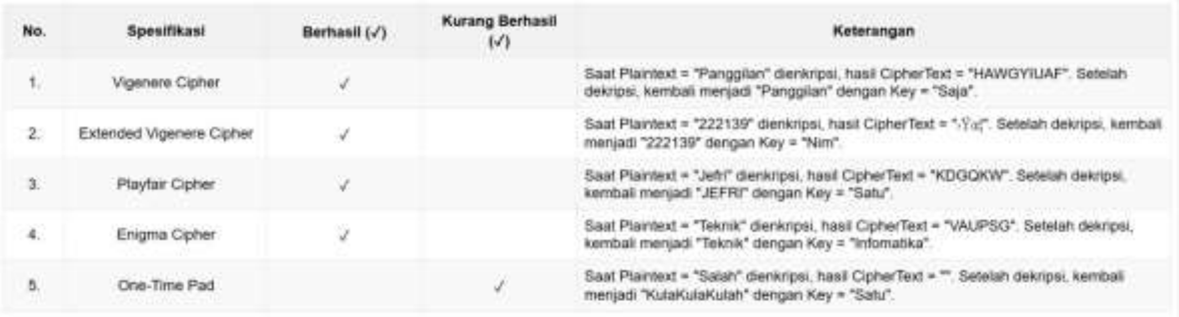
# LAPORAN TUGAS 1

| | | |
|---|---|---|
| Nama | : | Royandi |
| Nim | : | 222271 |
| Kelas | : | 5TKKO-G |
| Mata Kuliah | : | KEAMANAN KOMPUTER/KRIPTOGRAFI |

