

**第三次作业报告**

**课程名称： 物联网安全技术**

**授课教师： 姚宏**

**学生姓名： 郭鸿绅**

**学生学号： 20181002139**

**学生班号： 193182**

目录

[一、题目 1](#_Toc67995734)

[二、程序设计与实现 1](#_Toc67995735)

[三、程序流程图 2](#_Toc67995736)

[3.1 初始化 2](#_Toc67995737)

[3.2 加密和解密 2](#_Toc67995738)

[3.3 main 4](#_Toc67995739)

[四、实现结果 4](#_Toc67995740)

[五、调试过程 5](#_Toc67995741)

[六、使用手册 5](#_Toc67995742)

[七、源程序 5](#_Toc67995743)

[7.1 DES.hpp 5](#_Toc67995744)

[7.2 main.cpp 16](#_Toc67995745)

[八、推导题 17](#_Toc67995746)

# 一、案例需求

移动节点资料参考京东页面：https://item.jd.com/62429564776.html#crumb-wrap

希望能够利用移动节点，获取工作区中，主要区域（地信学院、运动场、体育馆、教工食堂、学生中心、图书馆、公教楼、北大门）等的温度、光照数据。并能够将数据传回位于计算机学院的服务器上。



阶段一任务要求：根据任务要求，选择合适的设备搭配，设计网络的逻辑架构。

# 二、网络部署

（一）网络需求分析

所提供的移动节点支持红外遥控或者WiFi控制。由于服务区域大致为350m×400m的矩形，而红外遥控距离大致在10m左右，所以采用WiFi控制方式。

（二）校园网质量测试



1、测试方式：通过手机状态栏显示的CUG信号强度和Speedtest测速软件对服务区域的个点进行网络测速（下图反映了Speedtest软件测试出的当前下载速度、上传速度、网络时延等情况）。



2、测试结果：

A区（操场）网络状态良好，下载速度在3M以上，并无信号差或者无信号的情况；

B区网络信号略弱。但网速比操场略佳；

C区网络状况与B区大致相同；

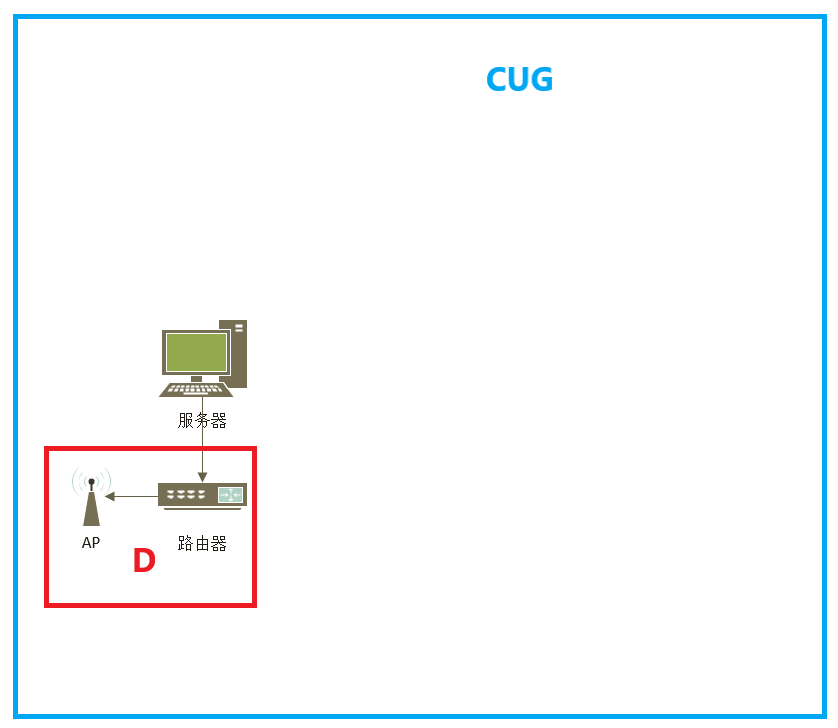
D区网络状况极差，尤其是D区中心处，手机会检测不到CUG信号。

其他区域：网络良好。

3、测试结果分析

除D去以外，其他区域完全可满足小车对网络质量的需求。故需要在D区域搭建内网，否则小车在此区域会失去信号导致服务器无法通过WiFi实时控制小车。D区域面积较小，仅需一个AP或者使用一个无线路由器即可实现D区的全面覆盖。

