dm-crypt/Encrypting an entire system

< Dm-crypt

The following are examples of common scenarios of full system encryption with *dm-crypt*. They explain all the adaptations that need to be done to the normal **installation procedure**. All the necessary tools are on the **installation image (https://archlinux.org/download/)**.

If you want to encrypt an existing unencrypted file system, see **dm-crypt/Device encryption#Encrypt an existing unencrypted file system**.

1 Overview

Securing a root file system is where *dm-crypt* excels, feature and performance-wise. Unlike selectively encrypting non-root file systems, an encrypted root file system can conceal information such as which programs are installed, the usernames of all user accounts, and common data-leakage vectors such as **mlocate** and `/var/log/`. Furthermore, an encrypted root file system makes tampering with the system far more difficult, as everything except the **boot loader** and (usually) the kernel is encrypted.

All scenarios illustrated in the following share these advantages, other pros and cons differentiating them are summarized below:

Scenarios	Advantages	Disadvantages
#LUKS on a partition shows a basic and straightforward set-up for a fully LUKS encrypted root.	▪ Simple partitioning and setup ▪ On a GPT partitioned disk, systemd can auto-mount the root partition.	▪ Inflexible; disk-space to be encrypted has to be pre-allocated
#LUKS on a partition with TPM2 and Secure Boot Similar to the example above, with Secure Boot and TPM2 providing additional layers of security.	Same advantages as above, and ▪ Secure Boot protects against Evil maid attacks ▪ TPM2 prevents the system from being unlocked if Secure Boot is disabled or modified	Same disadvantages as above, and ▪ LUKS volume is automatically unlocked (only if the system was not tampered with).
#LVM on LUKS achieves partitioning flexibility by using LVM inside a single LUKS encrypted partition.	▪ Simple partitioning with knowledge of LVM ▪ Only one key required to unlock all volumes (e.g. easy resume-from-disk setup) ▪ Volume layout not visible when locked ▪ Easiest method to allow suspension to disk	▪ LVM adds an additional mapping layer and hook ▪ Less useful, if a singular volume should receive a separate key
#LUKS on LVM uses dm-crypt only after the LVM is setup.	▪ LVM can be used to have encrypted volumes span multiple disks ▪ Easy mix of un-/encrypted volume groups	▪ Complex; changing volumes requires changing encryption mappers too ▪ Volumes require individual keys ▪ LVM layout is visible when locked ▪ Slower boot time; each encrypted LV must be unlocked seperately
#LUKS on software RAID uses dm-crypt only after RAID is setup.	▪ Analogous to LUKS on LVM	▪ Analogous to LUKS on LVM
#Plain dm-crypt uses dm-crypt plain mode, i.e. without a LUKS header and its options for multiple keys. This scenario also employs USB devices for `/boot` and key storage, which may be applied to the other scenarios.	▪ Data resilience for cases where a LUKS header may be damaged ▪ Allows Full Disk Encryption ▪ Helps addressing problems with SSDs	▪ High care to all encryption parameters is required ▪ Single encryption key and no option to change it
#Encrypted boot partition (GRUB) shows how to encrypt the boot partition using the GRUB bootloader. This scenario also employs an EFI system partition, which may be applied to the other scenarios.	▪ Same advantages as the scenario the installation is based on (LVM on LUKS for this particular example) ▪ Less data is left unencrypted, i.e. the boot loader and the EFI system partition, if present	▪ Same disadvantages as the scenario the installation is based on (LVM on LUKS for this particular example) ▪ More complicated configuration ▪ Not supported by other boot loaders
#Root on ZFS		

While all above scenarios provide much greater protection from outside threats than encrypted secondary file systems, they also share a common disadvantage: any user in possession of the encryption key is able to decrypt the entire drive, and therefore can access other users' data. If that is of concern, it is possible to use a combination of block device and stacked file system encryption and reap the advantages of both. See **Data-at-rest encryption** to plan ahead.

See **dm-crypt/Drive preparation#Partitioning** for a general overview of the partitioning strategies used in the scenarios.

Another area to consider is whether to set up an encrypted swap partition and what kind. See **dm-crypt/Swap encryption** for alternatives.

If you anticipate to protect the system's data not only against physical theft, but also have a requirement of precautions against logical tampering, see **dm-crypt/Specialties#Securing the unencrypted boot partition** for further possibilities after following one of the scenarios.

For **solid state drives** you might want to consider enabling TRIM support, but be warned, there are potential security implications. See **dm-crypt/Specialties#Discard/TRIM support for solid state drives (SSD)** for more information.

Warning:

- In any scenario, never use file system repair software such as **fsck** directly on an encrypted volume, or it will destroy any means to recover the key used to decrypt your files. Such tools must be used on the decrypted (opened) device instead.
- For the LUKS2 format:
  - GRUB's support for LUKS2 is limited; see **GRUB#Encrypted /boot** for details. Use LUKS1 (`cryptsetup luksFormat --type luks1`) for partitions that GRUB will need to unlock.
  - The LUKS2 format has a high RAM usage per design, defaulting to 1GB per encrypted mapper. Machines with low RAM and/or multiple LUKS2 partitions unlocked in parallel may error on boot. See the `--pbkdf-memory` option to control memory usage.[1] (https://gitlab.com/cryptsetup/cryptsetup/issues/372)

2 LUKS on a partition

This example covers a full system encryption with *dm-crypt* + LUKS in a simple partition layout:

```
+-----------------------+------------------------+-----------------------+
| Boot partition        | LUKS2 encrypted root   | Optional free space   |
|                       | partition              | for additional        |
|                       |                        | partitions to be set  |
| /boot                 | /                      | up later              |
|                       |                        |                       |
|                       | /dev/mapper/root       |                       |
|                       |------------------------|                       |
| /dev/sda1             | /dev/sda2              |                       |
+-----------------------+------------------------+-----------------------+
```

The first steps can be performed directly after booting the Arch Linux install image.

2.1 Preparing the disk

Prior to creating any partitions, you should inform yourself about the importance and methods to securely erase the disk, described in **dm-crypt/Drive preparation**.

Then create the needed partitions, at least one for `/` (e.g. `/dev/sda2`) and `/boot` (`/dev/sda1`). See **Partitioning**.

2.2 Preparing non-boot partitions

This and the next section replace the instructions of **Installation guide#Format the partitions**.

