Hardware

Get your gizmos up to speed

- Graphics Cards
- Sound
- Bluetooth
- Printing
- Trusted Platform Module
- Universial 2nd Factor (U2F)
- Fingerpint Reader

Graphics Cards

Most graphical user interfaces these days are hardware accelerated, so the appropriate graphics driver will be needed for optimal performance and a smooth desktop experience. Additionally, these drivers provide 3D acceleration and hardware video decoding/encoding capabilities.

The Linux graphics stack consists of several components, but the main component is the `mesa` package.

Manufacturer	OpenGL	Vulkan	Video Acceleration
Intel	`mesa`	`vulkan-intel`	(see note below)
AMD	`mesa`	`vulkan-radeon`	part of `mesa`
NVIDIA (Proprietary)	`mesa`	`nvidia-utils`	`nvidia-utils`, `nvidia-vaapi-driver`
NVIDIA (Nouveau)	`mesa`	`vulkan-nouveau`	part of `mesa`

Intel

For modern Intel integrated graphics (Broadwell generation / 5th Gen Core and newer) and Intel Arc, install the following packages:

```
pacman -S mesa vulkan-intel
```

Hardware acceleration for video playback depends on the generation of Intel hardware:

- `intel-media-driver`: If your CPU is **5th gen or newer** (Broadwell, Skylake, Kaby Lake, up to modern Alder Lake/Raptor Lake and Intel Arc).
- `libva-intel-driver`: If your CPU is **4th gen or older** (Haswell, Ivy Bridge, Sandy Bridge).

AMDGPU

For modern AMD integrated and dedicated graphics cards (RDNA architectures and modern Vega/GCN configurations), install the following packages:

```
pacman -S mesa vulkan-radeon
```

NVIDIA

In the case of NVIDIA, there's the option to either use the open-source Nouveau drivers, or the official Linux kernel modules provided by NVIDIA themselves.

If you have a relatively recent NVIDIA card, it is generally recommended to go with the official NVIDIA drivers for optimal performance. For older cards or strictly open-source setups, use the Nouveau/Mesa stack.

Nouveau open-source driver

The open-source NVIDIA stack uses Nouveau:

> **ATTENTION:** For older cards (Maxwell/Pascal), dynamic re-clocking requires manual firmware extraction via the AUR package `nouveau-fw`. For newer Turing cards (RTX 20xx / GTX 16xx) and above, the required GSP firmware is natively included in the `linux-firmware` package.

```
pacman -S mesa vulkan-nouveau
```

Official NVIDIA driver

Modern Cards (RTX 20xx onwards)

Upstream and Arch Linux officially recommend the **open-source kernel modules** for all modern architectures.

```
pacman -S nvidia-open nvidia-vaapi-driver
```

Legacy Cards (GTX 10xx and earlier)

If you are running older supported hardware or need the closed-source driver, you must use the proprietary DKMS package available via the AUR:

```
yay -S nvidia-580xx-dkms nvidia-vaapi-driver
```

Early KMS Loading

To ensure the graphics drivers load as early as possible during the boot sequence (preventing display manager race conditions), add the modules to your initramfs configuration.

Make sure the following is included in the `MODULES` array of your `/etc/mkinitcpio.conf` file (or a drop-in config file under `/etc/mkinitcpio.conf.d/`):

> **WARNING:** Early loading these kernel modules into the initramfs will break system hibernation features by default because video memory preservation is enabled.

```
MODULES=(nvidia nvidia_modeset nvidia_uvm nvidia_drm)
```

Enable Kernel Mode Setting

Since `nvidia-utils` version 560+, Kernel Mode Setting (KMS) is enabled by default. However, when using older legacy drivers, KMS must be explicitly enabled through a kernel command-line argument at boot time to ensure Wayland compositors function properly:

```
nvidia_drm.modeset=1
```

To verify that kernel mode setting is active, query sysfs:

```
cat /sys/module/nvidia_drm/parameters/modeset
```

`Y` means Kernel Mode Setting was enabled on boot.
`N` means Kernel Mode Setting was **not** enabled on boot.

Sound

For audio handling on Linux, PipeWire is the currently recommended framework.

PipeWire is a server and user space API that provides a platform to handle multimedia pipelines. It is a modern, low-latency audio and video server designed to work with the latest audio use cases and handle professional audio interfaces and applications.

