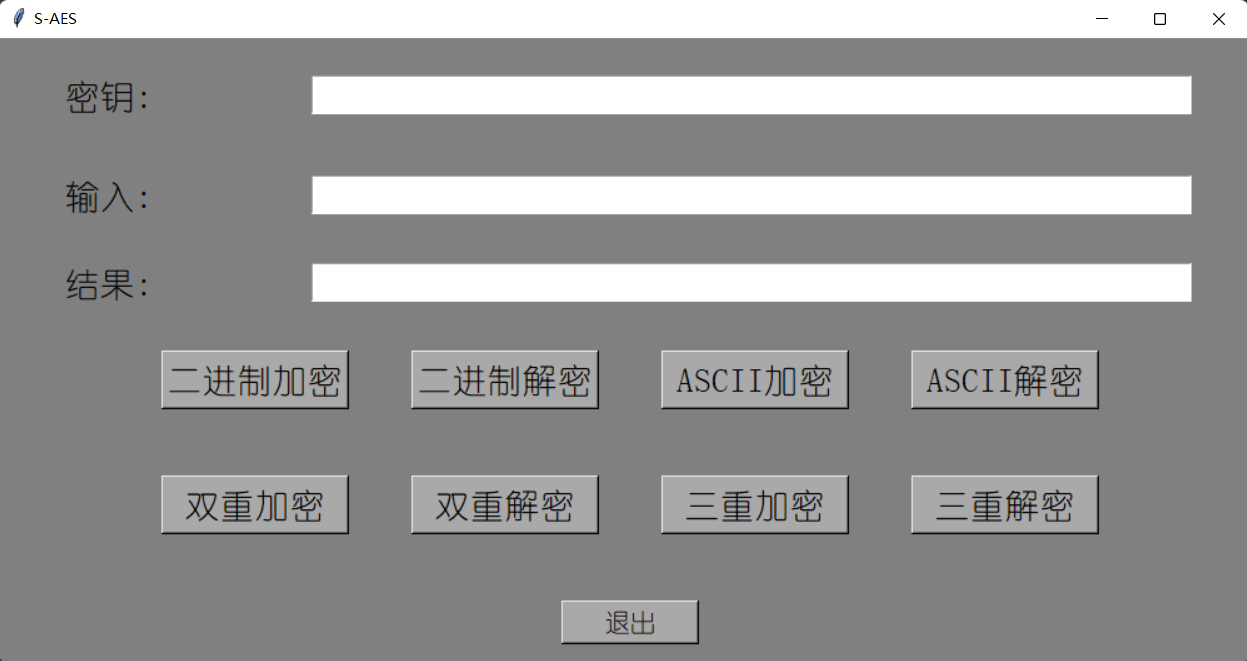
**1.引言**  
S-AES(简化AES)算法是一个面向教育的加密算法。它与AES的性质和结构类似，但其只使用了16位的明文和16位的密钥进行加密。

**2. 软件概述**  
**2.1目标**

基于S-AES算法编程实现加、解密程序。  
**2.2功能**

二进制加密、二进制解密、ASCII加密、ASCII解密、双重加密、双重解密、三重加密、三重解密

**2.3 UI设计**

  
  
**3. 开发工具**

Python、pycharm  
  
**4. 算法实现**

①轮密钥加：16bits的状态矩阵和16bits的轮密钥按位进行XOR运算。

def change\_LandR(w):  
 w\_left=w[0:4]  
 w\_right=w[4:8]  
 temp=w\_left  
 w\_left=w\_right  
 w\_right=temp  
 ans=w\_left+w\_right  
 return ans  
  
def xor(a,b):  
 c=""  
 for i in range(len(a)):  
 temp1=int(a[i])  
 temp2=int(b[i])  
 c+=str(temp1^temp2)  
 return c

②半字节代替：状态矩阵中每个半字节的左边两位确定行值，右边两位确定列值，然后查表进行代替，加密的时候查S盒，解密的时候查逆S盒。

def S\_change(s,w):  
 w\_1=w[0:2]  
 w\_2=w[2:4]  
 w\_3=w[4:6]  
 w\_4=w[6:8]  
 w1=two\_to\_ten(w\_1)  
 w2=two\_to\_ten(w\_2)  
 w3=two\_to\_ten(w\_3)  
 w4=two\_to\_ten(w\_4)  
 w\_s=s[w1][w2]+s[w3][w4]  
 return w\_s  
  
def g1(w1):  
 R1="10000000"  
 w1\_1=change\_LandR(w1)  
 w1\_1\_s=S\_change(s\_0,w1\_1)  
 w1\_1\_s\_r=xor(w1\_1\_s,R1)  
 return w1\_1\_s\_r  
def g2(w1):  
 R2="00110000"  
 w1\_1=change\_LandR(w1)  
 w1\_1\_s=S\_change(s\_0,w1\_1)  
 w1\_1\_s\_r=xor(w1\_1\_s,R2)  
 return w1\_1\_s\_r

def ban\_change(s,ming):  
 ming0=ming[0:4]  
 ming1=ming[4:8]  
 ming2=ming[8:12]  
 ming3=ming[12:16]  
 ming0\_1=(S\_change(s,ming0+ming1))[0:4]  
 ming1\_1=(S\_change(s,ming0+ming1))[4:8]  
 ming2\_1=(S\_change(s,ming2+ming3))[0:4]  
 ming3\_1=(S\_change(s,ming2+ming3))[4:8]   
 ming\_new=ming0\_1+ming1\_1+ming2\_1+ming3\_1  
 return ming\_new

