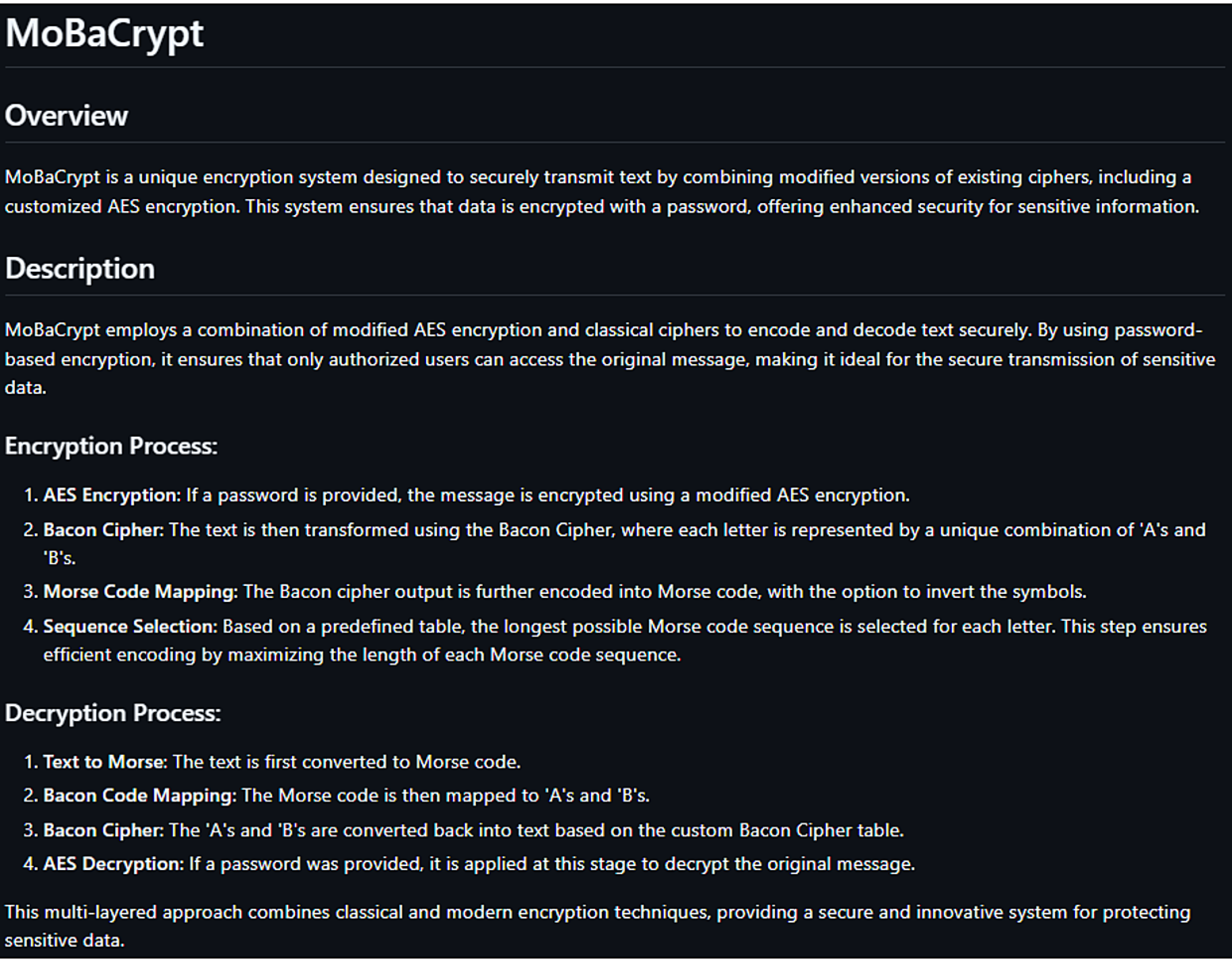


MoBaCrypt

Overview

MoBaCrypt is a unique encryption system designed to securely transmit text by combining modified versions of existing ciphers, including a customized AES encryption. This system ensures that data is encrypted with a password, offering enhanced security for sensitive information.

