Partition Encryption in Linux

Matthew Gracie, Information Security Administrator, Canisius College
graciem@canisius.edu

There are two broad classes of encryption that are commonly deployed on computer storage. The first is partition encryption, also known as “whole-disk encryption”. This refers to encrypting an entire logical block device with a single key, so that it can be decrypted and accessed all at once. The second is filesystem-level encryption, which refers to individually encrypting each file on a disk, often with different keys.

This paper deals with partition encryption in a Linux environment. By following the instructions herein, the reader will be able to create, mount, unmount, and otherwise manipulate an encrypted block device, such as a hard drive, on a computer running a Linux-based operating system. The instructions have been tested on computers running the Debian GNU/Linux operating system, but they should be transferable with little or no modification to Red Hat, SuSE, or other variants.

Why Encrypt?

Encryption acts as an excellent safeguard against the loss of data in an environment where the physical security of a system cannot be ensured. Examples are abundant.

A Linux-based server, placed at a disaster recovery site, a co-location facility, or a satellite campus could use an encrypted data partition to prevent access outside of the traditional operating system means. Even if someone pulls the hard drive from the machine and places it in another one as a slave, or boots from a LiveCD, the encrypted partitions will be unreadable without the key.

A Linux-based laptop, if stolen, would prevent the thief from retrieving the data stored on it. Any attempt to read the drive without the key would be unsuccessful; the only way to use the laptop for anything would be to reformat the drive, rendering the data on it unusable to the thief.

A USB flash drive or external hard drive, accidentally misplaced, would be useless to someone finding it unless they reformat the device and destroy the data on it.

In short, if a storage device is not completely physically secure, and the data on that device is worth anything at all, then encryption is a wise decision.

Please note that, like any other security measure, partition encryption is not a silver bullet. If a device is mounted, and the host operating system has access to it, then an intruder who has compromised the host operating system will have access to the data. Encryption does not solve problems related to unpatched application vulnerabilities, weak passwords, poorly implemented permissions models, social engineering, or other classic system administration pitfalls. It is simply one more layer in a properly deployed defense-in-depth strategy to defend the data on a device from a hostile or curious attacker.

Tools to Use

There are three basic elements that make up the partition encryption solution built into the 2.6 series of Linux kernels.

Device Mapper: The device mapper is an interface that allows for block devices to be created, or “mapped”, on top of other block devices. It is a crucial piece of kernel infrastructure that allows for technologies like Linux Volume Management and the software RAID interface, as well as the loopback file interface. This functionality is native to the 2.6 series of Linux kernels.

DM-Crypt: the dm-crypt subsystem in the kernel adds additional encryption capabilities to the device mapper outlined above. It transparently encrypts any block device, using tools provided by the Linux Crypto API. This means that various hashes and block ciphers can be used by dm-crypt, as long as the kernel in question supports them.

LUKS: Linux Universal Key Storage (LUKS) is a standard, on-disk format for storage of encryption keys. This means that multiple tools can read volumes that use the LUKS standard, and the keys are managed in a safe, standard, and well-documented fashion.

So, with these three tools, we have a method for abstracting block devices, a method for encryption on block devices, and a method for handling keys and passphrases. As you might imagine, these combine to give us a simple and powerful way to handle partition encryption.

Creating an Encrypted Partition

Throughout this example, I will assume that the partition to be encrypted is the first primary partition on the second hard drive, designated by the kernel as “/dev/hdb1”. These instructions should work for any block device; simply substitute the name of the device that you are formatting and encrypting for “hdb1” in the examples.

WARNING: Encrypting a partition will destroy all of the data that is currently stored on it. Do not use these directions to encrypt a partition that you are using to store data that you care about! Back it up first, and restore to the encrypted volume later.

The first thing that you need to do is make sure that your Linux installation has the proper tools for creating an encrypted partition. As root, run the following command:

```
# cryptsetup --help
```

This should print a few lines of text, listing all of the options available through the cryptsetup tool. Make sure that there are details on actions like “luksFormat”, to be sure that your version of cryptsetup supports the LUKS standard. If you are unable to run the cryptsetup command, or if the help screen does not include LUKS commands, check the documentation for your Linux distribution for installation instructions for these features.

Now that we know that we have the correct tool, we need to establish the partition. We will use the dm-crypt facility in the device mapper to create a logical partition mapped to our physical partition on the second hard disk. This step will ask for a passphrase; be sure to use common techniques for generating a passphrase (mixed case, numbers, punctution, and so on) and record it somewhere. There is **no** recovery method for a forgotten passphrase in LUKS.

```
# cryptsetup luksFormat /dev/hdb1
```

This will tell the system that /dev/hdb1 will be an encrypted block device that we will be accessing through a device mapping. By default, the volume will use AES encryption and use a passphrase as a key. If you would like to use a different cipher, or a key file rather than a passphrase, details are available in the cryptsetup man page.

Now we need to create the device mapping:

```
# cryptsetup luksOpen /dev/hdb1 vault
```

You will be prompted for the LUKS passphrase for the device, and once that is entered, the mapping will be created. If you look in the /dev/mapper directory, there should now be a file named /dev/mapper/vault. This is the mapped block device that you will mount and manipulate, like any other file system.

Note: the name “vault” is purely arbitrary in this example; you can name the mapped device whatever you like. Many people advocate using a syntax like “mapped_device_X” where the X is the block device being mapped: in this case, it would be “mapped_device_hdb1”.

Now we need to format the filesystem. Again, all interactions with the encrypted device take place through the mapped block device.

```
# mkfs.ext3 /dev/mapper/vault
```

This will format the encrypted drive as an ext3 volume. Any filesystem can be used, so use whatever your institution has standardized on.

Once the volume is formatted, it can be mounted as normal.

```
# mkdir /home2
# mount –t ext3 /dev/mapper/vault /home2
```

You now have a freshly formatted, encrypted partition mounted on the server at /home2. It can be added to /etc/fstab, shared over NFS, or otherwise manipulated in any way that a normal partition can be.

Future Maintenance

Obviously, the partition will need some maintenance in the future. The computer that you made it on is not going to just continue running forever, so at the very least, we'll need to look at mounting and unmounting the encrypted partition.

To unmount the currently-mounted encrypted partition:

```
# umount /home2
# cryptsetup luksClose vault
```

This will unmount the partition from /home2 and remove the device mapping from /dev/mapper. This is very important, especially with removable media; if you attempt to remount the volume when you have not closed it, you will get an error.

To mount an encrypted partition:

```
# cryptsetup luksOpen /dev/hdb1 vault
# mount –t ext3 /dev/mapper/vault /home2
```

This will mount an encrypted partition that was previously formatted on the local machine.

Obviously, you don't want to have to manually mount this partition every time that you reboot the machine; it's onerous on a server-class machine, and downright unacceptable on a personal computer or a laptop. One option is to add the lines above to your rc.local, but the accepted method is to edit two other files in the /etc directory.

In /etc/fstab, add the mapped device as you would any other disk:

```
/dev/mapper/vault /home2 auto defaults 0 0
```

In /etc/crypttab, add the cryptographic settings for the device:

```
vault /dev/hdb1 none luks,retry=5
```

This will tell the computer to map /dev/mapper/vault to /dev/hdb1 on boot and give the user five tries to enter the passphrase. If all five entries are incorrect, the system boot will continue but the partition will not be mounted.

Finally, it is possible to add additional passphrases to the partition. To do so, use the command:

```
# cryptsetup luksAddKey /dev/hdb1
```

Each device using LUKS can have up to eight different passphrases; if your security policy states that you need to have an "emergency only" passphrase in escrow, this would be a good way to do it.

Conclusion

Partition encryption in Linux is fairly simple to implement, especially given the additional security that it provides. This document should have given you all of the information that you need to set up and experiment with an encrypted block device. For more information, check out the man pages for cryptsetup and crypttab. Additionally, there is good information at the following web sites:

LUKS Home Page
http://luks.endorphin.org/

dm-crypt wiki: LUKS
http://www.saout.de/tikiwiki/tiki-index.php?page=LUKS

device-mapper resource page
http://sources.redhat.com/dm/