Assignment 2
Implementation of Playfair Cipher

PRN No. 2018BTECS00212

AIM: Implementation of Playfair Cipher using Surname as a key for encryption and decryption.

THEORY:

The Playfair cipher is a digraph substitution cipher. It employs a table where one letter of the alphabet is omitted, and the letters are arranged in a 5x5 grid. Typically, the J is removed from the alphabet and an I takes its place in the text that is to be encoded.

The main weakness of the Playfair cipher is the fact that the sender would have to inform the recipient of the keyword. If an enemy were to intercept this information, the message would be decrypted in a very short amount of time. However, without information on the key, cracking this cipher would prove to be a daunting task.

In attempting to cryptanalyze a message like this, one would first need to figure out whether or not it is a Playfair cipher. The important characteristics are: an even number of letters, no double letters, and if it's a long message, a frequency analysis that shows no more than 25 letters. Once that is done, most of the approach involves a trial and error method.

PROCEDURE:

To encode a message, one breaks it into two-letter chunks. Repeated letters in the same chunk are usually separated by an X. The message, "HELLO ONE AND ALL" would become "HE LX LO ON EA ND AL LX". Since there was not an even number of letters in the message, it was padded with a spare X. Next, you take your letter pairs and look at their positions in the grid.

"HE" forms two corners of a rectangle. The other letters in the rectangle are C and K. You start with the H and slide over to underneath the E and write down K. Similarly, you take the E and slide over to the H column to get C. So, the first two letters are "KC". "LX" becomes "NV" in the same way.

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

"LO" are in the same row. In this instance, you just slide the characters one position to the right, resulting in "MP". The same happens for "ON", resulting in "PO". "EA" becomes "AB" in the same way, but the E is at the far edge. By shifting one position right, we scroll around back to the left side and get A.

"ND" is in a rectangle form and becomes "OC". "AL" are both in the same column, so we just move down one spot. "AL" is changed into "FQ". "LX" is another rectangle and is encoded as "NV". The resulting message is now "KC NV MP PO AB OC FQ NV" or "KCNVMPPOABOCFQNV" if you remove the spaces.

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

CipherText: **DBSUOQIP**
After Split: 'DB' 'SU' 'OQ' 'IP'

SOURCE CODE:

```
class Playfair {
    setKey(key) {
      if (key) {

        const alphabet = ['abcdefghiklmnopqrstuvwxyz'];
        const sanitizedKey = key.toLowerCase().replace('j',
'i').replace(/[^a-z]/g, '');
        const keyGrid = [...new Set(`${sanitizedKey}${alphabet}`)];
        this.grid = [];
        for (let i = 0; i < keyGrid.length; i += 5) {
          this.grid.push(keyGrid.slice(i, i + 5));
        }
      } else {
        this.grid = [
          ['a', 'b', 'c', 'd', 'e'],
          ['f', 'g', 'h', 'i', 'k'],
          ['l', 'm', 'n', 'o', 'p'],
          ['q', 'r', 's', 't', 'u'],
          ['v', 'w', 'x', 'y', 'z']
        ];
      }
    }
    preProcess({ input, decrypt }) {
```

