A Modern and Secure Desktop Setup

balint **Mar 2024**

It took me quite a while to get to this setup so I thought I would write a guide to help others and also to discuss feedback and other desktop setups.

I'm currently running NixOS on my Tuxedo Pulse 14, a Surface Pro 6, an 8-9 years old custom gaming tower PC with intel + nvidia, and a Hetzner VPS. Also had it on a ROG Ally which I ended up returning when a joystick stopped working on it.

For those who want to jump ahead and just look at the setup, **here** it is. Everything is open, except for one git-crypt encrypted file where I keep sensitive info. Feel welcome to fork the repo.

The setup is LUKS2 encrypted Btrfs root with a swapfile, auto unlock with TPM2, secureboot and a BIOS password.

It would be nice if someone could help me set up a disko partition layout, but for now I just do manual partitioning with a 1024 MB FAT32 boot partition and the rest as an encrypted BtrFS root partition. Other root partitions will also work but my personal choice is BtrFS right now.

If you already have a system installed with ext4, **here's** an easy to follow guide to convert to btrfs.

During the installation you should select disk encryption which will set it up for you. The new installer uses LUKS2 by default, however if you already have a LUKS1 encypted volume, which doesn't work with TPM unlock, **this** simple guide shows you how to upgrade it to LUKS2 and switch to a more secure algorithm after the installation while you're still on the live USB.

After rebooting into the installed system, I would start a nix-shell with git and git-crypt, clone my repo to my home directory, and unlock git-crypt with a key that I exported from another machine. The repo has a top level flake.nix which contains configurations for all of my computers based on the hostname, my home manager setup, and even the shell setup for the repo, note that using nix-shell to clone works just as well.

Assuming that this is the first time I'm bringing up a new device, I would just create a folder for it within "devices" and add a conf.nix and a hw.nix. I would copy the contents of the auto generated /etc/nixos/hardware-configuration.nix into the local hw.nix. For the conf.nix, something like this:

```
{ config, pkgs, inputs, nixos-hardware, ... }: {
```

Skip to main content

```
      ./hw.nix
      nixos-hardware.nixosModules.my-device
      ../../modules/personal.nix
    ];
    #
    networking.hostName = "<device-name>";
    # Swap
    swapDevices = [ { device = "/var/swapfile"; size = 10*1024; } ];  # 1
    #
    system.stateVersion = "24.11";  # Do not change
  }
```

The swapfile will be automatically created based on the config.

Then add a new machine config in the flake which imports this configuration.nix, like I did for the other 2 devices. If the device already had a working config and needs to be brought up after a fresh clean reinstall, then there is no need to recreate these 2 config files or to re-add it to the flake.

Then I would remove both files from /etc/nixos/ and create a symlink of my flake to /etc/nixos/flake.nix. There is also a symlink for my home manager configuration. The commands for both are in my README.

At this point we're still missing the secure boot signing keys, so the setup won't build. **Here's** a short guide to set up secureboot. First I would open a nix shell with "sbctl" and run the key generation command, then the build should already work. If you set up the symlinks, using the standard `sudo nixos-rebuild switch` works.

Enrolling the secureboot key was a different process for all my machines. On the surface I just ran the enroll command and it just worked after that. On the tower I had to turn ON the Windows 10 WHQL settings in BIOS, switch a newly appeared setting from CSM to UEFI and then put secureboot in a setup state, then reboot and run the enroll command. On the Tuxedo I had to put secureboot into "custom" mode, disable auto enroll of factory keys, and then put it in setup mode. After enabling secureboot I put a BIOS password on both machines, but I'm not sure if that significantly improves security.

After enrolling the keys and enabling secureboot, make sure to reboot and verify that it works according to the guide.

Finally I enrolled the disk decryption key in the TPM chip. To be secure, make sure to also add PCR15 with value 0 and add tpm2-measure-pcr=yes to crypttabExtraOpts.

```
systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs="0+2+3+5+7+8+12+13+1
```

Skip to main content

```
# Modify HW config
boot.initrd.luks.devices."luks-<...>" = {
  device = "/dev/disk/by-uuid/<...>";
  crypttabExtraOpts = [ "tpm2-device=auto" "tpm2-measure-pcr=yes" ];
};
```

The cryptenroll command and another command to find the name of your encrypted partition are both in my README.

