

武汉大学国家网络安全学院

密码学实验报告

学 号	2021302181156
姓 名	赵伯侯
实验名称	分组密码 DES
指导教师	何琨

一、实验名称: 分组密码 DES

二、实验目的及要求:

2.1 实验目的

- (1) 掌握分组密码的基本概念;
- (2) 掌握 DES (3DES) 密码算法;
- (3) 了解 DES (3DES) 密码的安全性;
- (4) 掌握分组密码常用工作模式及其特点;
- (5) 熟悉分组密码的应用。

2.2 实验要求

- (1) 复习掌握 (古典密码) 使用的置换、代替、XOR、迭代等技术;
- (2) 比较 DES 中代替技术与古典密码中的联系与区别;
- (3) 理解 S 盒、P 置换等部件的安全性准则;
- (4) 实现 DES 算法的编程与优化。

三、实验设备环境及要求:

Windows 操作系统, python 高级语言开发环境

四、实验内容与步骤:

4.1 DES 子密钥扩展算法的实现

子密钥拓展算法是将 64 位密钥经过置换选择 1、循环左移、置换选择 2 等变换，产出 16 个 48 位长的子密钥，子密钥的产生过程如下图所示

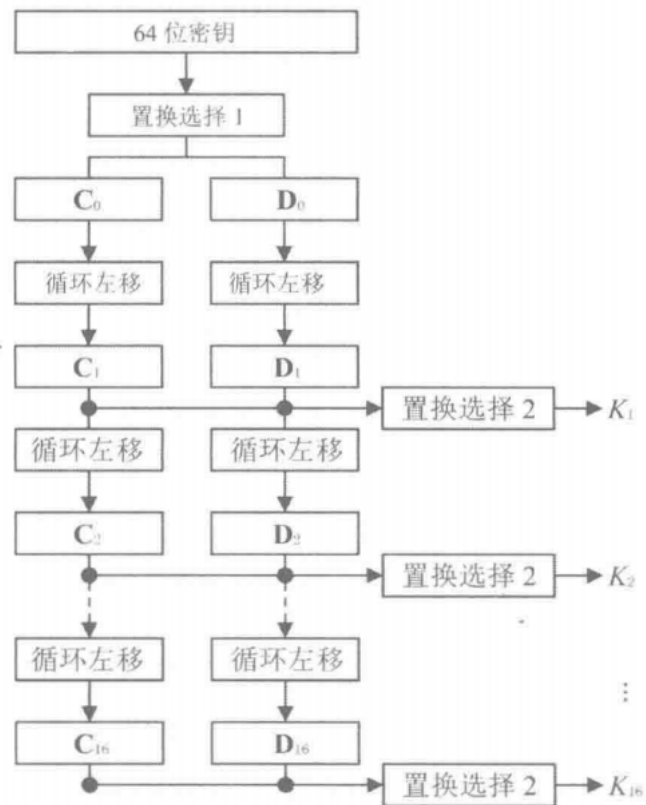


图 1: 子密钥的产生

其中子密钥的生成总体算法如下所示

```
1 def get_keys(key_path):
2     """获得16个子密钥函数"""
3     # 读取密钥文件
4     key = read_file(key_path)
5     # 密钥转换成比特流形式
6     key_bit = key_str_bit(key)
7     # key_bit = '
```

```

    0011000100110010001100110011010000110101001101100011011100111000 '
8     # 密钥置换选择1
9     key1 = PC_1(key_bit)
10    key_c = [[] * 28] * 17
11    key_d = [[] * 28] * 17
12    key_c[0] = key1[0:28]
13    key_d[0] = key1[28:]
14    # 将所有的c和d循环左移得到所有c和d
15    left_table = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
16    for i in range(16):
17        key_c[i + 1] = left_shift(key_c[i], left_table[i])
18        key_d[i + 1] = left_shift(key_d[i], left_table[i])
19
20    # 获取16个子密钥
21    keys = [[] * 48] * 17
22    for i in range(1, 17):
23        key_cx = key_c[i]
24        key_dx = key_d[i]
25        key_cx.extend(key_dx)
26        keys[i] = PC_2(key_cx)
27    return keys

```

代码 1: 子密钥生成代码

4.1.1 置换选择 1

置换选择 1 的作用一是为了从 64 位密钥中去掉 8 个奇偶校验位；二是为了把其余的 56 位密钥位打乱顺序重新排列，并且将前 28 位作为 C0，后 28 位作为 D0，其中置换选择 1 使用的置换矩阵如下图所示

C ₀								D ₀							
57	49	41	33	25	17	9		63	55	47	39	31	23	15	
1	58	50	42	34	26	18		7	62	54	46	38	30	22	
10	2	59	51	43	35	27		14	6	61	53	45	37	29	
19	11	3	60	52	44	36		21	13	5	28	20	12	4	

图 2: 置换选择 1 矩阵

该步骤的代码如下所示

```
1 def PC_1(input):
2     """置换选择 I"""
3     box = [57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,
4            10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,
5            63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,
6            14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4]
7
8     result = [3 for i in range(56)]
9     for index in range(0, 56):
10         result[index] = input[box[index] - 1]
11     return result
```

代码 2: 置换选择 1 代码

4.1.2 循环左移

循环左移是在每一轮次的子密钥生成过程中对 c_{n-1} 和 d_{n-1} 按照循环左移位数组中的值进行循环左移，得到 c_n 和 d_n ， n 为当前轮次。循环左移位数组如下图所示

