**DES**算法的加解密实现

苏文元（201722210013，13241244182）

**目录**

[**1 实验原理 3**](#_Toc500619641)

[**1.1 加密 3**](#_Toc500619642)

[**1.1.1 密钥生成 3**](#_Toc500619643)

[**1.1.2 明文和密钥部分 4**](#_Toc500619644)

[**1.1.3 逆初始置换 5**](#_Toc500619645)

[**1.2 解密 6**](#_Toc500619646)

[**2 算法实现 7**](#_Toc500619647)

[**2.1 程序流程图 7**](#_Toc500619648)

[**2.2 模块/函数功能说明 7**](#_Toc500619649)

[**2.2.1 将字符转成二进制 7**](#_Toc500619650)

[**2.2.2 二进制转成字符 8**](#_Toc500619651)

[**2.2.3 批置换 8**](#_Toc500619652)

[**2.2.4 得到16轮所需的密钥 8**](#_Toc500619653)

[**2.2.5 将加密后的密文转换为可读的明文 9**](#_Toc500619654)

[**2.2.6 将密文转换为真正的密文 9**](#_Toc500619655)

[**2.2.7 通用加/解密函数 9**](#_Toc500619656)

[**2.2.8 加密文本 10**](#_Toc500619657)

[**2.2.9 解密文本 11**](#_Toc500619658)

[**2.2.10 打开文件连接 12**](#_Toc500619659)

[**2.2.11 文件加密 12**](#_Toc500619660)

[**2.2.12 文件解密 13**](#_Toc500619661)

[**3 运行结果 14**](#_Toc500619662)

[**3.1 结果说明 14**](#_Toc500619663)

[**3.2 结果展示图 14**](#_Toc500619664)

[**3.2.1 加密字符串 14**](#_Toc500619665)

[**3.2.2 解密字符串 15**](#_Toc500619666)

[**3.2.3 加密文件 15**](#_Toc500619667)

[**3.2.4 解密文件 16**](#_Toc500619668)

[**3.3 实验结论 17**](#_Toc500619669)

[**4 总结 17**](#_Toc500619670)

[**4.1 完成工作 17**](#_Toc500619671)

[**4.2 任务分工 17**](#_Toc500619672)

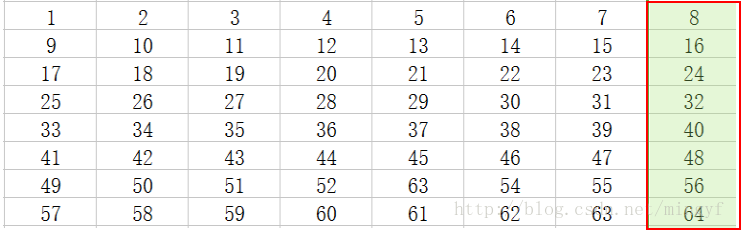
[**4.3 心得体会 17**](#_Toc500619673)

# 实验原理

## 加密

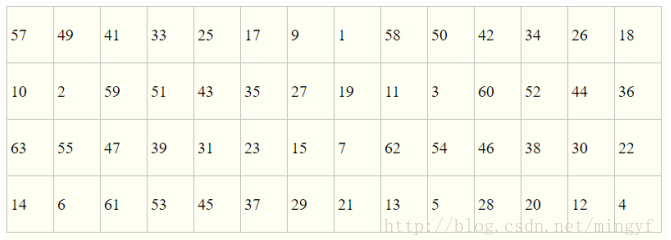
1. **密钥生成**
2. 密钥置换—64转56

将初始的64位密钥从1到64位进行编号，形成一个矩阵， 如下图所示，每一行的第八位将作为奇偶校验位被忽略



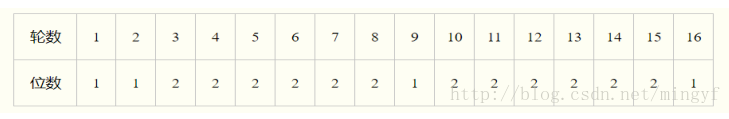
1. 密钥置换—56位数据互换位置

这个置换操作就是将第57位放到第1位，第49位放到第2位，将第4位放到最后一位，由于这里除去了奇偶校验位，所以不会包含6,16,24,32,40,48,56,64这八个数