At this point, you should be able to reboot with secureboot enabled and only have to enter your login password, not encryption password. Depending on your hardware, this setup is fairly secure. As long as your nix configuration doesn't install malicious software. If an attacker tampers with your unencrypted boot, secureboot will fail, and if they disable secureboot or load a live USB the TPM won't decrypt your root partition, so they won't be able to access your data. If they backup and replace your encrypted root partition with an unencrypted malicious setup and then try to unlock the backup with the TPM, it will fail the PCR 15 check thanks to binding it to value 0 and measuring the unlocked system in it after unlocking it.

If they steal your laptop, modify your boot partition, get past your BIOS password, disable secure boot, and return your laptop without you noticing, then you unknowingly turn it on, not notice that secure boot is off, then enter the encryption key to unlock your drive, then they could gain access to your data. Or using the cold boot attack…

After you confirmed that everything works, `nix-collect-garbage -d` can free up some space and make your boot menu cleaner by erasing old boot generations and unused packages. If you added `programs.nh` then `nh clean all` is more effective.

Note, the declarative email/calendar in the config doesn't work, hopefully in the future it will. I would also like to add impermanence to this setup (open to suggestions), but I'm not there yet.

🔗 Full disk encryption + TPM2

TLATER **Mar 2024**

> balint:
>
> The encryption is done via git-crypt and once it's setup it's transparent when using git.

Skip to main content out git-crypt in NixOS land, but it sounds like you've thought this through. Do you consider the fact that your secrets end up with `o+rwx` unproblematic?

Is it worth at least caveating for people following along that you shouldn't use remote builds/caches with this kind of setup? That you should not store secrets that shouldn't cross the UID barrier with git-crypt?

I still think agenix/sops-nix/scalpel should always be preferred over git-crypt for secrets in NixOS configurations…

---

balint **Mar 2024**

Hey, thanks for the feedback. What you said makes a lot of sense, and I agree, git crypt should not be used for real secrets. In this case I just used it for privacy for my email address, name, username, etc since I wouldn't want necessarily everyone who looks at my repo to see which email provider I use, though it was more of an exercise, since my name and email are visible here or on GitHub anyway.

It's a good point to avoid pushing to a public cache when your build contains plain text secrets and if there's someone else using your system they could potentially find it in the nix store, though to be fair the latter scenario sounds a bit unlikely.

The options you linked here definitely look like a better alternative for servers and similar but you can't use them for everything, eg the host name. I don't use SSH keys that much on my machines apart from GitHub and I'm not sure if I care enough about that to include it in my build.

I also thought about using Bitwarden CLI for secrets since I use it anyway for my passwords. That way the secrets don't get into the builds at all, but I imagine there can be downsides to that as well. It would at least make a lot of sense for my email configuration, since it's better for those to always be up to date, it doesn't make sense to roll it back.

TLATER **Mar 2024**

> balint:
>
> but you can't use them for everything, eg the host name.

You absolutely can, though it takes some custom module writing and runtime evaluation here and there. Especially scalpel is specifically designed to overcome those limitations.

That said, if the information is just confidential rather than secret, yep, git-crypt makes more sense!

Skip to main content

> balint:

> I don’t use SSH keys that much on my machines apart from GitHub

You can alternatively use gpg or age keys. But yeah, I appreciate that there’s a population of users for whom any encryption is a significant barrier to entry. I’ve previously talked about perhaps setting up a secret management standard that doesn’t require it, but just formalizes how secrets are stored outside the nix store.

> balint:
>
> I also thought about using Bitwarden CLI for secrets since I use it anyway for my passwords.

To be clear, we’re still talking about email addresses and other sensitive information, rather than encryption keys, passwords and other secrets in the security sense of the word here?

The home-manager modules and such simply are not designed to take that information from anything but plaintext NixOS modules, because their authors don’t consider that information confidential.

Git-crypt is an ok solution for this. The other alternative I prefer is to keep a separate flake in a private (perhaps even offline) repo that you depend on in your public flake which just holds that data - no encryption required. You can also just choose not to make your repositories public in the first place.

The “correct” solution would be to write your own modules that do treat the data confidentially, so that users can supply it any way they like (such as through bitwarden at runtime), and perhaps upstream it as an alternative.

