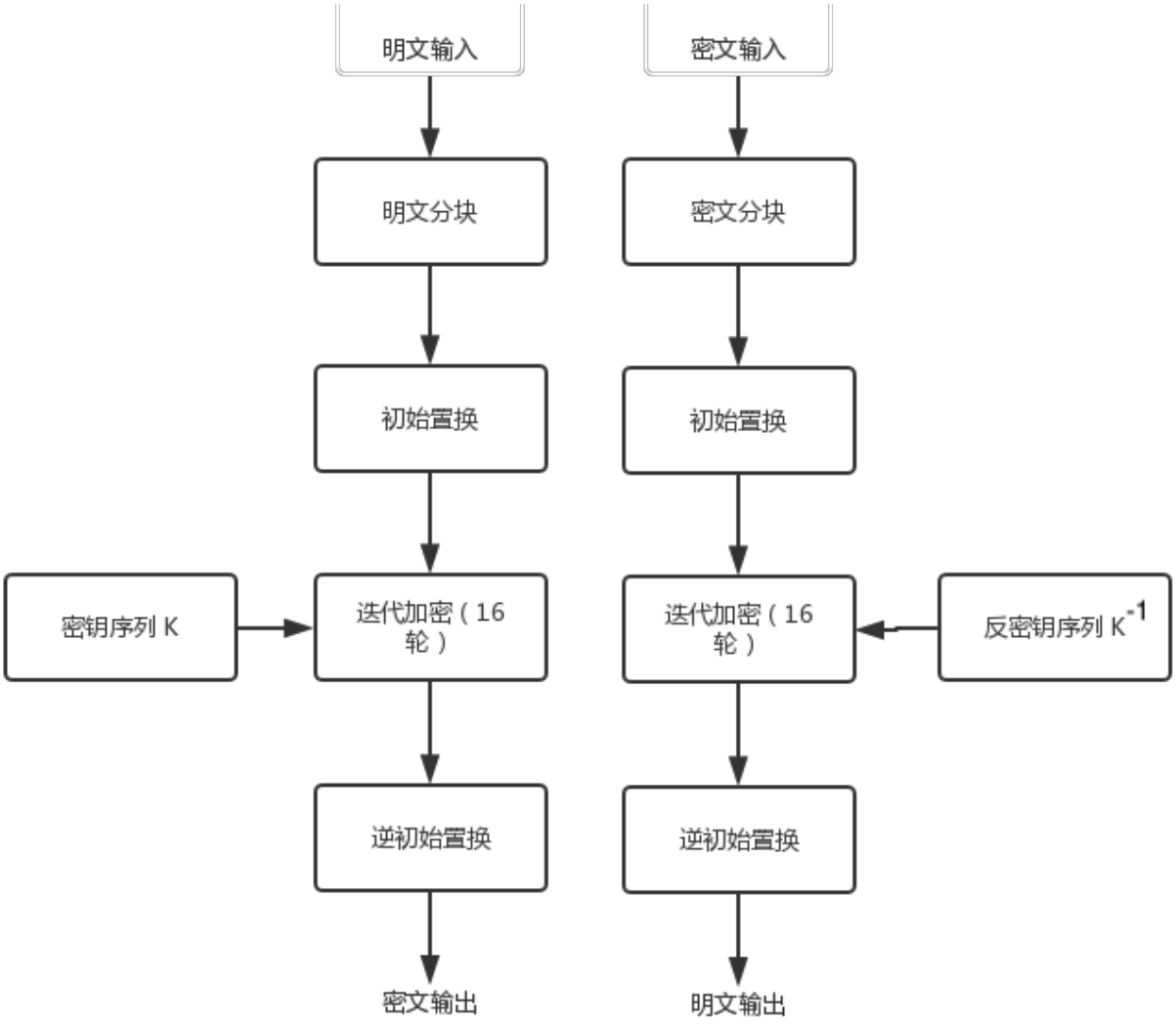
# DES加密实验报告

| 姓名 | 学号 | 联系方式 |
| --- | --- | --- |
| 张家豪 | 16337303 | 994328597@qq.com |

## 一、实验内容

1. 完成DES的加密和解密
2. 制作GUI

## 二、算法流程



# 三、算法原理

DES算法属于对称加密算法，而且加密和解密的算法和密钥也是相同。

## 1、置换原理

对于64位明文 $M = m_1 m_2 \cdots m_{64}$ ，按照下图所示的初始置换IP表指定的顺序从输出中取出指定的位放在这一位上，得到输出，对于置换后的序列 $C$ , $c_1 = m_{58}, c_2 = m_{50}, c3 = \ldots$ 以此类推。

