编号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |

课程实验(设计)报告

课程名称 ： 密码分析与实践

报告名称 ： 三轮和六轮DES的差分分析

专业(班) ： 信安9班

学 号 ： 2021302141191

姓 名 ： 夏云琨

任课教师 ： 王张宜

2024 年06月17日

# 三轮和六轮DES的差分密码分析

## 1 攻击目标介绍

### DES简介

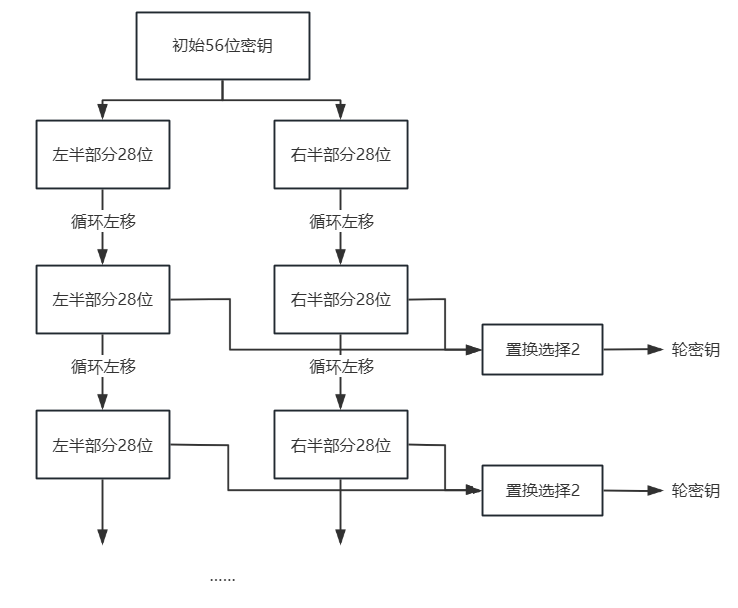
**数据加密标准**（Data Encryption Standard，简称**DES**）是一种对称密钥加密分组密码算法，其前身是于1972年被研制出的一种被称为Lucifer的加密算法。随后，IBM公司在其基础上作了改进，在1973年在美国国家标准局（NBS）向社会的公开征集中被选中，并于1976年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS）。

## 缩减的DES

本报告的分析目标是针对3轮和6轮DES的差分攻击。在完整DES中，有一些操作并不会对分析产生影响，因此为了简化分析过程，对原DES进行缩减。对于64bits的主密钥，略去密钥中的奇偶校验比特，缩减为56bits，在生成轮密钥时，略去置换选择1；对于加解密中的明文处理，略去了初始的IP变换和结尾的逆IP置换。缩减的DES详细流程如下：

* **轮密钥生成**

轮密钥生成流程如下：



相较于完整DES，缩减DES的轮密钥相当于在去除奇偶校验位和置换选择1的情况下使用DES的前N轮轮密钥，因为在循环移位时使用的是前N轮的移位值。代码如下：

key\_round = [[] for i in range(N)]

    for i in range(N):

        mov = move[i]

        tmp = key[mov:28] + key[:mov] + key[28 + mov:] + key[28:28 + mov]

        key\_round[i] = matrix\_trans(tmp, perm\_matrix\_after)

其中，N为DES轮数，move为每轮循环移位的位数列表：

move = [1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28]

matrix\_trans()为矩阵置换函数，其接受一个01字符串和一个矩阵，返回经过置换后的字符串，在此处用于进行置换选择2操作：

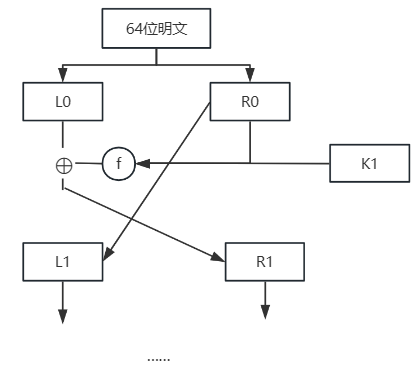
*# 矩阵置换*

matrix\_trans = lambda str, box: ''.join(str[i - 1] for i in box)