（三）网络部署图

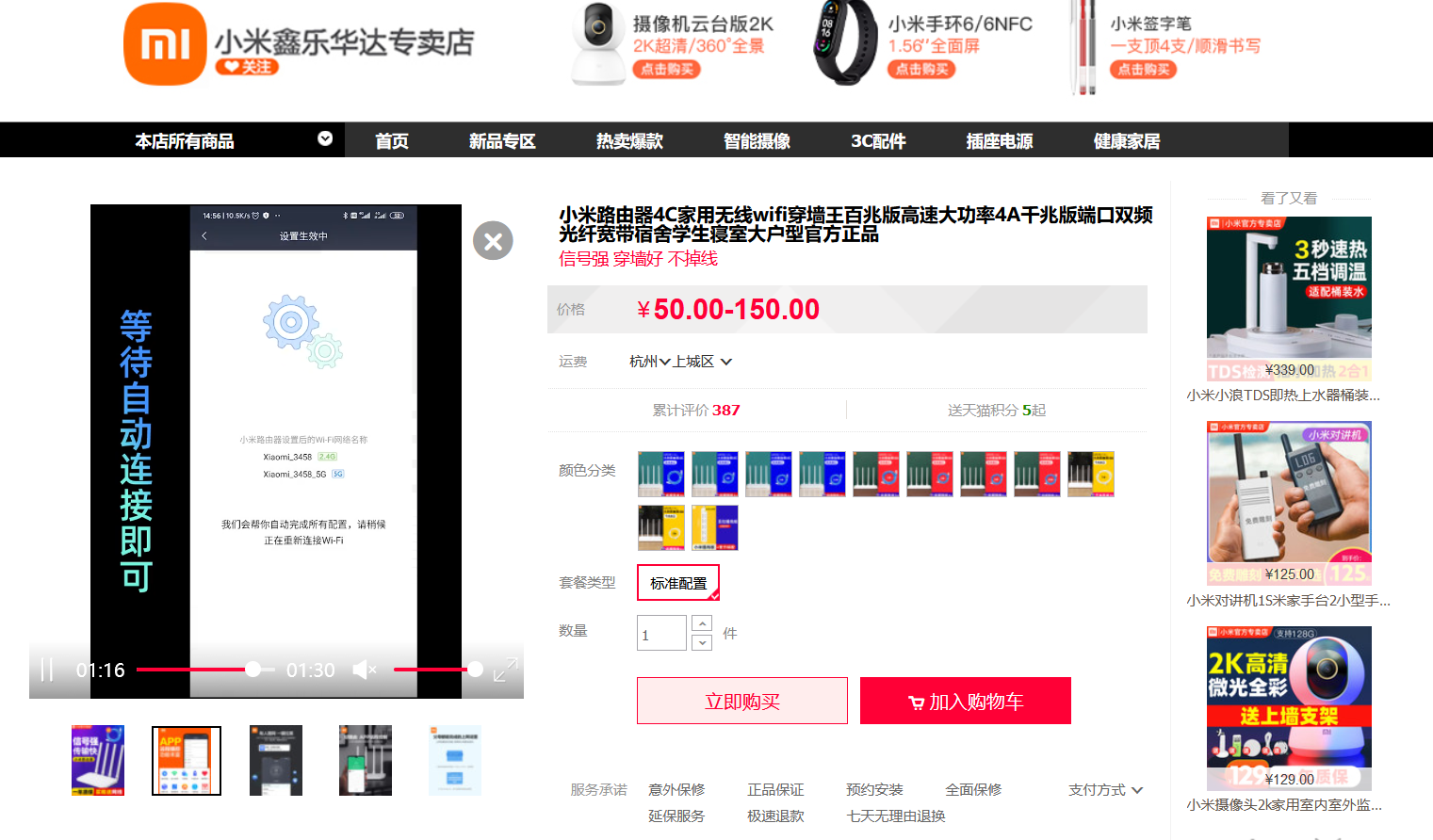


服务器设于CUG覆盖范围内（为客户端提供服务），D区的路由器通过有线连接服务器。服务器得到两个IP地址（CUG地址和内网地址）。

小车在D区域时，通过内网与服务器进行双向通信；位于CUG范围时通过CUG与服务器进行双向通信。

（四）设备选型

1、无线路由器



百兆传输速率已经可以满足小车环境数据的传输需求。

2、网线（连接服务器和无线路由器）



采用超五类线（百兆传输速率），长度50m即可满足要求。

# 三、业务规划

项目的业务分为两类：内部业务和外部业务。内部业务面向小车、外部业务面向CUG内的终端设备。

## （一）内部业务

1、小车控制

（二）数据存储

服务器将接收到的数据进行组织、过滤和合理存储，使用PostGIS或者SQL server，前者是目前流行。

（二）外部业务

无参构造函数采用默认的64位初始密钥，有参构造函数采用用户传入的64位初始密钥。8个S盒子的数据存放在文件里，所以构造函数会先加载S盒子的数据，再去调用生成轮密钥集合的函数（generateKeys()）。下图是生成轮密钥的简化流程图（图-1）。

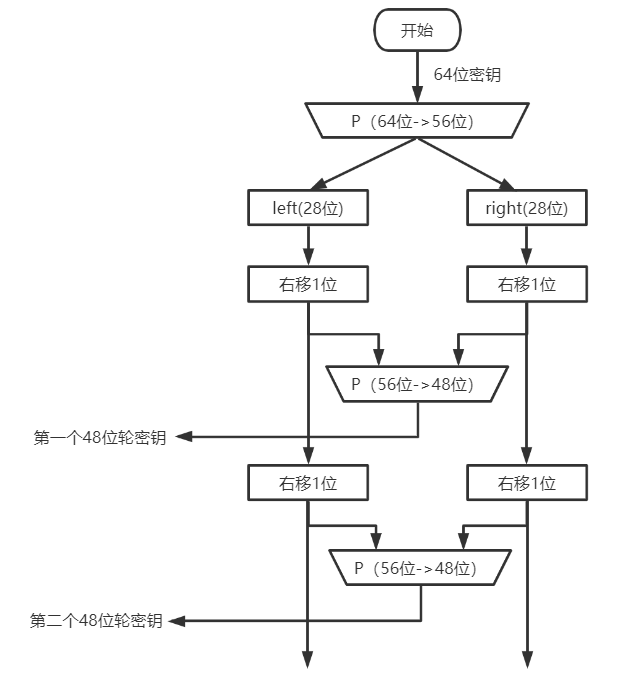


图-1

## 3.2 加密和解密

由于用户输入的是字符串（长度未必是8n字节），所以加密时，需要将明文拆分成多块，每一块8个字节，最后一块不足8个字节时往后补’\0’；解密时就不再需要拆分，因为密文一定是8n个字节。这个工作通过encrypt()和decrypt()来完成的。

encrypt64()和decrypt64()接受一个8字节string的参数，对其进行加密或解密，并将结果返回。encrypt64()的处理流程如下（图-2）：