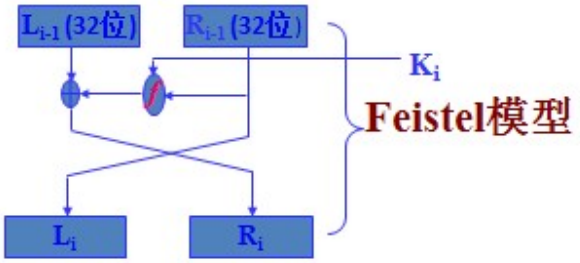
| 初 始 置 换 IP | | | | | | | |
|---|---|---|---|---|---|---|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

逆初始置换是初始置换的逆变换，用表格表示如下图，

| 逆初始置换 $IP^{-1}$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

## 2、迭代加密原理

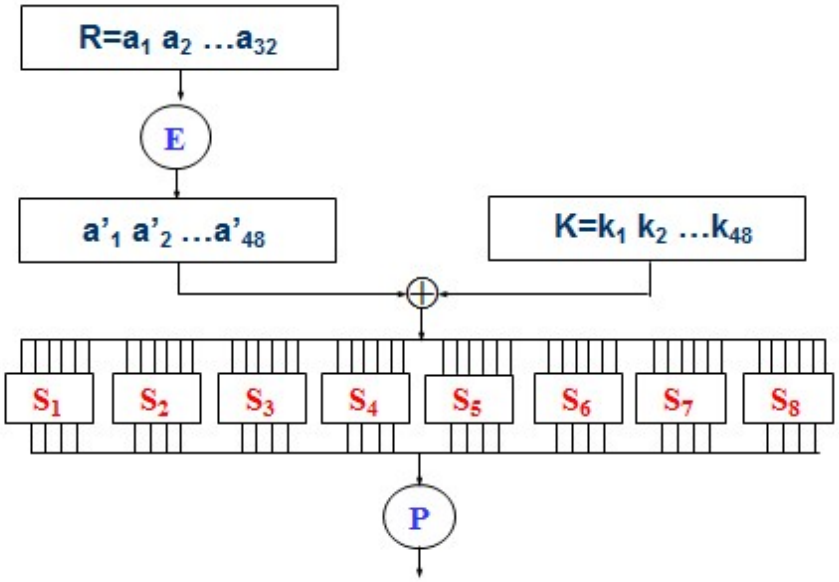
DES的迭代加密用的是Feistel模型




每一圈迭代过程中：

- 输出的左32位就是输入的右32位
- 输出的右32位是输入的右32位和圈密钥 $K_i$ 进行 $f$ 函数运算后和输入左32位异或运算得到。

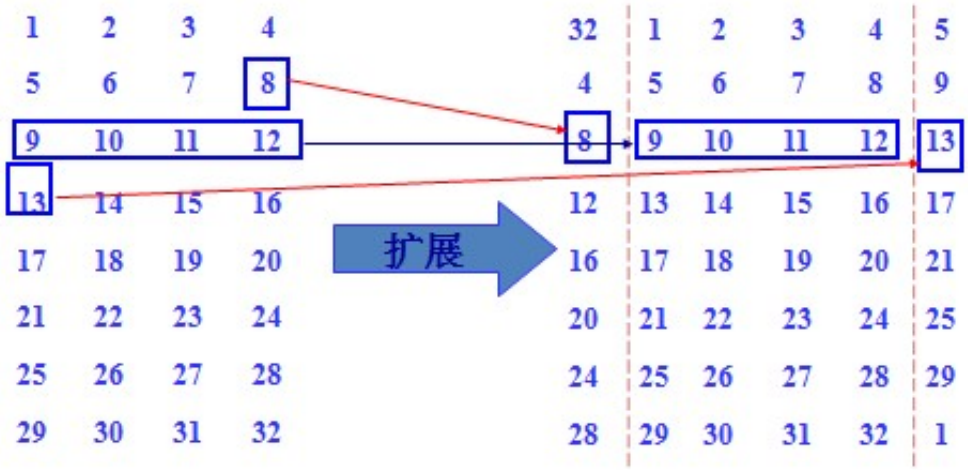
其中 $f$ 函数的数学表达式如下：$f(R,K)=PS(E(R)\oplus K$




- 先对右32位输入进行E盒运算得到48位输出
- 再把运算结果和48位的圈密钥进行异或之后输入到s1,s2...s8盒中
- 每个S盒都是6进4出（输入为6比特，输出4比特），综合得到32位输出
- 把结果输入到P盒中，进行P盒置换，最后得到32位输出。

