## 开发文档

这是一个用于 S-DES（Simplified Data Encryption Standard）加密和解密的 HTML 页面和 JavaScript 代码。以下是对代码的解释和文档：

**HTML 结构：**

1. `<!DOCTYPE html>`: HTML5 的文档类型声明

2. `<html lang="zh-CN">`: HTML 根元素，指定页面语言为中文

3. `<head>`: 包含页面的头部信息，如字符集和标题

- `<meta charset="UTF-8">`: 指定字符集为 UTF-8。

- `<title>S-DES Encryption & Decryption</title>`: 页面标题。

- `<style>`: 包含页面的 CSS 样式。

4. `<script>`: 包含 JavaScript 代码。

5. `<body>`: HTML 页面的主体部分。

·`<h1>`: 页面标题，显示"S-DES Encryption & Decryption"。

·`<div class="form-container">`: 包含整个表单的容器。

·表单中有一些输入框、下拉框、按钮等元素，用于用户输入和操作。

·有两个输入框，分别用于输入明文和密钥。

·用户可以选择输入的形式是 Binary 还是 ASCII。

·有按钮用于加密、解密、刷新和暴力破解。

·最后有一个输出框，用于显示加密或解密后的结果。

**JavaScript 代码：**

1. 定义了 S-DES 加解密过程中需要使用的 S 盒数据、初始置换盒、最终置换盒等。

  const data1 = {

        '00': ['01', '11', '00', '11'],

        '01': ['00', '10', '10', '01'],

        '10': ['11', '01', '01', '00'],

        '11': ['10', '00', '11', '10']

    };

    const data2 = {

        '00': ['00', '10', '11', '10'],

        '01': ['01', '11', '00', '01'],

        '10': ['10', '01', '01', '00'],

        '11': ['11', '00', '10', '11']

    };

    // 进行加解密的函数

2. 定义了加解密过程中的核心函数 `encrypt\_decrypt`，以及用于生成密钥的函数 `get\_keys`。

 function encrypt\_decrypt(subplaintext, key, flag) {

        // step1: 切割成左右两个小分组

        let left = subplaintext.slice(0, 4);

        let right = subplaintext.slice(4);

        // step2: 对右小组做4-8bit的拓展置换

        const EPBox = [3, 0, 1, 2, 1, 2, 3, 0]; // 注意 JavaScript 的数组下标从 0 开始[41232341]

        let tmpRight = "";

        EPBox.forEach(index => {

            tmpRight += right[index];

        });

        // step3: 用轮密钥与右分组做异或处理

        let tmpText = "";

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

            tmpText += tmpRight[i] !== key[i] ? '1' : '0';

        }

        // step4: 压缩替换 8-4bit

        const subsValue =

            data1[tmpText[1] + tmpText[2]][parseInt(tmpText[0] + tmpText[3], 2)] +

            data2[tmpText[5] + tmpText[6]][parseInt(tmpText[4] + tmpText[7], 2)];

        // step5: 直接替换

        const SPBox = [1, 3, 2, 0];//[2431]

        let tmp = "";

        SPBox.forEach(index => {

            tmp += subsValue[index];

        });

        // step6: 替换后的4-bit与left做异或处理得到加密后的左分组

        let enLeft = "";

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

            enLeft += tmp[i] !== left[i] ? '1' : '0';

        }

        // step7: 判断是否需要左右互换,最后输出

        return flag === 0 ? right + enLeft : enLeft + right;

    }

1. 定义了一些其他辅助函数，如置换函数 `IP` 和 `controyIP`，以及用于左移位的 `leftShift`。

    function IP(subplaintext) {

        var IP\_box = [2, 6, 3, 1, 4, 8, 5, 7];// 定义初始置换盒

        var textArray = subplaintext.split('');// 将输入的二进制字符串转换为字符数组

        var tmpText = [];// 初始化一个空数组来存储置换后的字符

        // 根据置换盒的规则，将原始数组中的字符按新的顺序添加到tmpText中

        for (var i = 0; i < IP\_box.length; i++) {

            tmpText.push(textArray[IP\_box[i] - 1]);  // 注意:因为索引从0开始，所以需要减1

        }

        var resText = tmpText.join('');// 将置换后的字符数组转换回字符串

        return resText;// 返回置换后的字符串

    }

    function controyIP(subplaintext) {

        var IP\_box = [4, 1, 3, 5, 7, 2, 8, 6];// 定义最终置换盒

        var textArray = subplaintext.split('');// 将输入的二进制字符串转换为字符数组

        var tmpText = [];// 初始化一个空数组来存储置换后的字符

        // 根据置换盒的规则，将原始数组中的字符按新的顺序添加到tmpText中

        for (var i = 0; i < IP\_box.length; i++) {

            tmpText.push(textArray[IP\_box[i] - 1]);  // 注意:因为索引从0开始，所以需要减1

        }

        var resText = tmpText.join('');// 将置换后的字符数组转换回字符串

        return resText;// 返回置换后的字符串

    }

4. `encryptORdecryptMessage` 函数用于根据用户的选择进行加密或解密。它调用了 `encrypt\_decrypt` 等核心函数，并更新页面上的输出框。

 function encryptORdecryptMessage(form,message,key,flag) {

        var keys = get\_keys(key);

        var key1 = keys.key1;

        var key2 = keys.key2;

        var plaintext = plainTextTransfer(message, form);

        let pt\_lst = split(plaintext);

        // 初始化结果数组

        let res = [];

        for (let i = 0; i < pt\_lst.length; i++) {

            // 调用IP函数对分组进行处理

            pt\_lst[i] = IP(pt\_lst[i]);

