DIY Verified Boot in 2024

Marco Felsch – m.felsch@pengutronix.de

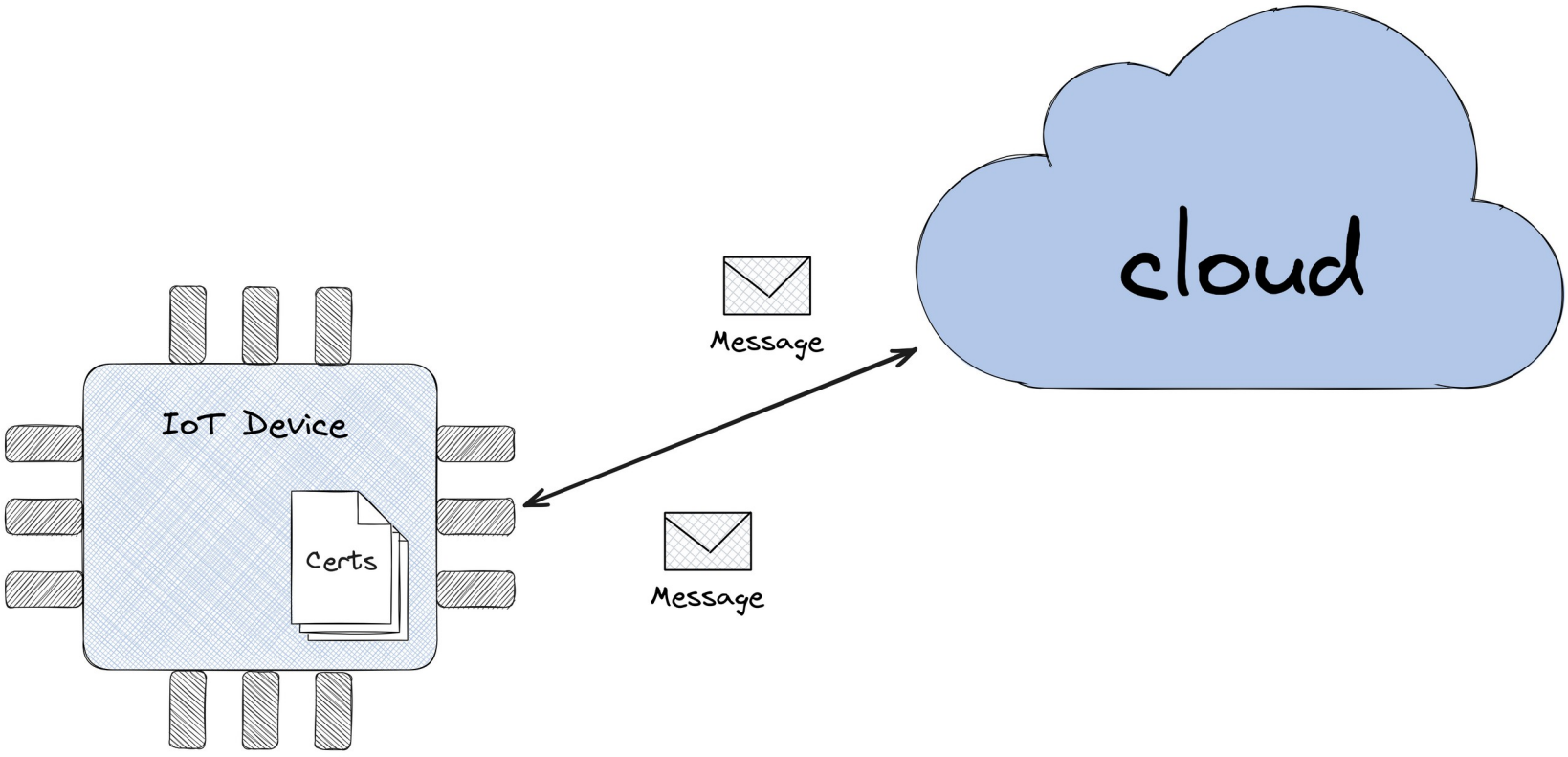
About Me

- Embedded software engineer at **Pengutronix**
  - Kernel, bootloader, graphic development
  - PTXdist/Yocto integration
- Open-Source contributor
- Vechta, Germany
- marco.felsch@pengutronix.de

Agenda

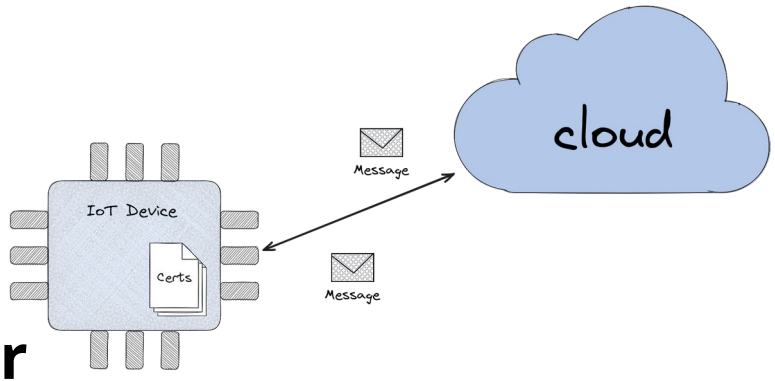
- Einführung
- Secure Speichermöglichkeiten
- Verified Boot Chain
- Umsetzung mittels Yocto

