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
```

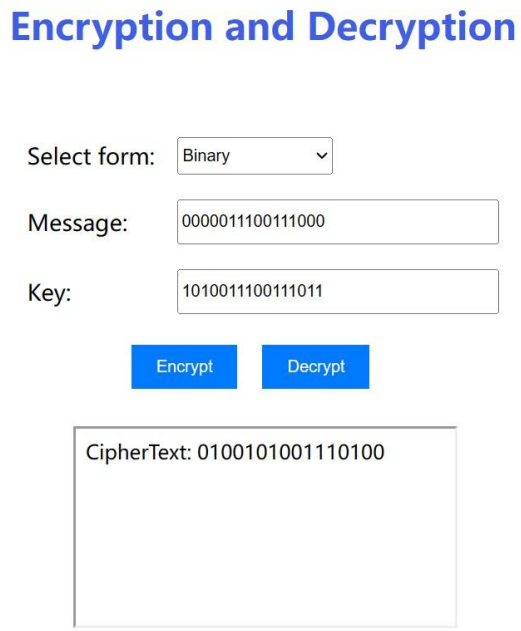
6.CBC 模式

```
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

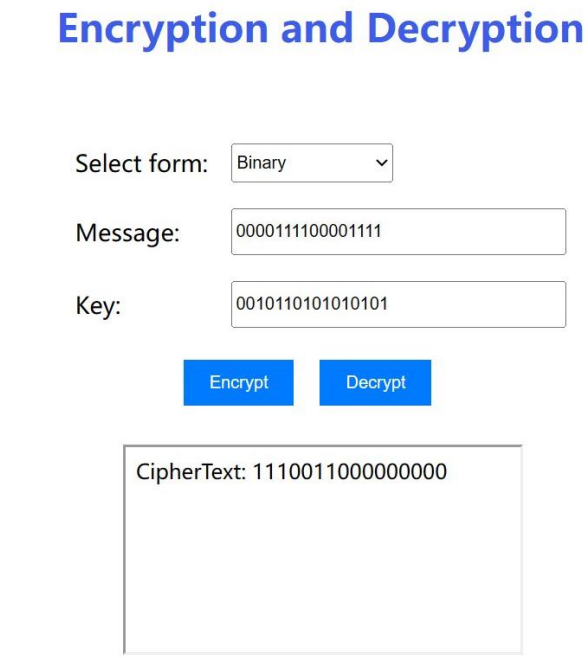
加密：



解密：


S-AES
密钥:
1010011100111011
输入:
0100101001110100
解密结果:
0000011100111000
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
退出

