Shrinking an encrypted partition with LVM on LUKS | Linux-Blog – Dr. Mönchmeyer

22-28 minutes

Whilst experimenting with with encrypted partitions I wanted to shrink an “oversized” LUKS-encrypted partition. I was a bit astonished over the amount of steps required; in all the documentations on this topic some points were missing. In addition I also stumbled over the mess of GiB and GB units used in different tools. Safety considerations taking into account the difference are important to avoid data loss. Another big trap was the fact that the tool “parted” expects an end point on the disk given in GB when resizing. I nearly did it wrong.

Otherwise the sequence of steps described below worked flawlessly in my case. I could reboot the resized root-system enclosed in the encrypted partition after having resized everything.

I list up the necessary steps below in a rather brief form, only. I warn beginners: This is dangerous stuff and the risk for a full data loss not only in the partition you want to resize is very high. So, if you want to experiment yourself – **MAKE A BACKUP of the whole disk** first.

Read carefully through the steps before you do anything. If you are unsure about what the commands mean and what consequences they have do some research on the Internet ahead of any actions. Likewise in case of trouble or error messages.

You should in general be familiar with partitioning, LVM, dm-crypt and LUKS. I take no responsibility for any damage or data loss and cannot guarantee the applicability of the described steps in your situation – you have been warned.

My disk layout

My layout of the main installation was

SSD –> partition 3 –> LUKS –> LVM –> Group “vg1” –> Volume “lvroot” –> ext4 fs [80 GiB]

SSD –> partition 3 –> LUKS –> LVM –> Group “vg1” –> Volume “lvswap” –> swap fs [2 GiB]

The partition had a size around 104 GiB before shrinking. “lvroot” included a bootable root filesystem of 80 GiB in size. The swap volume (2 GiB) will later help us to demonstrate that shrinking may lead to gaps between logical LVM volumes. We shall shrink the file system, its volume, the volume group and also the encrypted the partition in the end.

In my case I could attach the disk in question to another computer where I performed the resizing operation. If you have just *one* computer available use a Live System from an USB stick or a DVD. Or boot another Linux installation available on one of the other disks of your system. [I recommend to always have a second small bootable installation available on one of the disks of your PC/laptop besides the main installation for daily work :-). This second installation can reside in an encrypted LVM volume (LUKS on LVM), too. It may support administration and save your life one day in case your main installation becomes broken. There you may also keep copies of the LUKS headers and keyfiles …].

Resizing will be done by a sequence of 16 steps – following the disk layout in reverse order as shown above; i.e. we proceed with first resizing the filesystem, then taking care about the LVM layout and eventually changing the partition size.

You open an encrypted partition with LVM on LUKS just as any dm-crypt/LUKS-partition by

cryptsetup open /dev/YOUR_DEVICE… MAPPING-Name

For the mapping name I shall use "**cr-ext**". Note that manually closing the encrypted device requires to deactivate the volume groups in the kernel first; in our case

vgchange -a n vg1;
cryptsetup close cr-ext

Otherwise you may not be able to close your device.

A sequence of 16 steps ….

Let us now go through the details of the required steps – one by one.

Step 1: Get an overview over your block devices

You should use

lsblk

In my case the (external) disk appeared as "/dev/sdg" – the encrypted partition was located on "/dev/sdg3".

****

Step 2: Open the encrypted partition

cryptsetup open /dev/sdg3 cr-ext

Check that the mapping is done correctly by "**la /dev/mapper**" (la = alias for "ls -la"). Watch out not only for the "cr-ext"-device, but also for LVM-volumes inside the encrypted partition. They should appear automatically as distinct devices.

****

Step 3: Get an overview on LVM structure

Use the following standard commands:

pvdisplay
vgdisplay
lvdisplay

"pvdisplay"/"vgdisplay" should show you a PV device "/dev/mapper/cr-ext" with volume group "vg1" (taking full size). If the volume groups are not displayed you may need to make them available by "vgchange –available y" (see the man pages).

"lvdisplay" should show you the logical volumes consistently with the entries in "/dev/mapper". Note: "lvdisplay" actually shows the volumes as "/dev/vg1/lvroot", whereas the link in "/dev/mapper" includes the name of the volume group: "/dev/mapper/vg1-lvroot". In my case the original size of "lvroot" was 80 GiB and the size of the swap "lvswap" was 2 GiB.