Description

MoBaCrypt employs a combination of modified AES encryption and classical ciphers to encode and decode text securely. By using password-based encryption, it ensures that only authorized users can access the original message, making it ideal for the secure transmission of sensitive data.

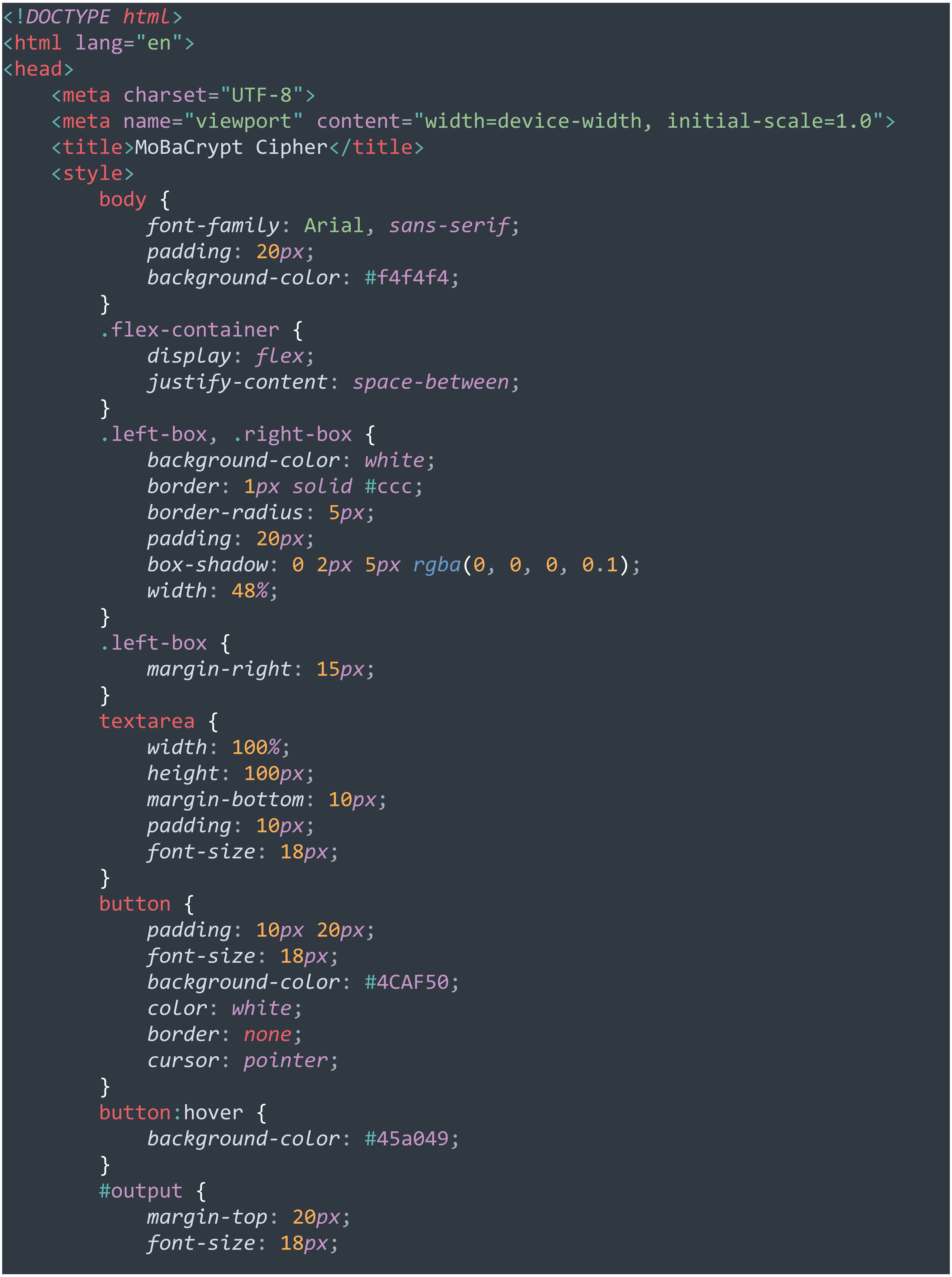
Encryption Process:

1. **AES Encryption:** If a password is provided, the message is encrypted using a modified AES encryption.
2. **Bacon Cipher:** The text is then transformed using the Bacon Cipher, where each letter is represented by a unique combination of 'A's and 'B's.
3. **Morse Code Mapping:** The Bacon cipher output is further encoded into Morse code, with the option to invert the symbols.
4. **Sequence Selection:** Based on a predefined table, the longest possible Morse code sequence is selected for each letter. This step ensures efficient encoding by maximizing the length of each Morse code sequence.