Einführung – Problembeschreibung



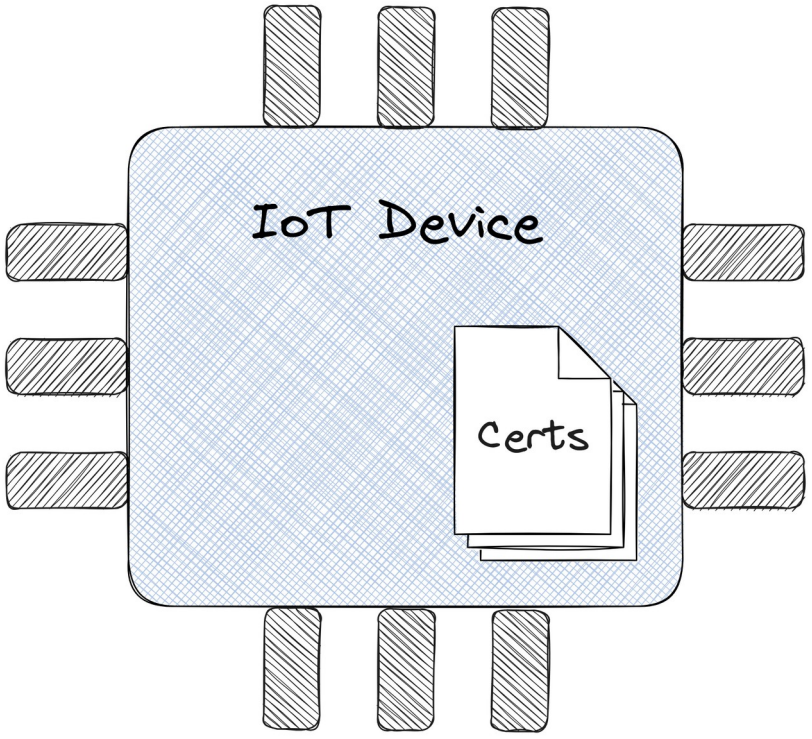
Einführung – Problembeschreibung

- (I)IoT Geräte müssen mit einer Cloud kommunizieren
- Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren
- Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren

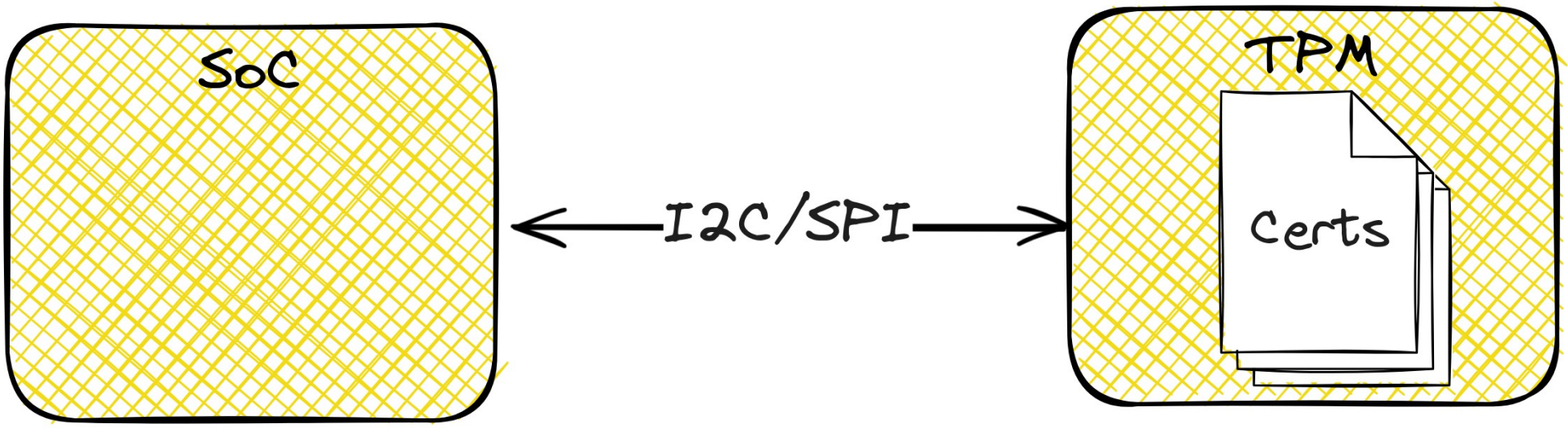


Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

Speicherung der Zertifikate

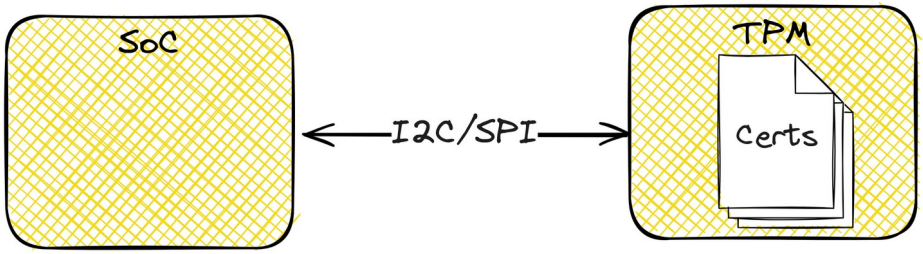


Speicherung der Zertifikate (TPM)



