

# 信息安全与技术作业一

## DES 算法

17343105 田皓

### 1.1 算法原理概述

#### 1.1.1 介绍

Data Encryption Standard (DES) 是一种典型的对称密钥算法，采用块加密方法，运行速度较慢，但较安全。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 使用加密密钥定义变换过程，因此算法认为只有持有加密所用的密钥的用户才能解密密文。DES 采用 64 位密钥，但由于每 8 位中的最后 1 位用于奇偶校验，实际有效密钥长度为 56 位。密钥可以是任意的 56 位的数，且可 随时改变。其中极少量的数被认为是弱密钥，但能容易地避开它们。所有的保密性依赖于密钥。DES 算法的基本过程是换位和置换。

#### 1.1.2 信息空间

DES 的信息空间由  $\{0, 1\}$  组成的字符串构成，原始明文消息和经过 DES 加密的密文信息是 8 个字节（64 位）的分组，密钥也是 64 位。

- 原始明文消息按 PKCS#5 (RFC 8018) 规范进行字节填充：
- 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），每个字节取值都是 08。
- 明文分组结构：  $M = m_1m_2 \cdots m_{64}$  ,  $m_i \in \{0, 1\}$  ,  $i = 1 \dots 64$ .      密文分组结构：  $C = c_1c_2 \cdots c_{64}$  ,  $c_i \in \{0, 1\}$  ,  $i = 1 \dots 64$ .
- 密钥结构：  $K = k_1k_2 \cdots k_{64}$  ,  $k_i \in \{0, 1\}$  ,  $i = 1 \dots 64$ .
- 除去  $k_8, k_{16}, \dots, k_{64}$  共 8 位奇偶校验位，起作用的仅 56 位。
- 算法过程用到的 16 个密钥记为  $K_1, K_2, \dots, K_{16}$ 。

#### 1.1.2 算法过程

DES 加密过程：

$$C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdot \cdots \cdot T_1 \cdot IP(M).$$

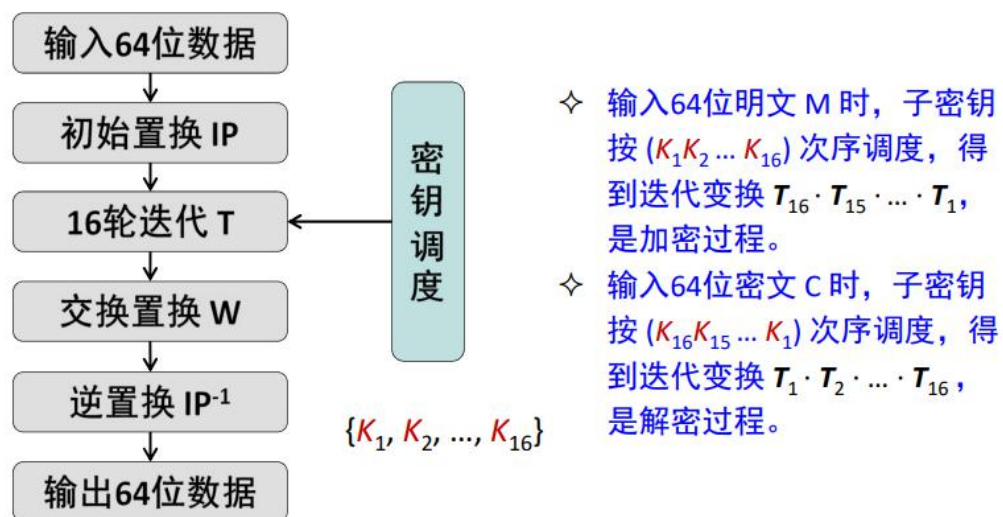
- $M$  为算法输入的 64 位明文块；
- $E_k$  描述以  $K$  为密钥的加密函数，由连续的过程复合构成；
- $IP$  为 64 位初始置换；
- $T_1, T_2, \dots, T_{16}$  是一系列的迭代变换；
- $W$  为 64 位置换，将输入的高 32 位和低 32 位交换后输出；
- $IP^{-1}$  是  $IP$  的逆置换；
- $C$  为算法输出的 64 位密文块。

