private static class Position {
    final int row;
    final int column;
    Position(int row, int column) {
        this.row = row;
        this.column = column;
    }
    @Override
    public String toString() {
        return "(" + this.row + ", " + this.column + ")";
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (obj == this)
                { return true;}
        if (!(obj instanceof Position))
                { return false; }
        final Position other = (Position) obj;
        return this.row == other.row && this.column == other.column;
    }
    @Override
    public int hashCode() {
        return Objects.hash(this.row, this.column);
    }}
}
```

Output:

```
C:\Users\Admin\Desktop\college\7th Semester\Information and network security (INS)\Lab\Programs>java playfair
Enter plaintext
instruments
Enter keyword
Monarchy
Encrypted text:gatlmzclrqtx
Decrypted text:instrumentsz
```

Learning Outcomes:

It is significantly harder to break since the frequency analysis technique used to break simple substitution ciphers is difficult but still can be used on (25*25) = 625 digraphs rather than 25 monographs which is difficult. Frequency analysis thus requires more cipher text to crack the encryption.