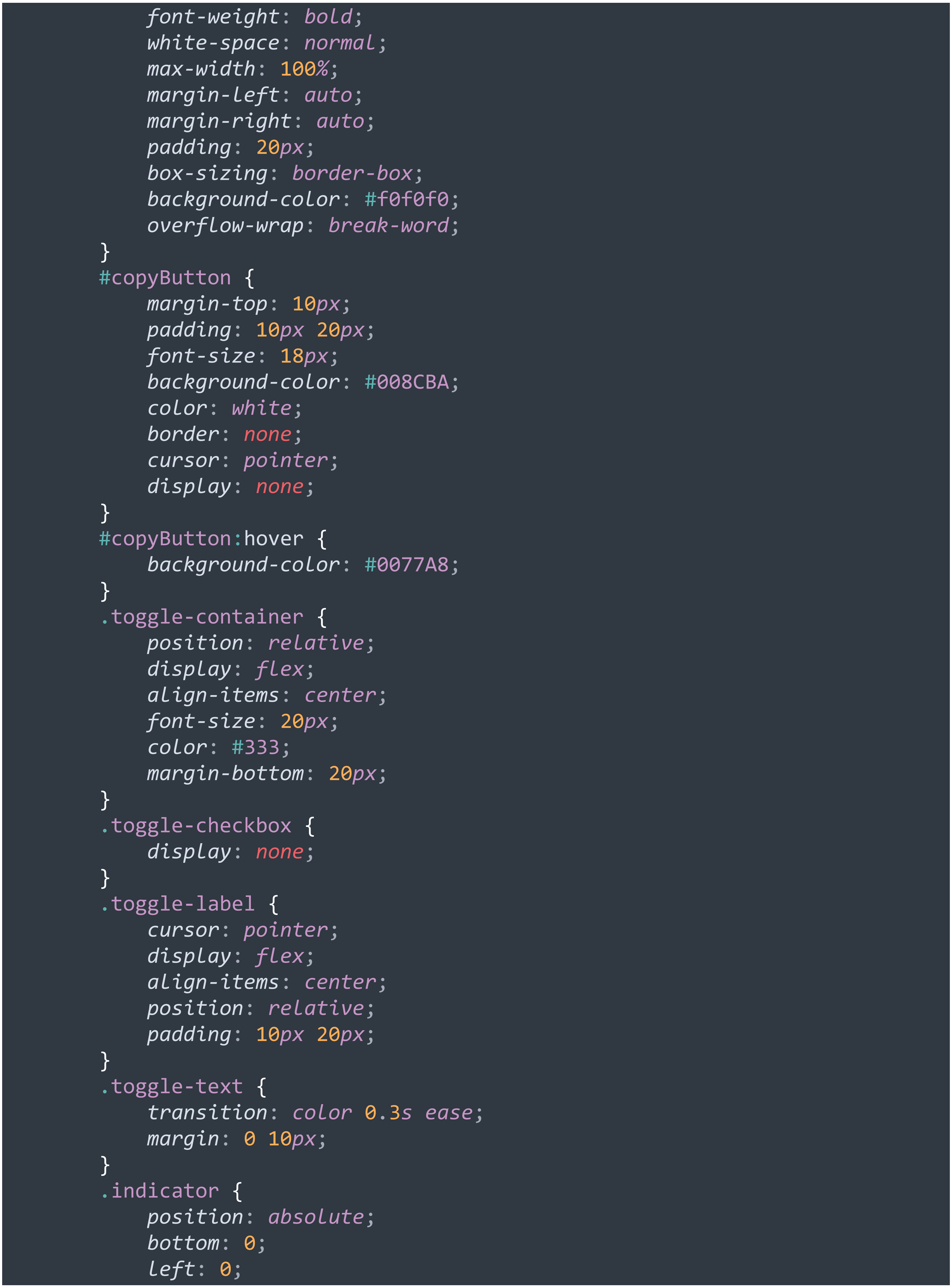
Decryption Process:

1. **Text to Morse:** The text is first converted to Morse code.
2. **Bacon Code Mapping:** The Morse code is then mapped to 'A's and 'B's.
3. **Bacon Cipher:** The 'A's and 'B's are converted back into text based on the custom Bacon Cipher table.
4. **AES Decryption:** If a password was provided, it is applied at this stage to decrypt the original message.

