

线性攻击算法

姓名：齐明杰 学号：2113997 班级：信安2班

1 算法原理

SPN (Substitution-Permutation Network) 是一种块密码结构，由多轮的替代 (substitution) 和置换 (permutation) 操作组成。这种加密方式有如下的特点：

1. **替代 (Substitution)**: 这是通过 S-box 实现的，它是一个预定义的表，用于将输入映射到输出。S-box 的选择对于算法的安全性至关重要。
2. **置换 (Permutation)**: 这是通过 P-box 实现的，它重新排列其输入位。
3. **密钥混合**: 每一轮都会有一个子密钥与数据进行异或操作。

线性攻击是一种针对块密码的密码分析方法，目的是找到明文和密文之间的线性关系。这种关系可以用于减少恢复密钥所需的尝试次数。

不妨设最后一轮密钥 $K = K_1K_2K_3K_4$ ，则

根据课本知识， K_2 和 K_4 可以由以下线性分析链得出：

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

另查阅资料，发现 K_1 和 K_3 可以由另一个线性分析链得出：

$$\begin{aligned} z_2 &= x_1 \oplus x_2 \oplus x_4 \oplus u_1^4 \oplus u_5^4 \oplus u_9^4 \oplus u_{13}^4 \\ z_3 &= x_9 \oplus x_{10} \oplus x_{12} \oplus u_3^4 \oplus u_7^4 \oplus u_{11}^4 \oplus u_{15}^4 \\ z^* &= z_2 + z_3 \end{aligned}$$

2 代码

我使用Python进行编程，完成了两份代码的编写：

- util.py

这份代码主要编写SPN加密算法的实现，用于产生密钥以及明文-密文对。

```
1  import random
2
3  l = 4      # 每组比特数
4  m = 4      # 组数
5  Nr = 4     # 轮数
6  Sbox = [0xE, 0x4, 0xD, 0x1, 0x2, 0xF, 0xB, 0x8,
7          0x3, 0xA, 0x6, 0xC, 0x5, 0x9, 0x0, 0x7]      # S盒
8  Pbox = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16] # P盒
9
10
```

```

11 def reSbox(Sbox):
12     '''
13     生成逆S盒
14     param Sbox: S盒
15     '''
16     reSbox = [0 for _ in Sbox]
17     for i in range(len(Sbox)):
18         reSbox[Sbox[i]] = i
19     return reSbox
20
21
22 def xor(w, K):
23     '''
24     异或运算
25     param w: 二进制字符串
26     param K: 二进制字符串
27     '''
28     assert len(w) == len(K)
29     u = ''
30     for _w, _k in zip(w, K):
31         u += str(int(_w) ^ int(_k))
32     return u
33
34 def sbbox(u):
35     '''
36     S盒运算
37     param u: 二进制字符串
38     '''
39     v = ''
40     for i in range(m):
41         ui = u[i * l : i * l + l]
42         vi = format(Sbox[int(ui, 2)], '04b')
43         v += vi
44     return v
45
46 def pbox(v):
47     '''
48     P盒运算
49     param v: 二进制字符串
50     '''
51     assert len(v) == l * m
52     w = ''
53     for i in range(len(v)):
54         w += v[Pbox[i] - 1]
55     return w
56

```

```

57
58 def spn(plain, key):
59     '''
60     SPN加密算法
61     param plain: 明文
62     param key: 密钥
63     '''
64     keys = [key[i * Nr : i * Nr + 1 * m] for i in range(Nr + 1)]
65     w = plain
66     for r in range(Nr - 1):
67         u = xor(w, keys[r])
68         v = sbbox(u)
69         w = pbox(v)
70     u = xor(w, keys[Nr - 1])
71     v = sbbox(u)
72     y = xor(v, keys[Nr])
73     return y
74
75
76 def generate_key(n):
77     '''
78     生成密钥
79     param n: 密钥个数
80     '''
81     keys = []
82     for _ in range(n):
83         key = ''
84         for _ in range(1 * m + 4 * Nr):
85             key += str(random.randint(0, 1))
86         keys.append(key)
87     return keys
88
89
90 def generate_data(T, key):
91     '''
92     生成明文-密文对
93     param T: 明文-密文对个数
94     '''
95     data = []
96     for _ in range(T):
97         plain = bin(random.randint(0, 2 ** (1 * m) - 1))[2:].zfill(1 * m)
98         y = spn(plain, key)
99         data.append((plain, y))
100    return data