迭代次数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
循环左移位	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

图 3: 循环左移位数组

该步骤的代码如下图所示

```
1 def left_shift(c, times):
2     """循环左移一定位数"""
3     d = c.copy()
4     for index in range(times):
5         d.insert(len(d), d[0])
6         d.remove(d[0])
7     return d
```

代码 3: 循环左移代码

4.1.3 置换选择 2

置换选择 2 是将 c_n 和 d_n 合并成一个 56 位的中间数据后按照置换选择 2 矩阵从中选择出一个 48 位的子密钥 K_n ，置换选择 2 的矩阵如下图所示

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

图 4: 置换选择 2 矩阵

该步骤的代码如下所示

```
1 def PC_2(input):
2     """置换选择2"""
3     box = [14, 17, 11, 24, 1, 5,
4            3, 28, 15, 6, 21, 10,
5            23, 19, 12, 4, 26, 8,
6            16, 7, 27, 20, 13, 2,
7            41, 52, 31, 37, 47, 55,
8            30, 40, 51, 45, 33, 48,
9            44, 49, 39, 56, 34, 53,
10           46, 42, 50, 36, 29, 32]
11
12     result = [3 for i in range(48)]
13     for index in range(0, 48):
14         result[index] = input[box[index] - 1]
15     return result
```

代码 4: 置换选择 2 代码

4.2 DES 局部加密函数 f 的实现

局部加密函数的作用是在第 i 次加密迭代中用子密钥 k_i 对 R_{i-1} 进行加密，首先进行选择运算 E 对 32 位的 R_{i-1} 的各位进行选择 and 排列，产生一个 48 位的结果，然后将得到的结果与子密钥 k_i 进行异或操作，然后送入整体 S 盒中，产生 32 位的数据组，经过置换运算 P 将各位打乱重拍后得到加密函数的密文输出

该加密函数 f 算法流程图如下图所示

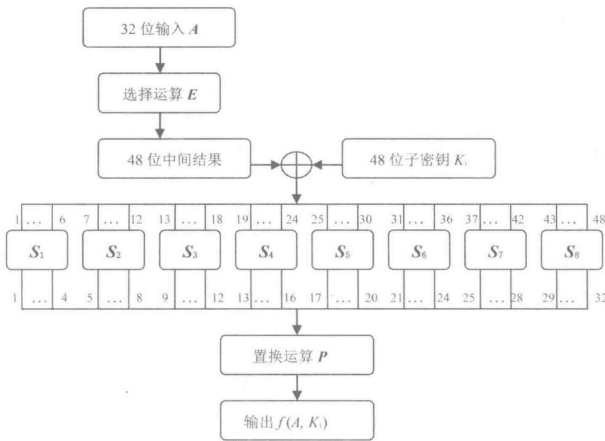


图 5: 加密函数 f

该加密函数的代码如下所示

```
1 def f(R, k):
2     """加密函数"""
3     # 选择运算
4     E = PC_E(R)
5     # 与子密钥异或
6     S_input = xor(E, k)
7     # 输入到S盒运算
8     S_output = S(S_input)
9     # 置换运算
10    result = PC_P(S_output)
11    return result
```

代码 5: 加密函数 f 代码

4.2.1 扩展置换 E

选择运算 E 对 32 位的数据 A 的各位进行重复选择某些数据产生一个 48 位的结果，以达到数据拓展的目的，其中选择运算 E 的矩阵如下图所示

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

图 6: 拓展置换矩阵

进行拓展置换的程序代码如下所示

```
1 def PC_E(input):
2     box = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13, 12, 13,
3           14, 15, 16, 17,
4           16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28,
5           29, 28, 29, 30, 31, 32, 1]
6     result = [3 for i in range(48)]
7     for index in range(0, 48):
8         result[index] = input[box[index] - 1]
9     return result
```

代码 6: 拓展置换函数 E 代码

4.2.2 异或操作

异或操作是针对输入的两个矩阵的每一位进行异或操作后输出，其程序代码如下所示

```
1 def xor(a, b):
2     """矩阵异或运算"""
3     c = []
4     for index in range(len(a)):
5         c.append(int(a[index]) ^ int(b[index]))
6     return c
```

代码 7: 异或操作代码

4.2.3 代替 S 盒

代替 S 盒由 8 个 S 盒组成，对于每一个 S 盒有 6 位输入，产生 4 位的输出，因此整体的 S 盒接收 48 位输入数据，返回 32 位的输出，在一个单个的 S 盒中，将 6 位输入的首尾两位作为选中的行号，将中间 4 位作为列号，在当前的 S 盒矩阵中进行查找，将查找到的结果以 4 位 2 进制的形式进行输出。S 盒的代替矩阵如下图所示

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

图 7: S 盒代替矩阵

进行 S 盒运算的整体代替代码如下所示

```
1 def S(input):
2     """整体S盒"""
3     output = []
4     for j in range(0, 48, 6):
5         s_after = sk(input[j:j + 6], int(j / 6))
6         for x in s_after:
7             output.append(x)
8     return output
```