This multi-layered approach combines classical and modern encryption techniques, providing a secure and innovative system for protecting sensitive data.



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MoBaCrypt Cipher</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            padding: 20px;
            background-color: #f4f4f4;
        }
        .flex-container {
            display: flex;
            justify-content: space-between;
        }
        .left-box, .right-box {
            background-color: white;
            border: 1px solid #ccc;
            border-radius: 5px;
            padding: 20px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
            width: 48%;
        }
        .left-box {
            margin-right: 15px;
        }
        textarea {
            width: 100%;
            height: 100px;
            margin-bottom: 10px;
            padding: 10px;
            font-size: 18px;
        }
        button {
            padding: 10px 20px;
            font-size: 18px;
            background-color: #4CAF50;
            color: white;
            border: none;
            cursor: pointer;
        }
        button:hover {
            background-color: #45a049;
        }
        #output {
            margin-top: 20px;
            font-size: 18px;
```



```
            font-weight: bold;
            white-space: normal;
            max-width: 100%;
            margin-left: auto;
            margin-right: auto;
            padding: 20px;
            box-sizing: border-box;
            background-color: #f0f0f0;
            overflow-wrap: break-word;
        }
        #copyButton {
            margin-top: 10px;
            padding: 10px 20px;
            font-size: 18px;
            background-color: #008CBA;
            color: white;
            border: none;
            cursor: pointer;
            display: none;
        }
        #copyButton:hover {
            background-color: #0077A8;
        }
        .toggle-container {
            position: relative;
            display: flex;
            align-items: center;
            font-size: 20px;
            color: #333;
            margin-bottom: 20px;
        }
        .toggle-checkbox {
            display: none;
        }
        .toggle-label {
            cursor: pointer;
            display: flex;
            align-items: center;
            position: relative;
            padding: 10px 20px;
        }
        .toggle-text {
            transition: color 0.3s ease;
            margin: 0 10px;
        }
        .indicator {
            position: absolute;
            bottom: 0;
            left: 0;
            height: 4px;
            width: 50%;
            background-color: #007BFF;
            transition: left 0.3s ease;
        }
        .toggle-checkbox:checked + .toggle-label .indicator {
            left: 50%;
        }
        .inputmess {
            height: 500px;
            width: 850px;
            resize: vertical;
        }
        .inputpass {
            height: 50px;
            width: 875px;
            resize: vertical;
        }
    </style>
</head>
<body>
    <div class="flex-container">
        <div class="left-box">
            <textarea id="inputText" placeholder="Enter your text here"
class="inputmess"></textarea>
        </div>
        <div class="right-box">
            <div class="toggle-container">
                <input type="checkbox" id="toggle" class="toggle-checkbox">
                <label for="toggle" class="toggle-label">
                    <span class="toggle-text decode">Encode</span>
                    <span class="toggle-text encode">Decode</span>
                    <div class="indicator"></div>
                </label>
            </div>
            <textarea id="password" placeholder="Optional: Enter password"
class="inputpass"></textarea>
            <label for="layering">Layering:</label>
            <input type="number" id="layering" min="1" max="10" value="1"
oninput="this.value = Math.max(1, this.value)">
            <br>
            <h3>Advanced Settings (Must be the same for Encrypting and
Decoding)</h3>
            <hr>
            <label for="tableShift1">Table Shift:</label>
            <input type="number" id="tableShift1" min="0" max="50" value="0"
placeholder="Shift 1">
            <input type="number" id="tableShift2" min="0" max="50" value="0"
placeholder="Shift 2">
```

