DES——Go语言实现

本文为Web安全技术第一次作业实验报告，本次实验使用 `Golang` 语言实现DES加密算法。

一、实验环境

OS：windows 10

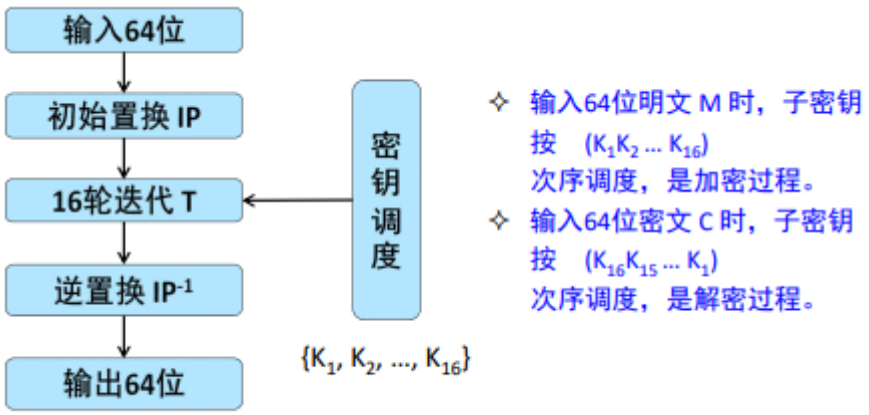
语言：Golang

IDE：Goland

二、算法原理概述及总体结构

DES 是一种典型的块加密方法：它以64位为分组长度，64位一组的明文作为算法的输入，通过一系列复杂的操作，输出同样64位长度的密文，DES是一种对称加密方式，通过相同的密钥可以对密文进行解密。

DES算法的总体结构



原始明文规范

原始明文消息最后的分组不够8个字节 (64位) 时，在末尾以字节填满，填入的字节取值相同，都是填充的字节数目，原始明文消息刚好分组完全时，在末尾填充8个字节 (即增加一个完整分组)，字节取值都是08，本次程序中输入的原始明文都为64位。

子密钥生成

1. 对 K 的56个非校验位实行置换 PC-1，得到 C0D0，其中 C0 和 D0分别由 PC-1 置换后的前28位和后28位组成。i = 1。
2. 计算 Ci = LSi(Ci-1) 和 Di = LSi(Di-1)， 当 i =1, 2, 9, 16 时，LSi(A) 表示将二进制串 A 循环左移一个位置；否则循环左移两个位置。
3. 对 56位的 CiDi 实行 PC-2 压缩置换，得到48位的 Ki 。i = i+1。
4. 如果已经得到 K16，密钥调度过程结束；否则转 (2)。

```
func GetKeys(k [64]int) (keys [16][48]int){
    /* PC-1置换获取C0,D0 */
    cd := pc1(k)
    for i:=0; i < 16; i++ {
        /* LS获取Ci,Di */
        cd = ls(cd, i+1)
        keys[i] = pc2(cd)
    }
    return keys
}
```

P1置换

```
package Key

/* PC-1置换 */
func pc1(k [64]int) (output [56]int){

   table := [56]int {
      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,
      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 }

   for i:= 0; i< 8; i++ {
      for j:= 0; j < 7; j++ {
         index := i*7 + j
         output[index] = k[table[index] - 1]
      }
   }
   return output
}
```

P2压缩置换

PC-2 压缩置换：从56位的 CiDi 中去掉第 9, 18, 22, 25, 35, 38, 43, 54位，将剩下的48位按照 PC-2 置换表作置换，得到 Ki 。

PC-2 压缩置换表					
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

```
/* PC-2压缩置换 */
func pc2(k [56]int) (output [48]int){

    table := [48]int {
        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}

    temp := [56]int{}
    for i:= 0; i< 8; i++ {
        for j:= 0; j < 6; j++ {
            index := i*6 + j
            temp[index] = k[table[index] - 1]
        }
    }
    count := 0
    for i:= 0; i< 56; i++ {
        j := i + 1
        if !(j == 9 || j == 18 || j == 22 || j == 25 ||
            j == 35 || j == 38 || j == 43 || j == 54) {
            output[count] = temp[i]
            count++
        }
    }
    return output
}
```

初始置换IP

给定64位明文块 M，通过一个固定的初始置换 IP 来重排 M 中的二进制位，得到二进制串 M0 = IP(M) = L0R0，这里 L0 和 R0分 别是M0 的前32位和后32位。下表给出 IP 置换后的下标编号序列。

IP 置换表 (64位)							
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

迭代T

根据 L0R0 按下述规则进行16次迭代，即Li = Ri-1, Ri = Li-1 XOR f(Ri-1, Ki), i = 1 .. 16.f 是输出32位的 Feistel轮函数。16 个长度为48位的子密钥 Ki (i = 1 .. 16)， 16次迭代后得到 L16R16 ， 左右交换输出 R16L16 。

```
func interationT(preLR [64]int, key [48]int)(LR [64]int){
    preL, preR := [32]int{}, [32]int{}
    /* L(i)=R(i+1)*/
    for i:=0; i < 32; i++ {
        preL[i] = preLR[i]
        preR[i] = preLR[i+32]
        LR[i] = preLR[i+32]
    }
    /*  获取Ri*/
    f := RoundFunction.Function(preR, key)
    for i:=0; i < 32; i++ {
        LR[i+32] = preL[i] ^ f[i]
    }
    return LR
}
```

Feistel 轮函数 f(Ri-1, Ki)