代码 8: S 盒整体代码

单个 S 盒的运算代码如下所示

```
1 def sk(input, i):
2     """单个s盒"""
3     s1 = [[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
4           [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
5           [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
6           [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]]
7     s2 = [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
8           [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
9           [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
10          [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]]
11     s3 = [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
12           [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
13           [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
14           [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]]
15     s4 = [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
16           [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
17           [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
18           [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]]
```

```

19 s5 = [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
20       [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
21       [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
22       [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]]
23 s6 = [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
24       [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
25       [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
26       [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]]
27 s7 = [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
28       [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
29       [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
30       [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]]
31 s8 = [[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
32       [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
33       [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
34       [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]
35 s = [s1, s2, s3, s4, s5, s6, s7, s8]
36
37 row = int(str(input[0]) + str(input[5]), 2)
38 col = int(str(input[1]) + str(input[2]) + str(input[3]) + str(input[4]), 2)
39 num = bin(s[i][row][col])[2:] # i-1号S盒的row*16+col号数
40 for index in range(4 - len(num)): # 补齐4位后输出
41     num = '0' + num
42 return num

```

代码 9: 单个 S 盒整体代码

4.2.4 置换运算 P

置换运算 P 将 S 盒输出的 32 位数据进行打乱重排，得到 32 位的加密函数输出，从而达到将 S 盒的混淆作用扩散的目的。

进行置换运算 P 的代码如下所示

```
1 def PC_P(input):  
2     box = [16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,  
3           2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25]  
4     result = [0 for i in range(32)]  
5     for index in range(0, 32):  
6         result[index] = input[box[index] - 1]  
7     return result
```

代码 10: 置换运算 P

4.3 DES 加密过程完整实现

DES 的整个加密过程用流程图表示如下图所示

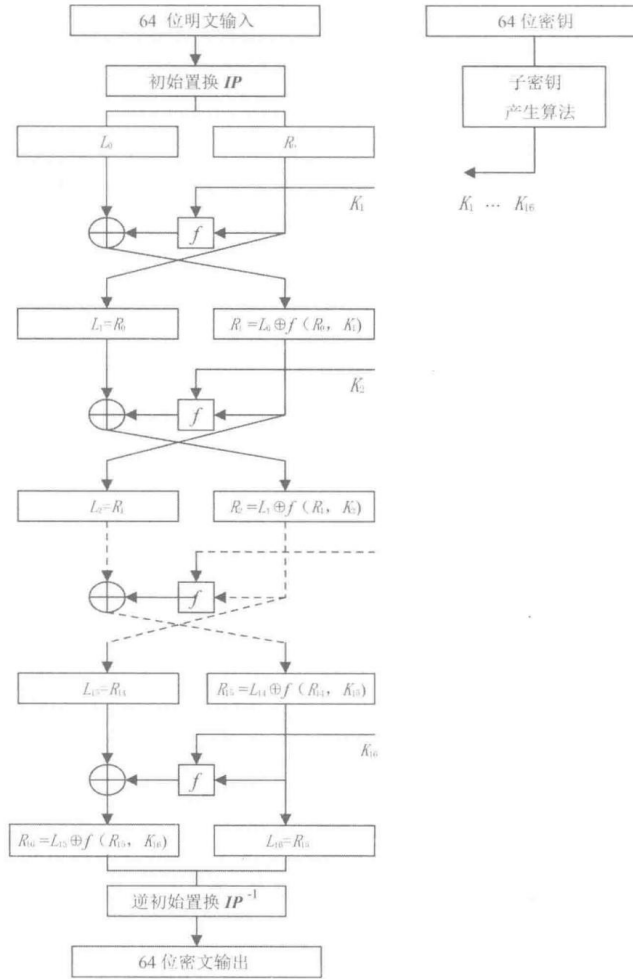


图 8: DES 整体流程图

整个 DES 加密过程用数学公式表示

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f(R_{i-1}, k_i) \\ i = 1, 2, 3, \dots, 16 \end{cases}$$

4.3.1 子密钥产生算法

64 位密钥经子密钥产生算法产生出 16 个子密钥： K_1, K_2, \dots, K_{16} ，分别供第一次，第二次，...，第十六次加密迭代使用。

4.3.2 初始置乱 IP

64 位明文首先经过初始置换 IP（Initial permutation），将数据打乱重新排列并分成左右两半。左边 32 位构成 L_0 ，右边 32 位构成 R_0

4.3.3 初次迭代

由加密函数 f 实现子密钥 K_1 对 R_0 的加密，结果为 32 位的数据组 $f(R_0, K_1)$ ，再与 L_0 进行异或运算，又得到一个 32 位的数据组 $L_0 \oplus f(R_0, K_1)$ 。以 $L_0 \oplus f(R_0, K_1)$ 作为第二次加密迭代的 R_1 ，以 R_0 作为第二次加密迭代的 L_1 。至此，第一次加密迭代结束。

4.3.4 16 次加密迭代

第二次加密迭代至第十六次加密迭代分别用子密钥 K_2, \dots, K_{16} 进行，其过程与第一次加密迭代相同。

4.3.5 逆初始置换 IP^{-1}

第十六次加密迭代结束后，产生一个 64 位的数据组。以 R_{16} 作为其左边 32 位，以 L_{16} 作为其右边 32 位，两者合并再经过逆初始置换 IP^{-1} ，将数据重新排列，便得到 64 位密文。至此加密过程全部结束。

整个 DES 加密函数的代码如下所示

```
1 def encode(message_bit):
2     """加密"""
3     # 初始置换
4     message_PC = PC_IP(message_bit)
5     # 取得L0和R0
6     Ls = [[] * 32] * 17
7     Rs = [[] * 32] * 17
8     Ls[0] = message_PC[0:32]
9     Rs[0] = message_PC[32:]
10
11    # 加密迭代
12    for i in range(16):
13        Ls[i + 1] = Rs[i]
14        Rs[i + 1] = xor(Ls[i], f(Rs[i], keys[i + 1]))
15    # 逆初始置换
16    LandR = Rs[16] + Ls[16]
17    result = PC_IP_1(LandR)
18    return result
```