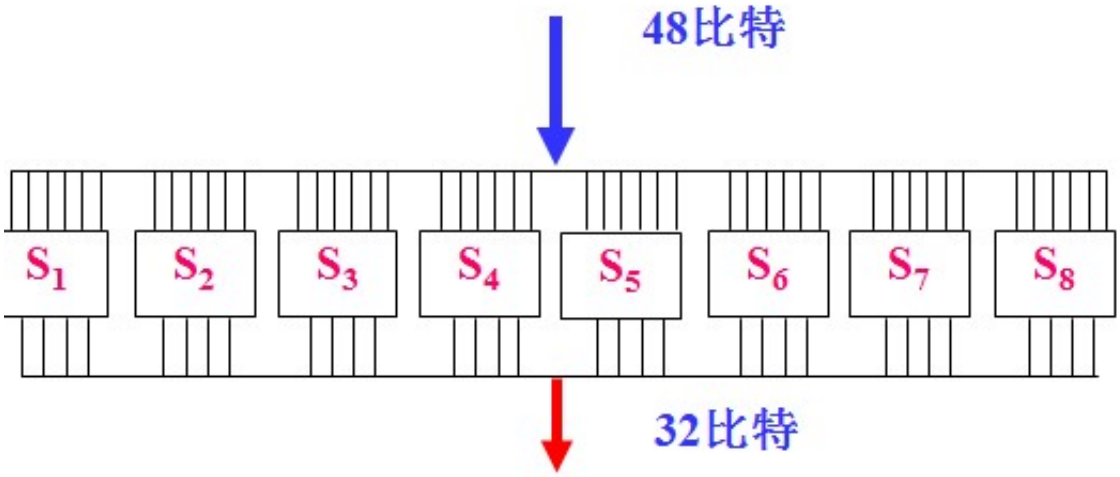
### 2.1 E盒拓展

E盒扩展的作用是把32比特的输入扩展成48比特，扩展的方式如下图所示，把输入的32比特从左到右编号为1,2,3.。。。32，并把 这32比特写成每行4个，共8行的形式。然后把第i-1行最右比特和第i+1行最左比特添加到第i行的左边和右边，这样就得到了48位的输出。（即c1c2c3c4c5c6 = m32m1m2m3m4m5 .... 以此类推)




**2.5 S盒压缩**

S盒压缩时，我们将48位输入按6位划分位8组，每组的第一位盒最后一位构成对应S盒的行号，中间4位构成对应S盒的列号。用对应S盒中的对应4比特数替代原来的6比特数。



8个6进4出S盒

| 行 \ 列 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_1$ | 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| | 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| | 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| | 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
| $S_2$ | 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| | 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| | 2 | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| | 3 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
| $S_3$ | 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| | 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| | 2 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| | 3 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
| $S_4$ | 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| | 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| | 2 | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| | 3 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
| $S_5$ | 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| | 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| | 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| | 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
| $S_6$ | 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| | 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| | 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| | 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
| $S_7$ | 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| | 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| | 2 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| | 3 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |
| $S_8$ | 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| | 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| | 2 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| | 3 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

# 四、实验结果

1. **加密过程**：
    - 密钥为**张家豪**
    - 明文为**密码学真是一门有意思的学科**
    - 加密密文为**5bf192db0942934a9ad336d69826fa3f2e096be62b9b72529b1af2bb037617aa125f7abad8da4a00**


2. 解密过程：
    - 密钥为张家豪
    - 密文为**5bf192db0942934a9ad336d69826fa3f2e096be62b9b72529b1af2bb037617aa125f7abad8da4a00**
    - 解密明文为**密码学真是一门有意思的学科**


3. 错误解密过程：
    - 密钥为**张家**
    - 密文为**5bf192db0942934a9ad336d69826fa3f2e096be62b9b72529b1af2bb037617aa125f7abad8da4a00**
    - 解密明文为**磥 ﹐﹐﹐﹐﹐&﹐﹐﹐﹐﹐﹐﹐h握﹐﹐﹐﹐@﹐窄%﹐﹐秣**

我们看到，如果密钥缺失或错误，我们无法正确解码

# 五、代码展示

## 1、GUI模块

1. mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QApplication>


namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

2. mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "des.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
```