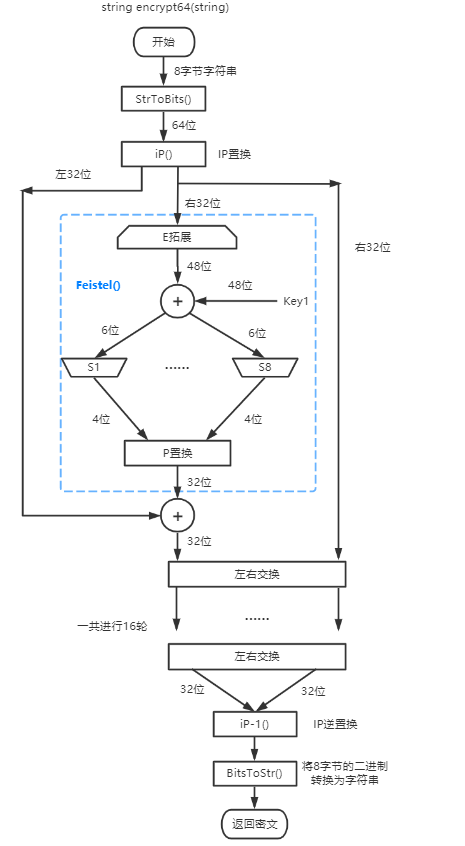


图-2

decrypt64()的流程同上，唯一区别是轮密钥的使用顺序相反。明文和密文的拆分通过encrypt()和decrypt()实现，这两个函数最后循环调用encrypt64()和decrypt64()完成文本的加密和解密，将最后的结果返回。其流程图如下（图-3）：

