LINUX SECURITY SUMMIT EUROPE

Please help, I’ve lost my keys

Recoverable, tamper-resistant full-disk encryption at the distributed Edge

Kobus van Schoor
DataProphet - https://dataprophet.com/

v.schoor.kobus@gmail.com
https://www.linkedin.com/in/kobus-van-schoor

dataprophet CONNECT

Who are we?

DataProphet is an advisory and technology company that helps manufacturers extract productivity gains from their factory data.

A core part of our IIoT platform offering is the Edge fleet, a **globally distributed fleet of gateway devices** ingesting factory data to the cloud.

dataprophet

EDGE

Quick overview of our Edge devices

An Edge device is a small industrial-grade Linux box that sits in factory environments.

They run on Ubuntu and we support running a variety of both containerized and direct workloads on these devices.

Some devices can run custom customer-specific workflows, most now run our standardised software (the Edge stack).

TLDL; this is a diverse set of devices that require a flexible and supporting environment.

Aim of this talk

I want to present a **framework**, backed by open-source tooling that you can **adapt to your own environment.**

The aim of this framework is to help you roll out FDE in a **maintainable, scaleable** and **recoverable** manner

Four components we need to consider

- A fully verified boot chain to resist tampering
- TPM-based unlocking mechanism to allow fully autonomous reboots
- Onboarding and recovery mechanisms
- Monitoring and alerting mechanisms to make it scale

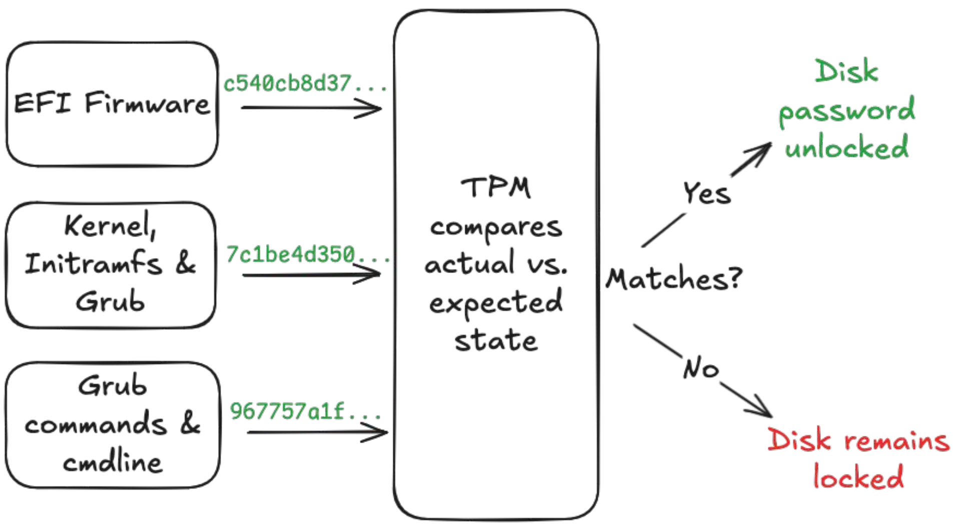
Verifying the boot chain

We need to verify the boot chain to ensure we're booting in a **trusted system state.**

We verify the boot chain by **hashing system components and config files** and comparing them to a **known state***

The TPM stores the disk unlock key and is responsible for comparing the actual system state with the expected state. If they match, it unseals the **disk encryption key**.

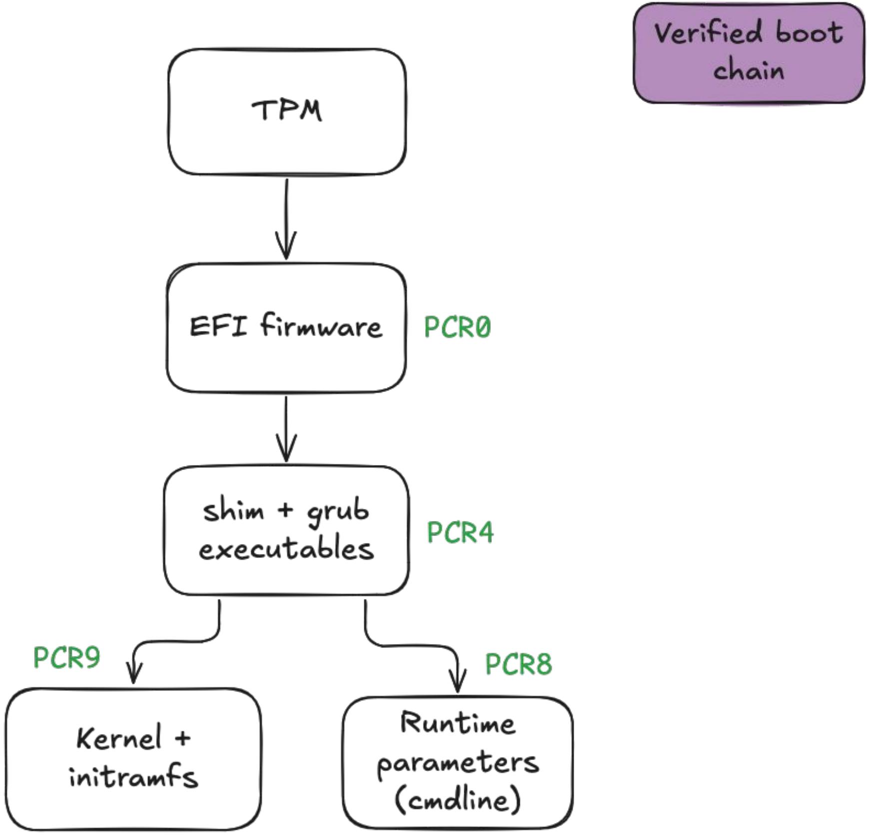
*authorized policies work a bit different, but it's the same idea



