



山东大学  
SHANDONG UNIVERSITY

## 实验三：不可能差分与线性分析

组员：刘舒畅，李昕，林宗茂

2023 年 10 月 22 日

成员信息及完成部分：

刘舒畅：202122460175 task1、4 原理分析/代码编写，报告校对

李昕：202100460065 task2、3 原理分析/代码编写，报告编写

林宗茂：202100460128 代码测试及报告编写

---

## 摘 要

在本次实验中,我们首先利用 U 算法分析并推导了 CLEFIA-128 的 9 轮不可能差分,得到了两条 9 轮不可能差分路线,分别是  $[0,0,0,1^*] \rightarrow [0,0,0,1^*]$  和  $[0,1^*,0,0] \rightarrow [0,1^*,0,0]$ 。

其次,我们根据老师给出的范式,手动推导了 des 第 1 个 S 盒三轮近似式  $P_l[31] \oplus P_h[23, 15, 9, 1] \oplus C_l[31] \oplus C_h[23, 15, 9, 1] = K_1[46] \oplus K_3[46]$  和对应的五轮近似式  $P_h[31] \oplus P_l[20, 17, 16, 23, 15, 9, 1] \oplus C_h[31] \oplus C_l[20, 17, 16, 23, 15, 9, 1] = K_1[29, 26, 25] \oplus K_2[46] \oplus K_4[46] \oplus K_5[29, 26, 25]$ , 其概率约为 0.5154, 同时,我们还利用代码,推导出了对于 8 个 S 盒偏差最大/次大/第三大的三轮近似式,探讨了其是否可以构造五轮近似式并构造了五轮近似式。

最后,我们讨论了利用五轮线性近似式恢复 DES 第六轮轮密钥所需要的必要条件,利用算法二尝试恢复了 DES 第六轮 4 个 S 盒共计 24bit 的轮密钥,结果为 0Xex0a00000170, 当我们选取选取  $\lceil |\varepsilon|^{-2} \rceil$  对明密文(约等于 4190 对)重复 100 次实验后,验证成功恢复的概率为 0.11。

**关键词:** CLEFIA-128 DES 不可能差分 线性分析

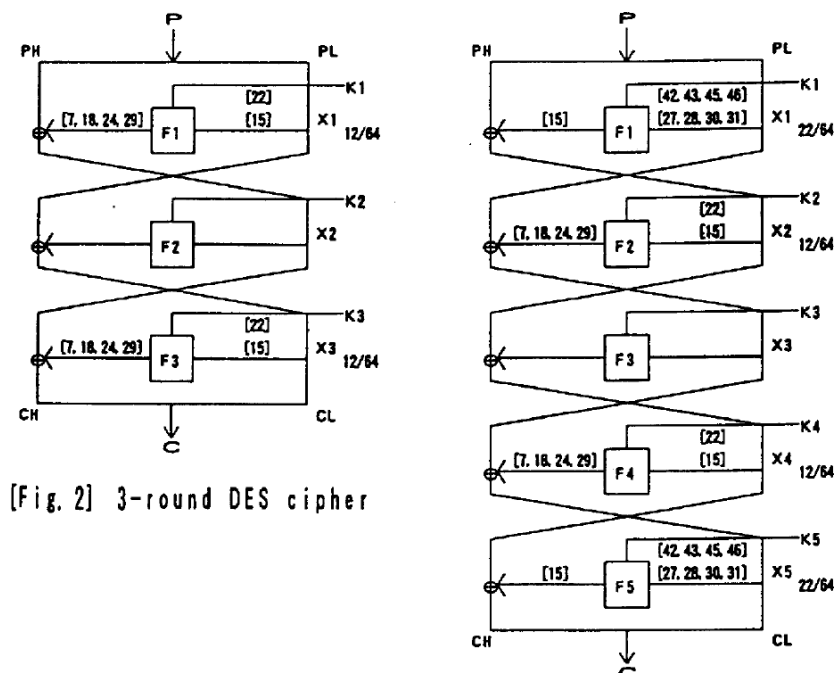
# 目录

1	问题重述	1
2	实验准备	1
2.1	利用 $\mu$ 方法寻找 CLEFIA-128 不可能差分 . . . . .	1
2.1.1	不可能差分原理简述 . . . . .	1
2.1.2	CLEFIA-128 算法 . . . . .	2
2.1.3	利用 $\mu$ 方法寻找不可能差分 . . . . .	3
2.2	手动构造 3 轮 DES 算法线性路线 . . . . .	4
2.3	在三轮近似式基础上构造五轮近似式 . . . . .	5
2.4	利用 Matsui 算法 2 实现 6 轮 DES 算法密钥恢复攻击 . . . . .	5
3	实验过程	6
3.1	利用 $\mu$ 方法寻找 CLEFIA-128 不可能差分 . . . . .	6
3.2	编程构造 3 轮 DES 算法线性路线 . . . . .	10
3.3	在三轮近似式基础上构造五轮近似式 . . . . .	12
3.4	利用 Matsui 算法 2 实现 6 轮 DES 算法密钥恢复攻击 . . . . .	16
3.4.1	确定待恢复轮密钥位置 . . . . .	16
3.4.2	利用 Matsui 算法 2 进行线性分析 . . . . .	19
A	附录	24
A.1	寻找 CLEFIA-128 不可能差分主要代码 . . . . .	24
A.2	寻找 DES 三轮线性近似式和五轮线性近似式代码 . . . . .	27
A.3	DES 六轮算法 2 线性分析代码 . . . . .	33

# 1 问题重述

**问题一** 根据分组密码算法 CLEFIA-128 或杂凑函数 Whirlpool 算法中分组密码的结构（任选其一即可），利用 U 算法（或手工推导等其他方式）尝试寻找不可能差分。

**问题二** 仿照以下 3 轮 DES 算法（左图，不考虑初始 IP 置换和最后的 IP 逆置换，最后一轮左右不交换）以及给出的迭代线性近似式，结合 S 盒的线性分布表（任一 S 盒均可），手动推导一条新的 3 轮迭代差分路线（与以下示例不同的），并计算相应概率。



[Fig. 2] 3-round DES cipher

**问题三** 在 2) 新找到的近似式的基础上，仿照问题二中右图，构造 5 轮线性近似式，并计算相应概率

**问题四** 在 3) 的基础上，利用 Matsui 算法 2（课上讲），给出 6 轮 DES 算法的密钥恢复攻击（仅恢复第 6 轮的部分轮密钥即可），重复 100 次实验，测试成功率（可能低于 50

## 2 实验准备

### 2.1 利用 $\mu$ 方法寻找 CLEFIA-128 不可能差分

#### 2.1.1 不可能差分原理简述

**寻找密码算法中的不可能差分** 根据密码算法的结构和特性，找到两个输入明文 M1 和 M2，使得经过算法的若干轮变换后，输出结果的差分  $\Delta X$  满足某个不可能的条件（例

---

如差分  $\Delta X$  必须为 0, 但根据差分传播特性这在实际中不可能发生)。在实际运用中, 可以从加密或解密方向推导输入和输出差分在若干轮加(解)密运算后的变化情况, 从中发现矛盾, 构造区分器。

**构造部分轮密码** 将导致不可能差分  $\Delta X$  的算法轮函数构造成一个部分轮的加密结构。一般取使得不可能差分产生的最小轮数, 以提高效率。

**采样** 根据不可能差分条件, 选择符合条件的一对明文  $M1$  和  $M2$  作为攻击的输入。

**去噪** 对每个可能的密钥  $K$ , 用其加密选择的明文对, 检查结果是否满足不可能差分。如果满足, 则排除该密钥。

**恢复密钥** 通过交叉验证和分析未被排除的密钥, 最终找到真正的密钥。

### 2.1.2 CLEFIA-128 算法

CLEFIA 是一个 128 位的分组密码, 其密钥长度分别为 128、192 和 256 位, 与 AES 兼容。CLEFIA 由两个部分组成: 数据处理部分和关键调度部分。CLEFIA 采用了一个具有四路并行的广义 Feistel 结构, 每条路线线的宽度为 32 位。此外, 在密码的开始和结束处都有关键的白化部分。对于 128 位、192 位和 256 位的密钥, CLEFIA 的轮数分别为 18、22 和 26 次。

**算法流程** CLEFIA-128 整体操作结构是一个 Feistel 网络, 包含了 18 轮的迭代过程。进行完所有的轮操作后, 最后在输出之前, 还有一次密钥混合的过程。

**子密钥生成算法** 输入为 128 位密钥, 用密钥调度算法产生子密钥, 在整个加密流程中需要 56 个子密钥。

**Feistel 轮函数** 每个 Feistel 轮中, 数据的左右两半部分分别进行操作, 右半部分先进行一次  $F$  函数变换, 然后与左半部分做异或操作, 再将左右两部分交换。 $F$  函数包括一系列的  $S$  盒与扩散部件。CLEFIA 有两个 8 位的  $S$  盒,  $S0$  和  $S1$ , 这两个  $S$  盒协同工作并提供了良好的密码属性。

**广义 Feistel 结构** (Generalized Feistel Network, GFN) GFN 是 Feistel 网络的一个扩展。在每一轮中, CLEFIA 的  $F$  函数使用 GFN 增强了混淆行和扩散性。在 CLEFIA 中, 广义 Feistel 结构包括 4 部分:  $F0$ ,  $FO1$ ,  $F1$ , 和  $F01$ 。这 4 部分组合一起, 提供了强大的安全性。

**扩散矩阵** CLEFIA 算法使用了两个扩散矩阵  $M0$ ,  $M1$ 。这两个矩阵与  $F$  函数一起使用, 完成了各位之间的扩散转换。

**多轮迭代** 通过 18 轮运算, 加密过程充分混淆了数据和密钥, 以达到高密度的混淆扩散乘效应。

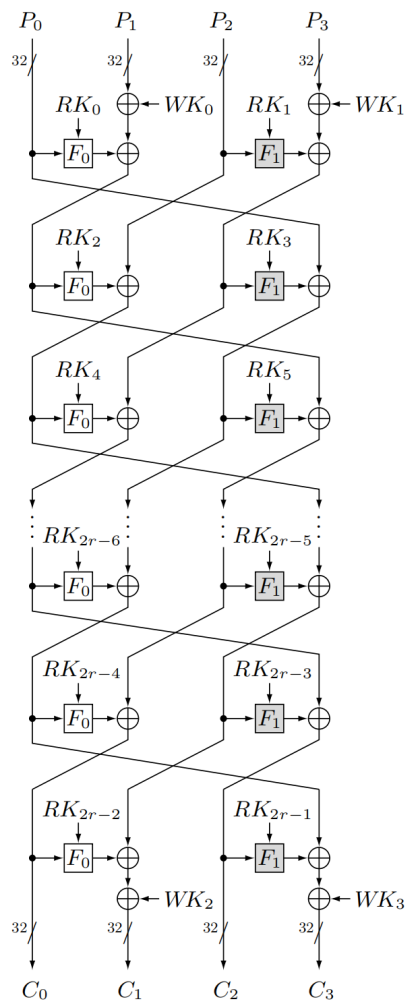


Figure 1: CLEFIA-128 算法结构

### 2.1.3 利用 $\mu$ 方法寻找不可能差分

$\mu$  方法通过以下表格中的符号代表 CLEFIA 算法内部的各种差分情况

差分情况	含义
0	0 差分
1	非 0 不确定差分
1*	非 0 确定差分
2*	非 0 确定差分 $\oplus$ 非 0 不确定差分
t(2)	不确定差分

在实际情况中会发生冲突的情况如下表

差分情况	与之相冲突的差分情况
0	1, 1*
1	0
1*	0, 1*, 2*
2*	1*

接下来分析如何寻找不可能差分。首先遍历各种输入情况，从头部开始加密、尾部开始解密各若干轮直至所有参数均为't'（此处选为 10 轮），遍历加密和解密的各个轮次，检验是否出现冲突，若出现冲突，则发现了一条不可能差分路线。

