LUKS on LVM (encrypted, cached, Desktop)

LUKS (Linux Unified Key Setup) is the standard for Linux hard disk encryption. By providing a standard on-disk-format, it does not only facilitate compatibility among distributions, but also provides secure management of multiple user passwords. LUKS stores all necessary setup information in the partition header, enabling to transport or migrate data seamlessly.

Management of LUKS encrypted devices is done via the `cryptsetup` utility.

Nomenclature

Term	Description
Physical Volume (PV)	On-disk partitioning format to be combined in a VG to a common storage pool
Volume Group (VG)	Grouping of one or more PVs to provide a combined storage pool from which storage can be requested in the form of LVs.
Logical Volume (LV)	Logical partition format which can be accessed like a block device to hold file systems and data.
Cache device	Fast storage used for caching reads/writes to slow storage
Origin device	Slow primary storage holding the actual data

Partitioning Setup

LUKS on LVM has the benefit of a LUKS container being able to span multiple disks, thanks to the machanisms of the underlying LVM. This, however, comes with the downside that if you want to have multiple volumes (e.g. for your root volume and a separate home volume or encrypted SWAP) you will have to take extra steps to unlock these volumes during the boot process.

> **NOTE:** If you want to utilize LVM cache this is the desired partioning scheme to use, as the encrypted LUKS container will reside inside an LVM LV and the LVM caching mechanism will cache the LV instead of the unlocked LUKS container, thus not leaking any secrets into the

> cache.

This guide assumes the following:

- This is used on a desktop computer without the need to resume (no SWAP partition)
- There are multiple drives: `/dev/nvme0n1` (SSD) and `/dev/sda` (HDD)
- The HDD will be cached by the SSD
- The root file system will be btrfs, with subvolumes for `/` and `/home`
- To tighten security, this setup assumes a unified kernel image and booting via EFISTUB, with the *ESP* mounted at `/efi`. Extra steps will be necessary to make the machine bootable.

Preparing partition layout

Start by listing available disks:

```
fdisk -l
```

Create a partition layout with `cfdisk` by pointing it to the first disk, e.g. `/dev/nvme0n1`:

> **ATTENTION:** `cfdisk` expects a device file, not a partition.

```
cfdisk /dev/nvme0n1
```

If `cfdisk` asks you about the partition table scheme to use, select `gpt`.

Create the following partition layout:

FS Type	Size	Mount Point	Comment
vfat	1G	/efi	EFI System
LVM	(remaining)		Linux LVM

Start `cfdisk` for the second disk, e.g. `/dev/sda`:

```
cfdisk /dev/sda
```

Create the following partition layout:

FS Type	Size	Mount Point	Comment
LVM	(all)		Linux LVM

Setting up LVM

Start by creating LVM PVs on the partitions we just laid out:

```
pvcreate /dev/nvme0n1p2   # SSD
pvcreate /dev/sda1        # HDD
```

Next, create a VG spanning both PVs:

> **NOTE:** `vg0` is used as an example here. Name your VG whatever you like.

```
vgcreate vg0 /dev/nvme0n1p2 /dev/sda1
```

Create an LV inside `vg0`, using 100% of the available space on the PV at `/dev/sda1` and label it `lv_root`:

```
lvcreate -l 100%FREE -n lv_root vg0 /dev/sda1
```

Create an LV inside `vg0`, using 100% of the available space on the PV at `/dev/nvme0n1p2` and label it `lv_cache`:

```
lvcreate -l 100%FREE -n lv_cache --type cache-pool vg0 /dev/nvme0n1p2
```

Finally, link both LVs together so that the LV on the HDD is being cached by the pool on the SSD:

```
lvconvert --type cache --cachepool vg0/lv_cache vg0/lv_root
```

Creating the LUKS container

Create the LUKS container inside the LV of the origin device:

> **WARNING:** Do **NOT** forget your passphrase! In case of loss you won't be able to access the data inside the container anymore!

```
cryptsetup luksFormat /dev/mapper/vg0-lv_root
```

Open the newly created LUKS container and supply the passphrase you just set:

> **NOTE:** `cryptroot` is used as an example here. Use whatever you like.

```
cryptsetup open /dev/mapper/vg0-lv_root cryptroot
```

Formatting and mounting partitions

Create file systems for the ESP and the root file system:

```
mkfs.fat -F 32 /dev/nvme0n1p1
mkfs.btrfs /dev/mapper/cryptroot
```

Mount the root btrfs file system and create the subvolumes:

```
mount /dev/mapper/cryptroot /mnt

btrfs subvolume create /mnt/@
btrfs subvolume create /mnt/@home
```

Unmount the root btrfs file system:

```
umount -R /mnt
```

Mount the `@` subvolume:

```
mount /dev/mapper/cryptroot -o noatime,compress-force=zstd,space_cache=v2,subvol=@ /mnt
```

Create mount points for `/efi` and `/home`:

```
mkdir -p /mnt/{efi,home}
```

Mount the remaining partitions and subvolumes:

```
mount /dev/nvme0n1p1 /mnt/efi
mount /dev/mapper/cryptroot -o noatime,compress-force=zstd,space_cache=v2,subvol=@home
/mnt/home
```

Revision #3
Created 2022-03-02 13:41:09 UTC by Sebin
Updated 2024-03-06 07:18:50 UTC by Sebin