PipeWire was created as a replacement for both the PulseAudio sound server and the Jack Audio Connection Kit (JACK) server. It provides a unified interface for handling video and audio streams and is intended to be flexible and extensible, allowing it to address not only audio but other multimedia tasks as well, such as video conferencing, screen capture, and other multimedia applications. It can also work with different hardware devices, including webcams, microphones, and professional audio devices.

Additionally, it integrates better with the security models of Flatpak and Wayland. It does so via sandboxing processes from one another, preventing an application from snooping on other applications' audio streams. Before allowing an application to record audio or sharing the screen(e.g. in a browser over WebRTC) it will ask the user for permission to do so.

PipeWire implements no connection logic internally, that is the responsibility of a program called a session manager. It watches for new streams and connects them to the appropriate output device or application.

There are two session managers to choose from:

- **PipeWire Media Session:** A very simple session manager that caters to some basic desktop use cases. It was mostly implemented for testing and as an example for building new session managers.
- **WirePlumber:** A more powerful manager and the current recommendation. It is based on a modular design, with Lua plugins that implement the actual management functionality.

WirePlumber is the recommended choice, as it is better maintained, receives regular updates and is more feature-rich.

Installation

The most basic PipeWire setup includes the following packages:

> **NOTE:** PipeWire handles Bluetooth audio devices if the `pipewire-audio` package is installed.

```
pacman -S pipewire pipewire-audio wireplumber
```

Additional packages can be installed to extend PipeWire's compatibility and capabilities:

Package	Description
`pipewire-alsa`	Support for routing ALSA clients through PipwWire
`pipewire-jack`	Support for JACK clients
`pipewire-pulse`	Support for PulseAudio clients (recommended)
`pipewire-v4l2`	Support for handling video devices, e.g. webcams, tuners, etc.
`pipewire-zeroconf`	Support for streaming audio over the network, e.g. an AirPlay receiver

Streaming audio to an AirPlay receiver

PipeWire can send audio to an AirPlay receiver via the `pipewire-zeroconf` package, which includes the necessary RTSP/RAOP modules to create a sink to send audio data to. This requires the Avahi zeroconf daemon.

Refer to the Network section on how to install and setup Avahi.

Firewall ports

If you're using a firewall, make sure that the following ports are open:

> **TIP:** `firewalld` has a preset for RTSP. Make sure to apply the firewall changes permanently.

Port	Protocol	Service
554	TCP	RTSP
554	UDP	RTSP
6001	UDP	Some 3rd party AirPlay receivers use this

Port	Protocol	Service
6002	UDP	Some 3rd party AirPlay receivers use this

Auto-load PipeWire RAOP discovery module

Create a new drop-in config file, e.g. `~/.config/pipewire/pipewire.conf.d/raop-discover.conf`:

```
context.modules = [
    {
        name = libpipewire-module-raop-discover
        args = {
            #raop.latency.ms = 1000
            stream.rules = [
                {
                    matches = [
                        {
                            raop.ip = "~.*"
                            #raop.ip.version = 4 | 6
                            #raop.ip.version = 4
                            #raop.port = 1000
                            #raop.name = ""
                            #raop.hostname = ""
                            #raop.domain = ""
                            #raop.device = ""
                            #raop.transport = "udp" | "tcp"
                            #raop.encryption.type = "RSA" | "auth_setup" | "none"
                            #raop.audio.codec = "PCM" | "ALAC" | "AAC" | "AAC-ELD"
                            #audio.channels = 2
                            #audio.format = "S16" | "S24" | "S32"
                            #audio.rate = 44100
                            #device.model = ""
                        }
                    ]
                    actions = {
                        create-stream = {
                            #raop.password = ""
```

```
                            stream.props = {
                                #target.object = ""
                                #media.class = "Audio/Sink"
                            }
                        }
                    }
                ]
            }
        }
]
```

Restart the `pipewire` user unit to make pipewire read the new drop-in config file and load the RAOP module automatically upon login:

```
systemctl restart --user pipewire
```

Scan for devices on the network

You can use the `avahi-browse` utility to scan for devices on your network:

```
avahi-browse --all --ignore-local --terminate
```

This will produce a list of devices broadcasting mDNS services over the network (not only AirPlay, but also file sharing, Spotify, Home Kit and various others).

