

# 实验名称：DES 算法设计过程

姓名：南风不竞  
学号：1533XXXX

## 一、实验要求

完成一个 DES 算法的详细设计，包括:算法原理概述；总体结构；模块分解；数据结构；类-C 语言算法过程。

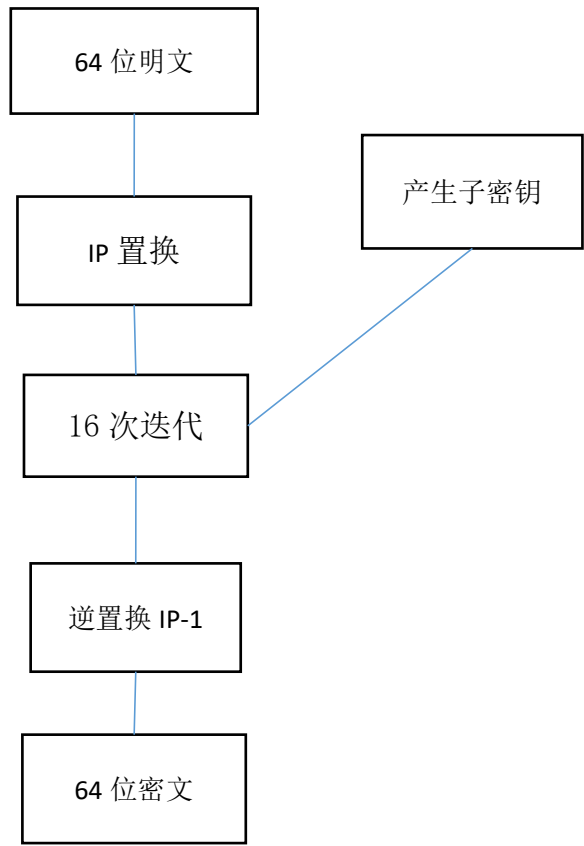
## 二、实验过程

### 1. 算法原理概述

DES 是一个对称加密算法，加密解密使用相同的密钥，主要步骤是换位和置换。DES 也是分组密码，输入的明文和密钥要求是 64 位长度，输出的密文长度也是 64 位，使用了 Feistel 网络，基于混淆和扩散这两种操作，增强了密码强度，另外加密和解密过程几乎相同，方便了软件和硬件上的实现。

### 2. 总体结构

加密过程：



解密过程：

除了子密钥应反序调用，其他与加密过程完全一致

### 3. 模块分解

比较主要和复杂的模块有以下几个：

#### 1. 产生 16 个子密钥

通过 64 位密钥，根据 PC-1 表压缩得到 56 位，分成前后 28 位 C0、D0，进行 16 次循环，第 1, 2, 9, 16 次循环时循环左移一位，否则左移两位，再经过 PC-2 表得到子密钥  $K_i$ 。

#### 2. 各种置换

举例 IP 置换：根据 IP 置换表让二进制元素重新排位。其他置换同理包括 E-扩展。

#### 3. 16 次迭代运算

加密：经过初始置换后获得  $L_0, R_0$ ，经过 16 次  $L_i = R_{i-1}$ ,  $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$ ,  $i = 1 \dots 16$ ,  $f$  为 Feistel 轮函数，最后对得到的  $L_{16}R_{16}$  进行左右交换。

解密算法：子密钥反序调用

#### 4. Feistel 轮函数

将 32 位的  $R$  进行 E 扩展成 48 位，再与对应的子密钥进行按位异或运算，分成 8 个长度为 6 的分组，通过对应的 S-盒进行 6-4 转换，输出 8 个长度为 4 的分组后合并，再进行 P 置换。

#### 5. S-盒 6-4 转换

S-盒是一个 4 行、16 列的表，输入为 6 位二进制数组  $b_0b_1b_2b_3b_4b_5$ ，输出为 S-盒 $[(b_0b_5)10][(b_1b_2b_3b_4)10]$  的 4 位二进制表示。

