测试结果

第 1 关：基本测试

根据 S-AES 算法编写和调试程序，提供 GUI 解密支持用户交互。输入可以是 16bit 的数据和 16bit 的密钥，输出是 16bit 的密文。测试：

加密操作

1、输入需要加密的 16bit 明文和 16 位二进制密钥。

2、点击“加密”按钮，结果将显示在“输出”框中：

3、加密结果为二进制数字。

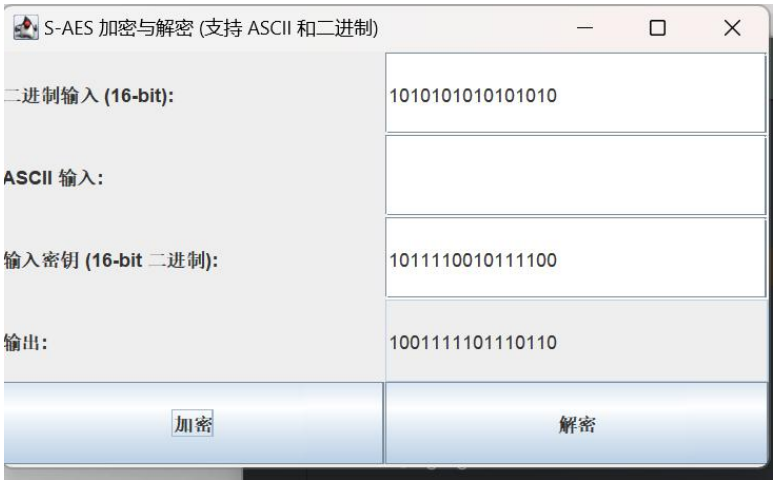

图 1　16 位二进制模式下的加密模式

解密操作

1、输入需要加密的 16bit 密文和 16 位二进制密钥。

2、点击“解密”按钮，结果将显示在“输出”框中：

3、解密结果为二进制数字。

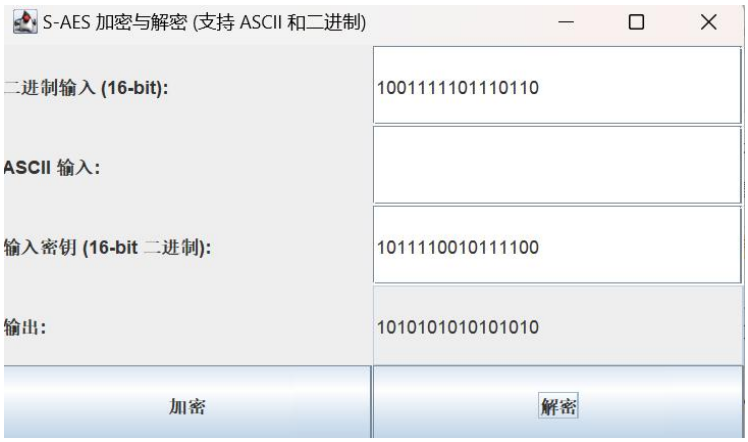

图 2　16 位二进制模式下的解密模式

加密与解密的明密文对相同。

第 2 关：交叉测试

考虑到是"算法标准"，所有人在编写程序的时候需要使用相同算法流程和转换单元(替换盒、列混淆矩阵等)，以保证算法和程序在异构的系统或平台上都可以正常运行。

设有 A 和 B 两组位同学(选择相同的密钥 K)；则 A、B 组同学编写的程序对明文 P 进行加密得到相同的密文 C；或者 B 组同学接收到 A 组程序加密的密文 C，使用 B 组程序进行解密可得到与 A 相同的 P。

