S-AES 加密解密工具开发手册

1. 概述

本项目实现了一个基于简化高级加密标准 (S-AES) 的图形化工具，允许用户输入明文和密钥，进行加密和解密操作。该工具支持 ASCII 和 16 位二进制输入模式，并通过 Swing 界面呈现。

2. 项目结构

- 包名: `aestest`
- 主类: `SAES`，实现 S-AES 加密解密逻辑和图形化界面
- 依赖库: Java 标准库 (javax.swing, java.awt)

3. 核心功能

3.1 输入模式

工具支持两种输入模式：

- **ASCII 模式**: 用户可以输入任意 ASCII 文本，程序将其转换为二进制进行加密。
- **16 位二进制模式**: 用户直接输入 16 位二进制数串，进行加密或解密。

3.2 加密和解密

- **加密**: S-AES 加密过程基于输入明文和 16 位二进制密钥，通过一系列操作如密钥扩展、S盒替换、行移位、列混淆以及轮密钥加等步骤，生成密文。
- **解密**: 解密过程是加密的逆过程，使用相同的密钥并逆向执行加密中的各个步骤，最终恢复明文。

4. 开发环境

4.1 开发工具

- **Java JDK 8** 或以上版本: 本项目使用 Java 语言开发，建议使用 JDK 8 或更新版本进行编译和运行。
- **集成开发环境 (IDE)**: 推荐使用 IntelliJ IDEA、Eclipse 或 NetBeans。

4.2 编译与运行

编译：
javac aestest/SAES.java
运行：
java aestest.SAES

5. 代码结构详解

5.1 密钥扩展

- **keyExpansion(int key)**: 使用原始密钥生成一系列子密钥用于各加密轮。

```
private static int[] keyExpansion(int key) {
    int[] w = new int[6];
    w[0] = (key >> 8) & 0xFF;  // High 8 bits of the key
    w[1] = key & 0xFF;         // Low 8 bits of the key

    for (int i = 2; i < 6; i++) {
       if (i % 2 == 0) {
          // Process using function g on w[i-1] (right part of the previous key)
          int g = functionG(w[i-1], (i / 2) - 1);
          w[i] = w[i - 2] ^ g;
       } else {
          w[i] = w[i - 1] ^ w[i - 2];
       }
    }
    return w;
 }
```

- **functionG(int byteVal, int rconIndex)**: 用于密钥扩展中的非线性变换，包括字节循环左移、S盒替换和轮常数异或。

```
private static int functionG(int byteVal, int rconIndex) {
    // Left rotate the byte
    int rotated = leftRotate(byteVal);
    // Apply S-box substitution on both nibbles
    int substituted = (SBOX[rotated >> 4] << 4) | SBOX[rotated & 0x0F];
    // XOR with the round constant
    return substituted ^ RCON[rconIndex];
 }
```

5.2 加密解密逻辑

- **encryptBinary(String plaintext, String key)**: 执行二进制明文的加密流程。

```
// 二进制字符串加密

    private String encryptBinary(String plaintext, String key) {

        int pt = Integer.parseInt(plaintext, 2);

        System.out.println("Initial plaintext: " + Integer.toBinaryString(pt));

        int k = Integer.parseInt(key, 2);



        int[] w = keyExpansion(k);

        printRoundKeys(w);  // 调试密钥

        //第一次轮密钥加

        int state = addRoundKey(pt, w[0], w[1]);

        String output = String.format("%16s", Integer.toBinaryString(state &
0xFFFF)).replace(' ', '0');

        System.out.println(output);



        state = nibbleSub(state);

        System.out.println("After nibbleSub: " + Integer.toBinaryString(state));

        state = shiftRows(state);

        System.out.println("After shiftRows: " + Integer.toBinaryString(state));

        state = mixColumns(state);

        String output2 = String.format("%16s", Integer.toBinaryString(state &
0xFFFF)).replace(' ', '0');

        System.out.println(output2);





        state = addRoundKey(state, w[2], w[3]);

        System.out.println("After second addRoundKey: " +
Integer.toBinaryString(state));

        state = nibbleSub(state);

        System.out.println("After second nibbleSub: " + Integer.toBinaryString(state));

        state = shiftRows(state);
```

```
        System.out.println("After second shiftRows: " + Integer.toBinaryString(state));

        int ciphertext = addRoundKey(state, w[4], w[5]);
        System.out.println("Final ciphertext: " + Integer.toBinaryString(ciphertext));

        return String.format("%16s", Integer.toBinaryString(ciphertext)).replace(' ', '0');
    }
```

