



UNIVERSITY OF
KARACHI

Network Security & Cryptography

Full Name: Muhammad Talha Idris

Seat No/Roll no: EB22210006102

Course Instructor: Sir Bari

Submitted on: July 05, 2025

1. OTP (One-Time Pad) Cipher

Objective:

To implement the One-Time Pad cipher, ensuring absolute secrecy by using a random key equal in length to the plaintext, and to provide a user-friendly interface for encryption and decryption.

Introduction:

The One-Time Pad (OTP) cipher is a theoretically unbreakable encryption technique that uses a random key, which is as long as the message itself. Each character of the plaintext is combined with a character from the key using modular arithmetic (usually XOR operation). As long as the key is truly random, used only once, and kept secret, OTP provides perfect security.

CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>One-Time Pad Cipher</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 40px;
      background-color: #f5f5f5;
    }
    h1 {
      color: #333;
    }
    textarea, input, button {
      width: 100%;
      padding: 10px;
      margin: 10px 0;
      font-size: 16px;
    }
    button {
      background-color: #007BFF;
```

```

        color: white;
        border: none;
        cursor: pointer;
    }
    button:hover {
        background-color: #0056b3;
    }
    #output {
        background-color: white;
        min-height: 100px;
    }
</style>
</head>
<body>

<h1>One-Time Pad (OTP) Cipher</h1>
<p>Enter your plaintext and a random key of the same length to encrypt or decrypt the message.</p>

<textarea id="inputText" placeholder="Enter Text"></textarea>
<textarea id="key" placeholder="Enter Random Key (same length as text)"></textarea>

<button onclick="encrypt()">Encrypt</button>
<button onclick="decrypt()">Decrypt</button>

<h2>Output:</h2>
<textarea id="output" readonly></textarea>

<script>
    function encrypt() {
        let text = document.getElementById('inputText').value;
        let key = document.getElementById('key').value;

        if (text.length !== key.length) {
            alert("Key must be the same length as the text!");
            return;

```

```

    }

    let encrypted = "";
    for (let i = 0; i < text.length; i++) {
        let encryptedChar = String.fromCharCode(text.charCodeAt(i) ^ key.charCodeAt(i));
        encrypted += encryptedChar;
    }
    document.getElementById('output').value = btoa(encrypted); // base64 encode for readability
}

function decrypt() {
    let encrypted = atob(document.getElementById('inputText').value); // decode from base64
    let key = document.getElementById('key').value;

    if (encrypted.length !== key.length) {
        alert("Key must be the same length as the encrypted text!");
        return;
    }

    let decrypted = "";
    for (let i = 0; i < encrypted.length; i++) {
        let decryptedChar = String.fromCharCode(encrypted.charCodeAt(i) ^ key.charCodeAt(i));
        decrypted += decryptedChar;
    }
    document.getElementById('output').value = decrypted;
}
</script>

</body>
</html>

```

One-Time Pad (OTP) Cipher

Enter your plaintext and a random key of the same length to encrypt or decrypt the message.

Encrypt

Decrypt

Output:

2. Rail Fence Cipher

Objective:

To implement the Rail Fence Cipher for secure transposition of characters and to offer an interactive interface for encrypting and decrypting messages.

Introduction:

The Rail Fence Cipher is a classical transposition cipher that rearranges the characters of the plaintext by writing it diagonally over a set number of "rails" (lines) and then reading it row by row. It is a simple yet effective way to obscure the original message's structure.

CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Rail Fence Cipher</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      margin: 40px;

      background-color: #f5f5f5;

    }

    h1 {

      color: #333;

    }

    textarea, input, button {

      width: 100%;

      padding: 10px;

      margin: 10px 0;

      font-size: 16px;

    }

    button {

      background-color: #28a745;

      color: white;

      border: none;
```

```

        cursor: pointer;
    }
    button:hover {
        background-color: #1e7e34;
    }
    #output {
        background-color: white;
        min-height: 100px;
    }
</style>
</head>
<body>

<h1>Rail Fence Cipher</h1>
<p>Enter your text and the number of rails to encrypt or decrypt the message.</p>

<textarea id="inputText" placeholder="Enter Text"></textarea>
<input type="number" id="rails" placeholder="Enter Number of Rails (e.g., 3)" min="2">

<button onclick="encrypt()">Encrypt</button>
<button onclick="decrypt()">Decrypt</button>

<h2>Output:</h2>
<textarea id="output" readonly></textarea>

<script>
    function encrypt() {
        let text = document.getElementById('inputText').value.replace(/s+/g, " ");
        let numRails = parseInt(document.getElementById('rails').value);

        if (numRails < 2 || text.length < 1) {
            alert("Please enter valid text and rails (minimum 2).");
            return;
        }
    }