③行移位：对状态矩阵的第一行保持不变，第二行循环左移半个字节。

def hang\_change(ming):  
 ming0=ming[0:4]  
 ming1=ming[4:8]  
 ming2=ming[8:12]  
 ming3=ming[12:16]  
 temp=ming1  
 ming1=ming3  
 ming3=temp  
 ming\_new=ming0+ming1+ming2+ming3  
 return ming\_new

④列混淆：列混淆函数在各列上执行

def lie\_change(ming):  
 a1 = int(ming[0]) ^ int(ming[6])  
 a2 = int(ming[1]) ^ int(ming[4]) ^ int(ming[7])  
 a3 = int(ming[2]) ^ int(ming[4]) ^ int(ming[5])  
 a4 = int(ming[3]) ^ int(ming[5])  
 s00 = str(a1) + str(a2) + str(a3) + str(a4)

b1 = int(ming[2]) ^ int(ming[4])  
 b2 = int(ming[0]) ^ int(ming[3]) ^ int(ming[5])  
 b3 = int(ming[0]) ^ int(ming[1]) ^ int(ming[6])  
 b4 = int(ming[1]) ^ int(ming[7])  
 s10 = str(b1) + str(b2) + str(b3) + str(b4)  
  
 c1 = int(ming[8]) ^ int(ming[14])  
 c2 = int(ming[9]) ^ int(ming[12]) ^ int(ming[15])  
 c3 = int(ming[10]) ^ int(ming[12]) ^ int(ming[13])  
 c4 = int(ming[11]) ^ int(ming[13])  
 s01 = str(c1) + str(c2) + str(c3) + str(c4)

d1 = int(ming[10]) ^ int(ming[12])  
 d2 = int(ming[8]) ^ int(ming[11]) ^ int(ming[13])  
 d3 = int(ming[8]) ^ int(ming[9]) ^ int(ming[14])  
 d4 = int(ming[9]) ^ int(ming[15])  
 s11 = str(d1) + str(d2) + str(d3) + str(d4)

ming\_new = s00 + s10 + s01 + s11

return ming\_new

**5.拓展功能**

**①ASCII加解密**

def convert\_to\_8bit(string):  
 result = ""  
 for char in string:  
 ascii\_code = bin(ord(char))[2:].zfill(8)  
 result += ascii\_code  
 if len(string)%2!=0:  
 print("字符数量不能两两分组")  
 result="00000000"+result  
 return result  
def fenzu(string):  
 num\_of\_group=len(string)/16  
 group=[0 for i in range(int(num\_of\_group))]  
 for i in range(int(num\_of\_group)):  
 group[i]=string[i\*16:(i+1)\*16]  
 print(group[i])  
 return group  
def bit\_to\_convert(fenzu):  
 result = ""  
 for i in fenzu:  
 i\_1=i[0:8]  
 i\_2=i[8:16]  
 ascii\_code\_1 = chr(two\_to\_ten(i\_1))  
 ascii\_code\_2 = chr(two\_to\_ten(i\_2))  
 ascii\_code=ascii\_code\_1+ascii\_code\_2  
 result += ascii\_code  
 return result

**②双重加解密**

def two\_encropt(mingwen,key):  
 key1=key[0:16]  
 key2=key[16:32]  
 first=encropt(mingwen,key1)  
 second=encropt(first,key2)  
 return second  
def two\_decropt(miwen,key):  
 key1=key[0:16]  
 key2=key[16:32]  
 first=decropt(miwen,key2)  
 second=decropt(first,key1)  
 return second

**③三重加解密**

def three\_encropt(mingwen,key):  
 key1=key[0:16]  
 key2=key[16:32]  
 key3=key[32:48]  
 first=encropt(mingwen,key1)  
 second=decropt(first,key2)  
 third=encropt(second,key3)  
 return third  
def three\_decropt(miwen,key):  
 key1=key[0:16]  
 key2=key[16:32]  
 key3=key[32:48]  
 first=decropt(miwen,key3)  
 second=encropt(first,key2)  
 third=decropt(second,key1)  
 return third