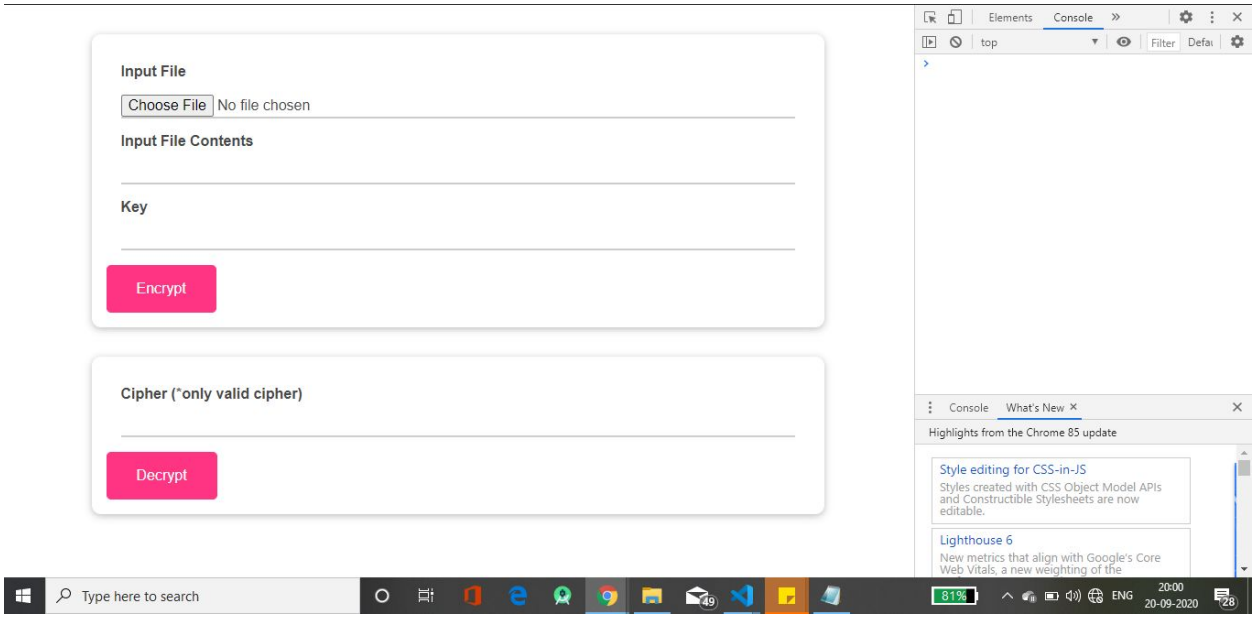
```
      const text = input.toLowerCase().replace(/[^a-z]/g,
'').replace(/j/g, 'i').split('').filter(x => x !== ' ');
      const duples = [];
      for (let i = 0; i < text.length; i += 2) {
        const currentDuple = text.slice(i, i + 2);
        if (!decrypt && currentDuple.length !== 2) {
          currentDuple.push('x');
          duples.push(currentDuple);
        } else if (!decrypt && currentDuple[0] === currentDuple[1]) {
          text.splice(i + 1, 0, 'x');
          duples.push(text.slice(i, i + 2));
        } else {
          duples.push(currentDuple);
        }
      }


      const coordinates = [];
      duples.forEach((duple) => {
        coordinates.push(duple.map((letter) => {
          let col;
          const row = this.grid.findIndex(row => {
            const rowIdx = row.findIndex(x => x === letter);
            if (rowIdx >= 0) {
              col = rowIdx;
              return true;
            }
            return false;
          });
          return [row, col];
        }));
      });

      return coordinates;
    }
    process({ input, decrypt }) {
      if (!this.grid) return 'First set the key!';
      if (input && decrypt && input.length % 2 !== 0) return 'Invalid
ciphertext';
      const coordinates = this.preProcess({ input, decrypt });
```