Speicherung der Zertifikate (TPM)

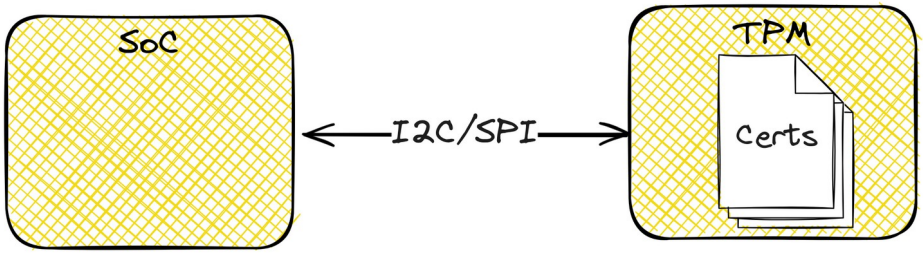
- Kommunikation nicht verschlüsselt -> abhörbar mittels Tastköpfen



- Kommunikation verschlüsselt -> der Schlüssel muss sicher auf dem SoC abgespeichert werden.

Speicherung der Zertifikate (TPM)

- Kommunikation nicht verschlüsselt -> abhörbar mittels Tastköpfen

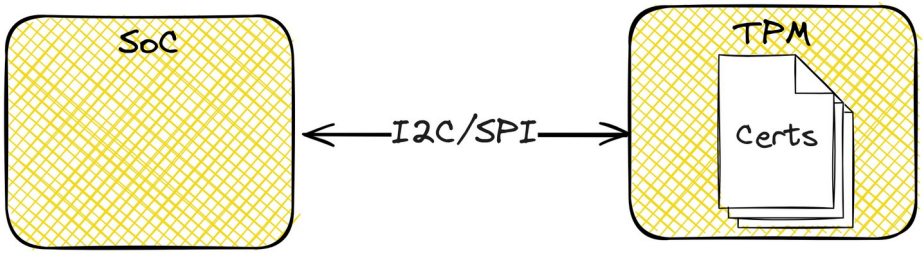


- Kommunikation verschlüsselt -> der Schlüssel muss sicher auf dem SoC abgespeichert werden.

„Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?“

Speicherung der Zertifikate (TPM)

- Kommunikation nicht verschlüsselt -> abhörbar mittels Tastköpfen

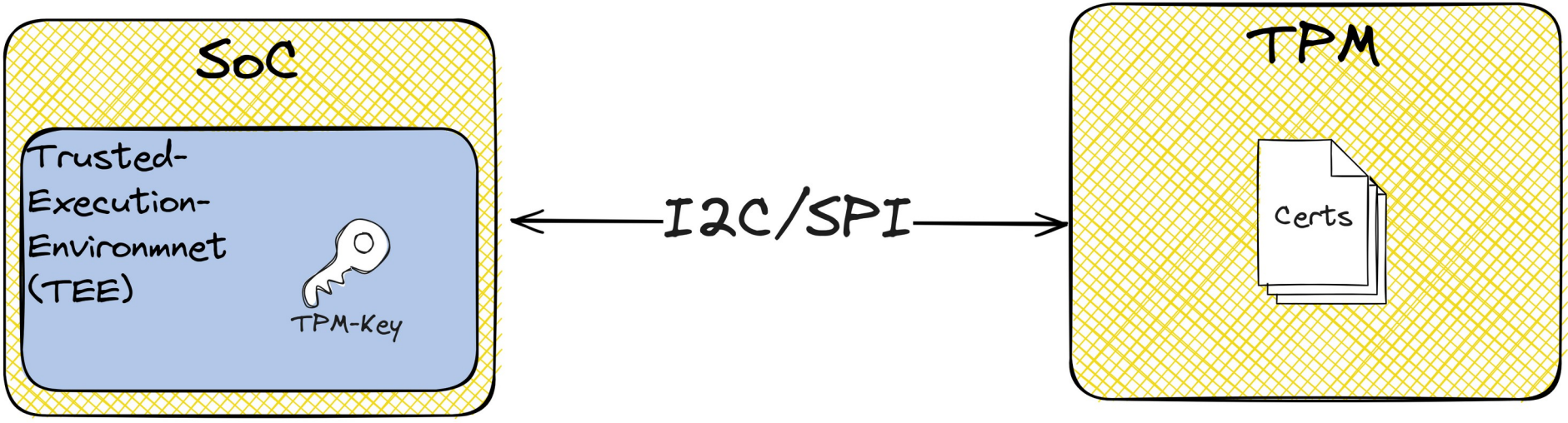


- Kommunikation verschlüsselt -> der Schlüssel muss sicher auf dem SoC abgespeichert werden.

