信息安全技术 Project:DES 算法实现

21307174 刘俊杰

May 2024

1 算法介绍

1.1 算法简介

DES (Data Encryption Standard) 是一种对称密钥加密算法，由 IBM 于上世纪 70 年代初开发，并在 1977 年被美国国家标准局 (NIST) 确定为联邦信息处理标准 (FIPS) 中的一部分。DES 是历史上最常用的加密算法之一，尽管因为使用的 56 位密钥过短导致它在现代计算机环境下已被认为是不安全的，但它的设计原理对于理解其他现代加密算法仍然具有重要意义。

1.2 算法特点

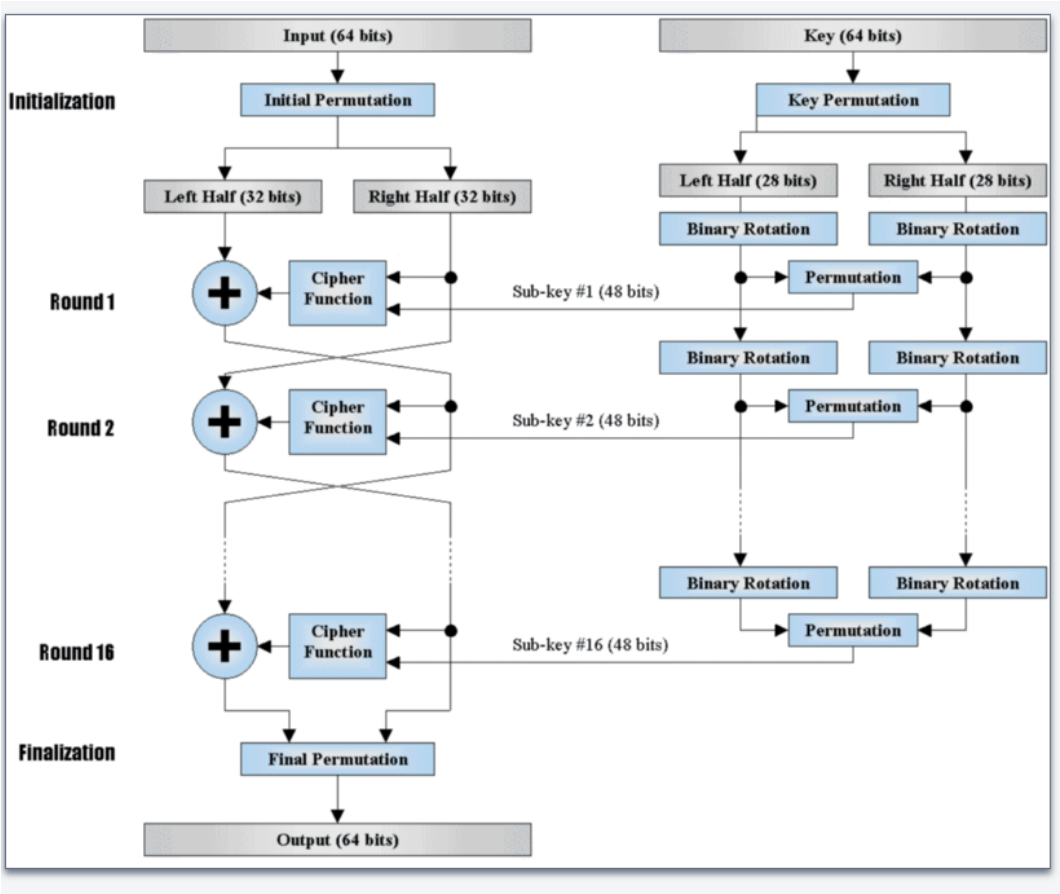
DES 算法具有以下特点:

1. 对称加密算法: DES 是一种对称密钥加密算法，这意味着加密和解密使用相同的密钥。这种算法的优点是速度快，但需要解决密钥分发的问题。
2. 分组密码: DES 是一种分组密码，它将明文分成固定大小 (64 位) 的块，并对每个块进行加密。由于 DES 是分组密码，因此它需要填充 (padding) 来处理不完整的块。
3. 密钥长度: DES 的密钥长度为 64 位，这意味着 DES 使用 65 位密钥对 64 位的明文进行加密。然而，由于每个字节的奇偶校验位，实际上只有 56 位用于加密。这在现代计算机环境下已被认为是不够安全的。
4. 轮函数: DES 使用一系列的轮函数 (round function) 来对明文进行加密。每一轮中，明文块被分成左右两部分，经过一系列的置换和替换操作，然后与上一轮的结果进行混合。

