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 = m1m2 … m64 , mi {0, 1}, i = 1 .. 64. 密文分组结构：C = c1c2 … c64 , ci {0, 1}, i = 1 .. 64.
- 密钥结构：K = k1k2 … k64 , ki {0, 1}, i = 1 .. 64.
- 除去 k8, k16, …, k64 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 K1, K2, …, K16。

1.1.2 算法过程

DES 加密过程：

C = Ek(M) = IP-1 • W • T16 • T15 •… • T1 • IP(M).

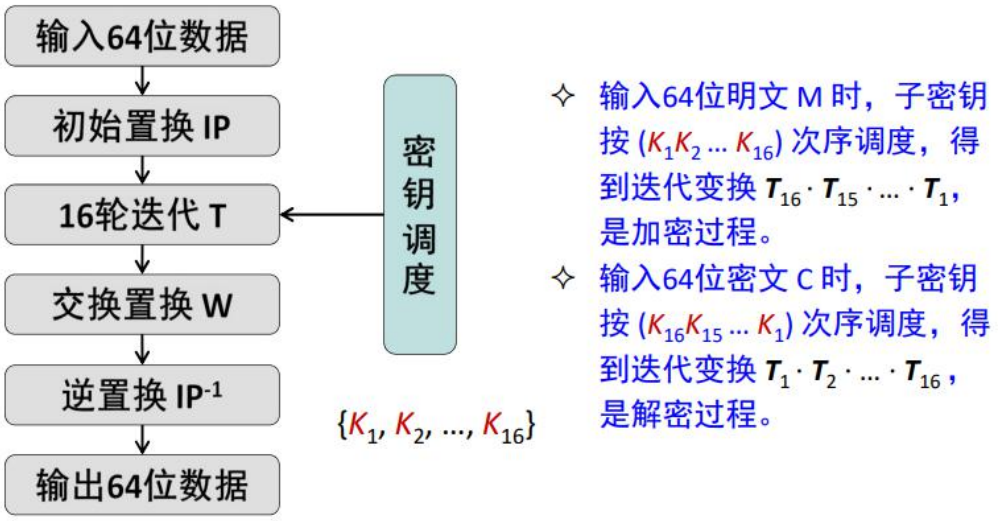
- M 为算法输入的 64 位明文块；
- Ek 描述以 K 为密钥的加密函数，由连续的过程复合构成；
- IP 为 64 位初始置换；
- T1, T2 , …, T16 是一系列的迭代变换；
- W 为 64 位置换，将输入的高 32 位和低 32 位交换后输出；
- IP-1 是 IP 的逆置换；
- C 为算法输出的 64 位密文块。

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 • W • T1 • T2 • … • T16 • 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。

```
//ip 初始置换表
int DES::ip[64] = { 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
bitset<64> DES::IP(const bitset<64> & plain) {
    bitset<64> M;
```

```
    for (int i = 0; i < 64; i++) {
            M[i] = plain[ip[i] - 1];
    }
    return M;
}
```

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。

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

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。

```
//轮次位移表
int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 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。

```
for (int round = 0; round < 16; round++) {
    left_shift(left, shiftBits[round]);
    left_shift(right, shiftBits[round]);
    //合并左右部分
    for (int i = 0; i < 28; ++i) {
        real_key[i] = right[i];
        real_key[i + 28] = left[i];
    }
    //使用 PC_2 重排并压缩为 48 位
    for (int j = 0; j < 48; j++) {
        subkeys[round][47 - j] = real_key[56 - PC_2[j]];
    }
}
```

进行 16 轮得到 16 个子密钥。

PC_1 和 PC_2 如下：

```
//置换选择 PC_1
//64bit 密钥转为 56bit
int DES::PC_1[56] = { 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 };

//置换选择 PC_2
//56bit 密钥转为 48bit
int DES::PC_2[48] = { 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 };
```

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

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

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。

```
//异或
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。

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

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：

```
//E 盒扩展置换表
//将 32bit 扩展为 48bit
int DES::E[48] = { 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 };
//P 盒置换表
int DES::P[32] = { 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 };
```