```

代码解析:

1. `reSbox()`: 计算逆 S-box, 使得我们可以从输出值回到输入值。
2. `xor()`: 这是一个简单的异或操作, 用于两个字符串。
3. `sbox()`: 应用 S-box 到输入字符串。
4. `pbox()`: 应用 P-box 到输入字符串。
5. `spn()`: SPN 的主要实现, 函数使用子密钥、S-box 和 P-box 对明文进行加密。
6. `generate_key()`: 输入要生成的数量, 生成密钥。
7. `generate_data()`: 输入要生成的数量, 生成随机的明文和对应的密文。

- `spn_re.py`

这份代码为核心代码, 实现了对SPN算法最后一轮密钥的线性攻击。

```

1  from util import generate_data, generate_key, reSbox
2  import numpy as np
3  from concurrent.futures import ProcessPoolExecutor
4
5  Sbox = [0xE, 0x4, 0xD, 0x1, 0x2, 0xF, 0xB, 0x8,
6          0x3, 0xA, 0x6, 0xC, 0x5, 0x9, 0x0, 0x7]      # S盒
7
8  reSbox = reSbox(Sbox)      # 逆S盒
9  T = 10000      # 每个密钥生成的明文-密文对个数
10 N = 50      # 待攻击密钥个数
11
12 def reverse_spn(data):
13     '''
14     线性攻击算法
15     param data: 明文-密文对
16     return: 最后一轮的密钥
17     '''
18     lens = len(data)
19     counter = np.zeros((16, 16), dtype=int)
20     for pair in data:
21         x, y = pair
22         y2, y4 = int(y[4:8], 2), int(y[12:], 2)
23         x = [None] + [int(i) for i in x]      # 获取明文的每一位
24         for L1 in range(16):
25             for L2 in range(16):
26                 u2 = [int(i) for i in format(reSbox[L1 ^ y2], '04b')]
27                 u4 = [int(i) for i in format(reSbox[L2 ^ y4], '04b')]
28                 u = [None] * 5 + u2 + [None] * 4 + u4
29                 z = x[5] ^ x[7] ^ x[8] ^ u[6] ^ u[8] ^ u[14] ^ u[16]
30                 if z & 1 == 0:
31                     counter[L1][L2] += 1
32     counter = np.abs(counter - 0.5 * lens)
33     k2, k4 = map(lambda x: x[0], np.where(counter == np.max(counter)))      #
    找到最大值的位置

```

```

34
35     counter1 = np.zeros((16, 16), dtype=int)
36     counter2 = np.zeros((16, 16), dtype=int)
37     for pair in data:
38         x, y = pair
39         y1, y2, y3, y4 = int(y[:4], 2), int(y[4:8], 2), int(y[8:12], 2),
int(y[12:], 2)
40         x = [None] + [int(i) for i in x]      # 获取明文的每一位
41         for L1 in range(16):
42             for L2 in range(16):
43                 u = [None] + [int(i) for i in format(reSbox[L1 ^ y1],
'04b') + \
44                                                         format(reSbox[k2 ^ y2],
'04b') + \
45                                                         format(reSbox[L2 ^ y3],
'04b') + \
46                                                         format(reSbox[k4 ^ y4],
'04b')]]
47                 z = x[1] ^ x[2] ^ x[4] ^ u[1] ^ u[5] ^ u[9] ^ u[13]
48                 if z & 1 == 0:
49                     counter1[L1][L2] += 1
50                 z = x[9] ^ x[10] ^ x[12] ^ u[3] ^ u[7] ^ u[11] ^ u[15]
51                 if z & 1 == 0:
52                     counter2[L1][L2] += 1
53     counter1 = np.abs(counter1 - 0.5 * lens)
54     counter2 = np.abs(counter2 - 0.5 * lens)
55     counter = counter1 + counter2
56     k1, k3 = map(lambda x: x[0], np.where(counter == np.max(counter))) #
找到最大值的位置
57
58     key = format(k1, '04b') + format(k2, '04b') + format(k3, '04b') +
format(k4, '04b')
59     return key
60
61
62 def process_key(key):
63     '''
64     并行处理
65     param key: 密钥
66     '''
67     data = generate_data(T=T, key=key)
68     reverse_key = reverse_spn(data)
69     return key, reverse_key
70
71
72 def print_result(future):