The following commands create and mount the encrypted root partition. They correspond to the procedure described in detail in **dm-crypt/Encrypting a non-root file system#Partition** (which, despite the title, *can* be applied to root partitions, as long as **mkinitcpio** and the **boot loader** are correctly configured). If you want to use particular non-default encryption options (e.g. cipher, key length), see the **encryption options** before executing the first command. For information on changing the default sector size, see **dm-crypt/Device encryption#Sector size**.

```
# cryptsetup -y -v luksFormat /dev/sda2
# cryptsetup open /dev/sda2 root
# mkfs.ext4 /dev/mapper/root
# mount /dev/mapper/root /mnt
```

Check the mapping works as intended:

```
# umount /mnt
# cryptsetup close root
# cryptsetup open /dev/sda2 root
# mount /dev/mapper/root /mnt
```

If you created separate partitions (e.g. `/home`), these steps have to be adapted and repeated for all of them, *except* for `/boot`. See **dm-crypt/Encrypting a non-root file system#Automated unlocking and mounting** on how to handle additional partitions at boot.

Note that each block device requires its own passphrase. This may be inconvenient, because it results in a separate passphrase to be input during boot. An alternative is to use a keyfile stored in the root partition to unlock the separate partition via `crypttab`. See **dm-crypt/Device encryption#Using LUKS to format partitions with a keyfile** for instructions.

2.3 Preparing the boot partition

What you do have to setup is a non-encrypted `/boot` partition, which is needed for an encrypted root. For an ordinary boot partition on BIOS systems, for example, execute:

```
# mkfs.ext4 /dev/sda1
```

or for an **EFI system partition** on UEFI systems:

```
# mkfs.fat -F32 /dev/sda1
```

Afterwards create the directory for the mountpoint and mount the partition:

```
# mount --mkdir /dev/sda1 /mnt/boot
```

2.4 Mounting the devices

At **Installation guide#Mount the file systems** you will have to mount the mapped devices, not the actual partitions. Of course `/boot`, which is not encrypted, will still have to be mounted directly. It should be mounted at `/mnt`.

2.5 Configuring mkinitcpio

Before following **Installation guide#Initramfs** you must do the following to your new system:

If using the default busybox-based initramfs, add the `keyboard` and `encrypt` hooks to **mkinitcpio.conf**. If you use a non-US console keymap or a non-default console font, additionally add the `keymap` and `consolefont` hooks, respectively.

```
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block encrypt filesystems fsck)
```

If using a systemd-based initramfs, instead add the `keyboard` and `sd-encrypt` hooks. If you use a non-US console keymap or a non-default console font, additionally add the `sd-vconsole` hook.

```
HOOKS=(base systemd autodetect modconf kms keyboard sd-vconsole block sd-encrypt filesystems fsck)
```

Regenerate the initramfs after saving the changes. See **dm-crypt/System configuration#mkinitcpio** for details and other hooks that you may need.

2.6 Configuring the boot loader

In order to unlock the encrypted root partition at boot, the following **kernel parameters** need to be set by the boot loader:

```
cryptdevice=UUID=device-UUID:root root=/dev/mapper/root
```

If using the **sd-encrypt** hook, the following need to be set instead:

```
rd.luks.name=device-UUID=root root=/dev/mapper/root
```

The *`device-UUID`* refers to the UUID of the LUKS superblock, in this case `/dev/sda2`. See **Persistent block device naming** for details.

Also see **dm-crypt/System configuration#Kernel parameters** for more details.

Tip: If the root partition is on the same disk as the `/boot` partition and your UEFI boot loader supports **GPT partition automounting**, you can configure the **partition type GUID** (type should be "Root partition", not "LUKS partition") and rely on **`systemd-gpt-auto-`**

generator(8) (https://man.archlinux.org/man/systemd-gpt-auto-generator.8) instead of using the kernel parameters.

3 LUKS on a partition with TPM2 and Secure Boot

This example is similar to **#LUKS on a partition**, but integrates the use of **Secure Boot** and a **Trusted Platform Module** (TPM), enhancing the overall security of the boot process.

In this configuration, only the **EFI system partition** remains unencrypted, housing a **unified kernel image** and **systemd-boot**—both signed for use with Secure Boot. The TPM plays a crucial role by monitoring the Secure Boot state. If Secure Boot is disabled or its key databases are tampered with, the encrypted partition will not unlock. This approach is akin to BitLocker on Windows or FileVault on macOS. A recovery-key will also be created to make sure the data remains accessible in case of a problem with the TPM unlocking mechanism (unsigned bootloader or kernel update, firmware update, etc.)

Warning: Implementing this method on your root volume means your computer will **automatically unlock** at boot without requiring an encryption password under specific conditions. However, be cautious of the following risks:

- On systems with multiple users, without additional protection, data is accessible between users.
- Weak user passwords increase the risk of data exposure in case of theft.
- The setup is susceptible to **cold boot attacks**, where an attacker, even after a prolonged power-off period, could exploit the automatic loading of the TPM key upon turning on the computer. This is particularly relevant for high-value targets.

In this example, partitions are created respecting **systemd#GPT partition automounting**, there is no need for an fstab or crypttab file.

```
+-----------------------+------------------------------------------------+
| ESP                   | Root partition                                 |
| unencrypted           | encrypted                                      |
|                       |                                                |
| /efi                  | /                                              |
|                       |                                                |
|                       | /dev/mapper/root                               |
|                       |------------------------------------------------|
| /dev/sda1             | /dev/sda2                                      |
+-----------------------+------------------------------------------------+
```

Follow the **Installation guide** up to step **Installation guide#Partition the disks**.

3.1 Preparing the disk

Prior to creating any partitions, you should inform yourself about the importance and methods to securely erase the disk, described in **dm-crypt/Drive preparation**.

Partition the drive with the **GUID Partition Table** (GPT). Then create the needed partitions.

Create an **EFI system partition** (`/dev/sda1` in this example) with an appropriate size. It will later be mounted at `/efi` .

In the remaining space on the drive create a root partition (`/dev/sda2` in this example) which will be encrypted and later mounted to `/`. Set its partition type GUID to `BC13C2FF-59E6-4262-A352-B275FD6F7172` ("Linux root (x86-64)" in **fdisk**, `8304` in **gdisk**).

3.2 Preparing the root partition

The following commands create and mount the encrypted root partition. See **dm-crypt/Encrypting a non-root file system#Partition** (which, despite the title, *can* be applied to root partitions, as long as **mkinitcpio** and the **boot loader** are correctly configured).

If you want to use particular non-default encryption options (e.g. cipher, key length), or if you don't want to use TPM based decryption, see the **encryption options** before executing the first command.

Create the luks volume (you can simply use a blank password, as it will be wiped later) and mount it:

```
# cryptsetup luksFormat /dev/sda2
# cryptsetup open /dev/sda2 root
# mkfs.ext4 /dev/mapper/root
# mount /dev/mapper/root /mnt
```

3.3 Preparing the EFI system partition

Format the newly created EFI system partition as instructed in **EFI system partition#Format the partition** and mount it afterwards.

```
# mount --mkdir /dev/sda1 /mnt/efi
```

Continue the installation process until **Installation guide#Initramfs**. You can skip **Installation guide#Fstab**.