DES 的解密过程与加密完全相同，只不过从后向前使用子密钥列表。

$$M = Dk(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdot \dots \cdot T_{16} \cdot IP(C).$$

## 1.2 总体结构

### ◆ DES 算法的总体结构 — Feistel 结构



## 1.3 模块分解

### 1.3.1 IP 初始置换

根据初始置换表对输入明文进行置换，置换过程就是把输入数据的第 58 位放到第 1 位，50 位放到第 2 位，42 位放到第 3 位，以此类推，初始置换以后得到的是一个 64 位的输出。

```
1. //ip 初始置换表
2. int DES::ip[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
3.                     60, 52, 44, 36, 28, 20, 12, 4,
4.                     62, 54, 46, 38, 30, 22, 14, 6,
5.                     64, 56, 48, 40, 32, 24, 16, 8,
6.                     57, 49, 41, 33, 25, 17, 9, 1,
7.                     59, 51, 43, 35, 27, 19, 11, 3,
8.                     61, 53, 45, 37, 29, 21, 13, 5,
9.                     63, 55, 47, 39, 31, 23, 15, 7 };
10.
11. //ip 置换
12. bitset<64> DES::IP(const bitset<64> & plain) {
13.     bitset<64> M;
```

```

14.     for (int i = 0; i < 64; i++) {
15.         M[i] = plain[ip[i] - 1];
16.     }
17.     return M;
18. }

```

### 1.3.2 生成子密钥

用户将初始密钥输入到子密钥生成器中, 首先经过 PC\_1 置换选择去掉 8 位奇偶校验位, 留下真正的 56bit 有效密钥。

```

1.  for (int i = 0; i < 56; i++) {
2.      real_key[i] = key[PC_1[i] - 1];
3.  }

```

接着, 分成两部分 left 和 right, 每部分 28bit, 左右两部分分别循环左移, 每部分循环左移, 每轮次根据轮次移位表左移位数不同, 然后再连接成 56bit。

```

1.  //轮次位移表
2.  int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1 };

```

再使用 PC\_2 重排并压缩为 48bit, 即得到一个子密钥。

```

1.  for (int round = 0; round < 16; round++) {
2.      left_shift(left, shiftBits[round]);
3.      left_shift(right, shiftBits[round]);
4.      //合并左右部分
5.      for (int i = 0; i < 28; ++i) {
6.          real_key[i] = right[i];
7.          real_key[i + 28] = left[i];
8.      }
9.      //使用 PC_2 重排并压缩为 48 位
10.     for (int j = 0; j < 48; j++) {
11.         subkeys[round][47 - j] = real_key[56 - PC_2[j]];
12.     }
13. }

```

进行 16 轮得到 16 个子密钥。

PC\_1 和 PC\_2 如下:

```

1.  //置换选择 PC_1
2.  //64bit 密钥转为 56bit
3.  int DES::PC_1[56] = { 57, 49, 41, 33, 25, 17, 9,

```

```

4.          1, 58, 50, 42, 34, 26, 18,
5.          10,  2, 59, 51, 43, 35, 27,
6.          19, 11,  3, 60, 52, 44, 36,
7.          63, 55, 47, 39, 31, 23, 15,
8.          7,  62, 54, 46, 38, 30, 22,
9.          14,  6, 61, 53, 45, 37, 29,
10.         21, 13,  5, 28, 20, 12,  4 };
11.
12. //置换选择 PC_2
13. //56bit 密钥转为 48bit
14. int DES::PC_2[48] = { 14, 17, 11, 24,  1,  5,
15.                        3, 28, 15,  6, 21, 10,
16.                        23, 19, 12,  4, 26,  8,
17.                        16,  7, 27, 20, 13,  2,
18.                        41, 52, 31, 37, 47, 55,
19.                        30, 40, 51, 45, 33, 48,
20.                        44, 49, 39, 56, 34, 53,
21.                        46, 42, 50, 36, 29, 32 };