****

Step 4: Check the integrity of the filesystem

Use fsck:

```
mytux:~ # fsck /dev/mapper/vg1-lvroot
```

```
fsck from util-linux 2.31.1
e2fsck 1.43.8 (1-Jan-2018)
/dev/mapper/vg1-lvroot: clean, 288966/5242880 files, 2411970/20971520
blocks
```

The filesystem should be clean. Otherwise run “fsck -f” – and answer the questions properly. “fsck” in write mode should be possible as we have not mounted the filesystem. We avoid doing so throughout the whole procedure.

****

Step 5: Check the physical block size of the filesystem and the used space within the filesystem

Use “fdisk -l” to get the logical and physical block sizes

```
Disk /dev/mapper/vg1-lvroot: 80 GiB, 85899345920 bytes, 167772160 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

Use **tunefs** to get some block information:

tune2fs -l /dev/mapper/vg1-lvroot

to get the number of total blocks (20971520), blocksize (4096) and free blocks (18559550) – in my case :

```
Total = 85899345920 Bytes = 85 GB = 80 GiB
free  = 76019916800 Bytes = 76 GB = 70,8 GiB
```

If you mount df -h may will show you less available space due to a 5% free limit set => 67G. So, not much space was occupied by my (Opensuse) test installation inside the volume.

****

Step 6: Plan the reduced volume and filesystem sizes ahead – perform safety and limit considerations

Plan ahead the new diminished size of the LVM-Volume (“lvroot”). Let us say, we wanted 60G instead of the present 80G. Then we ***must*** reduce the filesystem [fs] size even more and use a **safety-factor of 0.9** => 54G. I stress:

Make the filesystem size at least 10% smaller than the planned logical volume size!

Why this precaution? There may be a difference for the meaning of “**G**” for the use of the resize commands: “**resize2fs**” and the “**lvresize**“. It could be “**GB**” or “**GiB**” in either case.

Have a look into the man pages. This difference is really dangerous! What would be the worst case that could happen? We resize the filesystem in GiB and the volume size in GB – then the volume size would be too small.

So, let us assume that “lvresize” works with GB – then the 60G would correspond to 60 * 1024 * 1000 * 1000 Bytes = 6144000000 Bytes. Assume further that “resize2fs” works with GiB instead. Then we would get 54 * 1024 * 1024 * 1024 Bytes = 57982058496 Bytes. We would still have a reserve! We would potentially through away disk space;
however, we will compensate for it afterwards (see below).

If you believe what the present man-pages say: For “resize2fs” the “size” parameter can actually be given

in units of “G” – but internally GiB are used. For “lvresize”, however, the situation is unclear.

In addition we have to check the potential reduction range against the already used space! In our case there is no problem (see step 5). However, how far could you maximally shrink if you needed to?

Minimum-consideration: Give the 10 GiB used according to step 5 a good 34% on top. (We always want a free space of 25%). Multiply by a factor of 1.1 to account for potential GB-GiB differences => 15 GiB. This is a rough minimum limit for the minimum size of the filesystem. The logical volume size should again be larger by a factor of 1.1 at least => 16.5 GB.

Having calculated your minimum you choose your actual shrinking size above this limit: You reduce the volume size by something between the 80 GiB and the 16.5 GiB. You set a number – let us say 60 GiB – and then you reduce the filesystem by a factor 0.9 more => 54 GB.

****

Step 7: Shrink the filesystem

Check first that the fs is unmounted! Otherwise you may run into trouble:

```
mytux:~ # resize2fs /dev/mapper/vg1-lvroot 60G
resize2fs 1.43.8 (1-Jan-2018)
Filesystem at /dev/mapper/vg1-lvroot is mounted on /mnt2; on-line resizing
required
resize2fs: On-line shrinking not supported
```

So unmount, run fsck again and then resize:

```
maytux:~ # umount /mnt2
mytux:~ # e2fsck -f /dev/mapper/vg1-lvroot
e2fsck 1.43.8 (1-Jan-2018)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/mapper/vg1-lvroot: 288966/5242880 files (0.2% non-contiguous),
2411970/20971520 blocks
mytux:~ # resize2fs /dev/mapper/vg1-lvroot 54G
resize2fs 1.43.8 (1-Jan-2018)
Resizing the filesystem on /dev/mapper/vg1-lvroot to <pre
style="padding:8px;"> (4k) blocks.
The filesystem on /dev/mapper/vg1-lvroot is now 14155776 (4k) blocks long.
mytux:~ # e2fsck -f /dev/mapper/vg1-lvroot
e2fsck 1.43.8 (1-Jan-2018)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/mapper/vg1-lvroot: 288966/3538944 files (0.3% non-contiguous),
2304043/14155776 blocks
```

****

Step 8: Shrink the logical volume

We use “lvreduce” to resize the LVM volume; the option parameter “L” together with a “size” determines how big the volume will become. [Note that there is a subtle difference if you provide the *size* with a negative “-” sign! See the man page for the difference!]