3.4 Configuring mkinitcpio

To build a working systemd based initramfs, modify the `HOOKS=` line in **mkinitcpio.conf** as follows:

```
HOOKS=(systemd autodetect modconf kms keyboard sd-vconsole block sd-encrypt filesystems fsck)
```

Next, see **Unified kernel image#mkinitcpio** to configure mkinitcpio for **Unified kernel images**.

Do **not** regenerate the initramfs **yet**, as the `/efi/EFI/Linux` directory needs to be created by the boot loader installer first.

3.5 Installing the boot loader

Install **systemd-boot** with:

```
# bootctl install
```

The **Unified kernel image** generated by mkinitcpio will be automatically recognized and does not need an entry in `/efi/loader/entries/` .

See **systemd-boot#Updating the UEFI boot manager** and **systemd-boot#Loader configuration** for further configuration.

3.6 Finalizing the installation

First, **Regenerate the initramfs**, and make sure the image generation is successful.

Make sure you did not forget to **set a root password**, **reboot** to finish the installation.

3.7 Secure Boot

You can now sign the boot loader executables and the EFI binary, in order to enable **Secure Boot**. For a quick and easy way, see **Unified Extensible Firmware Interface/Secure Boot#Assisted process with sbctl**.

3.8 Enrolling the TPM

After signing the boot loader executables and enabling Secure Boot, you can now enroll the TPM in order to use it to unlock the LUKS volume. The following commands will remove the empty passphrase created during the LUKS format process, create a key bound to the TPM **PCR 7** (default, **Secure Boot** state and firmware certificates) and create a recovery key to be used in case of any problems. The TPM will automatically release the key as long as the boot chain is not tampered with. See **systemd-cryptenroll#Trusted Platform Module** and **`systemd-cryptenroll(1)` (https://man.archlinux.org/man/systemd-cryptenroll.1)**.

```
# systemd-cryptenroll /dev/sda2 --recovery-key
# systemd-cryptenroll /dev/sda2 --wipe-slot=empty --tpm2-device=auto
```

Warning:

- Make sure **Secure Boot** is active and in user mode when binding to PCR 7, otherwise, unauthorized boot devices could unlock the encrypted volume.
- The state of PCR 7 can change if firmware certificates change, which can risk locking the user out. This can be implicitly done by **fwupd[2] (https://raw.githubusercontent.com/systemd/systemd/ed272a9ff59a26beedaab508dd3c9d631de67165/TODO)** or explicitly by rotating Secure Boot keys.

4 LVM on LUKS

The straightforward method is to set up **LVM** on top of the encrypted partition instead of the other way round. Technically the LVM is setup inside one big encrypted block device. Hence, the LVM is not visible until the block device is unlocked and the underlying volume structure is scanned and mounted during boot.

The disk layout in this example is:

```
+-----------------------------------------------------------------------+ +----------------+
| Logical volume 1      | Logical volume 2      | Logical volume 3      | | Boot partition |
|                       |                       |                       | |                |
| [SWAP]                | /                     | /home                 | | /boot          |
|                       |                       |                       | |                |
| /dev/MyVolGroup/swap  | /dev/MyVolGroup/root  | /dev/MyVolGroup/home  | |                |
|_ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _| | (may be on     |
|                                                                       | | other device)  |
|                         LUKS2 encrypted partition                     | |                |
|                           /dev/sda1                                   | | /dev/sdb1      |
+-----------------------------------------------------------------------+ +----------------+
```

Note: While using the `encrypt` hook this method does not allow you to span the logical volumes over multiple disks; either use the sd-encrypt or see dm-crypt/Specialties#Modifying the encrypt hook for multiple partitions.

Tip: Two variants of this setup:

- Instructions at dm-crypt/Specialties#Encrypted system using a detached LUKS header use this setup with a detached LUKS header on a USB device to achieve a two factor authentication with it.
- Instructions at dm-crypt/Specialties#Encrypted /boot and a detached LUKS header on USB use this setup with a detached LUKS header, encrypted `/boot` partition, and encrypted keyfile all on a USB device.

4.1 Preparing the disk

Prior to creating any partitions, you should inform yourself about the importance and methods to securely erase the disk, described in dm-crypt/Drive preparation.

Tip: When using the GRUB boot loader for BIOS booting from a GPT disk, create a BIOS boot partition.

Create a partition to be mounted at `/boot` with a size of 200 MiB or more.

Tip: UEFI systems can use the EFI system partition for `/boot`.

Create a partition which will later contain the encrypted container.

Create the LUKS encrypted container at the designated partition. Enter the chosen password twice.

```
# cryptsetup luksFormat /dev/sda1
```

For more information about the available cryptsetup options see the LUKS encryption options prior to above command.

Open the container:

```
# cryptsetup open /dev/sda1 cryptlvm
```

The decrypted container is now available at `/dev/mapper/cryptlvm`.

4.2 Preparing the logical volumes

Create a physical volume on top of the opened LUKS container:

```
# pvcreate /dev/mapper/cryptlvm
```

Create a volume group (in this example named `MyVolGroup`, but it can be whatever you want) and add the previously created physical volume to it:

```
# vgcreate MyVolGroup /dev/mapper/cryptlvm
```

Create all your logical volumes on the volume group:

Tip: If a logical volume will be formatted with ext4, leave at least 256 MiB free space in the volume group to allow using `e2scrub(8) (https://man.archlinux.org/man/e2scrub.8)`. After creating the last volume with `-l 100%FREE`, this can be accomplished by reducing its size with `lvreduce -L -256M MyVolGroup/home`.

```
# lvcreate -L 8G MyVolGroup -n swap
# lvcreate -L 32G MyVolGroup -n root
# lvcreate -l 100%FREE MyVolGroup -n home
```

Format your file systems on each logical volume:

```
# mkfs.ext4 /dev/MyVolGroup/root
# mkfs.ext4 /dev/MyVolGroup/home
# mkswap /dev/MyVolGroup/swap
```

Mount your file systems:

```
# mount /dev/MyVolGroup/root /mnt
# mount --mkdir /dev/MyVolGroup/home /mnt/home
# swapon /dev/MyVolGroup/swap
```

4.3 Preparing the boot partition

The bootloader loads the kernel, initramfs, and its own configuration files from the `/boot` directory. Any file system on a disk that can be read by the bootloader is eligible.

Create a file system on the partition intended for `/boot`:

```
# mkfs.ext4 /dev/sdb1
```

Tip: When opting to keep `/boot` on an EFI system partition the recommended formatting is

```
# mkfs.fat -F32 /dev/sdb1
```

Mount the partition to `/mnt/boot` :

```
# mount --mkdir /dev/sdb1 /mnt/boot
```

At this point resume the common **Installation guide#Installation** steps. Return to this page to customize the **Initramfs** and **Boot loader** steps.