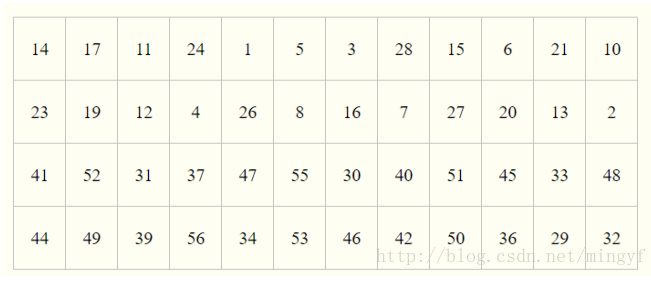
1. 密钥置换—循环移位

将刚才的56位输出分为两个部分，每个部分有28位，分别用L0和R0来表示，根据轮迭代的轮数，在第1,2,9,16轮时分别将L0和R0循环左移一位，其余轮数时循环左移两位，得到56位的输出

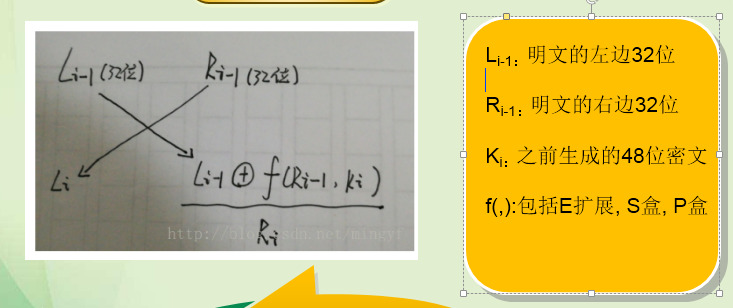


1. 密钥置换—密钥压缩置换56转48

得到的48位就是在后续的迭代里面输入的密钥Ki

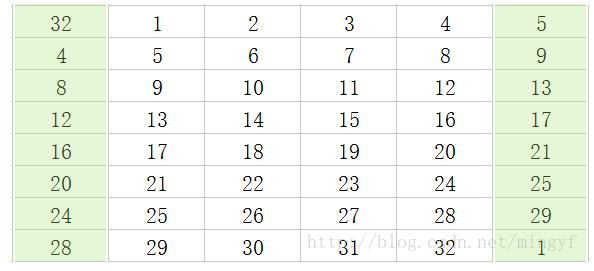


1. 16轮迭代



1. **明文和密钥部分**
2. **F函数之E扩展**

首先将原来的明文数据的右半部分R从32位扩展成为48位

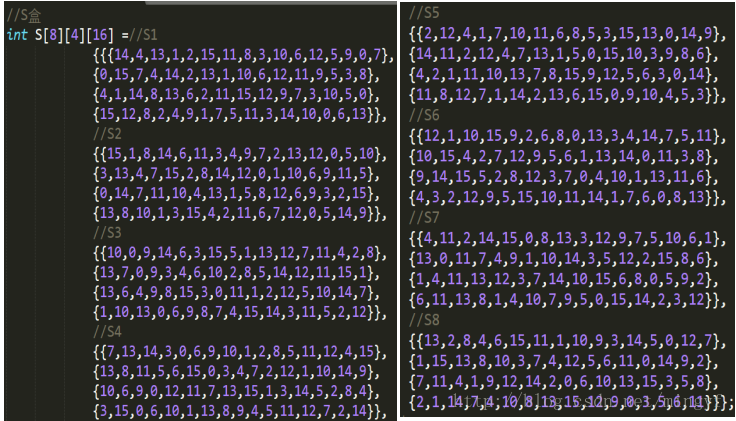


1. **F函数之异或**

将E扩展得到的48位的数据与密钥输入Ki进行异或运算

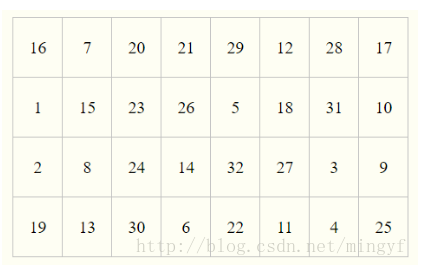
1. **F函数之S盒**

将48位数据按照每6位分为一组，一共分为8组，并分别输入S1， S2，S3，S4，S5，S6，S7，S8这8个盒子中，每个盒子产生4位的输出，将每个S 盒的输出拼接成32位。