其中一个 S 盒：

```
{
{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}
}
```

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。

```
//IP 逆置换
bitset<64> DES::IP_1(const bitset<64> & input) {
    bitset<64> output;
    int ip_1[64];
    for (int i = 0; i < 64; i++) {
        ip_1[ip[i] - 1] = i + 1;
    }
    for (int i = 0; i < 64; i++) {
        output[i] = input[ip_1[i] - 1];
    }
    return output;
}
```

IP 逆置换表。

```
//IP 逆置换表
int DES::IP_1[64] = {
    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
};
```

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

```
#pragma once
#include <bitset>
#include <string>
using namespace std;

class DES {
private:
    bitset<64> key;         //密钥
    string keystr;          //密钥
    bitset<48> subkeys[16];//子密钥
    static int IP[64];      //IP 初始置换表
    static int IP_1[64];    //IP 逆置换表

    //生成子密钥所用置换表
    static int PC_1[56];    //PC_1 置换表
    static int PC_2[48];    //PC_2 置换表

    //密码函数 F 所用置换表
    static int E[48];       //E 盒
    static int P[32];       //P 盒
    static int S_BOX[8][4][16]; //8 个 S 盒

    // IP 置换
    bitset<64> IP_replace(const bitset<64> & plain);
    // IP 逆置换
    bitset<64> IP_inverse_replace(const bitset<64> & bits);
    // Feistel 轮函数
    bitset<32> F(const bitset<32> & right, const bitset<48> & subKey);
    // 生成子密钥
    void generate_subkeys(const bitset<64> & key);
    //左移
    void left_shift(bitset<28> & bits, int shift);
    //char 转换为 bitset
    bitset<64> chars_to_bitset(const char s[8]);
```

```
    //bitset 转换为 char
    char* bitset_to_chars(const bitset<64> bits);

public:
    DES();
    //加密
    bitset<64> encrypt(const bitset<64> & plain);
    //解密
    bitset<64> decrypt(const bitset<64> & plain);

    //设置密钥
    void set_key(string keystr);
    string get_key();
};
```