4.4 Configuring mkinitcpio

Make sure the **lvm2 (https://archlinux.org/packages/?name=lvm2)** package is **installed**.

If using the default busybox-based initramfs, add the `keyboard` , `encrypt` and `lvm2` hooks to **mkinitcpio.conf**. If you use a non-US console keymap or a non-default console font, additionally add the `keymap` and `consolefont` hooks, respectively.

```
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block encrypt lvm2 filesystems fsck)
```

If using a systemd-based initramfs, instead add the `keyboard` , `sd-encrypt` and `lvm2` hooks. if you use a non-US console keymap or a non-default console font, additionally add the `sd-vconsole` hook.

```
HOOKS=(base systemd autodetect modconf kms keyboard sd-vconsole block sd-encrypt lvm2 filesystems fsck)
```

Regenerate the initramfs after saving the changes. See **dm-crypt/System configuration#mkinitcpio** for details and other hooks that you may need.

4.5 Configuring the boot loader

In order to unlock the encrypted root partition at boot, the following **kernel parameters** need to be set by the boot loader:

```
cryptdevice=UUID=device-UUID:cryptlvm root=/dev/MyVolGroup/root
```

If using the **sd-encrypt** hook, the following needs to be set instead:

```
rd.luks.name=device-UUID=cryptlvm root=/dev/MyVolGroup/root
```

The *`device-UUID`* refers to the UUID of the LUKS superblock, in this case `/dev/sda1` . See **Persistent block device naming** for details.

If using **dracut**, you may need a more extensive list of parameters, try:

```
kernel_cmdline="rd.luks.uuid=luks-deviceUUID rd.lvm.lv=MyVolGroup/root  rd.lvm.lv=MyVolGroup/swap  root=/
dev/mapper/MyVolGroup-root rootfstype=ext4 rootflags=rw,relatime"
```

See **dm-crypt/System configuration#Kernel parameters** for details.

5 LUKS on LVM

To use encryption on top of **LVM**, the LVM volumes are set up first and then used as the base for the encrypted partitions. This way, a mixture of encrypted and non-encrypted volumes/partitions is possible as well.

Tip: Unlike **#LVM on LUKS**, this method allows normally spanning the logical volumes over multiple disks.

The following short example creates a LUKS on LVM setup and mixes in the use of a key-file for the /home partition and temporary crypt volumes for `/tmp` and `/swap`. The latter is considered desirable from a security perspective, because no potentially sensitive temporary data survives the reboot, when the encryption is re-initialised. If you are experienced with LVM, you will be able to ignore/replace LVM and other specifics according to your plan.

If you want to span a logical volume over multiple disks that have already been set up, or expand the logical volume for `/home` (or any other volume), a procedure to do so is described in **dm-crypt/Specialties#Expanding LVM on multiple disks**. It is important to note that the LUKS encrypted container has to be resized as well.

5.1 Preparing the disk

Partitioning scheme:

```
+----------------+------------------------------------------------------------------------------------------
| Boot partition | dm-crypt plain encrypted volume | LUKS2 encrypted volume          | LUKS2 encrypted volume
|                |                                 |                                 |
| /boot          | [SWAP]                          | /                               | /home
|                |                                 |                                 |
|                | /dev/mapper/swap                | /dev/mapper/root                | /dev/mapper/home
|                |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _
|                | Logical volume 1                | Logical volume 2                | Logical volume 3
|                | /dev/MyVolGroup/cryptswap       | /dev/MyVolGroup/cryptroot       | /dev/MyVolGroup/crypth
|                |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _ _ _ _ _
|                |
|   /dev/sda1    |                                   /dev/sda2
+----------------+------------------------------------------------------------------------------------------
```

Randomise `/dev/sda2` according to **dm-crypt/Drive preparation#dm-crypt wipe on an empty disk or partition**.

5.2 Preparing the logical volumes

```
# pvcreate /dev/sda2
# vgcreate MyVolGroup /dev/sda2
# lvcreate -L 32G -n cryptroot MyVolGroup
# lvcreate -L 500M -n cryptswap MyVolGroup
# lvcreate -L 500M -n crypttmp MyVolGroup
# lvcreate -l 100%FREE -n crypthome MyVolGroup
```

```
# cryptsetup luksFormat /dev/MyVolGroup/cryptroot
# cryptsetup open /dev/MyVolGroup/cryptroot root
# mkfs.ext4 /dev/mapper/root
# mount /dev/mapper/root /mnt
```

More information about the encryption options can be found in **dm-crypt/Device encryption#Encryption options for LUKS mode**. Note that `/home` will be encrypted in **#Encrypting logical volume /home**.

Tip: If you ever have to access the encrypted root from the Arch-ISO, the above `open` action will allow you to after the **LVM shows up**.

5.3 Preparing the boot partition

```
# dd if=/dev/zero of=/dev/sda1 bs=1M status=progress
# mkfs.ext4 /dev/sda1
# mount --mkdir /dev/sda1 /mnt/boot
```

5.4 Configuring mkinitcpio

Make sure the **lvm2 (https://archlinux.org/packages/?name=lvm2)** package is **installed**.

If using the default busybox-based initramfs, add the `keyboard`, `encrypt` and `lvm2` hooks to **mkinitcpio.conf**. If you use a non-US console keymap or a non-default console font, additionally add the `keymap` and `consolefont` hooks, respectively.

```
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block lvm2 encrypt filesystems fsck)
```

If using a systemd-based initramfs, instead add the `keyboard`, `sd-encrypt` and `lvm2` hooks. if you use a non-US console keymap or a non-default console font, additionally add the `sd-vconsole` hook.

```
HOOKS=(base systemd autodetect modconf kms keyboard sd-vconsole block sd-encrypt lvm2 filesystems fsck)
```

Regenerate the initramfs after saving the changes. See **dm-crypt/System configuration#mkinitcpio** for details and other hooks that you may need.

5.5 Configuring the boot loader

In order to unlock the encrypted root partition at boot, the following **kernel parameters** need to be set by the boot loader:

```
cryptdevice=UUID=device-UUID:root root=/dev/mapper/root
```

If using the **sd-encrypt** hook, the following need to be set instead:

```
rd.luks.name=device-UUID=root root=/dev/mapper/root
```

The *`device-UUID`* refers to the UUID of the LUKS superblock, in this case `/dev/MyVolGroup/cryptroot`. See **Persistent block device naming** for details.

See **dm-crypt/System configuration#Kernel parameters** for details.