```

### 1.3.3 Feistel 轮函数

F 接受两个参数：32 位的 data 和 48 位的 subkey。

E 盒拓展，使用 E 置换表将 data 从 32 位扩展为 48 位。输出的每一位为

```

1. //E 盒扩展
2. bitset<48> expand;
3. for (int i = 0; i < 48; i++) {
4.     expand[i] = data[E[i] - 1];
5. }

```

密钥异或，将得到的 48bit 输出作为输入与对应的 48bit 子密钥进行异或运算。

```

1. //异或
2. expand = expand ^ subkey;

```

S 盒压缩，将异或之后得到 48bit 作为 S 盒输入，分成 8 组，每组 6bit，分别进入 8 个 S 盒进行压缩，最终输入 48bit 变为输出 32bit。DES 共有 8 个 S 盒，每个 S 盒有 4 行 16 列，每个元素 4bit 但一般用十进制表示。S 盒接收到 6bit

的输入, 6bit 中的第一个 bit 和最后一个 bit 构成 2 位二进制数为行号, 6bit 中间的 4bit 二进制为列号, 然后根据行号和列号查该 S 盒得到对应的值, 该值的二进制就是 S 盒变换的输出。S 盒压缩是 DES 算法唯一的非线性变换, 提高了 DES 算法的安全性。

```
1. //S 盒压缩
2. bitset<32> output;
3. for (int i = 0; i < 8; i++) {
4.     int pos = 47 - i * 6;
5.     int row = expand[pos] * 2 + expand[pos - 5];
6.     int col = expand[pos - 1] * 8 + expand[pos - 2] * 4 + expand[pos - 3] *
        2 + expand[pos - 4];
7.     bitset<4> result(S_BOX[i][row][col]);
8.     for (int j = 0; j < 4; j++) {
9.         output[31 - i * 4 - j] = result[3 - j];
10.    }
11. }
```

最后进行 P 盒置换, P 置换类似于 IP 初始置换。

E 置换表和 P 置换表:

```
1. //E 盒扩展置换表
2. //将 32bit 扩展为 48bit
3. int DES::E[48] = { 32,  1,  2,  3,  4,  5,
4.                    4,  5,  6,  7,  8,  9,
5.                    8,  9, 10, 11, 12, 13,
6.                    12, 13, 14, 15, 16, 17,
7.                    16, 17, 18, 19, 20, 21,
8.                    20, 21, 22, 23, 24, 25,
9.                    24, 25, 26, 27, 28, 29,
10.                   28, 29, 30, 31, 32,  1 };
11. //P 盒置换表
12. int DES::P[32] = { 16,  7, 20, 21,
13.                   29, 12, 28, 17,
14.                   1, 15, 23, 26,
15.                   5, 18, 31, 10,
16.                   2,  8, 24, 14,
17.                   32, 27,  3,  9,
18.                   19, 13, 30,  6,
19.                   22, 11,  4, 25 };
```

其中一个 S 盒:

```

1. {
2. {14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7},
3. {0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8},
4. {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0},
5. {15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13}
6. }

```

### 1.3.4 IP 逆置换

过程与 IP 初始置换完全相同，只不过使用的置换表不同。

```

1. //IP 逆置换
2. bitset<64> DES::IP_1(const bitset<64> & input) {
3.     bitset<64> output;
4.     int ip_1[64];
5.     for (int i = 0; i < 64; i++) {
6.         ip_1[ip[i] - 1] = i + 1;
7.     }
8.     for (int i = 0; i < 64; i++) {
9.         output[i] = input[ip_1[i] - 1];
10.    }
11.    return output;
12. }

