**置换函数**：

def permute(input\_str, table):  
 output\_str = ""  
 for bit\_position in table:  
 output\_str += input\_str[bit\_position - 1]  
 return output\_str

该函数根据table中指定的位置重新排列input\_str中的字符，并返回一个新的字符串。

**循环左移函数**：

def ls(key, n):  
 # 将密钥分成两段并循环左移 n 位  
 left\_half = key[:5]  
 right\_half = key[5:]  
 shifted\_left = left\_half[n:] + left\_half[:n]  
 shifted\_right = right\_half[n:] + right\_half[:n]  
 return shifted\_left + shifted\_right

该函数将"key"分成两半，并对每半进行左循环移位，移动"n"个位置。然后将移位后的两半连接起来并返回结果。

**子密钥生成：**

def generate\_key(k, p10\_table, p8\_table):  
 # 执行 P10 置换  
 p10\_key = permute(k, p10\_table)  
 # 对结果进行左移操作和P8置换，得到 K1  
 k1 = permute(ls(p10\_key, 1), p8\_table)  
 # 再次对上一步结果进行左移操作h和P8置换，得到 K2  
 k2 = permute(ls(ls(p10\_key, 1), 2), p8\_table)  
 return k1, k2

使用给定的密钥k和两个置换表p10\_table和p8\_table生成两个子密钥k1和k2。

1. **DES的F函数：**

def F(right\_half, k):  
 # 对右半部分进行 E/P 扩展置换  
 expanded = permute(right\_half, ep\_table)  
 # 对结果与 K1 进行异或操作  
 xored = '{0:08b}'.format(int(expanded, 2) ^ int(k, 2))  
 # 将结果分为两组，并根据 S-box 进行替换  
 s0\_input = xored[:4]  
 s1\_input = xored[4:]  
 # 根据S盒规则行列查找  
 s0\_row = int(s0\_input[0] + s0\_input[-1], 2)  
 s0\_col = int(s0\_input[1:-1], 2)  
 s1\_row = int(s1\_input[0] + s1\_input[-1], 2)  
 s1\_col = int(s1\_input[1:-1], 2)  
 s0\_output = '{0:02b}'.format(sbox0[s0\_row][s0\_col])  
 s1\_output = '{0:02b}'.format(sbox1[s1\_row][s1\_col])  
 # 对两个输出串进行 P4 置换得到最终结果  
 s\_output = s0\_output + s1\_output  
 return permute(s\_output, p4\_table)

接受一个right\_half输入，执行E/P扩展置换，将结果与k值进行异或运算，执行S盒替换，并通过最终的P4置换生成输出。

**加密过程：**

def encrypt(p, k1, k2):  
 # 执行初始置换  
 p = permute(p, ip\_table)  
 # 进行两轮 Feistel 加密  
 l0 = p[:4]  
 r0 = p[4:]  
 l1 = r0  
 # 第一轮的P4  
 f\_result = F(r0, k1)  
 # p41和L0异或  
 r1 = '{0:04b}'.format(int(l0, 2) ^ int(f\_result, 2))  
 # 第二轮的P4  
 f\_result = F(r1, k2)  
 # p42和L1异或  
 r2 = '{0:04b}'.format(int(l1, 2) ^ int(f\_result, 2))  
 # 逆置换并返回结果(左边R2右边R1)  
 return permute(r2 + r1, ip\_ni\_table)

1. 函数 encrypt 接受三个参数： p ， k1 和 k2 。其中， p 表示要加密的明文， k1 和 k2 是两个加密密钥。

2. 使用 permute 函数和 ip\_table 参数对明文 p 进行初始置换。

3. 将明文 p 分为两半：左半部分 l0 （前4位）和右半部分 r0 （后4位）。

4. 将右半部分 r0 作为下一轮加密的左半部分 l1 。

5. 使用密钥 k1 对右半部分 r0 应用 F 函数，得到结果 f\_result 。

6. 将左半部分 l0 转换为整数，并与转换为整数的 f\_result 进行异或运算。将结果转换回长度为4的二进制字符串，并赋值给 r1 。

7. 使用密钥 k2 对新的右半部分 r1 应用 F 函数，得到结果 f\_result 。

8. 将左半部分 l1 转换为整数，并与转换为整数的 f\_result 进行异或运算。将结果转换回长度为4的二进制字符串，并赋值给 r2 。

9. 通过连接 r2 和 r1 ，并使用 ip\_ni\_table 进行逆置换，得到最终的密文。

10. 将得到的密文作为 encrypt 函数的输出返回。

**主函数：**

s = ''  
K1, K2 = generate\_key(entry\_content2, p10\_table, p8\_table)  
# 对明文进行加密  
#提取单个字符转换成ACSII码后连接成新的字符串  
for i in entry\_content1:  
 st = str.zfill(bin(ord(i))[2:],8)  
 sr = encrypt(st, K1, K2)  
 ss = chr(eval('0b'+str(sr)))  
 s = s + ss

**暴力破解过程：**

#加密  
def enc(code, key):  
 s = ''  
 K1, K2 = main.generate\_key(key, p10\_table, p8\_table)  
 for i in code:  
 st = str.zfill(bin(ord(i))[2:], 8)  
 sr = main.encrypt(st, K1, K2)  
 ss = chr(eval('0b'+str(sr)))  
 s = s + ss  
 return s  
  
#循环暴力破解  
t1 = time.time()  
for i in range(1024):  
  
 k = str.zfill(bin(i)[2:], 10)  
 flag = 0  
 for text in range(len(cleartext)):  
 if enc(cleartext[text], k) != ciphertext[text]:  
 flag = 1  
 break  
 if flag == 0:  
 print('密钥为:', k)  
 t2 = time.time()  
 print('耗时:', t2-t1)