5.6 Configuring fstab and crypttab

Both **crypttab** and **fstab** entries are required to both unlock the device and mount the file systems, respectively. The following lines will re-encrypt the temporary file systems on each reboot:

```
/etc/crypttab
```

```
swap     /dev/MyVolGroup/cryptswap       /dev/urandom    swap,cipher=aes-xts-plain64,size=256
tmp      /dev/MyVolGroup/crypttmp        /dev/urandom    tmp,cipher=aes-xts-plain64,size=256
```

```
/etc/fstab
```

```
/dev/mapper/root        /       ext4            defaults        0       1
/dev/sda1               /boot   ext4            defaults        0       2
/dev/mapper/tmp         /tmp    tmpfs           defaults        0       0
/dev/mapper/swap        none    swap            sw              0       0
```

5.7 Encrypting logical volume /home

Since this scenario uses LVM as the primary and dm-crypt as secondary mapper, each encrypted logical volume requires its own encryption. Yet, unlike the temporary file systems configured with volatile encryption above, the logical volume for `/home` should of course be persistent. The following assumes you have rebooted into the installed system, otherwise you have to adjust paths. To save on entering a second passphrase at boot, a **keyfile** is created:

```
# mkdir -m 700 /etc/luks-keys
# dd if=/dev/random of=/etc/luks-keys/home bs=1 count=256 status=progress
```

The logical volume is encrypted with it:

```
# cryptsetup luksFormat -v /dev/MyVolGroup/crypthome /etc/luks-keys/home
# cryptsetup -d /etc/luks-keys/home open /dev/MyVolGroup/crypthome home
# mkfs.ext4 /dev/mapper/home
# mount /dev/mapper/home /home
```

The encrypted mount is configured in both **crypttab** and **fstab**:

```
/etc/crypttab
```

```
home     /dev/MyVolGroup/crypthome    /etc/luks-keys/home
```

```
/etc/fstab
```

```
/dev/mapper/home        /home   ext4        defaults        0       2
```

6 LUKS on software RAID

This example is based on a real-world setup for a workstation class laptop equipped with two SSDs of equal size, and an additional HDD for bulk storage. The end result is LUKS1 based full disk encryption (including `/boot`) for all drives, with the SSDs in a **RAID0** array, and keyfiles used to unlock all encryption after **GRUB** is given a correct passphrase at boot.

This setup utilizes a very simplistic partitioning scheme, with all the available RAID storage being mounted at `/` (no separate `/boot` partition), and the decrypted HDD being mounted at `/data` .

Please note that regular **backups** are very important in this setup. If either of the SSDs fail, the data contained in the RAID array will be practically impossible to recover. You may wish to select a different **RAID level** if fault tolerance is important to you.

The encryption is not deniable in this setup.

For the sake of the instructions below, the following block devices are used:

```
/dev/sda = first SSD
/dev/sdb = second SSD
/dev/sdc = HDD
```

```
+---------------------+---------------------------+---------------------------+ +---------------------+----
| BIOS boot partition | EFI system partition      | LUKS1 encrypted volume    | | BIOS boot partition | EFI
|                     |                           |                           | |                     |
|                     | /efi                      | /                         | |                     | /ef
|                     |                           |                           | |                     |
|                     |                           | /dev/mapper/root          | |                     |
|                     +---------------------------+---------------------------+ |                     +----
|                     | RAID1 array (part 1 of 2) | RAID0 array (part 1 of 2) | |                     | RAI
|                     |                           |                           | |                     |
|                     | /dev/md/ESP               | /dev/md/root              | |                     | /de
|                     +---------------------------+---------------------------+ |                     +----
| /dev/sda1           | /dev/sda2                 | /dev/sda3                 | | /dev/sdb1           | /de
+---------------------+---------------------------+---------------------------+ +---------------------+----
```

Be sure to substitute them with the appropriate device designations for your setup, as they may be different.

6.1 Preparing the disks

Prior to creating any partitions, you should inform yourself about the importance and methods to securely erase the disk, described in **dm-crypt/Drive preparation**.

For **BIOS systems** with GPT, create a **BIOS boot partition** with size of 1 MiB for GRUB to store the second stage of BIOS bootloader. Do not mount the partition.

For **UEFI systems** create an **EFI system partition** with an appropriate size, it will later be mounted at `/efi` .

In the remaining space on the drive create a partition (`/dev/sda3` in this example) for "Linux RAID". Choose partition type ID `fd` for MBR or partition type GUID `A19D880F-05FC-4D3B-A006-743F0F84911E` for GPT.

Once partitions have been created on `/dev/sda` , the following commands can be used to clone them to `/dev/sdb` .

```
# sfdisk -d /dev/sda > sda.dump
# sfdisk /dev/sdb < sda.dump
```

The HDD is prepared with a single Linux partition covering the whole drive at `/dev/sdc1`.

6.2 Building the RAID array

Create the RAID array for the SSDs.

Note:

- All parts of an EFI system partition RAID array must be individually usable, that means that ESP can only placed in a RAID1 array.
- The RAID superblock must be placed at the end of the EFI system partition using `--metadata=1.0`, otherwise the firmware will not be able to access the partition.

```
# mdadm --create --verbose --level=1 --metadata=1.0 --raid-devices=2 /dev/md/ESP /dev/sda2 /dev/sdb2
```

This example utilizes RAID0 for root, you may wish to substitute a different level based on your preferences or requirements.

```
# mdadm --create --verbose --level=0 --metadata=1.2 --raid-devices=2 /dev/md/root /dev/sda3 /dev/sdb3
```

6.3 Preparing the block devices

As explained in **dm-crypt/Drive preparation**, the devices are wiped with random data utilizing `/dev/zero` and a crypt device with a random key. Alternatively, you could use `dd` with `/dev/random` or `/dev/urandom`, though it will be much slower.

```
# cryptsetup open --type plain /dev/md/root container --key-file /dev/random
# dd if=/dev/zero of=/dev/mapper/container bs=1M status=progress
# cryptsetup close container
```

And repeat above for the HDD (`/dev/sdc1` in this example).

Set up encryption for `/dev/md/root`:

Warning: GRUB's support for LUKS2 is limited; see **GRUB#Encrypted /boot** for details. Use LUKS1 (`cryptsetup luksFormat --type luks1`) for partitions that GRUB will need to unlock.

```
# cryptsetup -y -v luksFormat --type luks1 /dev/md/root
# cryptsetup open /dev/md/root root
# mkfs.ext4 /dev/mapper/root
# mount /dev/mapper/root /mnt
```

And repeat for the HDD:

```
# cryptsetup -y -v luksFormat /dev/sdc1
# cryptsetup open /dev/sdc1 data
# mkfs.ext4 /dev/mapper/data
# mount --mkdir /dev/mapper/data /mnt/data
```

For UEFI systems, set up the EFI system partition:

```
# mkfs.fat -F32 /dev/md/ESP
# mount --mkdir /dev/md/ESP /mnt/efi
```

6.4 Configuring GRUB

Configure **GRUB** for the LUKS1 encrypted system by editing `/etc/default/grub` with the following:

```
GRUB_CMDLINE_LINUX="cryptdevice=/dev/md/root:root"
GRUB_ENABLE_CRYPTODISK=y
```

If you have a USB keyboard on a newer system either enable legacy USB support in firmware or add the following to `/etc/default/grub` :

```
GRUB_TERMINAL_INPUT="usb_keyboard"
GRUB_PRELOAD_MODULES="usb usb_keyboard ohci uhci ehci"
```

Otherwise you may not be able to use your keyboard at the LUKS prompt.