```

```

let rail = new Array(numRails).fill("");
let dirDown = false;
let row = 0;

for (let i = 0; i < text.length; i++) {
    rail[row] += text[i];
    if (row === 0 || row === numRails - 1) dirDown = !dirDown;
    row += dirDown ? 1 : -1;
}

let result = rail.join("");
document.getElementById('output').value = result;
}

function decrypt() {
    let cipher = document.getElementById('inputText').value.replace(/s+/g, " ");
    let numRails = parseInt(document.getElementById('rails').value);
    if (numRails < 2 || cipher.length < 1) {
        alert("Please enter valid cipher text and rails (minimum 2).");
        return;
    }
    let len = cipher.length;
    let rail = new Array(numRails).fill("").map(() => new Array(len).fill('\n'));
    let dirDown;
    let row = 0, col = 0;

    for (let i = 0; i < len; i++) {
        if (row === 0) dirDown = true;
        if (row === numRails - 1) dirDown = false;

        rail[row][col++] = '*';

        row += dirDown ? 1 : -1;
    }

    let index = 0;

```

```

for (let i = 0; i < numRails; i++) {
  for (let j = 0; j < len; j++) {
    if (rail[i][j] === '*' && index < len) {
      rail[i][j] = cipher[index++];
    }
  }
}

let result = "";
row = 0;
col = 0;
for (let i = 0; i < len; i++) {
  if (row === 0) dirDown = true;
  if (row === numRails - 1) dirDown = false;

  if (rail[row][col] !== '\n') {
    result += rail[row][col++];
  }

  row += dirDown ? 1 : -1;
}

document.getElementById('output').value = result;
}
</script>
</body>
</html>

```

Rail Fence Cipher

Enter your text and the number of rails to encrypt or decrypt the message.

Encrypt

Decrypt

Output:

3. Plain Text Cipher

Objective:

To demonstrate a basic cipher that outputs the plaintext without any modifications, helping to understand the base case of encryption-decryption processes.

Introduction:

The Plain Text Cipher, often referred to as the Identity Cipher, is not a cipher in the traditional sense. It simply transmits the plaintext as the ciphertext without any encryption. It is useful in understanding the baseline behavior of cryptographic systems before any transformations are applied.

CODE

<!DOCTYPE html>

```
<html lang="en">
```

<head>

```
<meta charset="UTF-8">
```

<title>Plain Text Cipher</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 40px;

```
background-color: #f5f5f5;
```

$$\}$$

h1 {

color: #333;

}

```
textarea, button {
```

width: 100%;

padding: 10px;

```
margin: 10px 0;
```

```
font-size: 16px;
```

}

```
button {
```

background-color: #17a2b8;

```
color: white;
```

border: none;

```
cursor: pointer;
```

}

```

    button:hover {
        background-color: #117a8b;
    }
    #output {
        background-color: white;
        min-height: 100px;
    }
</style>
</head>
<body>

<h1>Plain Text Cipher</h1>
<p>This cipher does not alter the text. It simply outputs the original plaintext.</p>

<textarea id="inputText" placeholder="Enter Text"></textarea>

<button onclick="showPlainText()">Show Plain Text</button>

<h2>Output:</h2>
<textarea id="output" readonly></textarea>
<script>
    function showPlainText() {
        let text = document.getElementById('inputText').value;
        document.getElementById('output').value = text;
    }
</script>
</body>
</html>

```

Plain Text Cipher

This cipher does not alter the text. It simply outputs the original plaintext.