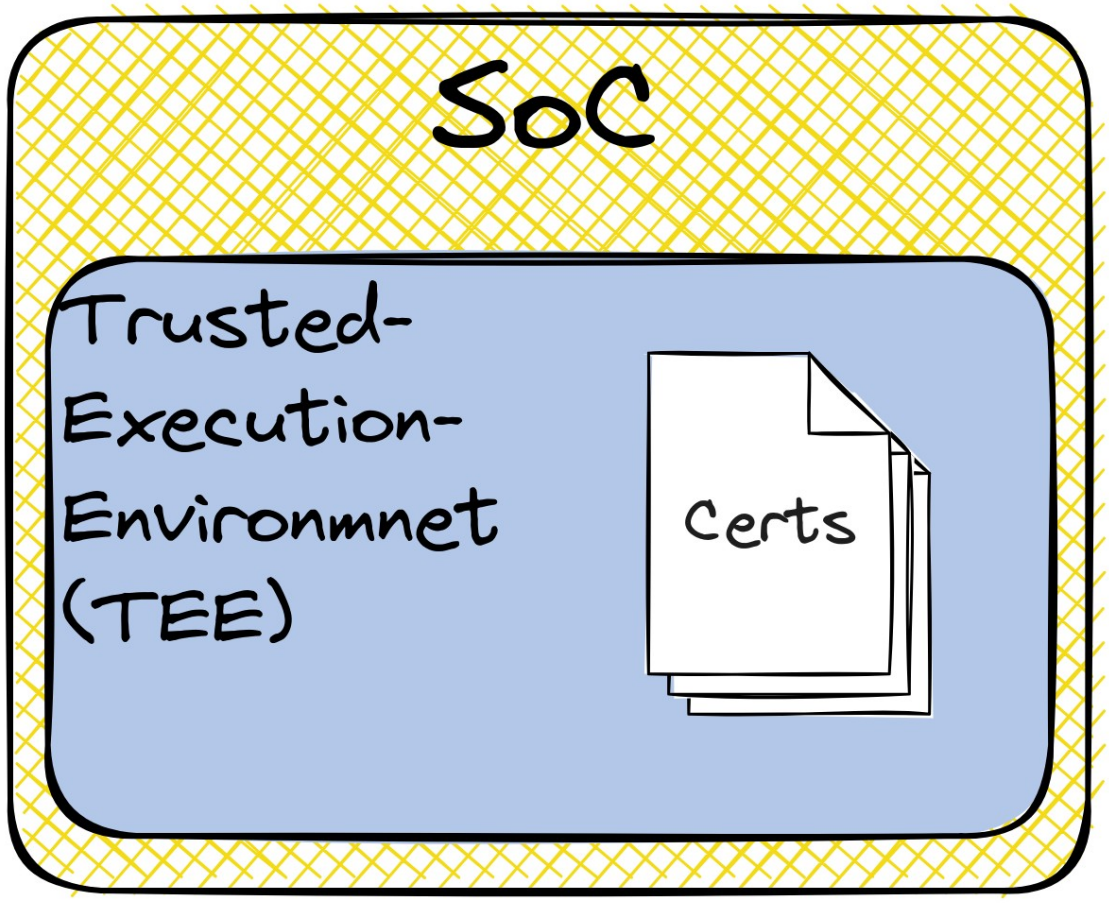
„Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?“

Wie können TPM Schlüssel sicher auf Millionen Geräten abspeichern?

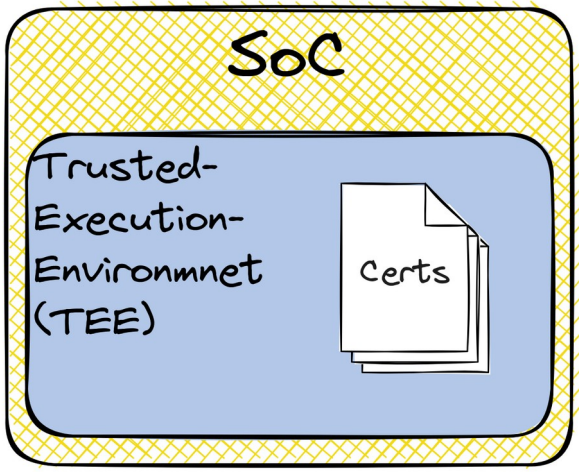
Speicherung der Zertifikate (TPM)



Speicherung der Zertifikate (TEE)