**④中间相遇攻击**

def center\_attack(mingwen, miwen):  
 maybe\_key = []  
 num = len(mingwen)  
 center\_value1 = []  
 center\_value2 = []  
 for i in range(2\*\*16):  
 key = bin(i)[2:].zfill(16)  
 str1=""  
 str2=""  
 for j in range(num):  
 temp=encropt(mingwen[j],key)  
 str1+=temp  
 center\_value1.append(str1)  
 for k in range(num):  
 temp=decropt(miwen[k],key)  
 str2+=temp  
 center\_value2.append(str2)  
 for k in range(2 \*\* 16):  
 for h in range(2 \*\* 16):  
 if center\_value1[k] == center\_value2[h]:  
 maybe\_key.append((k, h))  
 for m in maybe\_key:  
 key1=bin(m[0])[2:].zfill(16)  
 key2 = bin(m[1])[2:].zfill(16)  
 key\_real=str(key1)+str(key2)  
 print(key\_real)  
 return key\_real

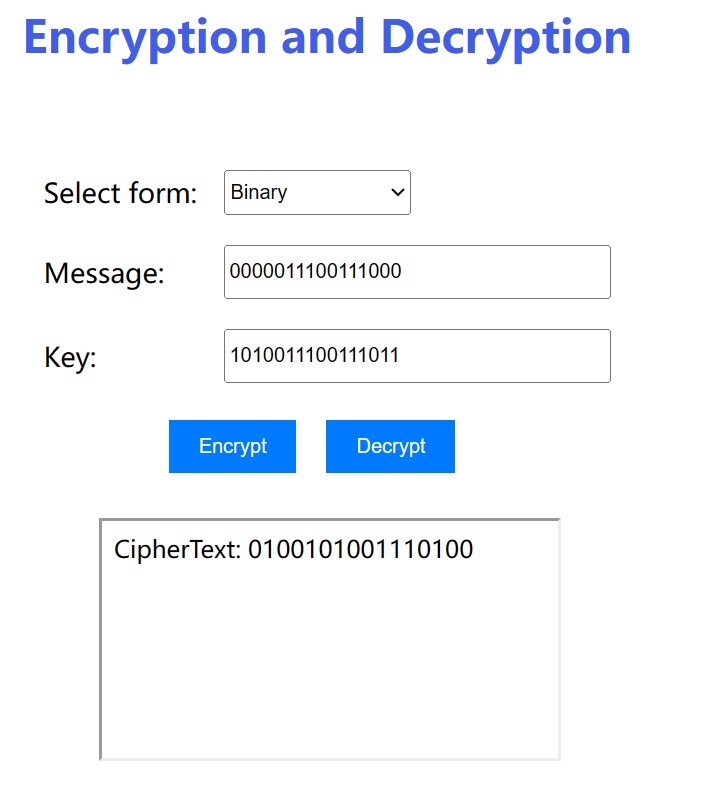
**6.CBC模式**

def CBC\_encropt(mingwen, key, IV):  
 temp\_Miwen = ["0" for i in range(len(mingwen))]  
 for i in range(len(mingwen)):  
 if i == 0:  
 temp\_Miwen [i] = (S\_AES\_16bits.encropt(S\_AES\_16bits.xor(mingwen[i], IV), key))  
 else:  
 temp\_Miwen [i] = (S\_AES\_16bits.encropt(S\_AES\_16bits.xor(mingwen[i], temp\_Miwen [i - 1]), key))  
 return temp\_Miwen  
def CBC\_decropt(miwen,key,IV):  
 temp\_decropt=[]  
 temp\_mingwen=[]  
 ans=[]  
 for i in miwen:  
 temp\_decropt.append(S\_AES\_16bits.decropt(i,key))  
 for i in range(len(miwen)-1,-1,-1):  
 if i !=0:  
 temp\_mingwen.append(S\_AES\_16bits.xor(temp\_decropt[i],miwen[i-1]))  
 if i==0:  
 temp\_mingwen.append(S\_AES\_16bits.xor(temp\_decropt[i],IV))  
 for i in reversed(temp\_mingwen):  
 ans.append(i)  
 return ans

