DES 算法实现
实验报告

姓名：宋晓彤

学号：16340192

方向：嵌入式软件与系统

2018. 10. 18

一、 原理概述

DES 算法是一种对称加密算法，明文和密钥均由 64 位二进制码构成，其中，密钥每个字节的最后一位为奇偶校验位，保证一个字节中 1 的个数为奇数，在实际加密过程中没有起到作用。

DES 算法通过置换、移位等操作，将原始密钥转化为 16 个子密钥，进而与置换后的明文进行异或、S 盒选择、P 盒置换等操作，再次交换、置换后产生与原始明文完全不同的输出，从而完成加密过程。解密过程与加密过程近似，不同在于应调换 16 个子密钥的顺序，所以只要接收方得到正确的密钥，便可对得到的加密后二进制串进行解码得到传输的原始明文。

二、 总体结构与流程

DES 算法具体流程如下：

1. IP 置换原始明文并分解：使用 IP 置换表将明文各二进制码的顺序进行调换，置换过后将得到的 IP 二进制码分解成两部分，左右两部分各 32 位；
2. 置换原始密钥并分解：使用 PC_1 置换规则将原始密钥置换成新密钥，此时，密钥的奇偶校验位已被省去，然后再将得到的 56 位密钥分解成两部分，前后各 28 位；
3. 生成 16 位子密钥：将第二步得到的两部分按照左移表分别进行连续的移位，每次移位后将两部分合并，经过 PC_2 置换得到一个子密钥，最终得到 16 个子密钥，每个子密钥为 48 位；
4. 循环操作：首先将明文的右半部分作为新的左半部分，再经过 E 置换扩展成 48 位，与第一个子密钥异或后，进入 S 盒进行选择，输出 32 位二进制码，再把其进入 P 盒后置换得到的新 32 位二进制码与旧的左半部分异或，得到结果作为新的右半部分，循环 16 次操作得到最后的左右两部分；
5. 交换并置换：将最后得到的第 16 个明文的循环结果的左右部分交换，进行 IP 的逆置换过后，得到的结果即为加密输出。

三、 模块分解

根据 DES 算法的流程，我们将算法的过程分为如下几个功能模块：置换并分解，子密钥生成，feistel 轮函数，交换并置换。

【模块一 置换并分解】

由于明文和密钥在一开始都要经过置换和分解的过程，所以此模块输入为明文/密钥、置换规则，输出为分解得到的左半部分和右半部分。其中，明文的置换规则为 IP，密钥的置换规则为 PC_1。

在代码中用 transforming()函数表示置换过程，用 trans_half()函数表示置换及分解的全过程。

【模块二 子密钥生成】

子密钥生成的部分可以划分为两部分，第一步是经过移位将密钥的分解结果分别移位并组合得到 16 个 56 位的二进制码，每个二进制码再通过 PC_2 置换的缩位得到 48 位的子密钥。

在代码中用 shift_16()函数表示移位得到 16 个 56 位二进制的过程，用 key_16()函数表示生成子密钥的过程。

【模块三 轮函数】

轮函数由 E 置换、S 盒选择、P 盒置换三个部分构成，输入起始明文拆分后的左右两部分，得到最后第 16 个输出的新的左右两部分

在代码中用 recycle()函数表示轮函数。

【模块四 交换并置换】

将模块三的输出的左右两部分交换后进行逆置换，得到结果

在代码中使用相同的置换方法得到输出。

四、 数据设计与文件结构

【文件结构】

运行文件----------main.py

函数文件----------function.py

置换及盒规则文件----------rule.py

【数据设计】

置换规则均由列表存储，列表中元素为数字类型，表示放入其所在位置的被置换数据的第 n 个元素，置换规则分别为 IP _IP PC_1 PC_2 E P。

16 位左移位数也由列表存储，列表中元素为数字类型，表示每一次移位的多少，规则为 left_shift。

s 盒用列表存储，列表中元素为以数字位数据类型的列表，作为每 6 位输入得到 4 位输出结果的十进制表示。

以上内容均在 rule.py 文件中。

简单函数分别为 shift()左移函数，exclusive()异或函数，transforming()置换函数，较为复杂的函数有 s_box()选择函数，shift_16()产生 16 个成对的移位输出，key_16()产生 16 个子密钥输出，recycle()轮函数。

以上内容均在 function.py 文件中。

在 main.py 文件中，data 为原始明文，key 为原始密钥；lr 是一个大小为 2 的列表，分别表示明文分解或每次轮循环时产生的左右两部分；c0，d0 表示原始密钥第一次分解产生的左右两部分；c，d 为两个大小为 17 的列表，分别表示 c0，d0 和其他 16 对移位输出；k 为大小为 16 的列表，表示 16 个子密钥；res 为最终产生的结果。