## 2.2 手动构造 3 轮 DES 算法线性路线

分析实验文档给出的一轮线性近似式  $X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22]$ ，该线性式只涉及一轮 S 盒，并通过该近似式构造三轮近似式，首先构造一轮近似式，根据线性分布表和 DES 轮函数拉线结构，找到以下关系（以实验文档给出的范例如为例子）：

$$\begin{cases} \langle \alpha, E(R_0) \oplus K_1 \rangle = \langle \beta, S_5(E(R_0) \oplus K_1) \rangle \quad (\alpha = 16, \beta = 15) \\ (E(R_0) \oplus K_1)[22] = S_5(E(R_0) \oplus K_1) = F(R_0, K_1)[7, 18, 24, 29] \\ R_2 = L_0 \oplus F(R_0, K_1) \end{cases} \quad (1)$$

由上式得到实验文档给出的一轮线性近似式，同时写出格式完全相同的第三轮线性近似式：

$$\begin{cases} X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22] \\ X_2[7, 18, 24, 29] \oplus C_H[7, 18, 24, 29] \oplus C_L[15] = K_3[22] \end{cases} \quad (2)$$

两者异或，得到三轮逼近式：

$$P_l[15] \oplus P_h[24, 18, 7, 29] \oplus C_l[15] \oplus C_h[24, 18, 7, 29] = K_1[22] \oplus K_3[22]$$

该步骤的重点在于找到偏差足够高的 (1) 式，通过查询 8 个 S 盒的线性分布表，可以找到每个分布表中偏差最大的一项，以此构造 (1) 式，并通过其推导三轮迭代线性路线。

下面我们手动推导第一个 S 盒线性近似式：

首先构造一轮近似式，根据线性分布表和 DES 轮函数拉线结构，找到以下关系

$$\begin{cases} \langle \alpha, E(R_0) \oplus K_1 \rangle = \langle \beta, S_1(E(R_0) \oplus K_1) \rangle \quad (\alpha = 16, \beta = 15) \\ (E(R_0) \oplus K_1)[46] = S_1(E(R_0) \oplus K_1) = F(R_0, K_1)[23, 15, 9, 1] \\ R_2 = L_0 \oplus F(R_0, K_1) \end{cases} \quad (3)$$

由上式得到实验文档给出的一轮线性近似式，同时写出格式完全相同的第三轮线性近似式：

$$\begin{cases} X_2[23, 15, 9, 1] \oplus P_H[23, 15, 9, 1] \oplus P_L[31] = K_1[46] \\ X_2[23, 15, 9, 1] \oplus C_H[23, 15, 9, 1] \oplus C_L[31] = K_3[46] \end{cases} \quad (4)$$

两者异或，得到三轮逼近式，其概率为  $\frac{14}{64}$ ：

$$P_l[31] \oplus P_h[23, 15, 9, 1] \oplus C_l[31] \oplus C_h[23, 15, 9, 1] = K_1[46] \oplus K_3[46]$$

实验文档只要求手动推导一条线性近似式，但是，不是所有三轮线性近似式都可以构造 task3 中的五轮线性近似式，同时，如果可以找到最优的 5 轮线性近似式，就能降低 task4 密码分析的复杂度和所需明密文数量，故在本报告 3.2 中，我们利用代码枚举出了 8 个 S 盒对应的偏差最大/次大/第三大的三轮线性相关路线，并针对上述问题进行了讨论。

## 2.3 在三轮近似式基础上构造五轮近似式

观察实验文档中给出的五轮线性近似式，其构造方式为利用已有的三轮线性近似式，在头和尾添加能够连接的一轮近似式，使新的第一轮  $P_h$  和原三轮式子中第一轮的  $P_l$  对应的比特位相同（最后一轮同理），并使新添加的第一轮和最后一轮的偏差在该条件下最大，以实验文档中的范例为例，固定新添加第一轮的  $P_h$  比特为 [15]，反推出其位于第一个 S 盒，寻找第一个 S 盒线性分布表中对应概率最大的一个  $\alpha$ ，即可构造出 5 轮线性近似式。

在本报告 3.3 中，我们利用代码，枚举出了 8 个 S 盒对应的偏差最大的三轮线性相关路线所对应的相对偏差最大的五轮线性近似式，并利用堆积引理计算其总概率。

## 2.4 利用 Matsui 算法 2 实现 6 轮 DES 算法密钥恢复攻击

要利用 Matsui 算法 2 实现 6 轮 DES 算法密钥恢复攻击，首先要选择一条合适的五轮线性近似式，选择的标准有两条，一是需要偏差尽可能的高，从而降低需要的明密文对数；而是涉及的 S 盒数量要适中，若涉及过多，则需要枚举的比特过多，复杂度过高，



若涉及 S 盒太少，则只能恢复较少的轮密钥信息。综合对比下，我们选择了在 2.2 中手动推导的三轮近似式  $P_l[31] \oplus P_h[23, 15, 9, 1] \oplus C_l[31] \oplus C_h[23, 15, 9, 1] = K_1[46] \oplus K_3[46]$  和其导出的五轮近似式  $X_2[31] \oplus P_H[31] \oplus P_L[20, 17, 16] = K_1[29, 26, 25]$ ，该式概率为 0.5154，并且涉及 4 个 S 盒，可以作为线性攻击的路线。

为了降低搜索密钥的复杂度，我们使用了老师提供的 32 位主密钥信息，利用密钥扩展算法算出了其对应第六轮轮密钥信息，在线性近似式涉及的 4 个活跃 S 盒在该情况下有 9bit 未知，作为枚举的对象。

同时，利用 Matsui 算法 2，在五轮加密的后面增加第六轮，先猜测第 6 轮轮密钥，进行一轮解密，得到中间状态，再利用五轮线性近似式进行验证，由于五轮近似式限定了其输出的部分比特，故只能在第六轮中恢复这些比特对应的 S 盒轮密钥（4 个 S 盒，24bit）。

## 3 实验过程

### 3.1 利用 $\mu$ 方法寻找 CLEFIA-128 不可能差分

使用 python 实现查找不可能差分。首先定义  $\mu$  方法中的加法、乘法、矩阵乘法以及加密、解密函数。

```
1 class U_way:
2     def __init__(self,E,D) -> None:
3         self.E = E
4         self.D = D
5
6     def mul(self,x,y):
7         if y == '1':
8             return x
9         elif y == '0':
10            return '0'
11        elif x == '1*':
12            return '1'
13        elif x == '2*':
14            return 't'
15        else:
16            return x
17
```

```

18  def add(self,x,y):
19
20      if x == '0':
21          return y
22      elif y == '0':
23          return x
24
25      elif ((x == '1') & (y == '1*')) | ((x == '1*') & (y == '1')):
26          return '2*'
27
28      else:
29          return 't'
30
31  def mat_mul(self,X,M):
32      res = ['0']*len(M)
33      for i in range(len(M)):
34          for j in range(len(M[i])):
35              tmp = self.mul(X[j],M[i][j])
36              res[i] = self.add(res[i],tmp)
37      return res
38
39  def enc(self,P,r) -> list:
40      l = []
41      for _ in range(r):
42          P = self.mat_mul(P,self.E)
43          l.append(P)
44      return l
45
46  def dec(self,C,r):
47      l = []
48      for _ in range(r):
49          C = self.mat_mul(C,self.D)
50          l.append(C)
51      return l

```

根据 CLEFIA-128 算法结构定义  $\mu$  方法中加密、解密时论函数对应的矩阵 E、D，每一行对应矩阵中的一列（即代码中的矩阵是实际矩阵的转置）。

```

1 E = [['f','1','0','0'],#加密时, 输出0为 F(p0) XOR p1
2      ['0','0','1','0'],#输出1为 p2
3      ['0','0','f','1'],#输出2为 F(p2) XOR p3
4      ['1','0','0','0']]#输出3为 p0
5
6 D = [['0','0','0','1'],#解密时, 输出0为 p3
7      ['1','0','0','f'],#输出1为 p0 XOR F(p3)
8      ['0','1','0','0'],#输出2为 p1
9      ['0','f','1','0']]#输出3为 F(p1) XOR p2

```

根据实验准备中提到的冲突情况, 定义判断是否产生冲突的检测函数

```

1 def check(x,y):
2     for i in range(4):
3         if(x[i] == '0'):
4             if (y[i] == '1')|(y[i] == '1*'):
5                 return 0
6         elif x[i] == '1':
7             if y[i] == '0':
8                 return 0
9         elif x[i] == '1*':
10            if (y[i] != '1')&(y[i] != 't'):
11                return 0
12        elif x[i] == '2*':
13            if y[i] == '1*':
14                return 0
15        else:
16            return 1

```

利用上述函数, 对 10 轮加密中的每一轮结果, 遍历 10 轮解密结果, 检测是否产生冲突, 若产生冲突则产生结果, 代码如下

```

1 for i in range(len(pi)):
2     for j in range(len(ci)):
3         p = pi[i]
4         c = ci[j]

```

```

5     #轮数
6     maxn = 0
7     pl0,cl0 = 0,0
8     for pl in range(10):
9         for cl in range(10):
10            if(check(p[pl],c[cl])==0) & (maxn < pl+cl+3):
11                pl0 = pl
12                cl0 = cl
13                maxn = pl+cl+2
14        if(maxn >= 9):
15            print(maxn,pc[i+1],reshape(pc[j+1]),pl0,cl0,p[pl0],c[cl0])

