web 安全：DES 算法

DES 加密算法原理概述

DES 加密算法可以简单分为四个部分

1. IP 置换
2. 子密钥生成
    a) PC-1 置换
    b) 循环左移
    c) PC-2 置换
3. 16 轮迭代
    A. 高 32 位和低 32 位置换
    B. 轮函数处理
        a) E 盒扩展
        b) 子密钥异或处理
        c) S 盒转换
        d) P 盒置换
4. IP^{-1} 逆置换

加密函数 C 语言实现如下：

输入八字节明文，八字节密钥，输出 64 位二进制密文

```
void encryption(const char input[8], const char keyInput[8], int output[64]) {
  //8 字节输入转 64 位明文
  int PlainText[64] = {0};
  CharToBit(input, PlainText);

  //8 字节密钥转 64 位密钥
```

```
  int key[64] = {0};
  CharToBit(keyInput, key);
  //子密钥生成
  int subkeys[16][48];
  subKey(key, subkeys);

  //初始置换 IP
  int L[17][32] = {0}, R[17][32] = {0};
  IP(PlainText, L[0], R[0]);

  //迭代
  int i, j;
  for(i = 1; i <= 16; i++) {
    for(j = 0; j < 32; j++) {
      L[i][j] = R[i - 1][j];
    }
    int temp[32] = {0};
    //轮函数处理
    Feistel(R[i - 1], subkeys[i - 1], temp);
    //异或处理
     xor(temp, L[i - 1], R[i], 32);
  }
  int temp[32] = {0};
  for(i = 0; i < 32; i++) {
    temp[i] = L[16][i];
    L[16][i] = R[16][i];
    R[16][i] = temp[i];
  }

  //逆置换 IP^-1
  int output_1[64] = {0};
  for(i = 0; i < 32; i++) {
    output_1[i] = L[16][i];
    output_1[i + 32] = R[16][i];
  }

  IPInverse(output_1, output);
}
```

IP 置换

web 安全：DES 算法

给定 64 位明文块 M，通过一个固定的初始置换 IP 表来重排 M 中的二进制位，得到二进制串 $M_0 = IP(M) = L_0R_0$，这里 L_0 和 R_0 分别是 M_0 的左 32 位和右 32 位。

IP 置换表：

表中从左到右第 i 个位置中的数字 n 表示：将 M_0 的第 i 位置为 M 的第 n 位。

如第一个数所示，M_0 的第一位为 M 的第 58 位

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

IP 置换函数 C 语言实现如下：

输入为 64 位二进制明文块 M，输出为左 32 位 L_0，右 32 位 R_0

```
//ip 置换
void IP(const int input[64], int outputLeft[32], int outputRight[32]) {
  int i;
  int output[64];
  for(i = 0; i < 64; i++) {
    output[i] = input[IP_Table[i] - 1];
  }
  for(i = 0 ; i < 32; i++) {
    outputLeft[i] = output[i];
    outputRight[i] = output[i + 32];
  }
}
```

子密钥生成

子密钥生成过程根据给定的 64 位密钥 K，生成 16 个 48 位的子密钥 K_1-K_{16}，供 Feistel 轮函数 $f(R_{i-1}, K_i)$ 调用。

子密钥生成 C 语言实现如下：

输入为 64 位密钥 K，输出为 16 个 48 位子密钥

```
//子密钥生成
void subKey(const int input[64], int output[16][48]) {
  int i;
  int leftCount = 0;
  int C[28] = {0}, D[28] = {0}, Ci[16][28] = {0}, Di[16][28] = {0};

  PC_1(input, C, D);
  for(i = 1; i <= 16; i++) {
    if(i == 1 || i == 2 || i == 9 || i == 16) {
      leftCount += 1;
    } else {
      leftCount += 2;
    }
    LeftShift(C, Ci[i - 1], leftCount);
    LeftShift(D, Di[i - 1], leftCount);
  }
  for(i = 0; i < 16; i++) {
    PC_2(Ci[i], Di[i], output[i]);
  }
}
```

A. PC-1 置换

对 K 的 56 个非校验位实行置换 PC-1，得到 C_0D_0，其中 C_0 和 D_0 分别由 PC-1 置换后的左 28 位和右 28 位组成。

PC-1 置换表：

a) 置换规则同 IP 置换表

b) 注意到密钥 K 的 8 个检验位（8,16,24,32,40,48,56,64）的下标不参与置换

PC-1 置换表

C_0	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
D_0	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

