ARCH LINUX TECHNICAL RUNBOOK

TPM 2.0 Full Disk Encryption on Arch Linux

Dedicated Single-Boot Architecture Using LUKS2, systemd-boot, Unified Kernel Images (UKIs), Secure Boot (sbctl), and systemd-cryptenroll

Target Profile: Arch Linux (Single-Boot UEFI) | **Bootloader:** systemd-boot | **Initramfs:** systemd-based mkinitcpio

1. Architecture Overview & Boot Chain

In a dedicated single-boot Arch Linux system without Windows, you can eliminate legacy GRUB complexity and sign your entire early-boot chain with your own Secure Boot keys. Because `systemd-boot` cannot verify cryptographic signatures on its own, pairing it with Unified Kernel Images (UKIs) and custom Secure Boot keys via `sbctl` ensures that the UEFI firmware validates the entire kernel, initramfs, and kernel command line before execution.

Boot Stage	Component	Cryptographic Measurement & Verification
1. Firmware	UEFI + TPM 2.0	Measures firmware code into `PCR 0` and Secure Boot state/ certificates into `PCR 7`.
2. Bootloader	`systemd-bootx64.efi`	Signed with your custom Secure Boot `db` key. Measured into `PCR 4` and `PCR 7`.
3. Kernel / UKI	`arch-linux.efi` (UKI)	Bundles kernel, microcode, initramfs, and cmdline into one PE binary signed via `sbctl`.
4. Early Userspace	`systemd-cryptsetup`	Queries TPM 2.0 to unseal the LUKS2 volume key against `PCR 7` (and optional PIN).

Why PCR 7 Only (Instead of PCR 0 + 2 + 4 + 7)?
On Arch Linux, rolling kernel updates happen weekly. Binding to `PCR 4` (Boot Manager Code) invalidates TPM auto-unlock every time a new UKI is built unless you re-enroll on every update. Binding to `PCR 7` (Secure Boot Policy) keeps auto-unlock working across kernel updates while still guaranteeing that only binaries signed with your private Secure Boot key can boot.

2. Pre-Installation & Hardware Checklist

- **BIOS/UEFI Settings:** Disable CSM (Legacy Boot), enable TPM 2.0 (PTT on Intel / fTPM on AMD), and place Secure Boot into **Setup Mode** (clear existing factory Platform Keys/PK).
- **Partition Layout (Single-Boot NVMe Example):**
  - `/dev/nvme0n1p1` : 1 GiB EFI System Partition (FAT32, mounted at `/efi`)
  - `/dev/nvme0n1p2` : Remainder of disk (LUKS2 encrypted container holding Btrfs or ext4 root)
- **Verify TPM 2.0 Visibility in Live ISO:** Run `ls /dev/tpmrm0` and `cat /sys/class/tpm/tpm0/tpm_version_major` (must return `2`).

3. Base Installation & LUKS2 Setup

Format the EFI partition, initialize the LUKS2 container with a strong initial recovery passphrase, and install the core packages required for UKIs, TPM2, and Secure Boot.

```
# 1. Format ESP and create LUKS2 root container
mkfs.fat -F 32 /dev/nvme0n1p1
cryptsetup luksFormat --type luks2 /dev/nvme0n1p2
cryptsetup open /dev/nvme0n1p2 cryptroot

# 2. Format root filesystem (ext4 example; Btrfs subvolumes also work seamlessly)
mkfs.ext4 /dev/mapper/cryptroot
mount /dev/mapper/cryptroot /mnt
mount --mkdir /dev/nvme0n1p1 /mnt/efi

# 3. Install base system, kernel, microcode, TPM2 stack, and sbctl
pacstrap -K /mnt base linux linux-firmware amd-ucode tpm2-tss sbctl efibootmgr nano
networkmanager

genfstab -U /mnt >> /mnt/etc/fstab
arch-chroot /mnt
```

> **Note:** Replace `amd-ucode` with `intel-ucode` if using an Intel processor.

4. Configuring systemd-based Initramfs & UKIs

Traditional Arch setups use BusyBox hooks (`udev`, `encrypt`). For native TPM 2.0 unlocking via `systemd-cryptsetup`, switch `mkinitcpio` to systemd hooks and configure Unified Kernel Image generation.

Step 4.1: Define Kernel Command Line

Retrieve the exact UUID of the raw encrypted partition (`/dev/nvme0n1p2`, **not** `/dev/mapper/cryptroot`) and write it to `/etc/kernel/cmdline`.

```
blkid -s UUID -o value /dev/nvme0n1p2
# Write cmdline (replace <LUKS-UUID> with output above)
echo "rd.luks.name=<LUKS-UUID>=cryptroot rd.luks.options=tpm2-device=auto root=/dev/mapper/
cryptroot rw quiet" > /etc/kernel/cmdline
```

The `rd.luks.options=tpm2-device=auto` option tells the initramfs to try the TPM first. Until you enroll the TPM in Section 6, the system simply asks for your passphrase.