5. **密钥调度：在 DES 加密过程中，密钥需要经过一系列的置换和轮密钥生成算法来生成子密钥。这些子密钥用于每一轮的加密操作。**
6. Feistel **结构：DES 采用了 Feistel 结构，这意味着加密和解密过程是相同的，只是在轮密钥的应用顺序上有所不同。**

1.3 算法过程

DES 算法的整体框架：



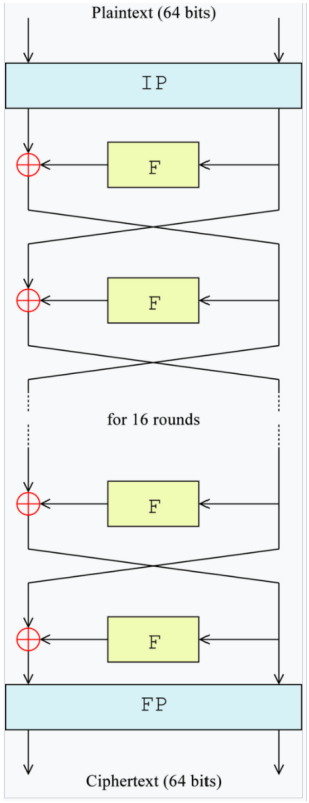
上述框架左侧是 DES 加解密的基本流程，右侧是密钥调度流程

1.3.1 **初始置换** (Initial Permutation)

将输入按规定的置换表做一次置换
(IP 和 FP 都是简单置换，对于密码安全没有任何意义)

1.3.2 round **轮转**

DES 中的 Feistel 网络结构:



经过 IP 后的结果，均等的切分为 L 和 R 两个部分，并加入 round 迭代过程。经过 16 轮迭代过后的结果 L 和 R 再拼接为 (R+L)，将 (R+L) 输入 FP 做最后一次置换得到最终结果。