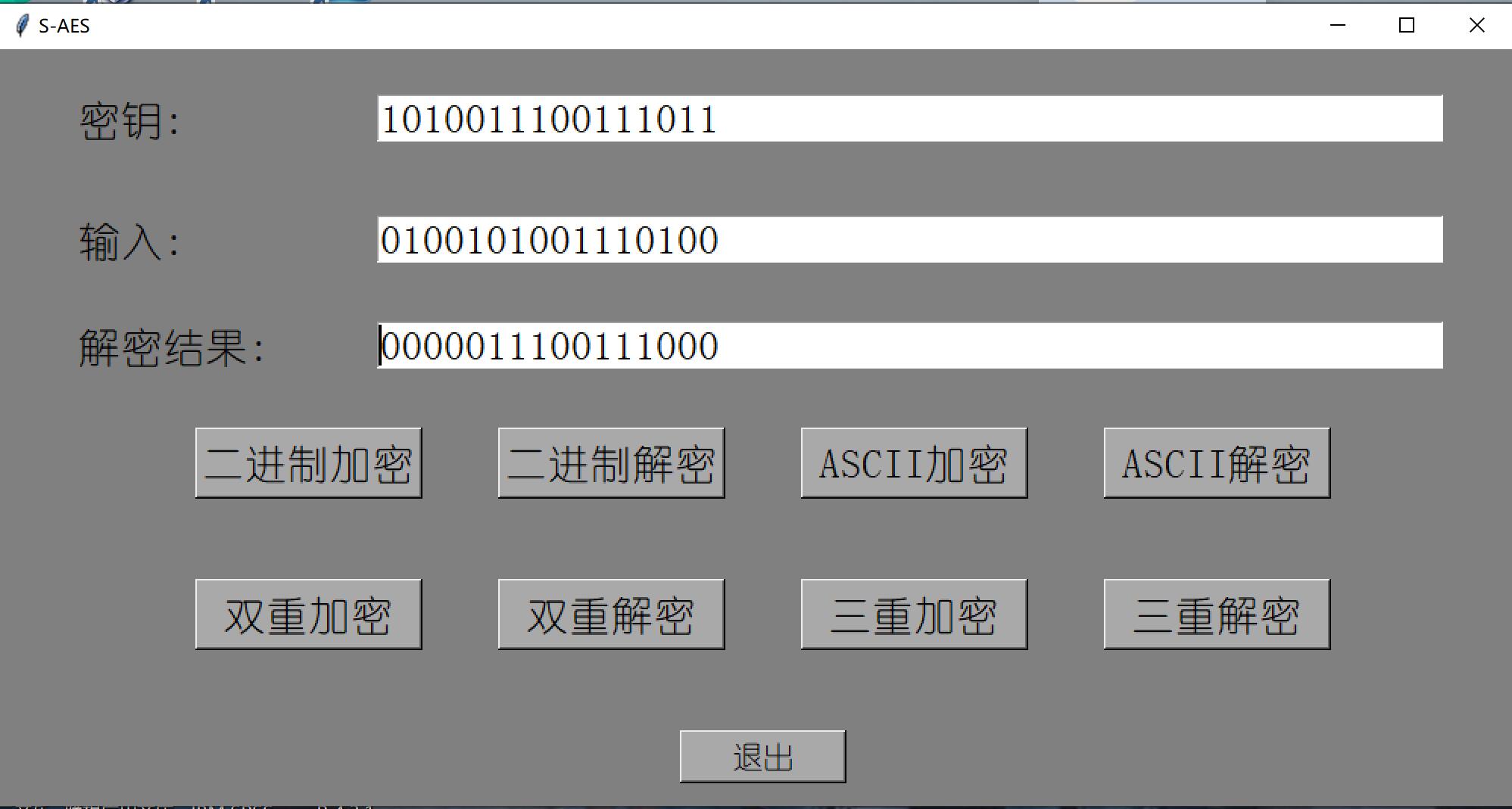
**7.交叉测试结果**

二进制加解密

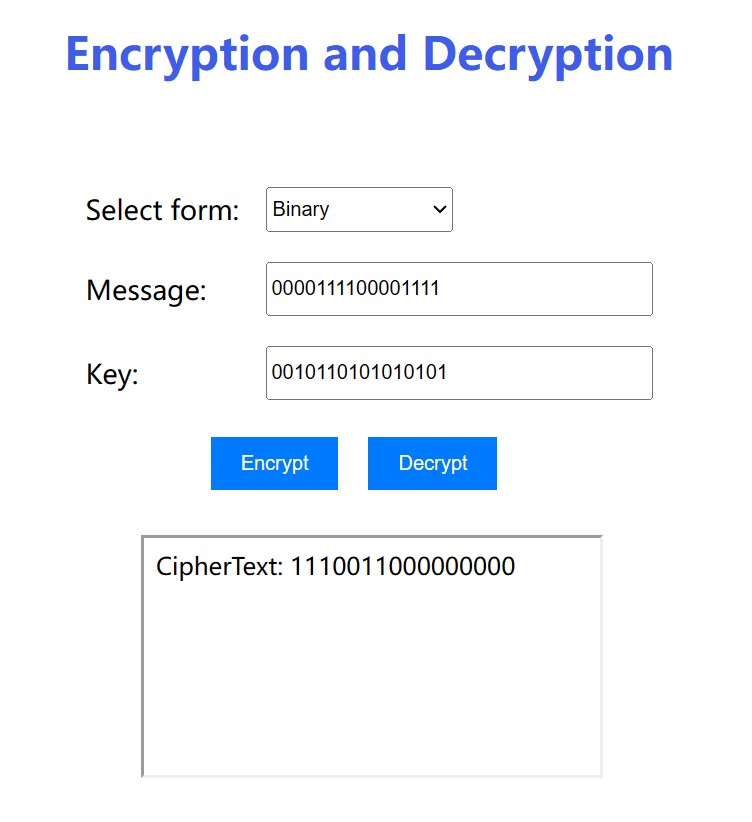
加密：