```

IP 逆置换表。

```

1. //IP 逆置换表
2. int DES::IP_1[64] = {
3.     40, 8, 48, 16, 56, 24, 64, 32,
4.     39, 7, 47, 15, 55, 23, 63, 31,
5.     38, 6, 46, 14, 54, 22, 62, 30,
6.     37, 5, 45, 13, 53, 21, 61, 29,
7.     36, 4, 44, 12, 52, 20, 60, 28,
8.     35, 3, 43, 11, 51, 19, 59, 27,
9.     34, 2, 42, 10, 50, 18, 58, 26,
10.    33, 1, 41, 9, 49, 17, 57, 25
11. };

```

## 1.4 数据结构

除 S 置换表之外的置换表都使用一维数组表示。

S 置换表使用三维数组表示，每一个 S 盒为二维数组。

密钥，数据等使用 bitset 表示，方便进行位操作。

子密钥列表使用 bitset 的一维数组表示。

## 1.5 源代码

### 1.5.1 DES. h

```
1. #pragma once
2. #include <bitset>
3. #include <string>
4. using namespace std;
5.
6. class DES {
7. private:
8.     bitset<64> key;          //密钥
9.     string keystr;          //密钥
10.    bitset<48> subkeys[16]; //子密钥
11.    static int IP[64];       //IP 初始置换表
12.    static int IP_1[64];     //IP 逆置换表
13.
14.    //生成子密钥所用置换表
15.    static int PC_1[56];     //PC_1 置换表
16.    static int PC_2[48];     //PC_2 置换表
17.
18.    //密码函数 F 所用置换表
19.    static int E[48];        //E 盒
20.    static int P[32];        //P 盒
21.    static int S_BOX[8][4][16]; //8 个 S 盒
22.
23.    // IP 置换
24.    bitset<64> IP_replace(const bitset<64> & plain);
25.    // IP 逆置换
26.    bitset<64> IP_inverse_replace(const bitset<64> & bits);
27.    // Feistel 轮函数
28.    bitset<32> F(const bitset<32> & right, const bitset<48> & subKey);
29.    // 生成子密钥
30.    void generate_subkeys(const bitset<64> & key);
31.    //左移
32.    void left_shift(bitset<28> & bits, int shift);
33.    //char 转换为 bitset
34.    bitset<64> chars_to_bitset(const char s[8]);
```

```

35. //bitset 转换为 char
36. char* bitset_to_chars(const bitset<64> bits);
37.
38. public:
39.     DES();
40.     //加密
41.     bitset<64> encrypt(const bitset<64> & plain);
42.     //解密
43.     bitset<64> decrypt(const bitset<64> & plain);
44.
45.     //设置密钥
46.     void set_key(string keystr);
47.     string get_key();
48. };

```

## 1.5.2 DES.cpp

```

1. #include "DES.h"
2.
3. //IP 初始置换表
4. int DES::IP[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
5.                     60, 52, 44, 36, 28, 20, 12, 4,
6.                     62, 54, 46, 38, 30, 22, 14, 6,
7.                     64, 56, 48, 40, 32, 24, 16, 8,
8.                     57, 49, 41, 33, 25, 17, 9, 1,
9.                     59, 51, 43, 35, 27, 19, 11, 3,
10.                    61, 53, 45, 37, 29, 21, 13, 5,
11.                    63, 55, 47, 39, 31, 23, 15, 7 };
12.
13. //IP 逆置换表
14. int DES::IP_1[64] = {
15.     40, 8, 48, 16, 56, 24, 64, 32,
16.     39, 7, 47, 15, 55, 23, 63, 31,
17.     38, 6, 46, 14, 54, 22, 62, 30,
18.     37, 5, 45, 13, 53, 21, 61, 29,
19.     36, 4, 44, 12, 52, 20, 60, 28,
20.     35, 3, 43, 11, 51, 19, 59, 27,
21.     34, 2, 42, 10, 50, 18, 58, 26,
22.     33, 1, 41, 9, 49, 17, 57, 25
23. };
24.
25.
26. //置换选择 PC_1
27. //64bit 密钥转为 56bit
28. int DES::PC_1[56] = { 57, 49, 41, 33, 25, 17, 9,

