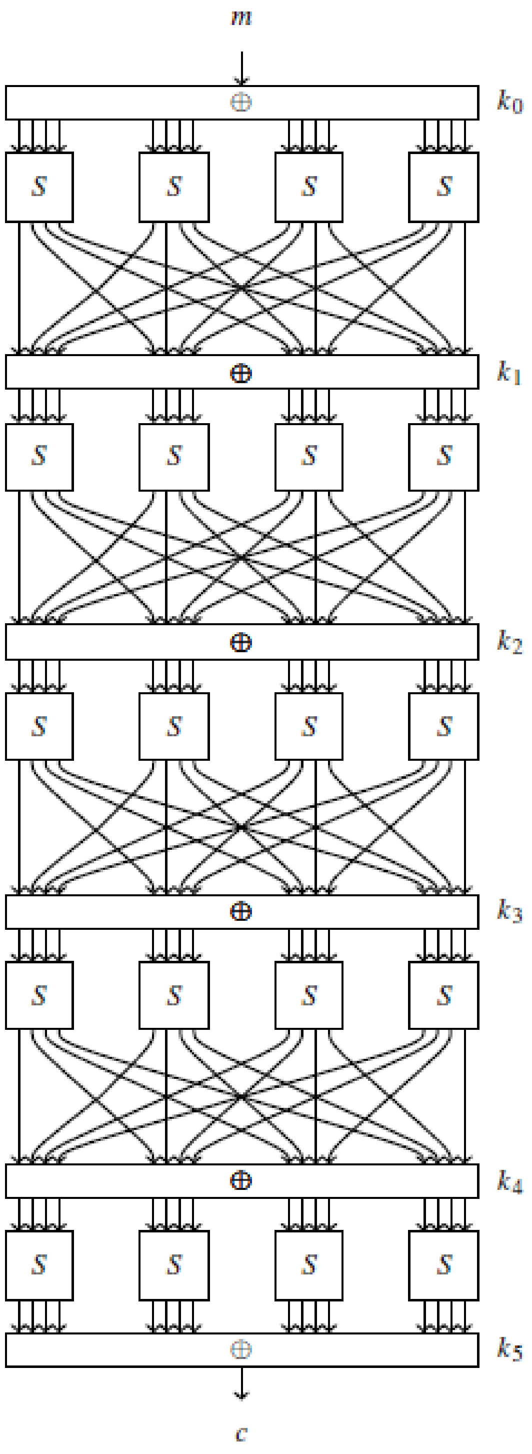
差分线性分析，目前的算法只画了5轮，可以加长，恢复部分密钥，看谁攻击的轮数长。自己设置密钥，自己生成数据。



| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S[x]$ | 6 | 4 | c | 5 | 0 | 7 | 2 | e | 1 | f | 3 | d | 8 | a | 9 | b |

差分线性分析，目前的算法只画了5轮，可以加长，恢复部分密钥，看谁攻击的轮数长。自己设置密钥，自己生成数据。

差分攻击过程：

根据 S 盒，计算出差分分布表如下：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 4 | 0 |
| 2 | 0 | 6 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 6 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 4 | 0 | 2 | 0 |
| 4 | 0 | 0 | 0 | 2 | 0 | 2 | 4 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 2 | 0 |
| 5 | 0 | 2 | 2 | 0 | 4 | 0 | 0 | 4 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 2 | 0 | 4 | 0 | 0 | 2 | 2 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 0 | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 4 | 0 | 0 | 4 | 0 | 2 | 0 | 2 |
| 9 | 0 | 2 | 0 | 0 | 0 | 2 | 2 | 2 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | 2 |
| a | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 4 | 4 | 0 | 2 | 2 | 0 | 0 |
| b | 0 | 0 | 0 | 2 | 2 | 0 | 2 | 2 | 2 | 0 | 0 | 4 | 0 | 0 | 2 | 0 |
| c | 0 | 4 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 6 | 0 |
| d | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 4 |
| e | 0 | 2 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 6 |
| f | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 2 |

为了**找到较高概率的差分特征，并尽量减少活跃 S 盒子的个数**，考虑到 P 置换的特性，我们发现(0020)这个差分经过 P 置换之后依旧是(0020)，而且恰好从差分分布表中可以看出(0020)→(0020)这个差分转移概率为$\frac{6}{16}$，所以对于这个 5 轮的加密算法来说，选择的差分特征为**(0020)→(0020)→(0020)→(0020)→(0020)→(0020)**，进而我们恢复的目标就是子密钥 $k_5$ 的第 9-12 位。

利用差分分析的步骤进行编程，可以顺利恢复出目标密钥的部分信息。具体的：先随机选择一组密钥[3, 2, **4,** 6], [3, 3, 15, 9], [3, 6, 14, 4], [0, 4, 13, 12], [6, 8, 2, 1], [11, 14, 4, 1]构成完整的加密算法。选择 100 组差分为(0020)的数据对，然后进行加密得到 100 条密文对，继续用差分进行分析，**得到 $k_5$ 的 9-12bit 为 4**。