Step 4.2: Edit `/etc/mkinitcpio.conf`

Ensure tpm2 tools/libraries are included and replace BusyBox hooks with systemd equivalents:

```
MODULES=(tpm_crb tpm_tis)
BINARIES=()
FILES=()
HOOKS=(base systemd autodetect microcode modconf kms keyboard sd-vconsole block sd-encrypt
filesystems fsck)
```

Step 4.3: Enable UKI Output in `/etc/mkinitcpio.d/linux.preset`

Configure mkinitcpio to output signed-ready `.efi` images directly into the ESP's `EFI/Linux/` directory instead of loose `vmlinuz-linux` and `initramfs-linux.img` files.

```
# /etc/mkinitcpio.d/linux.preset
ALL_kver="/boot/vmlinuz-linux"
PRESETS=('default' 'fallback')

default_uki="/efi/EFI/Linux/arch-linux.efi"
fallback_uki="/efi/EFI/Linux/arch-linux-fallback.efi"
fallback_options="-S autodetect"
```

```
# Create target directory, install systemd-boot, and build initial UKIs
mkdir -p /efi/EFI/Linux
bootctl install
mkinitcpio -P
```

5. Custom Secure Boot Keys with `sbctl`

With UEFI in Setup Mode, generate your own Platform Key (PK), Key Exchange Key (KEK), and Signature Database (db) keys, enroll them into firmware, and sign both `systemd-boot` and your UKIs.

```
# 1. Verify Setup Mode is active
sbctl status

# 2. Create and enroll custom Secure Boot keys
sbctl create-keys
sbctl enroll-keys

# 3. Sign systemd-boot and the generated UKIs (the -s flag saves files to sbctl's pacman hook
DB)
sbctl sign -s /efi/EFI/BOOT/BOOTX64.EFI
sbctl sign -s /efi/EFI/systemd/systemd-bootx64.efi
sbctl sign -s /efi/EFI/Linux/arch-linux.efi
sbctl sign -s /efi/EFI/Linux/arch-linux-fallback.efi

# 4. Confirm all EFI binaries show as Signed
sbctl verify
```

> **Warning: Option ROMs & Firmware Brick Risk**
> Certain laptops (especially Lenovo ThinkPads) verify integrated GPU or network Option ROMs using Microsoft's third-party UEFI CA certificate during POST. If your hardware relies on Microsoft-signed Option ROMs, run `sbctl enroll-keys --microsoft` instead of bare `sbctl enroll-keys` to avoid a black screen at boot.

Reboot into your UEFI firmware settings, verify Secure Boot now reads **Enabled / User Mode**, and boot into your newly installed Arch Linux system.

6. Enrolling the TPM 2.0 Chip

Once booted into Arch Linux with Secure Boot active, bind your LUKS2 volume to the TPM 2.0 chip using `systemd-cryptenroll`. Choose either unattended auto-unlock or TPM + pre-boot PIN.

Option A: Unattended Auto-Unlock (PCR 7)

```
# Verify TPM 2.0 device detection
systemd-cryptenroll --tpm2-device=list

# Enroll TPM2 bound to PCR 7 (Secure Boot state)
sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 /dev/nvme0n1p2
```

Option B: Two-Factor TPM + Pre-Boot PIN (Recommended for Laptops)

Adding `--tpm2-with-pin=yes` forces an attacker who steals your powered-off laptop to know your numeric/alphanumeric PIN before the TPM will release the volume key—defeating SPI bus sniffing and cold-boot attacks while leveraging TPM anti-hammering lockout.

```
sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 --tpm2-with-pin=yes /dev/nvme0n1p2
```

Verify LUKS2 Keyslots & Generate a Recovery Key

```
# Inspect active keyslots (Slot 0 = initial passphrase, Slot 1 = systemd-tpm2)
sudo cryptsetup luksDump /dev/nvme0n1p2

# Optional: Add a high-entropy 64-character recovery key and print/store offline
sudo systemd-cryptenroll --recovery-key /dev/nvme0n1p2
```

7. Maintenance, Updates & Troubleshooting

Scenario	Action Required
Routine `pacman -Syu`	Fully automatic. mkinitcpio rebuilds `arch-linux.efi`, and `sbctl`'s built-in pacman hook re-signs it automatically. PCR 7 remains unchanged.
BIOS / UEFI Firmware Update	Firmware updates can modify Secure Boot variables or reset TPM state. Keep your LUKS recovery passphrase handy; if auto-unlock prompts for a passphrase after a BIOS update, wipe and re-enroll the TPM slot: `sudo systemd-cryptenroll --wipe-slot=tpm2 --tpm2-device=auto --tpm2-pcrs=7 /dev/nvme0n1p2`
systemd-boot Package Update	Run `sudo bootctl update` to update the EFI binary on `/efi`; `sbctl`'s hook or `sudo sbctl sign-all` ensures the updated binary stays signed.
TPM Anti-Hammering Lockout	Entering an incorrect TPM PIN too many times triggers a hardware dictionary-attack cooldown. Boot using your fallback LUKS passphrase in slot 0 and wait out the TPM lockout timer (or clear via `tpm2_dictionarylockout -c`).