代码 11: DES 加密主函数

4.4 DES 解密过程实现

由于 DES 的运算是对和运算，所以解密和加密可共用同一个运算，只是子密钥使用的顺序不同。把 64 位密文当作明文输入，而且第一次解密迭代使用子密钥 K_{16} ，第二次解密迭代使用子密钥 K_{15} ，第十六次解密迭代使用子密钥 K_1 ，最后的输出便是 64 位明文。解密过程可用如下的数学公式描述：

$$\begin{cases} R_{i-1} = L_i \\ L_{i-1} = R_i \oplus f(R_{i-1}, k_i) \\ i = 16, 15, 14, \dots, 1 \end{cases}$$

解密过程的代码如下所示

```
1 def decode(secret):
2     """解密"""
3     keys = get_keys("key.txt")
4
5     message_PC = PC_IP(secret)
6     Ls = [[] * 32] * 17
7     Rs = [[] * 32] * 17
8     Ls[0] = message_PC[0:32]
9     Rs[0] = message_PC[32:]
10
11     for i in range(16):
12         Ls[i + 1] = Rs[i]
13         Rs[i + 1] = xor(Ls[i], f(Rs[i], keys[16 - i])) # 密钥从16开始使用
14
15     LandR = Rs[16] + Ls[16]
16     result = PC_IP_1(LandR)
17     letters = ""
18
19     for i in range(8):
```

```

20     bits = ""
21     for j in range(8):
22         bits += str(result[i * 8 + j])
23     letter = chr(int(bits, 2))
24     letters += letter
25
26     return letters

```

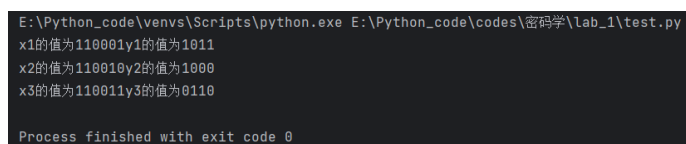
代码 12: DES 解密主函数

4.5 DES 的 S 盒密码学特性

通过编程实现或者手工计算，试验证 S 盒的以下准则：定义 S 盒的输入为 X，输出为 Y（X 和 Y 都以二进制表示）

4.5.1 输出不是输入的线性和仿射函数

（1）首先证明线性函数，令 $x_1 = 110001$ $x_2 = 110010$ $x_3 = 110011$ ，使用 S 盒 1，分别求得其输出值如下图所示



```

E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110001y1的值为1011
x2的值为110010y2的值为1000
x3的值为110011y3的值为0110
Process finished with exit code 0

```

图 9: 验证线性函数运行结果

得出的 $y_1 = 1011$, $y_2 = 1000$, $y_3 = 0110$, x 的值每位加 1 递增，得到的 y 的值却并不存在线性关系，由此可以证明 S 盒的输出不是线性函数

（1）接下来证明仿射函数，令 $x_1 = 0b110010 = 0d2$, $x_2 = 0b110100 = 0d4$, $z_1 = \alpha x_1 + \beta x_2$, $z_2 = \alpha x_1$, $z_3 = \beta x_2$, $\alpha = 0.5$, $\beta = 0.5$ ，求得 $y_1 = f(z_1)$, $y_2 = f(z_2)$, $y_3 = f(z_3)$ 的结果如下图所示

```

E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110010x2的值为110100
z1的值为110011y1的值为0110
z2的值为110001y2的值为1011
z2的值为110001y3的值为1000
Process finished with exit code 0

```

图 10: 验证仿射函数运行结果

根据计算出的结果 $y_1 = 0b0110, y_2 = 0b1011, y_3 = 0b1000$, 可以得出结论 $y_1 \neq y_2 + y_3$, 与仿射函数的定义相矛盾, 所以 S 盒不是仿射函数。

4.5.2 任意改变输入中的一位, 输出至少有两位发生变化

令 $x_1 = 110010, x_2 = 100010$, 使用 S 盒 2, 分别求得其输出值 y_1, y_2 并对 y_1, y_2 进行异或运算, 得到的结果如下图所示

```

E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110010 x2的值为100010
y1的值为 1000 y2的值为1110
y1与y2异或运算的结果为:
[0, 1, 1, 0]
Process finished with exit code 0

```

图 11: 验证修改一位输入后的输出变化的运行结果

通过观察结果发现, 异或运算的结果为 0110, 输出有两位发生变化, 得证。

4.5.3 对于任何 S 盒和任何输入 x , $S(x)$ 和 $S(x \oplus 001100)$ 至少有两位不同, 这里 x 是一个 6 位的二进制串

令 $x_1 = 110010, x_2 = x_1 \oplus 001100 = 111110$ 分别求得其输出值 y_1, y_2 并对 y_1, y_2 进行异或运算, 得到的结果如下图所示

```

E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110010 x2的值为111110
y1的值为 1000 y2的值为1111
y1与y2异或运算的结果为:
[0, 1, 1, 1]
Process finished with exit code 0