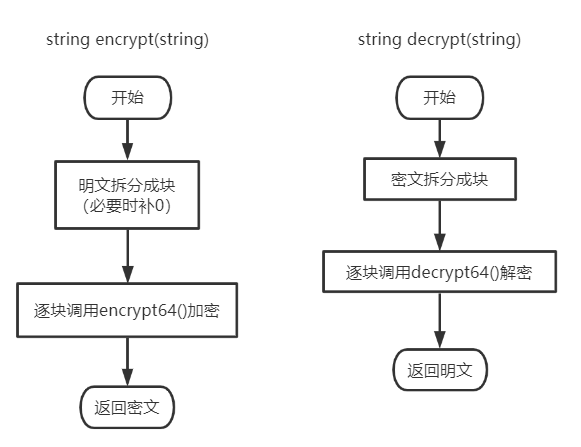


图-3

## 3.3 main

用户运行的是main.cpp程序，其逻辑流程如下图（图-4）。

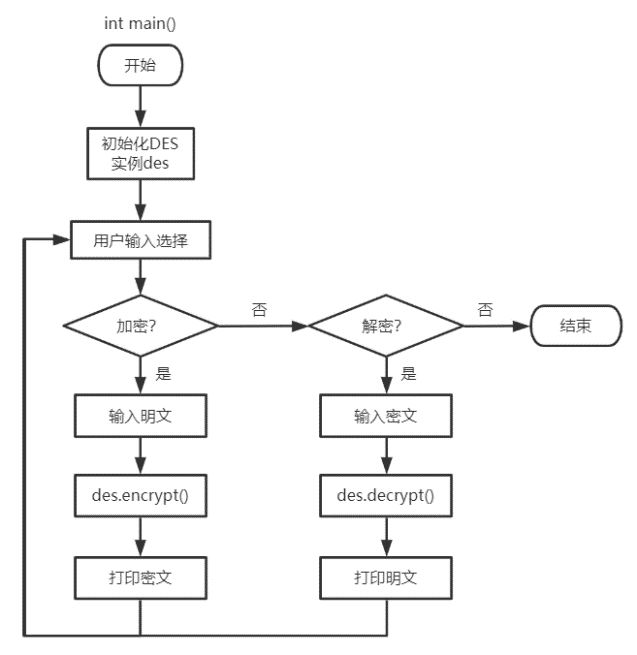


图-4

# 四、实现结果

程序使用Linux终端运行，加密和解密结果如下所示（图-5）：

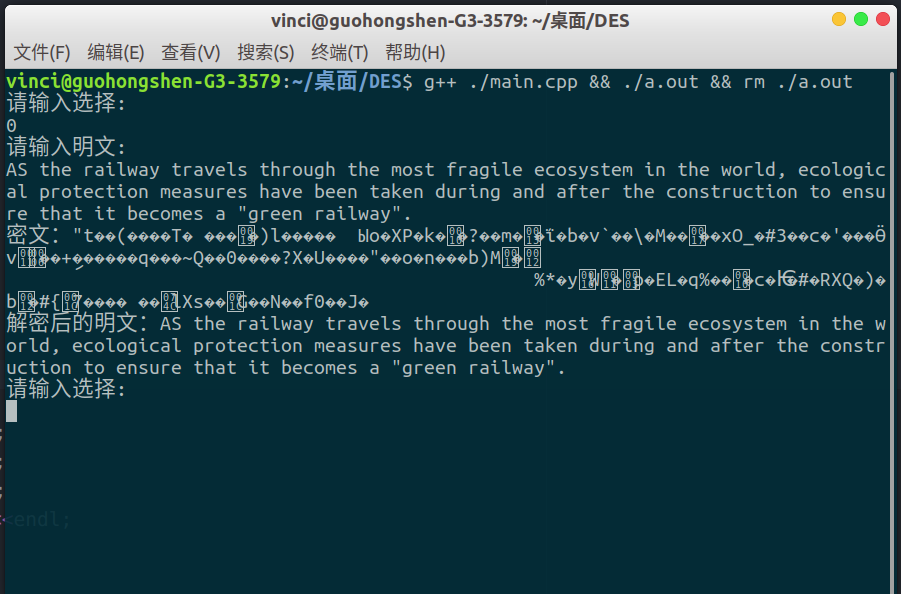


图-5

# 五、调试过程

倒序问题：bitset<n>类型支持将基本数据类型直接赋值给它，相反操作却不支持。将char类型赋值给bitset<8>之后，char中的高位对应于bitset<8>的高位，如char值为0000 1111的字符赋值给bitset<8>时，后者的第0位为1，而不是0。将bitset<n>输出时，程序会先输出高位，再输出低位。故string为abcd时，bitset<32>中为dcba。

bitset和string顺序相反的问题并不影响加密解密前后明文的一致性。只要程序中对待bitset的方式一致即可。

# 六、使用手册

编译并运行main.cpp文件，根据提示逐步操作即可。

# 七、源程序

## 7.1 DES.hpp

#ifndef DES\_HPP

#define DES\_HPP

#include <iostream>

#include <fstream>

#include <string>

#include <bitset>

#include <vector>

using namespace std;

class DES

{

private:

string chiperTxt;//密文

bitset<64> chiperBits;

string plainTxt;

bitset<64> plainBits;//明文

bitset<64> chiperKey;//密钥

int P64To56[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 };

int P56To48[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 };

int shift = 1;//循环左移1位

bitset<48> keys[17];//十六个轮密钥，keys[0]不用，keys[1~16]被使用。

int 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 };

int 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};//E/P拓展

bitset<4> Ss[8][4][16];//S盒子

int 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 };