Who does all the hashing?

Each component is responsible for hashing (measuring) the next component in the boot chain.

The hashes are stored in the TPM event log.

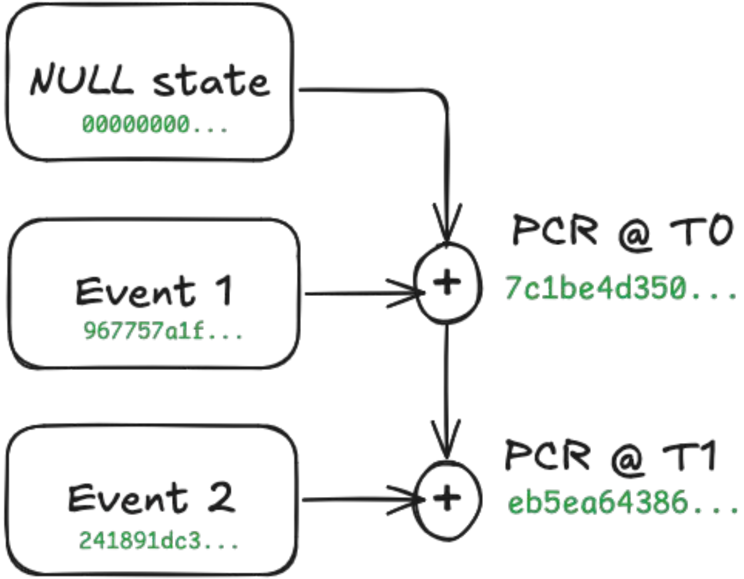


Quick overview of how the TPM event log works

Verified boot chain

The hashes are recorded in the **TPM event log** in different slots, called **PCRs**. Each PCR stores the hash of a different set of system components (UAPI reference).

The current state of each PCR is dependent on its entire event log history



Measured boot: implementation notes

Verified boot chain

For grub, you need to enable the `tpm` module to enable measurements (check the docs for `GRUB_PRELOAD_MODULES`). Note that this doesn't work with secure boot - booting with secure boot disables all sideloading of modules, including the `tpm` module.

Important note: you also need to measure the LUKS master key or taint the TPM to prevent LUKS spoofing attacks - more info here.

How do we determine what system state to trust?

TPM-based unlocking

By predicting the **PCR values of the next boot**.

Most of the time, the next boot will match the current PCR values. However they will change when you need to do **kernel and firmware upgrades**.

We can predict the PCR values of the next boot by merging the current TPM event log with some predicted values (e.g. by rehashing the new on-disk kernel).

We can then reseal the disk unlock key against the new set of predicted PCR values*.

* when using authorized policies, you'd sign the new set of PCRs instead of resealing the key

How can I predict the PCR values?

TPM-based unlocking

This is highly dependent on your environment.

- openSUSE/pcr-oracle: seems to be aimed at Suse-based environments
- systemd-measure: aimed at systemd-boot setups

We built DataProphet/pcr-predict - a simple Python script that works for Ubuntu + Grub setups.

```
$ ./pcr-predict.py

INFO:root:using kernel image
/boot/vmlinuz-5.15.0-25-generic
INFO:root:using initramfs image
/boot/initrd.img-5.15.0-25-generic
INFO:root:TPM event log has 92 entries
INFO:root:calculating digests against
which key will be sealed
INFO:root:PCR0 : c540cb8d372b0...
INFO:root:PCR4 : 7c1be4d3500ea...
INFO:root:PCR8 : 967757a1f96de...
INFO:root:PCR9 : 14684a7fa97fe...
INFO:root:PCR15 : 0000000000000...
{
  "0": "c540cb87f...",
  "4": "7c1be4d35...",
  "8": "967757a1f...",
  "9": "14684a7fa...",
  "15": "000000000..."
}
```

Unlocking the disk using the TPM

TPM-based unlocking

Unlocking the disk is made simple with latchset/clevis.

