



南开大学
Nankai University

南开大学

计算机学院和网络空间安全学院

《密码学》课程作业

SPN 线性密码分析 & SM4 差分密码分析

姓名：陆皓喆

学号：2211044

专业：信息安全

指导教师：苏明

2024 年 10 月 14 日

目录

1	实验要求	2
2	github 仓库	2
3	实验原理	2
3.1	线性密码分析	2
3.2	差分密码分析	2
4	Question1	3
5	Question2	7
6	附录 1:Random_Generate_Key_Pairs.py	7
7	附录 2:Linear_Attack.py	8
8	附录 3:Differential_Attack.py	13

1 实验要求

1. 请利用线性密码分析确定 SPN 分组加密算法的 (轮) 密钥;
2. 请计算国密分组加密算法 SM4 的 SBox 差分分布表。

2 github 仓库

本次实验的有关代码和文件, 都已经上传至我的个人 github 中。
您可以通过访问[此链接](#)来查阅我的代码文件。

3 实验原理

3.1 线性密码分析

SPN 线性密码分析是一种基于 S 盒逼近的方法, 已知明文和密文, 分析出对应的密钥。此方法可以分析出最后一轮的 16 位密钥值, 虽然课本上只提供了分析其中八位的伪代码, 但是我们经过分析之后, 得出了剩下八位的分析过程, 进而完成了对整个密钥的破解。总体的流程为:

- 通过上一次实验中所使用的 SPN 生成代码, 随机生成 8000 对明文密文对;
- 构造出一个固定的表达式;
- 通过构建的表达式和前面所生成的明文密文对, 来构建对应的线性逼近表;
- 选取线性逼近表中偏差最大的作为输出, 即可得到对应的密钥值;
- 重复做两次上述操作, 即可获得全部的 16 位密钥值。

对于本次实验, 我们构建的第一轮表达式与课本上所提供的一致, 为:

$$z_1 = x_5 \oplus x_7 \oplus x_8 \oplus u_6^4 \oplus u_8^4 \oplus u_{14}^4 \oplus u_{16}^4$$

通过查阅资料和分析, 我们可以得到另外两组的密钥所对应的表达式为:

$$z_2 = x_1 \oplus x_2 \oplus x_4 \oplus u_1^4 \oplus u_5^4 \oplus u_9^4 \oplus u_{13}^4$$

$$z_3 = x_9 \oplus x_{10} \oplus x_{12} \oplus u_3^4 \oplus u_7^4 \oplus u_{11}^4 \oplus u_{15}^4$$

$$z^{final} = z_2 + z_3$$

3.2 差分密码分析

差分密码分析 (Differential Cryptanalysis) 是一种密码分析方法, 旨在通过观察密码算法在不同输入差分下产生的输出差分, 来推断出密码算法的密钥信息。

差分密码分析的基本原理是利用输入差分对输出结果的影响来推断出密钥信息。具体来说, 攻击者通过控制明文的差分, 观察密文的差分变化, 从而得到密钥的一些信息。这种方法的关键在于寻找输入差分和输出差分之间的关联, 以便从中提取出有用的信息。

本次实验，我们要求解出 SM4 的差分分布表，所以我们只需要对其分布情况进行求解，并不需要进行差分攻击。那就变得很简单了，我们根据课本上所提供的方法便可以很快完成对应的任务。

具体的算法为：

$$N_D(x', y') = |\{(x, x^*) \in \Delta(x') : \pi_S(x) \oplus \pi_S(x^*) = y'\}|$$

4 Question1

本问题中，我们主要使用上一次实验所编写的 SPN 脚本来生成本次实验所需要的明文密文对，然后使用线性分析攻击算法来实现对 16 位全体密钥的破解。下面我对代码做一下详细的分析。

首先分析一下 Random_Generate_Key_Pairs.py 脚本。该脚本主要完成了 8000 对明文密文对的随机生成，并且将生成结果存储到了 pairs.txt 中。具体的代码由于过长，我将其放在了附录 1 中，此处主要做一下核心代码部分的分析。

```
1 def generate_random_binary_string(length=16, times=1):
2     binary_strings = [] # 创建一个空列表来存储生成的二进制字符串
3     for _ in range(times): # 循环指定的次数
4         binary_strings.append(''.join(random.choice('01') for _ in range(length)))
5     return binary_strings
```