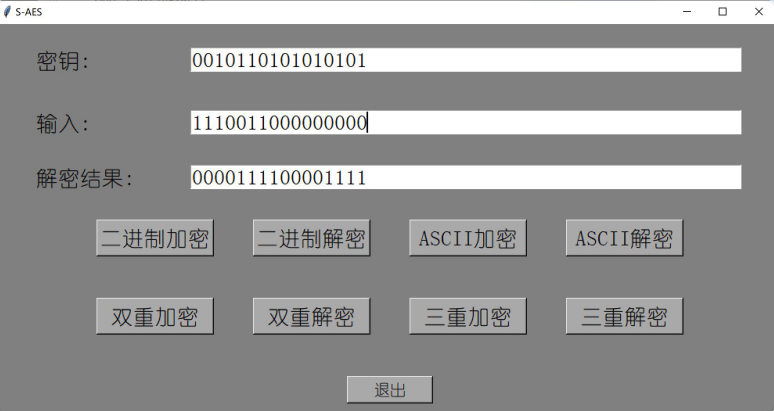
解密：



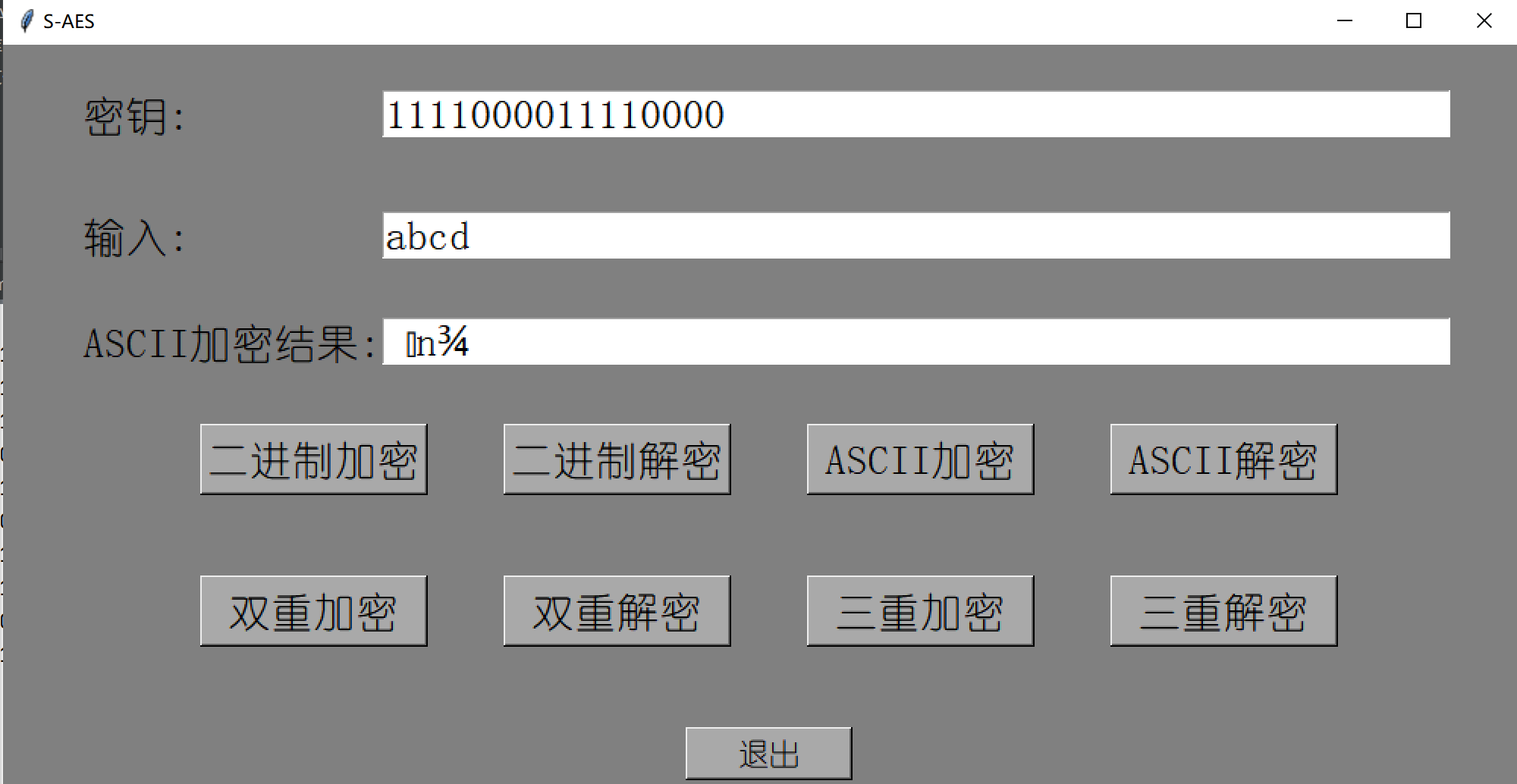
加密



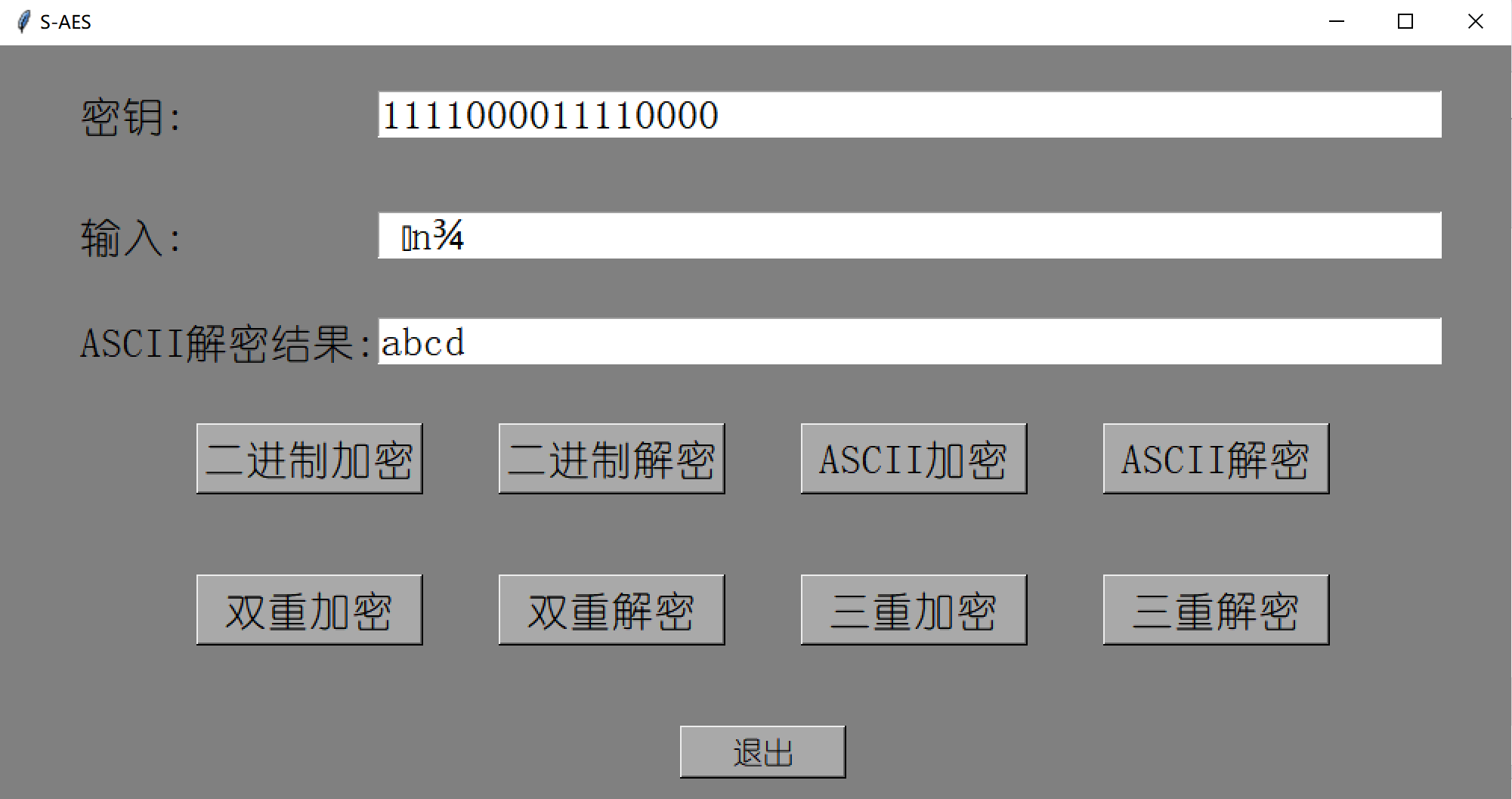