PC-1 置换表 C 语言实现如下：

输入为二进制 64 位密钥，输出为左 28 位 C_0，右 28 位 D_0

```
//PC_1 置换
void PC_1(const int input[64], int outputLeft[28], int outputRight[28]) {
  int i;
  int output[56];
  for(i = 0; i < 56; i++) {
    output[i] = input[PC1_Table[i] - 1];
  }
  for(i = 0; i < 28; i++) {
    outputLeft[i] = output[i];
    outputRight[i] = output[i + 28];
  }
}
```

B. 循环左移

计算 $C_i = LS_i(C_{i-1})$ 和 $D_i = LS_i(D_{i-1})$

a) 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

b) 比如 C_1 为 C_0 每个位左移一个位置，比如 C_0 第 1 位左移为 C_1 的第 28 位，C_0 第 2 位左移为 C_1 的第 1 位

c) 最后得到 16 个 C_iD_i，各自拼接成对应子密钥 $subKey_i$

循环左移 C 语言实现：

输入为 28 位二进制串和左移位数，输出为 28 位二进制串

```
//密钥循环左移
void LeftShift(const int input[28], int output[28], int leftCount) {
  int i;
  for(i = 0; i < 28; i++) {
    output[i] = input[(i + leftCount) % 28];
  }
};
```

C. PC-2 置换

a) 置换规则与 IP 置换相同

b) 从 56 位的 C_iD_i 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 K_i 。

PC-2 置换表：

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 置换 C 语言实现：

输入为 28 位 C_i，28 位 D_i，输出为 48 位子密钥 K_i

```
//PC_2 置换
void PC_2(const int inputLeft[28], const int inputRight[28], int output[48]) {
  int i;
  int input[56];
  for(i = 0; i < 28; i++) {
    input[i] = inputLeft[i];
    input[i + 28] = inputRight[i];
  }
  for(i = 0; i < 48; i++) {
    output[i] = input[PC2_Table[i] - 1];
  }
}
```

16 轮迭代

A. 根据 L_0R_0 按下述规则进行 16 次迭代，即

$$L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i-1}, K_i), i = 1..16.$$

a) 这里 $\oplus$ 是 32 位二进制串按位异或运算，f 是输出 32 位的 Feistel 轮函数

b) 16 个长度为 48 位的子密钥 K_i (i = 1 .. 16) 由密钥 K 生成

c) 16 次迭代后得到 $L_{16}R_{16}$

d) 左右交换输出 $R_{16}L_{16}$

e) 使用 $R_{16}L_{16}$ 合成的 64 位数据进行 IP-1 逆置换

迭代 C 语言实现如下：

```
//迭代
int i, j;
for(i = 1; i <= 16; i++) {
  for(j = 0; j < 32; j++) {
    L[i][j] = R[i - 1][j];
  }
  int temp[32] = {0};
  //轮函数处理
  Feistel(R[i - 1], subkeys[i - 1], temp);
  //异或处理
  xor(temp, L[i - 1], R[i], 32);
}
//最后一次迭代结果 L16 R16 交换输出
int temp[32] = {0};
for(i = 0; i < 32; i++) {
  temp[i] = L[16][i];
  L[16][i] = R[16][i];
  R[16][i] = temp[i];
}
```

B. Feistel 轮函数

轮函数 C 语言实现如下：

输入为 32 位 R_{i-1}，48 位子密钥 K_i，输出为 32 位 R_i

```
//轮函数
void Feistel(const int input[32], const int subkey[48], int output[32]) {
  int temp[48] = {0}, temp2[48] = {0}, temp3[32] = {0};
  E(input, temp);
  xor(temp, subkey, temp2, 48);
  S(temp2, temp3);
  P(temp3, output);
}
```

a) E 盒扩展

将长度为 32 位的串 R_{i-1} 作 E-扩展，成为 48 位的串 $E(R_{i-1})$

E 盒扩展表：

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

E 盒扩展 C 语言实现：

```
//E 置换
void E(const int input[32], int output[48]) {
  int i;
  for(i = 0; i < 48; i++) {
    output[i] = input[E_Table[i] - 1];
  }
}
```

b) 子密钥异或处理

将 $E(R_{i-1})$ 和长度为 48 位的子密钥 K_i 作 48 位二进制串按位异或运算，K_i 由密钥 K 生成（见“子密钥生成”）；

异或处理 C 语言实现如下：

```
//异或运算
void xor(const int* inputOne, const int* inputTwo, int* output, int len) {
  int i;
  for(i = 0; i < len; i++) {
```

```
    output[i] = inputOne[i] ^ inputTwo[i];
  }
}
```

c) S 盒转换

i. 将异或运算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

ii. 将上一步得到的 8 个分组结果顺序连接得到长度为 32 位的串

iii. 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。

iv. 假设 S_i 的 6 位输入为 $b_1b_2b_3b_4b_5b_6$，则由 $n = (b_1b_6)_{10}$ 确定行号，由 $m = (b_2b_3b_4b_5)_{10}$ 确定列号，$[S_i]n,m$ 元素的值的二进制形式即为所要的 S_i 的输出。