### 4. 数据结构

所有表包括 PC-1、PC-2、P 以及 S 盒在 python 程序中均看成一维数组，用 list 表示。

## 三、python 代码实现与测试

### 1. 代码实现

#### 1. 产生 16 个子密钥

```

SUBKEYS = []          # 子密钥集合
def general_subkeys(K):
    """
    生成子密钥
    """

    HALF_SUBKEY_LENGTH = 28
    # 64位K通过PC-2表压缩成56位
    subkey = get_PC1_subkey(K)
    # 获得 C0, D0
    C, D = subkey[:HALF_SUBKEY_LENGTH], subkey[HALF_SUBKEY_LENGTH:]
    # 获得 C1, D1
    for i in range(1, 17):
        if i == 1 or i == 2 or i == 9 or i == 16:
            C, D = move_onebit_subkey(C), move_onebit_subkey(D) # 循环左移一位
        else:
            C, D = move_twobit_subkey(C), move_twobit_subkey(D) # 循环左移两位
        SUBKEYS.append(get_PC2_subkey(C + D)) # 56位K通过PC-2表压缩成48位

```

移位和经过 PC-1、PC-2 表压缩：

```

def get_PC1_subkey(K):
    """
    对64位的k进行压缩置换得到56位
    """

    subK = []
    for i in PC1:
        subK.append(K[i-1])
    return subK

def get_PC2_subkey(K):
    """
    对56位的k进行压缩置换得到48位
    """

    subK = []
    for i in PC2:
        subK.append(K[i-1])
    return subK

def move_onebit_subkey(K):
    """
    i = 1,2,9,16 左移一位
    """

    return K[1:] + K[:1]

def move_twobit_subkey(K):
    """
    i = 1,2,9,16 左移两位
    """

    return K[2:] + K[:2]

```

## 2. 各种置换

```

def permute_plain_text_by_IP(plain_text):
    """
    通过IP重排明文块
    """
    return [plain_text[ip_index - 1] for ip_index in IP]

def inverse_permutate_by_IP_1(R_L):
    """
    逆置换IP-1
    """
    return [R_L[i - 1] for i in IP_1]

def permute_by_P(thiry_two_bit_R):
    """
    32位R逆置换
    """
    return [thiry_two_bit_R[i - 1] for i in P]

```

E-扩展:

```

def E_extended(R):
    """
    E扩展
    """
    extended_R = []
    for i in range(len(EXTENDED_RULE_TABLE)):
        extended_R.append(R[EXTENDED_RULE_TABLE[i][0] - 1])
        for j in range(4):
            extended_R.append(R[4 * i + j])
        extended_R.append(R[EXTENDED_RULE_TABLE[i][1] - 1])
    return extended_R

```

3. 16 次迭代运算

```

def DES_encryption_or_decryption(text, secret_key, is_decrypt):
    """
    对text进行加密或解密，secret_key是密钥，is_decrypt不为0则加密，否则解密
    """
    # IP置换
    deal_text = permute_plain_text_by_IP(text)
    L, R = deal_text[:int(HALFSIZE)], deal_text[int(HALFSIZE):]

    # 产生子密钥
    general_subkeys(secret_key)

    # 16次迭代
    start, end, interval = 0, ITERATION_COUNTERS, 1
    # 若为解密，反序调用子密钥
    if is_decrypt:
        start, end, interval = ITERATION_COUNTERS - 1, -1, -1
    for i in range(start, end, interval):
        t_L = R
        R = exclusive_or(L, feistel(R, i))
        L = t_L
    # 逆置换IP-1
    return inverse_permutate_by_IP_1(R + L)