Show Plain Text

Output:

```
CODE
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Vigenère Cipher</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 40px;

            background-color: #f5f5f5;

        }

        h1 {

            color: #333;

        }

        textarea, input, button {

            width: 100%;

            padding: 10px;

            margin: 10px 0;

            font-size: 16px;

        }

        button {

            background-color: #ffc107;

            color: black;

            border: none;

            cursor: pointer;

        }

    
```

```

    button:hover {
        background-color: #e0a800;
    }
    #output {
        background-color: white;
        min-height: 100px;
    }
</style>
</head>
<body>

<h1>Vigenère Cipher</h1>
<p>Enter your text and a keyword to encrypt or decrypt using the Vigenère Cipher.</p>

<textarea id="inputText" placeholder="Enter Text"></textarea>
<input type="text" id="key" placeholder="Enter Key Word">

<button onclick="encrypt()">Encrypt</button>
<button onclick="decrypt()">Decrypt</button>

<h2>Output:</h2>
<textarea id="output" readonly></textarea>

<script>
function formatKey(text, key) {
    key = key.replace(/[^a-zA-Z]/g, "").toUpperCase();
    let formattedKey = "";
    let keyIndex = 0;
    for (let i = 0; i < text.length; i++) {
        if (/[a-zA-Z]/.test(text[i])) {
            formattedKey += key[keyIndex % key.length];
            keyIndex++;
        } else {
            formattedKey += text[i];
        }
    }
}

```

```
    }  
    return formattedKey;  
}
```

```
function encrypt() {  
    let text = document.getElementById('inputText').value;  
    let key = document.getElementById('key').value;  
  
    if (key.length === 0) {  
        alert("Key cannot be empty!");  
        return;  
    }  
  
    let formattedKey = formatKey(text, key);  
    let encrypted = "";  
  
    for (let i = 0; i < text.length; i++) {  
        let char = text[i];  
        if (/[a-zA-Z]/.test(char)) {  
            let base = (char === char.toUpperCase()) ? 65 : 97;  
            let shift = formattedKey[i].toUpperCase().charCodeAt(0) - 65;  
            let encryptedChar = String.fromCharCode((char.charCodeAt(0) - base + shift) % 26 + base);  
            encrypted += encryptedChar;  
        } else {  
            encrypted += char;  
        }  
    }  
  
    document.getElementById('output').value = encrypted;  
}
```

```
function decrypt() {  
    let text = document.getElementById('inputText').value;  
    let key = document.getElementById('key').value;
```

```

if (key.length === 0) {
    alert("Key cannot be empty!");
    return;
}
let formattedKey = formatKey(text, key);
let decrypted = "";
for (let i = 0; i < text.length; i++) {
    let char = text[i];
    if (/[a-zA-Z]/.test(char)) {
        let base = (char === char.toUpperCase()) ? 65 : 97;
        let shift = formattedKey[i].toUpperCase().charCodeAt(0) - 65;
        let decryptedChar = String.fromCharCode((char.charCodeAt(0) - base - shift + 26) % 26 + base);
        decrypted += decryptedChar;
    } else {
        decrypted += char;
    }
}
document.getElementById('output').value = decrypted;
}
</script>
</body>
</html>

```

Vigenère Cipher

Enter your text and a keyword to encrypt or decrypt using the Vigenère Cipher.

Output:

5.Playfair Cipher

Objective:

To implement the Playfair cipher for digraph-based substitution encryption and decryption, and to provide a user interface for secure text transformation using a 5x5 key matrix.