1. **F函数之P盒**

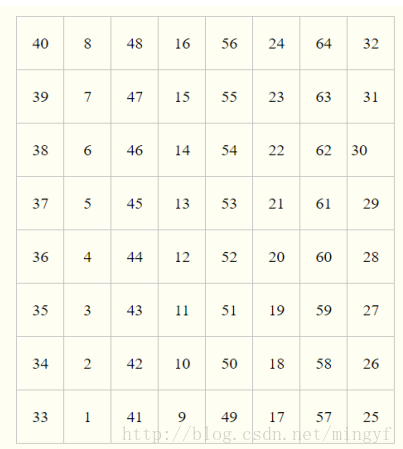
将S盒的输出拼接得到32位的数据，对这些数据再一次进行置换。



1. **F函数之异或**

将轮函数的输出结果与上一轮得到的输出结果Li-1进行异或运算，得到的就是这 一轮的输出结果Ri

1. **逆初始置换**



## 解密

密码顺序：如果加密的时候子密钥的使用顺序是K1，K2……K16，那么解密的时候子密钥 的使用顺序是K16，K15……K1；

输入输出：输入是64位密文，输出是64位明文

密钥：加密是向左循环, 解密是向右循环

# 算法实现

## 2.1 程序流程图



## 2.2 模块/函数功能说明

1. **将字符转成二进制**

|  |
| --- |
| void ChToBit(char\* dest, char\* src, int length) {  int i, j;  char t;  //一共有length个字符  for (i = 0; i < length; i++) {  //完成一个字符的转化  for (j = 8, t = src[i]; j > 0; j--) {  //左移三位为每个字符心的位置  dest[(i << 3) + j] = t & 1; // 取字符末位  //逐渐循环取得最后一位  t >>= 1;  }  }  } |

1. **二进制转成字符**

|  |
| --- |
| void BitToCh(char\* dest, char\* src, int length) {  int i;  for (i = 0; i < length << 3; i++) {  dest[i >> 3] <<= 1;  dest[i >> 3] |= src[i + 1]; // 添加到末位  }  dest[length] = 0;  } |

1. **批置换**

|  |
| --- |
| void BatchSet(char\* dest, char\* src, char\* offset, int count) {  int i;  for (i = 0; i < count; i++)  dest[i + 1] = src[offset[i]];  } |

1. **得到16轮所需的密钥**

|  |
| --- |
| void getKeys() {  char tk[128], bk[72];  char\* ptk = tk;  int i, j;  for (i = 0; i < 8; i++)  key[i] <<= 1; // 跳过奇偶校验位  ChToBit(bk, key, 8);  //将64位密码转化为56位  BatchSet(tk, bk, key\_ch, 56);  for (i = 0; i < 16; i++) {  for (j = 0; j < key\_mov[i]; j++, ptk++) {  ptk[57] = ptk[28];  ptk[28] = ptk[1];  // ptk++ 为亮点所在，实质上每一位都没有左移，只是指针右移了  }  //通过矩阵将56位密码转化为48位  BatchSet(keyb[i], ptk, key\_cmprs, 48);  }  } |

1. **将加密后的密文转换为可读的明文**

|  |
| --- |
| void msgPro(char\* dest, char\* src) {  int i, j;  for (i = 0; i < 16; i++) {  dest[i] = 0;  for (j = 1; j <= 4; j++) // 取4位按hash构造一个字符  dest[i] = (dest[i] << 1) | src[i \* 4 + j];  dest[i] = hs\_ch[dest[i]];  }  dest[i] = 0;  } |

1. **将密文转换为真正的密文**

|  |
| --- |
| void dropMsg(char\* dest, char\* src) {  int i;  for (i = 0; i < 16; i++) { // 为hash的逆运算过程  //初始dest[0]=0000 0000b sh\_ch表示十六进制对应的十进制数  dest[i >> 1] = (dest[i >> 1] << 4) | sh\_ch[src[i]];  }  } |

1. **通用加/解密函数**

|  |
| --- |
| void DES(char\* pmsg, int st, int cl, int step) {  int i, row, col;  char r[64], rt[48], s[8];  ChToBit(msgbt, pmsg, 8);  BatchSet(msgb, msgbt, msg\_ch, 64); // 初始置换  for (; st != cl; st += step) {  memcpy(rt, msgb + 33, 32);  // 扩展置换 ，将32位明文扩展为48位  BatchSet(r, msgb + 32, msg\_ex, 48);  //明文和密码的48位异或  for (i = 1; i <= 48; i++)  r[i] ^= keyb[st][i]; // 异或操作  // s\_box 代替，将48位数据按照每6位分为一组，一共分为8组，6位中的前两位为横坐标，后4位为纵坐标  for (i = 0; i < 48; i += 6) {  row = col = 0;  row = r[i + 1] << 1 | r[i + 6];  col = (r[i + 2] << 3) | (r[i + 3] << 2) | (r[i + 4] << 1) | r[i + 5];  s[i / 12] = (s[i / 12] <<= 4) | s\_box[i / 6][row][col];  }  ChToBit(r, s, 4);  // p\_box 置换，将S盒的输出拼接得到32位的数据，对这些数据再一次进行置换  BatchSet(msgb + 32, r, p\_box, 32);  //将明文的左边32位与p\_box盒置换的结果异或  for (i = 1; i <= 32; i++)  msgb[i + 32] ^= msgb[i]; // 异或  memcpy(msgb + 1, rt, 32);  }  memcpy(msgbt + 33, msgb + 1, 32);  memcpy(msgbt + 1, msgb + 33, 32);  BatchSet(msgb, msgbt, last\_ch, 64); // 末置换  if (step == 1)  msgPro(res, msgb); // 使密文可读  else{  if (strlen(msgb)/16!=0)  for (int i = strlen(msgb) / 16 + strlen(msgb) % 16; i < strlen(msgb); i++)  {  msgb[i] = '\0';  }  BitToCh(res, msgb, 8); // 转为原明文  }  } |