- **decryptBinary(String ciphertext, String key)**: 执行二进制密文的解密流程。

```
// 二进制字符串解密
    private String decryptBinary(String ciphertext, String key) {
        int ct = Integer.parseInt(ciphertext, 2);
        int k = Integer.parseInt(key, 2);

        int[] w = keyExpansion(k);
        printRoundKeys(w);  // 调试密钥

        int state = addRoundKey(ct, w[4], w[5]);
        System.out.println("After addRoundKey: " + Integer.toBinaryString(state));
        state = invShiftRows(state);
        System.out.println("After invShiftRows: " + Integer.toBinaryString(state));
        state = invNibbleSub(state);
        System.out.println("After invNibbleSub: " + Integer.toBinaryString(state));
        state = addRoundKey(state, w[2], w[3]);

        state = invMixColumns(state);
        state = invShiftRows(state);
        state = invNibbleSub(state);
        int plaintext = addRoundKey(state, w[0], w[1]);
```

```
    return String.format("%16s", Integer.toBinaryString(plaintext)).replace(' ', '0');
}
```

- **encryptASCII(String plaintext, String key)**: 将 ASCII 文本转换为二进制后加密

```
// ASCII 字符串加密
private String encryptASCII(String plaintext, String key) {
    StringBuilder encryptedResult = new StringBuilder();
    byte[] bytes = plaintext.getBytes(StandardCharsets.US_ASCII);

    // 按 2 字节为一组处理
    for (int i = 0; i < bytes.length; i += 2) {
        // 获取当前 2 字节，并将其组成 16-bit 二进制块
        int block = (bytes[i] << 8) | (i + 1 < bytes.length ? bytes[i + 1] : 0);

        // 将该块转换为二进制字符串，并加密
        String encryptedBinary = encryptBinary(Integer.toBinaryString(block), key);

        // 将加密后的二进制块转换为整数
        int encryptedBlock = Integer.parseInt(encryptedBinary, 2);

        // 将整数形式的加密块转换为 4 个 ASCII 字符并添加到结果
        encryptedResult.append((char) ((encryptedBlock >> 8) & 0xFF)); // 高位字符
        encryptedResult.append((char) (encryptedBlock & 0xFF));       // 低位字符
    }

    return encryptedResult.toString();
```

```
}
```

5.3 图形用户界面

- 使用 **Swing** 构建的用户界面允许用户输入数据和密钥，显示加密或解密的结果。

6. 测试

- **testEncryptionDecryption()**: 验证加密后的数据是否可以被正确解密。
- **testSBoxAndInvSBox()**: 测试 S 盒和逆 S 盒是否正确实现。

7. 错误处理

- 输入验证: 确保用户输入的密钥和数据符合要求（例如长度和字符类型）。
- 异常处理: 捕捉并处理可能出现的运行时错误，如非法参数或运行时异常。

8. 未来改进方向

- 增加更多的密钥验证和错误提示。
- 提供更多的输入和加密选项。
- 优化性能和用户界面。

9. 常见问题

- **如何输入有效的密钥？** 密钥必须是 16 位的二进制数。如果格式不正确，程序将显示错误信息。
- **ASCII 模式下的特殊处理？** 在 ASCII 模式下，输入文本将被转换为二进制后加密。