```

```

73     '''
74     打印结果
75     '''
76     key, reverse_key = future.result()
77     msg = 'success' if key[16:] == reverse_key else 'fail'
78     key = ' '.join([key[i:i+4] for i in range(0, len(key), 4)])
79     reverse_key = ' '.join([reverse_key[i:i+4] for i in range(0,
len(reverse_key), 4)])
80     print(f'key = {key}\nreverse_key = {reverse_key}\nresult: {msg}\n')
81
82
83 if __name__ == '__main__':
84     keys = generate_key(n=N) # 生成密钥
85     with ProcessPoolExecutor() as executor: # 多进程处理
86         futures = [executor.submit(process_key, key) for key in keys]
87         for future in futures:
88             future.add_done_callback(print_result)

```

• 代码流程:

1. 初始化参数和S盒:

- 初始化S盒（代替置换表）。
- 计算逆S盒。
- 设置明文-密文对个数 **T** 和待攻击密钥个数 **N**。

2. 线性攻击算法 (**reverse_spn**):

- 首先，初始化一个计数器矩阵。
- 遍历所有的明文-密文对。
- 对于每个明文-密文对，遍历所有可能的L1和L2值，进行线性逼近。
- 根据计算的线性逼近统计信息，得到可能的k2和k4。
- 再次遍历所有的明文-密文对，使用已知的k2和k4，以及所有可能的L1和L2值来进行线性逼近。
- 根据新的线性逼近统计信息，得到可能的k1和k3。
- 最后，返回由k1, k2, k3, k4组成的密钥。

3. 主函数:

- 生成待攻击的密钥列表。
- 使用多进程并行处理每个密钥。
- 为每个进程任务添加回调函数以打印结果。

3 运行结果

我采用了随机生成密钥的方式，在本次我令 `n=50`，即生成50组密钥来攻击，但由于速度较慢，我使用了并行来加速，最终结果如下：

```
1 key = 0000 0101 1100 1111 0011 0111 1111 0110
2 reverse_key = 0011 0111 1111 1011
3 result: fail
4
5 key = 0101 0001 0010 1110 1001 1010 0011 1010
6 reverse_key = 1001 1010 0011 1010
7 result: success
8
9 key = 1000 0111 1100 1100 1001 0100 1101 1100
10 reverse_key = 1001 1001 1101 1100
11 result: fail
12
13 key = 0111 1001 1101 0111 1011 0001 0000 0011
14 reverse_key = 1011 0001 0000 0011
15 result: success
16
17 key = 0011 1001 1010 1101 1000 1010 0100 1000
18 reverse_key = 1000 1010 0100 1000
19 result: success
20
21 key = 1100 1111 1010 0100 1011 1000 1011 0010
22 reverse_key = 1011 0101 1011 0010
23 result: fail
24
25 key = 1100 1110 0011 0110 0011 0101 0000 0000
26 reverse_key = 0011 0101 0000 0000
27 result: success
28
29 key = 0101 0101 1000 0000 1100 1001 1110 1010
30 reverse_key = 1100 1001 1110 1010
31 result: success
32
33 key = 0010 1110 1000 0111 1011 0101 0001 1100
34 reverse_key = 1011 0101 0001 1100
35 result: success
36
37 key = 0101 0111 0101 0010 0010 1011 0110 1100
38 reverse_key = 0010 1011 0110 1100
39 result: success
40
41 key = 0001 0000 1101 1110 1111 0001 1100 1000
```

```
42 reverse_key = 1111 0001 1100 1000
43 result: success
44
45 key = 0101 1101 1100 1001 0100 0111 0110 0101
46 reverse_key = 0100 0111 0110 0101
47 result: success
48
49 key = 1111 0010 1010 1001 0011 0011 1110 0010
50 reverse_key = 0011 0011 1110 0010
51 result: success
52
53 key = 0101 1101 0101 1001 1111 1110 0011 0000
54 reverse_key = 1111 1110 0011 0000
55 result: success
56
57 key = 0001 1111 1111 0000 0101 0111 1111 0100
58 reverse_key = 0101 0111 1111 0100
59 result: success
60
61 key = 1111 1110 0011 1001 1101 1000 0011 0110
62 reverse_key = 1101 0100 0001 0110
63 result: fail
64
65 key = 0001 0111 1001 1110 1001 0101 1000 1001
66 reverse_key = 1001 0101 1000 1001
67 result: success
68
69 key = 1101 1100 1011 0011 0100 0100 0110 1001
70 reverse_key = 0100 0110 0110 1001
71 result: fail
72
73 key = 0110 0001 0011 1110 1010 1010 1001 1010
74 reverse_key = 1010 1010 1001 1000
75 result: fail
76
77 key = 0100 1101 1000 1100 1111 0000 1100 1111
78 reverse_key = 1111 0000 1100 1111
79 result: success
80
81 key = 0011 0000 0011 1100 1000 1111 1001 0111
82 reverse_key = 1000 1111 1001 0111
83 result: success
84
85 key = 1100 1111 1011 0011 1110 1001 1111 0111
86 reverse_key = 1100 1000 1111 0011
87 result: fail
```



```
88
89 key = 0100 1110 0000 0101 0111 0011 1000 1100
90 reverse_key = 0111 0011 1000 1100
91 result: success
92
93 key = 0101 1111 0010 0111 1111 1000 0001 0011
94 reverse_key = 1111 1000 0001 0001
95 result: fail
96
97 key = 1111 1101 0100 0100 1001 0100 0000 0101
98 reverse_key = 1001 0100 0000 0101
99 result: success