int 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};

public:

DES(){//\*\*功能：构造函数。

bitset<64> temp("1010101010110101010010101010110101110101010101001001010100010101");

this->chiperKey = temp;

this->intialSs();//加载S-盒数据

this->generateKeys();//生成轮密钥

}

DES(string chiperKey\_x)//\*\*功能：构造函数，可选参数为x。

{

if(chiperKey\_x.length()!=64){

printf("The chiperkey's length must be 64. Try again please.");

}

else{

bitset<64> temp("1010101010110101010010101010110101110101010101001001010100010101");

this->chiperKey = temp;

this->intialSs();//加载S-盒数据

this->generateKeys();//生成轮密钥

}

}

~DES()//\*功能：析构函数。

{}

void intialSs()//?功能：从本地(SBox.txt)加载S-盒数据

{

string filename = "./SBox.txt";

ifstream inSsData;

inSsData.open(filename.c\_str(), ios::in);

int temp;

for(int i=0;i<8;++i){

for(int j=0;j<4;++j){

for(int k=0;k<16;++k){

if(!inSsData.eof()){

inSsData >> temp;

this->Ss[i][j][k] = temp;

}

}

}

}

inSsData.close();

}

void generateKeys()//\*\*功能：产生十六个轮密钥

{

bitset<28> left;

bitset<28> right;

int temp;

for (int i = 0; i < 28; ++i)

{

temp = this->P64To56[i] - 1;

left[i] = this->chiperKey[temp];

temp = this->P64To56[i + 28] - 1;

right[i] = this->chiperKey[temp];

}//P64To56置换

temp = 1;

while (temp < 17)

