TPM Owner Provisioning for Linux

Introduction

You are owner of a new x86 PC or VM, and want to install Linux, perhaps even dual boot, with bitlocker and Luks disk encryption. What do you need to do to provision the system's TPM? This article will discuss the recommended steps, including:

- Set a TPM Owner password.
- Create a persistent Storage Root Key (SRK).
- Create a persistent Default Recoverable Storage Key (DRSK) and back it up.

The easiest approach is to install Linux as normal, and then provision and utilize the TPM once booted. Link [1] provides a "provision.sh" script that does all of the recommended provisioning. This article explains what that script does, and why.

Set the TPM Owner password

Your PC or VM will come with a TPM that has been partially provisioned by the vendor, but it will most likely not have the TPM owner password set. This is because the owner (you) needs to know and set it. If you do not set an owner password, an attacker could delete any TPM keys you have created (including disk encryption keys), and could set an owner password, locking you out of the TPM. The TPM owner password is not needed for normal operation, such as booting. It is usually only used for initial provisioning.

You set the TPM owner password with

```
tpm2_changeauth -c owner <owner password>
```

Setting the TPM owner password will not affect Windows 11 on a dual boot system, even if you have bitlocker turned on.

Create an SRK

Before we can create any persistent keys, we need to know what keys have already been provisioned by the vendor. To get a list of all persistent key handles already provisioned, use:

```
tpm2_getcap --handles-persistent
```

and to get information about each of these keys, use:

```
tpm2_readpublic -c <handle>
```

My latest desktop PC with Windows 11 Pro (running bitlocker) showed these persistent keys:

```
0x81000001 (SRK)        rsa 2048 decrypt
0x81000002 ???          rsa 2048 sign
0x81010001 (EK)         rsa 2048 decrypt
0x81010016 ???          ecc 256 decrypt
```

The SRK is used by Windows, and the EK is needed for attestation, so we definitely do not want to delete them. The other two keys are unknown, but we probably should leave them alone too.

A new VM (hosted on Fedora 38 with libvirt) with Fedora 38 installed as guest showed just one key:

0x81010001 (EK)	rsa 2048 decrypt

Here we again would want to leave the EK alone, but all other handles are available.

At this point, it is also interesting to look at any provisioned NV files on the TPM. Use

```
tpm2_getcap handles-nv-index
```

and

```
tpm2_nvread
```

to display nv data. Note that you may not have authorization to read the data in some platform NV files. My desktop showed:

0x1410001 ???	8 bytes (0x100)
0x1410002 ???	8 bytes (9x01)
0x1410004 ???	8 bytes
0x1880001 ???	8 bytes (0x01)
0x1880011 ???	32 bytes

and my Fedora VM showed:

0x01c00002 (EK cert)	Swtpm-localca cert
0x01c00016	fedora cert
0x01c08000	fedora cert

Libvirt creates a software TPM for each guest. As the TPM “vendor”, it creates not only the EK key we saw before, but also creates CA and EK certificates for attestation, and stores them in these NV files. We will discuss the use of the EK and its certificates for attestation later. For now, we want to make sure we don’t modify these keys and certificates.

Since 0x81000001 and 0x81000002 are already in use by Windows, we will use 0x81000003 and 0x81000004 for the Linux SRK and DRSK respectively. Depending on your system, you may want or need to use other handles.

Create the SRK with:

```
tpm2_createprimary -c primary.ctx -P <owner password>
```

and make it persistent at 0x81000003 with

```
tpm2_evictcontrol -C o -c primary.ctx 0x81000003 -P <owner password>
```

Create the DRSK

Linux applications, such as trusted keys, LUKS disk encryption keys, and password hmac keys, can use this SRK at 0x81000003 for storage. The problem is that the SRK cannot be recovered in the event of TPM or motherboard hardware failure. While each application could manage its own backup and recovery scheme, it is much easier to use a recoverable storage root key, such as a DRSK. Then in the event of hardware failure, recovery of the DRSK transparently recovers all the applications using it. This is described in detail in [1], which provides scripts for backing up and restoring the DRSK, using a remote TPM to guarantee that the DRSK is never exposed outside of TPM hardware protection.

We want a persistent DRSK at 0x81000004 that can be loaded and used without authentication, so that it can be used by systemd-cryptsetup at boot time to decrypt the disk partitions. We also want a key that requires owner auuthentication for duplication, so that only the owner can perform the sensitive backup and recovery operations, so that we know that the DRSK is only ever duplicated to a trusted TPM.

This can be done with a duplication policy which includes tpm2_policysecret. Duplication always requires a policy, and this policy will also require a successful owner authentication. In detail this looks like:

```
tpm2_startauthsession -S session.dat
tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate
tpm2_policysecret -S session.dat -c o -L dpolicy.dat <owner password>
tpm2_flushcontext session.dat
tpm2_create -C 0x81000003 -r my_drsk.prv -u my_drsk.pub -L dpolicy.dat -a \
     "sensitivedataorigin|userwithauth|restricted|decrypt"
tpm2_load -C 0x81000003 -u my_drsk.pub -r my_drsk.prv -c my_drsk.ctx
tpm2_evictcontrol -C o -c my_drsk.ctx 0x81000004 -P <owner password>
```

With a DRSK provisioned in this way, the policy file and owner password are required for duplication, but loading and using the DRSK require neither policy nor owner password.

Provisioning and TPM Applications

Once the TPM is provisioned in this way, applications are able to use the TPM.

Trusted keys are currently used by the Linux kernel to provide the needed HMAC key for EVM. Trusted keys are created with keyctl, and using its optional "handle=" argument, trusted keys can use either the SRK or DRSK as desired.

crypt_tpmhmac [2] is a shadow password hashing scheme used by PAM and libxcrypt which uses a secret TPM key to HMAC a user's salt and password. Since the TPM's HMAC key cannot be retrieved by an attacker, offline attacks on the shadow password file are impossible. crypt_tpmhmac's configuration file, "/etc/tpmhmac.conf" specifies the storage key handle used, so it can use either the SRK or DRSK as desired.

If a partition (including root) is encrypted with LUKS2, a TPM can be used to unlock the encryption at boot time without someone having to enter a LUKS password. This unlocking can be tied to the successful secure booting of a signed distribution, so that the LUKS key is protected. TPM support for LUKS can be enabled on an existing system with systemd_cryptenroll. Currently systemd_cryptenroll tries to create its own non-persistent primary, which will fail if the owner password is set. In this case, it next tries to use a persistent SRK at a hardcoded 0x81000001 handle. A patch at [1] modifies systemd to add a "--tpm2-srk-handle=" argument, so that the SRK or DRSK can be used.

Attestation: As a system boots and runs, measurements are taken by UEFI, the bootloaders, systemd (see [3]) and Linux IMA. These measurements can be attested to a third party using client/server applications such as [4]. The attestation applications use the TPM to sign the current measurement PCR values, so that the server can verify the TPM signature, and thus verify the entire list of measurements. This does require the TPM's EK and EK certificate, to prove to the verifier that the signatures came from an authentic TPM.

In both example systems the EK itself is at handle 0x81010001. In the VM example, the certificates are already provisioned in NV files. On Intel and AMD systems, you can retrieve the cert with

```
tpm2_getekcertificate
```

(For AMD, you may need to upgrade to the latest upstream tpm2-tools for this to work.)

Links

[1] https://github.com/safforddr/tpm_keys
[2] https://github.com/safforddr/crypt_tpmhmac
[3] https://0pointer.net/blog/brave-new-trusted-boot-world.html
[4] https://sourceforge.net/projects/ibmtpm20acs/