See **dm-crypt/System configuration#Kernel parameters** and **GRUB#Encrypted /boot** for details.

Complete the GRUB install to both SSDs (in reality, installing only to `/dev/sda` will work).

```
# grub-install --target=i386-pc /dev/sda
# grub-install --target=i386-pc /dev/sdb
# grub-install --target=x86_64-efi --efi-directory=/efi --bootloader-id=GRUB
# grub-mkconfig -o /boot/grub/grub.cfg
```

6.5 Creating the keyfiles

The next steps save you from entering your passphrase twice when you boot the system (once so GRUB can unlock the LUKS1 device, and second time once the initramfs assumes control of the system). This is done by creating a **keyfile** for the encryption and adding it to the initramfs image to allow the encrypt hook to unlock the root device. See **dm-crypt/Device encryption#With a keyfile embedded in the initramfs** for details.

- Create the **keyfile** and add the key to `/dev/md/root` .
- Create another keyfile for the HDD (`/dev/sdc1`) so it can also be unlocked at boot. For convenience, leave the passphrase created above in place as this can make recovery easier if you ever need it. Edit `/etc/crypttab` to decrypt the HDD at boot. See **dm-crypt/System configuration#Unlocking with a keyfile**.

6.6 Configuring the system

Edit **fstab** to mount the root and data block devices and the ESP:

```
/dev/mapper/root  /       ext4  rw,noatime         0 1
/dev/mapper/data  /data   ext4  defaults           0 2
/dev/md/ESP       /efi     vfat rw,relatime,codepage=437,iocharset=iso8859-1,shortname=mixed,utf8,tz=UTC,
errors=remount-ro       0 2
```

Save the RAID configuration:

```
# mdadm --detail --scan >> /etc/mdadm.conf
```

Edit **mkinitcpio.conf** to include your keyfile and add the proper hooks:

```
FILES=(/crypto_keyfile.bin)
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block mdadm_udev encrypt filesystems
fsck)
```

See **dm-crypt/System configuration#mkinitcpio** for details.

7 Plain dm-crypt

Contrary to LUKS, dm-crypt *plain* mode does not require a header on the encrypted device: this scenario exploits this feature to set up a system on an unpartitioned, encrypted disk that will be indistinguishable from a disk filled with random data, which could allow **deniable encryption**. See also **wikipedia:Disk encryption#Full disk encryption**.

Note that if full-disk encryption is not required, the methods using LUKS described in the sections above are better options for both system encryption and encrypted partitions. LUKS features like key management with multiple passphrases/key-files or re-encrypting a device in-place are unavailable with *plain* mode.

Plain dm-crypt encryption can be more resilient to damage than LUKS, because it does not rely on an encryption master-key which can be a single-point of failure if damaged. However, using *plain* mode also requires more manual configuration of encryption options to achieve the same cryptographic strength. See also **Data-at-rest encryption#Cryptographic metadata**. Using *plain* mode could also be considered if concerned with the problems explained in **dm-crypt/Specialties#Discard/TRIM support for solid state drives (SSD)**.

Tip: If headerless encryption is your goal but you are unsure about the lack of key-derivation with *plain* mode, then two alternatives are:

- **dm-crypt LUKS mode with a detached header** by using the *cryptsetup* `--header` option. It cannot be used with the standard *encrypt* hook, but the hook may be modified.
- **tcplay** which offers headerless encryption but with the PBKDF2 function.

The scenario uses two USB sticks:

- one for the boot device, which also allows storing the options required to open/unlock the plain encrypted device in the boot loader configuration, since typing them on each boot would be error prone;
- another for the encryption key file, assuming it stored as raw bits so that to the eyes of an unaware attacker who might get the usbkey the encryption key will appear as random data

instead of being visible as a normal file. See also Wikipedia:Security through obscurity, follow dm-crypt/Device encryption#Keyfiles to prepare the keyfile.

The disk layout is:

```
+----------------------+----------------------+----------------------+ +----------------+ +----------------
| Logical volume 1     | Logical volume 2     | Logical volume 3     | | Boot device    | | Encryption key
|                      |                      |                      | |                | | file storage
| /                    | [SWAP]               | /home                | | /boot          | | (unpartitioned
|                      |                      |                      | |                | | in example)
| /dev/MyVolGroup/root | /dev/MyVolGroup/swap | /dev/MyVolGroup/home | | /dev/sdb1      | | /dev/sdc
|----------------------+----------------------+----------------------| |----------------| |----------------
| disk drive /dev/sda encrypted using plain mode and LVM             | | USB stick 1    | | USB stick 2
+--------------------------------------------------------------------+ +----------------+ +----------------
```

Tip:

- It is also possible to use a single USB key physical device:
  - By putting the key on another partition (/dev/sdb2) of the USB storage device (/dev/sdb).
  - By copying the keyfile to the initramfs directly. An example keyfile `/etc/keyfile` gets copied to the initramfs image by setting `FILES=(/etc/keyfile)` in `/etc/mkinitcpio.conf`. The way to instruct the `encrypt` hook to read the keyfile in the initramfs image is using `rootfs:` prefix before the filename, e.g. `cryptkey=rootfs:/etc/keyfile`.
- Another option is using a passphrase with good entropy.

7.1 Preparing the disk

It is vital that the mapped device is filled with random data. In particular this applies to the scenario use case we apply here.

See dm-crypt/Drive preparation and dm-crypt/Drive preparation#dm-crypt specific methods

7.2 Preparing the non-boot partitions

See dm-crypt/Device encryption#Encryption options for plain mode for details.

Using the device `/dev/sda`, with the aes-xts cipher with a 512 bit key size and using a keyfile we have the following options for this scenario:

```
# cryptsetup --cipher=aes-xts-plain64 --offset=0 --key-file=/dev/sdc --key-size=512 open --type plain /de
v/sda cryptlvm
```

Unlike encrypting with LUKS, the above command must be executed *in full* whenever the mapping needs to be re-established, so it is important to remember the cipher, and key file details.