1.3.2.1 round 迭代

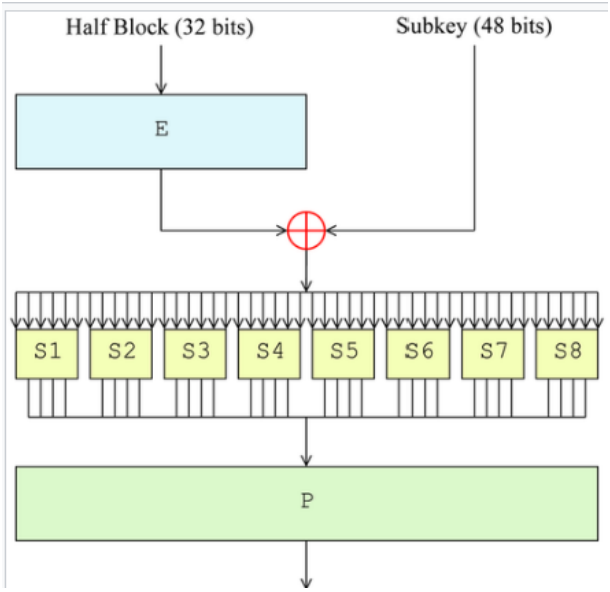
round 迭代的每一次过程如下：

$$L' = R$$
$$R' = L \oplus F(R, subkey)$$

其中 F() 为 Feistel 函数，subkey 为对应轮次产生的子密钥

1.3.2.2 Feistel 函数

Feistel 函数结构如下：



输入的 R 经过 Expand 置换后形成 48 位，在于 subkey 异或，异或结果再经过 S 置换和 P 置换后得到 Feistel 函数的最终结果。

1.3.2.3 Expand 置换

利用置换将输入的 32 位的 R 生成 48 位的结果

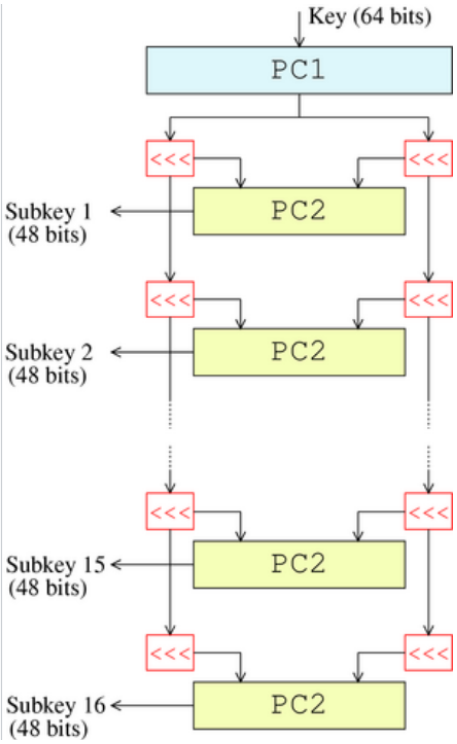
1.3.2.4 盒置换（S 置换）

把输入比特串分为 8 组，一组 6 bit，分别对每一组进行 S 盒代换。经过 S 盒，每一组由 6 bit 缩减为 4 bit。（S 盒的行号从 0 到 3，列号从 0 到 15）

1.3.2.5 P 置换

利用给定的置换表（精心设计的）来实现置换

1.3.3 密钥调度



密钥先经过 PC1 置换从 64 位 key 里面选出 56 位。经过 PC1 之后，可分为左、右两个半密钥，长度都是 28 位。接着，每一轮把左、右半密钥左旋几位，再调用 PC2 置换方法来构造每一轮的子密钥。

1.3.4 PC1

PC1 置换从 64 位 key 里面选出 56 位。经过 PC-1 之后，可分为左、右两个半密钥，长度都是 28 位。

1.3.5 PC2

PC2 置换，用于从左右半密钥拼起来的 56 位密钥中，选取 48 位作为一个子密钥

1.4 **最终置换** (Final Permutation)

将最终的 R 和 L 拼接成 (R+L) 做最后一次置换
(IP 和 FP 都是简单置换，对于密码安全没有任何意义)

2 算法实现

2.1 辅助函数

因为密钥和明文或者密文在代码实现中是用十六进制字符串表示，而在 DES 算法过程中使用二进制数字列表来存储数据，故定义构造一些辅助函数，方便后续调用实现十六进制字符串转为二进制数字列表、实现二进制数字列表转为十六进制字符串、二进制列表转为十进制整数、二进制列表异或等功能。

```
# 将十六进制表示的数字字符串转为二进制数保存在列表中(列表中每一位0或1)
    def hex2bin(self,input):
        # 将每个十六进制数转换为二进制表示，并存储在列表中
        binary_list = [bin(int(hex_char, 16))[2:].zfill(4) for hex_char in input]

        # 将二进制字符串转换为整数列表
        binary_list = [int(bit) for binary_string in binary_list for bit in binary_string]

        return binary_list

    # 将二进制数列表转为十六进制表示的数字字符串
```

```
12      def bin2hex( self , binary_list ):
13          # 将二进制列表分割成每四位一组，并转换为对应的十六进制字符
14          hexadecimal_number = ''
15          for i in range(0, len( binary_list ), 4):
16              binary_char = ''.join([str( bit ) for bit in binary_list [i:i+4]])
17              hexadecimal_number += hex(int(binary_char, 2))[2:]
18
19          return hexadecimal_number
20
21      # 二进制数组转十进制整数
22      def bin2dec( self , binary_list ):
23          dec = 0
24          for bin in binary_list :
25              dec *= 2
26              dec += bin
27          return dec
28
29      # 十进制数转为二进制列表
30      def int2bin ( self ,a, n):
31           assert 0 <= n and a < 2**n
32          res = [0] * n
33
34          for x in range(n):
35              res[n - x - 1] = a % 2
36              a = a // 2
37          return res
38
39       # 异或操作(按数组中的每一位相对应异或)
40      def xor( self ,a,b):
41          return [x^y for x, y in zip(a, b)]
```