```

#### 4. Feistel 轮函数

```

# Ri-1, Ki
def feistel(R, i):
    """
    feistel轮函数，R为长度为32位的Ri-1，i为迭代第i次
    """
    # 1 - 长度为32位的Ri-1作E-扩展，作为48位的串E(Ri-1)
    # 2 - 将E(Ri-1)和长度为48位的子密钥ki作48位二进制串按位异或运算

    t_R = exclusive_or(E_extended(R), get_subkey(i))
    # 3 - 将2得到的结果分成8个长度为6的分组，经过8个不同的S-盒
    #     进行6-4转换，得到8个长度分别为4位的分组
    # 4 - 将分组结果合并成长度为32位的串
    GROUP_LEN = 6
    thirty_two_bit_R = []
    # print("t_R", t_R)
    for i in range(S_BOX_NUM):
        thirty_two_bit_R += S_box_substitution(t_R[GROUP_LEN * i : GROUP_LEN * (i + 1)], S_BOXES[i])
        # print(thirty_two_bit_R)

    # 5 - 把32位的串进行P-置换
    return permute_by_P(thirty_two_bit_R)

```

#### 5. S-盒 6-4 转换

```

def S_box_substitution(R_group, S_BOX):
    """
    S盒替换
    """
    row = (R_group[0] << 1) + R_group[5]
    col = 0
    for i in range(3, 6):
        col = (col << 1) + R_group[i]

    demical_output = S_BOX[row * S_BOX_COLS + col]
    binary_output = []
    BINARY_OUTPUT_LEN = 4
    # 10进制转成4位二进制
    for i in range(BINARY_OUTPUT_LEN - 1, -1, -1):
        if demical_output - 2 ** BINARY_OUTPUT_LEN >= 0:
            binary_output.append(1)
            demical_output -= 2 ** BINARY_OUTPUT_LEN
        else:
            binary_output.append(0)

    return binary_output

```

## 2. 测试过程及结果

1. 加密测试代码：读取要加密的明文，随机生成密钥，加密获得密文，默认"utf-8"编码

```

def encrypt_test():
    """
    读取要加密的明文，随机生成密钥，加密获得密文，默认"utf-8"编码
    plain_text.txt : 明文
    secret_key.txt : 密钥
    """
    # -----读取将加密的明文-----
    with open("plain_text.txt", "r", encoding="utf-8") as fin:
        read_data = fin.read()
    # 获得密文的16进制的list
    hex_list = list(bytearray(read_data, encoding="utf-8"))
    # 判断密文是不是8字节的倍数，不是则补上缺的位数
    # 假如缺4个字节，则补上4个4；假如缺5，则补上5个5
    # 解密的时候再对应去掉
    hex_list_len = len(hex_list)
    if (hex_list_len % 8):
        lack_num = 8 - hex_list_len % 8
        for i in range(lack_num):
            hex_list.append(lack_num)
    # -----

    # 16进制list--->二进制字符串list--->合并成二进制字符串--->二进制list
    bin_str_list = list(map(lambda x: bin(x)[2:], hex_list))
    for i in range(len(bin_str_list)):
        if len(bin_str_list[i]) < 8:
            bin_str_list[i] = (8 - len(bin_str_list[i])) * '0' + bin_str_list[i]
    bin_list = list(map(lambda x: int(x), list(''.join(bin_str_list))))
    # -----

```



```

# -----随机生成64位密钥并保存在文件中-----
secret_key = random_generated_key()
print("secret_key = ", ''.join(list(map(lambda x: str(x), secret_key))))
with open("secret_key.txt", "w") as fout:
    fout.write(''.join(list(map(lambda x: str(x), secret_key))))
# -----

# -----对二进制list进行每次分64位进行加密-----
ciphertext = []
for i in range(int(len(bin_list) / 64)):
    t = DES_encryption_or_decryption(bin_list[i * 64 : (i + 1) * 64], secret_key, 0)
    ciphertext += t
# 将密文保存进文件中
with open("ciphertext.txt", "w") as fout:
    for i in ciphertext:
        fout.write(str(i))
# -----