```
mytux:~ # lvreduce -L 60G /dev/mapper/vg1-lvroot
  WARNING: Reducing active logical volume to 60.00 GiB.
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce vg1/lvroot? [y/n]: y
  Size of logical volume vg1/lvroot changed from 80.00 GiB (20480 extents)
to 60.00 GiB (15360 extents).
  Logical volume vg1/lvroot successfully resized.
mytux:~ #
```

Hey, we worked in GiB – good to know!

****

Step 9: Extend the fs to the volume size again

We compensate for the lost space of almost 6 GiB:

```
mytux:~ # resize2fs /dev/mapper/vg1-lvroot
resize2fs 1.43.8 (1-Jan-2018)
Resizing the filesystem on /dev/mapper/vg1-lvroot to 15728640 (4k) blocks.
The filesystem on /dev/mapper/vg1-lvroot is now 15728640 (4k) blocks long.
```

Run gparted to get an overview over the present situation.

****

Step 10: Check for gaps between the volumes of your LVM volume group

Besides the LVM volume for the OS and data related filesystem we had a swap volume, too! As we have not changed it there should be a gap between the volumes now – and indeed:

```
mytux:~ # pvs -v --segments /dev/mapper/cr-ext
    Wiping internal VG cache
    Wiping cache of LVM-capable devices
  PV                 VG  Fmt  Attr PSize   PFree  Start SSize LV     Start
Type   PE Ranges
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g     0 15360 lvroot     0
linear /dev/mapper/cr-ext:0-15359
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 15360  5120           0
free
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 20480   512 lvswap     0
linear /dev/mapper/cr-ext:20480-20991
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 20992  5623           0
free
mytux:~ #
```

Now, to close the gap we can move the second volume. This requires multiple trials with “pvmove –alloc anywhere”.

Note the information “/dev/mapper/cr-ext:20480-20991” and use this *exactly* in “pvmove”:

```
mytux:~ # pvmove --alloc anywhere /dev/mapper/cr-ext:20480-20991
  /dev/mapper/cr-ext: Moved: 0.20%
  /dev/mapper/cr-ext: Moved: 100.00%
mytux:~ # pvs -v --segments /dev/mapper/cr-ext
    Wiping internal VG cache
    Wiping cache of LVM-capable devices
  PV                VG  Fmt  Attr PSize   PFree  Start SSize LV     Start
Type   PE Ranges
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g     0 15360 lvroot     0
linear /dev/mapper/cr-ext:0-15359
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 15360  5632            0
free
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 20992   512 lvswap     0
linear /dev/mapper/cr-ext:20992-21503
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 21504  5111            0
free
```

You may have to apply “pvmove” several times, but each time with the new **changed** position parameters – be careful !!! (Are your backups OK???).

This may give you some seemingly erratic movements, but eventually, you should see a continuous alignment:

```
mytux:~ # pvmove --alloc anywhere /dev/mapper/cr-ext:20992-21503
  /dev/mapper/cr-ext: Moved: 1.76%
  /dev/mapper/cr-ext: Moved: 100.00%
mytux:~ # pvs -v --segments /dev/mapper/cr-ext
    Wiping internal VG cache
    Wiping cache of LVM-capable devices
  PV                VG  Fmt  Attr PSize   PFree  Start SSize LV     Start
Type   PE Ranges
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g     0 15360 lvroot     0
linear /dev/mapper/cr-ext:0-15359
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 15360   512 lvswap     0
linear /dev/mapper/cr-ext:15360-15871
  /dev/mapper/cr-ext vg1 lvm2 a--  103.96g 41.96g 15872 10743            0
free
mytux:~ #
```

****

Step 11: Resize/reduce the physical LVM

After having aligned the volumes of the volume group vg1 we now can resize the physical LVM volume supporting the vg1 group. We use “pvresize” for this purpose. “pvresize” works in GiB – but its parameters just have a “G”.
In my example I shrank the physical LVM area down to 80G = 80GiB. This left more than enough space for my two LVM volumes.