You should now be able to `ping` your AirPlay receiver using its `.local` DNS name:

```
ping my-airplay-receiver.local
```

If everything worked as intended `wpctl status` should list new sinks to output audio to:

```
...
Audio
 ├─ Devices:
 │      53. Starship/Matisse HD Audio Controller [alsa]
 │
 ├─ Sinks:
 │      47. My AirPlay Reciever                  [vol: 1.00]
 │  *   61. Starship/Matisse HD Audio Controller Analog Stereo [vol: 1.00]
...
```

Finally, use your desktop environment's audio settings panel to select your AirPlay receiver as audio output device.

Bluetooth

Install the following packages to enable Bluetooth functionality and the necessary tools to control them:

```
pacman -S bluez bluez-utils
```

Enable the systemd unit to initialize Bluetooth during boot:

```
systemctl enable bluetooth
```

Printing

Install the following packages for printer support:

```
pacman -S cups logrotate system-config-printer gutenprint
```

Enable the following systemd units to initialize the printing system during boot:

```
systemctl enable cups logrotate.timer
```

Trusted Platform Module

The Trusted Platform Module (TPM) is an international standard for a secure cryptoprocessor, which is a dedicated microprocessor designed to secure hardware by integrating cryptographic keys into devices.

In practice a TPM can be used for various different security applications, for example in conjunction with Secure Boot, encryption key storage and random number generation.

To start using a TPM, it has to be present on your hardware (an fTPM or a dedicated device connected to a header on your mainboard) and enabled in the device's firmware settings.

List of Platform Configuration Registers

Platform Configuration Registers (PCRs) hold cryptographic hashes that record the precise state of the system. While these registers can be read at any time, they can only be modified via an "extend" operation, blending new data with the preceding hashes to create an unalterable, cumulative cryptographic chain.

By verifying these measurements between boots, the system can detect unauthorized hardware or software tampering, for example in the scenario of an Evil Maid attack. If any component has been tampered with (i.e. the hash chain deviates from a previously known good configuration), the TPM will not release its secrets. Once the PCRs prove that the correct, untampered operating system has booted, they can be safely trusted to automatically unlock your disk encryption keys.

The UAPI Group describes the TPM PCRs as follows:

PCR	Measured Component	Notes
PCR0	Core System Firmware	BIOS/UEFI executable code; changes on firmware update.
PCR1	System Firmware Config	Firmware settings, hardware config, and boot order.
PCR2	Extended Executable Code	Option ROMs of PCIe cards, GPUs, RAID controllers, etc.

PCR	Measured Component	Notes
PCR3	Extended Configuration	Device data from the UEFI Boot Device Select (BDS) phase.
PCR4	Boot Manager Code	Bootloader binary (e.g. GRUB, `systemd-boot`) and boot attempts.
PCR5	Boot Manager Config	Bootloader configuration and the GPT partition table.
PCR6	State Transitions	ACPI power states and S4/S5 suspend/resume events.
PCR7	Secure Boot State	Secure Boot status and database keys (PK, KEK, db, dbx).
PCR8	Kernel Command Line	Literal kernel parameters passed by the bootloader.
PCR9	Initramfs / Initrd	External initramfs/initrd image passed by the bootloader.
PCR10	IMA	Linux Integrity Measurement Architecture (IMA) log.
PCR11	Unified Kernel Image (UKI)	UKI components (kernel, initrd, cmdline) via `systemd-stub`.
PCR12	Boot Credentials / Overrides	Systemd boot credentials and UKI command-line extensions.
PCR13	System Extensions	System extension (`sysext`) and config (`confext`) overlays.
PCR14	Shim / MOK	Shim bootloader binary and Machine Owner Key (MOK) certificates.
PCR15	System Identity	Operating system state (machine ID, LUKS keys, root UUID).

For further details on how PCR 11-13 are used, see `systemd-stub(7)`.

Usage

TPM2-based LUKS key

The most common practical application for a TPM on Linux is securing LUKS disk encryption. Integrating the TPM chip directly into a LUKS setup ties the security of the encrypted volume to the physical integrity of the hardware. This integration is handled via `systemd-cryptenroll`, which offers the option to either fully automate decryption at boot or require a hardware-backed PIN.

