LINUX VOICE MASTERCLASS

You wouldn't want other people opening your letters and your data is no different. Encrypt it today!

CIPHERSHED: ENCRYPTION FOR EVERYONE

TrueCrypt lives on as CipherShed, so it's still really easy to protect your valuable data.

JOHN LANE

LV PRO TIP

Truecrypt / Cyphershed requires root privileges. If you can "sudo" then you'll be ok.

Everybody has something to hide. It might be a little more mundane than what our governments get up to but, to each of us, that something is important and valuable enough to protect. It could be your personal finances, or perhaps that new app or book you've been working on. If your laptop were stolen, it would be pretty useless if your precious data were encrypted. The good news is that it's easy and, this month, we show you how.

One of the best freely available encryption tools over the past decade was *TrueCrypt*. It provided on-the-fly filesystem encryption and was a cross-platform solution that worked, not only on Linux, but on Windows and Mac OS X too.

Back in May, *TrueCrypt* as we know it ceased to exist. Its SourceForge site was replaced with some basic pages claiming that it is "insecure and may contain unfixed security issues". It now only provides guidance for migrating away, and the only download available is for version 7.2, a limited functionality version that can only decrypt. However, general opinion is that these claims are unfounded and the original developers just asserted their right to kill the product. But the free software community is making sure that the story doesn't end there.

The latest news is, of course, that *TrueCrypt* has been forked and is moving forward as *CipherShed*; you can install the current development version from its GitHub repository. It's based upon and named similarly to *TrueCrypt* version 7.1a. There are instructions for installing on Debian-based distros, and Arch Linux users can build it from a package in the Arch User Repository. Here's what you need to do if you're on Ubuntu or another Debian-based distro:

```
$ sudo apt-get git build-essential
$ sudo apt-get install libwxgtk2.8-dev nasm libfuse-dev
$ git clone https://github.com/CipherShed/CipherShed.git
$ cd CipherShed/src
$ LIBS="-ldl" make
$ sudo install -m755 {Main,/usr/bin}/ciphershed
```

So, what's it all about? Well, you get very straightforward encryption tools that you can configure using a GUI or command line interface. They make encrypted filesystems either on real disks or partitions, or as virtual disks contained within a file and mounted as a real disk. It claims to offer plausible deniability by creating volumes hidden undetectably

CipherShed's goals include a secure audited codebase that is released under an OSI-approved licence.

AES new instructions

If your computer has a recent Intel or AMD processor then it may support AES-NI. This is a set of new x86 CPU instructions that provide hardware-accelerated AES encryption, allowing encryption tasks to be performed four to eight times faster.

Support for AES-NI was introduced with *TrueCrypt* version 7.0. You can check whether your system supports it by looking at Settings > Preferences > Performance. If you have support but prefer not to use a proprietary encryption mechanism then you can disable it on the same screen.

Moving your mouse rapidly provides entropy, which is used to generate encryption keys.

within others and can even boot operating systems hidden in this way.

This is transparent encryption. Once you set up and mount an encrypted volume, you use it just like any other. Copy files there, work on them, edit them, delete them. Do whatever you would do with unencrypted files. All the while, the encryption happens in the background. Once you unmount the volume, the data inside is secure.

Your first encrypted volume

Begin by typing **truecrypt** or **ciphershed** at a command prompt to start the GUI. Press the Create Volume button to launch the Volume Creation Wizard. This offers two choices – you can either create an encrypted file container (a virtual encrypted disk within a file), or you can create a volume on a partition/drive – essentially any valid block device that you can create a filesystem on.

The first option is best to experiment with; select it and press "Next" to proceed. Now choose between a standard or hidden volume. Choose a standard volume and, on the following page, a location for it.

You're then offered the choice of several encryption and hashing algorithms, but the defaults offer an appropriate balance between speed and security. If you're paranoid, choose a stacked scheme like "AES-Twofish-Serpent" – these apply multiple algorithms one after the other but result in slower read/write times.

Also consider...

There are other *TrueCrypt* derivatives besides *CipherShed* that you may also like to try.

VeraCrypt contains enhanced security algorithms that, the developers claim, make it immune to new developments in brute-force attacks and solves vulnerabilities found in *TrueCrypt*. These enhancements, however, mean its storage format is incompatible with *TrueCrypt*. Read more on their website at **http://sourceforge.net/projects/veracrypt**.

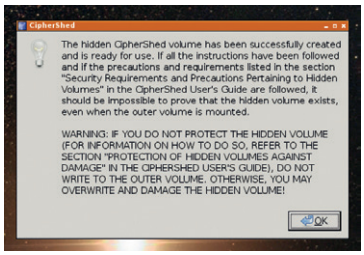
Realcrypt is essentially *TrueCrypt* with the branding changed. It's available for Fedora users in the RPM Fusion repository **http://rpmfusion.org/Package/realcrypt**.

Tcplay is a free BSD-licensed command-line *TrueCrypt* implementation based on the Linux kernel's **dm-crypt** device mapper (**https://github.com/bwalex/tc-play**). It is compatible with *TrueCrypt* volumes.

Plausible deniability

People keen on privacy and encryption are well aware that the weak link is the person. You can have the strongest keys but they won't protect you from being forced to reveal them. Being able to reveal a fake key in such scenarios is attractive and a *TrueCrypt* hidden volume enables you to do just that.

A hidden volume is created within the free space of a normal volume in such a way that it cannot be detected. Each has different passphrases and the volume that gets mounted is selected by the given passphrase. This feature would allow someone under duress to give out the normal volume passphrase, allowing access to whatever seemingly important files were placed there while leaving the true secrets protected within the hidden volume, affording plausible deniability in the event that it should be necessary.



Hidden volumes offer an added layer of protection – just be careful not to overwrite it.

The final pages set the volume's size and password (this can be a passphrase – using single words is insecure). You can also use keyfiles to enhance security. These are just normal files that can be on your hard drive or removable media. The first 1024 kilobytes of each key file is considered as part of the passphrase that is required to unlock the volume (you should therefore only choose files that won't change). You can leave the password field empty if you use keyfiles, although it's less secure if you do. The final choice you have is the encrypted volume's filesystem and this can be Windows FAT format or Linux ext2–4.

You then land on the Volume Format screen, which will compute a volume key before formatting. It invites you to move your mouse around as a way of gathering entropy for the key.

Favourite mounts

You need to mount devices before you can use them. The Volume section of the main window is where you select a file or device and use the Mount button to mount it. This is when you need to supply the pass phrase and any required key files. Once the device is mounted, it's accessible as a subdirectory of **/media** and you use it like any other filesystem.

You can optionally cache the passphrases and key files in memory to avoid having to re-enter them on successive mounts. The cache only persists while the encryption driver is running. You close the *CipherShed* GUI by pressing its Exit button. If you have mounted volumes, *CipherShed* goes into the background and presents itself as a taskbar icon that you can use to re-open the main window or quickly mount/unmount favourite volumes via a right-click pop-up menu. *CipherShed* terminates if you exit when there are no mounted volumes in place.

TrueCrypt has, for a long time, been one of the easiest ways to use some of the most secure methods available for encrypting sensitive data. *CipherShed* aims to continue that legacy and should mean that we'll be able to continue securing our data with an easy-to-use GUI desktop application.

LV PRO TIP

You can download the TrueCrypt 7.1a User's Guide PDF **http://bit.ly/tc71a_ug**.

KEEP YOUR DATA SAFE WITH ENCRYPTION

Linux has baked-in encryption capabilities. Use them or regret it when your laptop gets stolen.

JOHN LANE

The Linux kernel has a feature called a device mapper. It allows virtual block devices to be created that are based on other block devices, and there's a device mapper module called **dm-crypt** that we can use to create encrypted block devices.

The device mapper allows devices to be stacked. You can, for example, create RAID or LVM devices and then encrypt them. You can also do it the other way around. You need userspace tools to work with the **dm-crypt** kernel module. The primary one, **cryptsetup**, is used to administer encrypted volumes and requires root privileges. The other tool is **cryptmount**; it allows unprivileged users to mount encrypted volumes.

Encrypted volumes can either be formatted or raw. Formatted volumes contain metadata that describes the encrypted payload, whereas raw volumes are just encrypted disk blocks. Use of raw volumes requires things like cipers, keys, etc, to be provided as command line parameters; they should be considered as an expert-level option.

The standard volume format on Linux is called LUKS – the Linux Unified Key Setup format. **Cryptsetup** also supports the *TrueCrypt* format. The LUKS format uses a header at the start of the volume that contains metadata including cipher details and eight key-slots. You can have up to eight different salted, hashed and changeable pass phrases that decrypt a master key to unlock the data payload. LUKS automatically configures non-default **dm-crypt** parameters to make it more secure. The format occupies a header that can consume between 1 and 2MB of the volume's capacity.

Begin by installing the userspace tools; your distro should carry them in its package repository:

```
$ sudo apt-get install cryptsetup cryptmount
```

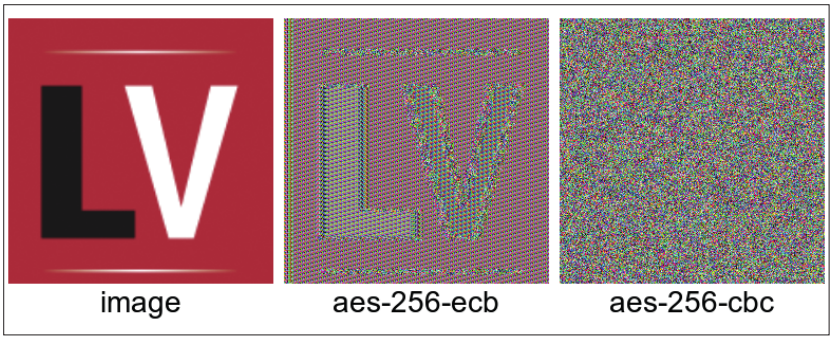
We'll begin by using **cryptsetup** to format a block device with LUKS.

LV PRO TIP

The LUKS default header size of 2MB maintains sector alignment of the LUKS volume with the underlying device. LUKS uses 512-byte blocks.

LV PRO TIP

Earlier **cryptsetup** versions required **luksOpen** and **luksClose** instead of **open** and **close**.



Choose an appropriate cipher: an encrypted bitmap can reveal cryptographic weaknesses.

A block device in a file

You aren't restricted to real block devices – you can create an encrypted volume in a regular file. To do this, just create a file of whatever size device you want:

```
$ head -c 100M /dev/urandom > /path/to/myvolume
```

You can then use the file's path wherever **cryptsetup** expects a device.

```
$ cryptsetup luksFormat /dev/mydevice
```

The default cipher that you get depends on the version of **cryptsetup** that you have. Since version 1.6.0, this is **aes-xts-plain64**, where **aes** is the cipher and **xts** is the chaining mode that affects how the cipher is applied to subsequent blocks of data. **xts** is an improvement over the **cbc** mode used by prior versions. Go with the defaults unless you have reason to change them; you can specify an alternative with the **--cipher** command line argument.

So far, we have an encrypted block device but it needs to be opened (you'll be asked for the pass phrase). You can can then put a filesystem onto it and mount it:

```
$ cryptsetup open /dev/mydevice myvolume
$ mkfs.ext4 /dev/mapper/myvolume
$ mount /dev/mapper/myvolume /mnt
```

myvolume is how the device mapper will identify the unlocked device; you can use any meaningful label. To take an encrypted filesystem offline, unmount and then close it:

```
$ umount /mnt
$ cryptsetup close myvolume
```

Boot configuration

You can automatically unlock encrypted devices when your system boots. The encrypted device table is a file called **/etc/crypttab** that is similar to the **/etc/fstab** used for mounts. You specify four things: a device mapper name, the device path, an optional key file (or just "none") and options. Use the "luks" option to specify the format:

```
myvolume /dev/mydevice none luks
```

The listed volumes will be opened at boot time and this will require entry of the pass phrases. An alternative to passphrase entry is to store it in an appropriately secured key file:

```
$ sudo echo -n 'my secret passphrase' > /root/keyfile
$ sudo chmod 0400 /root/keyfile
```

You can, if you want, use a more complex key now that it won't need to be manually entered. It's

customary to use a large blob of random data:

```
$ head -c 4096 /dev/urandom > /root/keyfile
```

Specify the key file in **/etc/crypttab** instead of "none" and add it to the LUKS header – you'll need to enter an existing passphrase to unlock it before the new one can be added:

```
$ cryptsetup luksAddKey /dev/myvolume /root/keyfile
```

Once a volume is opened, it can be mounted in the usual way with an entry in the **/etc/fstab** file.

You can use **/etc/crypttab** for all filesystems and swap devices. However, if you want to encrypt your root partition, then your system's **initrd** will need cryptography support. You should refer to your distro's docs for more information about this because boot configurations vary. The system's BIOS needs to read the boot partition, so that cannot be encrypted.

Encrypted filesystems

Another approach to encryption is to use Encrypted Filesystems. These are virtual filesystems stacked on top of existing ones. They provide cleartext read/write access to encrypted files stored in the underlying filesystem. Encrypted filesystems work at the filesystem level, whereas **dm-crypt** operates at the block level, beneath the filesystem.

Two encrypted filesystems available for Linux are eCryptFS and EncFS; the latter runs entirely in userspace (it's FUSE-based) and, therefore, doesn't require elevated privileges to use it. Using EncFS is straightforward. You specify an encrypted directory where your files will be stored and an unencrypted directory where you'll read and write them:

```
$ encfs ~/.mysecrets_encrypted ~/mysecrets
```

You need to use absolute paths. Follow the instructions: select the "Pre-Configured Paranoia Mode" for suitable defaults or, for more control, use the advanced mode. When your shell prompt returns, you'll have an encrypted filesystem. A file written to **mysecrets** will be transparently encrypted and stored in **.mysecrets_encrypted**:

```
$ echo "This is my secret" > mysecrets/test
$ ls -l mysecrets
-rw-rw-r-- 1 myuser users    18 Oct 20 14:08 test
$ ls -la .mysecrets_encrypted
-rw-rw-r-- 1 myuser users  1092 Oct 20 13:58 .encfs6.xml
-rw-rw-r-- 1 myuser users    34 Oct 20 14:08
5gk8Df5Gk3eN0sJx1fiqPppA
```

Notice the encrypted file is larger and has an indecipherable name. There's also a hidden file called **.encf6.xml** containing the metadata required. You use the FUSE mount command to unmount your encrypted filesystem:

```
$ fusermount -u ~/mysecrets
```

Mounting is performed using the same **encfs** command that was used above to create the filesystem – it only creates a configuration if it doesn't already exist. A useful thing to know is that the encrypted and plaintext directories can be on different filesystems. One useful application for this is encrypting files in cloud-based storage like DropBox: you can do something like this:

```
$ encfs ~/Dropbox/Private ~/Private
```

The other part to EncFS is **encfstcl**, an administrative tool that can display information about an encrypted filesystem but is mostly useful to change its password:

```
$ encfsctl passwd ~/.mysecrets_encrypted
```

You can also use **encfsctl** to access an encrypted directory without mounting it.

```
$ encfsctl ls ~/.mysecrets_encrypted
```

We've covered the two main ways to perform transparent encryption but neither suit if you just want to secure a single file. You can do this quickly with nothing more than OpenSSL – you can encrypt a file like this:

```
$ openssl aes-256-cbc < plaintext > encrypted
```

and decrypt it

```
$ openssl aes-256-cbc -d < encrypted > plaintext
```

The **aes-256-cbc** cipher gives good protection but, while this method achieves its objective, it's more practical to use a public key infrastructure to share encrypted files. OpenPGP is an example of this that we'll explore next month. LV

John Lane provides technical solutions to business problems. He has yet to find something Linux can't solve.

cryptmount volumes for unprivileged users

The other userspace tool for **dm-crypt** is called **cryptmount**, which offers an easy way for unprivileged users to use encrypted volumes. Root privileges are required to create an encrypted volume for an unprivileged user but that user can mount and unmount it without any special privileges.

```
$ sudo cryptmount-setup
```

Follow the prompts – you need to enter a volume name, username and absolute paths to a mount point and the file that will contain the encrypted volume. Both are created automatically and an ext3 filesystem is created inside the file. The volume's conifguration is written into **/etc/cryptmount/cmtab** and the key is securely stored in a file in the **/etc/cryptmount** directory. You'll also be asked for a passphrase to secure the key and the user will need to enter this when mounting their volume. They do that with **cryptmount**:

```
$ cryptmount myvolume
```

Unmounting is similar:

```
$ cryptmount --unmount myvolume
```

Cryptmount might be more appropriate for some applications. Unprivileged users could, for example, have encrypted volumes on USB sticks and be able to use them without help from the systems administrator.

Use TrueCrypt volumes with cryptsetup

Cryptsetup has been able to open *TrueCrypt* volumes since version 1.6. You just need to specify the volume type:

```
$ cryptsetup open --type tcrypt /path/to/myvolume myvolume
$ mount /dev/mapper/myvolume /mnt
```

If you want to mount a hidden volume, add the **--tcrypt-hidden** command-line argument and use **--key-file** if you need to specify key files.

LV PRO TIP

Ubuntu users can do **cryptdisks_start myvolume** to open a volume in **/etc/crypttab**. **cryptdisks_stop** closes it. systemd has **cryptsetup.target**.

LV PRO TIP

cat /proc/crypto lists the kernel's available ciphers and supported key sizes.

LV PRO TIP

EncFS is used by **boxcryptor.com/classic**, which may be handy if you need to access protected directories from other platforms.