```
            <br>
            <label for="inverseTable">Inverse Table:</label>
            <input type="checkbox" id="inverseTable">
            <br><br>
            <button onclick="funcheck()">Run</button>
        </div>
    </div>
    <br>
    <button id="copyButton" onclick="copyToClipboard()">Copy</button>
    <div id="output"></div>
    <script type="text/javascript">
        function copyToClipboard() {
            const outputText = document.getElementById('output').innerText;
            if (!outputText) return;

            navigator.clipboard.writeText(outputText).then(() => {
            }).catch(err => {
                console.error("Failed to copy: ", err);
            });
        }

        async function funcheck() {
            const toggle = document.getElementById('toggle');
            const pass = document.getElementById('password').value;
            const mess = document.getElementById('inputText').value;
            const layers = Math.max(1,
parseInt(document.getElementById('layering').value) || 1);
            const tableShift1 =
parseInt(document.getElementById('tableShift1').value) || 0;
            const tableShift2 =
parseInt(document.getElementById('tableShift2').value) || 0;
            const inverseTable =
document.getElementById('inverseTable').checked ? 1 : 0;
            const outputElement = document.getElementById('output');
            const copyButton = document.getElementById('copyButton');

            if (!mess) {
                outputElement.innerText = "Please enter text to encode or
decode.";
                copyButton.style.display = "none";
                return;
            }

            const action = toggle.checked ? "decode" : "encode";
            const result = await encryptDecryptMessage(action, layers,
tableShift1, tableShift2, inverseTable, mess, pass);

            outputElement.innerText = result;
            copyButton.style.display = result ? "block" : "none";
        }
    </script>
    <script src="js.js"></script>
</body>
</html>
```

```
async function encryptDecryptMessage(action, layer, tb1offset, tb2offset, inver, message, password) {
    let finalResult = message; // Initialize finalResult with the initial message

    // Morse Code Map
    const morseCodeMap = {
        '.-': 'A',     '-...': 'B',   '-.-.': 'C',   '-..': 'D',    '.': 'E',
        '..-.': 'F',   '--.': 'G',    '....': 'H',   '..': 'I',     '.---': 'J',
        '-.-': 'K',    '.-..': 'L',   '--': 'M',     '-.': 'N',     '---': 'O',
        '.--.': 'P',   '--.-': 'Q',   '.-.': 'R',    '...': 'S',    '-': 'T',
        '..-': 'U',    '...-': 'V',   '.--': 'W',    '-..-': 'X',   '-.--': 'Y',
        '--..': 'Z',
        '.----': '1',  '..---': '2',  '...--': '3',  '....-': '4',   '.....': '5',
        '-....': '6',  '--...': '7',   '---..': '8',  '----.': '9',   '-----': '0',
        '.-.-.-': '.', '--..--': ',', '..--..': '?', '.----.': "'",   '-.-.--': '!',
        '-..-.': '/',  '-.--.': '(',   '-.--.-': ')', '.-...': '&',    '---...': ':',
        '-.-.-.': ';', '-...-': '=',   '.-.-.': '+',   '-....-': '-',   '..--.-': '_',
        '.-..-.': '"', '...-..-': '$',  '.--.-.': '@',  ' ': ' '
    };