```

完整代码见附件。代码运行结果如下，输出结果从左往右依次是路线轮数，头部差分情况，尾部差分情况，加密轮数，解密轮数，加密后的差分情况，解密后的差分情况。

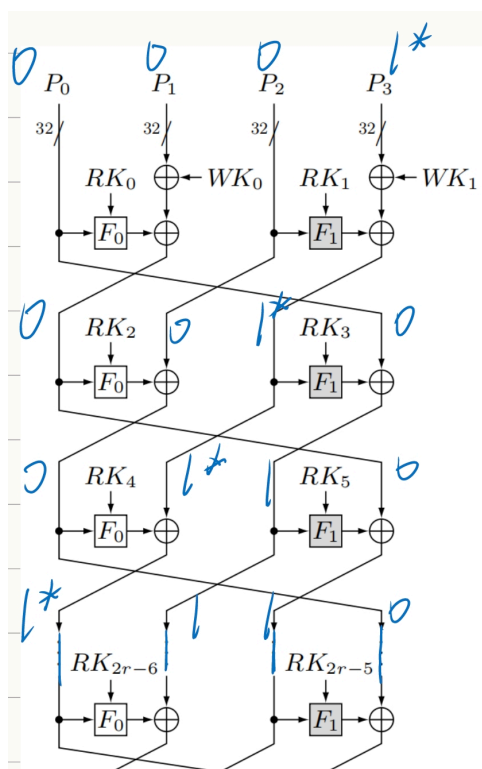
```

9  ['0', '0', '0', '1*'] ['0', '0', '0', '1*'] 2 5 ['1*', '1', '1', '0'] ['2*', 't', 't', 't']
9  ['0', '1*', '0', '0'] ['0', '1*', '0', '0'] 4 3 ['2*', 't', 't', '1'] ['1*', 't', '1', '1']

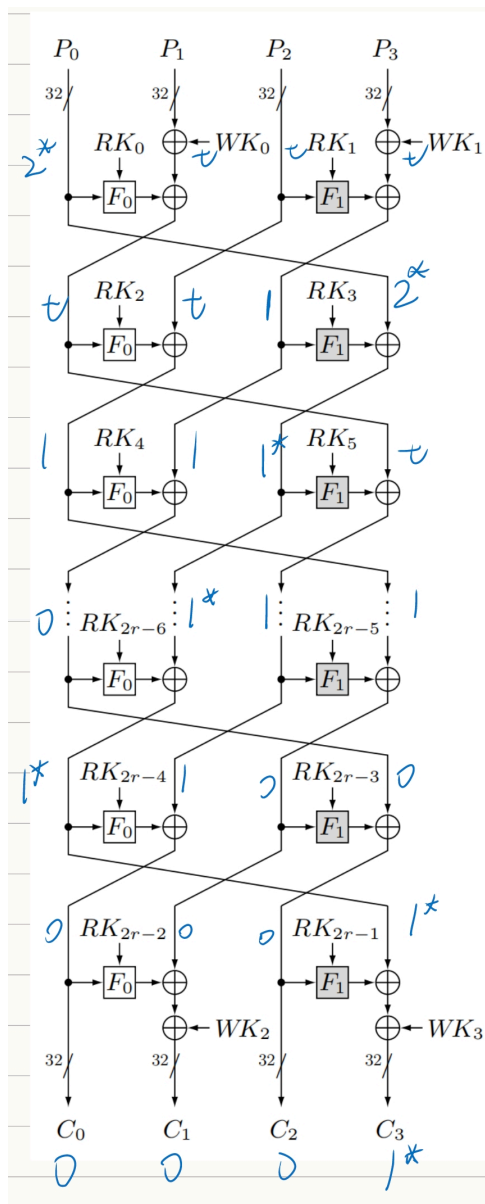
```

根据程序运行结果,我们得到了两条 9 轮不可能差分路线,分别是  $[0,0,0,1^*] \rightarrow [0,0,0,1^*]$  和  $[0,1^*,0,0] \rightarrow [0,1^*,0,0]$ , 再根据该结果手工推导验证。

加密三轮后的手工验算结果如下



解密六轮后的手工验算结果如下



手工验算结果与程序运行结果相同, 第一条不可能差分路线是正确的。

$[0,0,0,1^*] \rightarrow [0,0,0,1^*]$  在加密 3 轮后的结果为  $[1^*,1,1,0]$ , 解密 6 轮后的结果为  $[2^*,t,t,t], 1^*$  与  $2^*$  出现冲突, 另一种情况同理可得。

### 3.2 编程构造 3 轮 DES 算法线性路线

首先，需要构造 8 个 S 盒对应的线性分布表，判断  $\alpha$  和  $\beta$  对应掩码下，线性近似式是否成立，对应的 NS+1，代码如下所示

```
1 def get_S_table(): #生成线性分布表
```

```

2   for i in range(8):
3       for alpha in range(64):
4           for beta in range(16):
5               for x in range(64):
6                   if(xor(x & alpha,6)==xor(S(i,x) & beta,4)):
7                       S_table[i][alpha][beta] += 1

```

对于每个 S 盒的线性分布表，首先，找到分布表中绝对值最大的项，用来构造线性路线，并存储每对符合条件的  $\alpha$  和  $\beta$ ，代码如下：

```

1   max = [0,0,0] # (i,j,NS)
2   for i in range(1,64):
3       for j in range(16):
4           if abs(S_table[num][i][j]) > abs(max[2]):
5               max[0],max[1],max[2] = i,j,S_table[num][i][j] #找到线性分布表里最大的i,j
6   best=[] # (目前最大概率可不可以在当前i找到j与之对应，当前最大概率)
7   for i in range(1,64):
8       try:
9           best.append([i, S_table[num][i].index(max[2]), max[2]]) #如果有就放到best
10      except ValueError:
11          pass

```

当找到最大偏差对应的掩码 1 后，需要利用 S 盒的输入和输出掩码计算 F 函数的输入和输出掩码，即通过 E 扩展和逆 P 置换。特别的，在 Matsui 的论文中给出的近似式（即实验文档中给出的范例）其标号与现在惯用的 E 扩展和 P 置换标号相反，且从 1 开始计数，为了与实验文档保持一致，我们此处也使用与 Matsui 论文一致的标号规则，得到代码如下：

```

1   for i in alpha: #i
2       R.append(32 - E_box[i]) # 输入过E扩展
3       K.append(47 - i) #S盒输入对应的轮密钥K的位置
4   for i in beta: #j
5       F.append(31-P_box.index(i+1)) #输出过逆p置换

```

运行代码，可以得到 8 个 S 盒对应的偏差最大的三轮线性近似式：

S 盒	掩码与偏差	线性近似式
1	16,15, -18	$X_2[23, 15, 9, 1] \oplus P_H[23, 15, 9, 1] \oplus P_L[31] = K1[46]$
2	34,11, -16	$X_2[19, 30, 14] \oplus P_H[19, 30, 14] \oplus P_L[28, 24] = K1[41, 37]$
3	34,15, 16	$X_2[8, 16, 2, 26] \oplus P_H[8, 16, 2, 26] \oplus P_L[24, 20] = K1[35, 31]$
4	34,15, -16	$X_2[6, 12, 22, 31] \oplus P_H[6, 12, 22, 31] \oplus P_L[20, 16] = K1[29, 25]$
	40,15,-16	$X_2[6, 12, 22, 31] \oplus P_H[6, 12, 22, 31] \oplus P_L[20, 18] = K1[29, 27]$
	43,9,-16	$X_2[6, 31] \oplus P_H[6, 31] \oplus P_L[20, 18, 16, 15] = K1[29, 27, 25, 24]$
5	16,15, -20	$X_2[24, 18, 7, 29] \oplus P_H[24, 18, 7, 29] \oplus P_L[15] = K1[22]$
6	16,7,-14	$X_2[3, 21, 13] \oplus P_H[3, 21, 13] \oplus P_L[11] = K1[16]$
7	59,4,-18	$X_2[20] \oplus P_H[20] \oplus P_L[8, 7, 6, 4, 3] = K1[11, 10, 9, 7, 6]$
8	16,15,-16	$X_2[27, 5, 17, 11] \oplus P_H[27, 5, 17, 11] \oplus P_L[3] = K1[4]$

我们还搜索了 8 个 S 盒对应的偏差次大/第三大的 16 个三轮线性近似式，并在 3.3 中对其构造的五轮近似式进行了讨论。

### 3.3 在三轮近似式基础上构造五轮近似式

下面讨论如何利用已经从找到的三轮路线构造五轮近似式，首先，该路线指定了 F 函数的输出掩码，只需要枚举对应的输入掩码找到最大偏差即可，特别的，由于当第一轮的输出掩码非 0 位大于一时，P 置换导致同一列几乎不可能由同一个 S 盒生成，所有使得该路线涉及两个及以上 S 盒，增大了分析难度，故我们只筛选与论文中一样的掩码非 0 位只有 1bit 的情况，在枚举之前，需要将该掩码恢复到 S 盒的输出掩码的形式上，利用 P 置换的值得到其对应哪个 S 盒，代码如下：

```

1  beta=[]
2  for i in left:
3      beta.append(P_box[31-i]-1)#yes
4  bb=[0,0,0,0]
5  for k in beta:
6      num=k//4
7      bb[(k - num*4)] = 1  #反解S盒序号
8  item=0
9  for i in range(4):
10     if bb[i]==1:

```

```

11         item += bb[i] * (2 ** (3 - i))
12     max = [0,0,0] # (i,j,NS)
13     for i in range(1,64):
14         if abs(S_table[num][i][item]) > abs(max[2]):#
15             max[0],max[1],max[2] = i,item,S_table[num][i][item]
16             #找到线性分布表里最大的i,j, 赋值给max

```

得到偏差最大的路线后，重复在 3.2 中进行的操作，经过 E 扩展和逆 P 置换，得到完整的线性近似式，并计算其概率，代码如下：

```

1     for i in alpha_loc: #i
2         R_local.append(32 - E_box[i]) # 输入过E扩展
3         K_local.append(47 - i) #S盒输入对应的轮密钥K的位置
4     for i in beta_loc: #j
5         F_local.append(31-P_box.index(i+1)) #输出过逆p置换
6     print('\t新添加头尾部在第{}个S盒'.format(num+1))
7     print('\t = {}, = {}, NS = {}, \n\t头尾构造: X_2{f} P_H{f} P_L{r} = K1{k}'.format(
item[0],item[1],item[2],r=R_local,f=F_local,k=K_local))
8     print("\tp1=",p,"p2=",max[2])
9     for i in F1: #组成整个P_L比特
10         R_local.append(i) #输出过逆p置换
11     print('\t五轮总逼近式: P_h{r1} P_l{r} C_h{r1} C_l{r} = K1{k} K2{k1} K4{k1} K5{
k}'.format(r=R_local,f=F_local,k=K_local,r1=left,f1=F1,k1=K1))
12     print("\t5轮总概率为: ",0.5+(2**4)*(p/64)*(max[2]/64)*(p/64)*(max[2]/64)*0.5)
13     print()

```

运行结果如下：



```

α=34, β=1b, NS=16, 第一轮逼近式: X_2[18, 16, 2, 26] ⊕ P_H[8, 16, 2, 26] ⊕ P_L[24, 20] = K1[39, 31]
三轮总逼近式: P_low[24, 20] ⊕ P_h[8, 16, 2, 26] ⊕ C_low[24, 20] ⊕ C_h[8, 16, 2, 26] = K1[35, 31] ⊕ K3[35, 31]
无法构造五轮逼近式

#####第4个S盒, 找到偏差最高三轮逼近式#####
α=34, β=15, NS=16, 第一轮逼近式: X_2[6, 12, 22, 31] ⊕ P_H[6, 12, 22, 31] ⊕ P_L[20, 16] = K1[29, 25]
三轮总逼近式: P_low[20, 16] ⊕ P_h[6, 12, 22, 31] ⊕ C_low[20, 16] ⊕ C_h[6, 12, 22, 31] = K1[29, 25] ⊕ K3[29, 25]
无法构造五轮逼近式

α=40, β=15, NS=-16, 第一轮逼近式: X_2[6, 12, 22, 31] ⊕ P_H[6, 12, 22, 31] ⊕ P_L[20, 18] = K1[29, 27]
三轮总逼近式: P_low[20, 18] ⊕ P_h[6, 12, 22, 31] ⊕ C_low[20, 18] ⊕ C_h[6, 12, 22, 31] = K1[29, 27] ⊕ K3[29, 27]
无法构造五轮逼近式

α=43, β=9, NS=-16, 第一轮逼近式: X_2[6, 31] ⊕ P_H[6, 31] ⊕ P_L[20, 18, 16, 15] = K1[29, 27, 25, 24]
三轮总逼近式: P_low[20, 18, 16, 15] ⊕ P_h[6, 31] ⊕ C_low[20, 18, 16, 15] ⊕ C_h[6, 31] = K1[29, 27, 25, 24] ⊕ K3[29, 27, 25, 24]
无法构造五轮逼近式

#####第5个S盒, 找到偏差最高三轮逼近式#####
α=16, β=7, NS=-20, 第一轮逼近式: X_2[24, 18, 7, 29] ⊕ P_H[24, 18, 7, 29] ⊕ P_L[15] = K1[22]
三轮总逼近式: P_low[15] ⊕ P_h[24, 18, 7, 29] ⊕ C_low[15] ⊕ C_h[24, 18, 7, 29] = K1[22] ⊕ K3[22]
新添加头尾部在第1个S盒
α=27, β=4, NS=-10, 头尾构造: X_2[19] ⊕ P_H[19] ⊕ P_L[31, 30, 28, 27] = K1[46, 45, 43, 42]
五轮总逼近式: P_h[15] ⊕ P_l[31, 30, 28, 27, 24, 18, 7, 29] ⊕ C_h[15] ⊕ C_l[31, 30, 28, 27, 24, 18, 7, 29] = K1[46, 45, 43, 42] ⊕ K2[22] ⊕ K4[22] ⊕ K5[46, 45, 43, 42]
5轮总概率为: 0.519073486328125

#####第6个S盒, 找到偏差最高三轮逼近式#####
α=16, β=7, NS=-14, 第一轮逼近式: X_2[3, 21, 13] ⊕ P_H[3, 21, 13] ⊕ P_L[11] = K1[16]
三轮总逼近式: P_low[11] ⊕ P_h[3, 21, 13] ⊕ C_low[11] ⊕ C_h[3, 21, 13] = K1[16] ⊕ K3[16]
新添加头尾部在第8个S盒
α=43, β=1, NS=-10, 头尾构造: X_2[11] ⊕ P_H[11] ⊕ P_L[4, 2, 0, 31] = K1[5, 3, 1, 0]
五轮总逼近式: P_h[11] ⊕ P_l[4, 2, 0, 31, 3, 21, 13] ⊕ C_h[11] ⊕ C_l[4, 2, 0, 31, 3, 21, 13] = K1[5, 3, 1, 0] ⊕ K2[16] ⊕ K4[16] ⊕ K5[5, 3, 1, 0]
5轮总概率为: 0.5093460083007812