解密



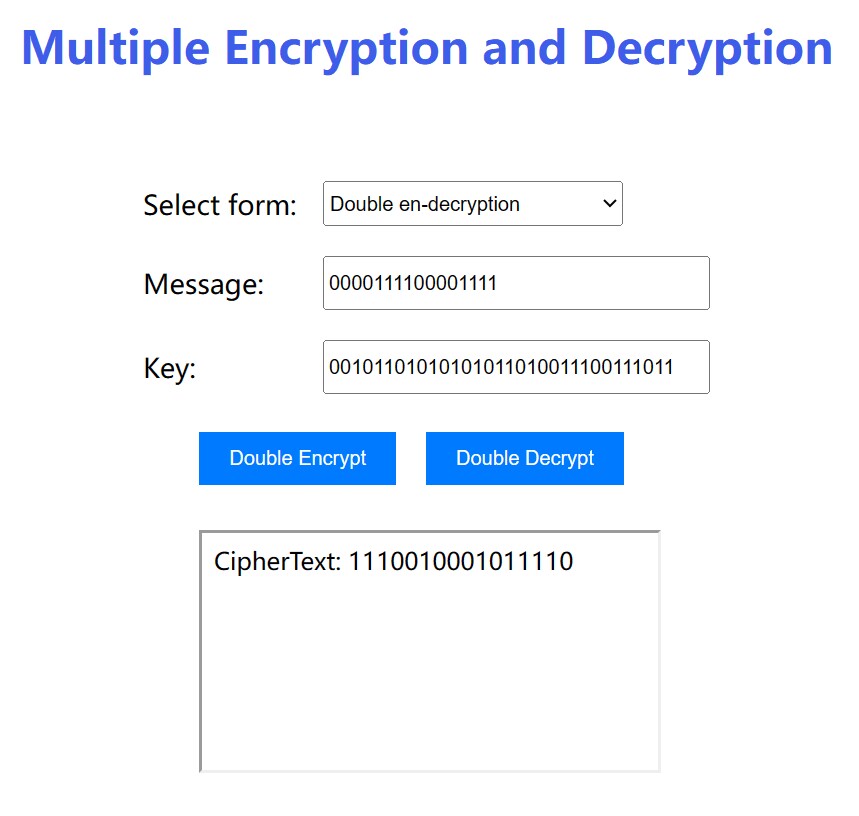
ASCII加密



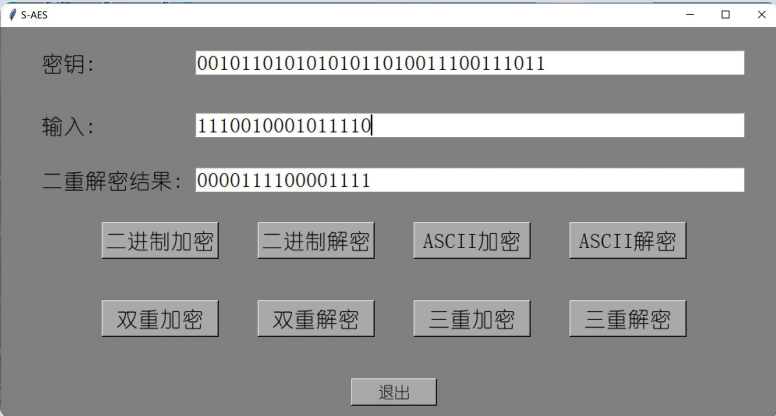
ASCII解密