```



```

29.          1, 58, 50, 42, 34, 26, 18,
30.          10,  2, 59, 51, 43, 35, 27,
31.          19, 11,  3, 60, 52, 44, 36,
32.          63, 55, 47, 39, 31, 23, 15,
33.          7, 62, 54, 46, 38, 30, 22,
34.          14,  6, 61, 53, 45, 37, 29,
35.          21, 13,  5, 28, 20, 12,  4 };

```

```
36.
```

```
37. //置换选择 PC_2
```

```
38. //56bit 密钥转为 48bit
```

```

39. int DES::PC_2[48] = { 14, 17, 11, 24,  1,  5,
40.                        3, 28, 15,  6, 21, 10,
41.                        23, 19, 12,  4, 26,  8,
42.                        16,  7, 27, 20, 13,  2,
43.                        41, 52, 31, 37, 47, 55,
44.                        30, 40, 51, 45, 33, 48,
45.                        44, 49, 39, 56, 34, 53,
46.                        46, 42, 50, 36, 29, 32 };

```

```
47.
```

```
48. //E 盒扩展置换表
```

```
49. //将 32bit 扩展为 48bit
```

```

50. int DES::E[48] = { 32,  1,  2,  3,  4,  5,
51.                    4,  5,  6,  7,  8,  9,
52.                    8,  9, 10, 11, 12, 13,
53.                    12, 13, 14, 15, 16, 17,
54.                    16, 17, 18, 19, 20, 21,
55.                    20, 21, 22, 23, 24, 25,
56.                    24, 25, 26, 27, 28, 29,
57.                    28, 29, 30, 31, 32,  1 };

```

```
58. //P 盒置换表
```

```

59. int DES::P[32] = { 16,  7, 20, 21,
60.                    29, 12, 28, 17,
61.                    1, 15, 23, 26,
62.                    5, 18, 31, 10,
63.                    2,  8, 24, 14,
64.                    32, 27,  3,  9,
65.                    19, 13, 30,  6,
66.                    22, 11,  4, 25 };

```

```
67. //8 个 S 盒置换表
```

```
68. //将 48bit 压缩为 32bit
```

```

69. int DES::S_BOX[8][4][16] = {
70.     {
71.         {14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7},
72.         {0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8},

```

73. {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0},

74. {15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13}

75. },

76. {

77. {15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10},

78. {3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5},

79. {0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15},

80. {13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9}

81. },

82. {

83. {10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8},

84. {13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1},

85. {13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7},

86. {1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12}

87. },

88. {

89. {7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15},

90. {13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9},

91. {10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4},

92. {3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14}

93. },

94. {

95. {2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9},

96. {14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6},

97. {4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14},

98. {11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3}

99. },

100. {

101. {12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11},

102. {10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8},

103. {9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6},

104. {4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13}

105. },

106. {

107. {4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1},

108. {13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6},

109. {1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2},

110. {6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12}

111. },

112. {

113. {13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7},

114. {1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2},

115. {7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8},

116. {2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11}

```

117.     }
118. };
119.
120. DES::DES() {
121.     keystr = "";
122. }
123.
124. //加密
125. bitset<64> DES::encrypt(const bitset<64> & input) {
126.     bitset<32> left;
127.     bitset<32> right;
128.     bitset<32> next;
129.     bitset<64> output;
130.     //IP 初始置换
131.     output = IP_replace(input);
132.     //分为左右两部分
133.     for (int i = 0; i < 32; i++) {
134.         left[i] = output[i + 32];
135.         right[i] = output[i];
136.     }
137.     //生成子密钥
138.     generate_subkeys(key);
139.     //进行 16 轮 F 函数
140.     for (int i = 0; i < 16; i++) {
141.         next = right;
142.         right = left ^ F(right, subkeys[i]);
143.         left = next;
144.     }
145.     //合并
146.     for (int i = 0; i < 32; i++) {
147.         output[i] = left[i];
148.         output[i + 32] = right[i];
149.     }
150.     //IP 逆置换
151.     bitset<64> tmp = output;
152.     output = IP_inverse_replace(tmp);
153.     return output;
154. }
155.
156. //解密
157. bitset<64> DES::decrypt(const bitset<64> & input) {
158.     //解密过程从后往前使用子密钥，其余部分与加密完全相同
159.     bitset<32> left;
160.     bitset<32> right;