该段代码实现了 01 二进制数的随机生成，规定了长度为 16，times 表示我们需要生成的对数，我们在程序中的最前面也声明了对数为 8000 对。

```
1 exe_file_path = "./SPN.exe"
2
3 def SPN_Process(times=1):
4     processing_strings = []
5     for i in range(times):
6         temp_data = subprocess.run([exe_file_path],
7                                     input=binary_strings_list[i].encode(), stdout=subprocess.PIPE,
8                                     stderr=subprocess.PIPE)
9         processing_strings.append(temp_data.stdout.decode().strip())
10    return processing_strings
```

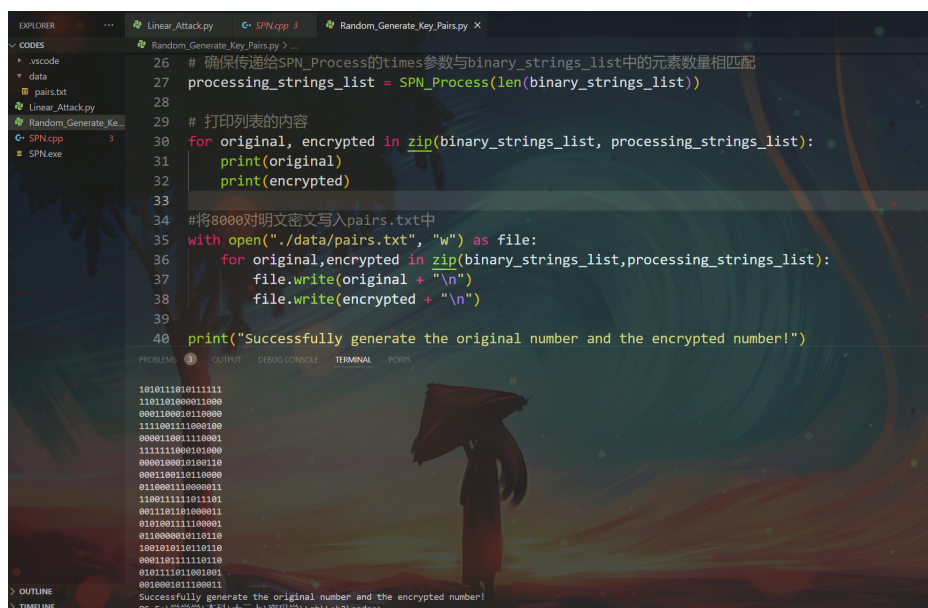
这段代码，我首先使用了上次实验中所编写的 C++ 文件生成了对应的 exe 可执行文件，然后使用 python 中的 subprocess 库对其进行了调用运算。

然后，我们使用在前面所生成的 8000 个随机明文数，通过 SPN_process 函数的运算，将输出的密文结果存储到了 processing_strings_list 中，这样我们就拥有了两个长度为 8000 的列表，分别为我们生成的明文和所计算出的密文。

```
1 with open("./data/pairs.txt", "w") as file:
2     for original, encrypted in zip(binary_strings_list, processing_strings_list):
3         file.write(original + "\n")
4         file.write(encrypted + "\n")
```

然后将两个列表的值写入 pairs.txt 中即可，明文密文对所在的文件的格式为：明文 1、密文 1、明文 2、密文 2.....

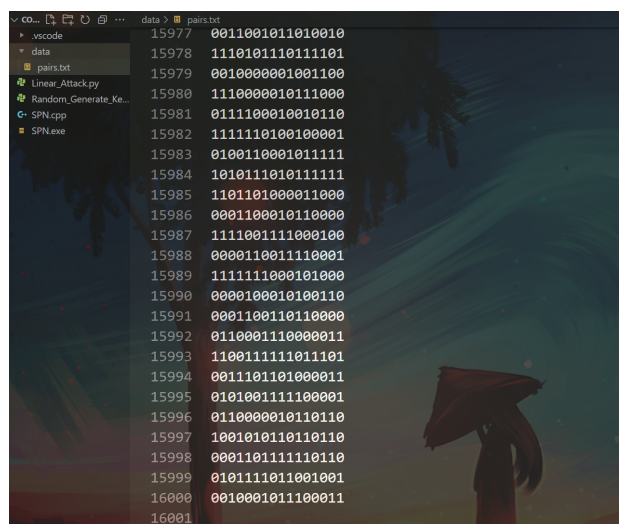
这样我们的明文密文对的生成过程就完成了，运行 py 程序，得到结果如图4.1所示。



```
26 # 确保传递给SPN_Process的times参数与binary_strings_list中的元素数量相匹配
27 processing_strings_list = SPN_Process(len(binary_strings_list))
28
29 # 打印列表的内容
30 for original, encrypted in zip(binary_strings_list, processing_strings_list):
31     print(original)
32     print(encrypted)
33
34 #将8000对明文密文写入pairs.txt中
35 with open("./data/pairs.txt", "w") as file:
36     for original, encrypted in zip(binary_strings_list, processing_strings_list):
37         file.write(original + "\n")
38         file.write(encrypted + "\n")
39
40 print("Successfully generate the original number and the encrypted number!")
```