```
mytux:~ # pvresize --setphysicalvolumesize 80G /dev/mapper/cr-ext
/dev/mapper/cr-ext: Requested size 80.00 GiB is less than real size 103.97
GiB. Proceed?  [y/n]: y
  WARNING: /dev/mapper/cr-ext: Pretending size is 167772160 not 218034943
sectors.
  Physical volume "/dev/mapper/cr-ext" changed
```

```
  1 physical volume(s) resized / 0 physical volume(s) not resized
mytux:~ # pvdisplay
....
  --- Physical volume ---
  PV Name               /dev/mapper/cr-ext
  VG Name               vg1
  PV Size               80.00 GiB / not usable 3.00 MiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              20479
  Free PE               4607
  Allocated PE          15872
  PV UUID               eFf9we-......
```

Again, we obviously worked in GiB – good.

****

Step 12: Set new size of the encrypted region

It is disputed whether this step is required. LUKS never registers the size of the encrypted area. However, if you had an active mounted partition … In our case, I think the step is not required, but …..

“cryptsetup resize” wants sectors as a size unit. We have to calculate a bit again based on the amount of sectors (each with 512 Byte; see below) used:

```
mytux:~ # cryptsetup status cr-ext
/dev/mapper/cr-ext is active.
  type:    LUKS1
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: dm-crypt
  device:  /dev/sdg3
  sector size:  512
  offset:  65537 sectors
  size:    218034943 sectors
  mode:    read/write
```

We have to determine the new size in sectors to reduce the size of the LUKS area. I chose: 185329702 blocks (rounded up) – this corresponds to around 88.4 GiB; according to our 10% safety rule this should be sufficient.

```
mytux:~ # cryptsetup -b 185329702 resize cr-ext
mytux:~ # cryptsetup status cr-ext
/dev/mapper/cr-ext is active.
  type:    LUKS1
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: dm-crypt
  device:  /dev/sdg3
  sector size:  512
  offset:  65537 sectors
  size:    185329702 sectors
  mode:    read/write
```

****

Step 13: Reduce the size of the physical partition – a pretty scary step!

This is one of the most dangerous steps. And irreversible … Interestingly, gparted will not let you do a resize. Yast's partitioner allows for it after several confirmations. I used "**parted**" in the end.

But especially with parted you have to be extremely careful – parted expects a partition's ***end*** **position in GB**. This means that to calculate the resulting size correctly you must also look at the partition's ***starting*** **position**. In the end it is the **difference** that counts. Read the man pages and have a look into other resources.

```
mytux:~ # parted /dev/sdg
GNU Parted 3.2
Using /dev/sdg
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: USB 3.0 (scsi)
Disk /dev/sdg: 512GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name     Flags
 1      1049kB  16.8MB  15.7MB               primary  bios_grub
 2      33.6MB  604MB   570MB                primary  boot, esp
 3      604MB   112GB   112GB                primary  legacy_boot, msftdata
 4      291GB   312GB   21.5GB                        lvm
 5      312GB   314GB   2147MB               primary  msftdata

(parted) resizepart
Partition number? 3
End?  [112GB]? 89GB
Warning: Shrinking a partition can cause data loss, are you sure you want
to continue?
Yes/No? Yes
(parted) print
Model: USB 3.0 (scsi)
Disk /dev/sdg: 512GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name     Flags
 1      1049kB  16.8MB  15.7MB               primary  bios_grub
 2      33.6MB  604MB   570MB                primary  boot, esp
 3      604MB   89.0GB  88.4GB               primary  legacy_boot, msftdata
 4      291GB   312GB   21.5GB                        lvm
 5      312GB   314GB   2147MB               primary  msftdata

(parted) q
Information: You may need to update /etc/fstab.
```

As you see: We end up with a size of 88.4 GB and **NOT** 89 GB !!!

In addition we shall see in a second that the GB are **NOT** GiB here! So our safety factor is really

important to get close to the aspired 80 GiB.

****

Step 14: Set new size of the encrypted region

As we have some space left now in the partition we can enlarge the encryption area and later also the PV size to the possible maximum.
First, we set back the size of the encryption area to the full partition size; though not required in our case – it does not harm:

```
mytux:~ # cryptsetup  resize cr-ext
mytux:~ # cryptsetup status cr-ext
/dev/mapper/cr-ext is active and is in use.
  type:    LUKS1
  cipher:  aes-xts-plain64
  keysize: 512 bits
  key location: dm-crypt
  device:  /dev/sdg3
  sector size:  512
  offset:  65537 sectors
  size:    172582959 sectors
  mode:    read/write
```

****

Step 15: Reset the PV size to the full partition size

We use “pvsize” again without any size parameter. The volume group will follow automatically.