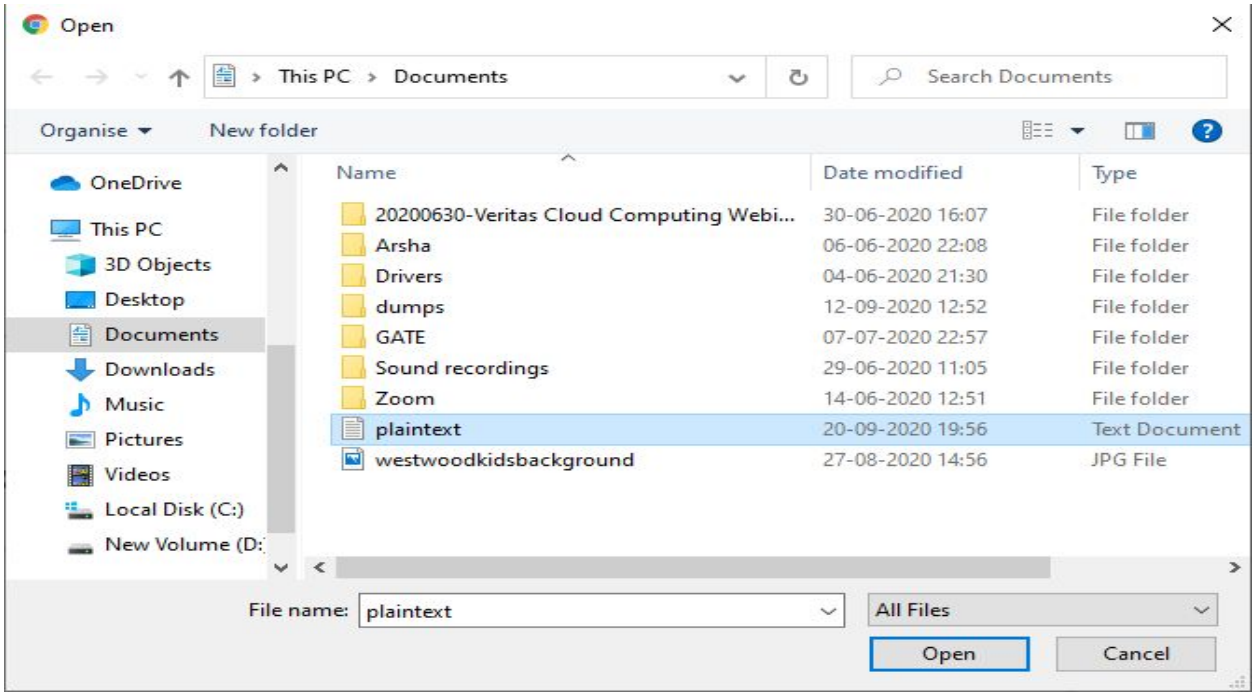
```
      const modifier = decrypt ? -1 : 1;
      const wall = decrypt ? 0 : 4;
      const phase = decrypt ? 4 : -4;

      const processedLocs = [];
      coordinates.forEach((loc) => {

        let modifiedLoc = [];

        if (loc[0][0] === loc[1][0]) {

          modifiedLoc[0] = loc[0][1] === wall ? [loc[0][0], wall + phase]
: [loc[0][0], loc[0][1] + modifier];
          modifiedLoc[1] = loc[1][1] === wall ? [loc[1][0], wall + phase]
: [loc[1][0], loc[1][1] + modifier];
          return processedLocs.push(modifiedLoc);
        }


        if (loc[0][1] === loc[1][1]) {

          modifiedLoc[0] = loc[0][0] === wall ? [wall + phase, loc[0][1]]
: [loc[0][0] + modifier, loc[0][1]];
          modifiedLoc[1] = loc[1][0] === wall ? [wall + phase, loc[1][1]]
: [loc[1][0] + modifier, loc[1][1]];
          return processedLocs.push(modifiedLoc);
        }

        modifiedLoc[0] = [loc[0][0], loc[1][1]];
        modifiedLoc[1] = [loc[1][0], loc[0][1]];
        processedLocs.push(modifiedLoc);
      });


      const processedText = processedLocs
        .map((loc) => [this.grid[loc[0][0]][loc[0][1]],
this.grid[loc[1][0]][loc[1][1]]].join(''))
        .join('');
```