```

```

161.     bitset<32> next;
162.     bitset<64> output;
163.     output = IP_replace(input);
164.     for (int i = 0; i < 32; i++) {
165.         left[i] = output[i + 32];
166.         right[i] = output[i];
167.     }
168.     generate_subkeys(key);
169.     for (int i = 0; i < 16; i++) {
170.         next = right;
171.         right = left ^ F(right, subkeys[15 - i]);
172.         left = next;
173.     }
174.     for (int i = 0; i < 32; i++) {
175.         output[i] = left[i];
176.         output[i + 32] = right[i];
177.     }
178.     bitset<64> tmp = output;
179.     output = IP_inverse_replace(tmp);
180.     return output;
181. }
182.
183. //IP 置换
184. bitset<64> DES::IP_replace(const bitset<64> & input) {
185.     bitset<64> output;
186.     for (int i = 0; i < 64; i++) {
187.         output[i] = input[IP[i] - 1];
188.     }
189.     return output;
190. }
191.
192. //IP 逆置换
193. bitset<64> DES::IP_inverse_replace(const bitset<64> & input) {
194.     bitset<64> output;
195.     for (int i = 0; i < 64; i++) {
196.         output[i] = input[IP_1[i] - 1];
197.     }
198.     return output;
199. }
200.
201. //密码函数 F
202. bitset<32> DES::F(const bitset<32> & data, const bitset<48> & subkey) {
203.     //E 盒扩展
204.     bitset<48> expand;

```

```

205.     for (int i = 0; i < 48; i++) {
206.         expand[i] = data[E[i] - 1];
207.     }
208.     //子密钥异或
209.     expand = expand ^ subkey;
210.     //S 盒压缩
211.     bitset<32> output;
212.     for (int i = 0; i < 8; i++) {
213.         int pos = 47 - i * 6;
214.         int row = expand[pos] * 2 + expand[pos - 5];
215.         int col = expand[pos - 1] * 8 + expand[pos - 2] * 4 + expand[pos -
            3] * 2 + expand[pos - 4];
216.         bitset<4> result(S_BOX[i][row][col]);
217.         for (int j = 0; j < 4; j++) {
218.             output[31 - i * 4 - j] = result[3 - j];
219.         }
220.     }
221.     //P 盒置换
222.     bitset<32> tmp = output;
223.     for (int i = 0; i < 32; i++) {
224.         output[i] = tmp[P[i] - 1];
225.     }
226.     return output;
227. }
228.
229. //左移函数
230. void DES::left_shift(bitset<28> & bits, int shift) {
231.     bitset<28> temp = bits;
232.     for (int i = 0; i < 28; ++i) {
233.         bits[i] = temp[(i - shift + 28) % 28];
234.     }
235. }
236.
237. //生成子密钥
238. void DES::generate_subkeys(const bitset<64> & key) {
239.     bitset<56> real_key;
240.     bitset<28> left;
241.     bitset<28> right;
242.     //使用 PC_1 去掉奇偶校验位并重排
243.     for (int i = 0; i < 56; i++) {
244.         real_key[i] = key[PC_1[i] - 1];
245.     }
246.     //分为左右两部分
247.     for (int i = 0; i < 28; ++i) {

