Kingdom of Saudi Arabia
Ministry of Education
King Faisal University
College of Computer Sciences & Information Technology

Course Name: Computer Security 320

Report Title: Encrypt & Decrypt
Using Affine Cipher & Playfair Cipher

Sec: 66

Student's Name	Student's ID
Manar Mohammed Alsalman	221416796
Fatima Fadel Alamar	221413772

Supervised by Dr. Abdullah Albuali

12 May 2024

EVALUATION CRITERIA

	Assignment Component	Max. Marks	Marks Obtained
REPORT	Format (Table of Contents, References, formatting and font, etc....)	10	
	English language (grammar and spelling)	10	
	Introduction	15	
	References	10	
	Contents Algorithms, and Discussions / Coding	30	
Demonstration	Questions and answers	15	
	Interface Design & Usability	15	
	Testing (Various Input Conditions)	15	
	Presentation skills	15	
	Understanding / Explanation of the code	15	
	Total Marks	150	
	Grade for this Project	15	
	Plagiarism report more than 10% to 25% Plagiarism report more than 25% to 40% Plagiarism report more than 40%	- 5 -10 -20	

- ## Introduction:

As programmer we must enhance system security and secure data from being interpreted as being illegally entered without authorization. Encryption and decryption are two methods among several to do this. Converting plaintext “the original, readable message or data” into ciphertext “a modified, unintelligible message” is the first step in the encryption process. The second step is decrypt, which returns the ciphertext back into plaintext so that it can be comprehended initially. Numerous algorithms perform encryption and decryption; their methods and levels of effectiveness are different. We plan to discuss and provide an explanation of the Affine Cipher algorithm in this report.

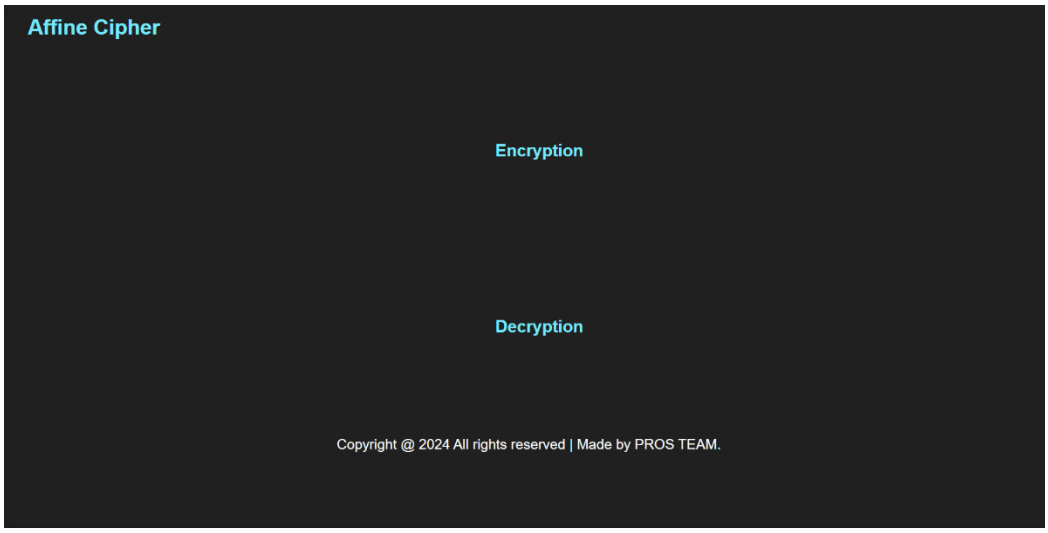
- ## Description of Project implementation:

  - We used “Java Servlet” to create an Affine Cipher encryption/Decryption web Program. What is “Java Servlet” you may ask. Basically, a Java program runs on the server. On the other hand, Java Servlet is a program that runs on a web or application server and acts as a middle layer between a request coming from a web browser or other HTTP client and databases or application on the HTTP server. Resides at server side and generates dynamic web pages. Servlets are loaded and executed by a web server in the same manner that applets are loaded and executed by a web browser. Java Servlets handle data/requests sent by users, create, and format results and send results back to user.
  - We also used “HTML5/JS” to create Playfair Cipher encryption/Decryption web Program. Typically, an HTML5 application consists of HTML, CSS, and JavaScript files. The JavaScript server in the web browser typically handles handling JavaScript on the client side, which is used to manipulate and process objects in the application. A common use case for an HTML5 application is for a client to leverage different web services.