Introduction:

The Playfair Cipher is a classical encryption method that encrypts digraphs (pairs of letters) using a 5x5 matrix of letters based on a keyword. It avoids simple monoalphabetic substitution and makes frequency analysis more difficult. It handles duplicate letters and even-length padding, making it slightly more secure than Caesar or Vigenère ciphers.

CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Playfair Cipher</title>

  <style>

    body{ font-family: Arial; padding: 30px; background: #f8f9fa; }

    h1 { color: #333; }

    textarea, input, button {

      width: 100%; padding: 10px; margin: 10px 0; font-size: 16px;

    }

    button { background: #007bff; color: white; border: none; cursor: pointer; }

    button:hover { background: #0056b3; }

    #output { background: white; min-height: 80px; }

  </style>

</head>
```

```
<body>

  <h1>Playfair Cipher</h1>

  <p>Enter plaintext and a keyword to encrypt or decrypt using the Playfair Cipher.</p>

  <textarea id="inputText" placeholder="Enter text (letters only)"></textarea>

  <input type="text" id="key" placeholder="Enter keyword (e.g., MONARCHY)">

  <button onclick="encrypt()">Encrypt</button>

  <button onclick="decrypt()">Decrypt</button>

  <h2>Output:</h2>

  <textarea id="output" readonly></textarea>

  <script>

    function generateMatrix(key) {

      key = key.toUpperCase().replace(/J/g, "I").replace(/[^A-Z]/g, "");

      let seen = new Set();

      let matrix = [];

      for (let c of key + "ABCDEFGHIJKLMNOPQRSTUVWXYZ") {

        if (!seen.has(c)) {

          seen.add(c);

          matrix.push(c);

        }

      }

      return matrix;

    }

    function preprocess(text) {

      text = text.toUpperCase().replace(/J/g, "I").replace(/[^A-Z]/g, "");

      let result = "";

      for (let i = 0; i < text.length; i += 2) {
```



```

    let a = text[i];

    let b = text[i + 1] || "X";

    if (a === b) {
        result += a + "X";

        i--;
    } else {
        result += a + b;
    }
}

return result;
}

function findPosition(matrix, char) {
    let idx = matrix.indexOf(char);

    return [Math.floor(idx / 5), idx % 5];
}

function transform(text, matrix, encrypt = true) {
    text = preprocess(text);

    let result = "";

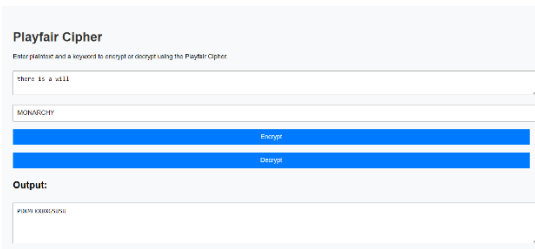
    for (let i = 0; i < text.length; i += 2) {
        let [aRow, aCol] = findPosition(matrix, text[i]);
        let [bRow, bCol] = findPosition(matrix, text[i + 1]);

        if (aRow === bRow) {
            aCol = (aCol + (encrypt ? 1 : 4)) % 5;
            bCol = (bCol + (encrypt ? 1 : 4)) % 5;
        } else if (aCol === bCol) {
            aRow = (aRow + (encrypt ? 1 : 4)) % 5;

```