100
101 key = 1010 0101 0101 1000 1010 1011 0000 0011
102 reverse_key = 1010 1011 0000 0011
103 result: success
104
105 key = 0100 0011 1011 1000 1000 0001 0010 1001
106 reverse_key = 1000 0001 0010 1001
107 result: success
108
109 key = 0011 1010 0001 0110 1000 0011 0000 1111
110 reverse_key = 1000 0011 0000 1111
111 result: success
112
113 key = 0010 1110 1001 0011 0011 1011 1000 0011
114 reverse_key = 0011 1011 1000 0011
115 result: success
116
117 key = 1110 1111 0101 0110 1011 0000 1100 1111
118 reverse_key = 1011 0000 1100 1111
119 result: success
120
121 key = 0000 0010 1001 0010 0101 0011 1101 1100
122 reverse_key = 0101 1101 1101 0010
123 result: fail
124
125 key = 1010 0100 0101 1110 0101 1101 1010 1110
126 reverse_key = 0101 1101 1010 1110
127 result: success
128
129 key = 1101 1000 1010 0111 1001 1111 0011 0101
130 reverse_key = 0001 0010 0011 0101
131 result: fail
132
133 key = 1001 1000 1010 0010 1011 0101 0011 1000
```

```
134 reverse_key = 1011 0101 0011 1000
135 result: success
136
137 key = 0100 0011 1001 0000 0011 0001 1001 0101
138 reverse_key = 0011 0001 1001 0101
139 result: success
140
141 key = 1111 0000 1110 1011 0001 0001 0000 1011
142 reverse_key = 0001 0001 0000 1011
143 result: success
144
145 key = 1111 0011 1110 1100 0001 0101 1000 0110
146 reverse_key = 0001 1000 1000 0110
147 result: fail
148
149 key = 1011 1011 1100 0011 0011 1110 1011 0110
150 reverse_key = 0011 1000 1011 0011
151 result: fail
152
153 key = 1111 0001 1100 0000 0110 1101 1000 0101
154 reverse_key = 0110 1101 1000 0101
155 result: success
156
157 key = 1110 1100 0111 1101 1101 1100 0101 0011
158 reverse_key = 1101 1100 0101 0011
159 result: success
160
161 key = 1000 0101 0001 0000 0101 1110 0110 1001
162 reverse_key = 0101 1110 0110 1001
163 result: success
164
165 key = 1000 0100 1110 1100 1111 0000 0010 1010
166 reverse_key = 1111 0000 0010 1010
167 result: success
168
169 key = 1010 1101 1111 0111 0001 0001 0110 1001
170 reverse_key = 1101 1100 0111 0100
171 result: fail
172
173 key = 0100 1111 0111 1011 0011 0000 1011 0000
174 reverse_key = 0011 0000 1011 0000
175 result: success
176
177 key = 1110 1000 1001 0011 1010 0011 1100 0111
178 reverse_key = 1010 0011 1100 0111
179 result: success
```

```

180
181 key = 0110 1100 1111 1110 1110 1011 1010 0100
182 reverse_key = 1110 1011 1010 0100
183 result: success
184
185 key = 0010 1101 0101 1010 0001 0111 1110 1011
186 reverse_key = 0001 0111 1110 1011
187 result: success
188
189 key = 0000 1100 0010 1010 1000 1001 0000 1001
190 reverse_key = 1000 1010 0000 1001
191 result: fail
192
193 key = 1110 1110 1100 1011 0100 0000 0100 0001
194 reverse_key = 0100 0000 0100 0001
195 result: success
196
197 key = 0011 0001 1011 0110 0110 0001 0001 1001
198 reverse_key = 0110 0001 0001 1001
199 result: success