- ## Description of the Algorithm:

  - ### Affine Cipher:

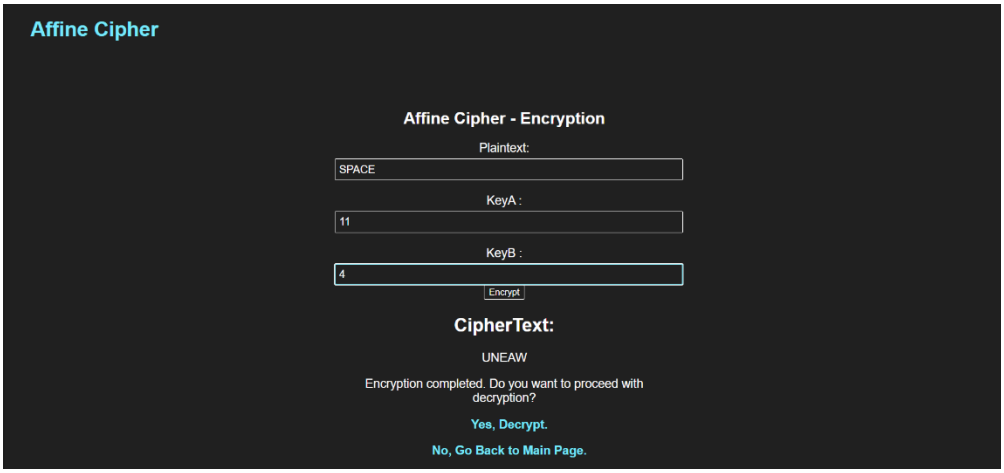
[The Affine Cipher is a type of monoalphabetic shift cipher, where in each letter in an alphabet is mapped to its numeric equivalent.]

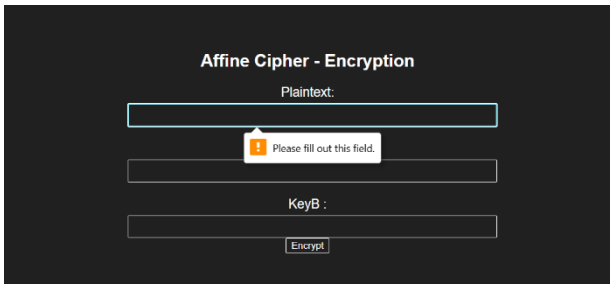


This is the user’s interface for the Affine Cipher webpage. We have an operation menu one for the

encryption and the other is for decryption, so the user can choose which operation they want to do. Each block will lead the user to the operation's page. If the user clicked on the "Encryption" block they will enter the encrypt page.

▪ Encryption:





The user entered the encryption page. First, they'll be asked to input a plaintext and encryption key which are required. After entering all the required data click on the "Encrypt" button to get the result. How did that work?

```
private String encrypt(String plaintext, int a, int b) {
    StringBuilder ciphertext = new StringBuilder();
    for (char c : plaintext.toCharArray()) {
        if (Character.isUpperCase(c) || c == ' ') {
            int x = (int) c - 'A';
            int encrypted = (a * x + b) % 26;
            char encryptedChar = (char) (encrypted + 'A');
            ciphertext.append(encryptedChar);
        }
    }
    return ciphertext.toString();
}
```

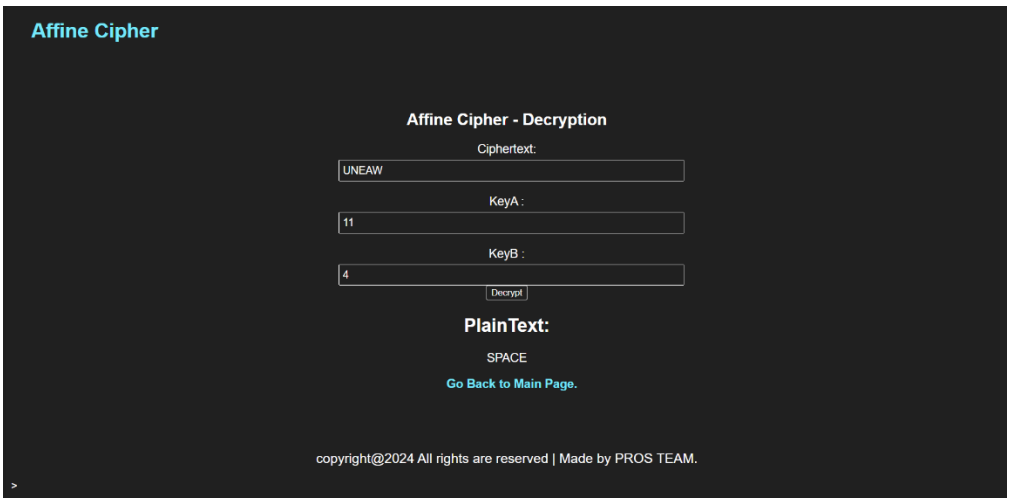
Now we entered the encryption method:

1- It starts by initializing a StringBuilder object ciphertext to store the encrypted characters.
2- The method then iterates over each character c in the plaintext string.
3- Inside the loop, it checks if the character c is uppercase or a space. It ignores any other characters.

4- If the character c is uppercase or a space, it calculates the encrypted value using the formula (a * x + b) % 26, where x is the numeric representation of the character c (0 for 'A', 1 for 'B', and so on).
5- The calculated encrypted value is then converted back to a character using (char) (encrypted + 'A') and appended to the ciphertext string.
6- After the loop completes, the method returns the final ciphertext string.

If the user wants to do the Decryption operation, they can click on the “yes, Decrypt” to enter decryption webpage.

▪ Decryption:



Similarly to Encryption, we ask the user to input ciphertext and decryption key. By clicking on “Decrypt” button the result will be displayed.

```
private String decrypt(String ciphertext, int a, int b) {
    StringBuilder plaintext = new StringBuilder();
    int aInverse = 0;
    boolean inverseExists = false;

    // Find the modular inverse of 'a'
    for (int i = 0; i < 26; i++) {
        int temp = (a * i) % 26;
        if (temp == 1) {
            aInverse = i;
            inverseExists = true;
            break;
        }
    }

    if (!inverseExists) {
        return "No modular inverse exists.";
    }

    for (char c : ciphertext.toCharArray()) {
        if (Character.isUpperCase(c) || c == ' ') {
            int y = (int) c - 'A';
            int decrypted = (aInverse * (y - b + 26)) % 26;
            char decryptedChar = (char) (decrypted + 'A');
            plaintext.append(decryptedChar);
        }
    }
    return plaintext.toString();
}
```

Decrypt method:

1- It starts by initializing a StringBuilder object plaintext to store the decrypted characters.
2- The method then calculates the modular inverse of a (if it exists) using a loop.
3- Inside the loop, it checks if (a * i) % 26 is equal to 1, where i is the loop variable. If a modular inverse is found, it sets aInverse to i and sets inverseExists to true.
4- If no modular inverse exists, the method returns the string "No modular inverse exists."
5- Next, the method iterates over each character c in the ciphertext string.

6- Inside the loop, it checks if the character c is uppercase or a space. It ignores any other characters.
7- If the character c is uppercase or a space, it calculates the decrypted value using the formula (aInverse * (y - b + 26)) % 26, where y is the numeric representation of the character c (0 for 'A', 1 for 'B', and so on).
8- The calculated decrypted value is then converted back to a character using (char) (decrypted + 'A') and appended to the plaintext string.
9- After the loop completes, the method returns the final plaintext string.

▪ HTML CODES:

- Index.html" Main page":

```html
<body>
<header>
    <nav class="Topnav">
        <a href="index.html">Affine Cipher</a>
    </nav>
</header>

<div class="operation-block">
    <a href="encrypt.jsp">Encryption</a>
</div>

<div class="operation-block">
    <a href="decrypt.jsp">Decryption</a>
</div>

<footer class="footer">
    <p id="copy"> Copyright @ 2024 All rights reserved | Made by PROS TEAM.</p>
</footer>
</body>
</html>
```