```
        bRow = (bRow + (encrypt ? 1 : 4)) % 5;
    } else {
        [aCol, bCol] = [bCol, aCol];
    }
    result += matrix[aRow * 5 + aCol] + matrix[bRow * 5 + bCol];
}
return result;
}
function encrypt() {
    let text = document.getElementById("inputText").value;
    let key = document.getElementById("key").value;
    let matrix = generateMatrix(key);
    document.getElementById("output").value = transform(text, matrix, true);
}
function decrypt() {
    let text = document.getElementById("inputText").value;
    let key = document.getElementById("key").value;
    let matrix = generateMatrix(key);
    document.getElementById("output").value = transform(text, matrix, false);
}
</script>
</body>
```

</html>



The screenshot shows a web application titled "Playfair Cipher". Below the title is a small instruction: "Enter plaintext and a keyword to encrypt or decrypt using the Playfair Cipher". There are two input fields: the first contains the text "there is a skill" and the second contains the keyword "MONARCHY". Below these fields are two buttons: "Encrypt" (highlighted in blue) and "Decrypt" (highlighted in blue). Underneath the buttons is an "Output:" label followed by a text area containing the encrypted text "HREH KXREKXRUH".

6. Row-Column Cipher

Objective:

To implement the Row-Column Transposition Cipher, which reorders plaintext based on a specified key, and to provide a basic UI for encrypting and decrypting text using the row-column transposition technique.

Introduction:

The Row-Column Cipher is a type of transposition cipher where the plaintext is written in rows and then read out column-wise based on a key order. The key defines the order in which columns are read during encryption and reassembled during decryption. It's a classical and simple method to scramble text without altering the characters themselves.

CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Row-Column Cipher</title>

  <style>

    body{

      font-family: Arial, sans-serif;

      margin: 40px;

      background-color: #f5f5f5;
```

```
}  
  
h1{  
    color: #333;  
}  
  
textarea, input, button{  
    width: 100%;  
    padding: 10px;  
    margin: 10px 0;  
    font-size: 16px;  
}  
  
button {  
    background-color: #6f42c1;  
    color: white;  
    border: none;  
    cursor: pointer;  
}  
  
button:hover{  
    background-color: #563d7c;  
}  
  
#output{  
    background-color: white;  
    min-height: 100px;  
}  
  
</style>  
</head>  
<body>
```

<h1>Row-Column Cipher</h1>

<p>Enter plaintext and a numeric key (like 3 1 4 2) to encrypt or decrypt using the Row-Column Cipher.</p>

<textarea id="inputText" placeholder="Enter Text (no special characters)"></textarea>

<input type="text" id="key" placeholder="Enter Key Order (e.g., 3 1 4 2)">

<button onclick="encrypt()">Encrypt</button>

<button onclick="decrypt()">Decrypt</button>

<h2>Output:</h2>

<textarea id="output" readonly></textarea>

<script>

```
function parseKey(keyStr) {  
    return keyStr.trim().split(/\s+/).map(Number);  
}
```

```
function encrypt() {  
    let text = document.getElementById("inputText").value.replace(/\s+/g, "");  
    let key = parseKey(document.getElementById("key").value);  
    let cols = key.length;  
    let rows = Math.ceil(text.length / cols);  
  
    // Fill matrix row-wise  
    let matrix = [];  
    let i = 0;  
    for (let r = 0; r < rows; r++) {  
        let row = [];  
        for (let c = 0; c < cols; c++) {  
            row.push(text[i] || 'X'); // pad with X  
            i++;  
        }  
        matrix.push(row);  
    }  
    let output = matrix.map(row => row.join(' ')).join('\n');  
    document.getElementById("output").value = output;  
}
```

```

    }
    matrix.push(row);
}
// Read column-wise based on key
let result = "";
for (let k = 0; k < cols; k++) {
    let collIndex = key.indexOf(k + 1);
    for (let r = 0; r < rows; r++) {
        result += matrix[r][collIndex];
    }
}
document.getElementById("output").value = result;
}

function decrypt() {
    let cipher = document.getElementById("inputText").value.replace(/\s+/g, "");
    let key = parseKey(document.getElementById("key").value);
    let cols = key.length;
    let rows = Math.ceil(cipher.length / cols);
    let matrix = Array.from({ length: rows }, () => Array(cols).fill(""));
    let collLens = Math.floor(cipher.length / cols);
    let extra = cipher.length % cols;
    let pos = 0;
    for (let k = 0; k < cols; k++) {
        let collIndex = key.indexOf(k + 1);
        for (let r = 0; r < rows; r++) {
            if (r * cols + collIndex < cipher.length) {

```

```

        matrix[r][colIndex] = cipher[pos++];
    }
}
}

let result = "";

for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
        result += matrix[r][c];
    }
}

document.getElementById("output").value = result;
}

</script>

</body>

</html>

```

Row-Column Cipher

Enter plaintext and a numeric key (like 3 1 4 2) to encrypt or decrypt using the Row-Column Cipher.