{

int index = 0;//索引

temp = (temp >> (28 - this->shift))|(temp << this->shift);//循环左移shift位

right = (right >> (28 - this->shift))|(right << this->shift);//循环左移shift位

for (int i = 0; i < 48; i++)

{

index = this->P56To48[i] - 1;

if (index < 28)

{

keys[temp][i] = left[index];

}

else

{

keys[temp][i] = right[index - 28];

}

}

++temp;

}

}

bitset<64> iP(bitset<64> x)//\*\*功能：IP置换（只接受bitset<64>类型）。

{

this->IP[26] = 48;

this->IP[27] = 40;

bitset<64> temp;

temp.reset();

for(int i=0;i<64;++i){

temp[i] = x[this->IP[i] - 1];

}

return temp;

}

bitset<64> iP\_1(bitset<64> x)//\*\*功能：对x进行IP逆置换。

{

bitset<64> temp;

for (int i = 0; i < 64; ++i)

{

temp[i] = x[this->IP\_1[i] - 1];

}

return temp;

}

bitset<32> Feistel(bitset<32>& R0, bitset<48> &key)//\*\*功能：轮函数。

{

//E拓展

bitset<48> expanded;

for (int i = 0; i < 48; i++)

{

expanded[i] = R0[this->E[i] - 1];

}

//拓展后的结果和轮密钥进行异或运算：

expanded ^= key;

//分成八组，每组6位：

bitset<6> ins[8];

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 6; j++)

{

ins[i][j] = expanded[j + i \* 6];

}

}

bitset<6> row;

bitset<6> col;

bitset<6> row\_and("100001");//提取行号

bitset<6> col\_and("011110");//提取列号

bitset<6> temp\_and("000001");

vector<bitset<4>> outs;//8个盒子的输出

int rowSelected;//选中的行

int colSelected;//选中的列

char \*\*endptr = NULL;

for (int i = 0; i < 8; i++)

{

row = ins[i] & row\_and;

row = (row >> 4) | (row & temp\_and);

col = ins[i] & col\_and;

col = col >> 1;

rowSelected = strtol(row.to\_string().c\_str(), endptr, 2);

colSelected = strtol(col.to\_string().c\_str(), endptr, 2);

//

bitset<4> temp = this->Ss[i][rowSelected][colSelected];

//

outs.push\_back(temp);

}

bitset<32> feistel;

int k = 0;

for (int i = 0; i < 8; ++i)

{

for (int j = 0; j < 4; j++)

{

feistel[k++] = outs[i][j];

}

}

outs.clear();

feistel = this->p(feistel);

return feistel;

}

bitset<32> p(bitset<32> &feistel)//\*\*功能：P置换。

{

bitset<32>temp;

for (int i = 0; i < 32; i++)

{

temp[i] = feistel[this->P[i] - 1];

}

return temp;

}

string encrypt64(string x)//\*\*功能：将x(64 bits)进行加密并存储到密文中（不必要），

//主要是将其打印在屏幕上。

{

while (x.length() < 8)//规范化

{

x += '0';

}

//IP置换：

bitset<64> temp = this->iP(this->StrToBits(x));

bitset<32> L0;

bitset<32> R0;

bitset<32> L1;

bitset<32> R1;

for (int i = 0; i < 32; i++)

{

L0[i] = temp[i];

R0[i] = temp[i + 32];

}

int rotor = 1;//轮次

bitset<32> feistel;

while (rotor <= 16)

{

L1 = R0;

feistel = this->Feistel(R0, this->keys[rotor]);

R1 = L0 ^ feistel;//异或

L0 = L1;

R0 = R1;

++rotor;

}

for (int i = 0; i < 32; ++i)

{

temp[i] = R1[i];//最后再次交换一下

temp[i + 32] = L1[i];

}

//IP逆置换：

this->chiperBits = this->iP\_1(temp);

string y = this->BitsToStr(this->chiperBits);

this->chiperTxt = y;

return y;