UNIVERSITAS DIPA MAKASSAR

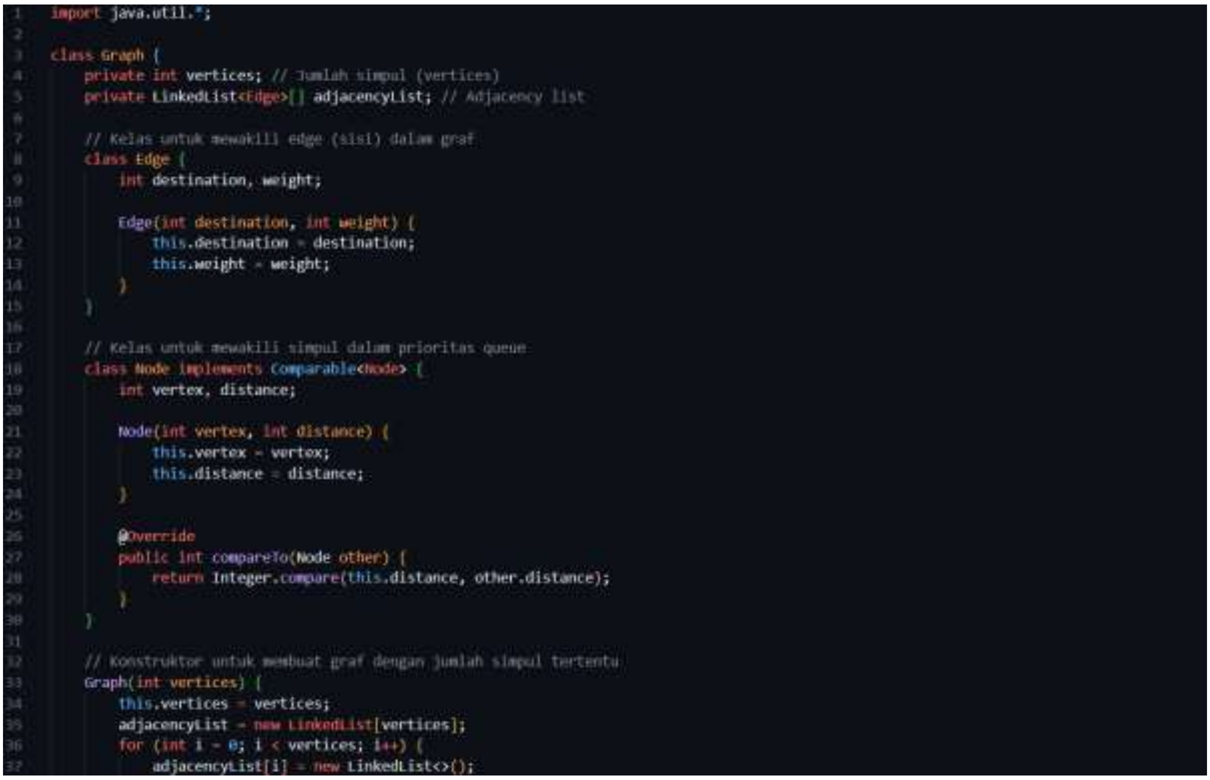
2024/2025

## A. Tabel Laporan

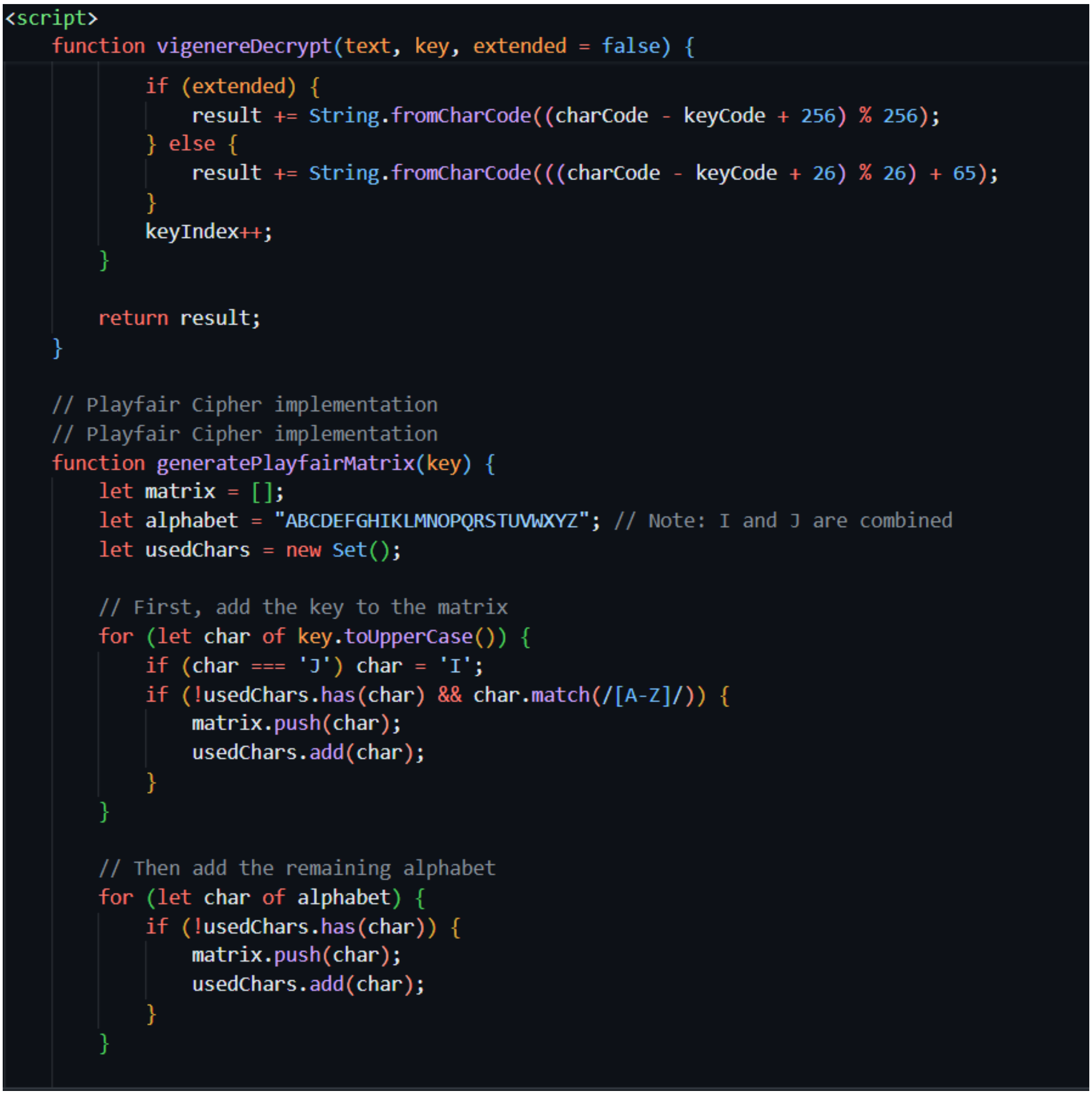


| No. | Spesifikasi | Berhasil (✓) | Kurang Berhasil (✓) | Keterangan |
|---|---|---|---|---|
| 1. | Vigenere Cipher | ✓ |  | Saat Plaintext = "Panggilan" dienkripsi, hasil CipherText = "HAWGYIUAF". Setelah dekripsi, kembali menjadi "Panggilan" dengan Key = "Saja". |
| 2. | Extended Vigenere Cipher | ✓ |  | Saat Plaintext = "222139" dienkripsi, hasil CipherText = "·Ÿα". Setelah dekripsi, kembali menjadi "222139" dengan Key = "Nim". |
| 3. | Playfair Cipher | ✓ |  | Saat Plaintext = "Jefri" dienkripsi, hasil CipherText = "KDGQKW". Setelah dekripsi, kembali menjadi "JEFRI" dengan Key = "Satu". |
| 4. | Enigma Cipher | ✓ |  | Saat Plaintext = "Teknik" dienkripsi, hasil CipherText = "VAUPSG". Setelah dekripsi, kembali menjadi "Teknik" dengan Key = "Infomatika". |
| 5. | One-Time Pad |  | ✓ | Saat Plaintext = "Salah" dienkripsi, hasil CipherText = "". Setelah dekripsi, kembali menjadi "KulaKulaKulah" dengan Key = "Satu". |

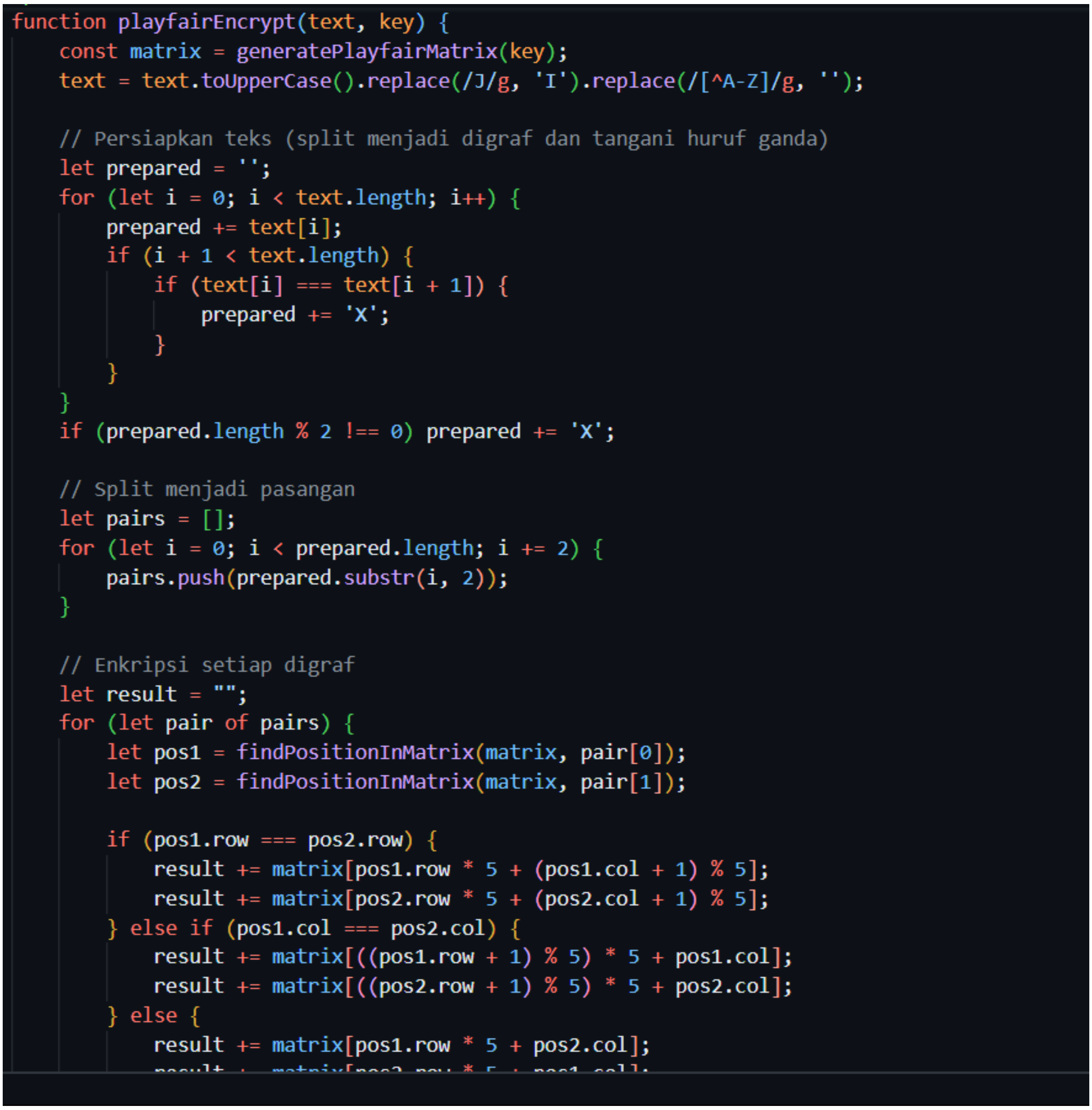
## B. Source Code program



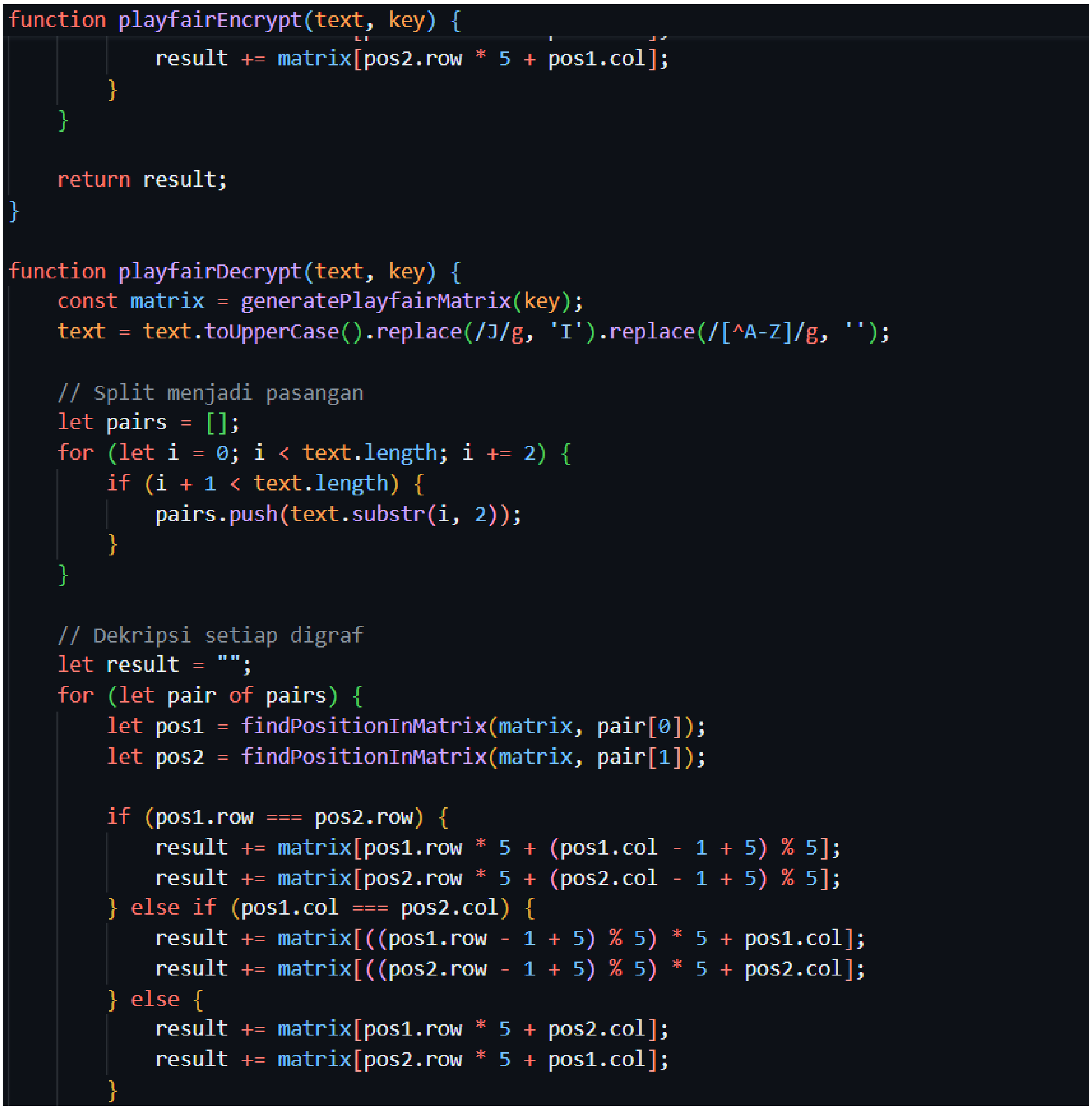
```java
import java.util.*;

class Graph {
    private int vertices; // Jumlah simpul (vertices)
    private LinkedList<Edge>[] adjacencyList; // Adjacency list

    // Kelas untuk mewakili edge (sisi) dalam graf
    class Edge {
        int destination, weight;

        Edge(int destination, int weight) {
            this.destination = destination;
            this.weight = weight;
        }
    }

    // Kelas untuk mewakili simpul dalam prioritas queue
    class Node implements Comparable<Node> {
        int vertex, distance;

        Node(int vertex, int distance) {
            this.vertex = vertex;
            this.distance = distance;
        }

        @Override
        public int compareTo(Node other) {
            return Integer.compare(this.distance, other.distance);
        }
    }

    // Konstruktor untuk membuat graf dengan jumlah simpul tertentu
    Graph(int vertices) {
        this.vertices = vertices;
        adjacencyList = new LinkedList[vertices];
        for (int i = 0; i < vertices; i++) {
            adjacencyList[i] = new LinkedList<>();
```



