**开发手册 - 基于S-AES算法的加密解密工具**

**1. 项目概述**

本项目实现了一个基于S-AES（简化版AES）算法的加密解密工具，涵盖基本加密解密、双重和三重加密、多种编码模式（包括ASCII编码）和密码分组链接（CBC）模式。S-AES的轻量化设计使其适用于学习环境和资源受限的应用场景。整个项目分为算法核心代码（AES\_Methods.java）和用户界面代码（SAES\_Mainpage.java）两部分。

**2. 文件结构**

AES\_Methods.java：实现了S-AES算法的核心逻辑，包含基础加解密、多重加密、随机密钥生成和CBC模式的操作方法。

SAES\_Mainpage.java：实现用户交互的图形界面，包含加密解密功能的导航、数据输入和结果显示，并提供按钮事件监听。

**3. 核心代码模块详解**

**3.1 AES\_Methods.java**

AES\_Methods.java文件实现了S-AES的所有核心算法，分为以下几个部分：

**加密方法** - encrypt()

S-AES加密包括三个主要步骤：替换、移位和混淆。

**步骤说明：**

字节替换（SubBytes）：使用S-box（替换盒）替换明文数据中的字节。

行移位（ShiftRows）：改变数据行顺序，增加数据的扩散效果。

列混淆（MixColumns）：将数据列混合，通过数学运算进一步加密。

// 加密函数  
public static int[][] encrypt(int[][] state, int[][]Keys) {  
 int[][][] roundKeys = *keyExpansion*(Keys);  
 // 第一步：明文与密钥w0，w1进行轮密钥加  
 state = *addRoundKey*(state, roundKeys[0]);  
  
 // 第二步：字节替代，行移位，列混淆  
 state = *subNibbles*(state, *SBOX*);  
  
 state = *shiftRows*(state);  
  
 state = *mixColumns*(state, *MIX\_COLUMN\_MATRIX*);  
  
 // 第三步：扩展密钥w2，w3进行轮密钥加  
 state = *addRoundKey*(state, roundKeys[1]);  
  
 // 第四步：字节替代，行移位（无列混淆）  
 state = *subNibbles*(state, *SBOX*);  
  
 state = *shiftRows*(state);  
  
 // 第五步：扩展密钥w4，w5进行最后一轮轮密钥加  
 state = *addRoundKey*(state, roundKeys[2]);  
  
 return state;  
}

**解密方法** - decrypt()

解密过程是加密过程的逆运算。通过逆向的字节替换、行移位和列混淆还原明文。

// 解密函数  
public static int[][] decrypt(int[][] state, int[][] Keys) {  
 int[][][] roundKeys = *keyExpansion*(Keys);  
  
 // 第一步：使用扩展密钥w4，w5进行轮密钥加  
 state = *addRoundKey*(state, roundKeys[2]);  
 // 第二步：逆行移位，逆字节替代  
 state = *shiftRows*(state);  
 state = *subNibbles*(state, *INV\_SBOX*);  
  
 // 第三步：扩展密钥w2，w3进行轮密钥加  
 state = *addRoundKey*(state, roundKeys[1]);  
  
 // 第四步：逆列混淆  
 state = *mixColumns*(state, *INV\_MIX\_COLUMN\_MATRIX*);  
  
 // 第五步：逆行移位，逆字节替代  
 state = *shiftRows*(state);  
 state = *subNibbles*(state, *INV\_SBOX*);  
  
 // 第六步：初始密钥w0，w1进行最后一轮轮密钥加  
 state = *addRoundKey*(state, roundKeys[0]);  
  
 return state;  
}

**多重加密方法**

双重加密 - doubleEncrypt()：通过两个密钥对明文进行两次加密。

// 第一次加密  
int[][] state = *convertToStateMatrix*(input);  
int[][] keyMatrix1 = *convertToStateMatrix*(key1);  
int[][] encrypted1 = *encrypt*(state, keyMatrix1);  
  
// 第二次加密  
int[][] keyMatrix2 = *convertToStateMatrix*(key2);  
int[][] encrypted2 = *encrypt*(encrypted1, keyMatrix2);

三重加密 - tripleEncrypt()：通过三个密钥对明文进行三次加密，提高安全性。

// 第一次加密  
int[][] state = *convertToStateMatrix*(input);  
int[][] keyMatrix1 = *convertToStateMatrix*(key1);  
int[][] encrypted1 = *encrypt*(state, keyMatrix1);  
  
// 第二次加密  
int[][] keyMatrix2 = *convertToStateMatrix*(key2);  
int[][] encrypted2 = *encrypt*(encrypted1, keyMatrix2);  
  
//第三次加密  
int[][] keyMatrix3 = *convertToStateMatrix*(key3);  
int[][] encrypted3 = *encrypt*(encrypted2, keyMatrix3);

**CBC模式加密** - cbcEncrypt()

采用CBC模式（Cipher Block Chaining），适用于较长数据。

初始化向量（IV）：每组加密前将明文与前一密文块异或，从而每个密文块都依赖于前一个块，增强安全性。

示例代码：

// CBC 加密方法  
public static String cbcEncrypt(String plaintextBinary, int[][] key,String iv) {  
 // 1. 生成 IV 并转换为状态矩阵  
 int[][] IV = *convertToStateMatrix*(iv);  
  