```
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{
    std::string key_ = (ui->lineEdit->text()).toStdString();
    std::string message_ = (ui->plainTextEdit->toPlainText()).toStdString();
    bits key = genKey(key_);
    vector<bits> message = string2bytes(message_);
    vector<bits> cipher = doEncrypt(message, key);
    QString cipher_ = QString::fromStdString(bytes2hex(cipher));
    ui->textBrowser->setText(cipher_);
}

void MainWindow::on_pushButton_2_clicked()
{
    std::string key_ = (ui->lineEdit->text()).toStdString();
    std::string cripher_ = (ui->plainTextEdit->toPlainText()).toStdString();
    bits key = genKey(key_);
    vector<bits> cipher = hex2bytes(cripher_);
    vector<bits> message = doDecrypt(cipher, key);
    QString message_ = QString::fromStdString(bytes2string(message));
    ui->textBrowser->setText(message_);
}
```

## 2、DES算法模块

1. des.h(有些调试用的代码没有删除)

```
//
//  DES.h
//  DES
//
//  Created by 张家豪 on 2018/12/7.
//  Copyright © 2018 张家豪. All rights reserved.
//

#ifndef DES_H
#define DES_H
#include <string>
#include <iostream>
```

```cpp
#include <iterator>
#include <vector>
#include <algorithm>
#include "DEStable.h"

using namespace std;
typedef vector<bool> bits;


bits charToByte(const char *In, int bit_count) {
    bits Out;
    for (int i = 0; i < bit_count; i++)
        Out.push_back( (In[i / 8] >> (i % 8)) & 1);
    return Out;
}

void charToByte(bits Out, const char *In, int bit_count) {
    for (int i = 0; i < bit_count; i++)
        Out[i] = (In[i / 8] >> (i % 8)) & 1;
}

//从bool数到char数组

void byteToChar(char *Out, const bool *In, int bits) {
    memset(Out, 0, (bits + 7) / 8);
    for (int i = 0; i < bits; i++)
    {
        Out[i / 8] |= In[i] << (i % 8);
    }
}


ostream & operator<<(ostream &out, vector<bool> tmp) {
    for(int i = 0, size = tmp.size(); i < size; i++){
        out << tmp[i] << " ";
        if((i+1) % 8 == 0)
            out << endl;
    }
    return out;
}

ostream & operator<<(ostream &out, vector<bits> tmp) {
    for(int i = 0, size = tmp.size(); i < size; i++)
        out << tmp[i] << endl;
    return out;
}

bits operator ^ (const bits &tmp, const bits & p){//操作必须位数相同
    bits res;
```

```
    for(int i = 0, size = tmp.size(); i < size; i++)
        res.push_back(tmp[i] ^ p[i]);
    return res;
}

bits operator>>(const bits& tmp, int shift){
    vector<bool> res(tmp);
    for(int i = 0; i < shift; i++){
        bool t = res.back();
        res.pop_back();
        res.insert(res.begin(), t);
    }
    return res;
}

bits operator<<(const bits& tmp, int shift){
    vector<bool> res(tmp);
    for(int i = 0; i < shift; i++){
        bool t = res.front();
        res.erase(res.begin());
        res.push_back(t);
    }
    return res;
}

bits char2byte(const string in, int bit){
    vector<bool> res;
    string tmp = in + "        "; //至少8位
    for(int i = 0; i < bit; i++){
        //        int p = tmp[i / 8];
        //        cout << p << ":" << ((tmp[i / 8] >> (i % 8)) &
1) << endl;
        res.push_back( (tmp[i / 8] >> (i % 8)) & 1);
    }
    return res;
}

bits bits_merge(const bits& left, const bits& right){
    //    cout << "left:" << endl << left << endl;
    bits res = left;
    //    cout << res << endl;
    for(int i = 0, size = right.size(); i < size; i++)
        res.push_back(right[i]);
    return res;
}

bits Extend(const int *extendTable, bits input){ //32位扩展
    bits res;
    for(int i = 0, size = input.size(); i < size; i++){
```