图 4.1: 生成 8000 对明文密文对

打开 pairs.txt 文件查看，发现确实写入了 16000 行数据 (如图4.2所示)。



```
15977 0011001011010010
15978 1110101110111101
15979 0010000001001100
15980 1110000010111000
15981 0111100010010110
15982 1111110100100001
15983 0100110001011111
15984 1010111010111111
15985 1101101000011000
15986 0001100010110000
15987 1111001111000100
15988 0000110011110001
15989 1111111000101000
15990 0000100010100110
15991 0001100110110000
15992 0110001110000011
15993 1100111111011101
15994 0011101101000011
15995 0101001111100001
15996 0110000010110110
15997 1001010110110110
15998 0001101111110110
15999 0101111011001001
16000 0010001011100011
16001
```

图 4.2: pairs.txt 中生成的内容

我们成功生成了 8000 对明文密文对，并且将其存储在了 pairs.txt 中。

下面，我们就要进行相应的线性密码分析了。对应的 python 脚本由于长度问题，我将其放在了附录 2 中，此处仅仅对其进行关键部分代码的分析。

```
1 def solve_reverse_s_box(s_box):
2     reverse_box=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
3     for i in range(len(s_box)):
```

```
4     reverse_box[s_box[i]]=i
5     return reverse_box
```

solve_reverse_s_box 函数主要完成了对 S 盒求逆的操作。操作比较简单，此处不再赘述。

然后，我们完成了对代码所需要的列表的初始化操作，并打开 pairs.txt 文件，分别读取对应的明文和密文，存入 original 列表和 encrypted 列表中。

然后，开始我们的核心部分代码——线性攻击。我们主要分为两个步骤对其进行攻击，首先根据课本上的伪代码，对其中的八位进行攻击；然后根据所构建的另一个表达式，完成剩下八位的攻击。最后，将分析结果输出。

首先分析一下第一次攻击。

```
1 def Linear_Attack_First(original,encrypted): #攻击课本中的八位密钥
2     for i in range(8000):
3         original_temp=original[i]
4         encrypted_temp=encrypted[i]
5         original_temp = original_temp.replace('\n', '').replace('\r', '')
6         encrypted_temp = encrypted_temp.replace('\n', '').replace('\r', '')
7         encrypted_test2=int(encrypted_temp[4:8],2)
8         encrypted_test4=int(encrypted_temp[12:],2)
9         x=[None]+[int(i) for i in original_temp]
10        for i in range(16):
11            for j in range(16):
12                u2 = [int(i) for i in format(reverse_pai_s[i ^ encrypted_test2],
13                    '04b')]
14                u4 = [int(i) for i in format(reverse_pai_s[j ^ encrypted_test4],
15                    '04b')]
16                u = [None] * 5 + u2 + [None] * 4 + u4
17                z = x[5] ^ x[7] ^ x[8] ^ u[6] ^ u[8] ^ u[14] ^ u[16]
18                if z&1==0:
19                    count_list[i][j]=count_list[i][j]+1;
20
21        for i in range(16):
22            for j in range(16):
23                count_list[i][j]=count_list[i][j]-4000
24                if count_list[i][j]<0:
25                    count_list[i][j]=-count_list[i][j]
26
27        temp1=0
28        temp2=0
29        count_max=0
30        for i in range(16):
31            for j in range(16):
32                if count_list[i][j]>count_max:
```

```
31         count_max=count_list[i][j]
32         temp1=i
33         temp2=j
34
35     return [temp1,temp2]
```