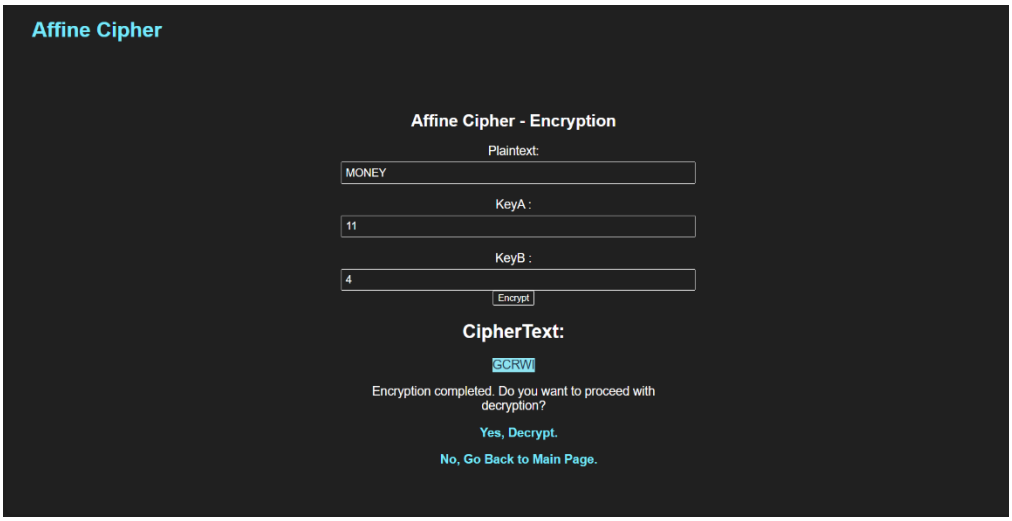
- Encrypt.jsp" Encryption page":

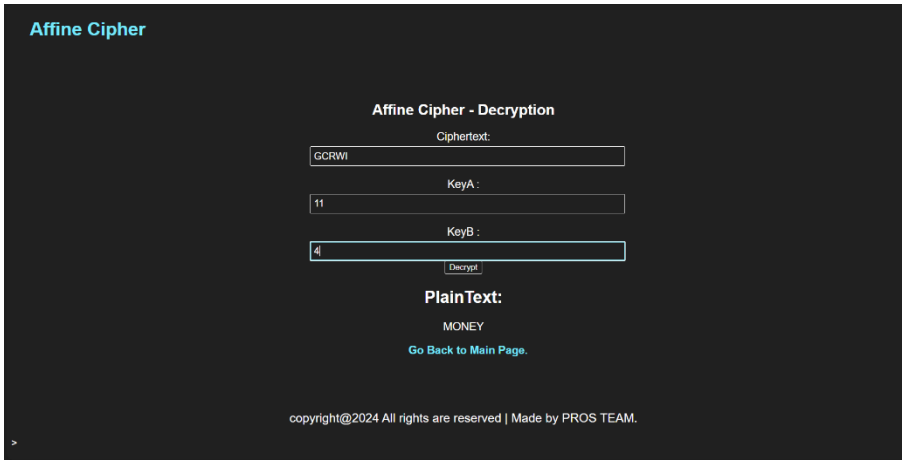
```html
<div class="container">
    <h1>Affine Cipher - Encryption</h1>
    <form action="AffineCipherServlet" method="GET">
        <label for="plaintext">Plaintext:</label>
        <input type="text" id="plaintext" name="plaintext" required>

        <label for="a"> KeyA :</label>
        <input type="text" id="a" name="a" required>

        <label for="b"> KeyB :</label>
        <input type="text" id="b" name="b" required>

        <input type="hidden" name="operation" value="encrypt">
        <input type="submit" value="Encrypt">

    </form>

    <% String result = (String) request.getAttribute("result");
    if (result != null && !result.isEmpty()) { %>
        <div class="result">
            <h2>CipherText:</h2>
            <p><%= result %></p>
            <p>Encryption completed. Do you want to proceed with decryption?</p>
            <p><a href="decrypt.jsp">Yes, Decrypt.</a></p>
            <p><a href="index.html">No, Go Back to Main Page.</a></p>
        </div>
    <% } %>
</div>

<footer class="footer">
    <p id="copy"> copyright@2024 All rights are reserved | Made by PROS TEAM. </p>
</footer>
```

- Decrypt.jsp" Decryption page":

```jsp
<div class="container">
    <h1>Affine Cipher - Decryption</h1>
    <form action="AffineCipherServlet" method="GET">
        <label for="ciphertext">Ciphertext:</label>
        <input type="text" id="ciphertext" name="ciphertext" required>

        <label for="a"> KeyA :</label>
        <input type="text" id="a" name="a" required>

        <label for="b"> KeyB :</label>
        <input type="text" id="b" name="b" required>



        <input type="hidden" name="operation" value="decrypt">
        <input type="submit" value="Decrypt">

    </form>

    <% String result = (String) request.getAttribute("result");
    if (result != null && !result.isEmpty()) { %>
        <div class="result">
            <h2>PlainText:</h2>
            <p><%= result %></p>
            <p><a href="index.html">Go Back to Main Page.</a></p>
        </div>
    <% } %>
</div>


<footer class="footer">
    <p id="copy"> copyright@2024 All rights are reserved | Made by PROS TEAM. </p>
</footer>
```