```
        res.push_back(input[extendTable[i]-1]);
    }
    return res;
}

bits Pbox_f(const int *IPTable, bits input, int bit_count){ //
    bits res;
    for(int i = 0; i < bit_count; i++){
        res.push_back(input[IPTable[i]-1]);
    }
    return res;
}

bits Sbox_f(const bits &input){//48位输入，32位输出
    bits res;
    for(int i = 0; i < 8; i=i+6){
        int row = (input[i] << 1) + (input[i+5]);
        int column = (input[i+1] << 3) + (input[i+2] << 2) +
(input[i+3] << 1) + input[i+4];
        int tmp = SBox[i][row][column];
        for(int i = 0 ; i < 4; i++){
            res.insert(res.begin(), (tmp & 1));
            tmp >>= 1;
        }
    }
    return res;
}

bits circle_fun(bits input, bits key) {//圈函数
    bits Extended = Extend(extendTable, input); //E盒拓展变48位
    Extended = Extended ^ key;  //与48位圈密钥异或
    bits res = Sbox_f(Extended);    //S盒压缩为32位
    return Pbox_f(PTable, res, 32); //P盒置换，返回32位结果
}

bits encrypt(bits input_, bits key, int round_time){
    bits res;
    //    cout << "明文输入: " << endl << input_ << endl;
    bits input = Pbox_f(initIPTable, input_, 64);
    //    cout << "初始置换: " << endl << input << endl;

    auto mid = input.begin();
    for(int i = 0; i < 32; i++, mid++);
    bits left = bits(input.begin(), mid);
    bits right = bits(mid, input.end());
    //    cout << "L0: " << endl << left << endl << "R0:" << endl
<< right << endl;

    bits key56 = Pbox_f(PC_1Table, key, 56);
```

```
    mid = key56.begin();
    for(int i = 0; i < 28; i++, mid++);
    bits key_left = bits(key56.begin(), mid);
    bits key_right = bits(mid, key56.end());
    bits key_tmp1 = bits_merge(key_left, key_right);

    for(int i = 0; i < round_time; i++){
        key_left = key_left << (shiftTable[i]);
        key_right = key_right << (shiftTable[i]);
        bits key_tmp = bits_merge(key_left, key_right);
        bits key48 = Pbox_f(PC_2Table, key_tmp, 48);

        bits tmp = left;
        left = right;
        right = circle_fun(right, key48) ^ tmp;
        //        cout << "M" << i+1 << ":" << endl << bits_merge(left, right) << endl;
        //        cout << "L" << i+1 << ": " << endl << left << endl << "R" << i+1 << ":" << endl << right << endl;
    }
    res = bits_merge(right, left);
    res = Pbox_f(inverseIPTable, res, 64);
    //    cout << "逆初始置换:" << endl << res << endl;
    return res;
}

bits decrypt(bits input_, bits key, int round_time){
    bits res;
    //    cout << "密文输入: " << endl << input_ << endl;
    bits input = Pbox_f(initIPTable, input_, 64); //初始置换
    //    cout << "初始置换: " << endl << input << endl;

    auto mid = input.begin();
    for(int i = 0; i < 32; i++, mid++);
    bits left = bits(input.begin(), mid);
    bits right = bits(mid, input.end());
    //    cout << "L0: " << endl << left << endl << "R0:" << endl << right << endl;

    bits key56 = Pbox_f(PC_1Table, key, 56);
    mid = key56.begin();
    for(int i = 0; i < 28; i++, mid++);
    bits key_left = bits(key56.begin(), mid);
    bits key_right = bits(mid, key56.end());
    key_left = key_left << 29;
    key_right = key_right << 29;

    bits key_tmp1 = bits_merge(key_left, key_right);

```

```
    for(int i = 0; i < round_time; i++){
        key_left = key_left >> (shiftTable[(16-i)%16]);
        key_right = key_right >> (shiftTable[(16-i)%16]);
        bits key_tmp = bits_merge(key_left, key_right);
        bits key48 = Pbox_f(PC_2Table, key_tmp, 48);

        bits tmp = left;
        left = right;
        right = circle_fun(right, key48) ^ tmp;
        //        cout << "M" << 16-i << ":" << endl<< bits_merge(left, right) << endl;
        //        cout << "L" << i+1 << ": " << endl << left << endl << "R" << i+1 << ":" << endl << right << endl;
    }
    res = bits_merge(right, left);
    res = Pbox_f(inverseIPTable, res, 64); //初始置换逆置换
    //    cout << "逆初始置换:" << endl << res << endl;
    return res;
}

bits genKey(string key){
    string tmp = key + "        ";//补全不足64位的密钥
    string realKey = tmp.substr(0, 8);
    return char2byte(realKey, 64);
}

vector<bits> string2bytes(string Message){
    vector<bits> res;
    int i = 0;
    for(int size = Message.size(); i + 7 < size; i = i + 8){
        res.push_back(char2byte(Message.substr(i, i+8), 64));
    }
    if((Message.size() % 8) == 0)
        return res;
    else{
        string last = Message.substr(i);
        char tmp = 0;
        while(last.size() != 8)
            last += tmp;
        res.push_back(char2byte(last, 64));
    }
    return res;
}

string b2s(bits Cipher, int bits_count){
    int tmp[8] = {0};
    string res;
    for (int i = 0; i < bits_count; i++){
        int temp = Cipher[i];
```