测试：

经过与高俪洪组的交叉测试，能够得到相同的明文和密文

第 3 关：扩展功能

考虑到向实用性扩展，加密算法的数据输入可以是 ASII 编码字符串(分组为 2 Bytes)，对应地输出也可以是 ACII 字符串(很可能是乱码)。

测试：

1、选择输入模式为 ASCII 模式。

2、输入要加密的二进制明文。

3、输入 16 位二进制密钥。

4、点击“加密”按钮，结果将以 ASCII 字符显示。

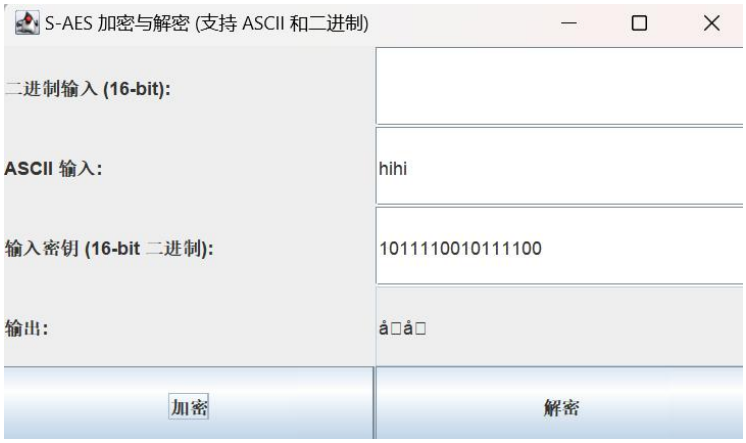

图 3 ASCII 模式下的加密模式

3.4 第 4 关：多重加密

3.4.1 双重加密

将 S-AES 算法通过双重加密进行扩展，分组长度仍然是 16 bits，但密钥长度为 32 bits。

这一关中我的明文设置一直是 1010101010101010，以便我检验解密后是否正确。

通过使用两个 16 位的密钥进行两次加密来实现。

```
public static String doubleEncrypt(String plaintext, String key1, String key2) {
    String firstEncryption = encryptBinary(plaintext, key1);
    return encryptBinary(firstEncryption, key2);
}

// 双重解密
1 usage
public static String doubleDecrypt(String ciphertext, String key1, String key2) {
    String firstDecryption = decryptBinary(ciphertext, key2);
    return decryptBinary(firstDecryption, key1);
}
```

结果：

```
密文: 1010100000110111
解密文本: 1010101010101010
测试通过: 加密和解密一致。
双重加密: 1110110100000100
双重解密: 1010101010101010
```

3.4.2 中间相遇攻击

假设你找到了使用相同密钥的明、密文对(一个或多个)，请尝试使用中间相遇攻击的方法找到正确的密钥 Key(K1+K2)。

解答：

一个基本思路是：

1. 从已知明文开始加密，记录每一步的中间结果。
2. 反向进行解密，看看是否能找到相同的中间结果，以此推导出可能的密钥。

代码过长，不放在这了

```
Meet-in-the-Middle Attack Success!
K1: 1110011101111101
K2: 0000000000000000
Found Keys: K1 = 1110011101111101, K2 = 0000000000000000
```

再将这两个密码对明文进行加密进行验证

```
双重加密: 1110110100000100
双重解密: 1010101010101010
```

发现确实是这一对明密文的密码

与此同时还发现这一明密文不止一对密钥，因为我最初加密使用的是

K1=1010101010101010；K2=1111111111111111

所以双重加密一组明密文之间不止一组密码

3.4.3 三重加密

将 S-AES 算法通过三重加密进行扩展，下面两种模式选择一种完成：

(1)按照 32 bits 密钥 Key(K1+K2)的模式进行三重加密解密，

```java
// 三重加密（32-bit模式）
1 usage
public static String tripleEncrypt32(String plaintext, String key1, String key2) {
    String firstEncrypt = encryptBinary(plaintext, key1);
    String secondDecrypt = decryptBinary(firstEncrypt, key2);
    return encryptBinary(secondDecrypt, key1);
}

// 三重解密（32-bit模式）
1 usage
public static String tripleDecrypt32(String ciphertext, String key1, String key2) {
    String firstDecrypt = decryptBinary(ciphertext, key1);
    String secondEncrypt = encryptBinary(firstDecrypt, key2);
    return decryptBinary(secondEncrypt, key1);
}
```

结果：

```
三重加密（32位模式）: 1011111001100111
三重解密（32位模式）: 1010101010101010
```

可以发现解密后的

(2)使用 48bits(K1+K2+K3)的模式进行三重加解密。

```java
// 三重加密（48-bit模式）
1 usage
public static String tripleEncrypt48(String plaintext, String key1, String key2, String key3) {
    String firstEncrypt = encryptBinary(plaintext, key1);
    String secondDecrypt = decryptBinary(firstEncrypt, key2);
    return encryptBinary(secondDecrypt, key3);
}

// 三重解密（48-bit模式）
1 usage
public static String tripleDecrypt48(String ciphertext, String key1, String key2, String key3) {
    String firstDecrypt = decryptBinary(ciphertext, key3);
    String secondEncrypt = encryptBinary(firstDecrypt, key2);
    return decryptBinary(secondEncrypt, key1);
}
```

结果：

```
三重加密（48位模式）: 1100011010110011
三重解密（48位模式）: 1010101010101010
```

最终发现两种加密方式通过解密都得到了正确的明文

3.5 第 5 关：工作模式

基于 S-AES 算法，使用密码分组链(CBC)模式对较长的明文消息进行加密。注意初始向量(16 bits) 的生成，并需要加解密双方共享。

在 CBC 模式下进行加密，并尝试对密文分组进行替换或修改，然后进行解密，请对比篡改密文前后的解密结果。

1.加密时，先将明文与初始向量进行 XOR 运算，然后再加密。

```
// CBC模式加密
1 usage
public static String cbcEncrypt(String plaintext, String key, String iv) {
    String xorWithIv = xor(plaintext, iv);
    return encryptBinary(xorWithIv, key);
}
```

2. 解密时，先解密得到的密文，然后与初始向量进行 XOR 运算。

```
// CBC模式解密
2 usages
public static String cbcDecrypt(String ciphertext, String key, String iv) {
    String decryptedBlock = decryptBinary(ciphertext, key);
    return xor(decryptedBlock, iv);
}
```

篡改密文测试

对密文进行替换或修改，然后进行解密以比较篡改前后的解密结果。

```
CBC加密: 0101101110000010
CBC解密: 1010101010101010
篡改后的密文: 0101101110001010
篡改解密: 1010101000011001
```

发现篡改后的密文在解密就跟初始明文有了很大的差别。