Encryption prescription

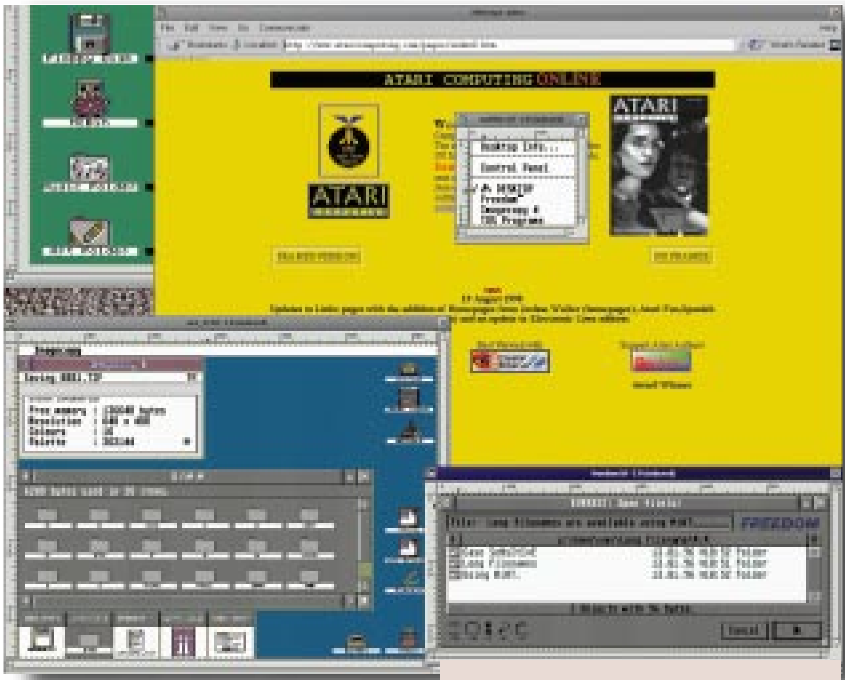
Chris Bidmead is not one for keeping secrets, but here he explains cryptographic file systems and shows you how to construct one, with hidden advantages.

A story on the BBC news web site at http://news.bbc.co.uk tells of IT journalist Kenneth Neil Cukier being stopped by UK Customs on his way back from Paris on the EuroStar. The official wanted to scan the hard drive of his laptop. Apparently, these checks for pornography at the borders are becoming routine but in this particular instance the Customs official was foiled as Kenneth's laptop wasn't a standard PC. "Our scanner doesn't work on Apples," said the official. I wonder what the scanner would have made of an ordinary laptop running, say, Linux?

As a writer who makes a living sharing his thoughts, I can't say I'm an avid supporter of privacy. After all, the whole free software movement is about the huge advantages of *not* keeping secrets. But Kenneth's brush with the State set me thinking about cryptography. The only encryption I've done before manually sets a password on individual files. A much better way is to run an encrypted file system; effectively a directory and its subdirectories where files are automatically encrypted on saving and decrypted when you open them. The encryption process is invisible and doesn't get in the way of your ordinary use of the disk.

So how does this keep your secrets? File systems need to be mounted before you can access them. Many operating systems do this automatically, and the fact that Unix uses an explicit mount command to connect its file systems to the operating system can be one of the first big points of bafflement for the beginner. Unix can be set up to mount particular file systems automatically at boot time or even dynamically on demand. Regular users of Unix see this flexibility as one of its big plusses.

A cryptographic file system (CFS) is designed to be manually mountable against a password, as in the CFS package developed by AT&T Labs employee, Matt Blaze. Linux users can snag this pre-compiled, either as a Debian package (at www.debian.org and search for CFS) or as an RPM (follow the links from www.relay.com).