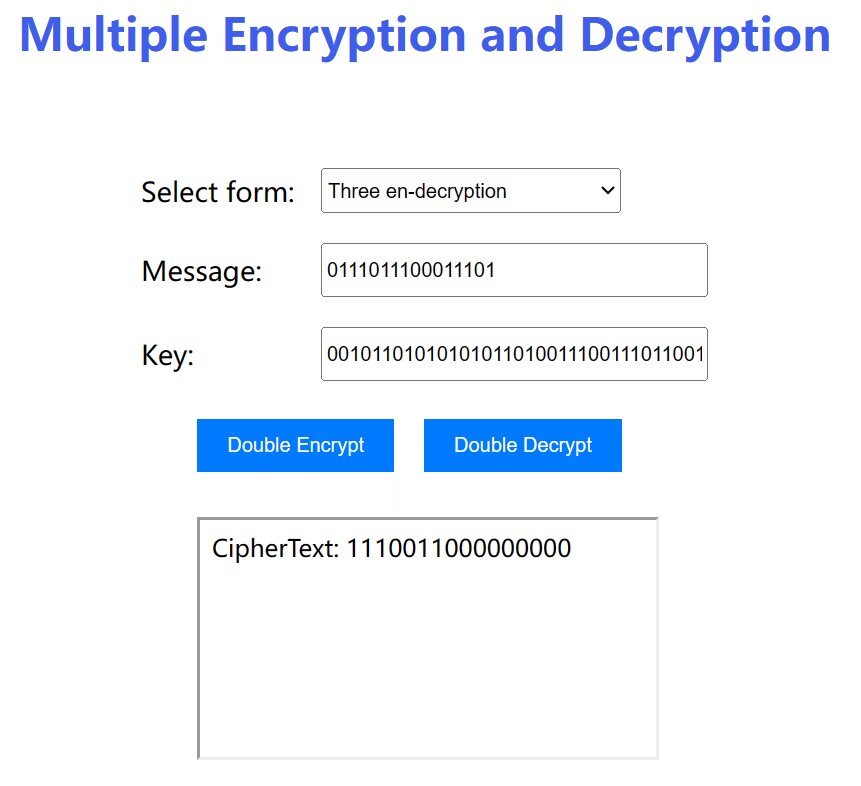
二重加密



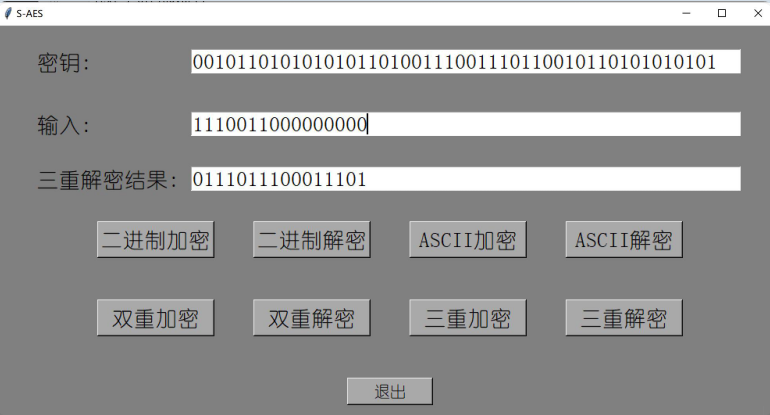
二重解密



三重加密



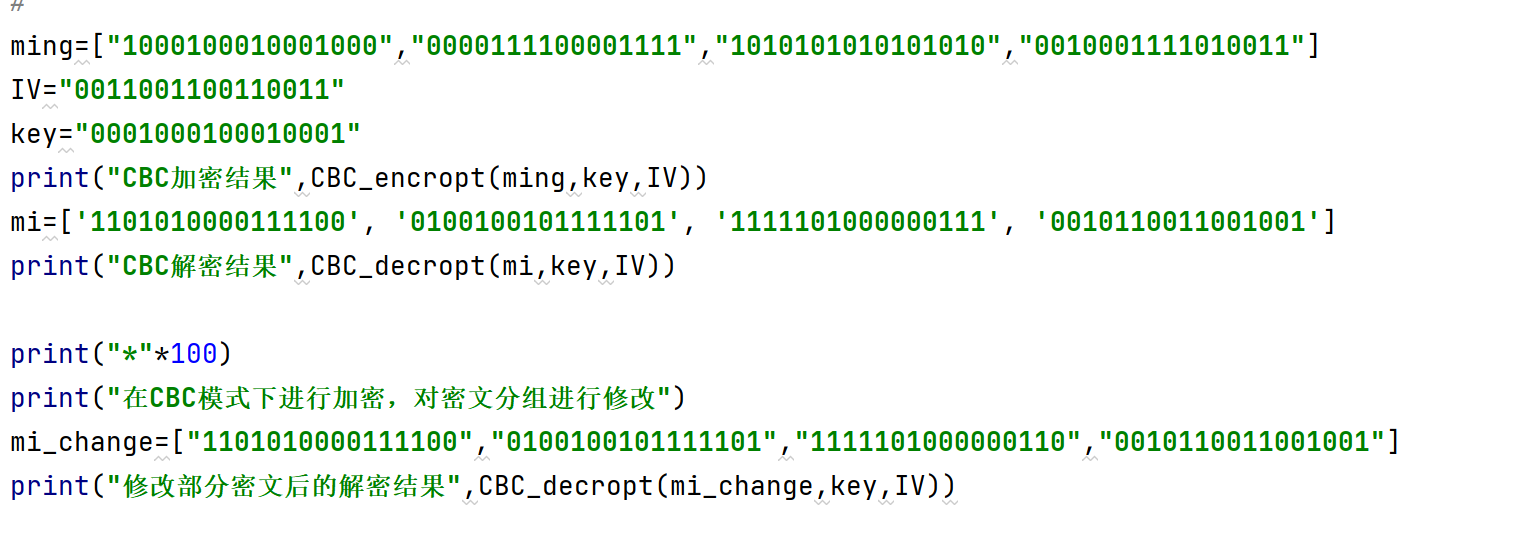
