Hardware

Get your gizmos up to speed

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

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`	`intel-media-driver`, `libva-intel-driver`
AMD	`mesa`	`vulkan-radeon`	`libva-mesa-driver`
NVIDIA	`mesa`, `nvidia`	`nvidia-utils`	`libva-mesa-driver`, `nvidia-utils`

Intel

For Intel integrated graphics and Intel Arc, install the following packages:

```
pacman -S mesa vulkan-intel intel-media-driver libva-intel-driver
```

AMDGPU

For AMD integrated and dedicated graphics, install the following packages:

```
pacman -S mesa libva-mesa-driver vulkan-radeon
```

NVIDIA

In the case of NVIDIA, there's the option to either use the open source Nouveau drivers, or the 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 older cards (GeForce 8xx or 9xx series or older) you should choose the Nouveau driver.

Nouveau open source driver

The Nouveau driver is included with `mesa`:

```
pacman -S mesa libva-mesa-driver
```

Additionally, NVIDIA cards after the "Tesla" line of GPUs (GeForce 8xxx, 9xxx) will need additional firmware files installed. Without these firmware files, the GPU will be stuck at the lowest performance level, because dynamic reclocking and power management of the graphics processor will not be available.

```
yay -S nouveau-fw
```

Proprietary driver

When using any NVIDIA graphics card after the "Maxwell" line of GPUs (GeForce GTX 9xx), the proprietary NVIDIA kernel module will provide the best performance for intensive graphic and video processing workloads.

NVIDIA provides two options for their GPU drivers: a closed and open kernel module.

For anything more recent than RTX 2xxx cards, recommends the `nvidia-open` kernel module, as they plan to support that one more long-term.

```
pacman -S nvidia-open nvidia-utils
```

For earlier GPUs (GTX 9xx, GTX 10xx) the closed `nvidia` driver remains available.

```
pacman -S nvidia nvidia-utils
```

Early KMS

In order to enable early KMS (Kernel mode switching) with the prorprietary NVIDIA driver, you will need to take additional steps.

The kernel modules of the proprietary driver need to be included explicitly in the `MODULES` array of your `/ets/mkinitcpio.conf` file (or a drop-in config file, e.g. `/etc/mkinitcpio.conf.d/modules.conf`):

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

Additionally, remove the `kms` hook from the `HOOKS` array. This is to prevent the unintentional loading of the `nouveau` kernel module, which will conflict with the proprietary driver.

Enable Kernel Mode Setting

Since `nvidia-ultis` version 560.35.03-5, Kernel Mode Setting (KMS) is enabled by default with NVIDIA proprietary drivers. However, when using an older version or a very old card with proprietary drivers, KMS must be explicitly enabled through a kernel command line argument at boot time, otherwise Wayland compositors may not function properly.

```
nvidia_drm.modeset=1
```

> **NOTE:** Refer to Boot Loader for how to add the parameter to your boot configuration.

To verify that kernel mode setting is enabled (in the installed system) query the sysfs info with the following command:

```
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

Port	Protocol	Service
6001	UDP	Some 3rd party AirPlay receivers use this
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
```

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

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

Trusted Platform Module

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 such as secure boot, key storage and random number generation.

TPM is naturally supported only on devices that have TPM hardware support. If your hardware has TPM support but it is not showing up, it might need to be enabled in the BIOS settings.

List of Platform Configuration Registers

Platform Configuration Registers (PCR) contain hashes that can be read at any time but can only be written via the extend operation, which depends on the previous hash value, thus making a sort of blockchain. They are intended to be used for platform hardware and software integrity checking between boots (e.g. protection against Evil Maid attack). They can be used to unlock encryption keys and proving that the correct OS was booted.