1.5.2 DES.cpp

```
#include "DES.h"

//IP 初始置换表
int DES::IP[64] = { 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 逆置换表
int DES::IP_1[64] = {
    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
};


//置换选择 PC_1
//64bit 密钥转为 56bit
int DES::PC_1[56] = { 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 };

//置换选择 PC_2
//56bit 密钥转为 48bit
int DES::PC_2[48] = { 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 };

//E 盒扩展置换表
//将 32bit 扩展为 48bit
int DES::E[48] = { 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 };
//P 盒置换表
int DES::P[32] = { 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 };
//8 个 S 盒置换表
//将 48bit 压缩为 32bit
int DES::S_BOX[8][4][16] = {
    {
        {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}
```

```
    }
};

DES::DES() {
    keystr = "";
}

//加密
bitset<64> DES::encrypt(const bitset<64> & input) {
    bitset<32> left;
    bitset<32> right;
    bitset<32> next;
    bitset<64> output;
    //IP 初始置换
    output = IP_replace(input);
    //分为左右两部分
    for (int i = 0; i < 32; i++) {
        left[i] = output[i + 32];
        right[i] = output[i];
    }
    //生成子密钥
    generate_subkeys(key);
    //进行 16 轮 F 函数
    for (int i = 0; i < 16; i++) {
        next = right;
        right = left ^ F(right, subkeys[i]);
        left = next;
    }
    //合并
    for (int i = 0; i < 32; i++) {
        output[i] = left[i];
        output[i + 32] = right[i];
    }
    //IP 逆置换
    bitset<64> tmp = output;
    output = IP_inverse_replace(tmp);
    return output;
}

//解密
bitset<64> DES::decrypt(const bitset<64> & input) {
    //解密过程从后往前使用子密钥，其余部分与加密完全相同
    bitset<32> left;
    bitset<32> right;
```

```
    bitset<32> next;
    bitset<64> output;
    output = IP_replace(input);
    for (int i = 0; i < 32; i++) {
        left[i] = output[i + 32];
        right[i] = output[i];
    }
    generate_subkeys(key);
    for (int i = 0; i < 16; i++) {
        next = right;
        right = left ^ F(right, subkeys[15 - i]);
        left = next;
    }
    for (int i = 0; i < 32; i++) {
        output[i] = left[i];
        output[i + 32] = right[i];
    }
    bitset<64> tmp = output;
    output = IP_inverse_replace(tmp);
    return output;
}

//IP 置换
bitset<64> DES::IP_replace(const bitset<64> & input) {
    bitset<64> output;
    for (int i = 0; i < 64; i++) {
        output[i] = input[IP[i] - 1];
    }
    return output;
}

//IP 逆置换
bitset<64> DES::IP_inverse_replace(const bitset<64> & input) {
    bitset<64> output;
    for (int i = 0; i < 64; i++) {
        output[i] = input[IP_1[i] - 1];
    }
    return output;
}

//密码函数 F
bitset<32> DES::F(const bitset<32> & data, const bitset<48> & subkey) {
    //E 盒扩展
    bitset<48> expand;
```

```
    for (int i = 0; i < 48; i++) {
        expand[i] = data[E[i] - 1];
    }
    //子密钥异或
    expand = expand ^ subkey;
    //S 盒压缩
    bitset<32> output;
    for (int i = 0; i < 8; i++) {
        int pos = 47 - i * 6;
        int row = expand[pos] * 2 + expand[pos - 5];
        int col = expand[pos - 1] * 8 + expand[pos - 2] * 4 + expand[pos - 3] * 2 + expand[pos - 4];
        bitset<4> result(S_BOX[i][row][col]);
        for (int j = 0; j < 4; j++) {
            output[31 - i * 4 - j] = result[3 - j];
        }
    }
    //P 盒置换
    bitset<32> tmp = output;
    for (int i = 0; i < 32; i++) {
        output[i] = tmp[P[i] - 1];
    }
    return output;
}

//左移函数
void DES::left_shift(bitset<28> & bits, int shift) {
    bitset<28> temp = bits;
    for (int i = 0; i < 28; ++i) {
        bits[i] = temp[(i - shift + 28) % 28];
    }
}

//生成子密钥
void DES::generate_subkeys(const bitset<64> & key) {
    bitset<56> real_key;
    bitset<28> left;
    bitset<28> right;
    //使用 PC_1 去掉奇偶校验位并重排
    for (int i = 0; i < 56; i++) {
        real_key[i] = key[PC_1[i] - 1];
    }
    //分为左右两部分
    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, 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: {
```

```
            cout << "当前密钥:" << endl;
            cout << des.get_key() << endl;
            break;
        }
        case 3: {
            cout << "请输入需要加密的文件:" << endl;
            string filename;
            cin >> filename;
            cout << "正在加密" << endl;
            in.open(filename, ios::binary);
            out.open("encrypt-" + filename, ios::binary);
            bitset<64> file_data;
            while (in.read((char*)& file_data, sizeof(file_data))) {
                bitset<64> cipher = des.encrypt(file_data);
                out.write((char*)& cipher, sizeof(cipher));
                file_data.reset();
            }
            in.close();
            out.close();
            cout << "加密成功" << endl;
            break;
        }
        case 4: {
            cout << "请输入需要解密的文件:" << endl;
            string filename;
            cin >> filename;
            cout << "正在解密" << endl;
            in.open(filename, ios::binary);
            out.open("decrypt-" + filename, ios::binary);
            bitset<64> file_data;
            while (in.read((char*)& file_data, sizeof(file_data))) {
                bitset<64> plain = des.decrypt(file_data);
                out.write((char*)& plain, sizeof(plain));
                file_data.reset();
            }
            in.close();
            out.close();
            cout << "解密成功" << endl;
            break;
        }
        case 5: {
            run = 0;
        }
        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