```
        tmp[i / 8] |= temp << (i % 8);
        //        cout << tmp[i/8] << endl;
    }
    for(int i = 0; i < 8; i++)
        res.push_back(tmp[i]);
    return res;
}

string bytes2string(vector<bits> Cipher){
    string res;
    for(int i = 0,size = Cipher.size(); i < size; i++){
        res += b2s(Cipher[i], 64);
    }
    return res;
}

vector<bits> doEncrypt(vector<bits> Message, bits key){
    vector<bits> res;
    for(int i = 0, size = Message.size(); i < size; i++)
        res.push_back(encrypt(Message[i], key, 16));
    return res;
}

vector<bits> doDecrypt(vector<bits> C, bits key){
    vector<bits> res;
    for(int i = 0, size = C.size(); i < size; i++)
        res.push_back(decrypt(C[i], key, 16) );
    return res;
}

bits hex2byte(string input){
    bits res;
    cout << input << endl;
    for(int i = 0, j = 3; i < 64; i++){
        int tmp = input[i/4];
        if(tmp <= '9' && tmp >= '0')
            tmp -= '0';
        else
            tmp -= 'a' - 10;
        res.push_back( (tmp >> j) & 1);
        j--;
        if(j == -1)
            j = 3;
    }
    return res;
}

vector<bits> hex2bytes(string input){
    vector<bits> res;
```

```
    for(int i = 0,size = input.size(); i < size; i = i+16){
        string tmp = input.substr(i,16);
        res.push_back(hex2byte(tmp));
    }
    return res;
}

string byte2hex(bits input){
    string res = "";
    for(int i = 0,size = input.size(); i < size; i=i+4){
        char tmp = 0;
        for(int j = 0; j < 4;j++){
            tmp |= int(input[i+j]) << (3-j);
        }
        if(tmp < 10)
            res += tmp +'0';
        else
            res += (tmp-10) + 'a';
    }
    return res;
}

string bytes2hex(vector<bits> input){
    string res;
    for(int i = 0,size = input.size(); i < size; i++){
        res += byte2hex(input[i]);
    }
    return res;
}
#endif /* DES_h */

```

## 3、置换表（包括64位置换表、32位置换表、P盒、S盒）

1. destable.h

```
//
//  DES.h
//  DES
//
//  Created by 张家豪 on 2018/12/7.
//  Copyright © 2018 张家豪. All rights reserved.
//

#ifndef DES_H
#define DES_H
#include <string>
#include <iostream>
#include <iterator>
```

```cpp
#include <vector>
#include <algorithm>
#include "DEStable.h"

using namespace std;
typedef vector<bool> bits;


bits charToByte(const char *In, int bit_count) {
    bits Out;
    for (int i = 0; i < bit_count; i++)
        Out.push_back( (In[i / 8] >> (i % 8)) & 1);
    return Out;
}

void charToByte(bits Out, const char *In, int bit_count) {
    for (int i = 0; i < bit_count; i++)
        Out[i] = (In[i / 8] >> (i % 8)) & 1;
}

//从bool数到char数组

void byteToChar(char *Out, const bool *In, int bits) {
    memset(Out, 0, (bits + 7) / 8);
    for (int i = 0; i < bits; i++)
    {
        Out[i / 8] |= In[i] << (i % 8);
    }
}


ostream & operator<<(ostream &out, vector<bool> tmp) {
    for(int i = 0, size = tmp.size(); i < size; i++){
        out << tmp[i] << " ";
        if((i+1) % 8 == 0)
            out << endl;
    }
    return out;
}

ostream & operator<<(ostream &out, vector<bits> tmp) {
    for(int i = 0, size = tmp.size(); i < size; i++)
        out << tmp[i] << endl;
    return out;
}

bits operator ^ (const bits &tmp, const bits & p){//操作必须位数相同
    bits res;
    for(int i = 0, size = tmp.size(); i < size; i++)
```