We can now check a mapping entry has been made for `/dev/mapper/cryptlvm`:

```
# fdisk -l
```

Tip: A simpler alternative to using LVM, advocated in the cryptsetup FAQ for cases where LVM is not necessary, is to just create a file system on the entirety of the mapped dm-crypt device.

Next, we setup **LVM** logical volumes on the mapped device. See **Install Arch Linux on LVM** for further details:

```
# pvcreate /dev/mapper/cryptlvm
# vgcreate MyVolGroup /dev/mapper/cryptlvm
# lvcreate -L 32G MyVolGroup -n root
# lvcreate -L 10G MyVolGroup -n swap
# lvcreate -l 100%FREE MyVolGroup -n home
```

We format and mount them and activate swap. See **File systems#Create a file system** for further details:

```
# mkfs.ext4 /dev/MyVolGroup/root
# mkfs.ext4 /dev/MyVolGroup/home
# mount /dev/MyVolGroup/root /mnt
# mount --mkdir /dev/MyVolGroup/home /mnt/home
# mkswap /dev/MyVolGroup/swap
# swapon /dev/MyVolGroup/swap
```

7.3 Preparing the boot partition

The `/boot` partition can be installed on the standard vfat partition of a USB stick, if required. But if manual partitioning is needed, then a small 200 MiB partition is all that is required. Create the partition using a **partitioning tool** of your choice.

Create a **file system** on the partition intended for `/boot`:

```
# mkfs.fat -F32 /dev/sdb1
# mount --mkdir /dev/sdb1 /mnt/boot
```

7.4 Configuring mkinitcpio

Make sure the **lvm2 (https://archlinux.org/packages/?name=lvm2)** package is **installed**.

If using the default busybox-based initramfs, add the `keyboard`, `encrypt` and `lvm2` hooks to **mkinitcpio.conf**. If you use a non-US console keymap or a non-default console font, additionally add the `keymap` and `consolefont` hooks, respectively.

```
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block encrypt lvm2 filesystems fsck)
```

Regenerate the initramfs after saving the changes. See **dm-crypt/System configuration#mkinitcpio** for details and other hooks that you may need.

7.5 Configuring the boot loader

In order to boot the encrypted root partition, the following **kernel parameters** need to be set by the boot loader (note that 64 is the number of bytes in 512 bits):

```
cryptdevice=/dev/disk/by-id/disk-ID-of-sda:cryptlvm cryptkey=/dev/disk/by-id/disk-ID-of-sdc:0:64 crypto=:
aes-xts-plain64:512:0:
```

The `disk-ID-of-disk` refers to the id of the referenced disk. See **Persistent block device naming** for details.

See **dm-crypt/System configuration#Kernel parameters** for details and other parameters that you may need.

Tip: If using **GRUB**, you can install it on the same USB as the `/boot` partition.

For BIOS:

```
# grub-install --target=i386-pc --recheck /dev/sdb
```

For UEFI:

```
# grub-install --target=x86_64-efi --efi-directory=/boot --removable
```

7.6 Post-installation

You may wish to remove the USB sticks after booting. Since the `/boot` partition is not usually needed, the `noauto` option can be added to the relevant line in `/etc/fstab`:

```
/etc/fstab
```

```
# /dev/sdb1
/dev/sdb1 /boot vfat noauto,rw,noatime 0 2
```

However, when an update to anything used in the initramfs, or a kernel, or the bootloader is required; the `/boot` partition must be present and mounted. As the entry in `fstab` already exists, it can be mounted simply with:

```
# mount /boot
```

8 Encrypted boot partition (GRUB)

This setup utilizes the same partition layout and configuration as the previous **#LVM on LUKS** section, with the difference that the **GRUB** boot loader is used since it is capable of booting from an LVM logical volume and a LUKS1-encrypted `/boot`. See also **GRUB#Encrypted /boot**.

The disk layout in this example is:

```
+---------------------+----------------------+----------------------+----------------------+---------------
| BIOS boot partition | EFI system partition | Logical volume 1     | Logical volume 2     | Logical volume
|                     |                      |                      |                      |
|                     | /efi                 | /                    | [SWAP]               | /home
|                     |                      |                      |                      |
|                     |                      | /dev/MyVolGroup/root | /dev/MyVolGroup/swap | /dev/MyVolGrou
| /dev/sda1           | /dev/sda2            |----------------------+----------------------+---------------
| unencrypted          | unencrypted          | /dev/sda3 encrypted using LVM on LUKS1
+----------------------+----------------------+---------------------------------------------------------------
```

Tip:

- All scenarios are intended as examples. It is, of course, possible to apply both of the two above distinct installation steps with the other scenarios as well. See also the variants linked in #LVM on LUKS.
- You can use `cryptboot` script from `cryptboot (https://aur.archlinux.org/packages/cryptboot/)`AUR package for simplified encrypted boot management (mounting, unmounting, upgrading packages) and as a defense against Evil Maid (https://www.schneier.com/blog/archives/2009/10/evil_maid_attac.html) attacks with UEFI Secure Boot. For more information and limitations see cryptboot project (https://github.com/kmille/cryptboot) page.

8.1 Preparing the disk

Prior to creating any partitions, you should inform yourself about the importance and methods to securely erase the disk, described in dm-crypt/Drive preparation.

For BIOS/GPT systems create a BIOS boot partition with size of 1 MiB for GRUB to store the second stage of BIOS bootloader. Do not mount the partition. For BIOS/MBR systems this is not necessary.

For UEFI systems create an EFI system partition with an appropriate size, it will later be mounted at `/efi`.

Create a partition of type `8309`, which will later contain the encrypted container for the LVM.

Create the LUKS encrypted container:

Warning: GRUB's support for LUKS2 is limited; see GRUB#Encrypted /boot for details. Use LUKS1 (`cryptsetup luksFormat --type luks1`) for partitions that GRUB will need to unlock.

```
# cryptsetup luksFormat --type luks1 /dev/sda3
```

For more information about the available cryptsetup options see the LUKS encryption options prior to above command.

Your partition layout should look similar to this:

```
# gdisk -l /dev/sda
```

```
...
Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048            4095   1024.0 KiB  EF02  BIOS boot partition
   2            4096         1130495   550.0 MiB   EF00  EFI System
   3         1130496        68239360   32.0 GiB    8309  Linux LUKS
```

Open the container:

```
# cryptsetup open /dev/sda3 cryptlvm
```

The decrypted container is now available at `/dev/mapper/cryptlvm`.

8.2 Preparing the logical volumes

The LVM logical volumes of this example follow the exact layout as the **#LVM on LUKS** scenario. Therefore, please follow **#Preparing the logical volumes** above and adjust as required.