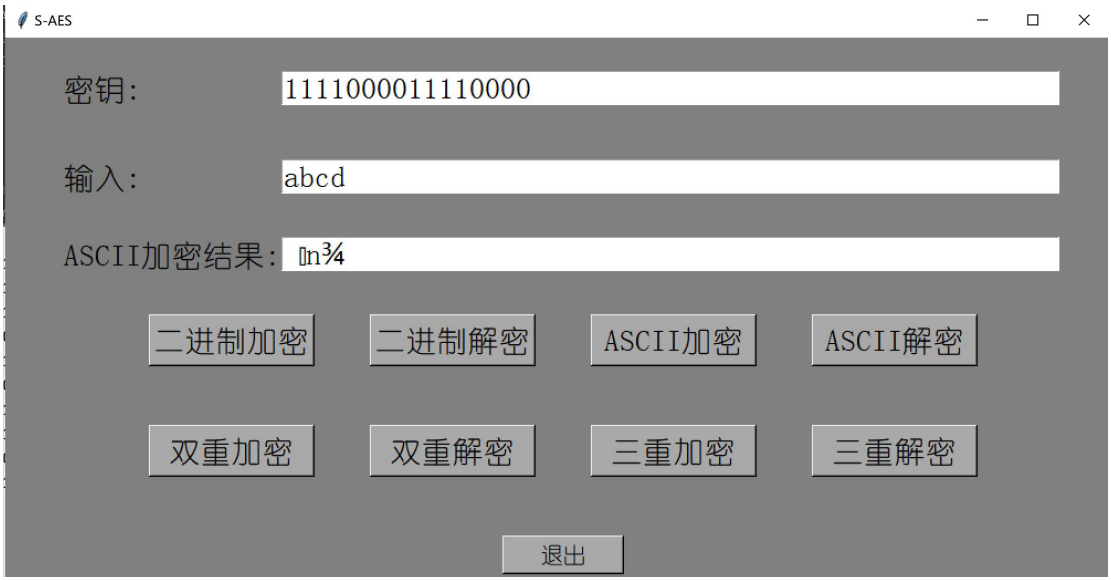
加密


Encryption and Decryption
Select form:
Binary
Message:
0000111100001111
Key:
0010110101010101
Encrypt
Decrypt
CipherText: 1110011000000000

解密


S-AES
密钥:
0010110101010101
输入:
1110011000000000
解密结果:
0000111100001111
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
退出