    // Bacon Cipher Key
    const baconCipherKey = {
        'A': 'AAAAAAA', 'B': 'AAAAAAB', 'C': 'AAAAABA', 'D': 'AAAAABB', 'E': 'AAAABAA',
        'F': 'AAAABAB', 'G': 'AAAABBA', 'H': 'AAAABBB', 'I': 'AAABAAA', 'J': 'AAABAAB',
        'K': 'AAABABA', 'L': 'AAABABB', 'M': 'AAABBAA', 'N': 'AAABBAB', 'O': 'AAABBBA',
        'P': 'AAABBBB', 'Q': 'AABAAAA', 'R': 'AABAAAB', 'S': 'AABAABA', 'T': 'AABAABB',
        'U': 'AABABAA', 'V': 'AABABAB', 'W': 'AABABBA', 'X': 'AABABBB', 'Y': 'AABBAAA',
        'Z': 'AABBAAB', '0': 'AABBABA', '1': 'AABBABB', '2': 'AABBBAA', '3': 'AABBBAB',
        '4': 'AABBBBA', '5': 'AABBBBB', '6': 'ABAAAAA', '7': 'ABAAAAB', '8': 'ABAAABA',
        '9': 'ABAAABB', '~': 'ABAABAA', '!': 'ABAABAB', '@': 'ABAABBA', '#': 'ABAABBB',
        '$': 'ABABAAA', '%': 'ABABAAB', '^': 'ABABABA', '&': 'ABABABB', '*': 'ABABBAA',
        '(': 'ABABBAB', ')': 'ABABBBA', '-': 'ABABBBB', '_': 'ABBAAAA', '=': 'ABBAAAB',
        '+': 'ABBAABA', '[': 'ABBAABB', ']': 'ABBABAA', '{': 'ABBABAB', '}': 'ABBABBA',
        '\\': 'ABBABBB', '|': 'ABBBAAA', ';': 'ABBBAAB', ':': 'ABBBABA', '\'': 'ABBBABB',
        '"': 'ABBBBAA', ',': 'ABBBBAB', '<': 'ABBBBBA', '.': 'ABBBBBB', '>': 'BAAAAAA',
        '/': 'BAAAAAB', '?': 'BAAAABA', '`': 'BAAAABB', ' ': 'BAAABAA',
        'a': 'BAAABAB', 'b': 'BAAABBA', 'c': 'BAAABBB', 'd': 'BAABAAA', 'e': 'BAABAAB',
        'f': 'BAABABA', 'g': 'BAABABB', 'h': 'BAABBAA', 'i': 'BAABBAB', 'j': 'BAABBBA',
        'k': 'BAABBBB', 'l': 'BABAAAA', 'm': 'BABAAAB', 'n': 'BABAABA', 'o': 'BABAABB',
        'p': 'BABABAA', 'q': 'BABABAB', 'r': 'BABABBA', 's': 'BABABBB', 't': 'BABBAAA',
        'u': 'BABBAAB', 'v': 'BABBABA', 'w': 'BABBABB', 'x': 'BABBBAA', 'y': 'BABBBAB',
        'z': 'BABBBBA'
    };

    // Function to apply offset to a given table
    function applyOffset(table, offset) {
        const keys = Object.keys(table);
        const values = Object.values(table);
        const newTable = {};

        for (let i = 0; i < keys.length; i++) {
            const newIndex = (i + offset) % keys.length; // Wrap around using modulo
            newTable[keys[newIndex]] = values[i];
        }
        return newTable;
    }

    // Apply offsets to Morse and Bacon tables
    const morseCodeMapWithOffset = applyOffset(morseCodeMap, tb1offset);
    const baconCipherKeyWithOffset = applyOffset(baconCipherKey, tb2offset);

    function stringToArrayBuffer(str) {
        const encoder = new TextEncoder();
        return encoder.encode(str);
    }
    function arrayBufferToString(buffer) {
        const decoder = new TextDecoder();
        return decoder.decode(buffer);
    }
    async function hashPassword(password) {
        const encoder = new TextEncoder();
        const passwordBuffer = encoder.encode(password);
        const hashBuffer = await crypto.subtle.digest('SHA-256', passwordBuffer);
        return new Uint8Array(hashBuffer);
```

```
}
async function deriveKey(passwordHash, salt) {
    const saltBuffer = stringToArrayBuffer(salt);
    const keyMaterial = await crypto.subtle.importKey(
        "raw",
        passwordHash,
        { name: "PBKDF2" },
        false,
        ["deriveKey"]
    );

    return await crypto.subtle.deriveKey(
        { name: "PBKDF2", salt: saltBuffer, iterations: 100000, hash: "SHA-256" },
        keyMaterial,
        { name: "AES-GCM", length: 256 },
        false,
        ["encrypt", "decrypt"]
    );
}

for (let i = 0; i < layer; i++) {
    if (action === "encode") {
        if (password !== undefined && password !== "") {
            const salt = crypto.getRandomValues(new Uint8Array(16)); // 16 bytes salt
            const iv = crypto.getRandomValues(new Uint8Array(12)); // 12 bytes IV for AES-GCM
            const passwordHash = await hashPassword(password);
            const key = await deriveKey(passwordHash, salt);

            const encodedMessage = stringToArrayBuffer(finalResult);
            const cipherText = await crypto.subtle.encrypt(
                { name: "AES-GCM", iv: iv },
                key,
                encodedMessage
            );