首先，我们分 8000 次来完成对 count 的计数累加。本次读取的 encrypted_test2 为 5-8 位密钥，encrypted_test4 为最后四位密钥。x 就是将我们的初始明文填充之后的结果。然后，通过 for 循环遍历 16*16 数组的每一个位置，通过 S 盒逆运算，将得到的值 u2 和 u4 存入列表 u 中，然后按照课本所提供的表达式进行计算即可，得到 z，通过判断 z 是否为 0，如果是 0 的话就将 count 的值加一。

然后，通过运算，count 减去 8000 的一半，再取绝对值。最后找出最大的那个，也就是偏差最大的，也就是我们所需要的部分密钥了！

另外的第二次攻击大体上与第一次攻击类似，我分析一下不同之处。

```
1 result1=Linear_Attack_First(original,encrypted)[0]
2 result2=Linear_Attack_First(original,encrypted)[1]
```

此处代码的含义是，将第一次分析的结果，直接在第二次分析中使用，分别为 result1 和 result2。

```
1 encrypted_test1=int(encrypted_temp[:4],2)
2 encrypted_test2=int(encrypted_temp[4:8],2)
3 encrypted_test3=int(encrypted_temp[8:12],2)
4 encrypted_test4=int(encrypted_temp[12:],2)
```

此处，我们需要将四组密文数据均读入。

```
1 u = [None] + [int(i) for i in format(reverse_pai_s[i ^ encrypted_test1], '04b') + \
2             format(reverse_pai_s[result1 ^ encrypted_test2], '04b') + \
3             format(reverse_pai_s[j ^ encrypted_test3], '04b') + \
4             format(reverse_pai_s[result2 ^ encrypted_test4], '04b')]
5 z=x[1] ^ x[2] ^ x[4] ^ u[1] ^ u[5] ^ u[9] ^ u[13]
6 if z&1==0:
7     count_list_first[i][j]=count_list_first[i][j]+1
8 z = x[9] ^ x[10] ^ x[12] ^ u[3] ^ u[7] ^ u[11] ^ u[15]
9 if z&1==0:
10    count_list_second[i][j]=count_list_second[i][j]+1
```

此处，我们通过此处的数据以及前一轮分析的结果构造出列表 u，然后首先计算 z1，叠加 z1；再计算 z2，同样叠加 z2。最后，我们判断 z 是否为 0，去对应的计算 count 的值。

最后，我们需要取的值，为两个 count 之和偏差最大的横纵坐标的值。

两轮分析都结束后，我们只需要将分析结果转为二进制字符串进行输出即可。我们测试了不同的 10 组密钥，每组密钥测试了 5 轮，攻击成功率为 90% 以上。下面仅展示某一次测试时的结果。

```
0101111011001001
0010001011100011
Successfully generate the original number and the encrypted number!
PS E:\学学学\本科\大三上\密码学\Lab\Lab2\codes> & C:/Users/Lenovo/AppData/Local
1101011000111111
PS E:\学学学\本科\大三上\密码学\Lab\Lab2\codes>
```

图 4.3: 线性攻击结果

我们通过运行 py 脚本进行测试, 得到了对应的密钥 1101011000111111, 这和我们在 SPN 生成脚本中所使用的密钥的后 16 位 (第五轮) 相同 (如图 4.4 所示)。这说明我们的攻击成功了!

```
int main() {
    string plaintext, key; // 定义明文和密钥
    cin >> plaintext; // 输入明文
    key = '00111010100101001101011000111111';
```

图 4.4: SPN 生成中使用的 key

5 Question2

本部分我们需要求解 SM4 表的差分分布表, 实际上这只是差分攻击的第一步, 因此我们只需要编写几行代码就可以完成对其的求解。我将完整代码放到了附录 3 中, 此处挑一些核心代码做一下说明。

```
1 for i in range(len(s_box)):
2     for j in range(len(s_box)):
3         s_box_diff[j^i][s_box[j]^s_box[i]] = s_box_diff[j^i][s_box[j]^s_box[i]] + 1
```

该部分代码, 我们完成了对 256×256 的差分分布的遍历, 对于每一个差分值都进行计算, 运算规则为: 遍历 256×256 的矩阵, 对于横坐标为 j^i 和纵坐标为 $s_box[j] \oplus s_box[i]$ 的位置进行加一操作, 全部循环一轮即可。