五、 代码实现

【function.py】

```
# 置换并分解
# data----要处理的数据
# trans---置换规则
# len-----新得到的二进制位数
def tran_half(data, trans, len):
    res = [['' for i in range(0, len//2)] for i in range(0, 2)]
    for i in range(0, 2):
```

```
        for j in range(0, len//2):
            res[i][j] = data[trans[i*(len//2)+j]-1]
    return res[0], res[1]

# c0 d0-----第一次分解密钥的两部分
# left------左移规则
# bit-------左右两部分的大小（32）
# len-------循环次数
def shift_16(c0, d0, left, bit, len):
    c = [['' for i in range(bit)] for i in range(len+1)]
    d = [['' for i in range(bit)] for i in range(len+1)]
    c[0] = c0;
    d[0] = d0;
    for i in range(1, len+1):
        c[i] = shift(c[i-1], left[i-1])
        d[i] = shift(d[i-1], left[i-1])
    return c, d
# 产生 16 个子密钥
# c d-------左右两部分的列表
# num-------循环次数
# trans-----置换规则
# len-------子密钥位数
def key_16(c, d, num, trans, len):
    k = [['' for i in range(len)] for i in range(num)]
    for i in range(num):
        k[i] = transforming(c[i+1]+d[i+1], trans, len)
    return k

# s 盒选择
# data-------处理数据
# s----------s 盒的列表，元素仍为列表
# inputlen, output_len-------输入 6 位，输出 4 位
# times------操作次数 8
def s_box(data, s, input_len, output_len, times):
    res = ['' for i in range(output_len * times)]
    for i in range(times):
        x = int(data[i*input_len])*2 + int(data[i*input_len+5])
        y = int(data[i*input_len+1])*8 + int(data[i*input_len+2])*4 +
int(data[i*input_len+3])*2 + int(data[i*input_len+4])
        binary =  bin(s[i][x*16 + y])
        for j in range(4):
            if len(binary)-1-j >= 0:
                if binary[len(binary)-1-j] != 'b':
                    res[(i+1)*output_len-1-j] = binary[len(binary)-1-j]
```

```
                else:
                    res[(i+1)*output_len-1-j] = '0'
            else:
                res[(i+1)*output_len-1-j] = '0'
    return res

# 轮函数
# l r--------一轮循环被操作的左右两部分
# k---------当前要用的单个子密钥
# ex_len----扩展位数 48
# s---------整个 s 盒
# len--------输出位数
def recycle(l, r, k, ex_len, s, len):
    res = [['' for i in range(len)] for i in range(2)]
    res[0] = r
    ex_r_k = exclusive(transforming(r, rule.E_Trans, ex_len), k, ex_len)
    s_32 = s_box(ex_r_k, s, 6, 4, 8)
    p_32 = transforming(s_32, rule.P, 32)
    res[1] = exclusive(l, p_32, 32)
    return res
```

【main.py】

数据

```
# set l0 and r0
lr = function.tran_half(data, rule.ip, 64)
# set c0 and d0
c0, d0 = function.tran_half(key, rule.pc_1, 56)
# left shift the cds
c = [['' for i in range(28)] for i in range(17)]
d = [['' for i in range(28)] for i in range(17)]
c, d = function.shift_16(c0, d0, rule.left_shift, 28, 16)
# 16 key[0-15]
k = function.key_16(c, d, 16, rule.PC_2, 48)
```

加密

```
#16 operations
for r in range(16):
    lr = function.recycle(lr[0], lr[1], k[r], 48, rule.S, 32)
#change l and r then do the transform
res = function.transforming(lr[1]+lr[0], rule._ip, 64)
```

解密

```
lr = function.tran_half(res, rule.ip, 64)
for r in range(16):
```

```
    lr = function.recycle(lr[0], lr[1], k[15-r], 48, rule.S, 32)
res = function.transforming(lr[1]+lr[0], rule._ip, 64)
```

六、 实验结果

在网络上查询我们可以得到一对样例，当明文是 0000000100100011010001010110011110001001101010111100110111101111，密钥是 0001001100110100010101110111100110011011101111001101111111110001 时，输出为 1000010111101000000100110101010000001111000010101011010000000101，我们使用程序输出加密后的 cipher 和解密结果的 checking reference 和输出结果与原始明文进行对比，我们可以发现实验成功。

```
use the example to test the programme
--------------------------------------------------------
The expected result of example is: 1000010111101000000100110101010000001111000010101011010000000101
The ciphertext of the same data is:1000010111101000000100110101010000001111000010101011010000000101
Get the result successfully!

The original input data is:0000000100100011010001010110011110001001101010111100110111101111
The checking reference is: 0000000100100011010001010110011110001001101010111100110111101111
Return to the data successfully!
--------------------------------------------------------
```

自输入数据，得到解码结果判断是否成功解码，仍旧得到试验成功的结果。

```
use the example to test the programme
--------------------------------------------------------
please input the data(64): 0000000000000000000000000000000011111111111111111111111111111111
The ciphertext of the same data is:10011000111010111101011001010111100001110111100111001010011011
The original input data is:0000000000000000000000000000000011111111111111111111111111111111
The checking reference is: 0000000000000000000000000000000011111111111111111111111111111111
Return to the data successfully!
--------------------------------------------------------
```