---

balint **Mar 2024**

In the last case I meant using Bitwarden in the email configuration password command instead of adding the password to the build in any form.

---

balint **Mar 2024**

Also, do you know if it is possible to make the home manager email/calendar system work with the gnome apps?

---

bobvanderlinden **Mar 2024**

Skip to main content

I learned about TPM2 👍 Thanks for the guide! Quite useful to have a kind of checklist of all elements that go into a secure desktop.

TLATER **Mar 2024**

Never experimented with that, sorry. There's *probably* a way, but may not be implemented.

ChocolateLoverRaj **Apr 2024**

I'm going to also setup secure boot and tpm auto unlock luks. In the hacker scenario you described, should I do the following:

If nixos asks me to enter the luks password, I should reboot and check that secure boot is on. If it was turned off, then I should reinstall the boot partition?

TLATER **Apr 2024**

If it's off, someone tampered with your device. You'd wipe the boot partition, take out the hard drive, copy your data somewhere else and replace the device.

Malware in the firmware of your motherboard is a real possibility at that point, since the attacker was both motivated enough to and clearly capable of bypassing any protections the firmware had in place (and that firmware is notoriously awful at security, so you should expect it to have folded like paper).

ChocolateLoverRaj **Apr 2024**

> since the attacker was both motivated enough to and clearly capable of bypassing any protections the firmware had in place

I guess this is very unlikely, but if it does happen, I should take serious precautions before entering any passwords.

Does having easily modifyable firmware affect security? I have a Chromebook which has MrChromebox coreboot firmware and anyone can modify coreboot, build it, and then flash firmware on my Chromebook externally. I'm guessing that doing this would stop the TPM from auto-unlocking the drive, but I'm not sure. I think that will happen even when I upgrade the firmware.

Skip to main content

---

saldor **Oct 2024**

The linked repository seems to have been taken offline. Now this guide is not really usable anymore. Any alternatives?

---

balint **Oct 2024**

I updated the link just now. Also a lot of info is outdated as the new installer iso uses LUKS2 by default. Ideally I would also do the formatting with disko but I don't have enough time to figure that out 🙂 I hope it helps anyway

Edit: LUKS2 instead of TPM2

ElvishJerricco **Oct 2024**

> balint:
>
> Also a lot of info is outdated as the new installer iso uses TPM2 by default

Wait what? Did I miss something in the article about a custom ISO or something? Because the standard NixOS ISO certainly *does not* use TPM2 by default for anything.

---

balint **Oct 2024**

I'm pretty sure it does, from 24.05 at least
I just installed it on my new laptop a couple weeks ago.

Edit: sorry I meant LUKS2

---

saldor **Oct 2024**

I just successfully installed a new NixOS machine with an encrypted root (yes, the latest installer does LUKS2) and followed the **latest quickstart guide** in order to setup secure boot. Enabling the UEFI setup mode on my new Asus laptop worked the same as described for Thinkpads.

Next up, trying to see if I can enable TPM LUKS unlocking. (For me, this is not a must, but nice-to-have).

UPDATE: Setting up TPM Unlocking as described **here** worked flawlessly. Awesome.

Skip to main content

ElvishJerricco **Oct 2024**

I probably need to note that TPM2 unlocking is dangerous and difficult to do correctly. Not impossible, but difficult (and I've yet to see it done correctly in any online guide with NixOS). For instance with this guide, an attacker could decrypt the hard drive if they stole the laptop. If the attacker took out the hard drive, backed up its (ciphertext) contents, and then replaced the root partition with a malicious one for which they know the passphrase, the system would happily boot the unmodified ESP, which would then boot the malicious root partition, and they'd be running an OS that they control. Then they can plug in their ciphertext backup and use the TPM2 to decrypt it because secure boot was still honored and therefore PCRS 0,1,2,3,7 are all still valid.

And there are a variety of different ways to attack at this angle that I've explored and demonstrated. The way to defend against this is with something like `systemd-pcrphase` or `systemd-pcrlock`, where you guarantee that the TPM2 state changes before handing control over to potentially compromised code. But lanzaboote isn't set up for those specifically yet, so I use a slightly different mechanism in my system.