▪More outputs:

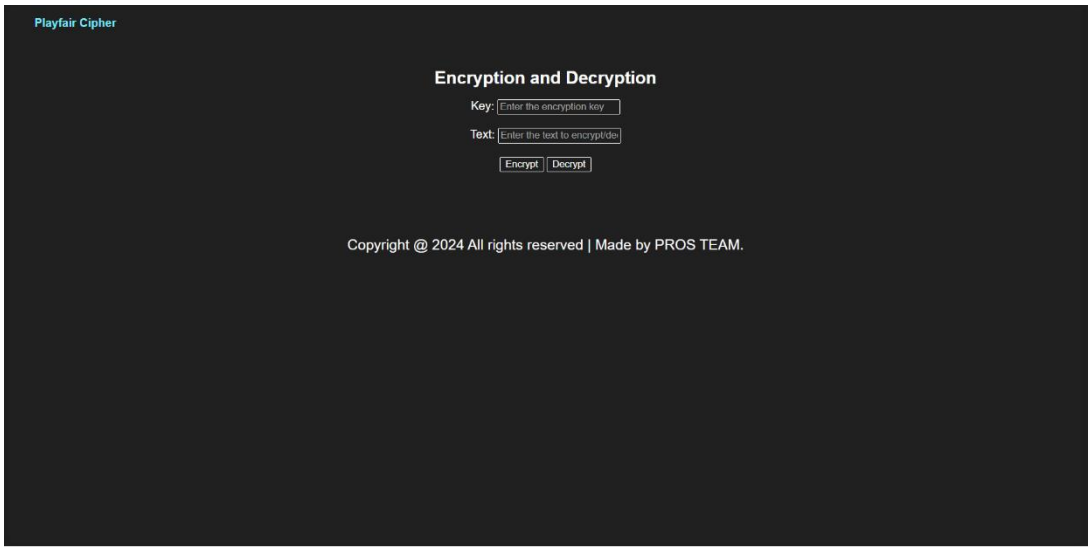




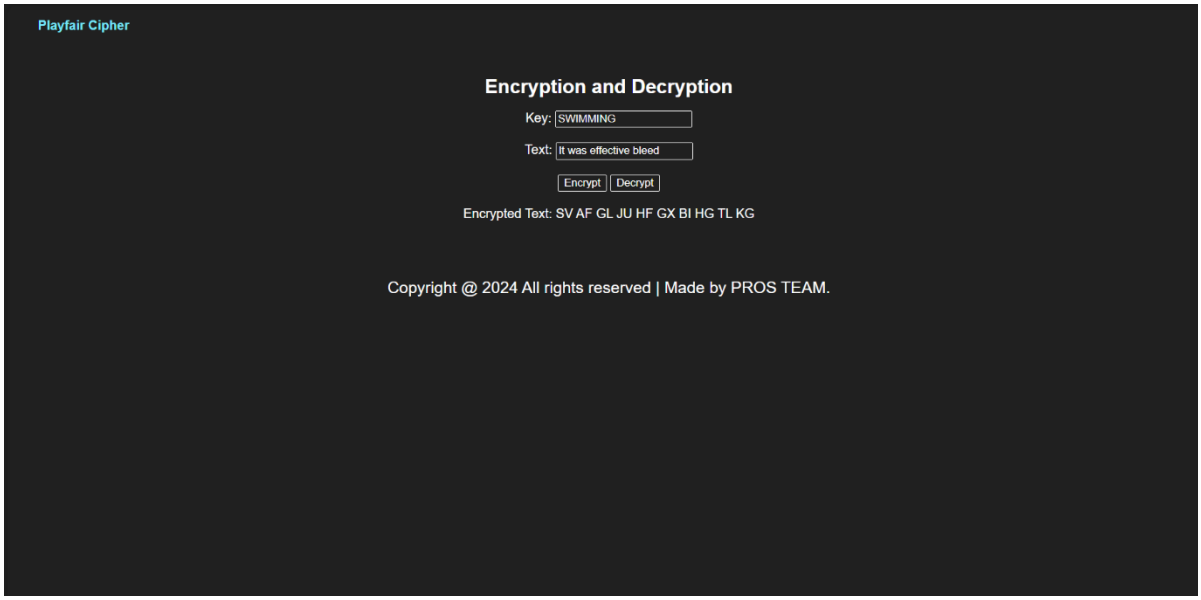
- ## Playfair Cipher:

  The Playfair cipher or Playfair square or Wheatstone–Playfair cipher is a manual symmetric encryption and was the first literal diagram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but bears the name of Lord Playfair for promoting its use.