然后我们对整个 s_box_diff 的矩阵进行输出, 输出到 csv 文件中, 就完成了本题的解答。本题的输出为一个 256×256 的矩阵, 由于内容过于庞大, 我将其放在了文件夹中, 就不在此处展示了。

6 附录 1: Random_Generate_Key_Pairs.py

```
1 import random
2 import subprocess
3
4 counts=8000
5
6 # 生成了16位的二进制数
7 def generate_random_binary_string(length=16, times=1):
8     binary_strings = [] # 创建一个空列表来存储生成的二进制字符串
9     for _ in range(times): # 循环指定的次数
```



```
10     binary_strings.append(''.join(random.choice('01') for _ in range(length)))
11     return binary_strings
12
13 # 使用函数生成n个长度为16的随机二进制字符串
14 binary_strings_list = generate_random_binary_string(16, times=counts)
15
16 # 规定生成密钥的可执行文件路径
17 exe_file_path = "./SPN.exe"
18
19 def SPN_Process(times=1):
20     processing_strings = []
21     for i in range(times):
22         temp_data = subprocess.run([exe_file_path],
23                                     input=binary_strings_list[i].encode(), stdout=subprocess.PIPE,
24                                     stderr=subprocess.PIPE)
25         processing_strings.append(temp_data.stdout.decode().strip())
26     return processing_strings
27
28 # 确保传递给SPN_Process的times参数与binary_strings_list中的元素数量相匹配
29 processing_strings_list = SPN_Process(len(binary_strings_list))
30
31 # 打印列表的内容
32 for original, encrypted in zip(binary_strings_list, processing_strings_list):
33     print(original)
34     print(encrypted)
35
36 #将8000对明文密文写入pairs.txt中
37 with open("./data/pairs.txt", "w") as file:
38     for original, encrypted in zip(binary_strings_list, processing_strings_list):
39         file.write(original + "\n")
40         file.write(encrypted + "\n")
41
42 print("Successfully generate the original number and the encrypted number!")
```