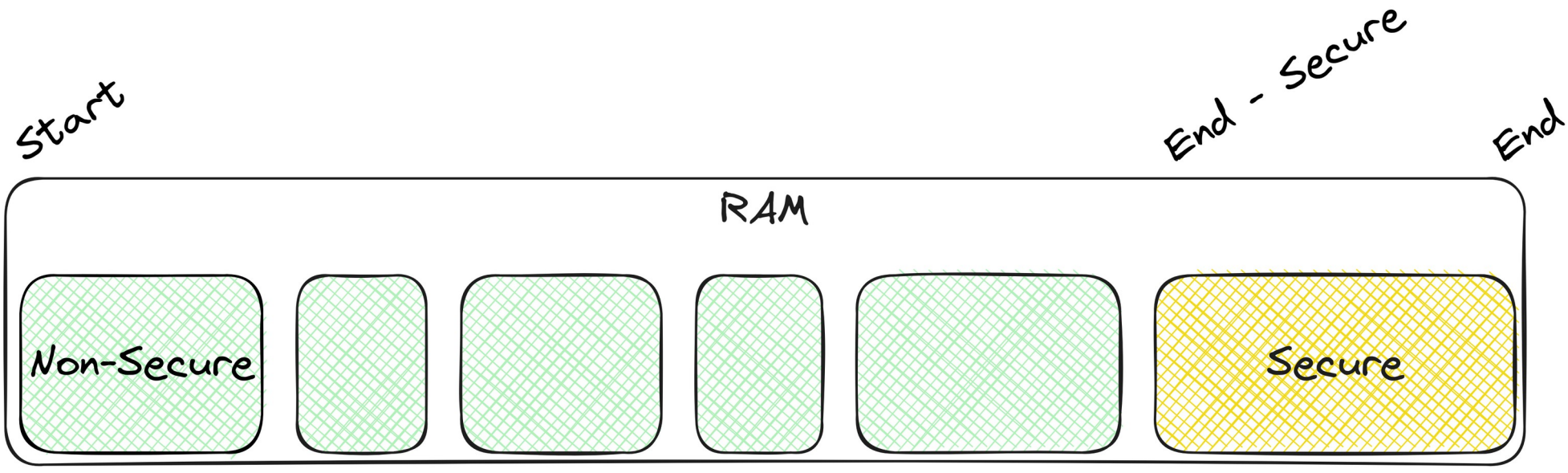
Speicherung der Zertifikate (TEE)



- TEE als Secure Speichermöglichkeit
  - Beispiel: OP-TEE(1)
- SoC benötigt die ARM TrustZone(2) Extension
- Kein zustätzliches TPM nötig
- OP-TEE bietet verschiedene Kommunikation APIs, damit die non-secure mit der secure World reden kann (Bsp.: PKCS#11)
- Microsoft hat ebenfalls eine fTPM(3) Trusted-Applikation geschrieben

(1) https://www.trustedfirmware.org/projects/op-tee/
(2) https://www.arm.com/technologies/trustzone-for-cortex-a
(3) https://github.com/microsoft/MSRSec/tree/master

Speicherung der Zertifikate (TEE)



Verified Boot Chain (Einführung)

- TEE wird wie eine Applikation in den RAM geladen
  - Versch. Methoden wie das TEE geladen wird
- Es muss sichergestellt werden, dass die TEE Software und die vorhergehende Software, vom Hersteller verfiziert und nicht verändert wurde!

Verified Boot Chain erfoderlich!

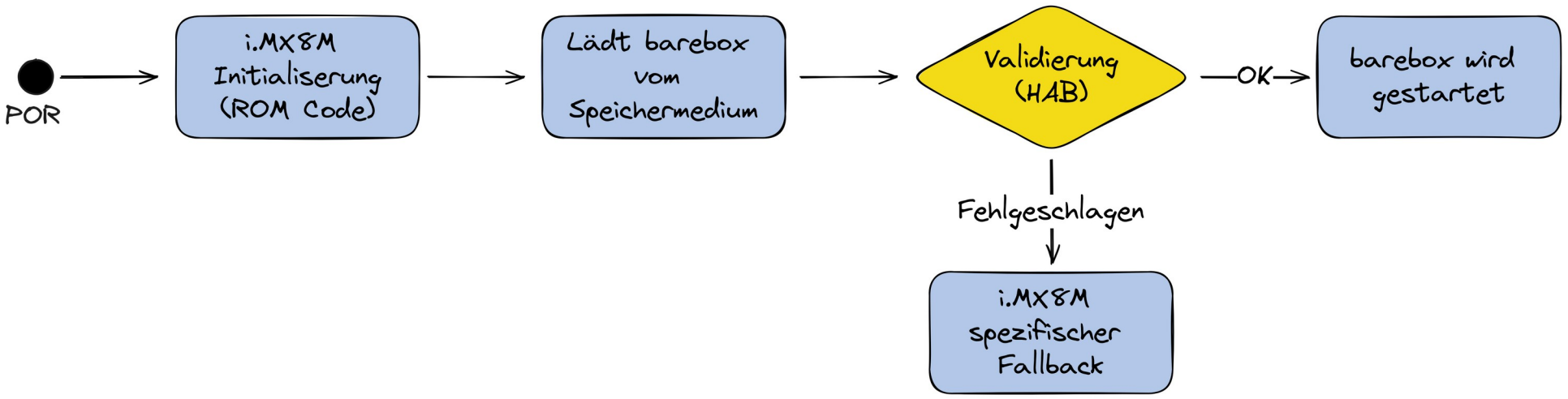
Verified Boot Chain (Einführung)

- Während des System-Boot stellt jedes Element der Kette die Authentizität des nachfolgenden Elements sicher.
- Bei erfolgreicher Verifizierung wird der Bootvorgang vorgesetzt.
- Bei fehlgeschlagener Verifizierung wird der reguläre Bootvorgang abgebrochen. Ja nach Software-Element kann ein Fallback erfolgen.