 // 2. 将明文二进制字符串转换为状态矩阵块  
 int[][][] plaintextBlocks = *convertBinaryStringToBlocks*(plaintextBinary);  
  
 // 3. 创建密文块数组用于存储每一个加密块  
 int[][][] ciphertextBlocks = new int[plaintextBlocks.length][2][2];  
  
 // 4. 初始化前一个块为 IV  
 int[][] previousBlock = IV;  
  
 // 5. 开始加密每一个明文块  
 for (int i = 0; i < plaintextBlocks.length; i++) {  
 // 5.1. 明文块与前一个块按位异或  
 int[][] currentBlock = *xorStateMatrices*(plaintextBlocks[i], previousBlock);  
  
 // 5.2. 使用 S-AES 进行加密  
 int[][] encryptedBlock = *encrypt*(currentBlock, key);  
  
 // 5.3. 存储加密后的密文块  
 ciphertextBlocks[i] = encryptedBlock;  
  
 // 5.4. 更新前一个块为当前密文块  
 previousBlock = encryptedBlock;  
 }  
  
 // 6. 将每个密文块转换为二进制字符串并拼接  
 StringBuilder ciphertextBinary = new StringBuilder();  
 for (int i = 0; i < ciphertextBlocks.length; i++) {  
 ciphertextBinary.append(*convertStateMatrixToString*(ciphertextBlocks[i]));  
 }  
  
 // 7. 返回最终的密文二进制字符串  
 return ciphertextBinary.toString();  
}

随机密钥生成 - generateRandomKey()

生成随机16位密钥，用于不同模式的加解密。

示例代码：

public static int generateRandomKey() {  
 return (int)(Math.*random*() \* 65536); // 生成16位随机数  
}

**3.2 SAES\_Mainpage.java**

SAES\_Mainpage.java实现了用户交互的界面，包含按钮、文本框和事件监听器。该界面结构如下：

**主界面设计**

界面包含用于明文和密钥输入的JTextField，以及显示结果的resultField。

三个主要按钮：加密、解密和生成随机密钥。

加密和解密功能

encryptButton和decryptButton的事件监听器会获取用户输入的数据，调用AES\_Methods中的加密或解密方法并显示结果。

代码片段：

public SAES\_en() {  
 // 设置窗口标题  
 setTitle("S-AES 加解密算法");  
 setSize(400, 250);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
  
 // 设置布局  
 setLayout(new GridLayout(6, 2));  
  
 // 创建并添加组件  
 add(new JLabel("密钥 (16位二进制):"));  
 keyField = new JTextField(16);  
 add(keyField);  
  
 add(new JLabel("输入 (16位二进制):"));  
 inputField = new JTextField(16);  
 add(inputField);  
  
 add(new JLabel("输出结果 (二进制):"));  
 outputField = new JTextField();  
 outputField.setEditable(false);  
 add(outputField);  
  
 encryptButton = new JButton("加密");  
 decryptButton = new JButton("解密");  
 generateKeyButton = new JButton("随机生成密钥");  
 backButton = new JButton("返回");  
  
 add(encryptButton);  
 add(decryptButton);  
 add(generateKeyButton);  
 add(backButton);

**随机密钥生成**

generateKeyButton调用AES\_Methods.generateRandomKey()，生成并显示随机密钥。

代码片段：

generateKeyButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 int randomKey = AES\_Methods.generateRandomKey();  
 keyField.setText(Integer.*toBinaryString*(randomKey));  
 }  
});

**启动界面** - main()

调用主界面，确保界面初始化在事件派发线程中完成。

代码片段：

public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 new S\_AES.SAES\_Mainpage().setVisible(true);  
 }  
 });  
 }

**4. 代码扩展和自定义**

增加加密模式：可在AES\_Methods.java中新增加密模式（如CTR、OFB模式），并在SAES\_Mainpage.java中添加相应的按钮和逻辑。

日志记录：可以为加解密操作增加日志记录，记录输入、输出和执行时间等信息，帮助调试和跟踪。

多线程支持：若需要处理较大数据量，可以在加密和解密过程中使用多线程提升性能。

**5. 测试方案**

单元测试：为每个方法编写单元测试，确保加解密逻辑正确无误。可以使用JUnit测试框架自动化测试。

GUI测试：手动运行SAES\_Mainpage.java，验证界面按钮的正常功能。

跨平台测试：在不同操作系统上运行该应用程序，确保界面一致性和算法兼容性。

**6. 常见问题与解决方案**

错误输入：用户输入非二进制数据时可能导致加解密失败，建议在界面上增加输入验证，确保只接受二进制字符串。

密钥长度不匹配：加解密过程中，密钥应与算法所需长度一致，用户界面可提供提示或限制。

加密结果异常：如遇到结果不正确的情况，可逐步检查每个加解密步骤，确保算法操作符合S-AES标准。

**7. 维护与更新**

算法优化：可在AES\_Methods中进一步优化算法的执行效率，减少不必要的运算步骤。

代码重构：随着项目扩展和功能增加，定期对代码进行重构，确保代码清晰、简洁