```
mytux:~ # pvresize  /dev/mapper/cr-ext
  Physical volume "/dev/mapper/cr-ext" changed
  1 physical volume(s) resized / 0 physical volume(s) not resized
mytux:~ # pvdisplay
...
  --- Physical volume ---
  PV Name               /dev/mapper/cr-ext
  VG Name               vg1
  PV Size               82.29 GiB / not usable 1000.50 KiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              21067
  Free PE               5195
  Allocated PE          15872
  PV UUID               eFf9we-......
..
mytux:~ # vgdisplay
....
  --- Volume group ---
  VG Name               vg1
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  16
  VG Access             read/write
  VG Status             resizable
```

```
  MAX LV                0
  Cur LV                2
  Open LV               0
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               82.29 GiB
  PE Size               4.00 MiB
  Total PE              21067
  Alloc PE / Size       15872 / 62.00 GiB
  Free  PE / Size       5195 / 20.29 GiB
  VG UUID               dZakZi-......
```

Our logical volumes are alive, too, and have the right size.

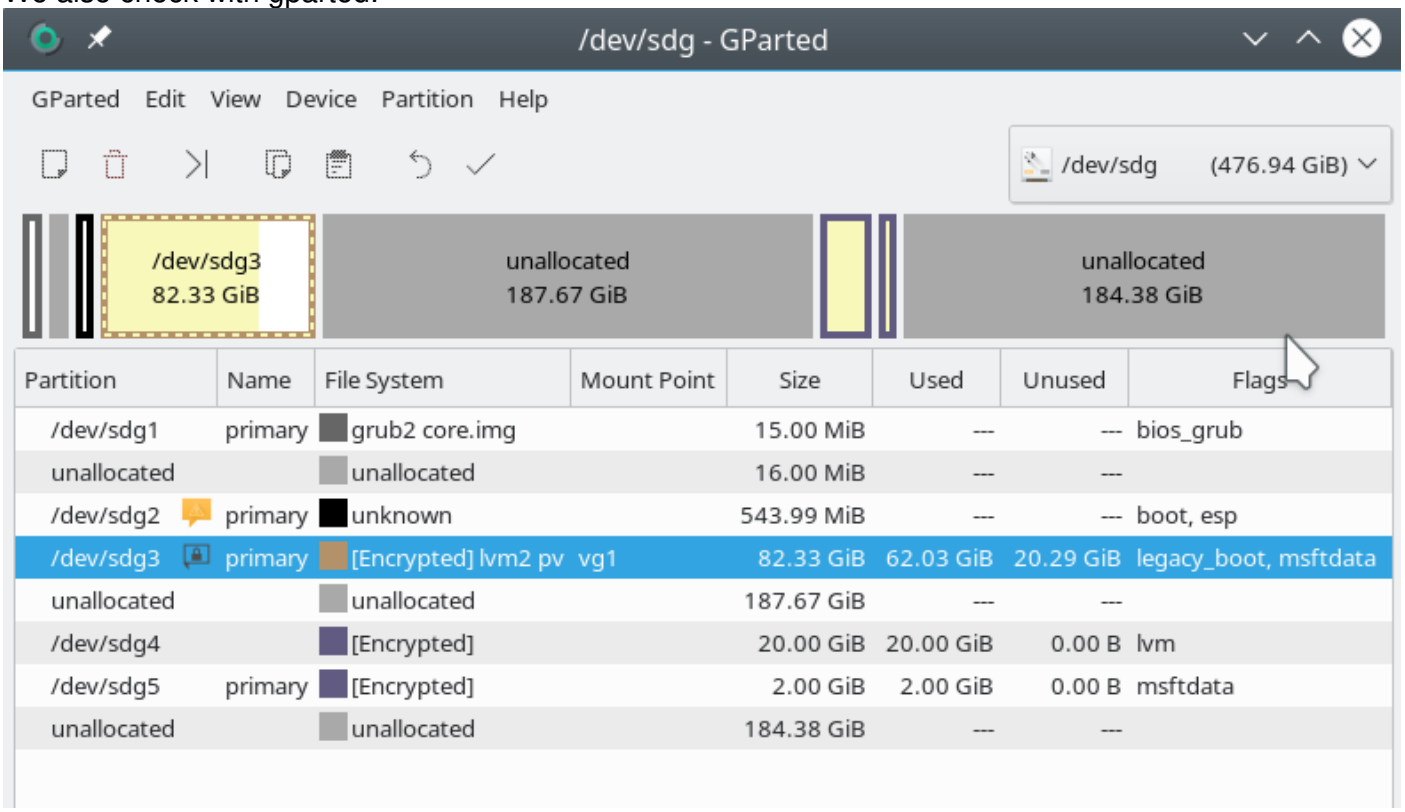
```
mytux:~ # lvdisplay
...
  --- Logical volume ---
  LV Path                /dev/vg1/lvroot
  LV Name                lvroot
  VG Name                vg1
  LV UUID                5cDvmf-......
  LV Write Access        read/write
  LV Creation host, time install, 2018-11-06 15:33:52 +0100
  LV Status              available
  # open                 0
  LV Size                60.00 GiB
  Current LE             15360
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     1024
  Block device           254:17

  --- Logical volume ---
  LV Path                /dev/vg1/lvswap
  LV Name                lvswap
  VG Name                vg1
  LV UUID                lCU75L-......
  LV Write Access        read/write
  LV Creation host, time install, 2018-11-06 18:21:11 +0100
  LV Status              available
  # open                 0
  LV Size                2.00 GiB
  Current LE             512
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     1024
  Block device           254:18
```

And:

```
mytux:~ # pvs -v --segments /dev/mapper/cr-ext
    Wiping internal VG cache
```

```
    Wiping cache of LVM-capable devices
  PV                VG  Fmt  Attr PSize  PFree  Start SSize LV     Start
Type   PE Ranges
  /dev/mapper/cr-ext vg1 lvm2 a--  82.29g 20.29g     0 15360 lvroot     0
linear /dev/mapper/cr-ext:0-15359
  /
dev/mapper/cr-ext vg1 lvm2 a--  82.29g 20.29g 15360   512 lvswap     0
linear /dev/mapper/cr-ext:15360-15871
  /dev/mapper/cr-ext vg1 lvm2 a--  82.29g 20.29g 15872  5195           0
free
```

We also check with gparted:



Everything is fortunately Ok!

****

Step 16: Closing and leaving the encrypted device

```
mytux:~ # vgchange -a n vg1

  0 logical volume(s) in volume group "vg1" now active
mytux:~ #
mytux:~ # cryptsetup close cr-ext
mytux:~ #
```

Conclusion

Shrinking an encrypted Luks partition which is used as a physical LVM volume and contains a LVM group with volumes is no rocket science. One has to apply the necessary shrinking steps starting from the innermost objects to the containers around them. I.e, we start shrinking the LVM volumes first, then take care of the LVM volume group and the LVM PV. Eventually, we deal with the encryption area and the

hard disk partition.

During each step one has to be careful regarding the tools and sizing units: For each tool it has to be clarified whether it works in GB or GiB. Safety margins during each shrinking step have to be calculated and to be taken into account.

Links

https://www.rootusers.com/lvm-resize-how-to-decrease-an-lvm-partition/
https://wiki.archlinux.org/index.php/Resizing_LVM-on-LUKS
https://blog.shadypixel.com/how-to-shrink-an-lvm-volume-safely/
https://unix.stackexchange.com/questions/41091/how-can-i-shrink-a-luks-partition-what-does-cryptsetup-resize-do

LVM-fragmentation

See the discussion in the following link:
https://askubuntu.com/questions/196125/how-can-i-resize-an-lvm-partition-i-e-physical-volume
https://www.linuxquestions.org/questions/linux-software-2/how-do-i-lvm2-defrag-or-move-based-on-logical-volumes-689335/

Introduction into LVM

https://www.tecmint.com/create-lvm-storage-in-linux/
https://www.tecmint.com/extend-and-reduce-lvms-in-linux/