Clevis **abstracts key retrieval from various sources** (e.g. TPM, key servers)

It then also **integrates with various tools**, like LUKS & dracut, to use these keys to do useful things (like unlocking your disk).

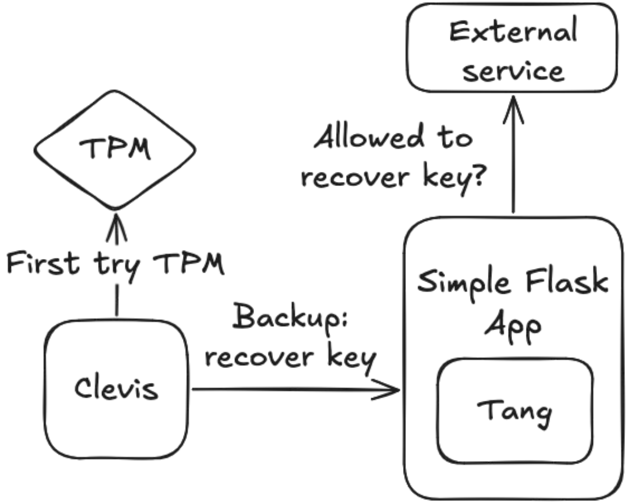
We use clevis to retrieve keys from both the TPM and our key recovery service.

Help, I lost my keys. What now?

Recovery

Eventually, something will go wrong and the TPM won't unseal your disk key. If you're really unlucky, this happens to your whole fleet.

Enter latchset/tang - a simple, stateless, key recovery service.



Reasons why Tang is great ❤️

Recovery

- **Keys can be encrypted without access to the server**
  The Tang server doesn't need to be reachable to set up the recovery key (you just need its advertisement, which can be distributed out-of-band).
- **Incredibly simple to set up, secure and maintain.**
  Stateless & zero-config. Secret-sharing crypto means compromising just the server doesn't compromise client keys.
- **Comms protocol consists of simple GETs and POSTs - TLS optional**
  The entire service only has two simple endpoints.

Recovery: what does it look like?

Recovery

If your disk isn't unlocking, it means your device will get stuck **pre-boot**, i.e. in your **initramfs environment** - a slimmed down environment with the basic tools needed to mount your disk and boot to it.

We use dracut - a modern initramfs used by Red Hat/Fedora, SUSE and a bunch of other distros (and perhaps future Ubuntu versions?).

Customising your dracut images is easy - you just need a **dracut module** (a simple bash script).

Recovery mode: staying in control

Recovery

We need to retain control of a device even if it's in recovery mode. We need:

1. **Working networking**
   This needs to work well before anything else will work
2. **Working remote access**
   Depending on your setup, this may require a VPN
3. **A way to control the device**
   SSH or some other orchestration method (e.g. Saltstack)

Dracut modules: basic overview

Recovery

The initramfs consists of a **small bootable filesystem**.

Extending the initramfs usually consists of:

- **Adding binaries and library files**
  Achieved with dracut helpers like `inst_binary`
- **Adding config files**
  They're simply copied over
- **Setting up a systemd service**
  Very similar to normal services, just with different `WantedBy` targets

Networking in the initramfs

Recovery

We use NetworkManager (along with its dracut module) + netplan to handle networking in the initramfs.

The host config is copied over into the initramfs by netplan generate'ing the configs into the initramfs root

```
# install the netplan configs
inst_multiple -o -H /etc/netplan/*.yaml
# render the network configs
netplan generate --root-dir "${initdir}"

# enable the neednet option, which enables networking in the initramfs
echo "rd.neednet=1" > "${initdir}/etc/cmdline.d/10-neednet.conf"
```

Implementation notes: networking

Recovery

Using `systemd-networkd` + `rd.neednet` made networking a hard requirement for booting which is why we ended up using `NetworkManager`

The `network-manager` dracut module had a bug (dracut#2123) which might break `NetworkManager` inside the initramfs on older versions of Debian/Ubuntu - it requires a manual workaround (we just fixed it with symlinks)