This is the user's interface for the Playfair Cipher webpage. The user can choose which operation they want to do after writing the keyword and the text. By clicking on the desired operation, the result will be displayed.



- ## Encryption:

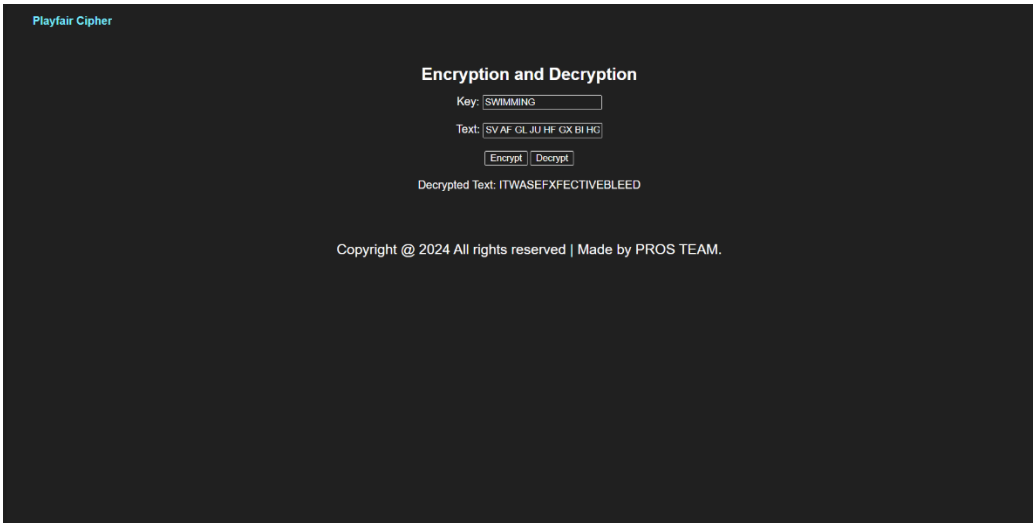


```
function encrypt(text, keyGrid) {
    var encryptedText = "";

    for (var i = 0; i < text.length; i += 2) {
        var ch1 = text.charAt(i);
        var ch2 = text.charAt(i + 1);

        var row1 = -1, col1 = -1, row2 = -1, col2 = -1;
        for (var j = 0; j < 5; j++) {
            for (var k = 0; k < 5; k++) {
                if (keyGrid[j][k] === ch1) {
                    row1 = j;
                    col1 = k;
                }
                if (keyGrid[j][k] === ch2) {
                    row2 = j;
                    col2 = k;
                }
            }
        }

        var encryptedCh1, encryptedCh2;
        if (row1 === row2) {
            encryptedCh1 = keyGrid[row1][(col1 + 1) % 5];
            encryptedCh2 = keyGrid[row2][(col2 + 1) % 5];
        } else if (col1 === col2) {
            encryptedCh1 = keyGrid[(row1 + 1) % 5][col1];
            encryptedCh2 = keyGrid[(row2 + 1) % 5][col2];
        } else {
            encryptedCh1 = keyGrid[row1][col2];
            encryptedCh2 = keyGrid[row2][col1];
        }

        encryptedText += encryptedCh1 + encryptedCh2;
    }

    return encryptedText;
```

Encrypt method:

1- It starts by initializing an empty string encryptedText that will hold the encrypted result.
2- The method then iterates over the preprocessed text in pairs of characters using a loop variable i that increments by 2 in each iteration.
3- Inside the loop, it retrieves the current pair of characters, ch1 and ch2, from the text.
4- Next, it searches for the positions (row and column) of ch1 and ch2 in the keyGrid using nested loops. It iterates over each row and column of the keyGrid and checks if the current element matches ch1 or ch2.
5- Once the positions of ch1 and ch2 are found, the method applies the Playfair Cipher rules to determine the encrypted characters encryptedCh1 and encryptedCh2.
    - If ch1 and ch2 are in the same row, it takes the character to the right of each in the same row, wrapping around to the beginning of the row if necessary.
    - If ch1 and ch2 are in the same column, it takes the character below each in the same column, wrapping around to the top of the column if necessary.
    - If ch1 and ch2 are in different rows and different columns, it takes the character in the same row as ch1 but at the column of ch2, and vice versa.
6- The encrypted characters encryptedCh1 and encryptedCh2 are concatenated to the encryptedText string.
7- After the loop completes, the method returns the final encryptedText string.

- # Decryption:



Decrypt method:

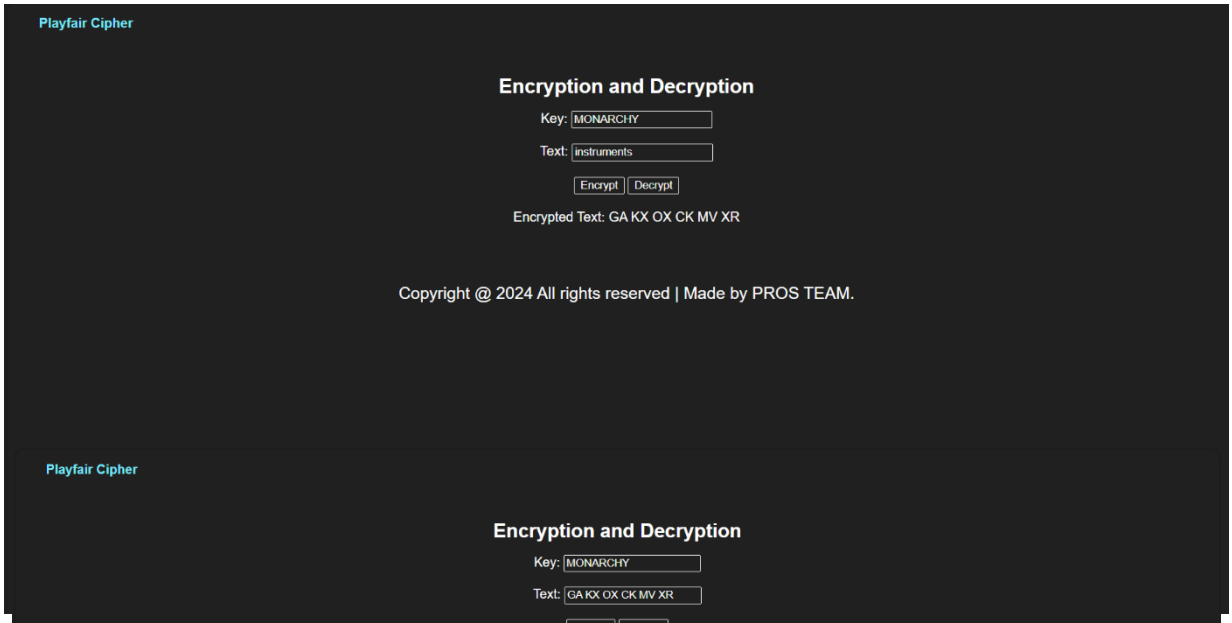
```javascript
function decrypt(encryptedText, keyGrid) {
  let decryptedText = "";

  for (let i = 0; i < encryptedText.length; i += 2) {
    let ch1 = encryptedText.charAt(i);
    let ch2 = encryptedText.charAt(i + 1);

    let [row1, col1] = findCharacterLocation(ch1, keyGrid);
    let [row2, col2] = findCharacterLocation(ch2, keyGrid);

    let decryptedCh1, decryptedCh2;
    if (row1 === row2) {
      decryptedCh1 = keyGrid[row1][(col1 + 4) % 5];
      decryptedCh2 = keyGrid[row2][(col2 + 4) % 5];
    } else if (col1 === col2) {
      decryptedCh1 = keyGrid[(row1 + 4) % 5][col1];
      decryptedCh2 = keyGrid[(row2 + 4) % 5][col2];
    } else {
      decryptedCh1 = keyGrid[row1][col2];
      decryptedCh2 = keyGrid[row2][col1];
    }

    decryptedText += decryptedCh1 + decryptedCh2;
  }

  return decryptedText;
}
```

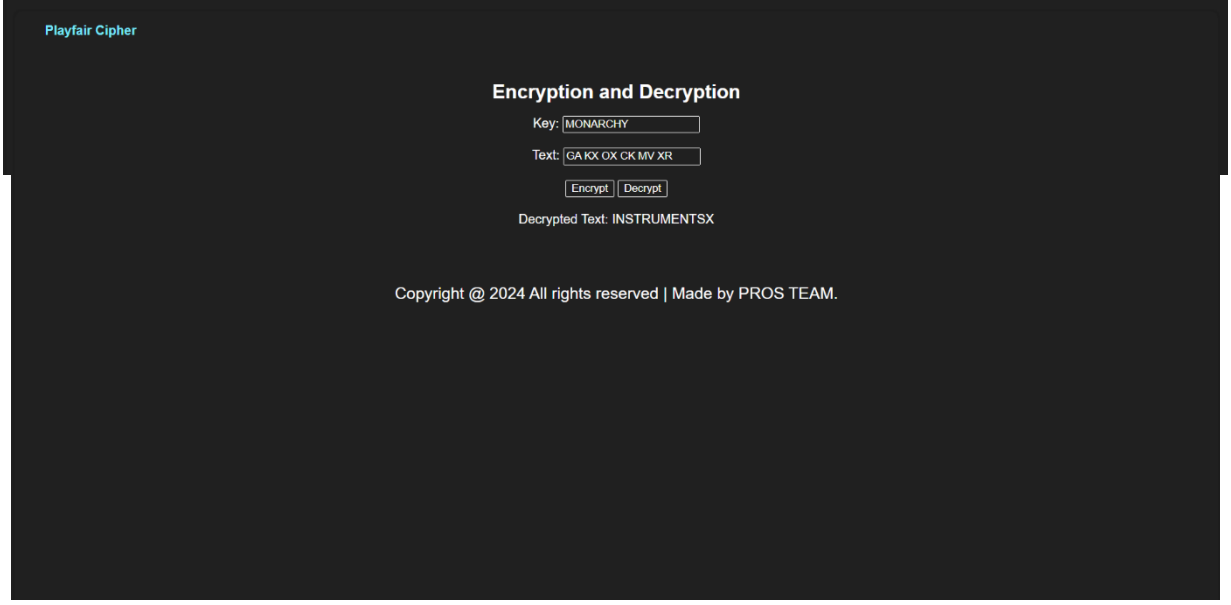
1- It starts by initializing an empty string decryptedText that will hold the decrypted result.
2- The method then iterates over the encryptedText in pairs of characters using a loop variable i that increments by 2 in each iteration.
3- Inside the loop, it retrieves the current pair of characters, ch1 and ch2, from the encryptedText.
4- Next, it uses the findCharacterLocation helper method to find the positions (row and column) of ch1 and ch2 in the keyGrid.
5- Once the positions of ch1 and ch2 are found, the method applies the reverse Playfair Cipher rules to determine the decrypted characters decryptedCh1 and decryptedCh2.
   - If ch1 and ch2 are in the same row, it takes the character to the left of each in the same row, wrapping around to the end of the row if necessary.
   - If ch1 and ch2 are in the same column, it takes the character above each in the same column, wrapping around to the bottom of the column if necessary.
   - If ch1 and ch2 are in different rows and different columns, it takes the character in the same row as ch1 but at the column of ch2, and vice versa.
6- The decrypted characters decryptedCh1 and decryptedCh2 are concatenated to the decryptedText string.
7- After the loop completes, the method returns the final decryptedText string.

▪ HTML CODE:

```html
<body>
    <div class="navbar">
        <a href="#">Playfair Cipher</a>
    </div>

    <div class="container">
        <h1>Encryption and Decryption</h1>
        <label for="key">Key:</label>
        <input type="text" id="key" placeholder="Enter the encryption key" /><br /><br />
        <label for="text">Text:</label>
        <input type="text" id="text" placeholder="Enter the text to encrypt/decrypt" /><br /><br />
        <button onclick="handleEncrypt()">Encrypt</button>
        <button onclick="handleDecrypt()">Decrypt</button><br /><br />
        <div id="result"></div>
    </div>

    <script src="PlayfairCipher.js"></script>

    <footer class="footer">
    <p id="copy"> Copyright @ 2024 All rights reserved | Made by PROS TEAM.</p>
</footer>
</body>
</html>
```

- **More outputs:**





- References:

- Fundamentals of Network Programming - Java Servlet.
- https://en.wikipedia.org/wiki/Affine_cipher
- https://en.wikipedia.org/wiki/Playfair_cipher
- Computer Security Principles and Practice Book.
- https://docs.oracle.com/netbeans/nb81/netbeans/develop/dev_html_apps.htm#NBDAG1525