7 附录 2:Linear__Attack.py

```
1 pai_s=[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7]
2 l = 4
3 m = 4
4 Nr = 4
5
```

```
6 #define some functions
7 def solve_reverse_s_box(s_box):
8     reverse_box=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
9     for i in range(len(s_box)):
10         reverse_box[s_box[i]]=i
11     return reverse_box
12
13 #init the s box
14 reverse_pai_s=solve_reverse_s_box(pai_s)
15 pairs_num=8000 #用于攻击的随机生成密钥对数量
16
17 count_list = [[0 for _ in range(16)] for _ in range(16)] #初始化count数组
18
19 original=[0 for _ in range(8000)]
20 encrypted=[0 for _ in range(8000)]
21
22 # 打开文件并读取内容
23 with open('data/pairs.txt', 'r') as file:
24     lines = file.readlines() # 读取所有行
25
26 # 将两行一对分组
27 for i in range(0, len(lines), 2):
28     original[i//2]=lines[i]
29     encrypted[i//2]=lines[i+1]
30
31 def Linear_Attack_First(original,encrypted): #攻击课本中的八位密钥
32     for i in range(8000):
33         original_temp=original[i]
34         encrypted_temp=encrypted[i]
35         original_temp = original_temp.replace('\n', '').replace('\r', '')
36         encrypted_temp = encrypted_temp.replace('\n', '').replace('\r', '')
37         encrypted_test2=int(encrypted_temp[4:8],2)
38         encrypted_test4=int(encrypted_temp[12:],2)
39         x=[None]+[int(i) for i in original_temp]
40         for i in range(16):
41             for j in range(16):
42                 u2 = [int(i) for i in format(reverse_pai_s[i ^ encrypted_test2],
43                     '04b')]
43                 u4 = [int(i) for i in format(reverse_pai_s[j ^ encrypted_test4],
44                     '04b')]
44                 u = [None] * 5 + u2 + [None] * 4 + u4
45                 z = x[5] ^ x[7] ^ x[8] ^ u[6] ^ u[8] ^ u[14] ^ u[16]
```

```
46         if z&1==0:
47             count_list[i][j]=count_list[i][j]+1;
48
49     for i in range(16):
50         for j in range(16):
51             count_list[i][j]=count_list[i][j]-4000
52             if count_list[i][j]<0:
53                 count_list[i][j]=-count_list[i][j]
54
55     temp1=0
56     temp2=0
57     count_max=0
58     for i in range(16):
59         for j in range(16):
60             if count_list[i][j]>count_max:
61                 count_max=count_list[i][j]
62                 temp1=i
63                 temp2=j
64
65     return [temp1,temp2]
66
67 count_list_first = [[0 for _ in range(16)] for _ in range(16)]
68 count_list_second = [[0 for _ in range(16)] for _ in range(16)]
69 count_list_all=[[0 for _ in range(16)] for _ in range(16)]
70
71 def Linear_Attack_Second(original,encrypted):
72     result1=Linear_Attack_First(original,encrypted)[0]
73     result2=Linear_Attack_First(original,encrypted)[1]
74     for i in range(8000):
75         original_temp=original[i]
76         encrypted_temp=encrypted[i]
77         original_temp = original_temp.replace('\n', '').replace('\r', '')
78         encrypted_temp = encrypted_temp.replace('\n', '').replace('\r', '')
79         encrypted_test1=int(encrypted_temp[:4],2)
80         encrypted_test2=int(encrypted_temp[4:8],2)
81         encrypted_test3=int(encrypted_temp[8:12],2)
82         encrypted_test4=int(encrypted_temp[12:],2)
83         x=[None]+[int(i) for i in original_temp]
84
85         for i in range(16):
86             for j in range(16):
87                 u = [None] + [int(i) for i in format(reverse_pai_s[i ^
```

```
encrypted_test1], '04b') + \
88         format(reverse_pai_s[result1 ^
            encrypted_test2], '04b') + \
89         format(reverse_pai_s[j ^ encrypted_test3],
            '04b') + \
90         format(reverse_pai_s[result2 ^
            encrypted_test4], '04b')]
91 z=x[1] ^ x[2] ^ x[4] ^ u[1] ^ u[5] ^ u[9] ^ u[13]
92 if z&1==0:
93     count_list_first[i][j]=count_list_first[i][j]+1
94 z = x[9] ^ x[10] ^ x[12] ^ u[3] ^ u[7] ^ u[11] ^ u[15]
95 if z&1==0:
96     count_list_second[i][j]=count_list_second[i][j]+1
97
98 for i in range(16):
99     for j in range(16):
100         count_list_first[i][j]=count_list_first[i][j]-4000
101         if count_list_first[i][j]<0:
102             count_list_first[i][j]=-count_list_first[i][j]
103
104 for i in range(16):
105     for j in range(16):
106         count_list_second[i][j]=count_list_second[i][j]-4000
107         if count_list_second[i][j]<0:
108             count_list_second[i][j]=-count_list_second[i][j]
109
110 for i in range(16):
111     for j in range(16):
112         count_list_all[i][j]=count_list_first[i][j]+count_list_second[i][j]
113
114 temp3=0
115 temp4=0
116 count_max=0
117 for i in range(16):
118     for j in range(16):
119         if count_list_all[i][j]>count_max:
120             count_max=count_list_all[i][j]
121             temp3=i
122             temp4=j
123
124 return [temp3,temp4]
125
```

```
126 result_list1=Linear_Attack_First(original,encrypted)
127 result_list2=Linear_Attack_Second(original,encrypted)
128
129
130 def Dec_To_Bin(a):
131     match a:
132         case 0:
133             return "0000"
134         case 1:
135             return "0001"
136         case 2:
137             return "0010"
138         case 3:
139             return "0011"
140         case 4:
141             return "0100"
142         case 5:
143             return "0101"
144         case 6:
145             return "0110"
146         case 7:
147             return "0111"
148         case 8:
149             return "1000"
150         case 9:
151             return "1001"
152         case 10:
153             return "1010"
154         case 11:
155             return "1011"
156         case 12:
157             return "1100"
158         case 13:
159             return "1101"
160         case 14:
161             return "1110"
162         case 15:
163             return "1111"
164
165 result1=Dec_To_Bin(result_list1[0])
166 result2=Dec_To_Bin(result_list1[1])
167 result3=Dec_To_Bin(result_list2[0])
```

```
168 result4=Dec_To_Bin(result_list2[1])
169
170 final_result=result3+result1+result4+result2
171
172 print(final_result)
```