三重解密

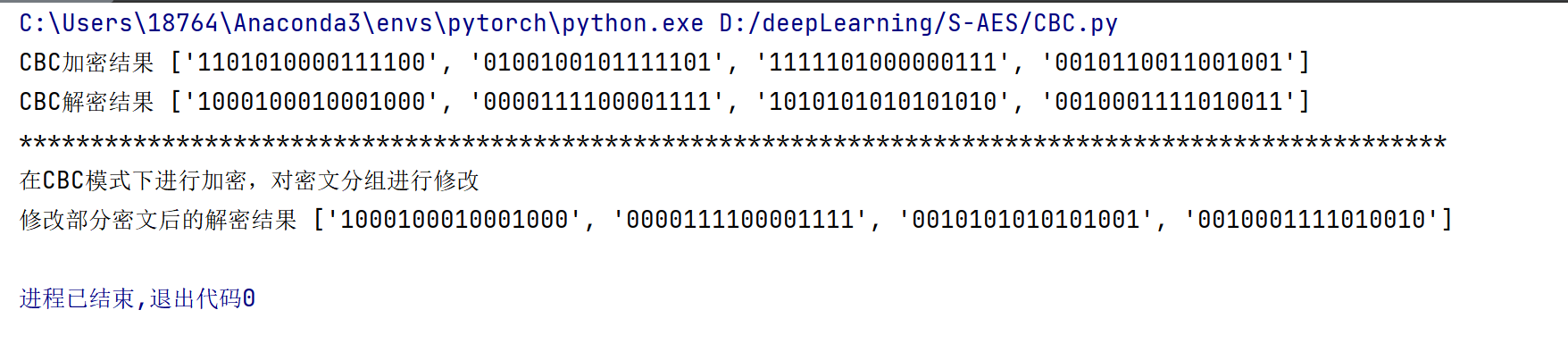


中间相遇攻击



CBC模式





修改中间的密文，导致的结果是在这个密文分组之前的解密结果没有影响，在这个密文分组之后的解密结果错误。