Shopping for an encrypted filesystem

SECRET CANDIDATES

If you’re not a security expert and you're looking for a crypto filesystem, you may be wondering about the alternatives. We took at a look at some popular crypto options for Linux.

BY PETER GUTMANN, CHRISTIAN NEY

Crypto filesystems keep your data safe – even if someone steals your computer. The recent popularity of portable PCs has brought increased awareness about the need for protection, but even users with desktop systems have reasons for wanting to ensure that their data stays secret.

Linux offers a number of encrypted filesystem options – each with a different approach to the encryption problem. We took a look at some of the alternatives, with the goal of providing some insights on encryption techniques, code quality, and the relative merits of the various solutions. We examined the following encrypted filesystems:

- Loop-AES
- DM-Crypt
- Truecrypt
- Crypto-FS
- Enc-FS

In addition to looking at technologies and techniques, we will also examined some performance parameters for these encryption options.

Because few of these systems document technical details – the laudable exceptions being Truecrypt and Enc-FS – auditing means wading through tons of source code (generated by reverse engineering). While researching this article, we couldn’t help feeling that source code analysis was more like an archaeological excavation than a code audit. Relevant functions were hidden away under multiple layers of code deposits, discarded software, and encryption experiments. But at least they shed some light on how the developers tried out various approaches to the problems of filesystem encryption.

Loop-AES

Loop-AES [1] is the oldest of the crypto filesystems we investigated for this article. It uses the venerable kernel *loop* module, and it will even run with the 2.0 kernel. The function of Loop-AES is similar to that of Cryptoloop, which has since proved to be insecure. Loop-AES works at the block device level and writes encrypted files to a container file or a partition designed specifically for that purpose.

Although Loop-AES performs some fairly low level system tricks, and the underlying technology is venerable to say the least, it is easy to use in practical applications. Crosscrypt [4] adds the ability to read Loop-AES containers on Windows.

A confusing number of options for building and using the Loop-AES filesystem prevents an effective security analysis. Encryption performance is radically different depending on your selection. The cluttered source code also prevents an audit, making it difficult to find out just what happens at the core of Loop-AES, and if the code really does what it claims to do.

Loop-AES does not use a salt (a series of random bits) to hash the password, which would prevent the case of the same password generating identical keys. Additionally, the Loop-AES algorithm uses a single hashing iteration (such as SHA-256 or SHA-512) to generate the encryption key. This approach leaves Loop-AES open to dictionary attacks – where a cracker uses dictionaries and rulesets to generate a list of possible passwords and calculate the hash in advance. The result is a collection of hashes that can be tried out, avoiding the need to test all possible combinations of numbers (which would be a brute force attack).

The readme for Loop-AES [3] maintains that Loop-AES is capable of using both salting and multiple iterations. The Loop-AES sample configuration files that Google turned up showed no trace of a salt and set the iteration value to 100 (which corresponds to 100,000 iterations). By default, Loop-AES does not use either salting or password hash iterations.

If a user explicitly selects iterations (and this only appears to be possible for AES-256), the software encrypts two 128-bit blocks with the initial key and exchanges the higher value 64 bits of the first 128-bit block with the lower value

Getting Started with Loop-AES

Installing Loop-AES from the source code package is the first obstacle. The package requires the matching kernel sources, including the components generated by the build – admins can look forward to building the kernel from scratch in most cases. You will also need to patch and recompile the fundamental *util-linux* [2] package, which contains important system commands, such as *mount*. At least the readme file [3] explains the process fairly well, providing useful tips at the same time, such as how to use Loop-AES in connection with software suspend. As this is very low level stuff, readers are well advised to stick to distribution-specific packages. Ubuntu users have a fairly easy time of things, as Listing 1 shows. The Aptitude command in Line 1 installs the required packages. The *module-assistant* in Lines 2 through 4 then generates an appropriate Loop-AES package from the source code. A minor bug occurs at this point, but one that is easily remedied: During the *build*, the system complains that *debian/rules* does not have the required privileges. The answer to this problem is to pop up a terminal window, change the permissions manually by giving the *chmod +x / usr/src/modules/loop-aes/debian/rules* command, and then repeat the build. Line 5 installs the resulting package.

Good system integration and a collection of customized tools make Loop-AES easy to use. Line 7 in Listing 1 shows a call to Losetup, which sets up a logical connection between the *loop0* loop device and the */dev/hda6* partition. You could just as easily specify a normal file as the block device here; the file would act as a container for the crypto block device. When prompted, enter a password with at least 20 characters; Losetup then writes the headers required by Loop-AES to the partition. An XFS filesystem is created in Line 8 to complete the loop device, which can then be mounted. After umounting, *losetup -d /dev/loop0* unmaps the logical connection to the physical device. To automate mounting, you can add the following to your */etc/ fstab*:

```
/dev/hda6 /home/chris/loopaes ↩
xfs defaults,loop=/dev/loop0,↩
encryption=AES256 0 0
```

advertisement

Listing 1: Loop-AES on Ubuntu

```
sudo aptitude install
   loop-aes-utils loop-aes-source
sudo module-assistant update
sudo module-assistant prepare
sudo module-assistant build
   loop-aes
sudo dpkg -i
   loop-aes-2.6.15-26-686_
   3.1b-8+2.6.15-26.45_i386.deb

losetup -e AES256 /dev/loop0 /
   dev/hda6
mkfs.xfs /dev/loop0
mount /dev/loop0 /home/chris/
   loopaes
```

64 bits of the second block. This algorithm is repeated for as long as you would like to repeat it in order to return the final 256-bit value.

As the name would suggest, Loop-AES uses the AES algorithm in CBC mode for encryption. The collection of options for handling the Initialization Vector (IV) is confusingly large, and includes options to set the sector number or an MD5 hash of this number as the IV.

Loop-AES has a major programming error in common with many crypto programs: the code entirely fails to check the return values of function calls. If an error occurs on calculating the key, the software just proceeds as normal without even noticing that the key is a bunch of zeros. This leaves the data practically unencrypted.

The code is so poor in part that the program would be more likely to crash with a null pointer dereference than actually with the use of an empty key. But relying on careless programming to protect oneself against more serious errors seems grossly negligent.

DM-Crypt

DM-Crypt [5] officially became part of the device mapper with kernel version 2.6.4, providing a transparent data encryption service. The system can use a separate partition or a container (by way of Losetup) for storage.

In contrast to Loop-AES, DM-Crypt is not restricted to a single algorithm: users can select any algorithm known to the kernel. In contrast to its insecure predecessor, Cryptoloop, the approach used by DM-Crypt, is useful for journaled filesystems such as Ext 3 or XFS. DM-Crypt even mounts Cryptoloop containers, making the transition from Cryptolook painless for users. In addition to the kernel module, the DM-Crypt system requires a number of user-space tools.

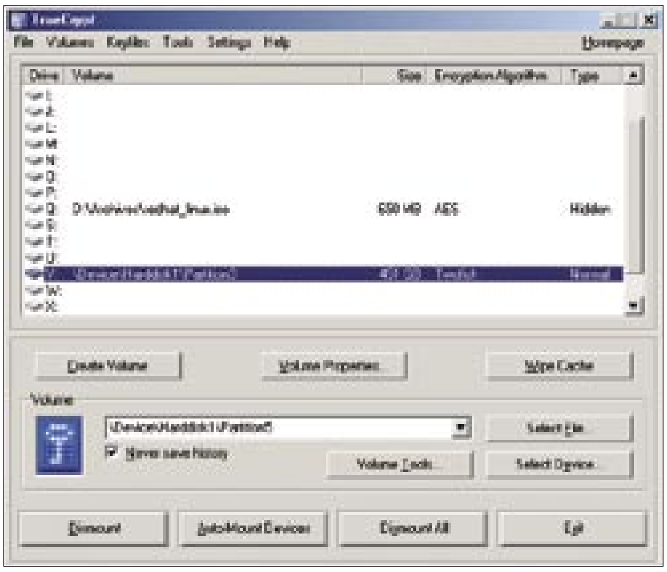

Figure 1: TrueCrypt provides a graphic user interface for managing file resources.

The *cryptsetup* program should have been included with every distribution by now. Unfortunately, Ubuntu, which is very user-friendly apart from this, steps out of line, hiding this critical package in the Universe repository.

Today, *cryptsetup* is typically seen in combination with Clemens Fruhwirth's LUKS (Linux Unified Key Setup, [6], [7]). Our tests of DM-Crypt included LUKS.

LUKS stores its metadata in the container header, managing multiple passwords, which the admin can revoke individually without needing to re-encrypt the data. This ensures secure access by multiple users. Windows can read DM-Crypt formats in connection with Free OTFE [8].

DM-Crypt's biggest advantage is its seamless integration with the system as a whole. Debian and Ubuntu show how admin-friendly this can be, using a configuration file to automate the whole process at system boot time. Unnoticed by the user, it sets up an encrypted swap in a very elegant way:

```
swap /dev/hda2 /dev/random swap
```

This entry in */etc/crypttab* applies to the swapspace on */dev/hda2*. At system

Getting Started with LUKS and DM-Crypt

A modified Cryptsetup package is required for LUKS – most modern distributions should have this by default. If your distribution doesn't, check out [9] for a collection of packages. A container is easily prepared:

```
cryptsetup luksFormat -y -c ⮧
aes-cbc-essiv:sha256 /dev/hda6
cryptsetup luksOpen ⮧
/dev/hda6 crypto mkfs.ext3 ⮧
/dev/mapper/crypto
```

The first command prepares */dev/hda6* as the device to be encrypted. It sets up the container header, which will store the key material among other things later. The *-y* option prompts twice for the passphrase. As an alternative, Cryptsetup can also use a key file.

The *-c aes-cbc-essiv:sha256* parameter tells DM-Crypt to write data in CBC (Cipher Block Chaining) mode and supply a SHA-256 hashed initialization vector. Without these parameters, the data would be susceptible to watermarking, a hack that involves the attacker preparing files and proving that the files reside within the container despite encryption. If you do not trust the default key length of 128 bits, you can double this by specifying *-s 256*.

luksOpen in the second command puts the physical device in the hands of the device mapper, and it can now be addressed as *crypto*. Just like for a RAID or LVM (Logical Volume Manager) device, the last line then creates a filesystem, Ext 3 in our case. The *cryptsetup luksRemove crypto* command then removes the device from the system.

launch time, DM-Crypt grabs a random key from */dev/random* and uses the key to encrypt the swap device. But enhanced security comes at a price: suspend to disk no longer works.

DM-Crypt has many characteristics in common with Loop-AES: both are kernel modules, and both confuse the user with too many build and runtime options. The default configuration is restricted to a single RIPEMD-160 hash iteration and does without a salt. The default IV mode is *plain*, which uses the 32-bit sector number as the IV.

Thankfully, there is a more secure mode known as ESSIV that encrypts the sector number (to prevent an attacker guessing it). However, our web search for configuration examples, and the example on the homepage, lead us to suspect that not many users actually go for ESSIV. Also, the security gains with ESSIV mode are minimal because the ESSIV is kept for all the data in this sector. From a cryptographic point of view, every sector should have a new IV for every change.

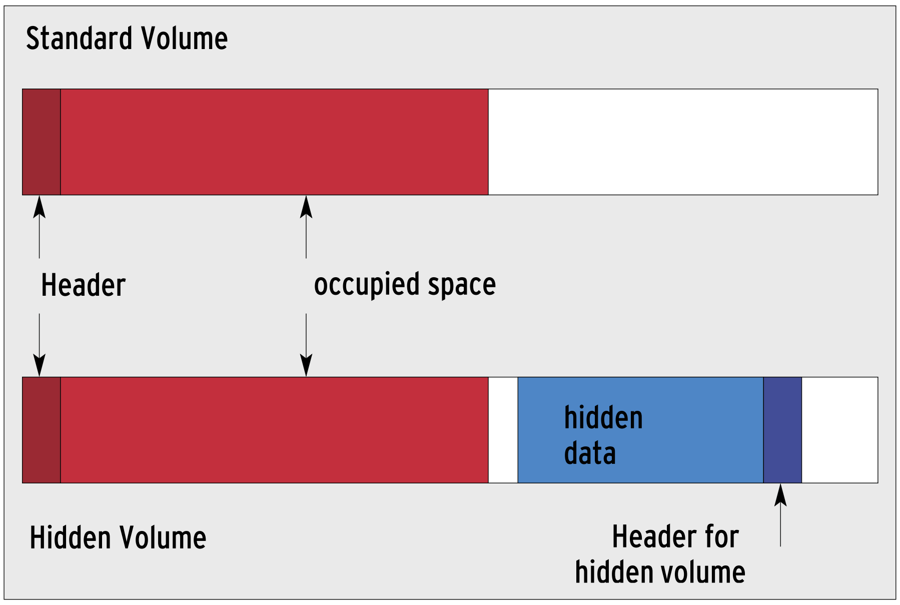

Figure 2: A Truecrypt container without (top) and with (bottom) a hidden volume.

The LUKS variant of the DM-Crypt system performs much better, although, for some reason, the DM-Crypt-LUKS configuration keeps the insecure DM-Crypt parameter by default. This said, Cryptsetup-LUKS uses the encrypted sector number as the default IV when creating a device in LUKS format.

The main task for the LUKS extension is key management, however, and there are some notable improvements in this area. LUKS uses the established PBKDF2

advertisement

Listing 2: Truecrypt on Ubuntu

```
01 sudo aptitude install
   build-essential linux-source
   gawk
02 cd /usr/src
03 sudo tar xvjf
   linux-source-2.6.15.tar.bz2
04 cd linux-source-2.6.15
05 sudo cp /boot/
   config-2.6.15-26-686 .config
06 sudo make prepare
07 cd /usr/src
08 tar xvzf
   truecrypt-4.2a-source-code.
   tar.gz
09 cd truecrypt-4.2a/Linux
10 sudo ./build.sh
11 sudo ./install.sh
```

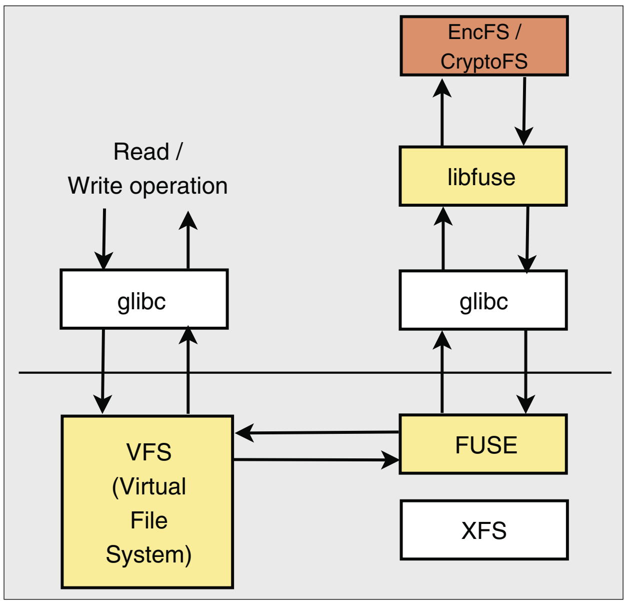

Figure 3: Encryption with a userspace filesystem: Glibc passes read and write operations through to the kernel, or to its VFS (Virtual filesystem) to be more precise. The FUSE module communicates with its userspace counterpart.

standard key generation method to derive a key from the password.

Truecrypt

Truecrypt [11] claims to be both secure and portable. The Truecrypt application uses encrypted containers on both Linux and Windows systems. Encryption algorithms include AES, Blowfish, Cast 5, Serpent, Triple-DES, and Twofish, and combinations of multiple variants are possible.

Three hash algorithms (RIPEMD-160, SHA-1, and Whirlpool) ensure integrity. Truecrypt (Figure 1) can use both files and partitions as containers. The former method is more popular with Truecrypt users.

The developers emphasize the user's ability to plausibly deny the existence of encrypted files. The way Truecrypt tries to do this is by using a prepared container that does not give rise to speculations on encryption. The software fills the container with random garbage data, and appends an inner container at the end of the shell – like Russian dolls (Figure 2).

With the release of Truecrypt version 4.2 (in April 2006), this open source program, which was originally developed for Windows, introduced support for creating encrypted files on Linux. The initial step was only possible on Windows prior to this.

The software can't deny its origins, and the port to Linux is still incomplete. For one thing, it does not have the user interface that Windows users may be familiar with. The documentation for Windows is very good and comprehensive;

Getting Started with Truecrypt

Depending on your system performance, the installation procedure can take anything from a couple of minutes to several hours. The build press launched in Line 10 of Listing 2 builds the whole kernel, without a good reason to do so, before going on to build the *truecrypt* module and accompanying tools. If you do not have the *gawk* package (and this is the case on Ubuntu, for example), the whole process terminates with an error message.

Admins can simply say *yes* to the first three prompts in the installation script (launched in Line 11 of Listing 2). If you also allow *non-admin* users to run Truecrypt by saying *yes* again, the installer sets the UID bit on the binary, opening up a potential attack vector. This said, the programming is pretty clean, and the risk is thus calculable.

You can run *truecrypt --create* to create a new Truecrypt-encrypted filesystem. If you do not supply all the required options at the command line, the tool will prompt you for them interactively. The example in Listing 3 creates a container file below */home/chris/truecrypt/test* (Lines 1 and 2) and formats the container with a FAT filesystem (Line 3). Truecrypt uses this smallest common denominator for the data exchange between Windows and Linux systems by default. If you prefer not to use FAT, you can set the filesystem for the container at a later stage.

If you opt for file-based encryption, the tool will additionally prompt you to specify the container size (100 MBytes, Line 4), before going on to offer a selection of hash and encryption algorithms. RIPEMD-160, and AES with a key length of 256 bits are the defaults.

In contrast to other crypto systems, Truecrypt uses LRW tweakable narrow-block encryption (see the "LRW" box). Although Truecrypt supports traditional, but less secure, Cipher Block Chaining (CBC), the developers sensibly advise against its use.

The tool then prompts you for a password; in addition to – or instead of – this, Truecrypt can optionally use key files. To use a file only, leave the password blank. Truecrypt collects mouse movements and keyboard input to generate entropy, instead of relying on */dev/random*. Based on the random value, Truecrypt overwrites the whole container with a pseudo-random sequence of numbers to prevent attempts to guess the files stored in the container.

The *truecrypt --dismount /mnt* command handles dismounting the filesystem. It is far more difficult to automatically mount Truecrypt containers at system boot time than with DM-Crypt, for example. You would need to write your own scripts to encrypt the */home* directory. Pamscript [15] could be useful here to run scripts during PAM (Pluggable Authentication Modules) authentication. Truecrypt suffers from a fundamental problem common to external kernel modules: if the external module does not work following a kernel update, users have no way to access their data.

Listing 3: Creating a Truecrypt-FS

```
Volume type: 1
Enter file or device name for
   new volume: /home/chris/
   truecrypt/test
Filesystem: FAT
Enter volume size (bytes -
   size/sizeK/sizeM/sizeG): 100M
Hash algorithm: 1
Encryption algorithm: 1
Enter password for new volume
   '/home/chris/truecrypt/test':
Re-enter password:
Enter keyfile path [none]:

truecrypt /home/chris/
   truecrypt/test /mnt
```

Filesystems in Userspace

The introduction of userspace-based filesystems, LUFS (Linux Userland Filesystem [16]), and the more recent FUSE (Filesystem in Userspace, [17]) led to a number of interesting spin-off projects that would never have made the kernel on their own merit. For example, Gmail-FS [18] and FTPFS [19] give users the ability to bind Google's mail service or an FTP server as if it were a local directory. It seems logical to integrate encryption functionality in a similar way (Figure 3). There are advantages this offers compared with kernel-based solutions:

- Filesystems that reside in userspace act as filters and make it easier to store data securely in places that are not under a user's control. For example, a root server operator could store encrypted data on a provider-side FTP server dedicated to backup processes.
- Userspace filesystems work at the file level. In contrast to their less flexible container-based counterparts, they use the existing filesystem and adapt to its size.
- As encrypted files and the accompanying metadata are stored directly on the filesystem, it is easy for backup tools to detect modified files and to process only these files.

At the same time, the visibility of the metadata is the biggest drawback to this approach. A user with access to the filesystem automatically knows the number of encrypted files, their permissions, and their approximate size (to 8 or 16 bytes). Depending on the system and configuration, even the cleartext filenames could be visible.

Linux users have to make do with a fairly terse manpage.

Users are more likely to be put off by the complicated usage, starting with the installation. The project homepage has a number of prebuilt packages for common Linux distributions, but most of them fail if you update your kernel. Even though the changelog claims that there is no need to rebuild within a kernel version, some manual steps are needed on Ubuntu 6.06, for example.

From a cryptographic point of view, Truecrypt is the most sophisticated and

advertisement

professional program of all our test candidates. It comes with useful documentation, and it is based on standards such as the PBKDF2 key generation function, as well as LRW mode (see the box titled “LRW”) for sector-based encryption. Truecrypt is the only program with safe defaults, and the software checks function return values, alerting the user in case of error.

The only thing that really bugged us about Truecrypt was the fact that the developers stick to the Rumpelstiltskin security model and disguise the container as a collection of digital garbage. They even take this Rumpelstiltskin approach so far as to have Truecrypt launch a brute force attack against the volume header in an attempt to access the data in the container.

Although the password is known, Truecrypt has to try out all hash and encryption algorithms until it finds usable data. This left us with an unnecessarily ambivalent impression: the program is convincing, and the programmers really know what they are doing. On the other hand, the developers are fairly bullheaded when it comes to controversial security properties.

Crypto-FS

Crypto-FS [20] is the simplest of the crypto filesystems we investigated. Crypto-FS is a userspace filesystem based on LUFS, but now it also runs on FUSE. (See the box titled “Filesystems in Userspace.”) As none of the major distributions has a software package for Crypto-FS at the time of this writing, some manual steps are required. A number of dependencies need to be fulfilled: developer packages for FUSE or LUFS, Libgcrypt [21] version 1.1.44 or newer, and Glib [22] version 2.6 or newer.

Following the standard *./configure && make && make install*, you might like to adapt the sample *cryptofs.conf* configuration file provided with the distribution to your own needs, and store it as *.cryptofs* in the directory where you will be storing the encrypted files later. The first time you do this, Crypto-FS will prompt you for a password. There are two ways to launch the tool: in typical FUSE fashion using a separate binary titled *cryptofs* or using the LUFS module with *lufsmount*:

```
cryptofs -r ↵
/home/chris/.cryptofs ↵
/home/chris/cryptofs
lufsmount cryptofs:↵
//home/chris/.cryptofs ↵
/home/chris/cryptofs
```

The absolute pathname is vital, as *mount* will point to a black hole otherwise. To give potential attackers as little information as possible about the encrypted files, Crypto-FS runs a Base-64 cipher against the filenames before storing them. However, the file size is still visible.

Crypto-FS encrypts individual files with a user-selectable cipher, typically AES in CBC mode. The program has its own special approach to generating the initialization vector. After converting the user password to an encryption key, it ciphers a buffer with null bytes to derive *n* IV values. For every *n*th file block,

Listing 4: Enc-FS

```
Directory "/home/chris/.
encfs" does not exist, create
(y,n)?
Directory "/home/chris/encfs"
does not exist, create (y,n)?
Creating new encrypted volume.
Please choose from one of the
following options:
 enter "x" for expert
configuration mode,
 enter "p" for pre-configured
paranoia mode,
 anything else, or an empty
line will select standard
mode.
```

LRW

LRW encryption mode ([12], named after its inventors Liskov, Rivest, and Wagner) is a tweakable block cipher, or to be more precise, a tweakable narrow block cipher. It resolves some of the issues that accompany CBC mode, without the overhead that comes with tweakable wide block ciphers, and also without patent issues. LRW does not produce much more overhead than CBC, and the algorithm is suitable for parallel processing on hardware.

In addition to an encryption key, LRW expects a tweak value (other crypto operations refer to comparable values as salts). Just like a salt, the tweak does not need to be secret. It simply ensures that the ciphertext is different for each tweak value, even if the cleartext is identical. It thus converts a single block cipher into a whole family of independent block ciphers. For hard disk encryption, it makes sense to use the sector number and position of the AES block within the sector as the tweak value. This makes it unique for each AES block.

While CBC allows encrypted blocks to be moved to a different position without comprising the ability to decipher, copy&paste attacks on LRW are doomed to failure. Encrypting with one tweak and decrypting with another just gives you garbage.

The tweak is usually calculated as the sector number times 32, plus the index within the sector. The factor of 32 is derived by dividing 512 (the number of bytes per sector) by 16 (bytes per AES block). Put simply, the tweak is the AES block number, counting from the start of the encrypted device [13]. Those of you who are familiar with AES from other contexts will be aware of a practical problem with draft standard LRW mode: years ago, the 16 AES candidates had six variant orders for the 128 bits of input and output. LRW rejoins the chaos. AES-GCM encryption mode (Galois/Counter Mode [14], used in the 802.11 WLAN standard) and LRW interpret the bit and byte orders in the block differently (Big-Endian and Little-Endian).

The computationally expensive result of this organizational mishap: GCM interprets the data as Little-Endian, whereas LRW assumes the data to be Big-Endian, and thus has to invert the order of the 128 bits in the block. As GCM is fairly widespread, in WLAN hardware for example, these two modes are mutually exclusive to a greater extent. It remains to be seen which mode will win at the end of the day. Of the test candidates for this article, only Truecrypt uses LRW mode.

While working on Cryptsetup-LUKS for DM-Crypt, Clemens Fruhwirth wrote a LRW patch [7], and attempted to have it accepted into the kernel early in 2005. Unfortunately, this didn’t work out, due to technical conflicts with memory management.

Crypto-FS uses the IV with the number n. As the number of file blocks will easily exceed n in practical applications, the filesystems uses each IV multiple times – a capital error in cryptography.

The developers are happy to use a single call to a hash function, typically SHA-1, to transform the password into a key, and they even do it without a salt. This makes the program susceptible to dictionary attacks. To top this, the program does not check function return values. If a function that processes keys or encrypts data happens to fail, Crypto-FS just goes on working with a blank key, or it simply passes the cleartext data.

Enc-FS

Enc-FS [23] is another userspace filesystem based on the modern FUSE (Filesystem in Userspace, [17]), which became a part of the default kernel with the Linux 2.6.14 release. Current Enc-FS versions need at least FUSE 2.5 and Rlog [24]. Enc-FS relies on OpenSSL for encryption purposes.

Basic use is really simple. When you give the *encfs* *~/.encfs ~/encfs* command, the program prompts you to supply the options shown in Listing 4. If one of the directories or both are missing, Enc-FS will create them for you (Lines 1 and 2). When run for the first time, the system creates a *.encfs5* file in the source directory with the information required to encrypt the files stored in this directory. When you back up your data, make sure you include this file.

Enc-FS offers the options of operating in a standard mode and a paranoid mode. Standard mode uses the Blowfish algorithm with a key length of 160 bits and encrypts file names. The software processes 512 byte blocks during the encryption procedure, links the initialization vectors, and initializes the file headers separately.

Paranoid mode promises more security by using AES with a block size of 256 bits. In addition to the steps performed by standard mode, paranoid Enc-FS stores every block with a checksum to detect modifications to the data. In addition to this checksum feature, the filename is included in the initialization

Listing 5: Enc-FS Code

```
void CipherV3::randomize( unsigned char *buf, int len ) const
{
    memset( buf, 0, len );
    if(RAND_bytes( buf, len ) == 0)
    {
        char errStr[120];
        unsigned long errVal = 0;
        if((errVal = ERR_get_error()) != 0)
        {
            rWarning("openssl error: %s", ERR_error_string( errVal, errStr ));
        }
    }
}
```

advertisement

vector for the content. Renaming a file thus leads to complete re-encryption. This process breaks hardlinks, causing problems for programs such as Mutt or Procmail.

In addition to this, Enc-FS has an expert mode that allows users to select their own settings from a choice of all OpenSSL algorithms. If you prefer to leave filenames in the clear, the program gives you the option to do so.

In contrast to Crypto-FS, Enc-FS has a number of useful functions that make the system easier or safer to use. An Enc-FS mount can be unmounted automatically after a pre-defined period of time. This is a very useful option in combination with PAM integration [25]. And Enc-FS can hold sway with its stable mates with respect to choice of platform: version 1.3 or newer of the software is available for native FreeBSD, and there is a Windows port at [26].

Enc-FS encrypts each file with a block cipher, such as AES in CBC mode. However, due to a programming error, the software uses CFB (Cipher Feedback) rather than CBC (Cipher Block Chaining). Each 512-byte sector contains 504 bytes of data and an 8-byte HMAC (hashed MAC) of the cleartext.

Instead of removing the HMAC, Enc-FS Xors all the bytes. This value is also used as the initialization vector. If a single bit in the sector cleartext changes, the IV and therefore the whole enciphered sector change, too.

Enc-FS protects filenames by first encrypting them and then storing a Base-64 encoded version. Again, the program uses an HMAC derived from the filename as the IV. To prevent files of the same name returning identical ciphertext, Enc-FS uses the absolute pathname to calculate the MAC.

The encryption mode is a do-it-yourself complex. The program starts by Xoring each byte with the following byte. If the sector size is the same as the encryption algorithm block size, Enc-FS then encrypts the whole sector in CBC mode. If not, it uses CFB. This looks like a programming error, as the securer CBC would be preferable if the volume of data is an integral multiple of the block size.

The sector is always going to be bigger than an encryption block, and thus CBC is never used. Enc-FS then goes on to reverse the order of the bytes in every 64 byte section. Finally, it goes through another round of Xor and encryption operations. This next round is obviously designed to achieve the same data masking effect as an adjustable wide-block cipher, but it does not achieve the same cryptographic security, although the operations are just as expensive in terms of CPU usage.

Password transformation does without a salt in Enc-FS and is limited to 16 hashing iterations. One or two thousand iterations are considered the minimum to prevent dictionary attacks.

Like many of its competitors, Enc-FS hardly bothers to check function return values. If something goes astray, the data remain unencrypted. Even where the code checks the return values, it makes serious errors. Instead of ensuring that the operation has been performed successfully, the program defaults to assum-

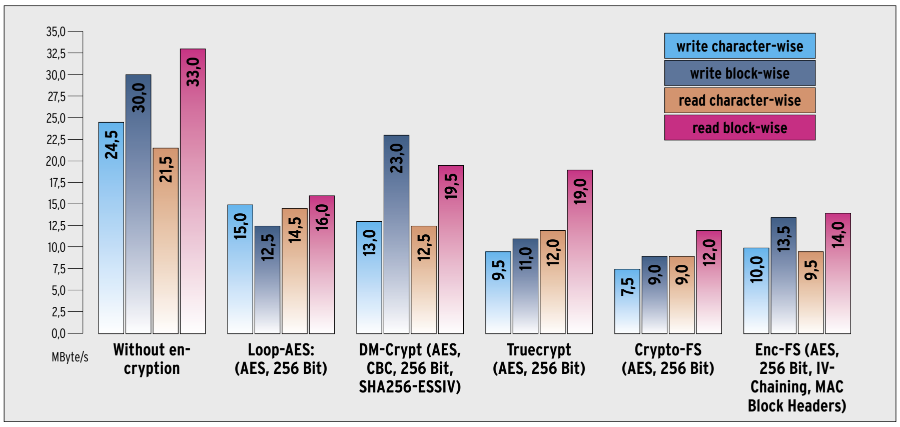

Figure 4: The Bonnie++ benchmark shows that encryption affects read and write performance. Loop-AES has the fastest character-wise write performance, but DM-Crypt reads faster block-wise. Truecrypt wins on block-wise reading. Enc-FS is amazingly quick for a user-space filesystem.

INFO

[1] Loop-AES: *http://loop-aes.sourceforge.net*

[2] Util-linux: *ftp://ftp.kernel.org/pub/linux/utils/util-linux/*

[3] Readme for Loop-AES: *http://loop-aes.sourceforge.net/loop-AES.README*

[4] Crosscrypt: *http://www.scherrer.cc/crypt/*

[5] DM-Crypt: *http://www.saout.de/misc/dm-crypt/*

[6] Cryptsetup-LUKS: *http://luks.endorphin.org/dm-crypt*

[7] Clemens Fruhwirth and Markus Schuster, "Secret Messages: Hard disk encryption with DDM-Crypt, LUKS, and cryptsetup," Linux Magazine 12/05, pg. 65.

[8] Free OTFE: *http://www.freeotfe.org*

[9] LUKS for the Masses: *http://luks.endorphin.org/masses*

[10] RFC 2898, "PKCS #5 – Password-Based Cryptography Specification Version 2.0": *http://tools.ietf.org/html/rfc2898*

[11] Truecrypt: *http://www.truecrypt.org*

[12] Moses Liskov, Ron Rivest, David Wagner, "Tweakable Block Ciphers": Proceedings of Crypto 2002, Springer-Verlag, Lecture Notes in Computer Science No. 2442

[13] Clement Kent (Editor), "Draft Proposal for Tweakable Narrow-block Encryption": IEEE P1619 Working Group, 6. August 2004

[14] Morris Dworkin, "Recommendation for Block Cipher Modes of Operation – Galois/Counter Mode (GCM) for Confidentiality and Authentication", NIST Special Publication 800-38D: *http://csrc.nist.gov/publications/drafts/Draft-NIST_SP800-38D_Public_Comment.pdf*

[15] Pamscript: *http://linux.bononline.nl/linux/pamscript/01/build.html*

[16] LUFS: *http://lufs.sourceforge.net/lufs/*

[17] FUSE: *http://fuse.sourceforge.net*

[18] Gmail-FS: *http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html*

[19] FTPFS: *http://ftpfs.sourceforge.net*

[20] Crypto-FS: *http://www.reboot.animeirc.de/cryptofs/*

[21] Libgcrypt: *ftp://ftp.gnupg.org/gcrypt/alpha/libgcrypt/*

[22] Glib: *http://www.gtk.org*

[23] Enc-FS: *http://arg0.net/wiki/encfs*

[24] Rlog: *http://arg0.net/wiki/rlog*

[25] Pam_encfs: *http://hollowtube.mine.nu/wiki/index.php?n=Projects.PamEncfs*

[26] Enc-FS for Windows: *http://www.crc32.net/encfs/*

[27] Bonnie++: *http://www.coker.com.au/bonnie++/*

ing that everything has turned out okay, and it just checks for a few of all possible error conditions. If the function fails with an unexpected return value, Enc-FS ignores the error and carries on regardless.

Speed

We asked all the crypto filesystems we tested to show their pace in a benchmark. Our lab machine was an IBM Thinkpad T40p with 1.5 GB RAM and a 7200 rpm Hitachi hard disk. The values were measured on Ubuntu 6.06 LTS using Bonnie++ [27] with 3-GB datasets. XFS was the filesystem in the encrypted container in each case.

The results in Figure 4 clearly show the performance hit that encryption causes. Although it only achieves half the normal throughput, the DM-Crypt filesystem demonstrates fairly constant read and write speeds, its strongest point being block-wise operations. Truecrypt's write performance is poor; it can't even match the Enc-FS userspace tool for speed. But it picks up speed again on read operations, almost matching DM-Crypt for pace.

Loop-AES is the fastest of the candidates for character-wise read and write operations, although it loses out to DM-Crypt on block operations. For a user-space filesystem Enc-FS does fairly well, its strongest point being block-wise read and write operations. However, smaller files see it drop down to a similar speed as Truecrypt, with particularly poor read performance. The Crypto-FS filesystem lags well behind the field, achieving just one third of the throughput for unencrypted data.

Where's the Beef?

Most crypto filesystems on Linux leave the user with a sour taste. They trip up over cryptographic pitfalls, and the implementations are typically fairly weak (typical errors: failure to check function return values). Of all things, Truecrypt – a program originally developed for Windows, and one that does not integrate well with Linux as of this writing – is the positive exception to the rule. DM-Crypt in the Cryptsetup LUKS variant takes second place. Its performance is good, and it evidences less cryptographic errors than the other contenders, assuming that the user opts for a secure configuration. The default settings are unnecessarily insecure. And it is disappointing that the LRW patch has not made it into the kernel. ■

THE AUTHOR

Christian Ney is a Unix and firewall administrator with a regional airline, and also provides security and high-availability consultancy services to mid-sized business. Peter Gutmann works for the Department of Computer Science at the University of Auckland, New Zealand. He is involved in designing and analyzing cryptographic security architectures, co-authored PGP, and has published numerous reports and RFCs on security and encryption. Peter Gutmann is also the author of the Cryptlib open source security toolkit, and of "Cryptographic Security Architecture Design and Verification" (Springer, 2003).