```

图 12: $x \oplus 001100$ 后的输出变化的运行结果

通过观察结果发现, 异或运算的结果为 0111, 输出有 3 位发生变化, 得证。

4.5.4 对于任何 S 盒和任何输入 \mathbf{x} ，以及 $y, z \in GF(2)$, $S(\mathbf{x}) \neq S(\mathbf{x} \oplus 11yz00)$ ，这里 \mathbf{x} 是一个 6 位的二进制串

令 $x_1 = 110010, x_2 = x_1 \oplus 110000, x_3 = x_1 \oplus 110100, x_4 = x_1 \oplus 111000, x_5 = x_1 \oplus 111100$ 分别求得其输出值 y_1, y_2, y_3, y_4, y_5 并对 y_1 与其余四个输出结果进行异或运算，得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110010 x2的值为110000x3的值为110100 x4的值为111000 x5的值为111100
y1的值为 1000 y2的值为0101 y3的值为1100 y4的值为1001 y5的值为0010
y1与y2异或运算的结果为:
[1, 1, 0, 1]
y1与y3异或运算的结果为:
[0, 1, 0, 0]
y1与y4异或运算的结果为:
[0, 0, 0, 1]
y1与y5异或运算的结果为:
[1, 0, 1, 0]
Process finished with exit code 0
```

图 13: $x \oplus 11yz00$ 后的输出变化的运行结果

通过观察结果发现， y_1 与其余四个输出结果进行异或运算的结果分别为 1101，0100，0001，1010，均不为全 0，得证。

4.5.5 保持输入中的 1 位不变，其余 5 位变化，输出中的 0 和 1 的个数接近相等。

令 $x_1 = 110010, x_2 = 101101$ 分别求得其输出值 y_1, y_2 并统计 0 和 1 的个数得到的结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\lab_1\test.py
x1的值为110010 x2的值为101101
y1的值为 1000 y2的值为0100
Process finished with exit code 0
```

图 14: 改变输入中的 5 位后的输出变化的运行结果

通过观察结果发现， y_1 的值为 1000， y_2 的值为 0100，两个输出的 0 和 1 的个数相同，得证。

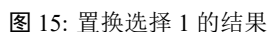
(1) 非线性变换: S 盒是 DES 中的一个非线性变换, 它将输入的 6 位数据映射到 4 位输出。这种非线性性质增加了密码算法的复杂性, 使得密码分析变得更加困难。如果 S 盒是线性的, 那么 DES 的安全性将大大降低, 因为线性变换容易受到线性密码分析的攻击。

(3) 防止经典攻击：S 盒设计的目标之一是抵御一些经典的密码分析攻击，例如差分密码分析和线性密码分析。合理设计的 S 盒可以使这些攻击变得非常困难，要求攻击者大量的计算和数据。

(4) 增加密钥空间：DES 中有 8 个不同的 S 盒，每个 S 盒都有自己的置换表。这些 S 盒的组合扩展了 DES 的密钥空间，使得暴力攻击变得更加困难，因为攻击者需要尝试大量的密钥组合。

因为在教材中的密钥并未涉及到将第 8 位做奇偶校验，所以在验证的过程中直接将给出的密钥作为已经处理好的密钥二进制数据进行验证。

将给出的密钥输入程序后进行调试，置换选择 1 的结果保存在变量 `key1` 中，`key1` 的值如下图所示，与教材中给出的置换选择 1 的值相同。



第 1 轮循环得到的结果如下图所示，其中，F 函数的 32 位输入保存在变量 R 中，选择运算的结果保存在变量 E 中，子密钥 k1 保存在变量 k 中，子密钥加后的结果保存在变量 S_input 中，S 盒输出的结果保存在变量 S_output 中，P 置乱后的结果保存在返回值 result 中，与教材中给出的值相同

```
> E = (list: 48) ['0','0','0','0','0','0','0','0','0','0','0','0','1','1','1','1','1','1','1','1','0','0','0','0','0','0','0','0','1','1','1','1','1','1','1','1','0','0','0','0','0','0','0','0']
> R = (list: 32) ['0','0','0','0','0','0','0','0','1','1','1','1','1','1','1','1','0','0','0','0','0','0','0','0','1','1','1','1','1','1','1','1','0','0','1','0','0','1','0','1']
> S_input = (list: 48) [0,1,0,1,0,0,0,0,0,0,1,1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,0,1,1,0,0,1,0,0,1,0,0,1,0,1,0]
> S_output = (list: 32) ['0','1','1','0','1','1','0','1','1','0','0','1','0','0','1','0','1,1,0,0,1,1,1,0,0,1,1,0,0,1,0,0,1,0,0,1,0,1,0]
> k = (list: 48) ['0','1','0','1','0','0','0','0','0','0','0','1','0','1','0','1','0','0','1','0','1','0','1','0','1','0','1','0','1','0','1','1','1','0','1','0','1','1','0','1','0','1,1,0]
> result = (list: 32) ['0','0','0','0','1','0','0','1','1','0','0','1','1','1','0','0','1','1','1','0','1,1,0,1,1,0,1,1,0,1,1,0,1,0,1,0,0,1,1,0]
```