```

```

248.     left[i] = real_key[i + 28];
249.     right[i] = real_key[i];
250. }
251. //轮次位移表
252. int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

253. //进行十六轮
254. for (int round = 0; round < 16; round++) {
255.     left_shift(left, shiftBits[round]);
256.     left_shift(right, shiftBits[round]);
257.     //合并左右部分
258.     for (int i = 0; i < 28; ++i) {
259.         real_key[i] = right[i];
260.         real_key[i + 28] = left[i];
261.     }
262.     //使用 PC_2 重排并压缩为 48 位
263.     for (int j = 0; j < 48; j++) {
264.         subkeys[round][47 - j] = real_key[56 - PC_2[j]];
265.     }
266. }
267. }
268.
269. bitset<64> DES::chars_to_bitset(const char s[8]) {
270.     bitset<64> bits;
271.     for (int i = 0; i < 8; i++) {
272.         for (int j = 0; j < 8; j++) {
273.             bits[i * 8 + j] = ((s[7 - i] >> j) & 1);
274.         }
275.     }
276.     return bits;
277. }
278.
279. char* DES::bitset_to_chars(const bitset<64> bits) {
280.     char s[8];
281.     bitset<8> c;
282.     for (int i = 0; i < 8; i++) {
283.         for (int j = 0; j < 8; j++) {
284.             c[j] = bits[i * 8 + j];
285.         }
286.         s[7 - i] = c.to_ulong();
287.     }
288.     return s;
289. }
290.

```

```

291. void DES::set_key(string keystr) {
292.     this->keystr = keystr;
293.     key = chars_to_bitset(keystr.c_str());
294. }
295.
296. string DES::get_key() {
297.     return keystr;
298. }

```

### 1.5.3 mian.cpp

```

1.  #include <iostream>
2.  #include <fstream>
3.  #include "DES.h"
4.
5.  using namespace std;
6.
7.  int main() {
8.      DES des;
9.      ifstream in;
10.     ofstream out;
11.     int ch = 0;
12.     int run = 1;
13.     while (run) {
14.         cout << "-----" << endl;
15.         cout << "1.设置密钥" << endl;
16.         cout << "2.查看密钥" << endl;
17.         cout << "3.开始加密" << endl;
18.         cout << "4.开始解密" << endl;
19.         cout << "5.退出" << endl;
20.         cout << "-----" << endl;
21.         cin >> ch;
22.         switch (ch) {
23.             case 1: {
24.                 cout << des.get_key() << endl;
25.                 cout << "输入密钥:" << endl;
26.                 string keystr = "";
27.                 cin >> keystr;
28.                 des.set_key(keystr);
29.                 cout << "密钥设置成功" << endl;
30.                 break;
31.             }
32.             case 2: {

```

```

33.         cout << "当前密钥:" << endl;
34.         cout << des.get_key() << endl;
35.         break;
36.     }
37.     case 3: {
38.         cout << "请输入需要加密的文件:" << endl;
39.         string filename;
40.         cin >> filename;
41.         cout << "正在加密" << endl;
42.         in.open(filename, ios::binary);
43.         out.open("encrypt-" + filename, ios::binary);
44.         bitset<64> file_data;
45.         while (in.read((char*)& file_data, sizeof(file_data))) {
46.             bitset<64> cipher = des.encrypt(file_data);
47.             out.write((char*)& cipher, sizeof(cipher));
48.             file_data.reset();
49.         }
50.         in.close();
51.         out.close();
52.         cout << "加密成功" << endl;
53.         break;
54.     }
55.     case 4: {
56.         cout << "请输入需要解密的文件:" << endl;
57.         string filename;
58.         cin >> filename;
59.         cout << "正在解密" << endl;
60.         in.open(filename, ios::binary);
61.         out.open("decrypt-" + filename, ios::binary);
62.         bitset<64> file_data;
63.         while (in.read((char*)& file_data, sizeof(file_data))) {
64.             bitset<64> plain = des.decrypt(file_data);
65.             out.write((char*)& plain, sizeof(plain));
66.             file_data.reset();
67.         }
68.         in.close();
69.         out.close();
70.         cout << "解密成功" << endl;
71.         break;
72.     }
73.     case 5: {
74.         run = 0;
75.     }
76.     default:

```



```

77.         break;
78.     }
79. }
80.     return 0;
81. }

```

## 1.6 运行结果

可以处理任意类型的文件