}

string decrypt64(string x)//\*\*功能：解密x(x是8个字节的string类型),将解出的明文返回。

{

//IP置换：

bitset<64> temp = this->iP(this->StrToBits(x));

bitset<32> R1;//REi LEi

bitset<32> L1;

bitset<32> R0;

bitset<32> L0;

for (int i = 0; i < 32; ++i)

{

R1[i] = temp[i];

L1[i] = temp[i + 32];

}

int rotor = 16;

while (rotor > 0)

{

R0 = L1;

L0 = R1 ^ this->Feistel(L1, this->keys[rotor]);

L1 = L0;

R1 = R0;

--rotor;

}

for (int i = 0; i < 32; i++)

{

temp[i] = L0[i];//最后再交换一下

temp[i + 32] = R0[i];

}

this->plainBits = this->iP\_1(temp);

string y = this->BitsToStr(this->plainBits);

this->plainTxt = y;

return y;

}

bitset<64> StrToBits(string x)//\*\*功能：将string x(默认8个字节)转换为Bits

{

bitset<64> bits;

bitset<8> temp;

for (int i=0; i<8; ++i){

temp = x[i];

for(int j=0;j<8;++j){

bits[i\*8+j] = temp[j];

}

}

return bits;

}

string BitsToStr(bitset<64> x)//\*\*功能：将Bits转换成string

{

string str(8, ' ');

unsigned long tmpa = x.to\_ulong();

unsigned long mask = 0xffL;

for(int i = 0; i < 8; ++i){

str[i] = (char)((tmpa >> (i \* 8)) & mask);

}

return str;

}

string encrypt(string plain){

string chiper = "";

int temp = plain.length()%8;

if(temp!=0){

for(int i=0;i<(8-temp);++i){

plain += '\0';

}

}else{}

temp = plain.length()/8;

chiper = "";

for(int i=0;i<temp;++i){

chiper += this->encrypt64(plain.substr(i\*8, 8));

}

return chiper;

}

string decrypt(string chiper){

string plain = "";

int temp = chiper.length()/8;

for(int i=0;i<temp;++i){

plain += this->decrypt64(chiper.substr(i\*8, 8));

}

return plain;

}

};

#endif

## 7.2 main.cpp

#include <iostream>

#include <string>

#include <iomanip>

#include "DES.hpp"

using namespace std;

int main()

{

char choose = '0';

string plain = "";

string chiper = "";

DES des;

while (true) {

cout<<"请输入选择:"<<endl;

cin>>choose;

getchar();

if(choose == '0'){

cout<<"请输入明文:"<<endl;

getline(cin, plain);

chiper = des.encrypt(plain);

cout<<"密文："<<chiper<<endl;

plain = des.decrypt(chiper);

cout<<"解密后的明文："<<plain<<endl;

}

else if(choose == '1'){

cout<<"请输入密文:"<<endl;

cin>>chiper;

cout<<"明文"<<des.decrypt(chiper)<<endl;

}

else{

break;

}

}

return 0;

}

# 八、推导题

问：若今天为星期五，那么3^(3^3000)天后为星期几？

解：

3^1 mod 6 = 3

3^2 mod 6 = 3

3^3 mod 6 = 3

利用数学归纳法：假设3^k mod 6 = 3成立，

3^(k+1) mod 6 = (((3^k) mod 6) \* (3 mod 6)) mod 6

= (3 \* 3) mod 6

= 3

则：3^n mod 6 = 3恒成立，也即：3^n = 3^(6b + 3),b为常数。

∴ 3^(3^3000) = 3^(6b + 3)

则3^(3^3000) mod 7 = 3^(6b + 3) mod 7 = ((3^(6b) mod 7)\*(3^3 mod 7)) mod 7

= (((3^6 mod 7)^b) \* (3^3 mod 7)) mod 7

= (((3^φ(7) mod 7)^b) \* (3^3 mod 7)) mod 7

∵ (3, 7) = 1，也即3和7的最大公因数为1，故3^φ(7) mod 7 = 1

∴ 原式 = ((1^b) \* (3^3 mod 7)) mod 7

= 3^3 mod 7

= 6

∵ (6 + 5) mod 7 = 4

∴ 结果为星期四.