图 22: 第 1 轮循环的结果

第 2 轮循环后得到的结果如下图所示，与教材中给出的值相同

```
> E = (list: 48) [1,0,0,0,1,0,1,0,0,1,0,1,0,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,0,1,0,1,1,1,1,0,1,1,0,1,0,0,1,1,0]
> R = (list: 32) [0,0,0,1,0,0,1,0,1,0,0,0,0,1,1,0,0,0,1,1,0,1,1,0,1,1,0,1,0,0,1,1]
> S_input = (list: 48) [1,1,0,1,1,0,1,0,1,1,1,1,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,0,1,1,0,1,1,0,0,0,0,1]
> S_output = (list: 32) ['0','1','1','1','0','0','1','0','1','0','1','1','0','1','1','0','1','1','0','1,1,0,1,1,0,1,1,0,0,0,0,1]
> k = (list: 48) ['0','1','0','1','0','0','0','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','1','1','0','1','0','1','1,0]
> result = (list: 32) ['1','1','1','0','0','0','1','0','1','0','1','1','0','1','0','1','0','1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0]
```

图 23: 第 2 轮循环的结果

第 3 轮循环后得到的结果如下图所示，与教材中给出的值相同

```
> E = (list: 48) [0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,1,1,1,1,0,0,1,0,0,0,0,0,0,1,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1]
> R = (list: 32) [1,1,1,0,0,0,0,1,1,0,0,1,1,1,0,0,1,0,0,0,0,1,1,0,1,0,0,0,1,0,1,0]
> S_input = (list: 48) [1,0,1,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,1,1,1,1,1,0,1,1,0,1,0,1,0,0,0,0,1,1,0,1,1,0,0,1]
> S_output = (list: 32) ['1','1','0','1','1','1','0','1','1','0','1','1','0','1','0','1','0','1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,0,1]
> k = (list: 48) ['1','1','0','1','0','0','0','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','1','1,0]
> result = (list: 32) ['1','1','0','0','0','1','0','1','0','1','0','1','1','0','1','0','1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0]
```

图 24: 第 3 轮循环的结果

第 4 轮循环后得到的结果如下图所示，与教材中给出的值相同

```
> E = (list: 48) [1,1,1,0,1,0,1,0,1,1,0,0,0,0,0,1,0,1,0,1,1,1,1,0,1,0,1,1,1,0,1,0,1,1,0,0,0,0,1,0,1,1]
> R = (list: 32) [1,1,0,1,0,1,1,0,0,0,1,0,1,1,1,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,0,1]
> S_input = (list: 48) [0,0,0,0,1,0,1,0,0,1,1,0,0,1,1,1,0,1,1,0,1,1,0,0,1,0,1,1,0,1,1,0,0,0,0,0,0,0,0]
> S_output = (list: 32) ['0','1','0','0','1','0','1','1','1','0','1','1','0','1','1','0','1','1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1]
> k = (list: 48) ['1','1','1','0','0','0','0','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1,1,0]
> result = (list: 32) ['1','1','1','1','1','1','1','1','1','1','1','1','0','1','1','1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
```

图 25: 第 4 轮循环的结果

第5轮循环后得到的结果如下图所示,与教材中给出的值相同

```
> E = (list: 48) [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0]
> R = (list: 32) [0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0]
> S_input = (list: 48) [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1]
> S_output = (list: 32) ['0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '1', '0', '1', '0', '1', '0']
> k = (list: 48) ['1', '1', '1', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '1', '0', '1', '0', '0', '0', '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '1', '0']
> result = (list: 32) ['1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '1', '1', '0', '0', '1', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0']
```

图 26: 第 5 轮循环的结果

第 6 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 27: 第 6 轮循环的结果

第7轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 28: 第 7 轮循环的结果

第8轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 29: 第 8 轮循环的结果

第9轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 30: 第 9 轮循环的结果

第 10 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 31: 第 10 轮循环的结果

第 11 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 32: 第 11 轮循环的结果

第 12 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 33: 第 12 轮循环的结果

第 13 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 34: 第 13 轮循环的结果

第 14 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 35: 第 14 轮循环的结果

第 15 轮循环后得到的结果如下图所示，与教材中给出的值相同

```
> E = (list: 48) [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0]
> R = (list: 32) [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0]
> S_input = (list: 48) [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1]
> S_output = (list: 32) ['0', '1', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0']
> K = (list: 48) ['0', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0', '1', '1', '0', '1', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '1', '1', '1']
> result = (list: 32) ['0', '0', '1', '0', '1', '1', '0', '1', '0', '1', '0', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '1', '1', '0', '1', '1', '1', '0', '1', '1', '1']
```

图 36: 第 15 轮循环的结果

第 16 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 37: 第 16 轮循环的结果

继续调试程序得到所有的 L 的结果如下图所示，与教材中给出的值一致

[illegible]

图 38: L 的值

继续调试程序得到所有的 R 的结果如下图所示，与教材中给出的值一致

[illegible]

图 39: R 的值

继续调试函数得到逆初始置换的结果即密文如下图所示

```
> result = {list: 64} [1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1]
```

图 40: 64 位密文

4.6.3 解密过程

教材中给出的密文序列是机密过程产生的密文，在程序中使用该序列作为密文，继续使用密钥拓展过程的密钥作为本次实验的密钥。

调试程序得到初始置换后的结果如下图所示，与教材中给出的值相同

```
> message_PC = (list: 64) [1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
```

图 41: 初始置换后结果

调试程序得到 L0 和 R0 的结果如下图所示,与教材中给出的相同

```
> Ls = {list: 17} [[1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
> Rs = {list: 17} [[0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
```

图 42: 得到 L0 和 R0 的结果

第 1 轮循环得到的结果如下图所示，其中，F 函数的 32 位输入保存在变量 R 中，选择运算的结果保存在变量 E 中，子密钥 k1 保存在变量 k 中，子密钥加后的结果保存在变量 S_input 中，S 盒输出的结果保存在变量 S_output 中，P 置乱后的结果保存在返回值 result 中，与教材中给出的值相同