Retaining remote access

Recovery

We use `wireguard` + `ssh` to retain remote access to our devices.

```
# install the wireguard binaries
inst_binary wg
inst_binary wg-quick
inst_binary sort  # needed by wg-quick

# install the config
# NOTE the config file also includes the keys
inst_simple "${CONFIG}"

# install the wg-quick service template
inst_simple
/lib/systemd/system/wg-quick@.service

# enable the wg-quick service for the dp config
override_and_enable "wg-quick@${INTERFACE}"
```

```
# add the sshd binary
inst_binary /usr/sbin/sshd

# add the current host keys
inst_multiple /etc/ssh/ssh_host_*

# install the system ssh service
inst_simple
/lib/systemd/system/ssh.service

# enable the ssh service
override_and_enable ssh

# copy in authorized keys
inst_simple /root/.ssh/authorized_keys
```

Monitoring and alerting

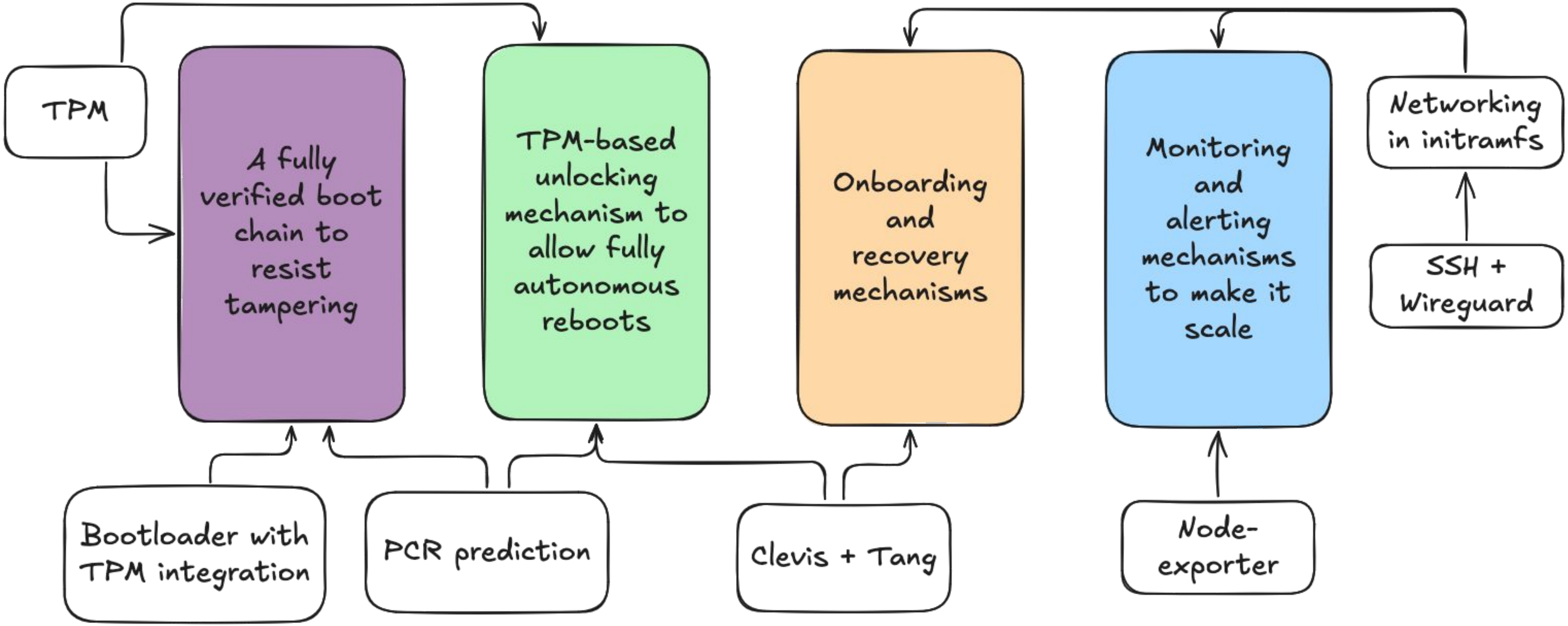
Monitoring

`node-exporter` + textfile-based metrics makes alerting easy

```
# install the node exporter binary and service
inst_binary prometheus-node-exporter
inst_simple "/lib/systemd/system/prometheus-node-exporter.service"

# add the static metrics file to indicate that the device is in recovery
# mode
cat > "${initdir}${METRICS_DIR}/recovery.prom" <<EOF
edge_fde_recovery{device="{{ device_id }}"} 1.0
EOF
```

Bringing it all together



LINUX SECURITY SUMMIT EUROPE

Thank you

If you see my keys, please let me know

dataprophet CONNECT