Perm\_matrix\_after为置换选择2矩阵。

* **明文加密**

明文加密的流程如下：



明文加密的单轮过程较完整DES没有变化。加密中使用到的模块有：

hex2bin()，将16进制字符串转为2进制字符串：

*# 16进制转2进制表*

hex2bin\_box = {'0': '0000', '1': '0001', '2': '0010', '3': '0011',

               '4': '0100', '5': '0101', '6': '0110', '7': '0111',

               '8': '1000', '9': '1001', 'a': '1010', 'b': '1011',

               'c': '1100', 'd': '1101', 'e': '1110', 'f': '1111'}

*# 16进制串转2进制串*

hex2bin = lambda x: ''.join(hex2bin\_box[i] for i in x)

xor\_bin()，二进制串异或：

*# 2进制串异或*

def xor\_bin(a, b):

    assert len(a) == len(b)

    return bin(int(a, 2) ^ int(b, 2))[2:].rjust(len(a), '0')

S\_box()，从S盒获得输出：

*# 从整型输入获得S盒坐标*

index\_x = lambda x: (x & 0x20) | (x & 1) << 4

index\_y = lambda x: (x >> 1) & 0xf

*# 取第i个S盒的输出*

S\_box = lambda i, x: S[i][index\_x(x) | index\_y(x)]

明文加密过程如下：

跳过IP置换将P分为L，R。将R经过矩阵E拓展为48位，与轮密钥异或；再经过S盒；最后经过p置换，与L异或后得到下一轮的L。经过N轮得到密文，跳过逆IP置换：

P = hex2bin(P.lower())

L, R = P[:32], P[32:]

for \_ in range(N):

    temp = xor\_bin(matrix\_trans(R, expand\_e), key\_round[\_])

    temp = ''.join(bin(S\_box(i, int(temp[i \* 6:i \* 6 + 6], 2)))[2:].rjust(4, '0') for i in range(8))

    temp = matrix\_trans(temp, p)

    L, R = R, xor\_bin(temp, L)

return bin2hex(L + R)

## 攻击原理介绍

差分攻击由Biham和Shamir于1991年提出的一种选择明文攻击，其核心思想是通过分析特定明文差分对对应的密文差分的影响来提取密钥，被认为是针对分组密码攻击最有效的方法之一。

假设用密钥K加密明文P得到的密文为T，用密钥K加密明文P’得到的密文为T’，那么P⊕P’称为明文对的差分，T⊕T’称为密文对的差分，差分攻击就是利用明文对差分和密文对差分与密钥之间的关联，通过选择特定明文差分来获得可能的密钥。

对于DES算法，由于DES算法中除S盒外的部分都是简单的矩阵置换和异或运算，所以只需对S盒进行差分分析即可。

S盒接受6位输入，输出4位结果，即输入差分也有26种即0~63，输出差分有24种即0~15。可通过以下代码计算出各个S盒的差分分布表：

*# S盒差分表*

S\_box\_diff\_table = [[[

    [] for i in range(16)

] for j in range(64)

] for x in range(8)]

*# 计算S盒差分表*

def get\_S\_box\_diff\_table():

    for i in range(8):

        for x in range(64):

            for in\_xor in range(64):

                out\_xor = S\_box(i, x) ^ S\_box(i, x ^ in\_xor)

                S\_box\_diff\_table[i][in\_xor][out\_xor].append(x)