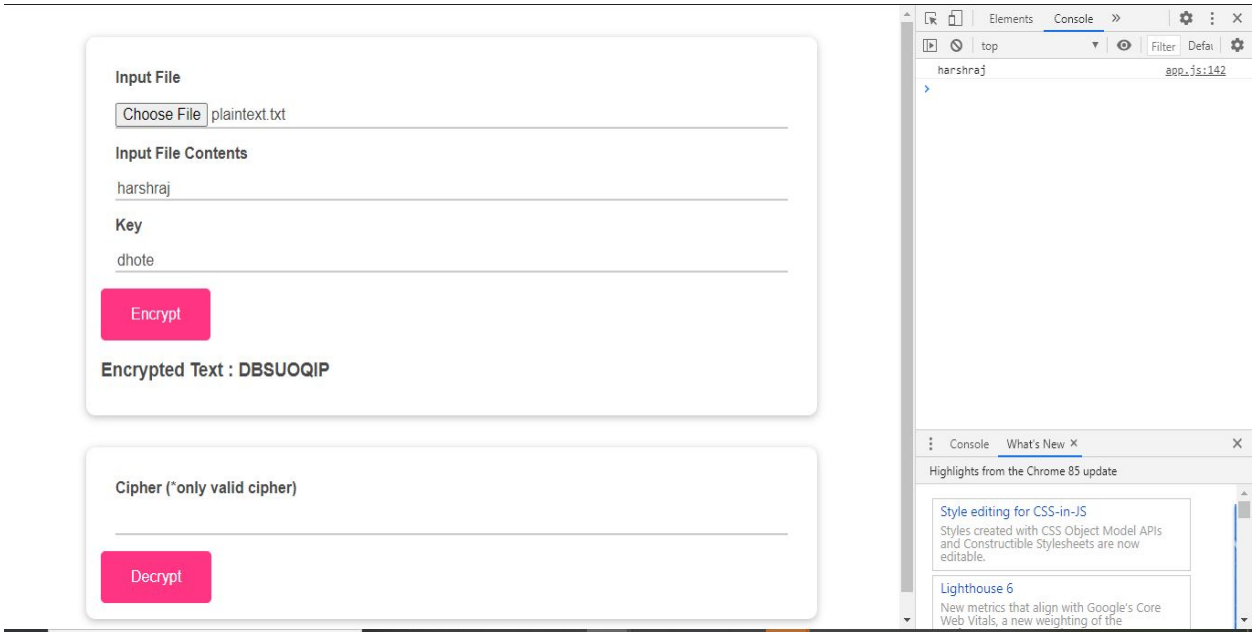
```
      return processedText.toUpperCase();
    }
  }

let plainText = document.getElementById('plainText');
let key = document.getElementById('key');
let decryptBtn = document.getElementById('decrypt-btn');
let encryptBtn = document.getElementById('encrypt-btn');
let encryptResult = document.getElementById('encryptResult');
let cipherText = document.getElementById('cipherText');
let decryptResult = document.getElementById('decryptResult');
const pf = new Playfair();

class PlayfairUtil{

   static encrypt(){
    this.key = key.value;
    this.input = plainText.value;
    pf.setKey(this.key);
    let cipherText = pf.process({input : this.input});
    encryptResult.innerHTML = `<h3>Encrypted Text : ${cipherText}</h3>`;
    }
   static decrypt(){
    this.key = key.value;
    pf.setKey(this.key);
    this.input = cipherText.value;
    console.log(this.key," ",this.input);
    let plainText = pf.process({input : this.input,decrypt:true});
    decryptResult.innerHTML = `<h3>Decrypted Text : ${plainText}</h3>`;
    }
}

encryptBtn.addEventListener('click',PlayfairUtil.encrypt);
decryptBtn.addEventListener('click',PlayfairUtil.decrypt);
document.getElementById('inputFile')
            .addEventListener('change', function() {

            var fr=new FileReader();
            fr.onload=function(){
                document.getElementById('plainText')
```

```
                    .value=fr.result;
                console.log(fr.result);
            }

            fr.readAsText(this.files[0]);
        })
```

O/P SCREENSHOTS :




Open
This PC
Documents
Search Documents
Organise
New folder
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
New Volume (D:
Name
Date modified
Type
20200630-Veritas Cloud Computing Webi...
30-06-2020 16:07
File folder
Arsha
06-06-2020 22:08
File folder
Drivers
04-06-2020 21:30
File folder
dumps
12-09-2020 12:52
File folder
GATE
07-07-2020 22:57
File folder
Sound recordings
29-06-2020 11:05
File folder
Zoom
14-06-2020 12:51
File folder
plaintext
20-09-2020 19:56
Text Document
westwoodkidsbackground
27-08-2020 14:56
JPG File
File name:
plaintext
All Files
Open
Cancel



Conclusion:

Hence I successfully implemented the Playfair Cipher using Surname as a key for encryption and decryption.