```
63          res.push_back(tmp[i] ^ p[i]);
64      return res;
65  }
66
67  bits operator>>(const bits& tmp, int shift){
68      vector<bool> res(tmp);
69      for(int i = 0; i < shift; i++){
70          bool t = res.back();
71          res.pop_back();
72          res.insert(res.begin(), t);
73      }
74      return res;
75  }
76
77  bits operator<<(const bits& tmp, int shift){
78      vector<bool> res(tmp);
79      for(int i = 0; i < shift; i++){
80          bool t = res.front();
81          res.erase(res.begin());
82          res.push_back(t);
83      }
84      return res;
85  }
86
87  bits char2byte(const string in, int bit){
88      vector<bool> res;
89      string tmp = in + "        "; //至少8位
90      for(int i = 0; i < bit; i++){
91          //        int p = tmp[i / 8];
92          //        cout << p << ":" << ((tmp[i / 8] >> (i % 8)) &
    1) << endl;
93          res.push_back( (tmp[i / 8] >> (i % 8)) & 1);
94      }
95      return res;
96  }
97
98  bits bits_merge(const bits& left, const bits& right){
99      //    cout << "left:" << endl << left << endl;
100     bits res = left;
101     //    cout << res << endl;
102     for(int i = 0, size = right.size(); i < size; i++)
103         res.push_back(right[i]);
104     return res;
105 }
106
107 bits Extend(const int *extendTable, bits input){ //32位扩展
108     bits res;
109     for(int i = 0, size = input.size(); i < size; i++){
110         res.push_back(input[extendTable[i]-1]);
```

```
111     }
112     return res;
113 }
114 
115 bits Pbox_f(const int *IPTable, bits input, int bit_count){ //
116     bits res;
117     for(int i = 0; i < bit_count; i++){
118         res.push_back(input[IPTable[i]-1]);
119     }
120     return res;
121 }
122 
123 bits Sbox_f(const bits &input){//48位输入，32位输出
124     bits res;
125     for(int i = 0; i < 8; i=i+6){
126         int row = (input[i] << 1) + (input[i+5]);
127         int column = (input[i+1] << 3) + (input[i+2] << 2) +
(input[i+3] << 1) + input[i+4];
128         int tmp = SBox[i][row][column];
129         for(int i = 0 ; i < 4; i++){
130             res.insert(res.begin(), (tmp & 1));
131             tmp >>= 1;
132         }
133     }
134     return res;
135 }
136 
137 bits circle_fun(bits input, bits key) {//圈函数
138     bits Extended = Extend(extendTable, input); //E盒拓展变48位
139     Extended = Extended ^ key;  //与48位圈密钥异或
140     bits res = Sbox_f(Extended);    //S盒压缩为32位
141     return Pbox_f(PTable, res, 32); //P盒置换，返回32位结果
142 }
143 
144 bits encrypt(bits input_, bits key, int round_time){
145     bits res;
146     //    cout << "明文输入: " << endl << input_ << endl;
147     bits input = Pbox_f(initIPTable, input_, 64);
148     //    cout << "初始置换: " << endl << input << endl;
149 
150     auto mid = input.begin();
151     for(int i = 0; i < 32; i++, mid++);
152     bits left = bits(input.begin(), mid);
153     bits right = bits(mid, input.end());
154     //    cout << "L0: " << endl << left << endl << "R0:" << endl
<< right << endl;
155 
156     bits key56 = Pbox_f(PC_1Table, key, 56);
157     mid = key56.begin();
```

```
    for(int i = 0; i < 28; i++, mid++);
    bits key_left = bits(key56.begin(), mid);
    bits key_right = bits(mid, key56.end());
    bits key_tmp1 = bits_merge(key_left, key_right);

    for(int i = 0; i < round_time; i++){
        key_left = key_left << (shiftTable[i]);
        key_right = key_right << (shiftTable[i]);
        bits key_tmp = bits_merge(key_left, key_right);
        bits key48 = Pbox_f(PC_2Table, key_tmp, 48);

        bits tmp = left;
        left = right;
        right = circle_fun(right, key48) ^ tmp;
        //        cout << "M" << i+1 << ":" << endl << bits_merge(left, right) << endl;
        //        cout << "L" << i+1 << ": " << endl << left << endl << "R" << i+1 << ":" << endl << right << endl;
    }
    res = bits_merge(right, left);
    res = Pbox_f(inverseIPTable, res, 64);
    //    cout << "逆初始置换:" << endl << res << endl;
    return res;
}

bits decrypt(bits input_, bits key, int round_time){
    bits res;
    //    cout << "密文输入: " << endl << input_ << endl;
    bits input = Pbox_f(initIPTable, input_, 64); //初始置换
    //    cout << "初始置换: " << endl << input << endl;