Encrypt

Decrypt

Output:

7. Hill Cipher

Objective:

To implement the Hill Cipher using matrix multiplication for encryption and decryption, and to provide a simple UI that supports 2x2 matrix key input for secure transformation of text blocks.

Introduction:

The Hill Cipher is a polygraphic substitution cipher based on linear algebra. It uses matrix multiplication to transform blocks of plaintext into ciphertext. A square matrix (typically 2x2 or 3x3) serves as the encryption key. For decryption, the inverse of the key matrix is used. This cipher is stronger than monoalphabetic ciphers because it works on multiple letters at once.

CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Hill Cipher</title>

  <style>

    body{

      font-family: Arial, sans-serif;

      margin: 40px;

      background-color: #f5f5f5;

    }

    h1 {

      color: #333;

    }

    textarea, input, button {

      width: 100%;

      padding: 10px;

      margin: 10px 0;

      font-size: 16px;

    }
```



```
button {
    background-color: #dc3545;
    color: white;
    border: none;
    cursor: pointer;
}
button:hover{
    background-color: #c82333;
}
#output{
    background-color: white;
    min-height: 100px;
}
</style>
</head>
<body>
    <h1>Hill Cipher (2x2)</h1>
    <p>Enter plaintext (letters only) and a 2x2 key matrix to encrypt or decrypt.</p>
    <textarea id="inputText" placeholder="Enter text (even number of letters)"></textarea>
    <label>Key Matrix (2x2 - only integers):</label>
    <input id="a" placeholder="a11">
    <input id="b" placeholder="a12">
    <input id="c" placeholder="a21">
    <input id="d" placeholder="a22">
    <button onclick="encrypt()">Encrypt</button>
    <button onclick="decrypt()">Decrypt</button>
```

<h2>Output:</h2>

<textarea id="output" readonly></textarea>

<script>

```
const mod = 26;
```

```
function getMatrix() {
```

```
    let a = parseInt(document.getElementById('a').value);
```

```
    let b = parseInt(document.getElementById('b').value);
```

```
    let c = parseInt(document.getElementById('c').value);
```

```
    let d = parseInt(document.getElementById('d').value);
```

```
    return [[a, b], [c, d]];
```

```
}
```

```
function modInverse(a, m) {
```

```
    a = a % m;
```

```
    for (let x = 1; x < m; x++) {
```

```
        if ((a * x) % m === 1) return x;
```

```
    }
```

```
    return null;
```

```
}
```

```
function encrypt() {
```

```
    let text = document.getElementById("inputText").value.toUpperCase().replace(/[^A-Z]/g, "");
```

```
    if (text.length % 2 !== 0) text += 'X';
```

```
    let matrix = getMatrix();
```

```
    let result = "";
```

```
    for (let i = 0; i < text.length; i += 2) {
```

```

    let x = text.charCodeAt(i) - 65;

    let y = text.charCodeAt(i + 1) - 65;

    let e1 = (matrix[0][0] * x + matrix[0][1] * y) % mod;

    let e2 = (matrix[1][0] * x + matrix[1][1] * y) % mod;

    result += String.fromCharCode(e1 + 65) + String.fromCharCode(e2 + 65);

}

document.getElementById("output").value = result;