            // 调用func\_encrypt函数进行加密操作

            if (flag === 0) {

                pt\_lst[i] = encrypt\_decrypt(pt\_lst[i], key1, 0);

                pt\_lst[i] = encrypt\_decrypt(pt\_lst[i], key2, 1);

            }else if (flag === 1){

                pt\_lst[i] = encrypt\_decrypt(pt\_lst[i], key2, 0);

                pt\_lst[i] = encrypt\_decrypt(pt\_lst[i], key1, 1);

            }

            // 调用FP函数处理加密后的分组

            pt\_lst[i] = controyIP(pt\_lst[i]);

            // 将处理后的分组添加到结果数组中

            res.push(pt\_lst[i]);

        }

        // 将所有处理后的分组连接成一个字符串

        let finalResult = res.join('');

        finalResult = cipherTextTransfer(finalResult, form);

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

        return finalResult;

5. `isValidBinaryMessage` 和 `isValidKey` 用于验证输入的消息和密钥的格式是否正确。

  // 验证消息是否是有效的Binary消息，长度必须是8的整数倍且只包含0和1

    function isValidBinaryMessage(message) {

        if (message.length % 8 !== 0 || !/^[01]+$/.test(message)) {

            return false;

        }

        return true;

    }

    // 验证密钥是否是有效的Binary(10-bit)字符串

    function isValidKey(key) {

        if (key.length !== 10 || !/^[01]+$/.test(key)) {

            return false;

        }

        return true;

    }

6. `plainTextTransfer` 和 `cipherTextTransfer` 用于在 ASCII 和二进制之间转换明文和密文。

7. `split` 函数用于将明文分割成 8 位的分组。

   function split(plaintext) {

        // 初始化分组列表: 用于存储每一个8-bit子串

        let pt\_lst = [];

        let length = plaintext.length;  // 计算明文的长度

        if (length % 8 === 0) {  // 判断明文长度是否是8的倍数

            for (let i = 0; i < length; i+=8) {  // 如果是，则直接分组

                pt\_lst.push(plaintext.substring(i, i+8));

            }

        } else {

            for (let i = 0; i < length - 8; i+=8) {  // 如果不是，则除了最后一个分组，其他都直接分组

                pt\_lst.push(plaintext.substring(i, i+8));

            }

            let rest = length % 8;  // 计算最后一个分组中实际字符的个数

            // 对最后一个分组进行处理，不足的部分用'0'进行补齐

            pt\_lst.push('0'.repeat(8 - rest) + plaintext.substring(length - rest));

        }

        return pt\_lst;  // 返回分组列表

    }

8. `filterDecrypt` 函数用于暴力破解时的过滤，根据之前验证的密钥来减少可能的密钥范围。

`bruteForceDecrypt` 函数用于进行暴力破解攻击，找出可能的密钥。

   function filterDecrypt(form, plaintext, ciphertext) {

        var validKeys = [];  // 存储所有有效的密钥

        let keysToRemove = [];

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

            if (i === 0) {

                // 对于第一个明文密文对，尝试所有可能的密钥

                for (let key = 0; key < 1024; key++) {

                    let binaryKey = key.toString(2).padStart(10, '0');  // 将密钥转换为二进制字符串，保留前导零

                    let decryptedText = encryptORdecryptMessage(form ,ciphertext[i], binaryKey, 1);  // 使用密钥解密密文

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

                    if (decryptedText === plaintext[i]) {

                        validKeys.push(binaryKey); // 如果解密后的文本与明文匹配，则将密钥添加到有效密钥列表中

                    }

                }

            } else {

                // 对于后续的明文密文对，仅考虑之前已经通过验证的密钥

                validKeys = validKeys.filter(key => {

                    let decryptedText = encryptORdecryptMessage(form ,ciphertext[i], key, 1); // 使用已验证的密钥解密密文

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

                    return decryptedText === plaintext[i];  // 只保留解密后文本与明文匹配的密钥

                });

            }

        }

        return validKeys;

    }

    function bruteForceDecrypt() {

        let plaintexts = document.getElementById("plaintexts").value.split("\n");  // 获取明文列表并按行分割

        let ciphertexts = document.getElementById("ciphertexts").value.split("\n");  // 获取密文列表并按行分割

        if (plaintexts.length !== ciphertexts.length) {

            alert("明文和密文的数量不匹配!");

            return;

        }

        let form = document.getElementById("form-select").value;  // 获取form的值

        let validKeys;

        validKeys = filterDecrypt(form, plaintexts, ciphertexts);

        // 显示有效密钥列表

        document.getElementById("validKeysOutput").value = validKeys.join("\n");

    }

9. `refresh` 函数用于清空页面上的输入和输出。

    function refresh() {

        // 清空消息内容输入框

        document.getElementById("message").value = "";

        // 清空密钥输入框

        document.getElementById("key").value = "";

        // 清空输出输入框（如果需要）

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

        document.getElementById("plaintexts").value= "";

        document.getElementById("ciphertexts").value= "";

        document.getElementById("validKeysOutput").value= "";

    }

页面交互：

- 用户可以选择输入的形式是 Binary 还是 ASCII。

- 用户可以输入明文和密钥，然后选择加密或解密。

- 用户可以输入多组明文和密文，然后点击 "BruteForceDecrypt" 进行暴力破解。

使用方法：

1. 打开 HTML 文件。

2. 输入明文和密钥。

3. 选择加密或解密。

4. 点击对应的按钮执行操作。

5. 查看输出结果或有效密钥列表。

注意事项：

- 对于暴力破解，需要提供多组明文和密文，以减少密钥的搜索空间。

这个页面提供了一个简单的 S-DES 加解密工具，方便用户进行实验和学习。需要注意的是，S-DES 是一个较为简化的加密算法，实际应用中应使用更为安全的加密算法。