ASCII 加密



ASCII 解密

二重加密

Multiple Encryption and Decryption

Select form: Double en-decryption

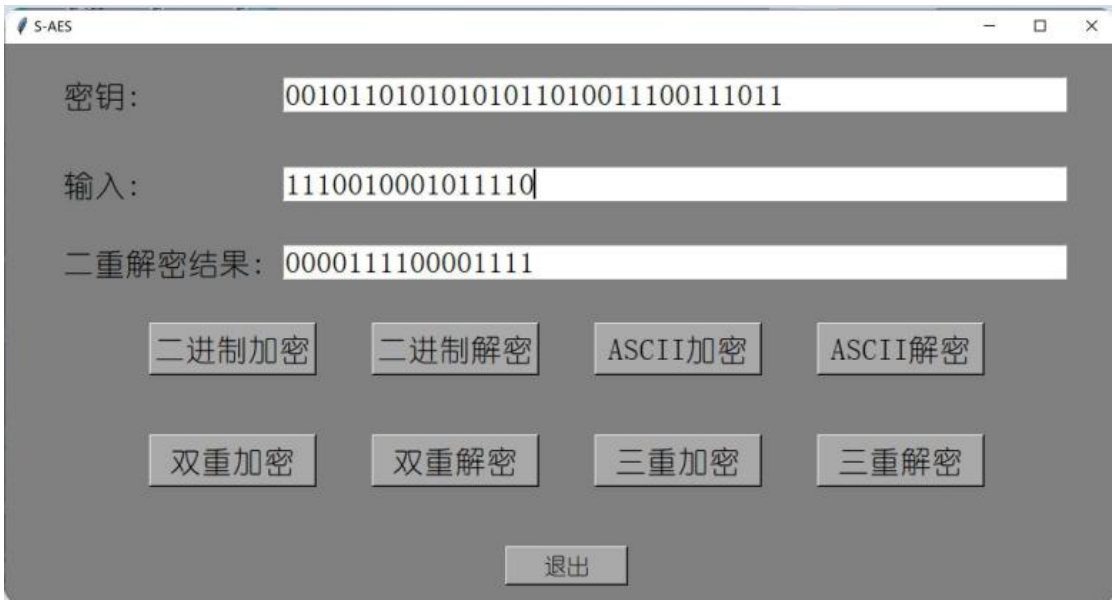
Message: 0000111100001111

Key: 00101101010101011010011100111011

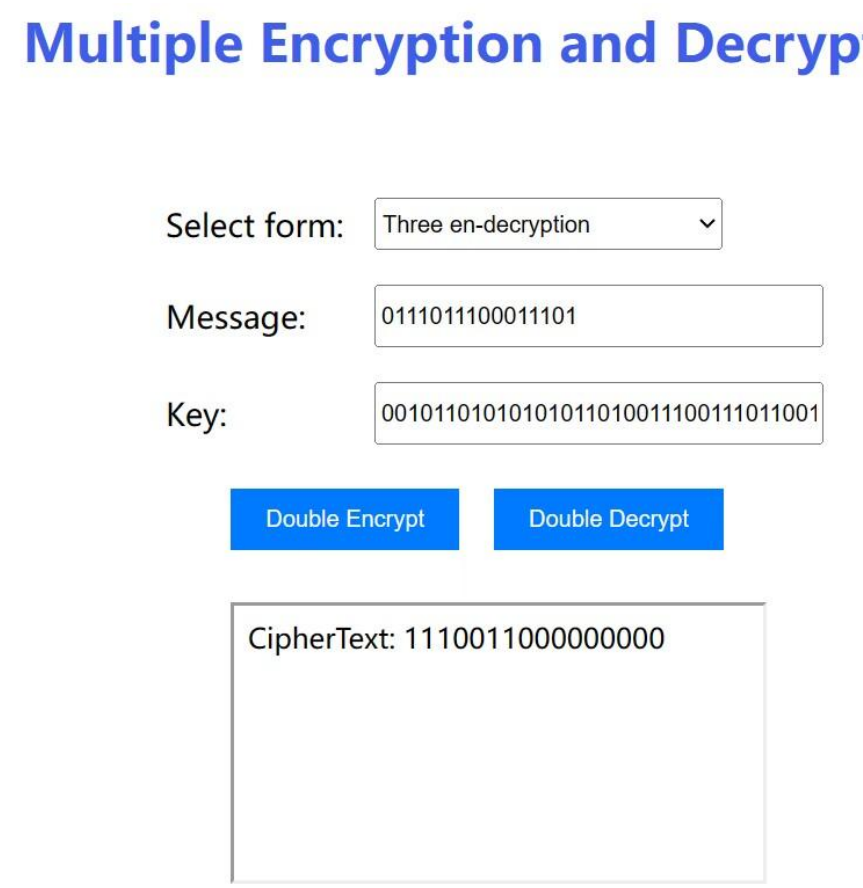
Double Encrypt Double Decrypt

CipherText: 1110010001011110

二重解密



三重加密


Multiple Encryption and Decryption
Select form:
Three en-decryption
Message:
0111011100011101
Key:
00101101010101011010011100111011001
Double Encrypt
Double Decrypt
CipherText: 1110011000000000

三重解密


S-AES
密钥:
00101101010101011010011100111011001011010101010101
输入:
1110011000000000
三重解密结果:
0111011100011101
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
退出

中间相遇攻击

```
371
372 keyword="11110000111100001111000011110000"
373 mingwen=["0110111101101011","1000101110000010","1101100000101000"]
374 miwen=["1000101110000010","1101100000101000","0110111101111011"]
375 center_attack(mingwen, miwen)
```

```
main × S_AES_16bits ×
C:\Users\18764\Anaconda3\envs\pytorch\python.exe D:/deepLearning/S-AES/S_AES_16bits.py
11110000111100001111000011110000
```

CBC 模式

```
#
ming=["1000100010001000","0000111100001111","1010101010101010","0010001111010011"]
IV="0011001100110011"
key="0001000100010001"
print("CBC加密结果",CBC_encropt(ming,key,IV))
mi=['1101010000111100', '0100100101111101', '1111101000000111', '0010110011001001']
print("CBC解密结果",CBC_decropt(mi,key,IV))

print("*"*100)
print("在CBC模式下进行加密，对密文分组进行修改")
mi_change=["1101010000111100","0100100101111101","1111101000000110","0010110011001001"]
print("修改部分密文后的解密结果",CBC_decropt(mi_change,key,IV))
```

```
C:\Users\18764\Anaconda3\envs\pytorch\python.exe D:/deepLearning/S-AES/CBC.py
CBC加密结果 ['1101010000111100', '0100100101111101', '1111101000000111', '0010110011001001']
CBC解密结果 ['1000100010001000', '0000111100001111', '1010101010101010', '0010001111010011']
****************************************************************************************************
在CBC模式下进行加密，对密文分组进行修改
修改部分密文后的解密结果 ['1000100010001000', '0000111100001111', '0010101010101001', '0010001111010010']

进程已结束,退出代码0
```

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。