[illegible]

图 43: 第 1 轮循环的结果

第2轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 44: 第 2 轮循环的结果

第3轮循环后得到的结果如下图所示,与教材中给出的值相同

```
> E = (list: 48) [1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0]
> R = (list: 32) [0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1]
> S_input = (list: 48) [1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0]
> S_output = (list: 32) ['O', 'O', 'O', 'I', 'I', 'I', 'I', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'I', 'O', 'I', 'O', 'O', 'I', 'I', 'O', 'I', 'O', 'I']
> k = (list: 48) ['O', 'O', 'O', 'I', 'I', 'I', 'I', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'O', 'I', 'O', 'O', 'I', 'I', 'O', 'I', 'O', 'O', 'I', 'I', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I']
> result = (list: 32) ['I', 'I', 'I', 'I', 'O', 'I', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'I', 'O', 'O', 'O', 'O', 'I', 'O', 'O', 'O', 'O', 'O', 'I', 'O', 'I', 'O', 'O', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I', 'O', 'I']
```

图 45: 第 3 轮循环的结果

第4轮循环后得到的结果如下图所示,与教材中给出的值相同

```
> E = (list: 48) [0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1]
> R = (list: 32) [1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0]
> S_input = (list: 48) [0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0]
> S_output = (list: 32) ["1","0","0","1","1","1","0","0","0","0","1","0","1","0","1","0","0","1","0","0","1","1","1","1","0","1","1","1","0","0","1","1","0","0","0"]
> k = (list: 48) ["0","0","1","0","1","1","1","1","0","1","0","0","0","1","1","0","0","0","1","1","0","0","0","0","1","0","0","1","0","1","1","1","0","1","1","0","1","0","1"]
> result = (list: 32) ["0","0","1","1","1","1","0","1","0","0","1","1","0","1","0","1","0","0","1","0","1","0","0","1","1","0","1","0","1","0","1","0","1","0","1"]
```

图 46: 第 4 轮循环的结果

第5轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 47: 第 5 轮循环的结果

第 6 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 48: 第 6 轮循环的结果

第 7 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 49: 第 7 轮循环的结果

第 8 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 50: 第 8 轮循环的结果

第9轮循环后得到的结果如下图所示,与教材中给出的值相同

[illegible]

图 51: 第 9 轮循环的结果

第 10 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 52: 第 10 轮循环的结果

第 11 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 53: 第 11 轮循环的结果

第 12 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 54: 第 12 轮循环的结果

第 13 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 55: 第 13 轮循环的结果

第 14 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 56: 第 14 轮循环的结果

第 15 轮循环后得到的结果如下图所示, 与教材中给出的值相同

[illegible]

图 57: 第 15 轮循环的结果

第 16 轮循环后得到的结果如下图所示，与教材中给出的值相同

[illegible]

图 58: 第 16 轮循环的结果

继续调试程序得到所有的 L 的结果如下图所示，与教材中给出的值一致

```
> ls = (list: 17) [[1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1], [0, 0]
> i0 = (list: 32) [1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1]
> i1 = (list: 32) [0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1]
> i2 = (list: 32) [0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0]
> o3 = (list: 32) [0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1]
> i4 = (list: 32) [1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0]
> i5 = (list: 32) [0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0]
> i6 = (list: 32) [0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1]
> i7 = (list: 32) [1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
> o8 = (list: 32) [0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0]
> i9 = (list: 32) [1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0]
> i10 = (list: 32) [0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0]
> i11 = (list: 32) [0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0]
> i12 = (list: 32) [0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0]
> i13 = (list: 32) [1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
> i14 = (list: 32) [1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0]
> i15 = (list: 32) [0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1]
> i16 = (list: 32) [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
```

图 59: L 的值

继续调试程序得到所有的 R 的结果如下图所示，与教材中给出的值一致

```

> is = {list: 17} [[0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1], [0, 0,
> is00 = {list: 32} [0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1]
> is01 = {list: 32} [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0]
> is02 = {list: 32} [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1]
> is03 = {list: 32} [1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0]
> is04 = {list: 32} [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0]
> is05 = {list: 32} [0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1]
> is06 = {list: 32} [1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0]
> is07 = {list: 32} [0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0]
> is08 = {list: 32} [1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0]
> is09 = {list: 32} [0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0]
> is10 = {list: 32} [0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0]
> is11 = {list: 32} [0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0]
> is12 = {list: 32} [1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1]
> is13 = {list: 32} [1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]
> is14 = {list: 32} [0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1]
> is15 = {list: 32} [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0]
> is16 = {list: 32} [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0]

```

图 60: R 的值

继续调试函数得到逆初始置换的结果即密文如下图所示

```
> result = {list: 64} [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1]
```

图 61: 64 位明文

4.7 扩展思考

4.7.1 Feistel 结构为什么可以保证算法的对合性？

(1) 轮函数可逆，Feistel 结构中的轮函数是一个可逆函数，它对明文的处理是可逆的。这意味着无论是加密过程还是解密过程，轮函数的应用都可以反转。在解密过程中，只需按相反的顺序应用轮函数即可恢复原始明文。

(2) 轮函数对明文两半的独立处理：Feistel 结构中的轮函数对明文的两半进行独立处理，因此在解密时，只需对每一轮的结果进行逆操作，并将左半部分和右半部分进行交换，就可以得到原始明文。