1. 将长度为32位的串 Ri-1 作 E-扩展，成为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，Ki 由密钥 K 生成；
3. 将 (2) 得到的结果平均分成8个分组 (每个分组长度6位)，各个分 组分别经过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 (3) 得到的分组结果顺序连接得到长度为32位的串；
5. 将 (4) 的结果经过 P-置换，得到的结果作为轮函数 f(Ri-1, Ki) 的最 终32位输出。

E-扩展规则

E-扩展规则 (比特-选择表)					
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

```
func eExtend(R [32]int)(E [48]int)  {
   table := [48]int {
      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}

   for i:= 0; i< 8; i++ {
      for j:= 0; j < 6; j++ {
         index := i*6 + j
         E[index] = R[table[index] - 1]
      }
   }
   return E
}
```

S-盒

S-盒是一类选择函数，用于二进制 6-4 转换。 Feistel 轮函数使用8个 S-盒 S1, ..., S8，每个 S-盒是一个4行 (编号 0-3)、16列 (编号 0-15) 的表，表中的每个元素是一个十进制数，取值在0-15之间，用于表示一个4位二进制数。

S_1-BOX															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	15	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

S_2-BOX															
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

S_3-BOX															
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

S_4-BOX															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
12	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

S_5-BOX															
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

S_6-BOX															
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

S_7-BOX															
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

S_8-BOX															
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

P置换

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

```
package Feistel

/* 初始置换ip */
func ipdisplace(k [64]int) (output [64]int){

    table := [64]int {
        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}

    for i:= 0; i< 8; i++ {
        for j:= 0; j < 8; j++ {
            index := i*8 + j
            output[index] = k[table[index] - 1]
        }
    }
    return output
}
```

IP逆置换

IP^{-1} 置换表 (64位)							
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

```
/* 逆置换ip-1 */
func ipinverse(k [64]int) (output [64]int){

    table := [64]int {
        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}

    for i:= 0; i< 8; i++ {
        for j:= 0; j < 8; j++ {
            index := i*8 + j
            output[index] = k[table[index] - 1]
        }
    }
    return output
}
```

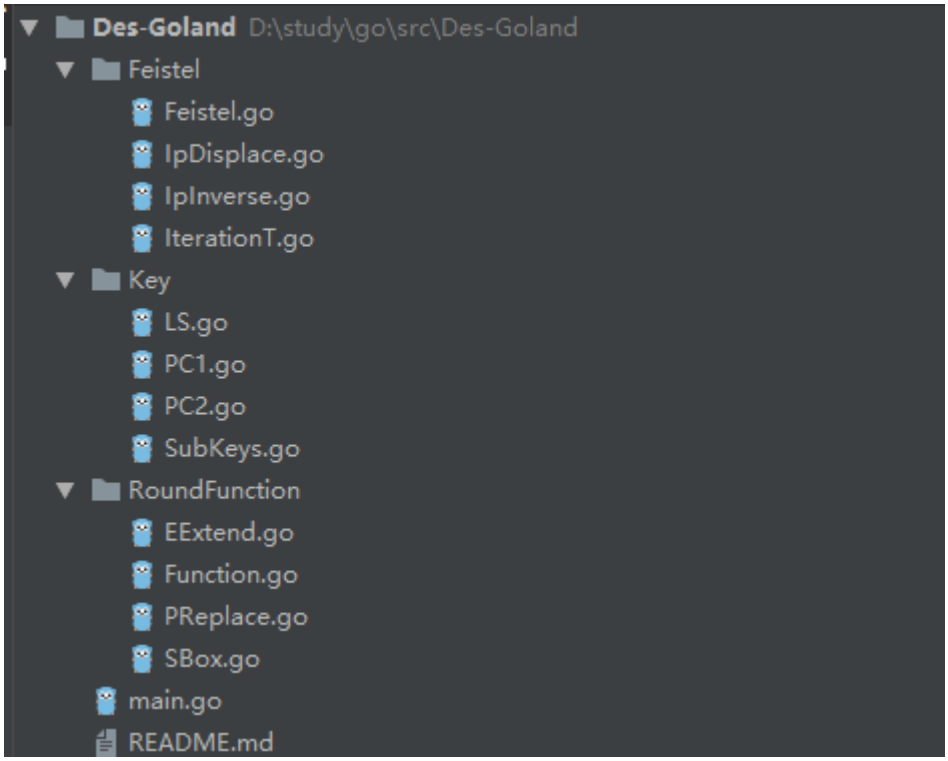
三、模块分解

由二可以了解到DES算法的总体结构以及实现原理，在实现DES算法的时候，先考虑到两个大的过程。

- 子密钥生成过程
- 加密解密过程

因此先分为两大模块 `Feistel` 和 `Key` ，其中Key模块负责子密钥生成，Feistel负责加密解密过程。

然后考虑到Feistel轮函数的复杂性，将其单独成一个模块 `RoundFunction` ，最终模块结构如下图



Feistel模块

算法的总体结构，负责主要的加密和解密过程，其中包括初始IP置换、中间的迭代以及最后的IP逆置换，可调用Key获取生成的子密钥，调用RoundFunction使用轮函数。

Key模块

管理子密钥的生成，该过程主要有PC-1置换、LS循环左移和PC-2压缩置换。

RoundFunction模块

轮函数实现，该过程主要有E-扩展规则、S-盒和P置换。

四、源代码

源代码可在 github 中查看。

五、数据结构

本次实验使用 一维数组 保存明文、密文以及置换表。

六、运行结果

执行结果如下：

```
DES2018
原始明文: DES2018
密文: hbG65��
解密后: DES2018
```

实验分析：

由实验截图可以看出，成功加密和解密，实验成功。