1. **加密文本**

|  |
| --- |
| void CDESDlg::OnTextEncode()  {  //加密  CString str;  int len;  char \*ptxtTemp;  e\_password.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  memcpy(key, ptxtTemp, len);  getKeys(); // 得到16轮要用到的密钥  e\_info.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  //len = str.GetLength();  memcpy(msg, ptxtTemp, len);  for (int i = 0; msg[i]; i += 8) {  DES(msg + i, ENCODE); // 加密  str = res;  e\_afterDes.SetWindowTextW(str);  }  MessageBox(L"加密完成");  } |

1. **解密文本**

|  |
| --- |
| void CDESDlg::OnTextDncode()  {  CString str;  int len;  char \*ptxtTemp;  e\_password.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  memcpy(key, ptxtTemp, len);  getKeys(); // 得到16轮要用到的密钥  e\_afterDes.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  memcpy(msg, ptxtTemp, len);  for (int i = 0; msg[i]; i += 16) {  dropMsg(res, msg + i); // 将密文转换为真正的密文  DES(res, DECODE); // 解密  str = res;  e\_info.SetWindowTextW(str);  }  MessageBox(L"解密完成");  } |

1. **打开文件连接**

|  |
| --- |
| void CDESDlg::OnOpenDir()  {  BOOL isOpen = TRUE; //是否打开(否则为保存)  CString defaultDir = L"E:\\FileTest"; //默认打开的文件路径  CString fileName = L""; //默认打开的文件名  CString filter = L"文件 (\*.doc; \*.ppt; \*.xls;\*.txt)|\*.doc;\*.ppt;\*.xls;\*.txt||"; //文件过虑的类型  CFileDialog openFileDlg(isOpen, defaultDir, fileName, OFN\_HIDEREADONLY | OFN\_READONLY, filter, NULL);  openFileDlg.GetOFN().lpstrInitialDir = L"E:\\FileTest\\test.doc";  INT\_PTR result = openFileDlg.DoModal();  CString filePath = defaultDir + "\\test.doc";  if (result == IDOK) {  filePath = openFileDlg.GetPathName();  }  m\_openDir = filePath;  UpdateData(FALSE);  //CWnd::SetDlgItemTextW(IDC\_EDIT\_SRC, filePath);  } |

1. **文件加密**

|  |
| --- |
| void CDESDlg::OnFileEncode()  {  UpdateData(TRUE);  CString str;  int len;  char \*ptxtTemp;  CFile openfile(m\_openDir, CFile::modeRead);//构造CFile对象  int length = openfile.GetLength();//获取文件长度  char \*strText;  strText = new char[length];  openfile.Read(strText, length);  openfile.Close();//关闭文件  e\_password.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  memcpy(key, ptxtTemp, len);  getKeys(); // 得到16轮要用到的密钥  CFile savefile(m\_saveDir, CFile::modeCreate | CFile::modeWrite);//构造CFile对象  for (int i = 0; strText[i]; i += 8) {  DES(strText + i, ENCODE); // 加密  savefile.Write(res, strlen(res));//写文件数据  }  savefile.Close();//关闭文件  delete[] strText;  MessageBox(L"加密完成");  } |

1. **文件解密**

|  |
| --- |
| void CDESDlg::OnFileDecode()  {  UpdateData(TRUE);  CString str;  int len;  char \*ptxtTemp;  e\_password.GetWindowTextW(str);  len = WideCharToMultiByte(CP\_ACP, 0, str, -1, NULL, 0, NULL, NULL);  ptxtTemp = new char[len + 1];  WideCharToMultiByte(CP\_ACP, 0, str, -1, ptxtTemp, len, NULL, NULL);  memcpy(key, ptxtTemp, len);  getKeys(); // 得到16轮要用到的密钥  CFile openfile(m\_saveDir, CFile::modeRead);//构造CFile对象  int length = openfile.GetLength();//获取文件长度  char \*strText;  strText = new char[length];  openfile.Read(strText, length);  openfile.Close();//关闭文件  CFile savefile(m\_openDir, CFile::modeCreate | CFile::modeWrite);//构造CFile对象  char datas[32];  for (int i = 0; strText[i]; i += 16) {  dropMsg(datas, strText + i); // 将密文转换为真正的密文  DES(datas, DECODE); // 解密  savefile.Write(res, strlen(res));  }  savefile.Close();//关闭文件  delete[] strText;  MessageBox(L"解密完成");  } |

# 3 运行结果

## 3.1 结果说明

可以成功把输入的字符串转化为密文，也可以成功将密文转化为明文；

可以成功把文件转化为密文，也可以把密文转化为明文，但是转化文件会有小bug，就是会在文件尾添加一段乱码。

## 3.2 结果展示图



上图中1：是清空左边的明文

上图中2：是加密时源文件的路径

上图中3：是密文文件的路劲

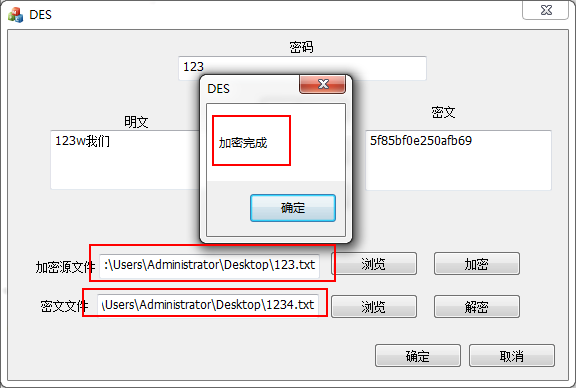
1. **加密字符串**

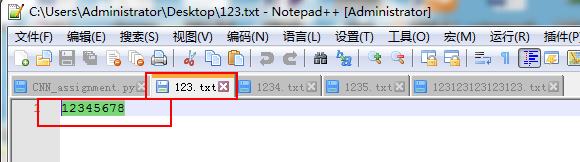


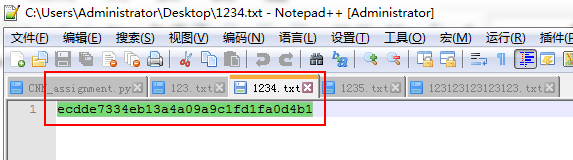
1. **解密字符串**



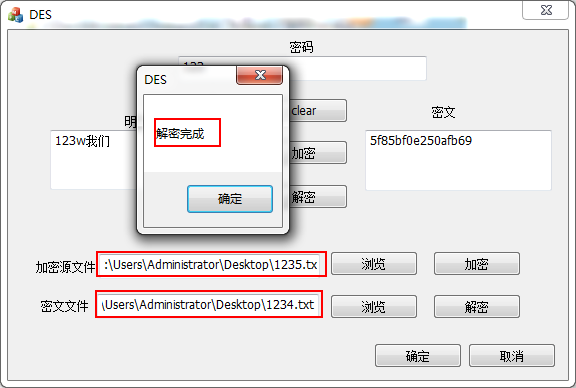
1. **加密文件**

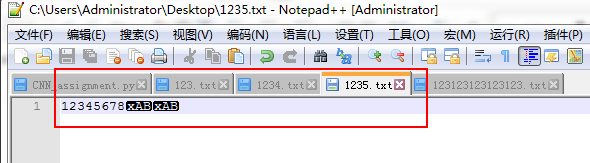






1. **解密文件**





故而文件解密相对于明文在结尾多了一些个乱码

## 3.3 实验结论

DES算法的安全性极高，到目前为止，除了用穷举搜索法对DES算法进行攻击外，还没有发现更有效的办法。通过穷举搜索空间，可获得总共256个可能的密钥。如果每一秒能检测一百万个的话，需要2000年才能完成检测。

# 4 总结

## 4.1 完成工作

总耗时大约是 25个小时左右，因为之前就完成了des的c设计，大约耗时20个小时，而后来有用图形化方式完成设计，大约耗时5个小时，在图形化过程中，因为MFC使用多字符编码，导致在调试的过程中遇到了些问题。

## 4.2 任务分工

自己独立完成。

## 4.3 心得体会

经过这一段大约20多个小时的学习，给我的感受是很深刻的，让我学习到了线性代数中矩阵的作用是多么大，而后又重新学习到了好多c语言的知识，各种字符编码间的区别，感谢老师详细而全面的授课，感谢师兄耐心的指导我们做实验。