# 通用文件加密器的设计与实现

葛晓滨 （合肥市统计局计算站）

**摘要:**通用文件加密器是适用加密保护 *DOS* 操作系统文件的工具软件，该软件采用了多种方式结合的加密手段，使被加密保护的文件具有较强的抗破译能力，本文对该软件的设计原理及实现方法作了详细的阐述。

## 一、引言

在计算机通讯、存储等多种应用场合，用户都希望将自己的文件以加密形式记录在载体上，以确保文件的安全。特别是明文形式的文挡资料、源程序等更需要采取加密保护手段，以防泄密或他人的非法剽窃。但目前 DOS 的操作系统并没有提供给用户以强有力的文件保护手段，使 DOS 用户的文件缺乏安全保障。

为此，笔者在对各种类型 DOS 文件的内部组成成分进行了系统的研究和分析的基础上，设计了一个适用于 DOS 文件的通用文件加密器，它可以处理任意类型的 DOS 文件，对其进行加密。通过加密器的加密处理，包括象源程序这样的 ASCⅡ码文本文件以及象 COM、EXE 文件这样的二进制可执行文件或象 OVL、BIN 这样的二进制非执行文件，加密器都可以加以保护。

## 二、原理与方法

### 1.通用性设计

撇开 DOS 文件的类别，分析各类 DOS 文件的组成成分，我们可以探知任何类型的 DOS 文件，无论其表现形式和内容如何，总是可以将其组成成分看作是二进制的数字文件。只不过组成象 EXE、COM 这类文件的二进制数内容是可以被 DOS 的命令解释系统所接收和执行的代码，而组成象文本文件这类的二制数内容是可显示，但不可执行的代码。

据此，我们将任意类型 DOS 文件的组成均作为二进制数码处理，这样，不仅易于设计加密器的加密算法程序，而且实现了加密器对任意类型 DOS 文件进行加密／解密的通用性。

### 2.密钥及加密算法

我们知道，密钥及加密算法的设计是加密器设计的关键因素。一般而言，密钥长度越长、个数越多，则保密性越好；加密算法越复杂，则加密的可靠性越强。

所以，我们采用了三重保险机制设计了密钥，同时采用了〝求反〞同〝异成〞相结合的加密算法进行加密，取得了较好的加密效果。

(1)密钥的设计。密钥的三重保险机制是指主密钥与副密钥相结合，同时辅以计算机产生的随机密钥共同作用的密钥体系，用户的加密文件必须同时具备这三把密钥才能够打开、还原和使用。

第一把密钥是由用户输入密码产生的，它是开启加密文件的主密钥，加密器规定，主密钥的长度在 3 至 8 个字符内；每二把密钥是由隐含在加密器内部的一个固定长度的信息串产生的，它的长度为 8 个字符，是开启加密文件的副密钥；第三把密钥是由计算机产生的随机密钥它的作用是破坏主密钥及副密钥可能具有的规律性，使密钥总体上具有不重复性和随机性。这样，即使用户对同一文件输入同一密码进行两次同样手续的加密操作，所产生的加密结果也不会相同。

这种设计增加了解密者对加密器加密原理的分析难度。加密器的三把密钥共同构成了对加密文件的保护，增加了加密文件的保险系数。

(2)加密算法的设计。加密算法的设计主要利用了〝求反〞运算和〝异或〞运算的右逆性构造而成。通过用户输入的主密钥及加密器中存储的副密钥选进行按位〝求反〞运算和按位〝异或〞运算，产生复合密钥将此复合密钥以及由计算机产生的随机密钥分别同待加密的源文件的二进制数内容进行循环〝异或〞和〝求反〞运算，产生的

运算结果即是加密的源代码。按这种方式对源文件中的每个字符逐个进行处理,并将处理结果写入加密文件中,即实现了对源文件的加密。

由于这种加密算法采用了按位"求反"和"异或"运算相结合的循环操作方法,打破了源文件同加密文件间所确定的一一对应关系,加之所使用的密钥在总体一具有的随机性,确保了加密算法的可靠性。

加密器的加密工作原理图如图 1 所示。

(3)密钥的保存。对于已加密的用户文件而言,密钥的保存至关重要,因为,即使加密手段再高明,在拥有密文件的文件头。经过加密处理的主密钥以及随机密钥都保存在这个文件头内,并混杂在其它的有关信息之中,这样增加了用户对密钥的搜寻难度,起到了较好的保护效果,而副密钥则保存在加密器中,用户无法从加密文件中发现和找到它。

这种密钥的分储与混储相结合的保护方法,安全系数较高。

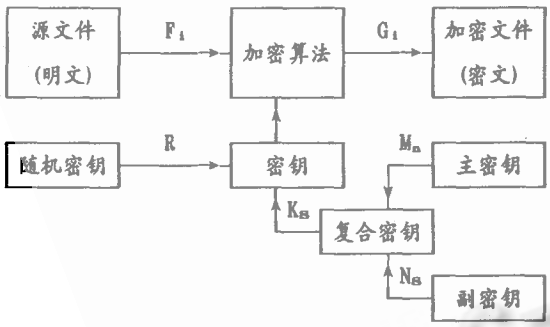


图 1　加密器的加密工作原理

## 三、实现步骤

加密器的加密操作可以描述为以下几个步骤:

(1)通过用户输入的密码构造密钥。设用户输入密码为序列:$M_N$(3<N<8),即主密钥 $M_N$,将 $M_N$ 序列与加密器中存储的副密钥匙 $N_S$(S=1～8)序列,按下标进行"异或"后"取反"的运算,由此可产生复合密钥 $K_S$(S=1～8),通过计算机的随机数发生器产生随机密钥 R。

(2)通过密钥与源文件字符进行运算产生加密字符。根据所构造的复合密钥 $K_S$ 及随机密钥 R,对待加密的源文件中的字符序列 $F_I$(I=1～源文件长度)采用算法:$G_I$= ～ ($F_I$^$K_J$^R)(其中运算符"～"代表"取反"运算,"^"代表"异或"运算,I=1～源文件长度,J=I MOD 8)从而获得加密的字符序列 $G_I$,即加密的源文件字符。

(3)构造文件头并产生加密文件。文件头中先说明加密文件标志:FFH,10H,然后说明的是加密文件的文头提示信息以及复合密钥及随机密钥等有关内容。文件头共占 50 字节的存储空间,

| 位移量 | 内容 |
|---|---|
| 00 | 加密文件第一标识符 FFH |
| 01 | 加密文件第二标识符 10H |
| 02-37 | 加密文件的文头提示信息 "This file has …" |
| 38 | 控制字符 07H (响铃) |
| 39 | 控制字符 1AH (结束符) |
| 40-47 | 复合密钥 $K_1$～$K_8$ |
| 48 | 随机密钥 R |
| 49 | 控制字符 1AH (结束符) |

图 2　加密文件文件头结构

将文件头写入加密文件首部,随后将加密的密文字符序列 $G_I$ 写入到加密文件中,直至文件结束,由此即可产生加密文件。

对文件的解密是上述加密操作的逆过程,篇幅所限,这里不再赘述。

## 四、实现程序

加密器的源程序采用 C 语言编写,并命名为 LOCKER.C,它实现了上述程序设计思想与方法,源程序 LOCKER.C 经 TURBO C 2.0 或 BORLAND C++1.0 软件的编译、链接后,即可产生可执行文件 LOCKER.EXE。

**1.运行格式**

LOCKER.EXE 运行格式如下:

LOCKER [-D] ,INPUT>,<OUTPUT>

其中:<INPUT>是用户输入的待加密的源文件名,<OUTPUT>是用户输入的由 LOCKER 加密产生的输出文件名,[-D]参数表示将<INPUT>文件解密为<OUTPUT>文件,无[-D] 参数,则是将<INPUT>加密为<OUTPUT>文件。

**2.软件操作**

进入该程序后,它可以自动判断用户的命令格式是否符合加密器的使用规则,然后对待加密的文件,可判定其是否曾加过密;对待解密的文件,可判定其结构和内容是否符合加密器的加密规格。通过这些检测后,加密器才要求用户输入密码,经两次输入核对无误后,加密器可以自动对文件进行加密/解密。在解密操作之前,加密器还需验证用户输入密码的正确性,只有合法用户才能对加密文件进行解密。

**3.软件设计技巧**

经加密器保护后的用户文件,其文件内容是杂乱无章的。若用户试图用 DOS 的 TYPE 等命令查看加密文件的内容,则屏幕上只显示信息:"This file has been locked by LOCKER!"然后响铃退出,加密文件的实际内容并不显示在屏幕上。

这是加密器设计上采用的一个实用技巧,它在构造加密文件的文件头时两次使用了文件结束标识符"1AH",这样,用户在查看文件内容时遇到这个文件结束控制符后便停止显示,这种制造文件"假结束"标志的方法,对保护加密文件的内容也起到一定的作用。

## 五、应用效果

经多次实践证明,通用文件加密器可以快速处理任意类型的 DOS 文件。由于加密器的加密方法所具有复杂度,使得被加密的源文件很难被破译,保障了 DOS 用户文件的安全。本文提供的加密器软件 LOCKER 具有操作简单、易于使用的特点,它可以作为 DOS 的一个实用外部命令程序随时为 DOS 用户提供文件加密保护服务。

程序 1:

```
/* 程 序 名: LOCKER.C   */
/* 编程语言: Turbo C 2.0 */
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "stdlib.h"
main(argc, argv)
 int argc;
 char *argv[];
 {
 FILE *in, *out;
 char *headmessage="This file has been locked by LOCKER!\007\x1a";
 char *exitmessage="Press any key to exit!";
 int lockhead[]={0xff, 0x10};
 int subkey[9]={0xab, 0xbc, 0xcd, 0xde, 0xa2, 0xa4, 0xa6, 0xa8};
 char mainkey[9]={0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20};
 char sfile[30], ofile[30], temp[9], filekey[9], key[9];
 int a, b, c, i, j, k, h, v, randkey;
 int right, inlock, locked, iden[2];
 clrscr();
 gotoxy(30, 2);
 printf("<<< LOCKER >>>");
 gotoxy(22, 3);
 printf("*Author: X.B. Ge*  *Version: 1.2*");
 if (argc<3) {
  gotoxy(22, 4);
  printf("Usage : LOCKER [-d] <input> <output>");
  exit(1);
 }
 if(strcmp(strlwr((char *)argv[1]), "-d")) {
  locked=0;
  a=1;
  }
   fseek(in, 40L, SEEK_SET);
   for(i=0; i<8; i++) {
    filekey[i]=fgetc(in);
    if (key[i]!=filekey[i])
     right=0;
   }
   if(!right) {
    gotoxy(26, 8);
    printf("Invalid password!");
    gotoxy(24, 9);
    printf("%s", exitmessage);
    getch();
    exit(1);
```

```
  else{
  locked=1;
  a=2;
 }
 b=a+1;
 inlock=0;
 in=fopen(argv[a], "rb");
 if(in==NULL) {
  gotoxy(28, 9);
  printf("Can't open the input file!");
  exit(1);
 }
 out=fopen(argv[b], "wb");
 if(in==NULL) {
  gotoxy(28, 9);
  printf("Can't open the output file!");
  exit(1);
 }
 rewind(in);
 right=inlock=1;
 for(h=0; h<2; h++) {
   iden[h]=fgetc(in);
   if(iden[h]!=lockhead[h])
    inlock=0;
 }
 if(!locked&&inlock) {
  gotoxy(20, 9);
  printf("Input file have been locked!");
  exit(1);
 }
 if(locked&&!inlock) {
  gotoxy(17, 9);
  printf("Input file hasn't locked file by LOCKER!");
  exit(1);
 }
 ENTER:
 for(j=5; j<=6; j++) {
  gotoxy(1, j);
  clreol();
 }
 gotoxy(18, 5);
 printf("Enter the password please :");
 gets(mainkey);
 if(strlen(mainkey)<3) {
  gotoxy(20, 9);
  printf("It must more than 3 characters!");
  goto ENTER;
 }
 gotoxy(18, 6);
 printf("Reenter the password please :");
 gets(temp);
 if(strcmp(mainkey, temp)) {
  gotoxy(20, 9);
  printf("Passwords are not identical!");
  goto ENTER;
 }
 gotoxy(1, 9);
 clreol();
 for(h=0; h<8; h++) {
  key[h]=mainkey[h]^subkey[h];
  key[h]=~key[h];
 }
 if(locked) {
   }
  }
  rewind(in);
  if (!locked) {
   randomize();
   randkey=rand();
   fputc(lockhead[0], out);
   fputc(lockhead[1], out);
   fprintf(out, headmessage);
   for(j=0; j<8; j++)
    fputc(key[j], out);
   fputc(randkey, out);
   fputc(0x1a, out);
   v=0;
   while(!feof(in)) {
    c=fgetc(in);
    c=c^randkey;
    c=c^key[v];
    c=~c;
    if(!feof(in))
     fputc(c, out);
    if(++v>7) v=0;
   }
  }
  if (locked) {
   fseek(in, 48L, SEEK_SET);
   randkey=fgetc(in);
   for(i=0; i<8; i++)
    key[i]=key[i]^randkey;
   fseek(in, 50L, SEEK_SET);
   v=0;
   while(!feof(in)) {
    c=fgetc(in);
    c=~c;
    c=c^key[v];
    if(!feof(in))
     fputc(c, out);
    if(++v>7) v=0;
   }
  }
  fclose(in);
  fclose(out);
  gotoxy(20, 8);
  printf("%s has been locked/unlocked to %s!", argv[a], argv[b]);
  gotoxy(24, 9);
  printf("%s", exitmessage);
  getch();
}
```