I also bind my disk to PCR `15:sha256=00000...` (meaning PCR 15 must have no measurements in order to decrypt), and add `crypttabExtraOpts = [ "tpm2-measure-pcr=yes" ];` (meaning that it will measure the decrypted LUKS volume key into PCR 15). So at first, PCR 15 is zero, and stage 1 will decrypt a disk. The volume key is then measured into PCR 15, so now I know that my real disk cannot be decrypted anymore. Only after this point is control handed over to a possibly malicious stage 2. But you have to be absolutely sure that this PCR 15 measurement will definitely take place before handing over control. For most systems this is as simple as ensuring the root device is `/dev/mapper/<name>` rather than `/dev/disk/by-uuid/asdf` or anything else, because the mapper path imposes the necessary systemd ordering.

Of course it'd be nice to have proper stage 2 verification to avoid all of this, **but that's significantly more difficult** , and it doesn't help in the case where you want a vendor-signed OS, so these TPM2 state changes are still necessary in the grand scheme.

EDIT: Another one of my systems does **this nonsense** , which only auto-unlocks a zvol with SSH host keys and Tailscale state on it and requires a passphrase (as pin for TPM2) for the root fs. That way I can log in remotely during stage 1 using Tailscale+SSH, know that secure boot was honored because the system was able to decrypt those keys, and manually enter a TPM2 pin that should only work for my actual root fs.

---

balint **Oct 2024**

@ElvishJerricco Thank you for the info, that's really good to know! Would you be willing

Skip to main content

to provide detailed step by step instructions for settings this up correctly?

ElvishJerricco

> balint:
>
> Would you be willing to provide detailed step by step instructions for settings this up correctly?

The trouble is that it depends on the setup. In your case, where there's just an encrypted partition with a btrfs file system on it, it's as simple as binding the volume to `15:sha256=00000...`, adding the `tpm2-measure-pcr=yes` crypttab option, and making sure to specify the root device as `/dev/mapper/<name>` instead of any other path for the device.

In the case of one of my systems that uses ZFS on LUKS, I had to make changes to the ZFS import service to make it depend on `/dev/mapper/<name>` correctly.

Jump to post

---

balint **Oct 2024**

@saldor
After recent updates the TPM2 auto unlock stopped working for me, actually. Does it work for you? Did you have to add the `crypttabExtraOpts = [ "tpm2-device=auto" ];` to make it work?

---

dinvlad **Oct 2024**

FWIW, in my case the threat model I'm protecting against is just a drive failure (which has happened to me in the past), so I wouldn't have to worry about sending it for shredding or warranty exchange. So from that perspective, TPM2 unlocking is a no-brainer and serves the intended purpose.

For those worried about physical theft though, yes it defeats the purpose, to a large extent.

ElvishJerricco

Skip to main content

> dinvlad:

> For those worried about physical theft though, yes it defeats the purpose, to a large extent.

Eh I don't entirely agree. Like I said, it *can* be done correctly. And when it's done correctly, it is as secure as the TPM2 itself is. So yes, if a thief is able to compromise the TPM2, then the purpose is defeated. But the point is that *this is a significantly difficult barrier*. You need to have an active exploit against the TPM2, or something like an electron microscope to extract its secrets. These things are indeed possible (there have been many TPM2 exploits in the past), but again the point is that it is a strong barrier, not a perfect one.

Regardless, even if you don't want to risk auto-unlocking, the TPM2 is still quite valuable. Binding your LUKS volume to the TPM2 *with a pin* provides at least the same security as a regular passphrase while also informing you if the secure boot state isn't as it should be since the pin will fail in that case. Not to mention strengthening the passphrase with the TPM2's dictionary attack lockout mode, which helps to prevent brute forcing.

Jump to post

ElvishJerricco **Oct 2024**

> dinvlad:
>
> For those worried about physical theft though, yes it defeats the purpose, to a large extent.

Eh I don't entirely agree. Like I said, it *can* be done correctly. And when it's done correctly, it is as secure as the TPM2 itself is. So yes, if a thief is able to compromise the TPM2, then the purpose is defeated. But the point is that *this is a significantly difficult barrier*. You need to have an active exploit against the TPM2, or something like an electron microscope to extract its secrets. These things are indeed possible (there have been many TPM2 exploits in the past), but again the point is that it is a strong barrier, not a perfect one.