(3) 多轮迭代：Feistel 结构通常包括多轮迭代，每一轮都应用可逆的轮函数。这使得解密过程需要反复应用逆操作，以确保对合性。

4.7.2 第 16 轮为什么不做左右互换？

(1) 简化最终置换操作：DES 算法的最后一步是一个固定的 IP 逆置换。如果在第 16 轮后进行左右互换，那么最终置换必须考虑这个交换的存在，从而增加了计算复杂度。不进行第 16 轮的左右互换，可以使得最终置换直接应用于最后一轮输出，从而简化了算法。

(2) 保持加密和解密过程的对合性：在 DES 算法中，加密和解密过程是对称的。如果在加密过程的第 16 轮执行了左右互换，则在解密过程的第一轮也必须进行相同的交换，以保持算法的对合性。由于解密时各轮的操作与加密过程是逆序的，不进行第 16 轮的左右互换可以使得加密和解密过程更加一致，从而简化了实现。

4.7.3 如果去掉初始置换和逆初始置换，对算法安全性有影响吗？（提示：算法所有的细节都是公开的）

对算法的安全性没有影响，原因是（1）置换是线性的且无密钥：初始置换和逆初始置换都是简单的线性操作，它们仅仅重新排列了输入和输出比特的位置。这些置换操作不涉及任何密钥材料，而且算法的细节公开，使得攻击者也可以进行置

换操作，因此并不增加对密钥的保护。

(2) 不影响加密的混淆和扩散属性：DES 算法的安全性主要依赖于其 16 轮的复杂变换，包括扩展置换、S 盒、P 盒及密钥调度算法。这些操作共同实现了加密过程中必需的混淆和扩散属性，而初始置换和逆初始置换并不对这些属性作出贡献。

4.7.4 证明 DES 解密算法是加密算法的逆，即 DES 的对合性。

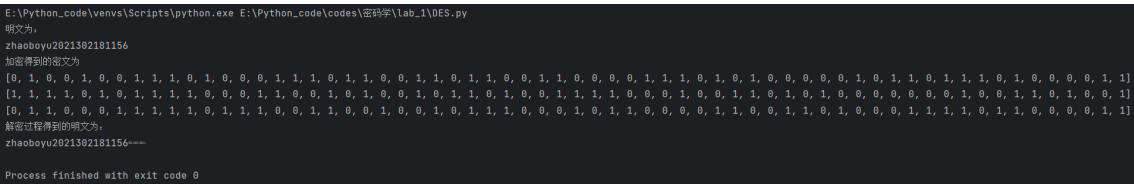
证明 DES 对合性的步骤如下图所示

令 L, R 分别为 64 位数据的左右两半，操作 T 是将 L, R 左右置换
 F 为对应的局部加密函数，
 IP 操作为初始置换， IP^{-1} 为逆初始置换
 $\therefore T(T(L, R)) = T(R, L) = L, R$
 $\therefore T$ 为对合运算
 $\therefore F_i F_i(L_{i-1}, R_{i-1}) = F_i(L_{i-1} \oplus f(R_{i-1}, k_i), R_{i-1})$
 $= (L_{i-1} \oplus f(R_{i-1}, k_i) \oplus f(R_{i-1}, k_i), R_{i-1})$
 $= (L_{i-1}, R_{i-1})$
 $\therefore F$ 也是对合运算
DES 过程可写为：
 $DES = IP^{-1} F_{16} T F_{15} T F_{14} \dots T F_2 T F_1 IP(M)$
DES 的解密过程可写为
 $DES^{-1} = IP^{-1} F_1 T F_2 \dots T F_{15} T F_{16} IP(DES)$
 $= IP^{-1} F_1 T F_2 \dots T F_{15} T F_{16} IP \cdot IP^{-1} F_{16} T F_{15} \dots T F_2 T F_1 IP(M)$
由于 IP, F_i, T 均为对合运算
 $DES^{-1} \cdot DES = M$
 $\therefore DES$ 也是对合运算

图 62: DES 对合性证明

五、实验结果与数据处理:

将想要加密的明文字符串保存到 message.txt 文件中，并将本次加密使用的 8 个字符串的密钥放到 key.tet 文件中，在本次结果演示中，使用的明文为姓名加学号：zhaoboyu2021302181156, 使用的密钥为姓名:zhaoboyu 程序运行加密解密得到的结果如下图所示



```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\密码学\Lab_1\DES.py
明文为:
zhaoboyu2021302181156
加密得到的密文为
[0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1]
[1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1]
[0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1]
解密过程得到的明文为:
zhaoboyu2021302181156---
Process finished with exit code 0
```

图 63: 程序运行结果

观察结果可以发现，明文被分成 3 组进行加密，加密得到的密文不仅有明文的内容，其后还跟随着由于最后一组不足 64 位而补齐的 0

六、分析讨论:

- (1) 在本次实验中实现了 DES 加密解密的程序，能够接收一个任意长度的明文字符串以及一个 8 个字符长度的密钥字符串进行加密操作，得到密文，然后根据得到的密文进行解密操作，得到明文。
- (2) 在本次实验中学习到了 S 盒在 DES 加密过程中做出的贡献即混淆和扩散的作用，使得 DES 加密算法能够抵抗一定的典型攻击。
- (3) 在本次实验中并未进行代码运行时间方面的设计，因此代码运行的时间以及需要消耗的空间资源可能会比较大。

七、教师评语: