Systems, Tools, and Terminal Science

Posted on

GnuPG provides us with a means to securely encrypt individual files on a filesystem, but for really high-security information or environments, it may be appropriate to encrypt an entire disk, to mitigate problems such as caching sensitive files in plaintext. The GNU/Linux kernel includes its own disk encryption solution in the kernel, `dm-crypt`. This can be leveraged with a low-end tool called `cryptsetup`, or more easily with LUKS, the , implementing strong cryptography with passphrases or keyfiles.

In this example, we'll demonstrate how this can work to encrypt a USB drive, which is a good method for securely storing really sensitive data such as PGP master keys that's only needed occasionally, rather than leaving it always mounted on a networked device. Be aware that this erases any existing files on the drive.

The cryptographic tools used by `dm-crypt` and LUKS are built-in to Linux kernels after 2.6, but you may have to install a package to get access to the `cryptsetup` frontend. On Debian-derived systems, it's available in `cryptsetup`:

```
# apt-get install cryptsetup
```

On RPM-based systems like Fedora or CentOS, the package has the same name, `cryptsetup`:

```
# yum install cryptsetup
```

After identifying the block device on which we want the encrypted filesystem, for example `/dev/sdc1`, we can erase any existing contents using a call to `wipefs`:

```
# wipefs -a /dev/sdc1
```

Alternatively, we can zero out the whole disk, if we want to completely overwrite any trace of the data previously on the disk; this can take a long time for large volumes:

```
# cat /dev/zero >/dev/sdc1
```

If you don't have a USB drive to hand, but would still like to try this out, you can use a loop block device in a file. For example, to create a 100MB loop device:

```
# dd if=/dev/zero of=/loopdev bs=1k count=102400
102400+0 records in
102400+0 records out
104857600 bytes (105 MB) copied, 0.331452 s, 316 MB/s
# losetup -f
/dev/loop0
# losetup /dev/loop0 /loopdev
```

You can then follow the rest of this guide using `/dev/loop0` as the raw block device in place of `/dev/sdc1`. In the above output, `losetup -f` returns the first available loop device for use.

Setting up a LUKS container on the block device is then done as follows, providing a passphrase of decent strength; as always, the longer the better. Ideally, you should not use the same passphrase as your GnuPG or SSH keys.

```
# cryptsetup luksFormat /dev/sdc1

WARNING!
========
This will overwrite data on /dev/sdc1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
```

This creates an abstracted encryption container on the disk, which can be opened by providing the appropriate passphrase. A virtual mapped device is then provided that encrypts all data written to it transparently, with the encrypted data written to the disk.

We can open the mapped device using `cryptsetup luksOpen`, which will prompt us for the passphrase:

```
# cryptsetup luksOpen /dev/sdc1 secret
```

The last argument here is the filename for the block device to appear under `/dev/mapper`; this example provides `/dev/mapper/secret`.

With this done, the block device on `/dev/mapper/secret` can now be used in the same way as any other block device; all of the disk operations are abstracted through encryption operations. You'll probably want to create a filesystem on it; in this case, I'm creating an `ext4` filesystem:

```
# mkfs.ext4 /dev/mapper/secret
mke2fs 1.42.8 (20-Jun-2013)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
25168 inodes, 100352 blocks
5017 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
13 block groups
8192 blocks per group, 8192 fragments per group
1936 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

We can then mount the device as normal, and data put into the newly created filesytem will be transparently encrypted:

```
# mkdir -p /mnt/secret
# mount /dev/mapper/secret /mnt/secret
```

For example, we could store a private GnuPG key on it:

```
# cp -prv /home/tom/.gnupg/secring.gpg /mnt/secret
```

We can get some information about the LUKS container and the specifics of its encryption using `luksDump` on the underlying block device. This shows us the encryption method used, in this case `aes-xts-plain64`.

```
# cryptsetup luksDump /dev/sdc1
LUKS header information for /dev/sdc1


Version:        1
Cipher name:    aes
Cipher mode:    xts-plain64
Hash spec:      sha1
Payload offset: 4096
MK bits:        256
MK digest:      87 6d 08 59 b2 f0 c6 6e ca ec 5f 72 2c e0 35 33 c2 9e
MK salt:        7f a5 38 4c 14 85 61 cb 6c 22 65 48 87 21 60 8f
                fa 40 2a ab ae 7d cc df c9 9b a4 e3 3c 64 b6 bb
MK iterations:  49375
UUID:           f4e5f28c-3b34-4003-9bcd-dbb2352042ba

Key Slot 0: ENABLED
        Iterations:             197530
        Salt:                   2d 57 f6 2b 44 a6 61 ee d6 ee e4 7d 64
                                55 16 09 83 b4 f0 94 ca 19 17 11 a9 34
        Key material offset:    8
        AF stripes:             4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

When finished with the data on the device, we should both unmount any filesystem on it, and also close the mapped device so that the passphrase is required to re-open it:

```
# umount /mnt/secret
# cryptsetup luksClose /dev/mapper/secret
```

If the data is a removable device, you should also consider physically removing the media from the machine and placing it in some secure location.

This post only scratches the surface of LUKS functionality; many more things are possible with the system, including automatic mounting of encrypted filesystems and the use of stored keyfiles instead of typed passphrases. The `cryptsetup` contains a great deal of information, including some treatment of data recovery, and the Arch Wiki has on various ways of using LUKS securely.

- 
- 
- 
- 
- 
- 
- 
- 
- GNU/Linux Crypto: Disks
- 

This entry is part 9 of 10 in the series .