然后我们增加加密的轮数，为了能够恢复密钥，我们需要相应的增加数据对的数量。经过不断增加数据和增加迭代轮数，**最终能够用 10000 条数据攻击出 20 轮加密体制的最后一轮密钥 $k_{20}$ 的 9-12bit 的信息。**

python 代码:

```python
import encrypto
import numpy as np
import random

#  建立差分分布表
def different_table():
    table = list(np.zeros((16, 16), dtype=np.int))
    for deta_in in range(16):
        for x in range(16):
                deta_out = (encrypto.S_box([x ^ deta_in])[0]) ^ (encrypto.S_box([x])[0])
                table[deta_in][deta_out] += 1
    return table

#S 盒逆，C 是密文串
def S_inv(c):
    l = [4, 8, 6, 0xa, 1, 3, 0, 5, 0xc, 0xe, 0xd, 0xf, 2, 0xb, 7, 9]
    for i in range(len(c)):
        c[i] = l[c[i]]
    return c


#  差分攻击，t 对数据对，(alpha,beta)是前四轮的差分器，candicate 是密文可能的差分，  恢复 k5 的第三 bit
def defferencial_attack(t, alpha, beta, candicate, key):
    # key = [[15, 8, 1, 0], [4, 1, 13, 10], [10, 8, 7, 14], [14, 14, 9, 0], [7, 12, 15, 6], [15, 15, 4, 4], [13, 8, 6, 12]]
    #随机产生 t 组数据对
    m_pairs = []
    c_pairs = []
    for k in range(t):
        m1 = []
        for i in range(4):
            a = random.randint(0, 15)
            m1.append(a)
        m2 = encrypto.XOR(m1, alpha) #  构造满足输入差分的明文对
        m_pairs.append(tuple([m1, m2]))

        c1 = encrypto.block_cipher(m1, key)
        c2 = encrypto.block_cipher(m2, key)
        #过滤
        deta_out = encrypto.XOR(c1, c2)
        if deta_out in candicate:
            c_pairs.append(tuple([c1, c2]))
    #恢复 key 的第 9-12bit
    key_3 = np.zeros(16, dtype=np.int)
    for k in range(16):
        for c_pair in c_pairs:
            tempc1 = c_pair[0][2] ^ k
            tempc2 = c_pair[1][2] ^ k
            if ((S_inv([tempc1])[0]) ^ (S_inv([tempc2])[0])) == beta[2]:
                key_3[k] += 1
    max_value = max(key_3)
    max_index = []
    for i in range(len(key_3)):
        if max_value == key_3[i]:
            max_index.append(i)

    return m_pairs, c_pairs, key_3, max_index


if __name__ == '__main__':
    N = 20000 #输入明文对
    n = 20 #密码迭代轮数
    alpha = beta = [0, 0, 2, 0]
    candicate = [[0, 0, 1, 0], [0, 0, 2, 0], [0, 0, 9, 0], [0, 0, 10, 0]]
    key = []    # 6 个密钥，十六进制
    for i in range(n+1):
        l = []
        for j in range(4):
            l.append(random.randint(0, 15))
        key.append(l)
    print("target keys:", key[n])
    print("total key:",key)
    count_after_filer = 0
    itera_num = 0
    while(count_after_filer < 5 and itera_num < 20):
        m_pairs, c_pairs, key_3, max_index = \
            defferencial_attack(t=N, alpha=alpha, beta=beta, candicate=candicate,
key=key)
        count_after_filer = len(c_pairs)
        itera_num += 1

    print(len(m_pairs), "textplain:", m_pairs)
    print("cipher text:", c_pairs)
    print("count after filter:", count_after_filer)
    print("Vote:", key_3)
    print("iterate times:", itera_num)
    print("key information:", max_index)


    # table = different_table()
    # for i in range(len(table)):
    #     print(table[i])

# Encrypto algorithm
import random
```

```
#Sbox---input(m),output(m),m is a list with 4 hexadecimal number
def S_box(m):
    l = [6, 4, 0xc, 5, 0, 7, 2, 0xe, 1, 0xf, 3, 0xd, 8, 0xa, 9, 0xb]
    # l = [0,3,5,8,6,9,0xc,7,0xd,0xa,0xe,4,1,0xf,0xb,2]
    for i in range(len(m)):
        m[i] = l[m[i]]
    return m



#将行如 0011 的字符串转成整数
def stobin(s):
    res = 0
    for i in range(len(s)-1):
        res = (res + int(s[i]))*2
    res = res + int(s[i+1])
    return res

def P_box(m):
    s=[]
    for i in range(len(m)): #m[i] is a hexadecimal nuber
        s1 = str(bin(m[i]))[2:]
        s2 = ''
        #给高位补 0
        for j in range(4-len(s1)):
            s2 = s2 + '0'
        s1 = s2 + s1
        s.append(s1)
    res = []
    for j in range(len(s[0])):
        s3 = ''
        for i in range(len(s)):
            s3 = s3 + s[i][j]
        s3 = stobin(s3)
        res.append(s3)
    return res


#输入两个四维列表，包括 4 个 16 进制数,m 和 ki 异或
def XOR(m, ki):
    res = []
    for i in range(len(m)):
        res.append(m[i]^ki[i])
    return res

#直接加长密钥个数，就可以增加轮数
def block_cipher(m, k):
    n = len(k)
    for i in range(n-2):
        m = XOR(m, k[i])
        # print("第%d 轮密钥异或值:"%(i+1), m)
        m = S_box(m) #S 盒变换
        # print("第%d 轮 S 盒输出:"%(i+1), m)
        m = P_box(m) #P 变换
        # print("第%d 轮 P 置换的输出:"%(i+1), m)
    i += 1
    m = XOR(m, k[i])
    # print("第%d 轮密钥异或值:"%(i+1), m)
    m = S_box(m) #S 盒变换
    # print("第%d 轮 S 盒输出:"%(i+1), m)
    i += 1
    return XOR(m, k[i])

if __name__ == '__main__':
    n = 7 #密钥个数
    width = 4

    m = [] #4 个十六进制的数
    for i in range(width):
        a = random.randint(0,15)
        m.append(a)
    print("textplain:",m)

    k = [] #6 个密钥，十六进制
    for i in range(n):
        l = []
        for j in range(width):
            l.append(random.randint(0,15))
        k.append(l)
    print("keys:",k)
    print(block_cipher(m,k))
```