```

可以看到对于第五个S盒,其偏差最大的路线即为实验文档中给出的  $X_2[7, 18, 24, 29] \oplus P_H[7, 18, 24, 29] \oplus P_L[15] = K_1[22]$ , 其第一轮和最后一轮新增的路线位于第一个S盒, 为  $X_2[15] \oplus P_H[15] \oplus P_L[31, 30, 28, 27] = K_1[46, 45, 43, 42]$ .

八个S盒完整的构造数据如下 (其顺序按照 3.2 中一轮近似式顺序, 对应构造, 当该掩码非 0 位个数大于等于 2 时标记为 none):

S 盒	新 S 盒	新添加线性近似式
1	4	$X_2[31] \oplus P_H[31] \oplus P_L[20, 17, 16] = K_1[29, 26, 25]$
2	none	none
3	none	none
4	none	none
	none	none
	none	none
5	1	$X_2[15] \oplus P_H[15] \oplus P_L[31, 30, 28, 27] = K_1[46, 45, 43, 42]$
6	8	$X_2[11] \oplus P_H[11] \oplus P_L[4, 2, 0, 31] = K_1[5, 3, 1, 0]$
7	none	none
8	6	$X_2[3] \oplus P_H[3] \oplus P_L[12, 11, 10, 9, 8, 7] = K_1[17, 16, 15, 14, 13, 12]$

与第一轮组合, 可以得到五轮线性近似式如下表 (由于图表篇幅限制, 表中省略 S 盒序号, 其顺序按照上图中不为 none 的近似式由上到下排序):

五轮线性近似式
$P_h[31] \oplus P_l[20, 17, 16, 23, 15, 9, 1] \oplus C_h[31] \oplus C_l[20, 17, 16, 23, 15, 9, 1] = K_1[29, 26, 25]$

$\oplus K2[46] \oplus K4[46] \oplus K5[29, 26, 25]$
$P_h[15] \oplus P_l[31, 30, 28, 27, 24, 18, 7, 29] \oplus C_h[15] \oplus C_l[31, 30, 28, 27, 24, 18, 7, 29] = K1[46, 45, 43, 42] \oplus K2[22] \oplus K4[22] \oplus K5[46, 45, 43, 42]$
$P_h[11] \oplus P_l[4, 2, 0, 31, 3, 21, 13] \oplus C_h[11] \oplus C_l[4, 2, 0, 31, 3, 21, 13] = K1[5, 3, 1, 0] \oplus K2[16] \oplus K4[16] \oplus K5[5, 3, 1, 0]$
$P_h[3] \oplus P_l[12, 11, 10, 9, 8, 7, 27, 5, 17, 11] \oplus C_h[3] \oplus C_l[12, 11, 10, 9, 8, 7, 27, 5, 17, 11] = K1[17, 16, 15, 14, 13, 12] \oplus K2[4] \oplus K4[4] \oplus K5[17, 16, 15, 14, 13, 12]$

其概率分别为 0.5154, 0.5191(论文中给出的路线), 0.5093, 0.5176.

得到四条路径后, 很自然可以想到, 五轮偏差最大的情况也许不是由三轮偏差最大路线导出的, 于是继续搜索三轮近似式偏差次大/第三大的情况, 发现所有 S 盒偏差次大时都不满足”掩码非零位只有一个“的条件, 结果如下:

```
#####第2个S盒, 找到偏差最高三轮逼近式#####
α=31, β=8, NS=-14, 第一轮逼近式: X_2[19]⊕P_H[19]⊕P_L[27, 26, 25, 24, 23]=K1[40, 39, 38, 37, 36]
三轮总逼近式: P_low[27, 26, 25, 24, 23]⊕P_h[19]⊕C_low[27, 26, 25, 24, 23]⊕C_h[19]=K1[40, 39, 38, 37, 36]⊕K3[40, 39, 38, 37, 36]
无法构造五轮逼近式

α=38, β=4, NS=-14, 第一轮逼近式: X_2[4]⊕P_H[4]⊕P_L[28, 25, 24]=K1[41, 38, 37]
三轮总逼近式: P_low[28, 25, 24]⊕P_h[4]⊕C_low[28, 25, 24]⊕C_h[4]=K1[41, 38, 37]⊕K3[41, 38, 37]
无法构造五轮逼近式

α=55, β=4, NS=-14, 第一轮逼近式: X_2[4]⊕P_H[4]⊕P_L[28, 27, 25, 24, 23]=K1[41, 40, 38, 37, 36]
三轮总逼近式: P_low[28, 27, 25, 24, 23]⊕P_h[4]⊕C_low[28, 27, 25, 24, 23]⊕C_h[4]=K1[41, 40, 38, 37, 36]⊕K3[41, 40, 38, 37, 36]
无法构造五轮逼近式

#####第3个S盒, 找到偏差最高三轮逼近式#####
α=18, β=13, NS=-14, 第一轮逼近式: X_2[8, 16, 26]⊕P_H[8, 16, 26]⊕P_L[23, 20]=K1[34, 31]
三轮总逼近式: P_low[23, 20]⊕P_h[8, 16, 26]⊕C_low[23, 20]⊕C_h[8, 16, 26]=K1[34, 31]⊕K3[34, 31]
无法构造五轮逼近式

α=59, β=1, NS=-14, 第一轮逼近式: X_2[26]⊕P_H[26]⊕P_L[24, 23, 22, 20, 19]=K1[35, 34, 33, 31, 30]
三轮总逼近式: P_low[24, 23, 22, 20, 19]⊕P_h[26]⊕C_low[24, 23, 22, 20, 19]⊕C_h[26]=K1[35, 34, 33, 31, 30]⊕K3[35, 34, 33, 31, 30]
无法构造五轮逼近式

#####第4个S盒, 找到偏差最高三轮逼近式#####
α=34, β=15, NS=-16, 第一轮逼近式: X_2[6, 12, 22, 31]⊕P_H[6, 12, 22, 31]⊕P_L[20, 16]=K1[29, 25]
三轮总逼近式: P_low[20, 16]⊕P_h[6, 12, 22, 31]⊕C_low[20, 16]⊕C_h[6, 12, 22, 31]=K1[29, 25]⊕K3[29, 25]
无法构造五轮逼近式

α=40, β=15, NS=-16, 第一轮逼近式: X_2[6, 12, 22, 31]⊕P_H[6, 12, 22, 31]⊕P_L[20, 18]=K1[29, 27]
三轮总逼近式: P_low[20, 18]⊕P_h[6, 12, 22, 31]⊕C_low[20, 18]⊕C_h[6, 12, 22, 31]=K1[29, 27]⊕K3[29, 27]
无法构造五轮逼近式

α=43, β=9, NS=-16, 第一轮逼近式: X_2[6, 31]⊕P_H[6, 31]⊕P_L[20, 18, 16, 15]=K1[29, 27, 25, 24]
三轮总逼近式: P_low[20, 18, 16, 15]⊕P_h[6, 31]⊕C_low[20, 18, 16, 15]⊕C_h[6, 31]=K1[29, 27, 25, 24]⊕K3[29, 27, 25, 24]
无法构造五轮逼近式

#####第5个S盒, 找到偏差最高三轮逼近式#####
α=34, β=14, NS=-16, 第一轮逼近式: X_2[24, 18, 7]⊕P_H[24, 18, 7]⊕P_L[16, 12]=K1[23, 19]
三轮总逼近式: P_low[16, 12]⊕P_h[24, 18, 7]⊕C_low[16, 12]⊕C_h[24, 18, 7]=K1[23, 19]⊕K3[23, 19]
无法构造五轮逼近式

#####第6个S盒, 找到偏差最高三轮逼近式#####
α=34, β=11, NS=14, 第一轮逼近式: X_2[28, 21, 13]⊕P_H[28, 21, 13]⊕P_L[12, 8]=K1[17, 13]
三轮总逼近式: P_low[12, 8]⊕P_h[28, 21, 13]⊕C_low[12, 8]⊕C_h[28, 21, 13]=K1[17, 13]⊕K3[17, 13]
无法构造五轮逼近式

#####第7个S盒, 找到偏差最高三轮逼近式#####
α=34, β=14, NS=-16, 第一轮逼近式: X_2[0, 20, 10]⊕P_H[0, 20, 10]⊕P_L[8, 4]=K1[11, 7]
三轮总逼近式: P_low[8, 4]⊕P_h[0, 20, 10]⊕C_low[8, 4]⊕C_h[0, 20, 10]=K1[11, 7]⊕K3[11, 7]
无法构造五轮逼近式

#####第8个S盒, 找到偏差最高三轮逼近式#####
α=34, β=14, NS=-16, 第一轮逼近式: X_2[27, 5, 17]⊕P_H[27, 5, 17]⊕P_L[4, 0]=K1[5, 1]
三轮总逼近式: P_low[4, 0]⊕P_h[27, 5, 17]⊕C_low[4, 0]⊕C_h[27, 5, 17]=K1[5, 1]⊕K3[5, 1]
无法构造五轮逼近式
```