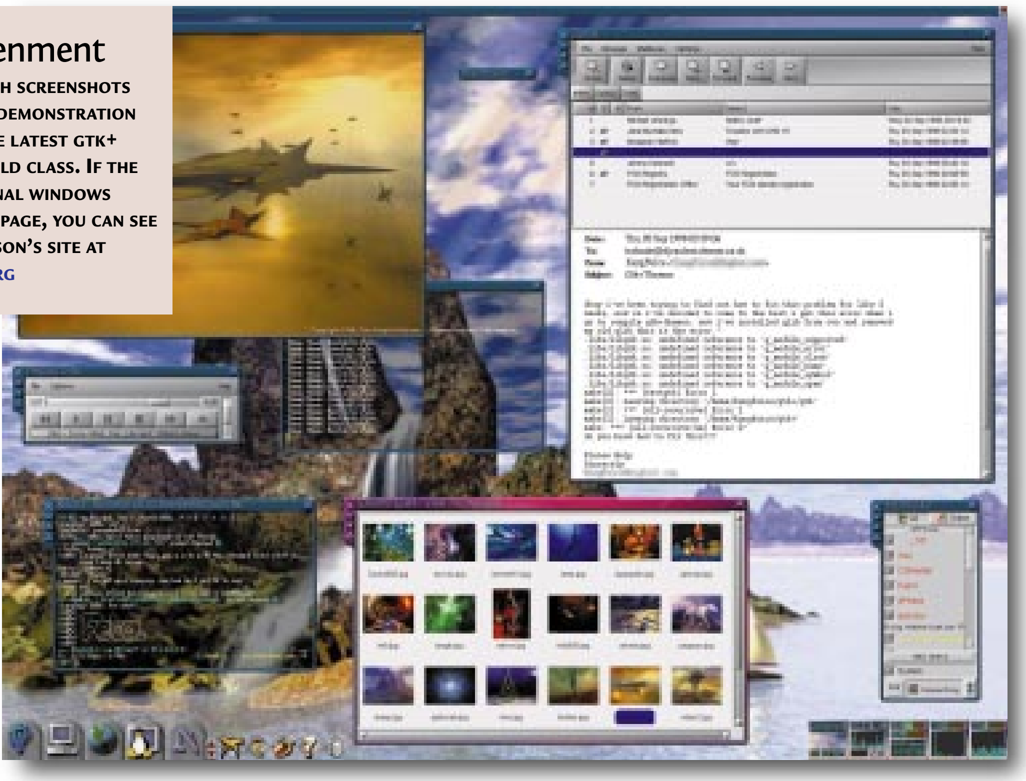


▲Unix on Atari

MARK CRUTCH XAV@COMPSOC.MAN.AC.UK LAST MONTH REMINDED THIS COLUMN THAT YOU CAN RUN PRETTY DECENT UNIX-LIKE OPERATING SYSTEMS ON ATARI STS AND COMMODORE AMIGAS. HERE ARE SOME SCREENSHOTS OF HIS OLD ATARI DESKTOP [BOTTOM LEFT], THE FREEDOM FILE SELECTOR [BOTTOM RIGHT], THE MULTITOS TASK-SWITCHER [CENTRE] AND SOME TYPICAL ATARI ICONS [TOP LEFT]. THE WEB PAGE IN THE BACKGROUND IS WWW.ATARICOMPUTING.COM, WHICH MARK CALLS "THE LAST BASTION OF SERIOUS ATARI NEWS"

LEARN ABOUT DES

DES stands for Data Encryption Standard and you can learn more about it at www.distributed.net/des/. The main page is about the DES Challenge: the use of idle time on multiple machines around the world to crack DES encryption. Links from this page take you to descriptions of what DES is, how it works, and why the US Government seems to think it's part of the armaments industry and therefore should be subject to draconian (but as it turns out practically meaningless) export restrictions. For a more detailed exposé of just how ludicrous this is, see Phil Karn's story at http://people.qualcomm.com/karn/export/index.html about how the US Department of Defense allows the export of a book on cryptography, but has ruled that a floppy disk containing source code from the same book is a "controlled encryption Item that cannot be legally exported from the US".



Path to Enlightenment

I get sent some pretty rakish screenshots from time to time, but this demonstration of Enlightenment, using the latest GTK+ toolkit with themes, is world class. If the amazing transparent terminal windows don't show up well on this page, you can see the original at David Coulson's site at http://technoir.nethead.org

The package cfs-1.3.3 includes the CFS daemon (cfsd) and a utility for building cfs directories (cmkdir). There are also two other utilities , cattach and cdetach, for attaching and detaching the cfs directories; roughly equivalent to mounting and dismounting them. The idea is that the encrypted data ends up on the directory you made with cmkdir but is saved and retrieved through a sort of "virtual mount point".
You create the initial directory using a password — or rather, because this is DES encryption (*see panel, p272*), a "pass phrase" with a minimum of 16 characters. You need the same phrase again in order to make the virtual connection with cattach. Once you've broken the virtual connection with cdetach, all the data on the encrypted file system becomes inaccessible.

Because Blaze's design uses NFS to integrate the file system into the operating system, the whole package runs in user space and you don't need to recompile your kernel.

Hardened hackers may prefer the alternative approach of a kernel patch. You can get this from http://ftp.replay.com/security/linux/all/. The NFS approach has the additional advantage that the real encrypted file system can be running on a remote machine, while what I'm calling the "virtual mount point" is local. Of course, this raises the interesting question of how to protect the flow of data between the virtual mount point and the remote file system — a snooper on the network will see it in clear, because the encryption only takes place once the data goes through the encrypted file system and hits the hard disk. For this reason, Blaze's package includes ESM, the encrypted session layer. In the simple

MORE FUN WITH MV AND CP

In the October issue, reader Alex Holden corrected an earlier implication I had made that the cp utility cannot preserve timestamps on the files it copies. I apologised for publishing "a piece of gnarled old Unix wisdom that may have been superseded by subsequent evolution of cp".
Alex then went on to say: "mv won't move files between file systems because it doesn't actually move the files at all but basically just modifies the directory entry..."
Several of you have mailed me to point out that this is wrong. Now Alex and I both have egg on our faces, because what we have here is a piece of even more gnarled old Unix wisdom. Back in the very early days of Unix, I seem to remember, mv simply used the rename(2) call, which left the target filename anchored to the same inode (i.e. they had to be on the same file system). For some reason, this piece of obsolete information has stuck in my memory. Today's mv supplements rename(2) with the equivalent of cp(1) and rm(1) as necessary. Alex has taken the trouble to go into this at some length with me.
Let us now get this straight. At least in the case of the GNU file utilities: cp can preserve timestamps (-p), faithfully copy symbolic links (-d), and move whole directories (-R). And mv can move individual files (but not directories) across file systems.
Alex continues: "It would be nice if mv behaved consistently... Apparently the FreeBDS mv command has already been improved to remedy this...by transparently performing a recursive copy and delete if necessary."
Oliver Kiddle opk@thoth.u-net.com tells me that the Silicon Graphics Unix variant called IRIX can do this, too. I don't have access to IRIX, but I can confirm that BDS's mv can move an entire directory across a partition (or "slice", as FreeBDSers prefer to call it).

[FIG1]

Adding the mount to /etc/rc.d/rc.local

```
# added to the end o f/etc/rc.d/rc.local
if [ -x /usr/sbin/cfsd ]; then
  /usr/sbin/cfsd && \
     /bin/mount -o port=3049,intr localhost:/.cfsfs /crypt
```

[FIG2]

Box: wiz.c

```
#include<stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
  int i;
  char buffer[1024];
  if (setuid(0) == -1){
    printf("Service not ➔
available.\n");
    exit(1);
  }
  if (argc < 2) system ➔
("su -");
  else {
    buffer[0] = '/0' ;
    for (i=1;i<argc;i++) {
      strcat(buffer,argv[i]);
      strcat(buffer," ");
    }
    system(buffer);
  }
  exit(0);
}
```

(➔ *continues on next line*)

implementation on my Caldera OpenLinux system I kept the encrypted file system local and omitted ESM.

● **Here are the basics.** Log in as root and make sure your NFS daemon is already running. Then create a permanent mount point as an anchor for NFS (I called mine /.cfsfs). Close down all its permissions with chmod 0 ./cfsfs. Include this directory in the NFS export list, which is usually the file /etc/exports. I added the following line:

```
/.cfsfs  localhost  # used
 for CFS
```

The localhost qualification ensures that only my machine is going to get its hands on the file system. So, /.cfsfs operates "below the waterline" and the visible directory is the one that is going to be NFS-mounted on this. I can perform the mount manually (provided I've remembered to restart the NFS daemon to get it to reread /etc/exports) but if I want encryption as a regular feature of my system it makes sense to add the mount to /etc/rc.d/rc.local.
(In case you haven't met this yet, this is the init file that is the last to run when you boot, and is tailored for local use; a sort of AUTOEXEC.BAT.) [Fig 1]

Now everything is ready for cmkdir and cattach. At this point you can login again as an ordinary user. The logical encrypted directory we're about to create will appear as a subdirectory under /crypt but first we need to decide where the physical data is going to be stored. We need a partition with plenty of space (*Hint*: *run the df command at this stage to see how much space you have on your partitions*.) Assuming that /opt/system already exists, the directory creation command will look something like this:

```
cmkdir /opt/system/secret_
garbage
```

If you've installed CFS from the RedHat RPM package you can type man cmkdir to see some interesting command-line options offering various kinds and levels of encryption. The default is two-key hybrid mode triple DES. The instruction cmkdir will offer a prompt for a Key, and this is where you type in a phrase or saying (minimum 16 characters). It had better be something you remember because without it you won't be able to get your data back. A second prompt asks you to repeat the key as a precaution against typing errors. So, cmkdir now churns that phrase into the directory and eventually returns you to the commandline prompt. Now we run something like:

```
cattach -l
/opt/system/secret
_garbage safe
```

This creates a directory by the name of safe under /crypt that the system will see and use as an ordinary directory. I use the -l command line option because by default the encryption algorithm will also crunch the inode number and the creation time into the encryption. This makes the already devilishly difficult DES even harder to crack, but it does mean that if you back up /opt/system/secret_garbage and then restore it, the files it contains will be on new inodes and therefore indecipherable. The -l switch (lower security mode) leaves out the inode and creation time, which means you can save the encrypted data, ensuring that wherever you restore it you can recover it with the same key phrase.

When you've finished using your /crypt/safe directory, run cdetach and the vault door closes. And, cattach even has a time lock on the virtual directory, automatically disconnecting after a certain length of time or after a preset period of inactivity.

Wizzing around

Chris Seager cseager@compuserve.com has come up with a neat fix for the problem I mentioned (*PCW*, October) about not being able to run X as a user. Wiz is a general-purpose utility that allows a preconfigured list of users to run utilities as root without the need to su to root first. It's very like a feature called Open Sesame I use a great deal in NeXTStep, so I've made it a regular part of my Linux setup.

1 **Save the code** in Box: wiz.c [Fig 2] and compile it thus:

```
gcc -o wiz wiz.c
```

2 **Then, as root**, copy it to somewhere on your path like /usr/local/bin, and change the ownership and group of the executable to root and sys (chown root.sys wiz).

3 **Set wiz setuid root** and remove the general executable permission (chmod +s,o-x wiz).

4 **Add any users** you want to permit to use wiz to the group called sys by editing /etc/group (see man group). They'll probably have to log out and log in again to get the benefit.

One of the main uses I have for wiz is doing quick admin tasks when I'm running as a user. To edit, for example, the group list, instead of having to su to root and then run emacs, I can just do:

```
wiz emacs /etc/group
```

PCW CONTACTS

Chris Bidmead *can be contacted via the PCW editorial office (address, p10) or email* **unix@pcw.co.uk**