两个例子：

✧ S-盒

○ 例1：设 S_1 的输入 $b_1b_2b_3b_4b_5b_6 = 101100$，则

$n = (b_1b_6)_{10} = (10)_{10} = 2$，

$m = (b_2b_3b_4b_5)_{10} = (0110)_{10} = 6$

查表得到 $[S_1]_{2,6} = 2 = (0010)_2$ 即为所要的输出。

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

web 安全：DES 算法

✧ S-盒

○ 例2：设 S_1 的输入 $b_1b_2b_3b_4b_5b_6 = 111001$，则

$n = (b_1b_6)_{10} = (11)_{10} = 3$，

$m = (b_2b_3b_4b_5)_{10} = (1100)_{10} = 12$

查表得到 $[S_1]_{3,12} = 10 = (1010)_2$ 即为所要的输出。

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 盒置换表：

✧ S-盒 S_1-S_4

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-盒 S_5-S_8

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

S 盒转换 C 语言实现如下：

输入为 48 位异或运算结果，输出为 32 位二进制串

```
//S-盒
void S(const int input[48], int output[32]) {
  int i;
  int a1[6], a2[6], a3[6], a4[6], a5[6], a6[6], a7[6], a8[6];
  int b1[4], b2[4], b3[4], b4[4], b5[4], b6[4], b7[4], b8[4];
  for(i = 0; i < 6; i++) {
    a1[i] = input[i];
    a2[i] = input[i + 6];
    a3[i] = input[i + 2 * 6];
    a4[i] = input[i + 3 * 6];
    a5[i] = input[i + 4 * 6];
    a6[i] = input[i + 5 * 6];
    a7[i] = input[i + 6 * 6];
    a8[i] = input[i + 7 * 6];
  }
  int n, m;
  n = (a1[0] << 1) + a1[5];
  m = (a1[1] << 3) + (a1[2] << 2) + (a1[3] << 1) + a1[4];

  n = (a2[0] << 1) + a2[5];
  m = (a2[1] << 3) + (a2[2] << 2) + (a2[3] << 1) + a2[4];

  n = (a3[0] << 1) + a3[5];
  m = (a3[1] << 3) + (a3[2] << 2) + (a3[3] << 1) + a3[4];

  n = (a4[0] << 1)+ a4[5];
  m = (a4[1] << 3) + (a4[2] << 2) + (a4[3] << 1) + a4[4];

  n = (a5[0] << 1) + a5[5];
  m = (a5[1] << 3) + (a5[2] << 2) + (a5[3] << 1) + a5[4];

  n = (a6[0] << 1) + a6[5];
  m = (a6[1] << 3) + (a6[2] << 2) + (a6[3] << 1) + a6[4];

  n = (a7[0] << 1) + a7[5];
  m = (a7[1] << 3) + (a7[2] << 2) + (a7[3] << 1) + a7[4];

  n = (a8[0] << 1) + a8[5];
  m = (a8[1] << 3)+ (a8[2] << 2) + (a8[3] << 1) + a8[4];

  for (i = 0; i < 4; i++) {
```

```
    b1[3 - i] = (S1_Box[n][m] >> i) & 1;
    b2[3 - i] = (S2_Box[n][m] >> i) & 1;
    b3[3 - i] = (S3_Box[n][m] >> i) & 1;
    b4[3 - i] = (S4_Box[n][m] >> i) & 1;
    b5[3 - i] = (S5_Box[n][m] >> i) & 1;
    b6[3 - i] = (S6_Box[n][m] >> i) & 1;
    b7[3 - i] = (S7_Box[n][m] >> i) & 1;
    b8[3 - i] = (S8_Box[n][m] >> i) & 1;
  }

  for(i = 0; i < 4; i++) {
    output[i] = b1[i];
    output[i + 4] = b2[i];
    output[i + 2 * 4] = b3[i];
    output[i + 3 * 4] = b4[i];
    output[i + 4 * 4] = b5[i];
    output[i + 5 * 4] = b6[i];
    output[i + 6 * 4] = b7[i];
    output[i + 7 * 4] = b8[i];
  }
}
```

d) P 盒置换

将长度为 32 的串经过 P 置换，得到的结果作为轮函数 $f(R_{i-1}, K_i)$ 的最终 32 位输出。

P 盒置换表：

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

P 盒置换 C 语言实现如下：

输入为 32 位二进制串，输出 32 位 R_i

```
          //P 置换
void P(const int input[32], int output[32]) {
  int i;
  for(i = 0; i < 32; i++) {
    output[i] = input[i];
  }
}
```

IP^{-1} 逆置换

对迭代 T 输出的二进制串 $R_{16}L_{16}$ 使用初始置换的逆置换 IP^{-1} 得到密文 C，即：$C = IP^{-1}(R_{16}L_{16})$.