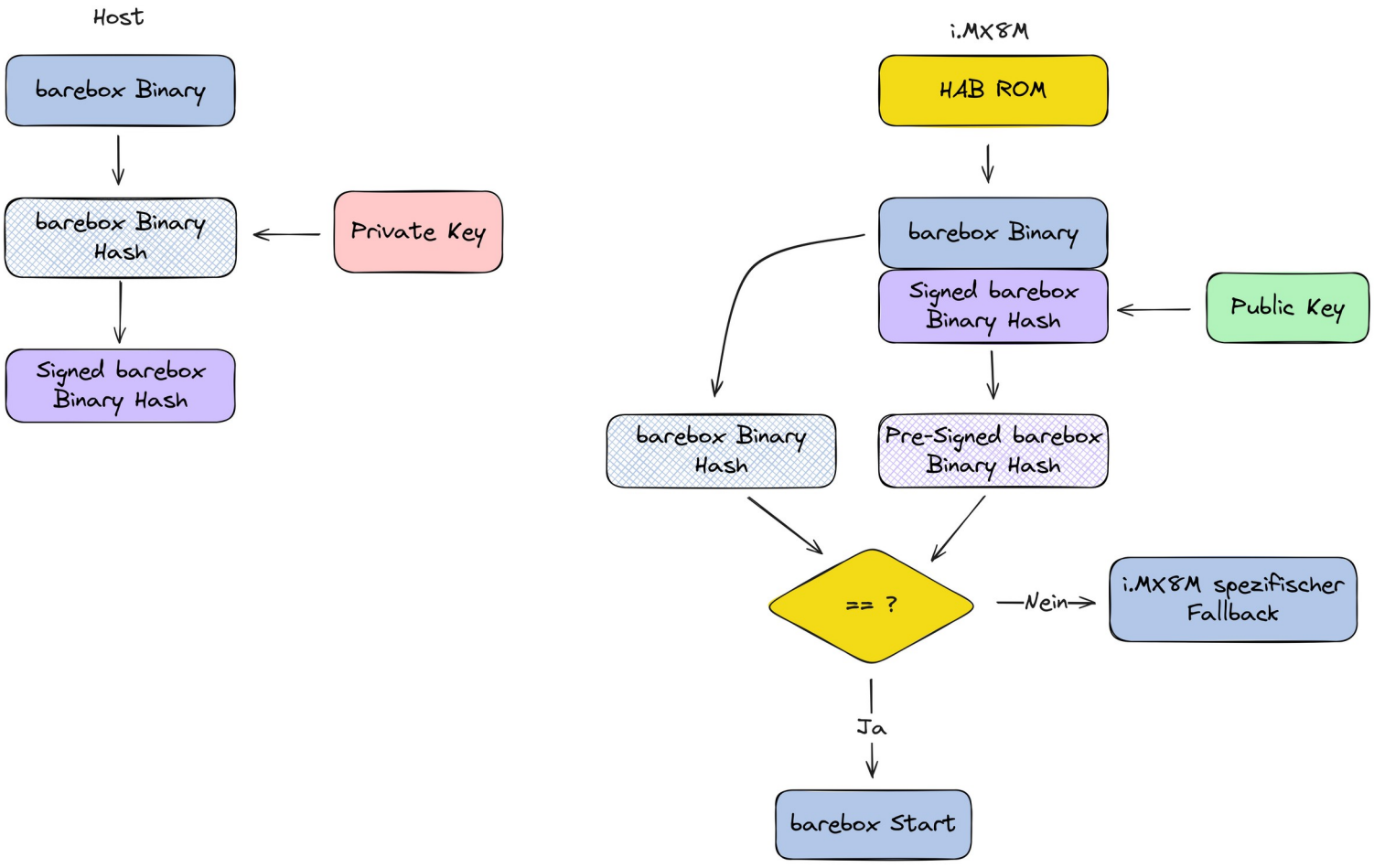
Verified Boot Chain (Beispiel)

- Das Beispiel ist eine vereinfachte Darstellung der Verified Boot Chain, die verwendeten Komponenten sind:
  - SoC: NXP i.MX8M SoC mit High-Assurance-Boot (HAB)
  - Bootloader: barebox
  - Secure-Monitor (EL3): Trusted-Firmware-A (TF-A)
  - TEE: OP-TEE

Verified Boot Chain (HAB – barebox)