2.2 IP 和 FP

```
# 初始置换
    def initial_permutations(self,input):
        ip = [58, 50, 42, 34, 26, 18, 10, 2,
              60, 52, 44, 36, 28, 20, 12, 4,
              62, 54, 46, 38, 30, 22, 14, 6,
              64, 56, 48, 40, 32, 24, 16, 8,
              57, 49, 41, 33, 25, 17, 9, 1,
              59, 51, 43, 35, 27, 19, 11, 3,
              61, 53, 45, 37, 29, 21, 13, 5,
              63, 55, 47, 39, 31, 23, 15, 7]
        return [input[i-1] for i in ip]

    # 最终置换
    def final_permutations(self,input):
        fp = [40, 8, 48, 16, 56, 24, 64, 32,
              39, 7, 47, 15, 55, 23, 63, 31,
              38, 6, 46, 14, 54, 22, 62, 30,
              37, 5, 45, 13, 53, 21, 61, 29,
              36, 4, 44, 12, 52, 20, 60, 28,
              35, 3, 43, 11, 51, 19, 59, 27,
              34, 2, 42, 10, 50, 18, 58, 26,
              33, 1, 41, 9, 49, 17, 57, 25]
        return [input[i-1] for i in fp]
```

2.3 round 轮转

```


    # 轮转
    def round(self,LeftHalf,RightHalf,subkeys):
        for i in range(0,16):
            tmp = [x for x in RightHalf]
```

```
            F_result = self.F(RightHalf,subkeys[i])
            RightHalf = self.xor(LeftHalf,F_result)
            LeftHalf = tmp
        return RightHalf + LeftHalf # 注意最后结果左右顺序调换
```

2.4 Feiste 函数

```
# 扩张置换，将32位的数据扩展到48位
    def Expand(self,input):
        e = [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]
        return [input[i-1] for i in e]



    # S盒变换，输入48位，输出32位
    def S(self,a):
        assert len(a) == 48

        S_box = [[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
                  0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
                  4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
                  15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13],
                 [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
                  3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
                  0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
                  13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9],
```

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

        a = np.array(a, dtype=int).reshape(8, 6)
        res = []

        for i in range(8):
            # 用 S_box[i] 处理6位a[i], 得到4位输出
            p = a[i]
            r = S_box[i][self.bin2dec([p[0], p[5], p[1], p[2], p[3], p[4]])]
            res.append(self.int2bin(r, 4))
```

```
60
61          res = np.array(res).flatten().tolist()
62          assert len(res) == 32
63
64          return res
65
66      # P置换
67      def P(self,a):
68          p = [16, 7, 20, 21,
69              29, 12, 28, 17,
70              1, 15, 23, 26,
71              5, 18, 31, 10,
72              2, 8, 24, 14,
73              32, 27, 3, 9,
74              19, 13, 30, 6,
75              22, 11, 4, 25]
76          return [a[x-1] for x in p]
77
78      # Feistel函数
79      def F(self,RightHalf,subkey):
80          t = self.xor(self.Expand(RightHalf), subkey)
81          t = self.S(t)
82          t = self.P(t)
83          return t
```

2.5 密钥调度（子密钥生成）

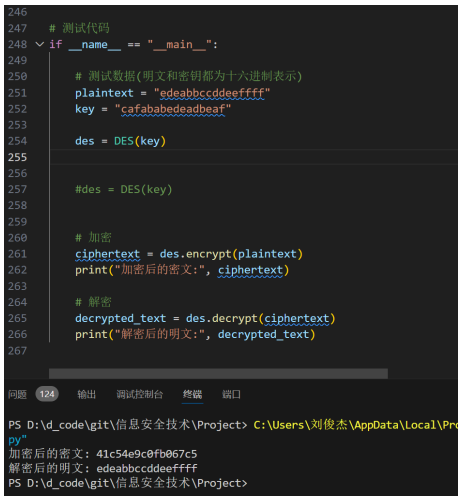
```
1   # 循环左移offset位
2       def leftRotate(self,a, offset):
3           return a[offset:] + a[:offset]
4
5       # PC2置换
6       def PC2(self,key):
```

```
        pc2 = [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]
        return [key[i-1] for i in pc2]

    # 根据密钥生成子密钥
    def generate_subkey(self):
        # PC1置换
        pc1_L = [57, 49, 41, 33, 25, 17, 9,
            1, 58, 50, 42, 34, 26, 18,
            10, 2, 59, 51, 43, 35, 27,
            19, 11, 3, 60, 52, 44, 36]
        pc1_R = [63, 55, 47, 39, 31, 23, 15,
            7, 62, 54, 46, 38, 30, 22,
            14, 6, 61, 53, 45, 37, 29,
            21, 13, 5, 28, 20, 12, 4]

        left = [self.key[i-1] for i in pc1_L]
        right = [self.key[i-1] for i in pc1_R]

        offset = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
        res = []

        for x in range(16):
            # LeftRotate
            left = self.leftRotate(left, offset[x])
            right = self.leftRotate(right, offset[x])
            # PC2置换产生子钥匙
```