During enrollment, `systemd-cryptenroll` generates a unique, secure unlock key and sends it to the TPM alongside a policy tied to specific PCR states which record the boot configuration of the system. The TPM encrypts (or "seals") this key using its own unique hardware, producing a protected data blob. This blob is then stored directly inside the LUKS volume's metadata header.

At boot, `systemd-cryptsetup` passes this blob back to the TPM. The TPM immediately measures the current, real-time PCR values to ensure no firmware, bootloader files, or core configurations have been altered. If the current PCR measurements match the policy bound to the blob, the TPM decrypts the blob and unlocks the LUKS partition; either instantly or upon the entry of a designated PIN. If a mismatch is detected, indicating the boot path has been tampered with, the TPM will deny the decryption request, making the system fall back to other configured methods of unlocking the volume.

> **WARNING:** When using this method on your root volume, there are a few caveats to be aware of.
>
> Provided the PCR banks you chose to seal against are considered valid by the system, the TPM will automatically unlock the LUKS volume at boot without the need to enter a passphrase.
>
> However, this also means that in case of theft, it's as if you didn't encrypt the system to begin with. Since the computer just has to be booted up to gain access to the decrypted data, all benefits of data-at-rest encryption are nullified.
>
> It is therefore **strongly recommended** to at least pass the `--tpm2-with-pin=yes` option to `systemd-cryptenroll` to still have a mechanism for user authentication. Despite the name of the option, alphanumeric values can be supplied during interactive enrollment.
>
> Also, make sure the system is in the exact state it should be for normal booting, i.e. the entire system is exactly set up the way you want it and there will be no further changes to core components, especially Secure Boot keys.

Enrolling a new key

Certain preconditions are necessary to use TPM2 in conjunction with a LUKS encrypted volume:

- `tpm2-tss` must be installed
- the volume uses LUKS2 encryption (default when using `cryptsetup`)
- the initramfs must be systemd-based (`mkinitcpio` hooks: `systemd` and `sd-encrypt`)

Install the `tpm2-tss` package from the repositories to provide the necessary TPM software stack tooling:

```
pacman -S tpm2-tss
```

List the TPM2 devices available on your machine:

```
systemd-cryptenroll --tpm2-device=list
```

> **NOTE:** If no devices are listed, make sure the TPM is enabled in your device's firmware settings. Devices from 2016 onwards usually have a TPM 2.0, as it is a requirement from Microsoft for Windows 10 certification for hardware manufacturers.

To enroll a new TPM-based key into a LUKS slot, specify the TPM device and the PCRs to seal against, followed by the LUKS volume to save a new slot to:

> **ATTENTION:** While binding a LUKS key to a broader set of PCRs increases system security by verifying more stages of the boot process, this hardening comes at the direct expense of system flexibility. On a rapid rolling-release distribution like Arch Linux, an overly restrictive PCR policy is virtually guaranteed to cause regular boot-time breakages. For instance, in a standard modern setup utilizing systemd-boot, UKIs, and Secure Boot, binding to volatile registers like PCR 4 (bootloader binary) or PCR 11 (hash of UKI) means that routine system updates will constantly alter these measurements, invalidating the automated decryption policy and reverting the boot process to a manual passphrase prompt.

> **NOTE:** In the unlikely event that your machine has more than 1 TPM device, supply its resource manager device node, e.g. `--tpm2-device=/dev/tpmrm0`.

```
systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 --tpm2-with-pin=yes /dev/disk/by-label/cryptroot
```

`systemd-cryptenroll` will require to authenticate the addition of a TPM2 token by prompting for an existing passphrase, followed by asking to enter a new TPM2 PIN. This PIN will become the new authentication method at boot. If validation fails for whatever reason, it will fall back to a regular passphrase.

Recovery key

When planning to rely only on TPM unlocking of LUKS volumes, it is also generally advised to add a recovery key, in case TPM validation fails for whatever reason.

A recovery key is generated automatically with a character set that's easy to type in while still having high entropy (brute force mitigation).

To generate a recovery key and have it saved to a slot in the LUKS device:

```
systemd-cryptenroll --recovery-key /dev/disk/by-label/cryptroot
```

Unlocking at boot

Making systemd use the TPM to unlock the volume can be done in one of two ways:

1. add kernel parameters, telling systemd which device to unlock with the TPM
2. use a `/etc/crypttab.initramfs` file to be included in the initramfs to point systemd to the correct volume

For the kernel command line, add the following:

> **NOTE:** In the unlikely event that your machine has more than 1 TPM device, supply its resource manager device node, e.g. `tpm2-device=/dev/tpmrm0`.

```
rd.luks.options=tpm2-device=auto
```

If you'd rather use a `crypttab.initramfs` file, add `tpm2-device=auto` to the `<options>` field:

```
# <name>□<device>□□□□□□□□□<passphrase>□<options>
root□□UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX□none□□□tpm2-device=auto
```

Universial 2nd Factor (U2F)

Universal 2nd Factor (U2F) is an open standard that strengthens and simplifies two-factor authentication (2FA) using specialized USB or NFC devices based on similar security technology found in smart cards.

For support of U2F in major web browsers and system authentication install the following packages:

```
pacman -S libfido2 pam-u2f
```

Generate U2F key for PAM

> **NOTE:** Generate keys as a regular user!

To start using a U2F key for system-level authentication, keys need to be created first.

The default directory these keys will usually be looked for is at `~/.config/Yubico` (since the `pam-u2f` package is developed by Yubico for use with their Yubikeys, but it works with other keys as well).

Create the directory under your home directory:

```
mkdir ~/.config/Yubico
```

The `pam-u2f` package comes with a utility to create keys from the USB device. Create new keys with `pamu2fcfg`:

> **WARNING:** This takes your machine's current host name and assumes it is not re-assigned on network changes! Changing your machine's host name might render the key unable to authenticate you until your machine returns to the original host name.

> **NOTE:** Keep an eye on your hardware security token, as it might silently indicate it is waiting on user interaction to continue.

```
pamu2fcfg -o pam://$HOST -i pam://$HOST > ~/.config/Yubico/u2f_keys
```

System-wide U2F prompts

> **ATTENTION:** A potentially undesirable side effect of this method is that any keychains that use the user password to unlock, such as the login keychain in GNOME or KDE, will immediately request the password after login. Since this allows passwordless logins, the user password for unlocking will not be passed on to the secrets provider. If you depend on automatic unlocking of the login keychain, e.g. for SSH key passphrases or Wi-Fi passwords, see one of the other methods below.

To use your physical security key system-wide and not just for specific use-cases, add the following line **before** the first `auth` line in `/etc/pam.d/system-auth`:

> **NOTE:** Be sure to replace `hostname` with the actual host name of your machine!

```
auth            sufficient      pam_u2f.so cue origin=pam://hostname appid=pam://hostname
```

This will prompt you to touch your physical security key during every attempt at authenticating with your user, whether it's in conjunction with graphical system administrator prompts, `sudo` prompts, display manager login prompts, TTY logins, etc.

If the security key is not connected, the system will fall back to regular password prompts.

Passwordless `sudo`

> **WARNING:** Changes to PAM configuration files apply immediately! Before making any changes to your configuration, start a separate shell with root permissions (e.g. `sudo -s`). This way you can revert any changes if something goes wrong.

A U2F key can be set up for `sudo` to allow for passwordless system maintenance tasks in the terminal.

Open `/etc/pam.d/sudo` and add the following line **before** the first `auth` line:

> **NOTE:** Be sure to replace `hostname` with the actual host name of your machine!

```
auth            sufficient      pam_u2f.so cue origin=pam://hostname appid=pam://hostname
```

To test, open a new terminal and type `sudo ls`. Your key's LED should flash and after clicking it the command is executed. The option `cue` causes an instruction to appear on what to do, e.g. `Please touch the device`.

Note that setting this does not include graphical prompts to elevate privileges in desktop environment such as GNOME or KDE. See the following section for these types of use cases.

Passwordless Polkit

Many graphical applications rely on Polkit to elevate privileges. Polkit can be set up for passwordless authentication in much of the same way as `sudo`.

By default, there is no Polkit PAM configuration present. To add it, copy the default configuration file that comes with Polkit into the PAM system configuration directory:

```
sudo cp /usr/lib/pam.d/polkit-1 /etc/pam.d/polkit-1
```

Then edit `/etc/pam.d/polkit-1`, adding the following line **before** the first `auth` line in the file:

> **NOTE:** Be sure to replace `hostname` with the actual host name of your machine!

```
auth       sufficient   pam_u2f.so cue origin=pam://hostname appid=pam://hostname
```

2nd factor in GDM

A U2F key can be used in addition to your password for added security.

Open `/etc/pam.d/gdm-password` and add the following line **after** the existing `auth` lines:

> **NOTE:** Be sure to replace `hostname` with the actual host name of your machine!

```
auth            required      pam_u2f.so nouserok cue origin=pam://hostname
appid=pam://hostname
```

This will require you to have your U2F physical key inserted to authenticate and log you in with your local user account.

> **WARNING:** If you lose your key you will also lose your ability to authenticate and log in to your user account. You could theoretically use `sufficient` instead of `required` but this would render the security benefits of this endeavour pointless, as the password would still be enough to gain access to your account.

Please note the use of the `nouserok` option which allows the rule to fail if the user did not configure a key or the key is not connected. The `cue` option will display a prompt to let you know the physical key is waiting for you to touch it.

Unlock LUKS container during boot

A FIDO2 key can also be used to unlock your LUKS encrypted drives. To register the key, you will need to use the `systemd-cryptenroll` utility and have a systemd-based initrd.

Run the following command to list your detected keys:

```
systemd-cryptenroll --fido2-device=list
```

Then you can register the key in a LUKS slot, specifying the path to the FIDO2 device, or using the `auto` value if there is only one device:

> **ATTENTION:** Make sure to pass the device node of your actual LUKS container!

```
systemd-cryptenroll --fido2-device=auto /dev/nvme0n1p2
```

To make systemd use the FIDO2 key for unlocking during boot, add the following option to your `rd.luks.options` list of options:

```
rd.luks.options=fido2-device=auto
```

Alternatively, if you do not want to pass this as a kernel command line option, add the option to your `/etc/crypttab.initramfs` and regenerate your initramfs after you've made changes:

```
# <name>□<device>□□□□□□□□□□<passphrase>    <options>
root□□UUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX□□none□□□fido2-device=auto
```

When booting your system, watch for the indicator on your FIDO2 hardware key prompting you to touch it.

Fingerpint Reader

Linux can utilize fringerprint readers through `fprintd`. It allows the Pluggable Authentication Modules (PAM) stack to check for registered biometric data before falling back to standard password entry.

Installation

Install the standard fingerprint imaging daemon from the official repositories:

```
sudo pacman -S fprintd
```

Enrollment

GNOME and KDE both offer fingerprint enrollment via their settings apps. Enrolling fingerprints this way also allows you to unlock your desktop session.

If you'd rather enroll via the terminal, use `fprintd-enroll`:

```
fprintd-enroll
```

Verify that the system successfully stored your fingerprint and reads the biometric data:

```
fprintd-verify
```

Configuring PAM

In order to use your fingerprint to authenticate `sudo` prompts in the terminal and on the desktop through Polkit, some additional setup is required.

`sudo`

> **WARNING:** Always keep a separate terminal window open with an active root shell (`sudo -s`) while modifying PAM configurations. There's no sanity checks in place and incorrect configurations can lock you out of your system.

Open the `sudo` configuration file:

```
sudo nano /etc/pam.d/sudo
```

Insert the `pam_fprintd.so` module as `sufficient` directly below the file header. This instructs PAM to succeed immediately if a fingerprint matches, or fail cleanly to the password prompt if it does not.

```
#%PAM-1.0
auth      sufficient pam_fprintd.so
auth      include    system-auth
account   include    system-auth
session   include    system-auth
```

Verify everything works as expected by opening a brand new terminal window and open an interactive root shell with `sudo -i`. The system should prompt you to touch the fingerprint reader with a finger you enrolled.

Polkit

To allow graphical authentication banners in GNOME and KDE to accept biometric inputs, modify the Polkit configuration:

```
sudo nano /etc/pam.d/polkit-1
```

Replicate the structure used for `sudo`:

```
#%PAM-1.0

auth       sufficient   pam_fprintd.so
auth       include      system-auth
account    include      system-auth
password   include      system-auth
session    include      system-auth
```