搜索偏差第三大, 可以发现只有第 7 个和第八个 S 盒存在可以构造的五轮近似式:

```

#####第7个S盒，找到偏差最高三轮逼近式#####
α=16, β=15, NS=-14, 第一轮逼近式: X_2[0, 20, 10, 25] ⊕ P_H[0, 20, 10, 25] ⊕ P_L[7]=K1[10]
二轮总逼近式: P_low[7] ⊕ P_h[0, 20, 10, 25] ⊕ C_low[7] ⊕ C_h[0, 20, 10, 25]=K1[10] ⊕ K3[10]
新添加头尾部在第5个S盒
α=52, β=2, NS=-12, 头尾构造: X_2[7] ⊕ P_H[7] ⊕ P_L[16, 15, 13]=K1[23, 22, 20]
二轮总逼近式: P_h[7] ⊕ P_l[16, 15, 13, 0, 20, 10, 25] ⊕ C_h[7] ⊕ C_l[16, 15, 13, 0, 20, 10, 25]=K1[23, 22, 20] ⊕ K2[10] ⊕ K4[10] ⊕ K5[23, 22, 20]
5轮总概率为: 0.513458251953125

α=18, β=11, NS=-14, 第一轮逼近式: X_2[0, 10, 25] ⊕ P_H[0, 10, 25] ⊕ P_L[7, 4]=K1[10, 7]
二轮总逼近式: P_low[7, 4] ⊕ P_h[0, 10, 25] ⊕ C_low[7, 4] ⊕ C_h[0, 10, 25]=K1[10, 7] ⊕ K3[10, 7]
无法构造五轮逼近式

#####第8个S盒，找到偏差最高三轮逼近式#####
α=4, β=15, NS=14, 第一轮逼近式: X_2[27, 5, 17, 11] ⊕ P_H[27, 5, 17, 11] ⊕ P_L[1]=K1[2]
二轮总逼近式: P_low[1] ⊕ P_h[27, 5, 17, 11] ⊕ C_low[1] ⊕ C_h[27, 5, 17, 11]=K1[2] ⊕ K3[2]
新添加头尾部在第1个S盒
α=63, β=1, NS=-14, 头尾构造: X_2[1] ⊕ P_H[1] ⊕ P_L[0, 31, 30, 29, 28, 27]=K1[47, 46, 45, 44, 43, 42]
二轮总逼近式: P_h[1] ⊕ P_l[0, 31, 30, 29, 28, 27, 5, 17, 11] ⊕ C_h[1] ⊕ C_l[0, 31, 30, 29, 28, 27, 5, 17, 11]=K1[47, 46, 45, 44, 43, 42] ⊕ K2[2] ⊕ K4[2] ⊕ K5[47, 46, 45, 44, 43, 42]
5轮总概率为: 0.5183181762695312

```

概率分别为 0.5135, 0.5183:

五轮线性近似式

$$P_h[7] \oplus P_l[16, 15, 13, 0, 20, 10, 25] \oplus C_h[7] \oplus C_l[16, 15, 13, 0, 20, 10, 25] = K1[23, 22, 20] \oplus K2[10] \oplus K4[10] \oplus K5[23, 22, 20]$$

$$P_h[1] \oplus P_l[0, 31, 30, 29, 28, 27, 5, 17, 11] \oplus C_h[1] \oplus C_l[0, 31, 30, 29, 28, 27, 5, 17, 11] = K1[47, 46, 45, 44, 43, 42] \oplus K2[2] \oplus K4[2] \oplus K5[47, 46, 45, 44, 43, 42]$$

### 3.4 利用 Matsui 算法 2 实现 6 轮 DES 算法密钥恢复攻击

#### 3.4.1 确定待恢复轮密钥位置

根据选择的五轮线性近似式的路线，需要恢复 1, 3, 7, 8 四个 S 盒的轮密钥信息，由于复杂度较高，我们首先使用老师提供的 32bit 主密钥信息降低复杂度，计算 32bit 到第六轮 48bit 的对应关系，方法如下：

对于每个第六轮轮密钥每个比特，使用 ReKeygen 函数对其进行逆扩展：使用函数 RePC1 对初始密钥进行置换操作，得到 56 位的中间结果 combined。使用函数 Rotate 对 combined 进行循环左移操作，生成每轮所需的子密钥。使用函数 RePC2 对每轮生成的子密钥进行 pc2 置换，得到 48 位的轮密钥。代码如下：

```

1 Bytes RePC1(Bytes l, Bytes r)
2 {
3     int pc1[56] = {
4         56, 48, 40, 32, 24, 16, 8,
5         0, 57, 49, 41, 33, 25, 17,
6         9, 1, 58, 50, 42, 34, 26,
7         18, 10, 2, 59, 51, 43, 35,
8         62, 54, 46, 38, 30, 22, 14,
9         6, 61, 53, 45, 37, 29, 21,

```

```

10     13, 5, 60, 52, 44, 36, 28,
11     20, 12, 4, 27, 19, 11, 3
12 };
13 int r_pc1[64] = { 0 };
14 ReBox(pc1, r_pc1, 56, 64);
15
16 Bytes combined={};
17
18 combined |= l;
19 combined <<= 28;
20 combined |= r;
21 return Permutate(combined, r_pc1, 56, 64);
22 }
23
24 Bytes RePC2(Bytes key)
25 {
26     int pc2[48] = {
27         13, 16, 10, 23, 0, 4,
28         2, 27, 14, 5, 20, 9,
29         22, 18, 11, 3, 25, 7,
30         15, 6, 26, 19, 12, 1,
31         40, 51, 30, 36, 46, 54,
32         29, 39, 50, 44, 32, 47,
33         43, 48, 38, 55, 33, 52,
34         45, 41, 49, 35, 28, 31
35     };
36     int r_pc2[56] = { 0 };
37     ReBox(pc2, r_pc2, 48, 56);
38     return Permutate(key, r_pc2, 48, 56);
39 }

```

对于缺失位，定义一个 mask 变量，通过 mask 来确定每轮中产生子密钥的位置。将 mask\_l 和 mask\_r 分别与 l 和 r 对应的部分进行循环左移，得到当前轮的 mask。定义 mask\_list 数组用于记录 mask 中为 1 的位置，并根据 mask\_list 更新主密钥 findkey 数组，代码如下：

```

1 final_key = RePC2(final_key);

```

```

2  int loss[8] = { 8, 17, 21, 24, 34, 37, 42, 53 };
3  Bytes mask = 0;
4  for (int i = 0; i < 8; i++)
5  {
6      mask = mask | 1ULL << (55 - loss[i]);
7  }
8  Bytes l = final_key >> 28;
9  Bytes r = final_key ^ l << 28;
10 Bytes mask_l = mask >> 28;
11 Bytes mask_r = mask ^ mask_l << 28;
12 int offset[16] = {
13     1, 1, 2, 2, 2, 2, 2, 2,
14     1, 2, 2, 2, 2, 2, 2, 1
15 };
16
17 for (int i = round - 1; i >= 0; i--)
18 {
19     l = Rotate(l, 28 - offset[i]);
20     r = Rotate(r, 28 - offset[i]);
21     mask_l = Rotate(mask_l, 28 - offset[i]);
22     mask_r = Rotate(mask_r, 28 - offset[i]);
23 }
24
25 Bytes key = RePC1(l, r);
26 mask = RePC1(mask_l, mask_r);
27 int j = 0;
28 for (int i = 63; i >= 0; i--)
29 {
30     if (mask & 1)
31     {
32         mask_list[j] = i;
33         j++;
34     }
35     mask >>= 1;
36 }

```

运行代码后可以确定第六轮轮密钥与主密钥 bit 的对应关系，得到第六轮轮密钥的掩码：

```

1 int known_mask[48] = {1,0,1,1,0,1,
2                       1,0,0,1,0,0,
3                       1,0,1,1,1,0,
4                       0,0,0,0,1,0,
5                       1,1,1,0,0,0,
6                       0,1,1,1,1,0,
7                       1,0,1,0,1,1,
8                       1,1,0,0,0,1,}; //1表示已知，0表示需要猜测

```

### 3.4.2 利用 Matsui 算法 2 进行线性分析

首先设置我们选取的路线的掩码：

```

1 #define PL_MASK 0x80000000
2 #define PR_MASK 0x00938202
3 #define CL_MASK 0x80000000
4 #define CR_MASK 0x00938202

```

定义一个轮密钥猜测函数 `gen_subkey` 和一个生成函数，`subkey_init()`，前者根据需要遍历的 4 个 S 盒的未知 bit，生成猜测密钥，并调用后者将已知的密钥比特（即老师提供的主密钥信息映射得到的轮密钥信息）加载进猜测密钥，通过这两个函数得到猜测未知信息的轮密钥：

```

1 uint64_t subkey_init()//把已知的密钥比特加载进去。
2 {
3     uint64_t key_init = 0;
4     for(int i = 0; i < 8; i++){
5         if(subkey_mask[i] == 1){
6             for(int j = 0; j < 6; j++){
7                 if(known_mask[i*6+j] == 1){
8                     key_init = (key_init << 1) | known_key[i*6+j];
9                 }else{
10                     key_init = (key_init << 1);
11                 }
12             }
13         }
14     }
15 }

```

```

13         // key_init = (key_init << 6)|subkey[i];
14     }else{
15         key_init = (key_init << 6);
16     }
17 }
18 return key_init;
19 }
20
21 uint64_t gen_subkey(uint64_t key_init, uint64_t buf)//根据遍历的4个S盒的未知bit，生成
    并返回塞完的密钥
22 {
23     uint64_t sub_key = key_init;
24     int cnt = 0;
25     for(int i = 0; i < 8; i++){
26         if(subkey_mask[i] == 1){
27             for(int j = 0; j < 6; j++){
28                 if(known_mask[i*6+j] == 0){
29                     sub_key |= (((buf>>(n0-cnt-1))&((uint64_t)1)) << (47-(i*6+j)));
30                     cnt++;
31                 }
32             }
33         }
34     }
35     return sub_key;
36 }

```

下面定义主函数和计数函数，通过读取明密文对 P 和 C，将其通过遍历所有可能的密钥，并对每个密钥进行解密操作，然后与预设的明文 m 进行比较，统计匹配次数，即计出现次数最多和最少的密钥，并根据最大/最小次数与 1/2 的关系，输出猜测的密钥，代码如下：

```

1 void get_cnt(uint64_t m, uint64_t c0)
2 {
3     //遍历所有密钥并判断计数
4     uint64_t key, key_init;
5     uint64_t c;
6     key_init = key0;
7     for(uint64_t i = 0; i < n; i++){

```

```

7     key = key_init | i;
8     key = gen_subkey(key_init, i);
9     k[i] = i;
10    c = des_dec(c0, key);
11    int res = get_xor(m,c);
12    if(res == 0)
13    {
14        T[i]++;
15    }
16 }
17 return ;
18 }
19 int main() {
20     freopen("1.txt","r",stdin);
21     uint64_t P,C;
22     for(int i = 0; i < 8; i++){
23         if(subkey_mask[i] == 1){
24             for(int j = 0; j < 6; j++){
25                 if(known_mask[i*6+j] == 0){
26                     n++;
27                 }
28             }
29         }
30     }
31     printf("%d\n",n);
32     n0 = n;
33     n = 1<<n;
34     key0 = subkey_init();
35     for(int i = 0; i < m0; i++){
36         scanf("%llx %llx", &P, &C);
37         get_cnt(P,C);
38     }
39     int maxn = 0, t_max;
40     int minn = 10086, t_min;
41     for(uint64_t i = 0; i < n; i++){
42         if(T[i] > maxn){
43             t_max = i;
44             maxn = T[i];