```
40                 res.append(self.PC2(left + right))
41             return res
```

2.6 加密与解密

```
1  #加密
2      def encrypt(self,input):
3          input = self.hex2bin(input)
4          ip = self.initial_permutations(input)
5          LeftHalf = ip[0:32]
6          RightHalf = ip[32:]
7          output = self.round(LeftHalf,RightHalf,self.subkeys)
8          binary = self.final_permutations(output)
9          return self.bin2hex(binary)
10
11     # 解密
12     def decrypt(self,input):
13         input = self.hex2bin(input)
14         ip = self.initial_permutations(input)
15         LeftHalf = ip[0:32]
16         RightHalf = ip[32:]
17         output = self.round(LeftHalf,RightHalf,self.subkeys[::-1])
18         binary = self.final_permutations(output)
19         return self.bin2hex(binary)
```



3 实验结果

4 总结与感悟

4.1 DES 算法的优缺点

4.1.1 DES 算法的优点：

1．速度较快：DES 是一种相对较快的加密算法，这使得它在许多应用中都有着良好的性能表现。
2．结构简单：DES 的算法结构相对简单，易于理解和实现。
3．对普通攻击有一定抵抗力：DES 能够抵抗一些基本的攻击，如差分攻击、线性攻击等，这使得它在某些情况下仍然可以被使用。

4.1.2 DES 算法的缺点：

1．密钥长度短：DES 的密钥长度只有 56 位，这在当前的计算能力下已经不够安全。使用较短的密钥长度容易受到穷举搜索等暴力攻击的威胁。
2．已被破解：由于 DES 的密钥长度较短，使得它易受到巨大计算能力的现代计算机和专用硬件的攻击。DES 已经被证明是不安全的，并且可以在相对

较短的时间内被破解。
3. 未来不可持续：随着计算能力的不断增强和密码分析技术的不断发展，DES 已经不再具有足够的安全性，因此不适合用于保护敏感数据或长期使用。

4.1.3 DES 算法的替代方案

安全性方面的考虑使得研究者在 1980 年代晚期和 1990 年代早期提出了一系列替代的块密码设计，包括 RC5，Blowfish，IDEA，NewDES，SAFER，CAST5 和 FEAL。这些设计的大多数保持了 DES 的 64 位的块大小，可以作为 DES 的直接替代方案，虽然这些方案通常使用 64 位或 128 位的密钥。苏联导入了 GOST 28147-89 算法，该算法的块大小为 64 位，而密钥长度为 256 位，并在晚些时候的俄罗斯得到了应用。

2000 年代，DES 逐渐被 3DES 替代。3DES 相当于用两个（2TDES）或三个（3TDES）不同的密钥对数据进行三次 DES 加密。2010 年代，3DES 逐渐被更安全的高级加密标准（AES）替代。

2000 年 10 月，在历时接近 5 年的征集和选拔之后，NIST 选择了高级加密标准（AES）替代 DES 和 3DES。2001 年 2 月 28 日，联邦公报发表了 AES 标准，以此开始了其标准化进程，并于 2001 年 11 月 26 日成为 FIPS PUB 197 标准。AES 算法在提交的时候称为 Rijndael。选拔中其它进入决赛的算法包括 RC6，Serpent，MARS 和 Twofish。

4.2 实验感悟

通过本次课程项目，我学习了 DES 对称加密算法，了解了 DES 算法加解密的框架和过程，并用代码来实现 DES 算法。这不仅让我对该算法的对称密码体系结构:Feistel 网络结构以及密钥调度等基本概念和原理更加熟悉，而且了解到了 DES 的安全性逐渐受到挑战。这提醒我们在设计和选择加密算法时，需要考虑到未来的发展和计算环境，以确保数据的安全性和机密性。

因此，DES 算法不仅在于它的历史地位和影响，更在于它对密码学发展的启示和警示，为我们理解和应用密码学提供了宝贵的经验和教训。