```

可以看到在50组密钥攻击中，其中36组密钥攻击成功，14组失败，准确率并不高，继续增加明文-密文对(即大于8000对)可以提高准确率。

因此令 `T = 10000`，再次运行结果如下：

```

1 key = 0101 1111 0100 0110 0011 1101 1110 0010
2 reverse_key = 0011 1101 1110 0010
3 result: success
4
5 key = 1011 1010 0011 1000 0111 1000 0111 0101
6 reverse_key = 0111 1000 0111 0101
7 result: success
8
9 key = 0110 0000 1010 1000 1100 1101 0110 0001
10 reverse_key = 1100 1101 0110 0001
11 result: success
12
13 key = 0110 0100 1111 1011 0110 0110 1010 1111
14 reverse_key = 0110 0110 1010 1111
15 result: success
16
17 key = 1011 1110 1010 1011 1001 0101 1101 1101
18 reverse_key = 1001 0101 1101 1110
19 result: fail
20

```

```
21 key = 1100 0100 1100 0000 0010 0010 0100 0100
22 reverse_key = 0010 0010 0100 0100
23 result: success
24
25 key = 0111 1100 1000 0000 0110 0001 0100 1101
26 reverse_key = 0110 0001 0100 1101
27 result: success
28
29 key = 0000 0101 0101 1110 0010 1101 1010 1100
30 reverse_key = 0010 1101 1010 1100
31 result: success
32
33 key = 0101 0101 0010 0101 1011 0110 1000 1101
34 reverse_key = 1011 0110 1000 1101
35 result: success
36
37 key = 0011 1011 1110 0100 1011 1111 1001 0001
38 reverse_key = 1011 1111 1001 0001
39 result: success
40
41 key = 1001 0110 1010 1110 1001 1000 1001 0011
42 reverse_key = 1001 1000 1001 0011
43 result: success
44
45 key = 0001 1110 0101 0100 1001 0101 1100 1100
46 reverse_key = 1001 0101 1100 1100
47 result: success
48
49 key = 1101 1111 1011 1001 1001 1101 1111 0001
50 reverse_key = 1001 1101 1111 1100
51 result: fail
52
53 key = 1111 1011 0110 0101 1101 0100 0010 1010
54 reverse_key = 1101 0100 0010 1010
55 result: success
56
57 key = 0101 0000 0001 0101 1011 1111 1001 0111
58 reverse_key = 1011 1111 1001 0111
59 result: success
60
61 key = 0100 1111 0100 1101 1101 0101 0110 1111
62 reverse_key = 1101 0101 0110 1111
63 result: success
64
65 key = 0010 0010 0110 1001 0010 0110 0110 1010
66 reverse_key = 0010 1011 0110 1010
```

```
67 result: fail
68
69 key = 1010 1010 1101 0100 0000 1001 0011 0010
70 reverse_key = 0000 1001 0011 0010
71 result: success
72
73 key = 1010 0101 1101 0100 1111 1101 1110 1100
74 reverse_key = 1111 1101 1110 1100
75 result: success
76
77 key = 1100 1111 1000 1011 0011 0011 0010 1010
78 reverse_key = 0011 0011 0010 1010
79 result: success
80
81 key = 0101 0000 1010 1100 0100 1000 1111 1010
82 reverse_key = 0100 1000 1111 1010
83 result: success
84
85 key = 0010 0100 0110 1110 1000 0111 1111 1100
86 reverse_key = 1000 1010 1111 0001
87 result: fail
88
89 key = 1111 1111 0001 0101 1101 1101 1100 0101
90 reverse_key = 1101 1101 1100 0101
91 result: success
92
93 key = 0010 0110 1110 0010 0111 1101 1101 1001
94 reverse_key = 0111 1101 1101 1001
95 result: success
96
97 key = 1100 1011 1000 0100 0000 1110 0001 1011
98 reverse_key = 0000 1110 0001 1011
99 result: success
100
101 key = 0001 0011 1110 0011 1111 0111 0011 0011
102 reverse_key = 1111 1001 0011 0011
103 result: fail
104
105 key = 1001 1100 0101 0111 1001 1111 1101 0110
106 reverse_key = 1001 1111 1101 0110
107 result: success
108
109 key = 1101 1001 0011 0000 0001 1001 1100 0011
110 reverse_key = 0001 1001 1100 1110
111 result: fail
112
```

```
113 key = 0110 0100 1011 1000 1001 1100 1100 1110
114 reverse_key = 1001 1100 1100 1110
115 result: success
116
117 key = 1101 0001 0000 1001 1100 0101 0001 1011
118 reverse_key = 1100 0101 0001 1011
119 result: success
120
121 key = 1100 1011 0000 1100 0011 1000 1111 1011
122 reverse_key = 0011 1000 1111 1011
123 result: success
124
125 key = 1111 1000 0101 1100 1111 0011 1000 0110
126 reverse_key = 1111 0011 1000 0110
127 result: success
128
129 key = 1010 0101 1011 1011 0000 0011 0010 1101
130 reverse_key = 0000 0011 0010 1101
131 result: success
132
133 key = 1011 1101 0010 1110 1110 1110 0000 1111
134 reverse_key = 1110 1110 0000 1111
135 result: success
136
137 key = 0010 0101 1001 1111 0011 0100 0000 1111
138 reverse_key = 0011 0100 0000 1111
139 result: success
140
141 key = 1000 0011 0001 0101 0010 1100 0100 0111
142 reverse_key = 0010 1100 0100 0111
143 result: success
144
145 key = 0000 1101 0011 0000 1101 0010 0100 1101
146 reverse_key = 1101 0010 0100 0000
147 result: fail
148
149 key = 0010 0111 0101 1000 1101 0001 1111 1100
150 reverse_key = 1101 0001 1111 1100
151 result: success
152
153 key = 0110 0101 0000 1010 1011 1100 0000 0000
154 reverse_key = 1011 1100 0000 0000
155 result: success
156
157 key = 1001 0110 0011 0000 1100 1011 1011 0001
158 reverse_key = 1100 1011 1011 0001
```

```
159 result: success
160
161 key = 1001 1101 0111 0010 1000 0011 0000 0111
162 reverse_key = 1000 0011 0000 0111
163 result: success
164
165 key = 0101 1000 1000 1001 1111 1101 1101 1001
166 reverse_key = 1111 1101 1101 1001
167 result: success
168
169 key = 1100 1001 0111 0100 0101 0000 1001 0011
170 reverse_key = 0101 0000 1001 0011
171 result: success
172
173 key = 0101 0010 0101 0011 0100 1110 1100 1000
174 reverse_key = 0100 1110 1100 1000
175 result: success
176
177 key = 1010 0100 1001 0000 1010 1100 1110 1001
178 reverse_key = 1010 1100 1110 1001
179 result: success
180
181 key = 1100 0010 1111 1011 1101 1010 1111 1010
182 reverse_key = 1101 1010 1111 1010
183 result: success
184
185 key = 0101 1111 0110 1010 1001 0100 1011 1101
186 reverse_key = 1001 0100 1011 1101
187 result: success
188
189 key = 1001 1101 0110 1111 0110 1111 1010 0110
190 reverse_key = 0110 1111 1010 0110
191 result: success
192
193 key = 0000 1001 1100 0100 1011 0010 0101 1101
194 reverse_key = 1011 1111 0101 1110
195 result: fail
196
197 key = 0100 0001 1010 1000 0000 0010 1110 1000
198 reverse_key = 0000 0010 1110 1000
199 result: success
```

可以看到50组成功42组，失败8组，准确率有所提升。