IP^{-1} 置换表

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

IP^{-1} 置换实现如下：

输入为 64 位二进制串，输出为 64 位密文

```
void IPInverse(const int input[64], int output[64]) {
  int i;
  for (i = 0; i < 64; i++) {
    output[i] = input[IPInverse_Table[i] - 1];
  }
};
```

辅助函数

```
void CharToBit(const char input[8], int output[64]) {
  int i, j;
  for (j = 0; j < 8; j++) {
    for (i = 0; i < 8; i++) {
      output[7 * (j + 1) - i + j] = (input[j] >> i) & 1;
    }
  }
};


void BitToChar(const int input[64], char output[8]) {
  int i, j;
  for (j = 0; j < 8; j++) {
    for (i = 0; i < 8; i++) {
      output[j] = output[j] * 2 + input[i + 8 * j];
    }
  }
};
```

DES 解密算法原理概述

A. 分析所有的代替、置换、异或和循环移动过程，获得一个非常有用的性质：DES 的加密和解密可使用相同的算法和密钥。

B. DES 的过程设计使得用相同的函数来加密或解密每个分组成为可能。加解密过程中使用由同一个密钥 K 经过相同的子密钥生成算法得到的子密钥序列，唯一不同之处是加解密过程中子密钥的调度次序恰好相反。

a) 加密过程的子密钥按 (K_1 K_2 … K_{15} K_{16}) 次序调度

b) 解密过程的子密钥按 (K_{16} K_{15} … K_2 K_1) 次序调度

DES 解密算法 C 语言实现如下：

输入为 64 位密文，8 字节密钥，输出为 8 字节明文

```
void Decryption(const int input[64], const char keyInput[8], char output[8]) {
  //8 字节密钥转 64 位密钥
  int key[64] = {0};
  CharToBit(keyInput, key);
  //子密钥生成
  int subkeys[16][48];
  subKey(key, subkeys);

  //初始置换 IP
  int L[17][32] = {0}, R[17][32] = {0};
  IP(input, L[0], R[0]);

  //迭代
  int i, j;
  for(i = 1; i <= 16; i++) {
    for(j = 0; j < 32; j++) {
      L[i][j] = R[i - 1][j];
    }
    int temp[32] = {0};
    Feistel(R[i - 1], subkeys[16 - i], temp);
    xor(temp, L[i - 1], R[i], 32);
  }
  int temp[32] = {0};
  for(i = 0; i < 32; i++) {
    temp[i] = L[16][i];
    L[16][i] = R[16][i];
    R[16][i] = temp[i];
  }
```

```
  //逆置换 IP^-1
  int output_1[64] = {0};
  int output_2[64] = {0};
  for(i = 0; i < 32; i++) {
    output_1[i] = L[16][i];
    output_1[i + 32] = R[16][i];
  }

  IPInverse(output_1, output_2);
  BitToChar(output_2, output);
}
```

编译运行结果

测试函数如下：

```
int main() {
  int CipherText[64] = { 0 };
  char PlainText[9] = { 0 };
  char key[9] = { 0 };

  printf("请输入明文(8 字节)\n");
  scanf("%s", PlainText);

  printf("请输入秘钥(8 字节)\n");
  scanf("%s", key);

  printf("\n 加密\n");
  encryption(PlainText, key, CipherText);
  printf("密文如下:\n");
  for (int i = 0; i < 64; i++) {
    printf("%d", CipherText[i]);
    if ((i + 1) % 8 == 0)
      printf("\n");
  }
  printf("\n");

  printf("解密\n");
  Decryption(CipherText, key, PlainText);
```

```
  printf("明文如下:\n");
  for (int i = 0; i<8; i++) {
    printf("%c", PlainText[i]);
  }
  printf("\n\n");
  return 0;
}
```

```
请输入明文(8字节)
12345678
请输入秘钥(8字节)
87654321

加密
密文如下:
10011000
00110111
11101011
00011101
10011010
11100111
01001110
00011010

解密
明文如下:
12345678
```

```
请输入明文(8字节)
adhksdds
请输入秘钥(8字节)
fsfwedwq

加密
密文如下:
01110010
01010100
11111000
01100011
11101101
01001010
10111101
01110111

解密
明文如下:
adhksdds
```