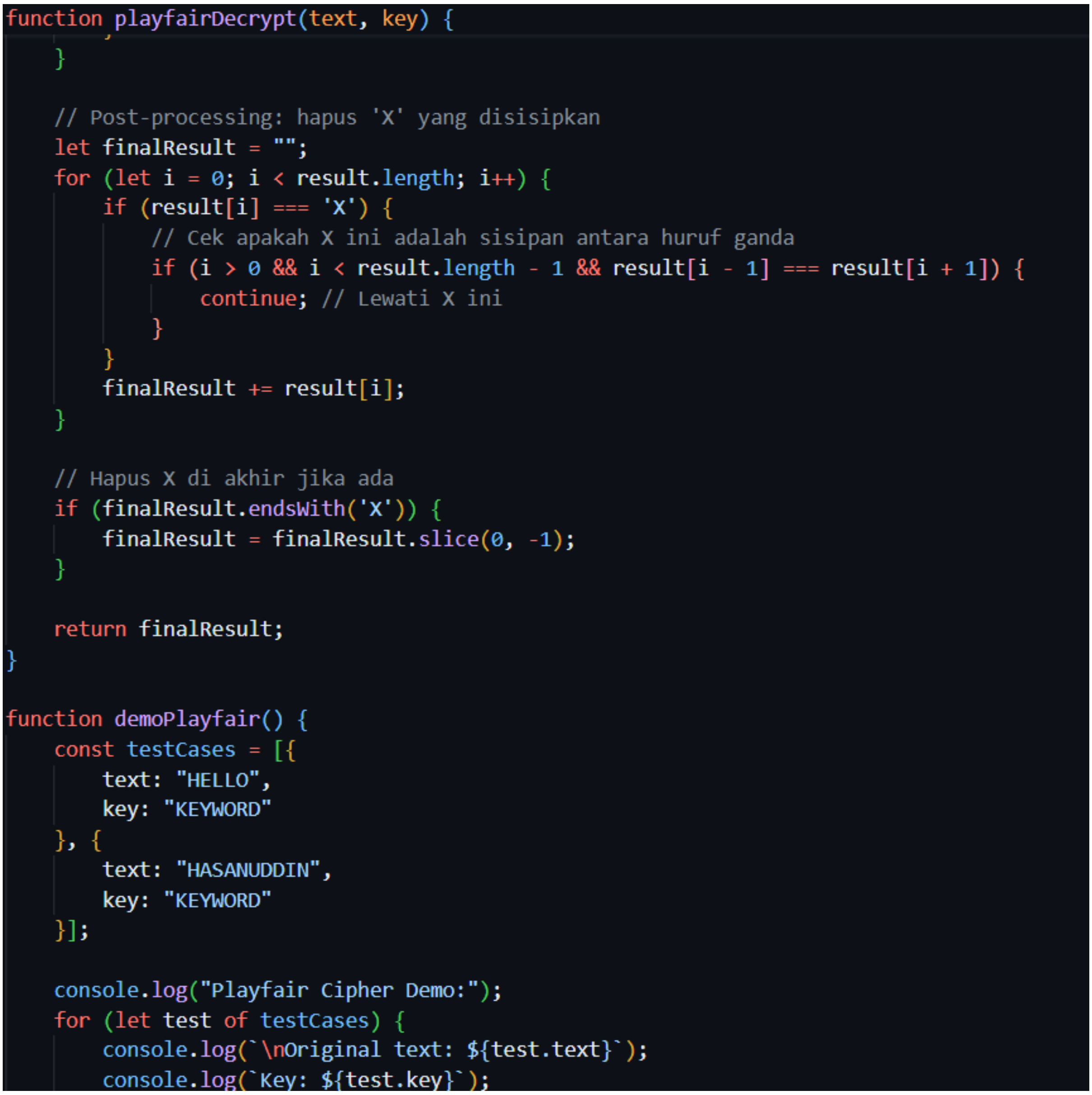
```
<script>
    function vigenereDecrypt(text, key, extended = false) {
            if (extended) {
                result += String.fromCharCode((charCode - keyCode + 256) % 256);
            } else {
                result += String.fromCharCode(((charCode - keyCode + 26) % 26) + 65);
            }
            keyIndex++;
        }

        return result;
    }

    // Playfair Cipher implementation
    // Playfair Cipher implementation
    function generatePlayfairMatrix(key) {
        let matrix = [];
        let alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"; // Note: I and J are combined
        let usedChars = new Set();

        // First, add the key to the matrix
        for (let char of key.toUpperCase()) {
            if (char === 'J') char = 'I';
            if (!usedChars.has(char) && char.match(/[A-Z]/)) {
                matrix.push(char);
                usedChars.add(char);
            }
        }

        // Then add the remaining alphabet
        for (let char of alphabet) {
            if (!usedChars.has(char)) {
                matrix.push(char);
                usedChars.add(char);
            }
        }
```



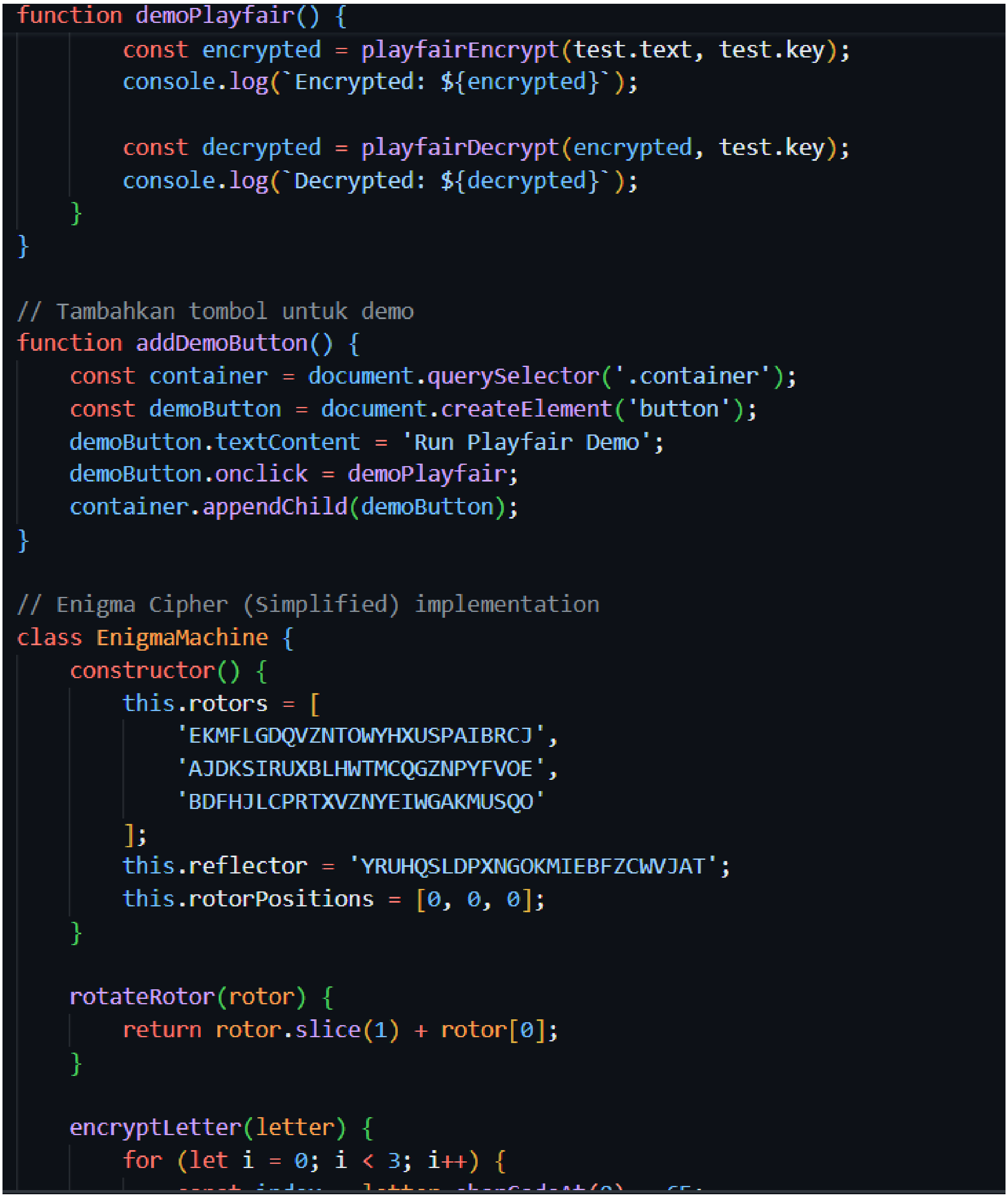
```
function playfairEncrypt(text, key) {
    const matrix = generatePlayfairMatrix(key);
    text = text.toUpperCase().replace(/J/g, 'I').replace(/[^A-Z]/g, '');

    // Persiapkan teks (split menjadi digraf dan tangani huruf ganda)
    let prepared = '';
    for (let i = 0; i < text.length; i++) {
        prepared += text[i];
        if (i + 1 < text.length) {
            if (text[i] === text[i + 1]) {
                prepared += 'X';
            }
        }
    }
    if (prepared.length % 2 !== 0) prepared += 'X';

    // Split menjadi pasangan
    let pairs = [];
    for (let i = 0; i < prepared.length; i += 2) {
        pairs.push(prepared.substr(i, 2));
    }

    // Enkripsi setiap digraf
    let result = "";
    for (let pair of pairs) {
        let pos1 = findPositionInMatrix(matrix, pair[0]);
        let pos2 = findPositionInMatrix(matrix, pair[1]);

        if (pos1.row === pos2.row) {
            result += matrix[pos1.row * 5 + (pos1.col + 1) % 5];
            result += matrix[pos2.row * 5 + (pos2.col + 1) % 5];
        } else if (pos1.col === pos2.col) {
            result += matrix[((pos1.row + 1) % 5) * 5 + pos1.col];
            result += matrix[((pos2.row + 1) % 5) * 5 + pos2.col];
        } else {
            result += matrix[pos1.row * 5 + pos2.col];
            result += matrix[pos2.row * 5 + pos1.col];
```



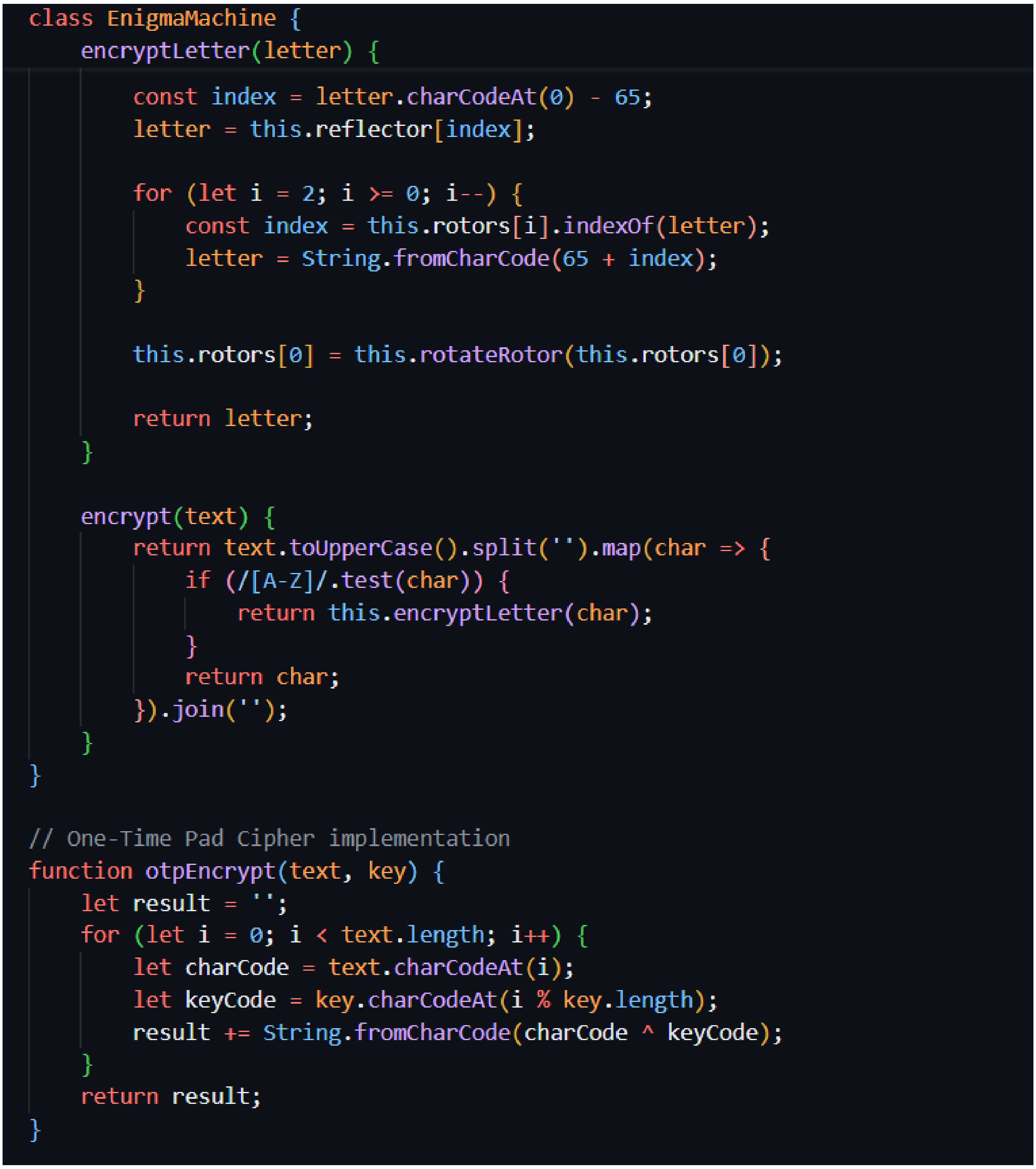
```
function playfairEncrypt(text, key) {
            result += matrix[pos2.row * 5 + pos1.col];
        }
    }

    return result;
}

function playfairDecrypt(text, key) {
    const matrix = generatePlayfairMatrix(key);
    text = text.toUpperCase().replace(/J/g, 'I').replace(/[^A-Z]/g, '');

    // Split menjadi pasangan
    let pairs = [];
    for (let i = 0; i < text.length; i += 2) {
        if (i + 1 < text.length) {
            pairs.push(text.substr(i, 2));
        }
    }

    // Dekripsi setiap digraf
    let result = "";
    for (let pair of pairs) {
        let pos1 = findPositionInMatrix(matrix, pair[0]);
        let pos2 = findPositionInMatrix(matrix, pair[1]);

        if (pos1.row === pos2.row) {
            result += matrix[pos1.row * 5 + (pos1.col - 1 + 5) % 5];
            result += matrix[pos2.row * 5 + (pos2.col - 1 + 5) % 5];
        } else if (pos1.col === pos2.col) {
            result += matrix[((pos1.row - 1 + 5) % 5) * 5 + pos1.col];
            result += matrix[((pos2.row - 1 + 5) % 5) * 5 + pos2.col];
        } else {
            result += matrix[pos1.row * 5 + pos2.col];
            result += matrix[pos2.row * 5 + pos1.col];
        }
```



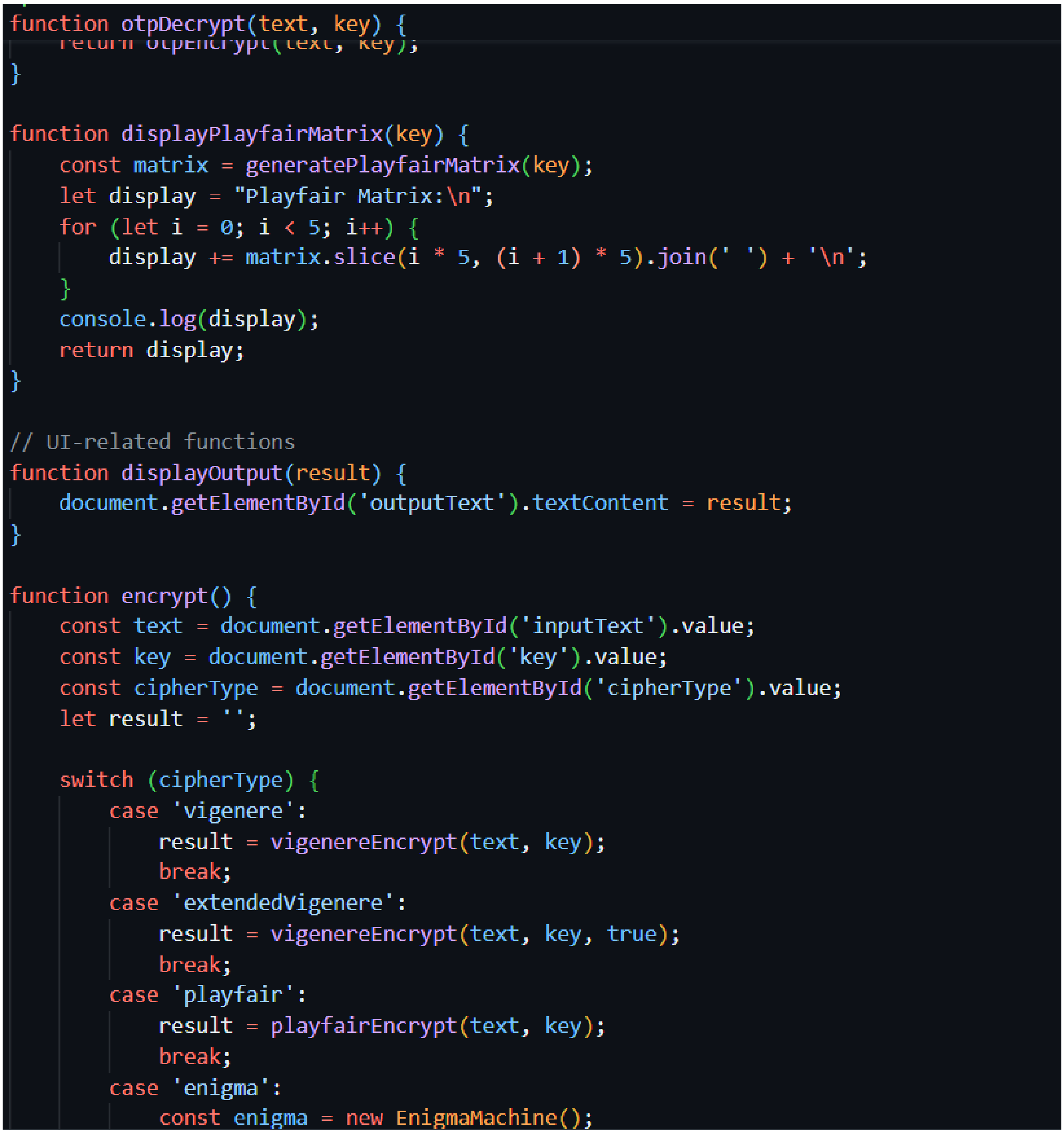
```javascript
function playfairDecrypt(text, key) {
    }

    // Post-processing: hapus 'X' yang disisipkan
    let finalResult = "";
    for (let i = 0; i < result.length; i++) {
        if (result[i] === 'X') {
            // Cek apakah X ini adalah sisipan antara huruf ganda
            if (i > 0 && i < result.length - 1 && result[i - 1] === result[i + 1]) {
                continue; // Lewati X ini
            }
        }
        finalResult += result[i];
    }

    // Hapus X di akhir jika ada
    if (finalResult.endsWith('X')) {
        finalResult = finalResult.slice(0, -1);
    }

    return finalResult;
}

function demoPlayfair() {
    const testCases = [{
        text: "HELLO",
        key: "KEYWORD"
    }, {
        text: "HASANUDDIN",
        key: "KEYWORD"
    }];

    console.log("Playfair Cipher Demo:");
    for (let test of testCases) {
        console.log(`\nOriginal text: ${test.text}`);
        console.log(`Key: ${test.key}`);
```



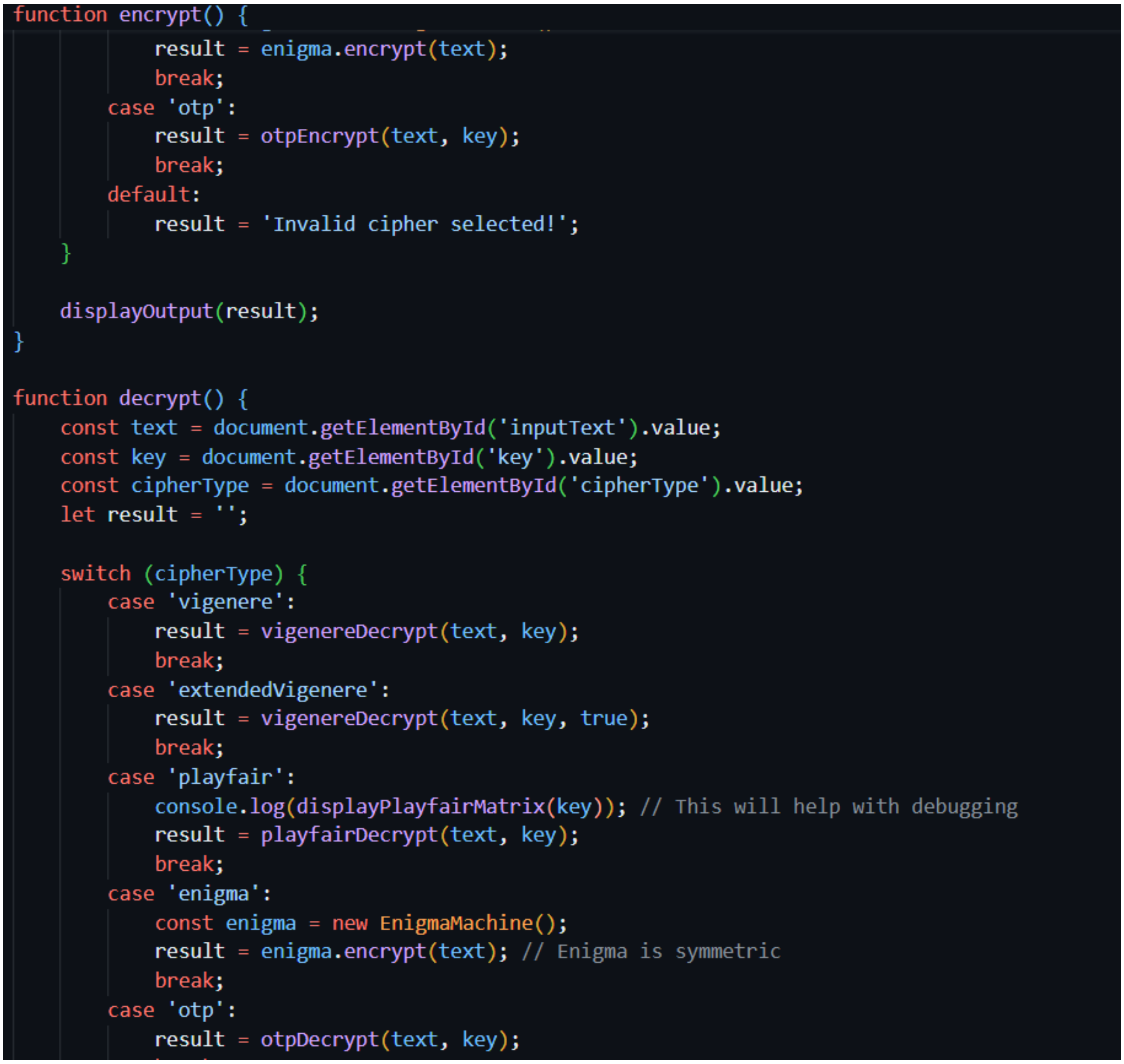
```
function demoPlayfair() {
        const encrypted = playfairEncrypt(test.text, test.key);
        console.log(`Encrypted: ${encrypted}`);

        const decrypted = playfairDecrypt(encrypted, test.key);
        console.log(`Decrypted: ${decrypted}`);
    }
}

// Tambahkan tombol untuk demo
function addDemoButton() {
    const container = document.querySelector('.container');
    const demoButton = document.createElement('button');
    demoButton.textContent = 'Run Playfair Demo';
    demoButton.onclick = demoPlayfair;
    container.appendChild(demoButton);
}

// Enigma Cipher (Simplified) implementation
class EnigmaMachine {
    constructor() {
        this.rotors = [
            'EKMFLGDQVZNTOWYHXUSPAIBRCJ',
            'AJDKSIRUXBLHWTMCQGZNPYFVOE',
            'BDFHJLCPRTXVZNYEIWGAKMUSQO'
        ];
        this.reflector = 'YRUHQSLDPXNGOKMIEBFZCWVJAT';
        this.rotorPositions = [0, 0, 0];
    }

    rotateRotor(rotor) {
        return rotor.slice(1) + rotor[0];
    }

    encryptLetter(letter) {
        for (let i = 0; i < 3; i++) {
            const index = letter.charCodeAt(0) - 65;
```



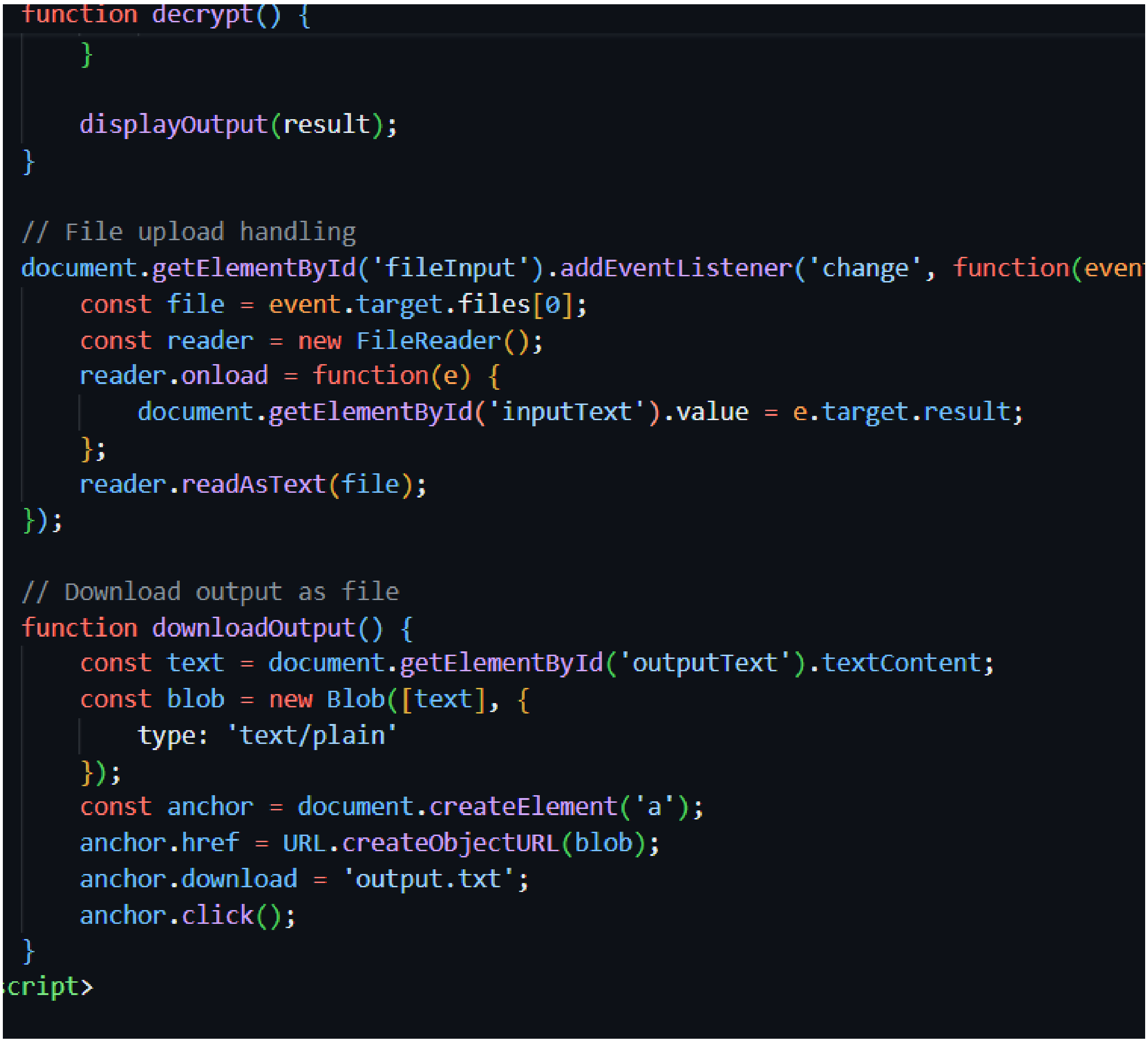
```
class EnigmaMachine {
    encryptLetter(letter) {
        const index = letter.charCodeAt(0) - 65;
        letter = this.reflector[index];

        for (let i = 2; i >= 0; i--) {
            const index = this.rotors[i].indexOf(letter);
            letter = String.fromCharCode(65 + index);
        }

        this.rotors[0] = this.rotateRotor(this.rotors[0]);

        return letter;
    }

    encrypt(text) {
        return text.toUpperCase().split('').map(char => {
            if (/[A-Z]/.test(char)) {
                return this.encryptLetter(char);
            }
            return char;
        }).join('');
    }
}

// One-Time Pad Cipher implementation
function otpEncrypt(text, key) {
    let result = '';
    for (let i = 0; i < text.length; i++) {
        let charCode = text.charCodeAt(i);
        let keyCode = key.charCodeAt(i % key.length);
        result += String.fromCharCode(charCode ^ keyCode);
    }
    return result;
}
```



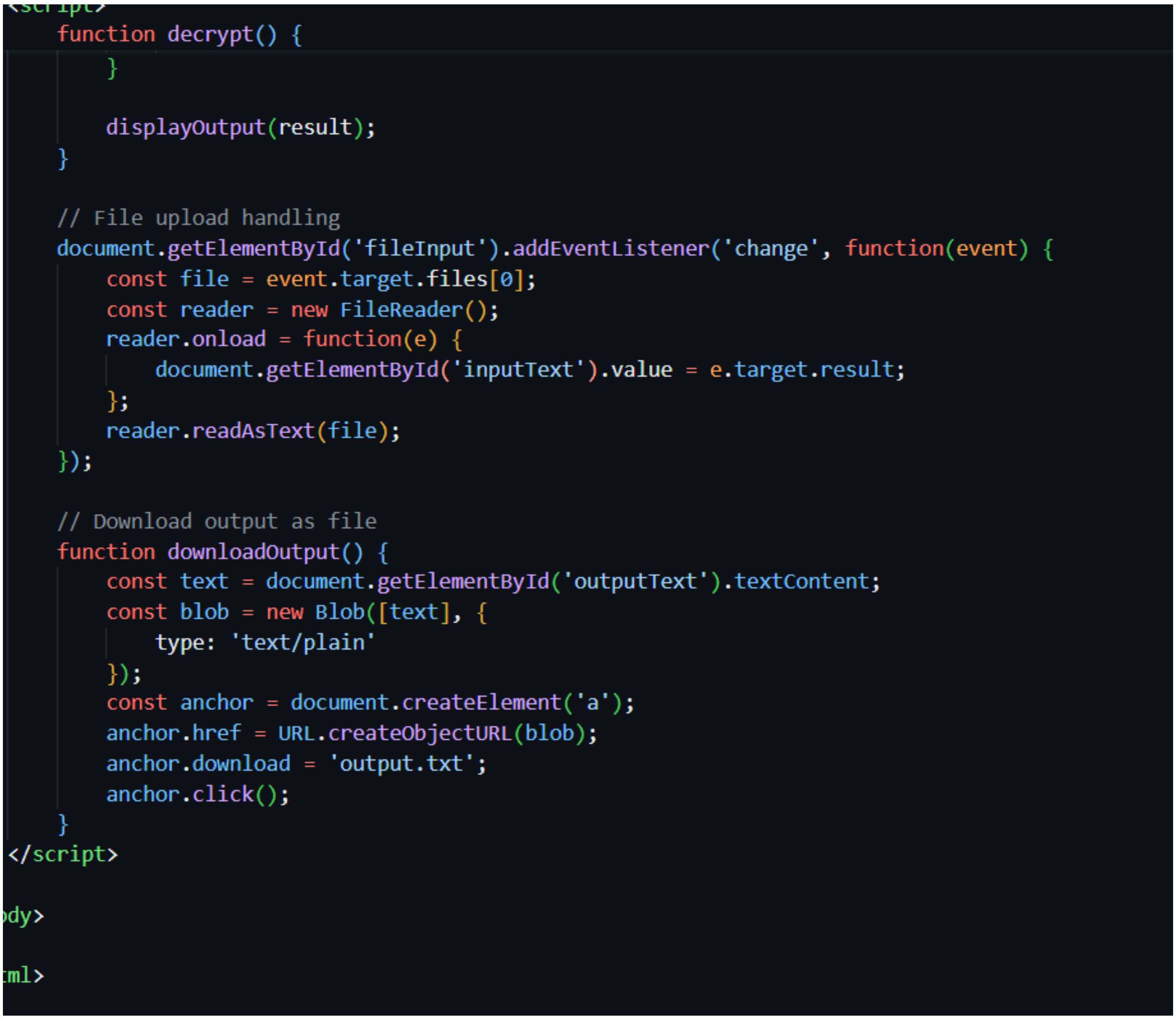
```
function otpDecrypt(text, key) {
    return otpEncrypt(text, key);
}

function displayPlayfairMatrix(key) {
    const matrix = generatePlayfairMatrix(key);
    let display = "Playfair Matrix:\n";
    for (let i = 0; i < 5; i++) {
        display += matrix.slice(i * 5, (i + 1) * 5).join(' ') + '\n';
    }
    console.log(display);
    return display;
}

// UI-related functions
function displayOutput(result) {
    document.getElementById('outputText').textContent = result;
}

function encrypt() {
    const text = document.getElementById('inputText').value;
    const key = document.getElementById('key').value;
    const cipherType = document.getElementById('cipherType').value;
    let result = '';

    switch (cipherType) {
        case 'vigenere':
            result = vigenereEncrypt(text, key);
            break;
        case 'extendedVigenere':
            result = vigenereEncrypt(text, key, true);
            break;
        case 'playfair':
            result = playfairEncrypt(text, key);
            break;
        case 'enigma':
            const enigma = new EnigmaMachine();
```



```
function encrypt() {
            result = enigma.encrypt(text);
            break;
        case 'otp':
            result = otpEncrypt(text, key);
            break;
        default:
            result = 'Invalid cipher selected!';
    }

    displayOutput(result);
}

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

    switch (cipherType) {
        case 'vigenere':
            result = vigenereDecrypt(text, key);
            break;
        case 'extendedVigenere':
            result = vigenereDecrypt(text, key, true);
            break;
        case 'playfair':
            console.log(displayPlayfairMatrix(key)); // This will help with debugging
            result = playfairDecrypt(text, key);
            break;
        case 'enigma':
            const enigma = new EnigmaMachine();
            result = enigma.encrypt(text); // Enigma is symmetric
            break;
        case 'otp':
            result = otpDecrypt(text, key);
```

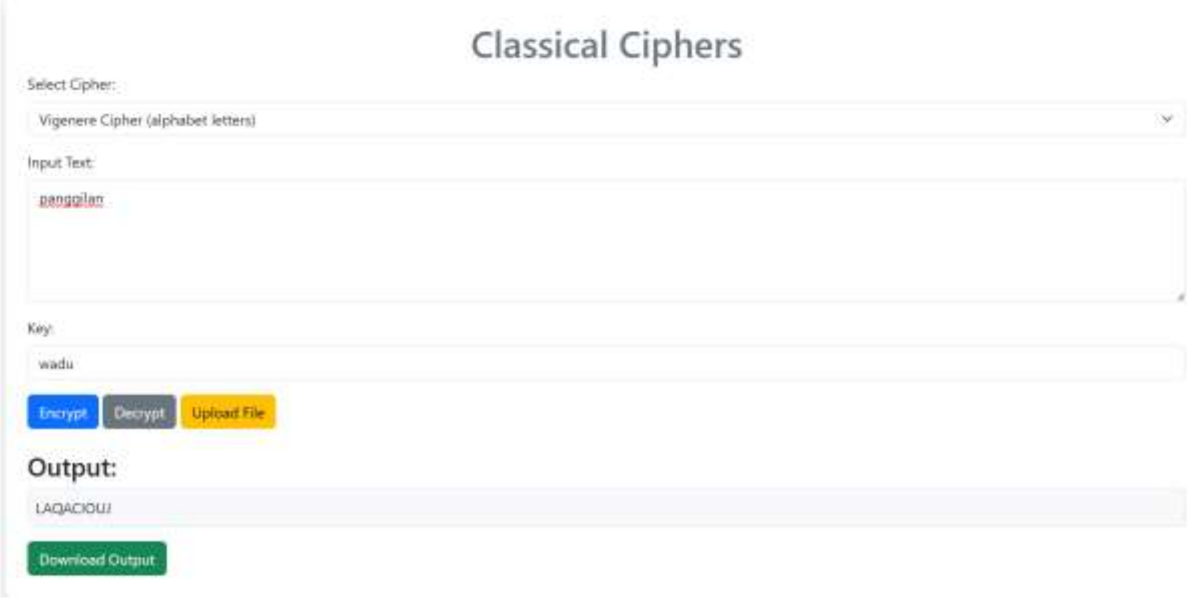


```
function decrypt() {
    }

    displayOutput(result);
}

// File upload handling
document.getElementById('fileInput').addEventListener('change', function(even
    const file = event.target.files[0];
    const reader = new FileReader();
    reader.onload = function(e) {
        document.getElementById('inputText').value = e.target.result;
    };
    reader.readAsText(file);
});

// Download output as file
function downloadOutput() {
    const text = document.getElementById('outputText').textContent;
    const blob = new Blob([text], {
        type: 'text/plain'
    });
    const anchor = document.createElement('a');
    anchor.href = URL.createObjectURL(blob);
    anchor.download = 'output.txt';
    anchor.click();
}
cript>
```



```
<script>
    function decrypt() {
        }

        displayOutput(result);
    }

    // File upload handling
    document.getElementById('fileInput').addEventListener('change', function(event) {
        const file = event.target.files[0];
        const reader = new FileReader();
        reader.onload = function(e) {
            document.getElementById('inputText').value = e.target.result;
        };
        reader.readAsText(file);
    });

    // Download output as file
    function downloadOutput() {
        const text = document.getElementById('outputText').textContent;
        const blob = new Blob([text], {
            type: 'text/plain'
        });
        const anchor = document.createElement('a');
        anchor.href = URL.createObjectURL(blob);
        anchor.download = 'output.txt';
        anchor.click();
    }
</script>

dy>

ml>
```

## C. Contoh PlainText dan ChiperText



Classical Ciphers

Select Cipher:

Vigenere Cipher (alphabet letters)

Input Text:

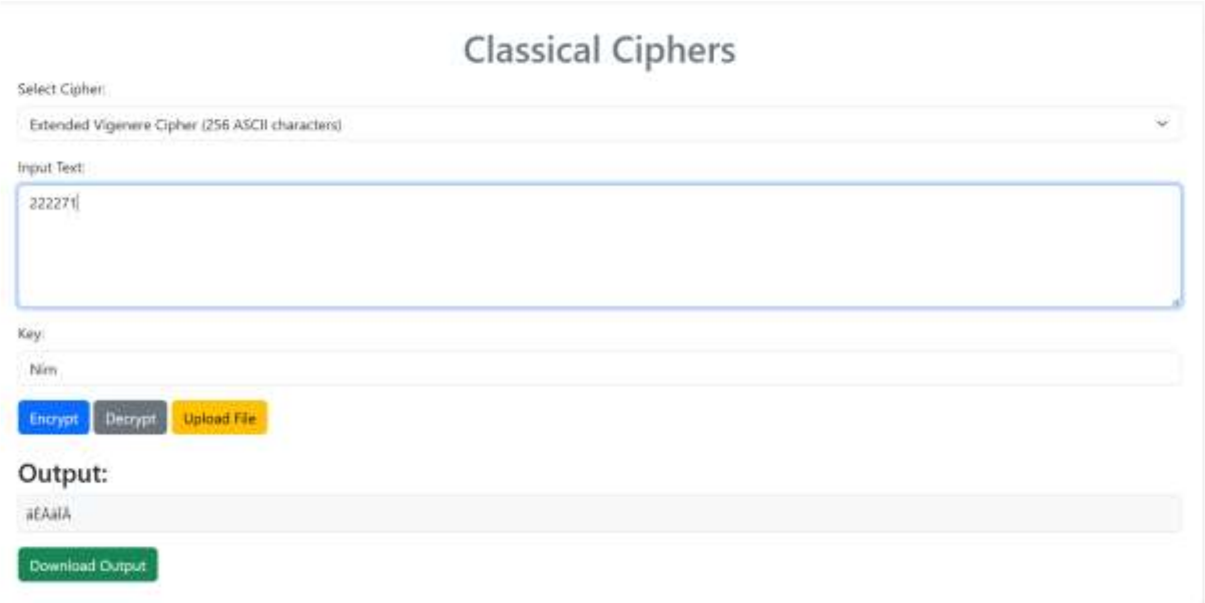
panggilan

Key:

wadu

Output:

LAQACIOUJ



Classical Ciphers

Select Cipher:

Extended Vigenere Cipher (256 ASCII characters)

Input Text:

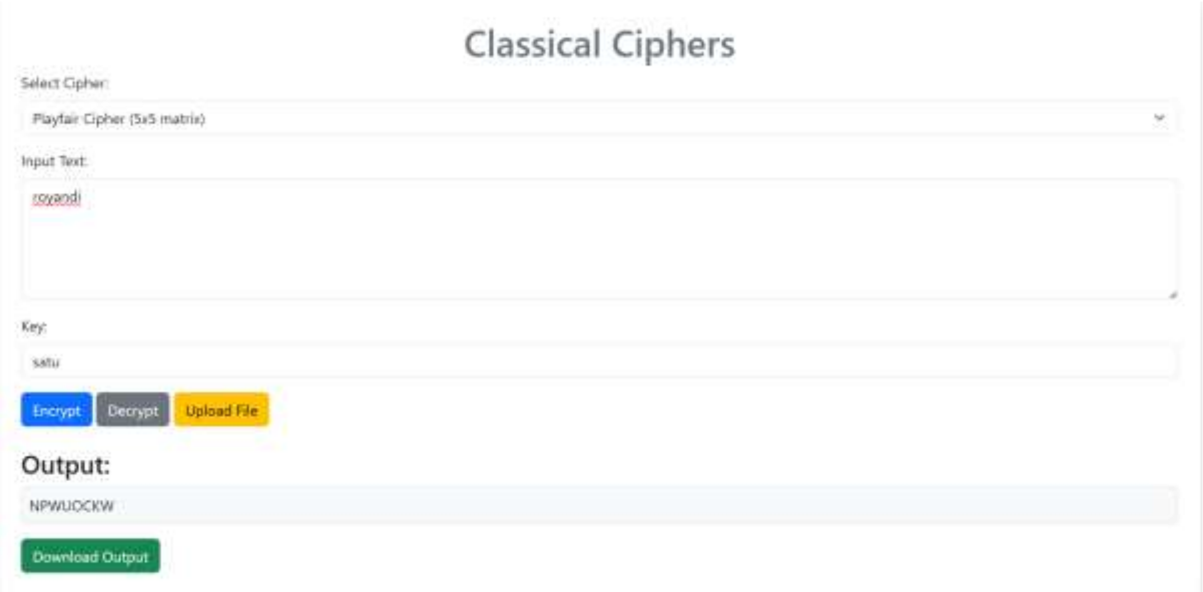
222271

Key:

Nim

Output:

aEAalA



Classical Ciphers

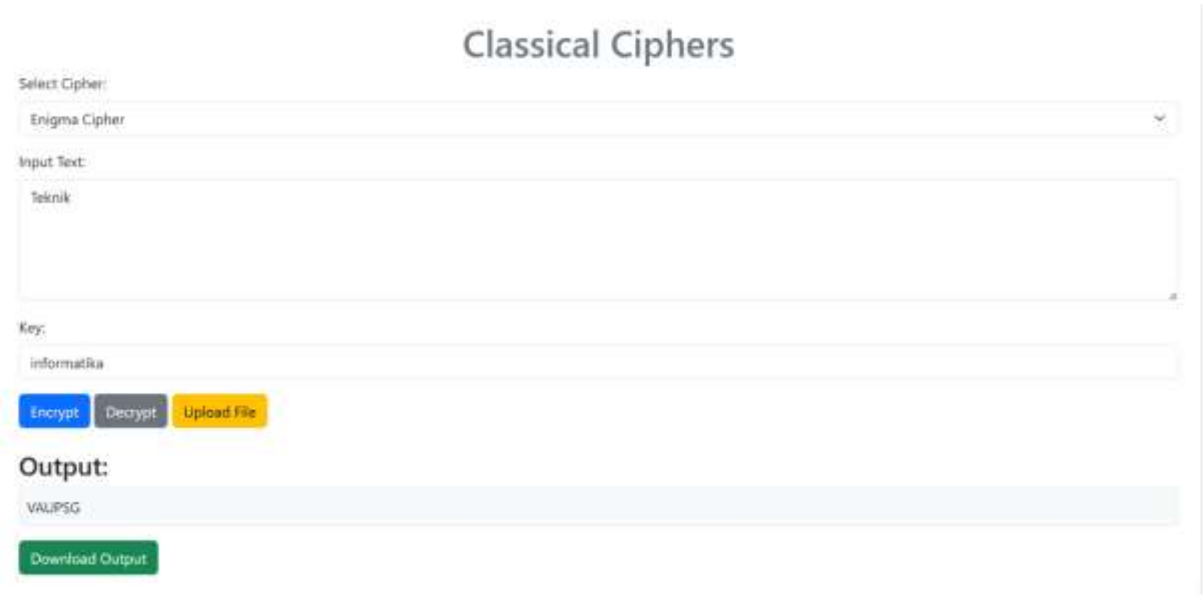
Select Cipher: Playfair Cipher (5x5 matrix)

Input Text: royandi

Key: satu

Output:

NPWUOCKW



Classical Ciphers

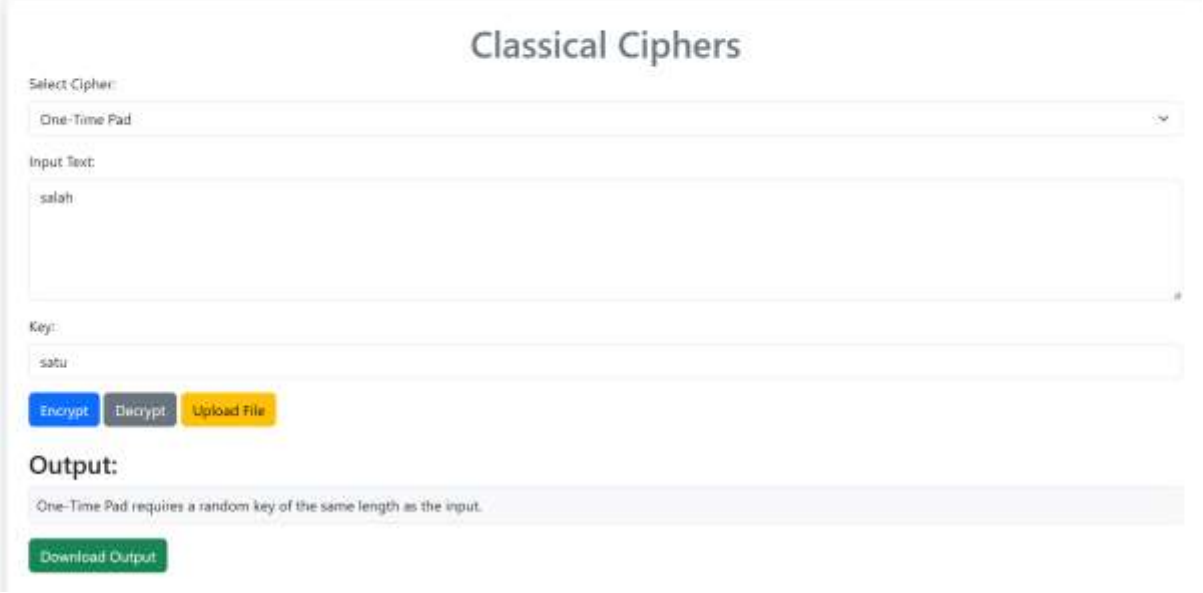
Select Cipher: Enigma Cipher

Input Text: Teknik

Key: informatika

Output:

VAUPSG



Classical Ciphers

Select Cipher: One-Time Pad

Input Text: salah

Key: satu

Output:

One-Time Pad requires a random key of the same length as the input.

royandixix/Tugas_cripto (github.com)