}

function decrypt() {

    let text = document.getElementById("inputText").value.toUpperCase().replace(/[^A-Z]/g, "");

    let matrix = getMatrix();

    let det = (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]) % mod;

    if (det < 0) det += mod;

    let invDet = modInverse(det, mod);

    if (invDet === null) {

        alert("Key matrix is not invertible (no modular inverse for determinant).");

        return;

    }

    let invMatrix = [

        [(matrix[1][1] * invDet) % mod, (-matrix[0][1] * invDet + mod) % mod],

        [(-matrix[1][0] * invDet + mod) % mod, (matrix[0][0] * invDet) % mod]

    ];

    let result = "";

    for (let i = 0; i < text.length; i += 2) {

```

```

        let x = text.charCodeAt(i) - 65;

        let y = text.charCodeAt(i + 1) - 65;

        let d1 = (invMatrix[0][0] * x + invMatrix[0][1] * y) % mod;

        let d2 = (invMatrix[1][0] * x + invMatrix[1][1] * y) % mod;

        result += String.fromCharCode(d1 + 65) + String.fromCharCode(d2 + 65);

    }

    document.getElementById("output").value = result;

}

</script>

</body>

</html>

```

Hill Cipher (2x2)

Enter plaintext (letters only) and a 2x2 key matrix to encrypt or decrypt.

Key Matrix (2x2 - only integers):

Encrypt

Decrypt

Output:

8.Caesar Cipher

Objective:

To implement the Caesar Cipher for basic character-shifting encryption and decryption, and to offer an interactive interface to visualize how the text changes based on a user-defined key.

Introduction:

The Caesar Cipher is one of the simplest and oldest encryption techniques. It shifts each letter of the plaintext by a fixed number of positions in the alphabet. For example, with a shift of 3, "A" becomes "D", "B" becomes "E", and so on. Despite its simplicity, it introduces the concept of substitution and modular arithmetic in cryptography.

```

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Caesar Cipher</title>

    <style>

```

```
body {
    font-family: Arial, sans-serif;
    margin: 40px;
    background-color: #f5f5f5;
}

h1 {
    color: #333;
}

textarea, input, button {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    font-size: 16px;
}

button {
    background-color: #20c997;
    color: white;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #198754;
}

#output {
    background-color: white;
    min-height: 100px;
}

</style>
</head>
<body>

<h1>Caesar Cipher</h1>

<p>Enter your text and a key (0–25) to encrypt or decrypt using Caesar Cipher.</p>

<textarea id="inputText" placeholder="Enter text (letters only)"></textarea>
```

```
<input type="number" id="key" placeholder="Enter Key (e.g. 3)" min="0" max="25">
```

```
<button onclick="encrypt()">Encrypt</button>
```

```
<button onclick="decrypt()">Decrypt</button>
```

```
<h2>Output:</h2>
```

```
<textarea id="output" readonly></textarea>
```

```
<script>
```

```
function shiftChar(char, key, encrypt = true) {  
    let base = char === char.toUpperCase() ? 65 : 97;  
    let shift = encrypt ? key : -key;  
    return String.fromCharCode(((char.charCodeAt(0) - base + shift + 26) % 26) + base);  
}
```

```
function processText(encrypt) {  
    let text = document.getElementById("inputText").value;  
    let key = parseInt(document.getElementById("key").value);  
    if (isNaN(key) || key < 0 || key > 25) {  
        alert("Please enter a valid key between 0 and 25.");  
        return;  
    }  
}
```

```
let result = "";  
for (let i = 0; i < text.length; i++) {  
    let char = text[i];  
    if (/[a-zA-Z]/.test(char)) {  
        result += shiftChar(char, key, encrypt);  
    } else {  
        result += char;  
    }  
}
```

```
document.getElementById("output").value = result;  
}
```

```
function encrypt() {  
    processText(true);  
}
```

```
function decrypt() {  
    processText(false);  
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Caesar Cipher

Enter your text and a key (0-25) to encrypt or decrypt using Caesar Cipher.

Encrypt

Decrypt

Output: