Appendix C. Disk Encryption

C.1. What is block device encryption?

Block device encryption protects the data on a block device by encrypting it. To access the device's decrypted contents, a user must provide a passphrase or key as authentication. This provides additional security beyond existing OS security mechanisms in that it protects the device's contents even if it has been physically removed from the system.

C.2. Encrypting block devices using dm-crypt/LUKS

Linux Unified Key Setup (LUKS) is a specification for block device encryption. It establishes an on-disk format for the data, as well as a passphrase/key management policy.

LUKS uses the kernel device mapper subsystem via the **`dm-crypt`** module. This arrangement provides a low-level mapping that handles encryption and decryption of the device's data. User-level operations, such as creating and accessing encrypted devices, are accomplished through the use of the **`cryptsetup`** utility.

C.2.1. Overview of LUKS

- What LUKS does:
  - LUKS encrypts entire block devices
    - LUKS is thereby well-suited for protecting the contents of mobile devices such as:
      - Removable storage media
      - Laptop disk drives
  - The underlying contents of the encrypted block device are arbitrary.
    - This makes it useful for encrypting **`swap`** devices.
    - This can also be useful with certain databases that use specially formatted block devices for data storage.
  - LUKS uses the existing device mapper kernel subsystem.
    - This is the same subsystem used by LVM, so it is well tested.
  - LUKS provides passphrase strengthening.
    - This protects against dictionary attacks.
  - LUKS devices contain multiple key slots.
    - This allows users to add backup keys/passphrases.
- What LUKS does *not* do:
  - LUKS is not well-suited for applications requiring many (more than eight) users to have distinct access keys to the same device.
  - LUKS is not well-suited for applications requiring file-level encryption.

More detailed information about LUKS is available from the project website at *http://code.google.com/p/cryptsetup/*.

C.2.2. How will I access the encrypted devices after installation? (System Startup)

During system startup you will be presented with a passphrase prompt. After the correct passphrase has been provided the system will continue to boot normally. If you used different passphrases for multiple encrypted devices you may need to enter more than one passphrase during the startup.

Tip

Consider using the same passphrase for all encrypted block devices in a given system. This will simplify system startup and you will have fewer passphrases to remember. Just make sure you choose a good passphrase!

C.2.3. Choosing a Good Passphrase

While dm-crypt/LUKS supports both keys and passphrases, the anaconda installer only supports the use of passphrases for creating and accessing encrypted block devices during installation.

LUKS does provide passphrase strengthening but it is still a good idea to choose a good (meaning "difficult to guess") passphrase. Note the use of the term "passphrase", as opposed to the term "password". This is intentional. Providing a phrase containing multiple words to increase the security of your data is important.

C.3. Creating Encrypted Block Devices in Anaconda

You can create encrypted devices during system installation. This allows you to easily configure a system with encrypted partitions.

To enable block device encryption, check the "Encrypt System" checkbox when selecting automatic partitioning or the "Encrypt" checkbox when creating an individual partition, software RAID array, or logical volume. After you finish partitioning, you will be prompted for an encryption passphrase. This passphrase will be required to access the encrypted devices. If you have pre-existing LUKS devices and provided correct passphrases for them earlier in the install process the passphrase entry dialog will also contain a checkbox. Checking this checkbox indicates that you would like the new passphrase to be added to an available slot in each of the pre-existing encrypted block devices.

Tip

Checking the "Encrypt System" checkbox on the "Automatic Partitioning" screen and then choosing "Create custom layout" does not cause any block devices to be encrypted automatically.

Tip

You can use **`kickstart`** to set a separate passphrase for each new encrypted block device.

C.3.1. What Kinds of Block Devices Can Be Encrypted?

Most types of block devices can be encrypted using LUKS. From anaconda you can encrypt partitions, LVM physical volumes, LVM logical volumes, and software RAID arrays.

C.3.2. Saving Passphrases

If you use a kickstart file during installation, you can automatically save the passphrases used during installation to an encrypted file (an *escrow packet*) on the local file system. To use this feature, you must have an X.509 certificate available at a location that **anaconda** can access. To specify the URL of this certificate, add the `--escrowcert` parameter to any of the **`autopart`**, **`logvol`**, **`part`** or **`raid`** commands. During installation, the encryption keys for the specified devices are saved in files in **`/root`**, encrypted with the certificate.

You can save escrow packets during installation only with the use of a kickstart file — refer to *Chapter 15, Kickstart Installations* for more detail. You cannot save an escrow packet during an interactive installation, although you can create one on an installed system with the **volume_key** tool. The **volume_key** tool also allows you to use the information stored in an escrow packet to restore access to an encrypted volume. Refer to the **volume_key** manpage for more information.

C.3.3. Creating and Saving Backup Passphrases

If you use a kickstart file during installation, **anaconda** can add a randomly generated backup passphrase to each block device on the system and save each passphrase to an encrypted file on the local file system. Specify the URL of this certificate with the `--escrowcert` parameter as described in *Section C.3.2, “Saving Passphrases”*, followed by the `--backuppassphrase` parameter for each of the kickstart commands that relate to the devices for which you want to create backup passphrases.

Note that this feature is available only while performing a kickstart installation. Refer to *Chapter 15, Kickstart Installations* for more detail.

C.4. Creating Encrypted Block Devices on the Installed System After Installation

Encrypted block devices can be created and configured after installation, using either the following method or **Disk Utility**.

C.4.1. Create the block devices

Create the block devices you want to encrypt by using **`parted`**, **`pvcreate`**, **`lvcreate`** and **`mdadm`**.

C.4.2. Optional: Fill the device with random data

Filling <device> (eg: **`/dev/sda3`**) with random data before encrypting it greatly increases the strength of the encryption. The downside is that it can take a very long time.

Warning

The commands below will destroy any existing data on the device.

- The best way, which provides high quality random data but takes a long time (several minutes per gigabyte on most systems):

  ```
  dd if=/dev/urandom of=<device>
  ```

- Fastest way, which provides lower quality random data:

  ```
  badblocks -c 10240 -s -w -t random -v <device>
  ```

C.4.3. Format the device as a dm-crypt/LUKS encrypted device

Warning

The command below will destroy any existing data on the device.

```
cryptsetup luksFormat <device>
```

Note

For more information, read the **`cryptsetup(8)`** man page.

After supplying the passphrase twice the device will be formatted for use. To verify, use the following command:

```
cryptsetup isLuks <device> && echo Success
```

To see a summary of the encryption information for the device, use the following command:

```
cryptsetup luksDump <device>
```

C.4.4. Create a mapping to allow access to the device's decrypted contents

To access the device's decrypted contents, a mapping must be established using the kernel **`device-mapper`**.

It is useful to choose a meaningful name for this mapping. LUKS provides a UUID (Universally Unique Identifier) for each device. This, unlike the device name (eg: **`/dev/sda3`**), is guaranteed to remain constant as long as the LUKS header remains intact. To find a LUKS device's UUID, run the following command:

```
cryptsetup luksUUID <device>
```

An example of a reliable, informative and unique mapping name would be **luks-<uuid>**, where <uuid> is replaced with the device's LUKS UUID (eg: **luks-50ec957a-5b5a-47ee-85e6-f8085bbc97a8**). This naming convention might seem unwieldy but is it not necessary to type it often.

```
cryptsetup luksOpen <device> <name>
```

There should now be a device node, **/dev/mapper/<name>**, which represents the decrypted device. This block device can be read from and written to like any other unencrypted block device.

To see some information about the mapped device, use the following command:

```
dmsetup info <name>
```

Tip

For more information, read the **dmsetup(8)** man page.

C.4.5. Create filesystems on the mapped device, or continue to build complex storage structures using the mapped device

Use the mapped device node (**/dev/mapper/<name>**) as any other block device. To create an **ext2** filesystem on the mapped device, use the following command:

```
mke2fs /dev/mapper/<name>
```

To mount this filesystem on **/mnt/test**, use the following command:

Important

The directory **/mnt/test** must exist before executing this command.

```
mount /dev/mapper/<name> /mnt/test
```

C.4.6. Add the mapping information to `/etc/crypttab`

In order for the system to set up a mapping for the device, an entry must be present in the **/etc/crypttab** file. If the file doesn't exist, create it and change the owner and group to root (**root:root**) and change the mode to **0744**. Add a line to the file with the following format:

```
<name>  <device>  none
```

The <device> field should be given in the form "UUID=<luks_uuid>", where <luks_uuid> is the LUKS uuid as given by the command **cryptsetup luksUUID <device>**. This ensures the correct device will be identified and used even if the device node (eg: **/dev/sda5**) changes.

Tip

For details on the format of the **`/etc/crypttab`** file, read the **`crypttab(5)`** man page.

C.4.7. Add an entry to `/etc/fstab`

Add an entry to /etc/fstab. This is only necessary if you want to establish a persistent association between the device and a mountpoint. Use the decrypted device, **`/dev/mapper/<name>`** in the **`/etc/fstab`** file.

In many cases it is desirable to list devices in **`/etc/fstab`** by UUID or by a filesystem label. The main purpose of this is to provide a constant identifier in the event that the device name (eg: **`/dev/sda4`**) changes. LUKS device names in the form of **`/dev/mapper/luks-<luks_uuid>`** are based only on the device's LUKS UUID, and are therefore guaranteed to remain constant. This fact makes them suitable for use in **`/etc/fstab`**.

Title

For details on the format of the **`/etc/fstab`** file, read the **`fstab(5)`** man page.

C.5. Common Post-Installation Tasks

The following sections are about common post-installation tasks.

C.5.1. Set a randomly generated key as an additional way to access an encrypted block device

These sections are about generating keys and adding keys.

C.5.1.1. Generate a key

This will generate a 256-bit key in the file **`$HOME/keyfile`**.

```

dd if=/dev/urandom of=$HOME/keyfile bs=32 count=1
chmod 600 $HOME/keyfile
```

C.5.1.2. Add the key to an available keyslot on the encrypted device

```
cryptsetup luksAddKey <device> ~/keyfile
```

C.5.2. Add a new passphrase to an existing device

```
cryptsetup luksAddKey <device>
```

After being prompted for any one of the existing passphrases for authentication, you will be prompted to enter the new passphrase.

C.5.3. Remove a passphrase or key from a device

```
cryptsetup luksRemoveKey <device>
```

You will be prompted for the passphrase you wish to remove and then for any one of the remaining passphrases for authentication.

Appendix D. Understanding LVM

LVM (Logical Volume Management) partitions provide a number of advantages over standard partitions. LVM partitions are formatted as *physical volumes*. One or more physical volumes are combined to form a *volume group*. Each volume group's total storage is then divided into one or more *logical volumes*. The logical volumes function much like standard partitions. They have a file system type, such as `ext4`, and a mount point.

To understand LVM better, imagine the physical volume as a pile of *blocks*. A block is simply a storage unit used to store data. Several piles of blocks can be combined to make a much larger pile, just as physical volumes are combined to make a volume group. The resulting pile can be subdivided into several smaller piles of arbitrary size, just as a volume group is allocated to several logical volumes.

An administrator may grow or shrink logical volumes without destroying data, unlike standard disk partitions. If the physical volumes in a volume group are on separate drives or RAID arrays then administrators may also spread a logical volume across the storage devices.

You may lose data if you shrink a logical volume to a smaller capacity than the data on the volume requires. To ensure maximum flexibility, create logical volumes to meet your current needs, and leave excess storage capacity unallocated. You may safely grow logical volumes to use unallocated space, as your needs dictate.

LVM and the Default Partition Layout

By default, the installation process creates `/` and swap partitions within LVM volumes, with a separate **`/boot`** partition.