    auto mid = input.begin();
    for(int i = 0; i < 32; i++, mid++);
    bits left = bits(input.begin(), mid);
    bits right = bits(mid, input.end());
    //    cout << "L0: " << endl << left << endl << "R0:" << endl << right << endl;

    bits key56 = Pbox_f(PC_1Table, key, 56);
    mid = key56.begin();
    for(int i = 0; i < 28; i++, mid++);
    bits key_left = bits(key56.begin(), mid);
    bits key_right = bits(mid, key56.end());
    key_left = key_left << 29;
    key_right = key_right << 29;

    bits key_tmp1 = bits_merge(key_left, key_right);

    for(int i = 0; i < round_time; i++){
```

```
        key_left = key_left >> (shiftTable[(16-i)%16]);
        key_right = key_right >> (shiftTable[(16-i)%16]);
        bits key_tmp = bits_merge(key_left, key_right);
        bits key48 = Pbox_f(PC_2Table, key_tmp, 48);

        bits tmp = left;
        left = right;
        right = circle_fun(right, key48) ^ tmp;
        //        cout << "M" << 16-i << ":" << endl<<
bits_merge(left, right) << endl;
        //        cout << "L" << i+1 << ": " << endl << left <<
endl << "R" << i+1 << ":" << endl << right << endl;
    }
    res = bits_merge(right, left);
    res = Pbox_f(inverseIPTable, res, 64); //初始置换逆置换
    //    cout << "逆初始置换:" << endl << res << endl;
    return res;
}

bits genKey(string key){
    string tmp = key + "        ";//补全不足64位的密钥
    string realKey = tmp.substr(0, 8);
    return char2byte(realKey, 64);
}

vector<bits> string2bytes(string Message){
    vector<bits> res;
    int i = 0;
    for(int size = Message.size(); i + 7 < size; i = i + 8){
        res.push_back(char2byte(Message.substr(i, i+8), 64));
    }
    if((Message.size() % 8) == 0)
        return res;
    else{
        string last = Message.substr(i);
        char tmp = 0;
        while(last.size() != 8)
            last += tmp;
        res.push_back(char2byte(last, 64));
    }
    return res;
}

string b2s(bits Cipher, int bits_count){
    int tmp[8] = {0};
    string res;
    for (int i = 0; i < bits_count; i++){
        int temp = Cipher[i];
        tmp[i / 8] |= temp << (i % 8);
```

```
251         //        cout << tmp[i/8] << endl;
252     }
253     for(int i = 0; i < 8; i++)
254         res.push_back(tmp[i]);
255     return res;
256 }
257 
258 string bytes2string(vector<bits> Cipher){
259     string res;
260     for(int i = 0,size = Cipher.size(); i < size; i++){
261         res += b2s(Cipher[i], 64);
262     }
263     return res;
264 }
265 
266 vector<bits> doEncrypt(vector<bits> Message, bits key){
267     vector<bits> res;
268     for(int i = 0, size = Message.size(); i < size; i++)
269         res.push_back(encrypt(Message[i], key, 16));
270     return res;
271 }
272 
273 vector<bits> doDecrypt(vector<bits> C, bits key){
274     vector<bits> res;
275     for(int i = 0, size = C.size(); i < size; i++)
276         res.push_back(decrypt(C[i], key, 16) );
277     return res;
278 }
279 
280 bits hex2byte(string input){
281     bits res;
282     cout << input << endl;
283     for(int i = 0, j = 3; i < 64; i++){
284         int tmp = input[i/4];
285         if(tmp <= '9' && tmp >= '0')
286             tmp -= '0';
287         else
288             tmp -= 'a' - 10;
289         res.push_back( (tmp >> j) & 1);
290         j--;
291         if(j == -1)
292             j = 3;
293     }
294     return res;
295 }
296 
297 vector<bits> hex2bytes(string input){
298     vector<bits> res;
299     for(int i = 0,size = input.size(); i < size; i = i+16){
```

```
        string tmp = input.substr(i,16);
        res.push_back(hex2byte(tmp));
    }
    return res;
}

string byte2hex(bits input){
    string res = "";
    for(int i = 0,size = input.size(); i < size; i=i+4){
        char tmp = 0;
        for(int j = 0; j < 4;j++){
            tmp |= int(input[i+j]) << (3-j);
        }
        if(tmp < 10)
            res += tmp +'0';
        else
            res += (tmp-10) + 'a';
    }
    return res;
}

string bytes2hex(vector<bits> input){
    string res;
    for(int i = 0,size = input.size(); i < size; i++){
        res += byte2hex(input[i]);
    }
    return res;
}
#endif /* DES_h */

```