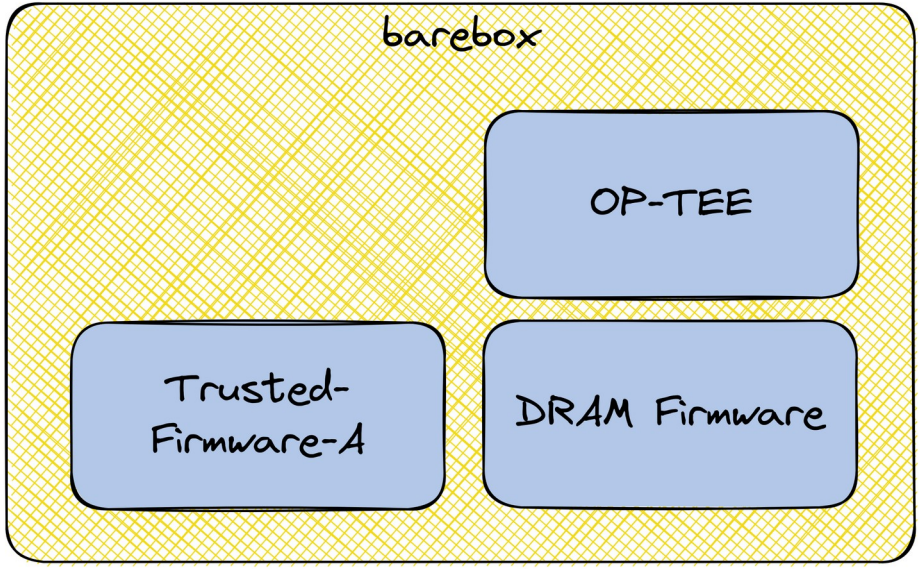


Verified Boot Chain (HAB – barebox)