```

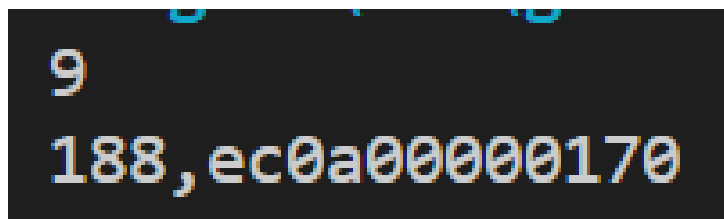


```

45     }
46     if(T[i] < minn){
47         t_min = i;
48         minn = T[i];
49     }
50 }
51 if(maxn-(m0/2) > (m0/2)-minn)
52     printf("%llx,%llx", k[t_max], gen_subkey(key0, k[t_max]));
53 else
54     printf("%llx,%llx", k[t_min], gen_subkey(key0, k[t_min]));
55 return 0;
56 }

```

选取  $4 \lceil |\varepsilon|^{-2} \rceil$  对明密文 (约等于 17000 对), 得到以下结果

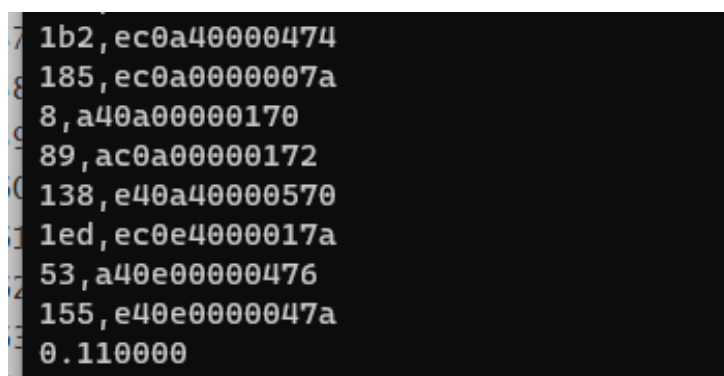


```

9
188,ec0a00000170

```

按照文档要求, 选取  $\lceil |\varepsilon|^{-2} \rceil$  对明密文 (约等于 4190 对), 重复 100 次实验, 得到以下结果:



```

7 1b2,ec0a40000474
8 185,ec0a0000007a
9 8,a40a00000170
10 89,ac0a00000172
11 138,e40a40000570
12 1ed,ec0e4000017a
13 53,a40e00000476
14 155,e40e0000047a
15 0.110000

```

测试得到攻击得到正确密钥的成功率约为 11%.

## 参考文献

- [1] 王美琴等. 密码分析学 [M]. 2023.

- 
- [2] Jongsung Kim, Seokhie Hong, Jaechul Sung, Sangjin Lee, Jongin Lim, , Soohak Sung. Impossible Differential Cryptanalysis for Block Cipher Structures[J]. 1990.
- [3] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher[J]. 2007.

---

## A 附录

### A.1 寻找 CLEFIA-128 不可能差分主要代码

```
1 class U_way:
2     def __init__(self,E,D) -> None:
3         self.E = E
4         self.D = D
5
6     def mul(self,x,y):
7         if y == '1':
8             return x
9         elif y == '0':
10            return '0'
11        elif x == '1*':
12            return '1'
13        elif x == '2*':
14            return 't'
15        else:
16            return x
17
18    def add(self,x,y):
19
20        if x == '0':
21            return y
22        elif y == '0':
23            return x
24
25        elif ((x == '1') & (y == '1*')) | ((x == '1*') & (y == '1')):
26            return '2*'
27
28        else:
29            return 't'
30
31    def mat_mul(self,X,M):
32        res = ['0']*len(M)
33        for i in range(len(M)):
34            for j in range(len(M[i])):
```

```

35         tmp = self.mul(X[j],M[i][j])
36         res[i] = self.add(res[i],tmp)
37     return res
38
39     def enc(self,P,r) -> list:
40         l = []
41         for _ in range(r):
42             P = self.mat_mul(P,self.E)
43             l.append(P)
44         return l
45
46     def dec(self,C,r):
47         l = []
48         for _ in range(r):
49             C = self.mat_mul(C,self.D)
50             l.append(C)
51         return l
52
53 E = [['f','1','0','0'],#加密时, 输出0为 F(p0) XOR p1
54      ['0','0','1','0'],#输出1为 p2
55      ['0','0','f','1'],#输出2为 F(p2) XOR p3
56      ['1','0','0','0']]#输出3为 p0
57
58 D = [['0','0','0','1'],#解密时, 输出0为 p3
59      ['1','0','0','f'],#输出1为 p0 XOR F(p3)
60      ['0','1','0','0'],#输出2为 p1
61      ['0','f','1','0']]#输出3为 F(p1) XOR p2
62
63
64 CLF = U_way(E,D)
65
66 pc = [['0' if j == '0' else '1*' for j in bin(i)[2:].zfill(4)] for i in range(0,16)]
67
68 pi = []
69 ci = []
70 for t in pc[1:]:
71     pi.append(CLF.enc(t,10))
72     ci.append(CLF.dec(t,10))

```

```

73
74 def check(x,y):
75     for i in range(4):#冲突判定
76         if(x[i] == '0'):
77             if (y[i] == '1')|(y[i] == '1*'):
78                 return 0
79         elif x[i] == '1':
80             if y[i] == '0':
81                 return 0
82         elif x[i] == '1*':
83             if (y[i] != '1')&(y[i] != 't'):
84                 return 0
85         elif x[i] == '2*':
86             if y[i] == '1*':
87                 return 0
88         else:
89             return 1
90
91 def reshape(l):
92     return [l[1],l[0],l[3],l[2]]
93
94 for i in range(len(pi)):
95     for j in range(len(ci)):
96         p = pi[i]
97         c = ci[j]
98         #轮数
99         maxn = 0
100        pl0,cl0 = 0,0
101        for pl in range(10):
102            for cl in range(10):
103                if(check(p[pl],c[cl])==0) & (maxn < pl+cl+3):
104                    pl0 = pl
105                    cl0 = cl
106                    maxn = pl+cl+2
107        if(maxn >= 9):
108            print('轮数: ',maxn,'头部差分',pc[i+1],'尾部差分',reshape(pc[j+1]),
109                  '加密轮数',pl0,'解密轮数',cl0,'加密差分情况',p[pl0],'解密差分情况',c[cl0])

```

## A.2 寻找 DES 三轮线性近似式和五轮线性近似式代码

```
1 import sys
2 S_box = [
3     [
4         0xe, 0x4, 0xd, 0x1, 0x2, 0xf, 0xb, 0x8, 0x3, 0xa, 0x6, 0xc, 0x5, 0x9, 0x0, 0x7,
5         0x0, 0xf, 0x7, 0x4, 0xe, 0x2, 0xd, 0x1, 0xa, 0x6, 0xc, 0xb, 0x9, 0x5, 0x3, 0x8,
6         0x4, 0x1, 0xe, 0x8, 0xd, 0x6, 0x2, 0xb, 0xf, 0xc, 0x9, 0x7, 0x3, 0xa, 0x5, 0x0,
7         0xf, 0xc, 0x8, 0x2, 0x4, 0x9, 0x1, 0x7, 0x5, 0xb, 0x3, 0xe, 0xa, 0x0, 0x6, 0xd,
8     ],
9     [
10        0xf, 0x1, 0x8, 0xe, 0x6, 0xb, 0x3, 0x4, 0x9, 0x7, 0x2, 0xd, 0xc, 0x0, 0x5, 0xa,
11        0x3, 0xd, 0x4, 0x7, 0xf, 0x2, 0x8, 0xe, 0xc, 0x0, 0x1, 0xa, 0x6, 0x9, 0xb, 0x5,
12        0x0, 0xe, 0x7, 0xb, 0xa, 0x4, 0xd, 0x1, 0x5, 0x8, 0xc, 0x6, 0x9, 0x3, 0x2, 0xf,
13        0xd, 0x8, 0xa, 0x1, 0x3, 0xf, 0x4, 0x2, 0xb, 0x6, 0x7, 0xc, 0x0, 0x5, 0xe, 0x9,
14    ],
15    [
16        0xa, 0x0, 0x9, 0xe, 0x6, 0x3, 0xf, 0x5, 0x1, 0xd, 0xc, 0x7, 0xb, 0x4, 0x2, 0x8,
17        0xd, 0x7, 0x0, 0x9, 0x3, 0x4, 0x6, 0xa, 0x2, 0x8, 0x5, 0xe, 0xc, 0xb, 0xf, 0x1,
18        0xd, 0x6, 0x4, 0x9, 0x8, 0xf, 0x3, 0x0, 0xb, 0x1, 0x2, 0xc, 0x5, 0xa, 0xe, 0x7,
19        0x1, 0xa, 0xd, 0x0, 0x6, 0x9, 0x8, 0x7, 0x4, 0xf, 0xe, 0x3, 0xb, 0x5, 0x2, 0xc,
20    ],
21    [
22        0x7, 0xd, 0xe, 0x3, 0x0, 0x6, 0x9, 0xa, 0x1, 0x2, 0x8, 0x5, 0xb, 0xc, 0x4, 0xf,
23        0xd, 0x8, 0xb, 0x5, 0x6, 0xf, 0x0, 0x3, 0x4, 0x7, 0x2, 0xc, 0x1, 0xa, 0xe, 0x9,
24        0xa, 0x6, 0x9, 0x0, 0xc, 0xb, 0x7, 0xd, 0xf, 0x1, 0x3, 0xe, 0x5, 0x2, 0x8, 0x4,
25        0x3, 0xf, 0x0, 0x6, 0xa, 0x1, 0xd, 0x8, 0x9, 0x4, 0x5, 0xb, 0xc, 0x7, 0x2, 0xe,
26    ],
27    [
28        0x2, 0xc, 0x4, 0x1, 0x7, 0xa, 0xb, 0x6, 0x8, 0x5, 0x3, 0xf, 0xd, 0x0, 0xe, 0x9,
29        0xe, 0xb, 0x2, 0xc, 0x4, 0x7, 0xd, 0x1, 0x5, 0x0, 0xf, 0xa, 0x3, 0x9, 0x8, 0x6,
30        0x4, 0x2, 0x1, 0xb, 0xa, 0xd, 0x7, 0x8, 0xf, 0x9, 0xc, 0x5, 0x6, 0x3, 0x0, 0xe,
31        0xb, 0x8, 0xc, 0x7, 0x1, 0xe, 0x2, 0xd, 0x6, 0xf, 0x0, 0x9, 0xa, 0x4, 0x5, 0x3,
32    ],
33    [
34        0xc, 0x1, 0xa, 0xf, 0x9, 0x2, 0x6, 0x8, 0x0, 0xd, 0x3, 0x4, 0xe, 0x7, 0x5, 0xb,
35        0xa, 0xf, 0x4, 0x2, 0x7, 0xc, 0x9, 0x5, 0x6, 0x1, 0xd, 0xe, 0x0, 0xb, 0x3, 0x8,
36        0x9, 0xe, 0xf, 0x5, 0x2, 0x8, 0xc, 0x3, 0x7, 0x0, 0x4, 0xa, 0x1, 0xd, 0xb, 0x6,
```

```

37         0x4, 0x3, 0x2, 0xc, 0x9, 0x5, 0xf, 0xa, 0xb, 0xe, 0x1, 0x7, 0x6, 0x0, 0x8, 0xd,
38     ],
39     [
40         0x4, 0xb, 0x2, 0xe, 0xf, 0x0, 0x8, 0xd, 0x3, 0xc, 0x9, 0x7, 0x5, 0xa, 0x6, 0x1,
41         0xd, 0x0, 0xb, 0x7, 0x4, 0x9, 0x1, 0xa, 0xe, 0x3, 0x5, 0xc, 0x2, 0xf, 0x8, 0x6,
42         0x1, 0x4, 0xb, 0xd, 0xc, 0x3, 0x7, 0xe, 0xa, 0xf, 0x6, 0x8, 0x0, 0x5, 0x9, 0x2,
43         0x6, 0xb, 0xd, 0x8, 0x1, 0x4, 0xa, 0x7, 0x9, 0x5, 0x0, 0xf, 0xe, 0x2, 0x3, 0xc,
44     ],
45     [
46         0xd, 0x2, 0x8, 0x4, 0x6, 0xf, 0xb, 0x1, 0xa, 0x9, 0x3, 0xe, 0x5, 0x0, 0xc, 0x7,
47         0x1, 0xf, 0xd, 0x8, 0xa, 0x3, 0x7, 0x4, 0xc, 0x5, 0x6, 0xb, 0x0, 0xe, 0x9, 0x2,
48         0x7, 0xb, 0x4, 0x1, 0x9, 0xc, 0xe, 0x2, 0x0, 0x6, 0xa, 0xd, 0xf, 0x3, 0x5, 0x8,
49         0x2, 0x1, 0xe, 0x7, 0x4, 0xa, 0x8, 0xd, 0xf, 0xc, 0x9, 0x0, 0x3, 0x5, 0x6, 0xb,
50     ],
51 ]
52
53 P_box = [
54     16, 7, 20, 21, 29, 12, 28, 17,
55     1, 15, 23, 26, 5, 18, 31, 10,
56     2, 8, 24, 14, 32, 27, 3, 9,
57     19, 13, 30, 6, 22, 11, 4, 25,
58 ]
59
60 E_box = [
61     32, 1, 2, 3, 4, 5,
62     4, 5, 6, 7, 8, 9,
63     8, 9, 10, 11, 12, 13,
64     12, 13, 14, 15, 16, 17,
65     16, 17, 18, 19, 20, 21,
66     20, 21, 22, 23, 24, 25,
67     24, 25, 26, 27, 28, 29,
68     28, 29, 30, 31, 32, 1,
69 ]
70 S_table = [
71     [
72         [
73             -32 for beta in range(16)
74         ] for alpha in range(64)