The UAPI Group describes the:

PCR	Used by	Notes
PCR0	System Firmware executable code	May change if you upgrade your firmware
PCR1	System Firmware settings	Settings like boot order, etc.
PCR2	Extended executable code	Extended or pluggable executable code (aka OpROMs)
PCR3	Extended executable data	Set during Boot Device Select UEFI boot phase
PCR4	Boot Manager Code + Boot Attempts	Measures boot manager the devices the firmware tried to boot from
PCR5	Boot Manager Configuration + Data	Can measure configuration of boot loaders; includes GPT Partition Table

PCR	Used by	Notes
PCR6	S4/S5 Resume + Power State Events	
PCR7	Secure Boot State	Full contents of PK/KEK/db to validate each boot application
PCR8	Hash of kernel cmdline	Supported by grub and systemd-boot
PCR9	Hash of initrd + EFI Load Options	Kernel 6.1 might measure the kernel cmdline
PCR10	IMA	Protection of the Integrity Measurement Architecture measurement log
PCR11	Hash of Unified kernel image	ELF kernel image, embedded initrd and other payload of the PE image
PCR12	Overridden kernel cmdline	Will be disregarded if Secure Boot is enabled and UKI has embedded kernel cmdline
PCR13	System extension images	System extensions built to extend a base system via overlay images
PCR14	shim	MOK certificates and hashes
PCR15	LUKS key, machine ID, mount points	Root FS encryption key, machine ID, UUID of root volume, FS mounts, UUIDs, etc.

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

Packages

Package	Usage
`tpm2-tss`	Implementation of the TCG Trusted Platform Module 2.0 Software Stack (TSS2)
`tpm2-tools`	Trusted Platform Module 2.0 tools based on `tpm2-tss`
`tpm2-abrmd`	**A**ccess **B**roker and **R**esource **M**anagement **D**aemon
`tpm2-tss-engine`	OpenSSL engine for Trusted Platform Module 2.0 devices
`tpm2-pkcs11`	PKCS#11 interface for Trusted Platform Module 2.0 hardware
`tpm2-totp`	Attest the trustworthiness of a device against a human using time-based one-time passwords

Configuration

1. Add user to the `tss` group

   ```
   sudo usermod -aG tss $USER
   ```

2. Enable access broker

   ```
   sudo systemctl enable --now tpm2-abrmd
   ```

3. Logout and login again

Usage

TPM2-based LUKS key

You can use the trusted platform module in your computer as a key store to unlock LUKS encrypted volumes with them.

Arch Linux comes with `systemd` which itself comes with `systemd-cryptenroll` and allows you to specify a TPM2 device for key storage.

Using a TPM for this purpose automates the process of unlocking your LUKS volumes, given that certain conditions are met, e.g. the firmare of the machine and the Secure Boot state.

> **WARNING:** When using this method on your root volume, there are a few caveats to be aware of.
>
> Provided the PCR slots 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 password.
>
> However, this also means that in case of theft, the data is no longer protected by the encryption. Furthermore, this also makes you more vulnerable to cold boot attacks, since the computer just has to be booted up to gain access to the decrypted data, without even the need to tamper with the device in order to crack the encryption.
>
> 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 verification (available with `systemd` version 251).

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`)

Start by getting a list of TPM2 devices available in your machine:

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

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

To enroll a new TPM-based key into a LUKS slot specify the TPM device to generate the key from and the PCRs to seal against, followed by the LUKS volume to save a new slot to (using `/dev/nvme0n1p2` as an example):

> **ATTENTION:** The more PCRs you bind to, the more hardened your setup becomes. But at the same time you can also end up with a less flexible setup — e.g. binding your TPM LUKS key to PCRs 8, 9 and/or 11 harden your system against attempts to boot a kernel image which's hashes aren't measured into these PCRs but the moment your kernel changes (i.e. you update your kernel, change initrd generation, etc.) the PCRs stop validating and trigger a passphrase or recovery key prompt!

> **TIP:** If your device only has one TPM (which is usually the case) you can supply `--tpm2-device=auto` to use the only device available.

```
systemd-cryptenroll --tpm2-device=/path/to/tpm2_device --tpm2-pcrs=0+7 --tpm2-with-pin=yes
/dev/nvme0n1p2
```

It will ask you for a PIN to enter, which you will be asked to put in every time you boot the system.

Recovery key

It is also generally advisable to let `systemd-cryptenroll` generate a recovery key, in case the key stored in the TPM doesn't validate anymore for whatever reason.

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

To generate a recovery key and have it saved to a slot in the LUKS device (using `/dev/nvme0n1p2` as an example):

```
systemd-cryptenroll --recovery-key /dev/nvme0n1p2
```

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:

> **TIP:** Again, if your device only has one TPM you can supply `tpm2-device=auto` to use the only device available.

```
rd.luks.options=tpm2-device=/path/to/tpm2_device
```

If you'd rather use a `crypttab.initramfs` file, the syntax is as follows:

```
# <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.