Regardless, even if you don't want to risk auto-unlocking, the TPM2 is still quite valuable. Binding your LUKS volume to the TPM2 *with a pin* provides at least the same security as a regular passphrase while also informing you if the secure boot state isn't as it should be since the pin will fail in that case. Not to mention strengthening the

Skip to main content

passphrase with the TPM2's dictionary attack lockout mode, which helps to prevent brute forcing.

ElvishJerricco **Oct 2024**

> balint:
>
> Would you be willing to provide detailed step by step instructions for settings this up correctly?

The trouble is that it depends on the setup. In your case, where there's just an encrypted partition with a btrfs file system on it, it's as simple as binding the volume to `15:sha256=00000...`, adding the `tpm2-measure-pcr=yes` crypttab option, and making sure to specify the root device as `/dev/mapper/<name>` instead of any other path for the device.

In the case of one of my systems that uses ZFS on LUKS, I had to make changes to the ZFS import service to make it depend on `/dev/mapper/<name>` correctly.

---

balint **Nov 2024**

Not sure what was going on there but it seems after upgrading to 24.11 the tpm cryptenroll works again without any extra fuss.

---

balint **Nov 2024**

Updated my setup and the guide based on the PCR 15 suggestions.

ElvishJerricco **Nov 2024**

> balint:
>
> ```
> --tpm2-
> pcrs="0+1+2+3+4+5+7+8+9+11+12+13+14+15:sha256=0000000000000000000000000000000000000000000000000000000000000000
> ```

This is too much. PCR 12 will all change with every generation, and PCRs 4, 9, and 11 will change on certain system updates, so this will break and need re-enrolling when you do a `nixos-rebuild`. I just wouldn't bind to PCRs that you don't need. I recommend `0000000000000000000000000000000000000000000000000000000000000000`, and maybe `1+3` if you want changes to bios settings to require re-enrolling.

Skip to main content

The `systemd-cryptenroll` manpage even suggests not including `0+2` because they're (effectively) covered by secure boot / PCR 7, but I personally still bind against them because I don't want to be vulnerable to firmware downgrade attacks.

It's also worth noting that the `tpm2-measure-pcr=yes` thing will only work if you use only one disk. You'd have to get more clever to support multiple.

---

balint **Nov 2024**

Thank you for the info, it's really hard for me to understand what actually goes into which PCR even with **this** info. For example based on that wiki I wouldn't assume that 9 and 11 necessarily change from a rebuild. Would be nice to have a better overview of what goes into which and what's the reason to include or not include it.

ElvishJerricco **Nov 2024**

The manpage for `systemd-cryptenroll` has a pretty good table about what goes into each PCR. **systemd-cryptenroll**

---

balint **13d**

Seems PCR 1 changes when the PC hibernates so I removed that. I'm still testing the rest to see which ones are not stable enough.

ElvishJerricco **12d**

This isn't a testing problem. They're documented. Like I said, the manpage for **systemd-cryptenroll** , also **the uapi-group** . You don't want to just apply a blanket binding to all PCRs and hope for the best. Use the ones where the documentation indicates it makes sense for you.

---

balint **12d**

That approach doesn't work for me because based on the short descriptions I don't understand what the implications are. For example based on the description using PCR1 makes sense for me but it doesn't work with hibernation.

I think most people don't have your level of understanding what each of them does.

Description for PCR1:

Skip to main content

lata/host platform configuration; typically contains serial and

model numbers

How does that implicate hibernation?

ElvishJerricco **12d**

The reason hibernation affects PCR 1 is probably because `systemctl hibernate` writes some state to EFI variables so that it knows which device / offset to resume from.

New & Unread Topics

Topic	Replies	Views	Activity
Fetching patches from lore.kernel.org	**0**	310	**May 2024**
Chromebook fingerprint KDE unlock	**0**	148	**Jul 2024**
Rsync backups for NixOS	**3**	776	**Nov 2024**
What do I need to know to use nix and where to find more? A concise overview for new users	**1**	2.5k	**Mar 2024**
[FIX] Freeze after wake from sleep/suspend with Intel Raptor Lake and Alder Lake-P	**0**	350	**Sep 2024**

Want to read more? Browse other topics in or view latest topics.