```

2. 解密测试代码：根据密文和密钥获得明文，并把结果输出到文件中，默认"utf-8"编码

```

def decrypt_test():
    """
    根据密文和密钥获得明文，并把结果输出到文件中，默认"utf-8"编码
    secret_key.txt : 密钥
    ciphertext.txt : 密文
    result.txt | : 解密结果
    """
    # -----读取密文和对应的密钥-----
    with open("secret_key.txt", "r") as fin:
        secret_key = list(map(lambda x: int(x), list(fin.read()))))
    with open("ciphertext.txt", "r") as fin:
        ciphertext = list(map(lambda x: int(x), list(fin.read()))))
    # -----

    # -----每次分64位进行解密获得二进制list-----
    bin_list = []
    for i in range(int(len(ciphertext) / 64)):
        t = DES_encryption_or_decryption(ciphertext[i * 64 : (i + 1) * 64], secret_key, 1)
        for i in range(int(len(t) / 8)):
            t = list(map(lambda x: str(x), t))
            bin_list.append(''.join(t[i * 8 : (i + 1) * 8]))
    # -----

```

```

# -----将二进制list转成16进制list转成明文-----
hex_list = list(map(lambda x: int(hex(int(x, 2)), 16), bin_list))
# 获取最后一位数字，判断是否是由于不足8字节而填补上的
end_num = hex_list[len(hex_list) - 1]
if end_num < 8:
    for i in range(-end_num, 0, 1):
        if hex_list[i] != end_num:
            break
    else:
        hex_list = hex_list[:-end_num]
# print(bytearray(hex_list).decode("utf-8"))
with open("result.txt", "w", encoding="utf-8") as fout:
    fout.write(bytearray(hex_list).decode("utf-8"))
# -----

```

### 3. main 函数代码

```
if __name__ == "__main__":  
    encrypt_test()  
    decrypt_test()
```

### 4. 运行

在 plain\_text.txt 放进中文和英文的文字，运行程序，得到运行结果。为了防止数据偶然性，多次运行程序，并更换 plain\_text.txt 中的数据。

### 5. 运行结果

每次运行程序密钥都随机生成：

```
C:\Users\NanXuan\Desktop\study  
λ python3 t.py  
secret_key = 010001110111010111000110110011011100101101111010001001001  
1010000  
  
C:\Users\NanXuan\Desktop\study  
λ python3 t.py  
secret_key = 111111001111011001011101100110011001010101111101010000101  
1001001  
  
C:\Users\NanXuan\Desktop\study  
λ python3 t.py  
secret_key = 001000101010010001111101011111001101010111101010100111110  
1001011
```

各个文件夹都有正确生成对应的内容，result.txt 也生成了正确的解密结果，由于人工对照 result.txt 和 plain\_text.txt 对照具有较大误差性，因此使用 diff 进行比较两者的内容是否一致：

```
C:\Users\NanXuan\Desktop\study  
λ python3 t.py  
secret_key = 0010001010100100011111010  
1001011  
  
C:\Users\NanXuan\Desktop\study  
λ diff result.txt plain_text.txt  
  
C:\Users\NanXuan\Desktop\study  
λ
```

没有任何提示输出，说明两个文件夹的内容一致。多次运行程序，结果相同，可以说明本次实验的代码错误的概率较低。

## 四、实验总结和感想

本次实验主要使用了 python，比用 c++ 实现节省了许多劳动力，但还是不得不多加细心和耐心。除了加密过程中对二进制 1 和 0 的处理容易出现错误，还有许多细节需要进一步的



思考处理，比如：

1. 如果输入的明文不是 8 字节的倍数要如何处理？

我的处理方式是假如明文字节数距离 8 字节的倍数少  $x$  个字节，则在明文末尾补上  $x$  个  $x$ ，若缺少 4 个字节，则再末尾补上 4 个 4 即可，解密时再去掉。也考虑到会有明文恰好是以  $x$  个  $x$  结尾，但考虑到出现这种情况的概率较少，故不作处理。

2. 如何将明文转换成二进制数组？以及解密后如何将二进制数组转成可读的字符串？

查资料找了各种各样的方法先把明文转成十六进制再转成二进制。

以及编码问题和字符串与各种进制数字的转换。

老师上课的 ppt 内容已经是讲的非常清晰简明易懂，非常有助于捋顺逻辑，非常良心，各个表的数据和网上查找的数据是一致的，所以在写代码的过程中节省较多时间。

另外，DES 算法比较复杂，加密和解密过程却几乎相同，非常方便，写代码时不由得佩服对算法设计者的智慧。