            // Convert to base64 for easier handling
            const base64CipherText = btoa(String.fromCharCode(...new Uint8Array(cipherText)));
            const base64Salt = btoa(String.fromCharCode(...salt));
            const base64Iv = btoa(String.fromCharCode(...iv));

            // Concatenate the salt, iv, and ciphertext in the output format (without the password hash)
            finalResult = `${base64Salt}.${base64Iv}.${base64CipherText}`;
        }

        let baconOutput = '';

        // Convert input text to Bacon Cipher using the offset table
        for (const char of finalResult) {
            baconOutput += baconCipherKeyWithOffset[char] || ''; // Use logical OR to skip characters not in the ke
        }
        if (inver == 0) {baconOutput = baconOutput.replace(/A/g, '.').replace(/B/g, '-');}
        else{baconOutput = baconOutput.replace(/A/g, '-').replace(/B/g, '.');}
        let output = '';
        for (let i = 0; i < baconOutput.length;) {
            let morseChar = '';
            let found = false;

            // Check for the longest valid Morse code character
            for (let j = 1; j <= 5 && i + j <= baconOutput.length; j++) {
                const subStr = baconOutput.substring(i, i + j);
                if (morseCodeMapWithOffset[subStr] !== undefined) {
                    morseChar = subStr; // Update morseChar to the latest valid substring
                    found = true; // Mark that we found a valid Morse code character
                }
            }

            if (found) {
                output += morseCodeMapWithOffset[morseChar]; // Append the corresponding character to output
                i += morseChar.length; // Move the index forward by the length of the found Morse code
            } else {
                i++; // If no valid Morse code was found, just move to the next character
            }
        }
```

```
            finalResult = output; // Update finalResult with the output of this iteration
        } else if (action === "decode") {
            let output = '';

            // Reverse the morseCodeMap to create a lookup for decoding
            const reverseMorseCodeMap = Object.fromEntries(
                Object.entries(morseCodeMapWithOffset).map(([morse, char]) => [char, morse])
            );

            // Convert each character to Morse code
            for (let char of finalResult) {
                if (reverseMorseCodeMap[char]) {
                    output += reverseMorseCodeMap[char]; // No space between Morse codes
                }
            }

            // Convert '.' to 'A' and '-' to 'B'
            if (inver == 0) {output = output.replace(/\./g, 'A').replace(/-/g, 'B');}
            else {output = output.replace(/\./g, 'B').replace(/-/g, 'A');}

            console.log("more " + output);
            const reverseBaconCipherKey = Object.fromEntries(
                Object.entries(baconCipherKeyWithOffset).map(([key, value]) => [value, key])
            );

            // Split the input string into chunks of 6 characters
            const chunks = output.match(/.{1,7}/g);

            // Decode each chunk using the reverse cipher key
            const decodedCharacters = chunks.map(chunk => reverseBaconCipherKey[chunk] || '');

            // Join the decoded characters into a final string
            finalResult = decodedCharacters.join('');

            if (password !== undefined && password !== "") {
                console.log("noPass");
                const parts = finalResult.split('.');

                const salt = new Uint8Array(atob(parts[0]).split("").map(c => c.charCodeAt(0)));
                const iv = new Uint8Array(atob(parts[1]).split("").map(c => c.charCodeAt(0)));
                const cipherText = new Uint8Array(atob(parts[2]).split("").map(c => c.charCodeAt(0)));

                const hashedPassword = await hashPassword(password);
                const key = await deriveKey(hashedPassword, salt);

                try {
                    const decrypted = await crypto.subtle.decrypt(
                        { name: "AES-GCM", iv: iv },
                        key,
                        cipherText
                    );
                    finalResult = arrayBufferToString(decrypted);
                } catch (e) {
                    console.error("Decryption failed", e);
                    finalResult = "Decryption failed!";
                }
            } else {
                finalResult = finalResult; // If no password, return the decoded message
            }
        } else {
            return "Invalid action!";
        }
    }

    return finalResult; // Return the final result at the end
}
```