```

```

75         ]for i in range(8)
76     ]
77     trans6 = lambda n: '{:0>6}'.format(bin(n)[2:]) #掩码转换
78     trans4 = lambda n: '{:0>4}'.format(bin(n)[2:])
79
80     def S(i,IN):
81         bit6 = trans6(IN)
82         location = int(bit6[0]+bit6[-1],2)*16 + int(bit6[1:-1],2)
83         return int(S_box[i][location])
84
85     def xor(num,wide):
86         count=0
87         for i in range(wide):
88             if (num&1):
89                 count+=1
90                 num>>=1
91             if(count%2==0):
92                 return 0
93             return 1
94
95     def get_S_table(): #生成线性分布表
96         for i in range(8):
97             for alpha in range(64):
98                 for beta in range(16):
99                     for x in range(64):
100                         if(xor(x & alpha,6)==xor(S(i,x) & beta,4)):
101                             S_table[i][alpha][beta] += 1
102     def print_S_table(num):
103         print('\t\t\t\t\tS{}线性分布表'.format(num))
104         print(' \ \t',end='')
105         for i in range(16):
106             print(i,end='\t')
107         print('\n')
108         count = 0
109         for alpha in S_table[num-1]:
110             print(count,end='\t\t')
111             count +=1
112             for NS in alpha:

```



```

113         print(NS,end='\t')
114     print('\n')
115 def get_trace(num):
116     max = [0,0,0]  #(i,j,NS)
117     for i in range(1,64):
118         for j in range(16):
119             if abs(S_table[num][i][j]) > abs(max[2]):#
120                 max[0],max[1],max[2] = i,j,S_table[num][i][j]#找到线性分布表里最大的i,j, 赋值
121                 给max
122             #print("i=",max[0], " j=",max[1])
123         best=[]  # (存储输入掩码, 目前最大概率可不可以在当前i找到j与之对应, 当前最大概
124                 率)
125         for i in range(1,64):
126             try:
127                 best.append([i, S_table[num][i].index(max[2]), max[2]])#进行试探, 如果有就放到
128                 best里然后终止寻找
129             except ValueError:
130                 pass
131         #print("max",best)
132     print('#####第{}个S盒, 找到偏差最高三轮逼近式#####'.
133           format(num+1))
134     for item in best:
135         a = trans6(item[0])#掩码化
136         b = trans4(item[1])#掩码化
137         #print("item=",item[1])
138         alpha = []
139         beta = []
140         for i in range(6):
141             if a[i]=='1':
142                 alpha.append(num*6+i)  # 此时是左向右计数,从0开始计, 掩码位置
143         for i in range(4):
144             if b[i]=='1':
145                 beta.append(num*4+i)  # 此时是左向右计数, 掩码位置
146         #print("b=",b)
147         R = []#右输入
148         K = []#密钥
149         F = []#左输入
150         #print("beta_loca=",beta_loca)

```

```

147     flag=0
148     for i in alpha: #i
149         R.append(32 - E_box[i]) # 输入过E扩展
150         K.append(47 - i) #S盒输入对应的轮密钥K的位置
151         flag=flag+1
152     for i in beta: #j
153         F.append(31-P_box.index(i+1)) #输出过逆p置换
154
155
156     print('\t = {}, = {}, NS = {}, \t 第一轮逼近式: X_2{f} P_H{f} P_L{r} = K1{k}'.format(
item[0], item[1], item[-1], r=R, f=F, k=K))
157     print('\t 三轮总逼近式: P_low{r} P_h{f} C_low{r} C_h{f} = K1{k} K3{k}'.format(r=
R, f=F, k=K))
158
159     if flag==1:
160         get_head(R, max[2], F, K) #R是需要X2构造的序列，max是三轮线路的NS，F1和K1
是为了写总表达式
161     else:
162         print("\t 无法构造五轮逼近式")
163
164     #print("概率: ", (max[2]+32)%32, "/64")
165     print()
166
167 def get_head(left, p, F1, K1):
168     #print("\t 构造测试掩码为: ", left)
169     beta=[]
170     for i in left:
171         beta.append(P_box[31-i]-1) #yes
172     #print("beta", beta)
173     num=0;
174     bb=[0,0,0,0]
175     for k in beta:
176         #print("k=", k)
177         num=k//4
178         bb[(k - num*4)] = 1 #反解S盒序号
179     #print("bb=", bb)
180     item=0
181     for i in range(4):

```

```

182     if bb[i]==1:
183         item += bb[i] * (2 ** (3 - i))
184     #print("item=",item)
185     max = [0,0,0]  #(i,j,NS)
186     for i in range(1,64):
187         if abs(S_table[num][i][item]) > abs(max[2]):#
188             max[0],max[1],max[2] = i,item,S_table[num][i][item]#找到线性分布表里最大的i,j
            , 赋值给max
189
190     best=[]
191     best.append([max[0],max[1],max[2]])
192     #print("i=",max[0], " j=",max[1])
193
194
195     for item in best:
196         a = trans6(item[0])#掩码化
197         b = trans4(item[1])#掩码化
198         #print("item=",item[1])
199         alpha_loc = []
200         beta_loc = []
201         for i in range(6):
202             if a[i]=='1':
203                 alpha_loc.append(num*6+i)  # 此时是左向右计数,从0开始计, 掩码位置
204         for i in range(4):
205             if b[i]=='1':
206                 beta_loc.append(num*4+i)  # 此时是左向右计数, 掩码位置
207         #print("b=",b)
208         R_local = []#右输入
209         K_local = []#密钥
210         F_local = []#左输入
211         #print("beta_loc=",beta_loc)
212         for i in alpha_loc:  #i
213             R_local.append(32 - E_box[i])  # 输入过E扩展
214             K_local.append(47 - i)  #S盒输入对应的轮密钥K的位置
215
216         for i in beta_loc:  #j
217             F_local.append(31-P_box.index(i+1)) #输出过逆p置换
218

```

```

219     print('\t新添加头尾部在第{}个S盒'.format(num+1))
220     print('\t t = {}, = {}, NS = {}, \t 头尾构造: X_2{f} P_H{f} P_L{r} = K1{k}'.format(item
[0], item[1], item[2], r=R_local, f=F_local, k=K_local))
221     #print("\t p1=", p, ", p2=", max[2])
222     for i in F1: #组成整个P_L比特
223         R_local.append(i) #输出过逆p置换
224
225     print('\t 五轮总逼近式: P_h{r1} P_l{r} C_h{r1} C_l{r} = K1{k} K2{k1} K4{k1} K5{
k}'.format(r=R_local, f=F_local, k=K_local, r1=left, f1=F1, k1=K1))
226     print("\t 5轮总概率为: ", 0.5 + (2**4)*(p/64)*(max[2]/64)*(p/64)*(max[2]/64)*0.5)
227     #print("\t 成功率总概率为: ", 0.5 + (2**4)*(p/64)*(max[2]/64)*(p/64)*(max[2]/64)*0.5)
228     print()
229
230     #print()
231
232 if __name__ == '__main__':
233     get_S_table()
234     for i in range(8):
235         get_trace(i)

```

### A.3 DES 六轮算法 2 线性分析代码

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdint.h>
4
5  /* Initial Permutation Table */
6  static char IP[] = {
7      58, 50, 42, 34, 26, 18, 10, 2,
8      60, 52, 44, 36, 28, 20, 12, 4,
9      62, 54, 46, 38, 30, 22, 14, 6,
10     64, 56, 48, 40, 32, 24, 16, 8,
11     57, 49, 41, 33, 25, 17, 9, 1,
12     59, 51, 43, 35, 27, 19, 11, 3,
13     61, 53, 45, 37, 29, 21, 13, 5,
14     63, 55, 47, 39, 31, 23, 15, 7

```

---

```

15 };
16
17 /* Inverse Initial Permutation Table */
18 static char PI[] = {
19     40, 8, 48, 16, 56, 24, 64, 32,
20     39, 7, 47, 15, 55, 23, 63, 31,
21     38, 6, 46, 14, 54, 22, 62, 30,
22     37, 5, 45, 13, 53, 21, 61, 29,
23     36, 4, 44, 12, 52, 20, 60, 28,
24     35, 3, 43, 11, 51, 19, 59, 27,
25     34, 2, 42, 10, 50, 18, 58, 26,
26     33, 1, 41, 9, 49, 17, 57, 25
27 };
28
29 /*Expansion table */
30 static char E[] = {
31     32, 1, 2, 3, 4, 5,
32     4, 5, 6, 7, 8, 9,
33     8, 9, 10, 11, 12, 13,
34     12, 13, 14, 15, 16, 17,
35     16, 17, 18, 19, 20, 21,
36     20, 21, 22, 23, 24, 25,
37     24, 25, 26, 27, 28, 29,
38     28, 29, 30, 31, 32, 1
39 };
40
41 /* Post S-Box permutation */
42 static char P[] = {
43     16, 7, 20, 21,
44     29, 12, 28, 17,
45     1, 15, 23, 26,
46     5, 18, 31, 10,
47     2, 8, 24, 14,
48     32, 27, 3, 9,
49     19, 13, 30, 6,
50     22, 11, 4, 25
51 };
52

```

---