8 附录 3:Differential_Attack.py

```
1 import csv
2 s_box=[
3     0xd6, 0x90, 0xe9, 0xfe, 0xcc, 0xe1, 0x3d, 0xb7, 0x16, 0xb6, 0x14, 0xc2, 0x28,
4     0xfb, 0x2c, 0x05,
5     0x2b, 0x67, 0x9a, 0x76, 0x2a, 0xbe, 0x04, 0xc3, 0xaa, 0x44, 0x13, 0x26, 0x49,
6     0x86, 0x06, 0x99,
7     0x9c, 0x42, 0x50, 0xf4, 0x91, 0xef, 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43, 0xed,
8     0xcf, 0xac, 0x62,
9     0xe4, 0xb3, 0x1c, 0xa9, 0xc9, 0x08, 0xe8, 0x95, 0x80, 0xdf, 0x94, 0xfa, 0x75,
10    0x8f, 0x3f, 0xa6,
11    0x47, 0x07, 0xa7, 0xfc, 0xf3, 0x73, 0x17, 0xba, 0x83, 0x59, 0x3c, 0x19, 0xe6,
12    0x85, 0x4f, 0xa8,
13    0x68, 0x6b, 0x81, 0xb2, 0x71, 0x64, 0xda, 0x8b, 0xf8, 0xeb, 0x0f, 0x4b, 0x70,
14    0x56, 0x9d, 0x35,
15    0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, 0xd1, 0xa2, 0x25, 0x22, 0x7c, 0x3b, 0x01,
16    0x21, 0x78, 0x87,
17    0xd4, 0x00, 0x46, 0x57, 0x9f, 0xd3, 0x27, 0x52, 0x4c, 0x36, 0x02, 0xe7, 0xa0,
18    0xc4, 0xc8, 0x9e,
19    0xea, 0xbf, 0x8a, 0xd2, 0x40, 0xc7, 0x38, 0xb5, 0xa3, 0xf7, 0xf2, 0xce, 0xf9,
20    0x61, 0x15, 0xa1,
21    0xe0, 0xae, 0x5d, 0xa4, 0x9b, 0x34, 0x1a, 0x55, 0xad, 0x93, 0x32, 0x30, 0xf5,
22    0x8c, 0xb1, 0xe3,
23    0x1d, 0xf6, 0xe2, 0x2e, 0x82, 0x66, 0xca, 0x60, 0xc0, 0x29, 0x23, 0xab, 0x0d,
24    0x53, 0x4e, 0x6f,
25    0xd5, 0xdb, 0x37, 0x45, 0xde, 0xfd, 0x8e, 0x2f, 0x03, 0xff, 0x6a, 0x72, 0x6d,
26    0x6c, 0x5b, 0x51,
27    0x8d, 0x1b, 0xaf, 0x92, 0xbb, 0xdd, 0xbc, 0x7f, 0x11, 0xd9, 0x5c, 0x41, 0x1f,
28    0x10, 0x5a, 0xd8,
29    0x0a, 0xc1, 0x31, 0x88, 0xa5, 0xcd, 0x7b, 0xbd, 0x2d, 0x74, 0xd0, 0x12, 0xb8,
30    0xe5, 0xb4, 0xb0,
31    0x89, 0x69, 0x97, 0x4a, 0x0c, 0x96, 0x77, 0x7e, 0x65, 0xb9, 0xf1, 0x09, 0xc5,
32    0x6e, 0xc6, 0x84,
33    0x18, 0xf0, 0x7d, 0xec, 0x3a, 0xdc, 0x4d, 0x20, 0x79, 0xee, 0x5f, 0x3e, 0xd7,
```

```
0xcb, 0x39, 0x48
19 ]
20
21 s_box_diff = [[0 for _ in range(len(s_box))] for _ in range(len(s_box))]
22
23
24 for i in range(len(s_box)):
25     for j in range(len(s_box)):
26         s_box_diff[j^i][s_box[j]^s_box[i]] = s_box_diff[j^i][s_box[j]^s_box[i]] + 1
27
28
29 with open('result/s_box_diff.csv', 'w', newline='') as csvfile:
30     writer = csv.writer(csvfile)
31     for i in range(len(s_box)):
32         row = [s_box_diff[i][j] for j in range(len(s_box))]
33         writer.writerow(row)
```