# PROJECT : TEXT ENCRYPTION – DECRYPTION

( _Explainer file_ )

Summary at the end.

Index.html :

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Text Encryption - Decryption</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Text Encryption - Decryption</h1>
        <div class="input-section">
            <label for="inputText"> Input text </label>
            <input type="text" id="inputText" placeholder=" Enter text here...">
        </div>
        <div class="button-section">
            <button id="encryptBtn">Encrypt</button>
            <button id="decryptBtn">Decrypt</button>
        </div>
        <div class="output-section">
            <label for="outputText"> Output Generated </label>
            <textarea id="outputText" rows="4" cols="50" readonly></textarea>
        </div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

HTML for structure of webpage.

Input field

Buttons

o/p field

style.css :

```css
body {
    font-family: Arial, sans-serif;
    background-image: url("backgnd.jpg");
    background-size: cover;
    background-position: center;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
```

CSS for styling.

```
    margin: 0;
}

.container {
    background-color: rgba(255, 255, 255, 0.4); /* Transparent background */
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    max-width: 500px;
    text-align: center;
    color: #fff; /* Text color */
    position: absolute;
    right: 20px; /* Adjust as needed */
    top: 50%; /* Center vertically */
    transform: translateY(-50%);
}

h1 {
    margin-bottom: 20px;
    color: #030303; /* Text color */
    text-shadow: 2px 2px 4px rgba(255, 255, 255, 0.5);
}

.input-section, .button-section, .output-section {
    margin-bottom: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
    color: #000000; /* Text color */
    text-shadow: 2px 2px 4px rgba(255, 255, 255, 0.5); /* Text shadow */
    backdrop-filter: blur(1.5px); /* Blur the background behind the label */
    background-color: rgba(255, 255, 255, 0.3); /* Semi-transparent white background for
readability */
    padding: 5px; /* Add padding for spacing */
}

input[type="text"], textarea {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
    color: #fff; /* Text color */
    background-color: rgba(255, 255, 255, 0.8); /* Semi-transparent white background for
input fields */
    color : #000000;
}
```

styling the form where user interact

Header for project name

labels for user convenience.

```css
button {
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    background-color: #ffffff;
    color: #000000;
    cursor: pointer;
    margin: 0 5px;
    font-size: 16px;
}

button:hover {
    background-color: wheat;
}
```

Buttons

Script.js :

```
Encrpyt input text and display result to o/p area.

document.getElementById('encryptBtn').addEventListener('click', async () => {
    const inputText = document.getElementById('inputText').value;
    const encryptedText = await encrypt(inputText);
    document.getElementById('outputText').value = encryptedText;
});

Decrpyt input text and display result to o/p area.

document.getElementById('decryptBtn').addEventListener('click', async () => {
    const inputText = document.getElementById('inputText').value;
    const decryptedText = await decrypt(inputText);
    document.getElementById('outputText').value = decryptedText;
});

Event listeners

const IV_LENGTH = 16; // For AES, this is always 16
const LOCAL_STORAGE_KEY = 'secureEncryptionKey';
const MASTER_KEY = 'masterPassword1234'; // Replace this with a secure passphrase and never hard-code in production

Constants (variables of fixed values)

// Helper function to convert array to hex string
function arrayBufferToHex(buffer) {
    return Array.from(new Uint8Array(buffer)).map(b => b.toString(16).padStart(2, '0')).join('');
}

Converts array buffer to hexadecimal string.

All these are javascript inbuilt functions.

// Helper function to convert hex string to array
function hexToArrayBuffer(hex) {
    let bytes = new Uint8Array(hex.match(/.{1,2}/g).map(byte => parseInt(byte, 16)));
```

```
    return bytes.buffer;
}

// Encrypt data using a passphrase
async function encryptWithPassphrase(data, passphrase) {
    let passphraseKey = await crypto.subtle.importKey(
        'raw',
        new TextEncoder().encode(passphrase),
        { name: 'PBKDF2' },
        false,
        ['deriveKey']
    );
    let salt = crypto.getRandomValues(new Uint8Array(16));
    let keyMaterial = await crypto.subtle.deriveKey(
        {
            name: 'PBKDF2',
            salt: salt,
            iterations: 100000,
            hash: 'SHA-256'
        },
        passphraseKey,
        { name: 'AES-GCM', length: 256 },
        true,
        ['encrypt', 'decrypt']
    );
    let iv = crypto.getRandomValues(new Uint8Array(IV_LENGTH));
    let encryptedData = await crypto.subtle.encrypt(
        { name: 'AES-GCM', iv: iv },
        keyMaterial,
        data
    );
    return { encryptedData, iv, salt };
}

// Decrypt data using a passphrase
async function decryptWithPassphrase(encryptedData, iv, salt, passphrase) {
    let passphraseKey = await crypto.subtle.importKey(
        'raw',
        new TextEncoder().encode(passphrase),
        { name: 'PBKDF2' },
        false,
        ['deriveKey']
    );
    let keyMaterial = await crypto.subtle.deriveKey(
        {
            name: 'PBKDF2',
            salt: salt,
            iterations: 100000,
            hash: 'SHA-256'
        },
```

func. to Convert hexadecimal string to array buffe

To encrypt

to derive enc. key

Asynchronous func. i.e it'll allow 'await'

Importing Passphrase key

method to import key in web crypto API

//Salt

Deriving key

//initialisation vector

encrypt data

//return

//Already we saw

```
        passphraseKey,
        { name: 'AES-GCM', length: 256 },
        true,
        ['encrypt', 'decrypt']
    );
    let decryptedData = await crypto.subtle.decrypt(
        { name: 'AES-GCM', iv: iv },
        keyMaterial,
        encryptedData
    );
    return decryptedData;
}

// Store the encryption key securely in localStorage
async function storeEncryptionKey(key) {
    const { encryptedData, iv, salt } = await encryptWithPassphrase(key, MASTER_KEY);
    localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify({
        encryptedData: arrayBufferToHex(encryptedData),
        iv: arrayBufferToHex(iv),
        salt: arrayBufferToHex(salt)
    }));
}

// Retrieve the encryption key from localStorage
async function getEncryptionKey() {
    let storedKeyData = localStorage.getItem(LOCAL_STORAGE_KEY);
    if (!storedKeyData) {
        const key = crypto.getRandomValues(new Uint8Array(32)); // Generate new key
        await storeEncryptionKey(key);
        return key;
    }
    storedKeyData = JSON.parse(storedKeyData);
    const encryptedData = hexToArrayBuffer(storedKeyData.encryptedData);
    const iv = hexToArrayBuffer(storedKeyData.iv);
    const salt = hexToArrayBuffer(storedKeyData.salt);
    const key = await decryptWithPassphrase(encryptedData, iv, salt, MASTER_KEY);
    return new Uint8Array(key);
}

async function encrypt(text) {
    const key = await getEncryptionKey();
    let iv = crypto.getRandomValues(new Uint8Array(IV_LENGTH));
    let cryptoKey = await crypto.subtle.importKey(
        'raw',
        key,
        { name: 'AES-CBC' },
        false,
        ['encrypt']
    );
    let encodedText = new TextEncoder().encode(text);
```

// storing in browsers local storage

// get key

// else generate if not found

// Separate datas

// get enc. key

// Generate random initialisation vector (As project demands, to give dif enc. text for same input).

Import passphrase key

```
    let encrypted = await crypto.subtle.encrypt(
        { name: 'AES-CBC', iv: iv },
        cryptoKey,
        encodedText
    );
    let ivHex = arrayBufferToHex(iv);
    let encryptedHex = arrayBufferToHex(new Uint8Array(encrypted));
    return ivHex + ':' + encryptedHex;
}

async function decrypt(text) {
    const key = await getEncryptionKey();
    let textParts = text.split(':');
    let iv = hexToArrayBuffer(textParts[0]);
    let encryptedText = hexToArrayBuffer(textParts[1]);
    let cryptoKey = await crypto.subtle.importKey(
        'raw',
        key,
        { name: 'AES-CBC' },
        false,
        ['decrypt']
    );
    let decrypted = await crypto.subtle.decrypt(
        { name: 'AES-CBC', iv: iv },
        cryptoKey,
        encryptedText
    );
    return new TextDecoder().decode(decrypted);
}
```

encrypt

Return encrypted hexadecimal text and iv vector

import key from web Crypto API

decrypt

return

Summary :

Overview:

This document provides a detailed explanation of the functionality and structure of the Text Encryption-Decryption Tool, along with a step-by-step breakdown of the JavaScript code responsible for its operation.

**index.html**

The index.html file defines the structure of the web page interface for the encryption-decryption tool. It includes:

*Structure:*

HTML5 doctype and language declaration.

Meta tags for character set and viewport settings.

Title and external stylesheet link (style.css).

*Content:*

A container (div.container) styled to center on the page with a semi-transparent background.

Header (h1) for the tool's title.

Input section (div.input-section) with a label and input field for text input.

Button section (div.button-section) containing 'Encrypt' and 'Decrypt' buttons.

Output section (div.output-section) with a label and textarea for displaying encrypted or decrypted output.

Script Inclusion: Link to script.js for handling encryption and decryption logic.

**script.js**

The script.js file contains the JavaScript code that handles the encryption and decryption operations using the AES-CBC algorithm with PBKDF2 for key derivation and AES-GCM for data encryption. Here's a breakdown of its components:

*Event Listeners:*

encryptBtn listener: Encrypts input text using the encrypt function and displays the result in the output textarea.

decryptBtn listener: Decrypts input text using the decrypt function and displays the result in the output textarea.

*Constants:*

IV_LENGTH: Specifies the initialization vector length (16 bytes) required for AES operations.

LOCAL_STORAGE_KEY: Key for storing the encrypted encryption key in localStorage.

MASTER_KEY: Master passphrase used for key derivation and decryption operations.

### *Helper Functions:*

arrayBufferToHex(buffer): Converts an array buffer to a hexadecimal string.

hexToArrayBuffer(hex): Converts a hexadecimal string back to an array buffer.

Encryption and Decryption Functions:

encryptWithPassphrase(data, passphrase): Encrypts data using a passphrase, employing PBKDF2 for key derivation and AES-GCM for encryption.

decryptWithPassphrase(encryptedData, iv, salt, passphrase): Decrypts data using a passphrase, with similar key derivation and AES-GCM decryption process.

storeEncryptionKey(key): Stores an encryption key securely in localStorage after encrypting it using the master passphrase.

getEncryptionKey(): Retrieves the encryption key from localStorage, generating and storing a new one if none exists.

### *Encrypt and Decrypt Functions:*

encrypt(text): Encrypts input text using AES-CBC encryption with a randomly generated IV, returning the IV and encrypted text as a concatenated string.

decrypt(text): Decrypts input text that contains IV and encrypted data, using the stored encryption key.

---.