```

53  /* The S-Box tables */
54  static char S[8][64] = {{
55      /* S1 */
56      14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
57      0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
58      4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
59      15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13
60  },{
61      /* S2 */
62      15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
63      3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
64      0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
65      13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9
66  },{
67      /* S3 */
68      10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
69      13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
70      13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
71      1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12
72  },{
73      /* S4 */
74      7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
75      13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
76      10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
77      3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14
78  },{
79      /* S5 */
80      2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
81      14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
82      4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
83      11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3
84  },{
85      /* S6 */
86      12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
87      10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
88      9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
89      4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13
90  },{

```

---

```

91  /* S7 */
92    4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
93    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
94    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
95    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12
96 },{
97  /* S8 */
98    13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
99    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
100   7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
101   2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11
102 }};
103
104 /* Permuted Choice 1 Table */
105 static char PC1[] = {
106     57, 49, 41, 33, 25, 17, 9,
107     1, 58, 50, 42, 34, 26, 18,
108     10, 2, 59, 51, 43, 35, 27,
109     19, 11, 3, 60, 52, 44, 36,
110
111     63, 55, 47, 39, 31, 23, 15,
112     7, 62, 54, 46, 38, 30, 22,
113     14, 6, 61, 53, 45, 37, 29,
114     21, 13, 5, 28, 20, 12, 4
115 };
116
117 /* Permuted Choice 2 Table */
118 static char PC2[] = {
119     14, 17, 11, 24, 1, 5,
120     3, 28, 15, 6, 21, 10,
121     23, 19, 12, 4, 26, 8,
122     16, 7, 27, 20, 13, 2,
123     41, 52, 31, 37, 47, 55,
124     30, 40, 51, 45, 33, 48,
125     44, 49, 39, 56, 34, 53,
126     46, 42, 50, 36, 29, 32
127 };
128

```

```

129 /* Iteration Shift Array */
130 static char iteration_shift[] = {
131 /* 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 */
132     1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
133 };
134
135 #define LB32_MASK  0x00000001
136 #define LB64_MASK  0x0000000000000001
137 #define L64_MASK   0x00000000ffffffff
138 #define H64_MASK   0xffffffff00000000
139
140 uint64_t des_dec(uint64_t input, uint64_t sub_key) {
141
142     int i, j;
143
144     /* 8 bits */
145     char row, column;
146
147     /* 28 bits */
148     uint32_t C          = 0;
149     uint32_t D          = 0;
150
151     /* 32 bits */
152     uint32_t L          = 0;
153     uint32_t R          = 0;
154     uint32_t s_output   = 0;
155     uint32_t f_function_res = 0;
156     uint32_t temp       = 0;
157
158     /* 48 bits */
159     uint64_t s_input     = 0;
160
161     /* 56 bits */
162     uint64_t permuted_choice_1 = 0;
163     uint64_t permuted_choice_2 = 0;
164
165     /* 64 bits */
166     uint64_t init_perm_res    = 0;

```



```

167  uint64_t inv_init_perm_res = 0;
168  uint64_t pre_output      = 0;
169
170  // /* initial permutation */
171  // for (i = 0; i < 64; i++) {
172
173  //     init_perm_res <<= 1;
174  //     init_perm_res |= (input >> (64-IP[i])) & LB64_MASK;
175
176  // }
177
178  L = (uint32_t) (input >> 32) & L64_MASK;
179  R = (uint32_t) input & L64_MASK;
180
181
182  // round轮DES加密
183  for (i = 0; i < 1; i++) {
184
185      /* f(R,k) function */
186      s_input = 0;
187
188      for (j = 0; j < 48; j++) {
189
190          s_input <<= 1;
191          s_input |= (uint64_t) ((R >> (32-E[j])) & LB32_MASK);
192
193      }
194
195      /*
196       * Encryption/Decryption
197       * XORing expanded Ri with Ki
198       */
199
200      s_input = s_input ^ sub_key;
201
202      /* S-Box Tables */
203      for (j = 0; j < 8; j++) {
204          // 00 00 RCCC CR00 00 00 00 00 00 s_input

```

```

205 // 00 00 1000 0100 00 00 00 00 00 row mask
206 // 00 00 0111 1000 00 00 00 00 00 column mask
207
208 row = (char) ((s_input & (0x0000840000000000 >> 6*j)) >> 42-6*j);
209 row = (row >> 4) | row & 0x01;
210
211 column = (char) ((s_input & (0x0000780000000000 >> 6*j)) >> 43-6*j);
212
213 s_output <<= 4;
214 s_output |= (uint32_t) (S[j][16*row + column] & 0x0f);
215
216 }
217
218 f_function_res = 0;
219
220 for (j = 0; j < 32; j++) {
221
222     f_function_res <<= 1;
223     f_function_res |= (s_output >> (32 - P[j])) & LB32_MASK;
224
225 }
226
227 temp = R;
228 R = L ^ f_function_res;
229 L = temp;
230 // uint64_t temp1 = (((uint64_t) L) << 32) | (uint64_t) R;
231 // printf ("inner: %016llx\n", temp1);
232 }
233
234 pre_output = (((uint64_t) L) << 32) | (uint64_t) R;
235 // /* inverse initial permutation */
236 // for (i = 0; i < 64; i++) {
237
238 //     inv_init_perm_res <<= 1;
239 //     inv_init_perm_res |= (pre_output >> (64-PI[i])) & LB64_MASK;
240
241 // }
242

```

```

243     return pre_output;
244     // return inv_init_perm_res;
245
246 }
247
248 uint16_t subkey[8] = {0x3b, 0x21, 0x28, 0x36, 0x14, 0x3c, 0x05, 0x30}; //no use
249
250 #define PL_MASK 0x80000000 //1
251 #define PR_MASK 0x00938202
252 #define CL_MASK 0x80000000
253 #define CR_MASK 0x00938202
254
255 int subkey_mask[8] = {1,0,1,0,0,0,1,1}; //这四个S盒和五轮线近似式有关，这是第6轮，
256 int known_mask[48] = {1,0,1,1,0,1,
257                        1,0,0,1,0,0,
258                        1,0,1,1,1,0,
259                        0,0,0,0,1,0,
260                        1,1,1,0,0,0,
261                        0,1,1,1,1,0,
262                        1,0,1,0,1,1,
263                        1,1,0,0,0,1,}; //老师给出的32bit推出的24bit位置
264 uint64_t known_key[48] = {1,1,1,0,1,1,
265                            1,0,0,0,0,1,
266                            1,0,1,0,0,0,
267                            1,1,0,1,1,0,
268                            0,1,0,1,0,0,
269                            1,1,1,1,0,0,
270                            0,0,0,1,0,1,
271                            1,1,0,0,0,0}; //
272
273 int get_xor(uint64_t m, uint64_t c)
274 {
275     uint32_t PL = 0, PR = 0;
276     uint32_t CL = 0, CR = 0;
277     PL = (uint32_t) (m >> 32) & L64_MASK;
278     PR = (uint32_t) m & L64_MASK;
279     CL = (uint32_t) (c >> 32) & L64_MASK;
280     CR = (uint32_t) c & L64_MASK;

```

```

281
282 PL = PL & PL_MASK;
283 PR = PR & PR_MASK;
284
285 CL = CL & CL_MASK;
286 CR = CR & CR_MASK;
287
288 uint32_t tmp = PL ^ PR ^ CL ^ CR;
289
290 int res = 0;
291
292 for(int i = 0; i < 32; i++){
293     // res ^= ((PL >> i) & 1);
294     // res ^= ((PR >> i) & 1);
295     // res ^= ((CL >> i) & 1);
296     // res ^= ((CR >> i) & 1);
297     res ^= ((tmp >> i) & 1);
298 }
299 return res;
300 }
301
302 int m0 = 19000;
303 int n = 0,n0;
304 int T[1<<24];
305 uint64_t k[1<<24];
306
307 uint64_t subkey_init()//把已知的密钥比特加载进去。
308 {
309     uint64_t key_init = 0;
310     for(int i = 0; i < 8; i++){
311         if(subkey_mask[i] == 1){
312             for(int j = 0; j < 6; j++){
313                 if(known_mask[i*6+j] == 1){
314                     key_init = (key_init << 1) | known_key[i*6+j];
315                 }else{
316                     key_init = (key_init << 1);
317                 }
318             }

```

```

319         // key_init = (key_init << 6)|subkey[i];
320     }else{
321         key_init = (key_init << 6);
322     }
323 }
324 // printf("%llx\n",key_init);
325 return key_init;
326 }
327
328 uint64_t gen_subkey(uint64_t key_init, uint64_t buf)//根据遍历的4个S盒的未知bit，生成
    并返回塞完的密钥
329 {
330     uint64_t sub_key = key_init;
331     int cnt = 0;
332     // for(int i = 0; i < 48; i++){
333     //     if(known_mask[i] == 0){
334     //         sub_key |= (((buf>>(n-cnt+1))&(uint64_t)1) << (47-i));
335     //     }
336     // }
337
338     for(int i = 0; i < 8; i++){
339         if(subkey_mask[i] == 1){
340             for(int j = 0; j < 6; j++){
341                 if(known_mask[i*6+j] == 0){
342                     sub_key |= (((buf>>(n0-cnt-1))&(uint64_t)1) << (47-(i*6+j)));
343                     cnt++;
344                 }
345             }
346         }
347     }
348
349     // sub_key = key_init | buf;
350     // printf("%d %d %d\n",buf,sub_key&0b111111,cnt);
351     return sub_key;
352 }
353
354 uint64_t key0;
355

```

```

356 void get_cnt(uint64_t m, uint64_t c0)
357 { //遍历所有密钥并jishu,
358     uint64_t key, key_init;
359     uint64_t c;
360     key_init = key0;
361     for(uint64_t i = 0; i < n; i++){
362         key = key_init | i;
363         key = gen_subkey(key_init, i);
364         k[i] = i;
365         c = des_dec(c0, key);
366         int res = get_xor(m,c);
367         if(res == 0)
368         {
369             T[i]++;
370         }
371     }
372     return ;
373 }
374
375 int main()
376 {
377     int flag = 0;
378     freopen("1.txt","r",stdin);
379     uint64_t P,C;
380     for(int i = 0; i < 8; i++){
381         if(subkey_mask[i] == 1){
382             for(int j = 0; j < 6; j++){
383                 if(known_mask[i*6+j] == 0){
384                     n++;
385                 }
386             }
387         }
388     }
389     printf("%d\n",n);
390     n0 = n;
391     n = 1<<n;
392
393     //for (int num = 0; num < 5; num++)

```

```

394 // {
395     key0 = subkey_init();
396     for (int i = 0; i < m0; i++)
397     {
398         scanf("%llx %llx", &P, &C);
399         get_cnt(P, C);
400     }
401     int maxn = 0, t_max;
402     int minn = 10086, t_min;
403     for (uint64_t i = 0; i < n; i++) {
404         if (T[i] > maxn) {
405             t_max = i;
406             maxn = T[i];
407         }
408         if (T[i] < minn) {
409             t_min = i;
410             minn = T[i];
411         }
412     }
413     if (maxn - (m0 / 2) > (m0 / 2) - minn)
414     {
415         uint64_t ans1 = gen_subkey(key0, k[t_max]);
416         printf("%llx,%llx\n", k[t_max], ans1);
417         if (ans1 == 0xa40a00000072) flag++;
418     }
419     else {
420
421         uint64_t ans2 = gen_subkey(key0, k[t_min]);
422         printf("%llx,%llx\n", k[t_min], ans2);
423         if (ans2 == 0xa40a00000072) flag++;
424     }
425
426 // }
427 printf("%d", flag);
428 return 0;
429 }

```