If you plan to boot in UEFI mode, create a mountpoint for the **EFI system partition** at `/efi` for compatibility with `grub-install` and mount it:

```
# mount --mkdir /dev/sda2 /mnt/efi
```

At this point, you should have the following partitions and logical volumes inside of `/mnt`:

```
$ lsblk
```

```
NAME                  MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sda                   8:0      0   200G  0 disk
├─sda1                8:1      0     1M  0 part
├─sda2                8:2      0   550M  0 part  /mnt/efi
└─sda3                8:3      0   100G  0 part
  └─cryptlvm          254:0    0   100G  0 crypt
    ├─MyVolGroup-swap 254:1    0     8G  0 lvm   [SWAP]
    ├─MyVolGroup-root 254:2    0    32G  0 lvm   /mnt
    └─MyVolGroup-home 254:3    0    60G  0 lvm   /mnt/home
```

8.3 Configuring mkinitcpio

Make sure the **`lvm2` (https://archlinux.org/packages/?name=lvm2)** package is **installed**.

If using the default busybox-based initramfs, add the `keyboard`, `encrypt` and `lvm2` hooks to **mkinitcpio.conf**. If you use a non-US console keymap or a non-default console font, additionally add the `keymap` and `consolefont` hooks, respectively.

```
HOOKS=(base udev autodetect modconf kms keyboard keymap consolefont block encrypt lvm2 filesystems fsck)
```

If using a systemd-based initramfs, instead add the `keyboard`, `sd-encrypt` and `lvm2` hooks. if you use a non-US console keymap or a non-default console font, additionally add the `sd-vconsole` hook.

```
HOOKS=(base systemd autodetect modconf kms keyboard sd-vconsole block sd-encrypt lvm2 filesystems fsck)
```

Regenerate the initramfs after saving the changes. See **dm-crypt/System configuration#mkinitcpio** for details and other hooks that you may need.

8.4 Configuring GRUB

Configure GRUB to allow booting from `/boot` on a LUKS1 encrypted partition:

```
/etc/default/grub
```

```
GRUB_ENABLE_CRYPTODISK=y
```

Set the kernel parameters, so that the initramfs can unlock the encrypted root partition. Using the `encrypt` hook:

```
/etc/default/grub
```

```
GRUB_CMDLINE_LINUX="... cryptdevice=UUID=device-UUID:cryptlvm ..."
```

If using the **sd-encrypt** hook, the following need to be set instead:

```
/etc/default/grub
```

```
GRUB_CMDLINE_LINUX="... rd.luks.name=device-UUID=cryptlvm ..."
```

See **dm-crypt/System configuration#Kernel parameters** and **GRUB#Encrypted /boot** for details. The *`device-UUID`* refers to the UUID of the LUKS superblock, in this case `/dev/sda3` (the partition which holds the lvm containing the root file system). See **Persistent block device naming**.

install GRUB to the mounted ESP for UEFI booting:

```
# grub-install --target=x86_64-efi --efi-directory=/efi --bootloader-id=GRUB --recheck
```

install GRUB to the disk for BIOS booting:

```
# grub-install --target=i386-pc --recheck /dev/sda
```

Generate GRUB's **configuration** file:

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

If all commands finished without errors, GRUB should prompt for the passphrase to unlock the `/dev/sda3` partition after the next reboot.

8.5 Avoiding having to enter the passphrase twice

While GRUB asks for a passphrase to unlock the LUKS1 encrypted partition after above instructions, the partition unlock is not passed on to the initramfs. Hence, you have to enter the passphrase twice at boot: once for GRUB and once for the initramfs.

This section deals with extra configuration to let the system boot by only entering the passphrase once, in GRUB. This is accomplished by **with a keyfile embedded in the initramfs**.

First create a keyfile and add it as LUKS key:

```
# dd bs=512 count=4 if=/dev/random of=/root/cryptlvm.keyfile iflag=fullblock
# chmod 000 /root/cryptlvm.keyfile
# cryptsetup -v luksAddKey /dev/sda3 /root/cryptlvm.keyfile
```

Add the keyfile to the initramfs image:

```
/etc/mkinitcpio.conf
```

```
FILES=(/root/cryptlvm.keyfile)
```

Recreate the initramfs image and secure the embedded keyfile:

```
# chmod 600 /boot/initramfs-linux*
```

Set the following kernel parameters to unlock the LUKS partition with the keyfile. Using the `encrypt` hook:

```
GRUB_CMDLINE_LINUX="... cryptkey=rootfs:/root/cryptlvm.keyfile"
```

Or, using the **sd-encrypt** hook:

```
GRUB_CMDLINE_LINUX="... rd.luks.key=device-UUID=/root/cryptlvm.keyfile"
```

If for some reason the keyfile fails to unlock the boot partition, systemd will fallback to ask for a passphrase to unlock and, in case that is correct, continue booting.

Tip: If you want to encrypt the `/boot` partition to protect against offline tampering threats, the **mkinitcpio-chkcryptoboot** hook has been contributed to help.

8.6 Using a USB drive to unlock /boot

To avoid having to memorise a complicated password, or using a simple one which may be guessed, a keyfile stored on an external USB drive can be used to unlock the LUKS volume. For this to be secure, this USB drive must be stored securely away from the computer when not in use.

First, generate a keyfile in the same way as in **#Avoiding having to enter the passphrase twice**. Do not use the same keyfile as if the USB drive is lost or compromised you will need to replace the keyfile embedded in initramfs.

Copy this keyfile to your USB drive and create a new GRUB configuration file:

```
/boot/grub/grub-pre.cfg
```

```
set crypto_uuid=UUID-of-the-luks-volume
set key_disk=UUID-of-the-volume-with-the-key
cryptomount -u $crypto_uuid -k ($key_disk)/the-location-of-the-key-on-your-usb
set root=UUID-of-the-unlocked-volume-as-in-grub.cfg
set prefix=($root)/boot/grub
insmod normal
normal
```

Create a GRUB image and install it (not all of these modules will be needed depending on your file system):

```
# grub-mkimage -p /boot/grub -O x86_64-efi -c /boot/grub/grub-pre.cfg -o /tmp/grubx64.efi  part_gpt  part
_msdos cryptodisk  luks  gcry_rijndael gcry_sha512 lvm ext2 ntfs fat exfat
```

```
# install -v /tmp/grubx64.efi /efi/EFI/GRUB/grubx64.efi
```

9 Root on ZFS

To use dm-crypt with ZFS, see ZFS#Encryption in ZFS using dm-crypt.

Additionally, ZFS features native encryption, which may also be utilized to encrypt the system root, excluding the boot loader and file system metadata. See:

- Arch Linux Root on ZFS (https://openzfs.github.io/openzfs-docs/Getting%20Started/Arch%20Linux/Root%20on%20ZFS.html) guide on the OpenZFS page,
- Install Arch Linux on ZFS#Installation.

After the installation, a boot loader can be verified with Secure Boot on UEFI-based systems.

-