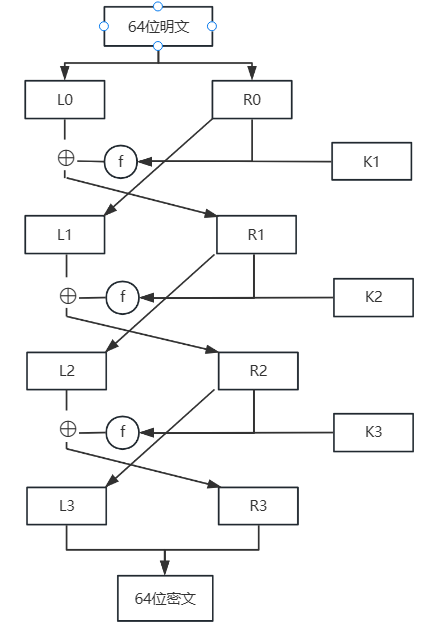
输出差分表中各项的长度。第一列为输入差分，第一排为输出差分，可以明显发现，除了输入差分为0时输出差分必为0外，两者分布并不均匀。例如在第2个S盒中，输入差分为8时，输出差分为10的情况有16种：



由于每一种输入差分对应了64对输入，所以输入差分为8时有1/4的概率得到的输出差分为10。正式因为这样的特性，高轮数的DES差分攻击成为可能。

* **三轮DES差分攻击原理**

三轮DES完整流程图如下：



差分攻击的对象为S盒，因此只需要针对某一轮的加密过程中的S盒操作进行分析即可，其中最后一轮直接暴露的信息最多。下面将x与x’的差分记为x\*：

第三轮S盒的输出B3可表示为(P-1为逆P置换)：

第三轮S盒的输入A3可表示为(E为E拓展)：

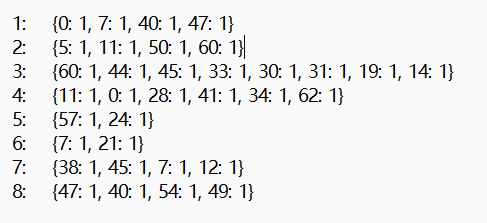
输出差分B3\*可表示为：

输入差分A3\*可表示为：

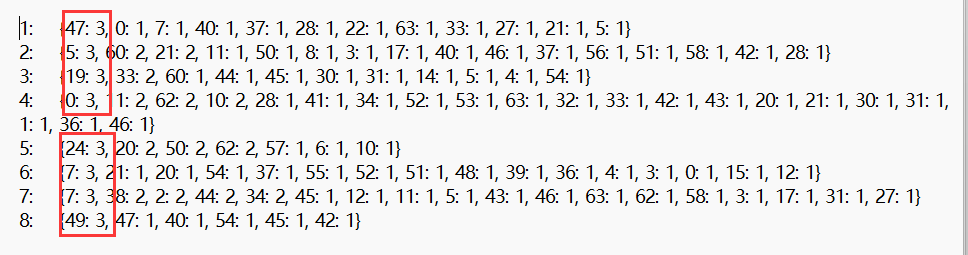
在这两个差分表达式中，L0\*、R3\*、L3\*均为已知项，唯独的未知项为B3\*中f运算部分。虽然这两个表达式的值未知，但是它们的异或结果可以通过选择特定明文来攻击以消除。在三轮DES中，可以使明文对的R0即右32位相同，如此这两个表达式的结果便会相同，异或值为0。B3\*的表达式变为：

因此选择异或值为*xxxxxxxx00000000*的明文对，可以直接计算得到第三轮S盒的输入输出差分，根据每一个S盒输入输出差分查表就能得到在该位置上S盒输入所有可能的值，即所有可能的A3，再进行反推得到key：

使用一组明密文对进行测试，可以看到各个位置可能的轮密钥值(冒号后为出现次数)：



由于三轮DES差分不存在不确定性，所有的明密文对均为有效，所以使用N组明密文对进行统计时，正确的密钥一定会出现N次。使用3组明密文对测试，可以看到各个位置都只有一个值出现了3次，那么该值就是对应位置的轮密钥值。



通过该方式可以获得第三轮轮密钥，该轮密钥有48位，根据DES轮密钥生成过程可以轻易推算出这48位在密钥中的位置，剩余的8位则穷举即可。

## 攻击过程记录

DES攻击过程可以统一为如下流程：

首先计算S盒的差分分布表：

get\_S\_box\_diff\_table()

然后使用多组明密文对进行单轮差分分析：

for i in range(0, pairs << 1, 2):

    DES\_diff\_round(P\_C\_pairs[i][0], P\_C\_pairs[i + 1][0], P\_C\_pairs[i][1], P\_C\_pairs[i + 1][1])

最后对差分分析得到的可能密钥根据出现次数排序，并进行最后的分析与检验得到密钥：

for i in range(8):

    possible\_key[i] = sorted(possible\_key[i].items(), key=lambda x: x[1], reverse=True)

print(analyze('748502CD38451097', '03C70306D8A09F10'))

对于三轮和六轮DES，单轮差分分析和最后归纳检验过程不同，下面分开阐述。

* **三轮DES**

**·单轮差分分析过程如下：**

首先对传入的明密文对进行处理，转为二进制串后进行分割：

*# 一轮差分分析，传入两对明密文对*

def DES\_diff\_round(P1, P2, C1, C2):

    P1 = hex2bin(P1.lower())

    P2 = hex2bin(P2.lower())

    C1 = hex2bin(C1.lower())

    C2 = hex2bin(C2.lower())

    L0, L0\_ = P1[:32], P2[:32]

    R0, R0\_ = P1[32:], P2[32:]

    LN, LN\_ = C1[:32], C2[:32]

    RN, RN\_ = C1[32:], C2[32:]

然后根据前文得到的输入输出差分的计算式，计算出差分：

in\_xor = matrix\_trans(xor\_bin(LN, LN\_), expand\_e)

out\_xor = matrix\_trans(xor\_bin(xor\_bin(L0, L0\_), xor\_bin(RN, RN\_)), inv\_p)

最后进行归纳统计，将所有可能的密钥加入possible\_key对应位置的字典中：

E = matrix\_trans(LN, expand\_e)

for i in range(8):

    idx0 = i << 2

    idx1 = idx0 + (i << 1)

    for input in S\_box\_diff\_table[i][int(in\_xor[idx1:idx1 + 6], 2)][int(out\_xor[idx0:idx0 + 4], 2)]:

        key = input ^ int(E[idx1:idx1 + 6], 2)

        if key in possible\_key[i]:

            possible\_key[i][key] += 1

        else:

            possible\_key[i][key] = 1

**·分析和检验过程如下：**

首先将各个位置的密钥拼接成第三轮的轮密钥，经过排序possible\_key中各项已从字典变为二维列表，列表第一项为出现次数最多的密钥和次数构成的列表，取首项即可：

def analyze(P, C):

    child\_key = ''

    for i in range(8):

        child\_key += bin(possible\_key[i][0][0])[2:].rjust(6, '0')

然后执行矩阵置换2的逆过程，将轮密钥填入密钥对应位置，剩余位置用‘\*’号标记。由于字符串不方便局部更改，所以使用列表：

*# 矩阵变换 循环右移*

    key = ['\*']\*56

    for i in range(48):

        key[perm\_matrix\_after[i] - 1] = child\_key[i]

    mov = move[2]

    key = key[28-mov:28] + key[:28-mov] + key[-mov:] + key[28:56-mov]

将需要填充的位置装入empty列表：

*# 需要填充的位置*

    empty = []

    for i in range(56):

        if key[i] == '\*':

            empty.append(i)

进行28次穷举，每次将当前i值转为8位2进制串依次填入密钥中空白的位置，生成temp\_key后验证正确性：

    temp\_key = ''

    for i in range(1 << 8):

        rand = ''.join(bin(i)[2:].rjust(8, '0'))

        for j in range(8):

            key[empty[j]] = rand[j]

        temp\_key = ''.join(key)

*# print(temp\_key)*

        if DES\_Nround\_test(temp\_key, P, N) == C.lower():

            return bin2hex(temp\_key).upper()

在六轮DES中，计算过程为：

in\_xor = matrix\_trans(xor\_bin(LN, LN\_), expand\_e)

out\_xor = matrix\_trans(xor\_bin(xor\_bin(RN, RN\_), Feature\_bin[k][32:]), inv\_p)

## 攻击结果分析