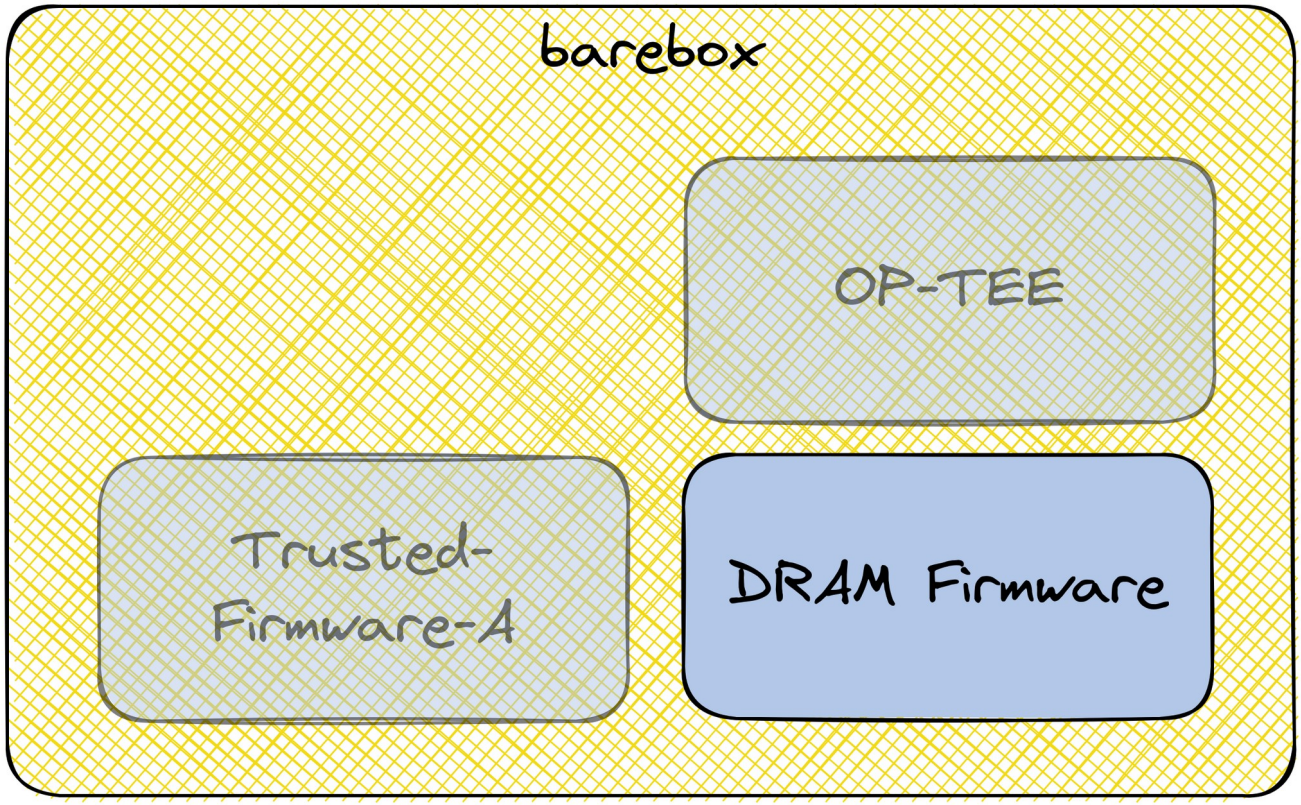


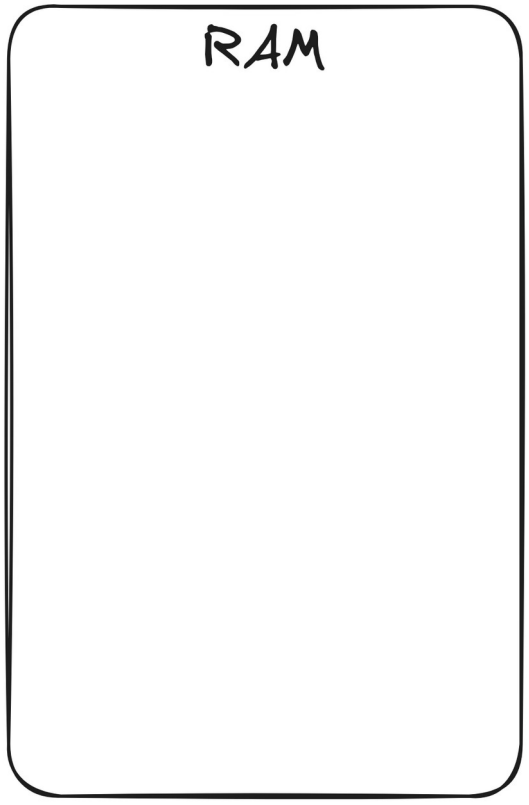
Verified Boot Chain (HAB – barebox)

- barebox beinhaltet neben dem eigentlichen Bootloader:
  - Firmware, TF-A Binary, OP-TEE Binary
- Das komplette Bunde wurde vor Ausführung vom i.MX8M HAB Code auf Authentizität überprüft, mittels pub/priv Key.

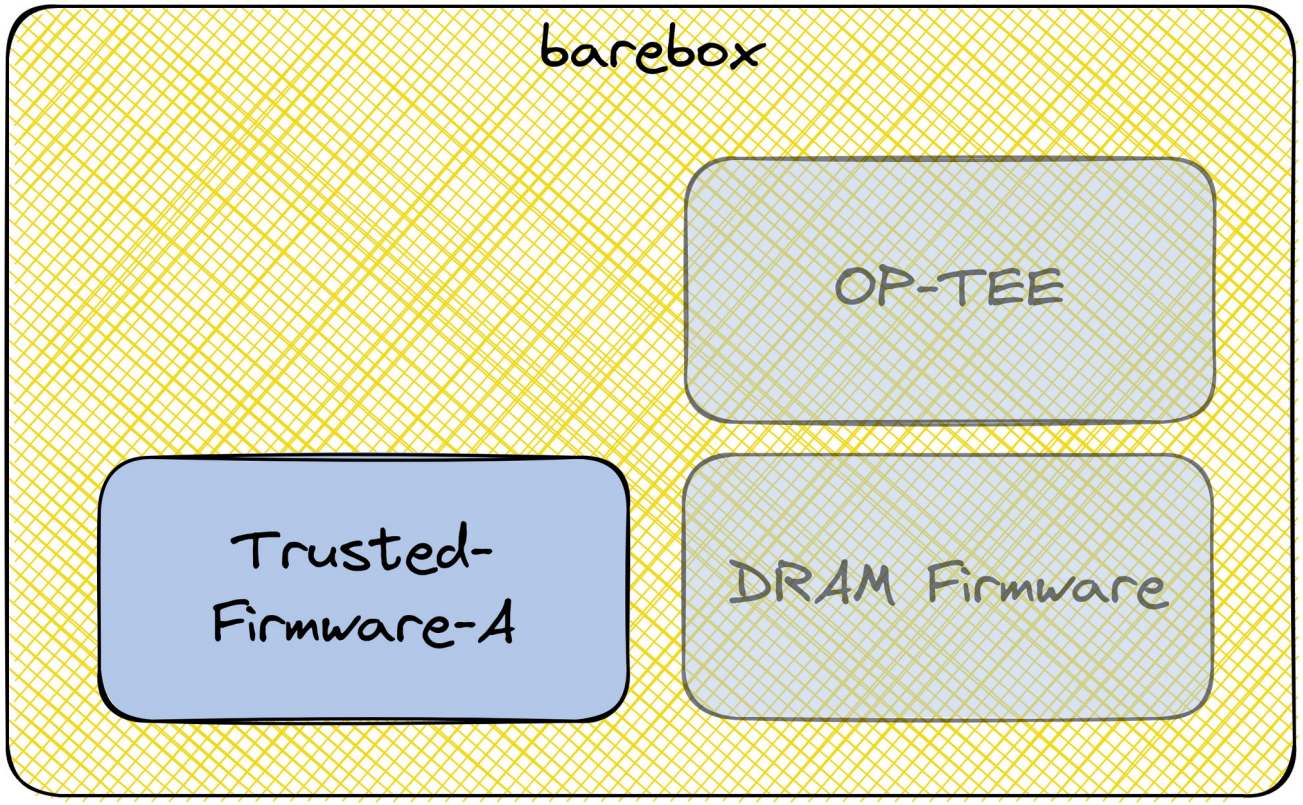


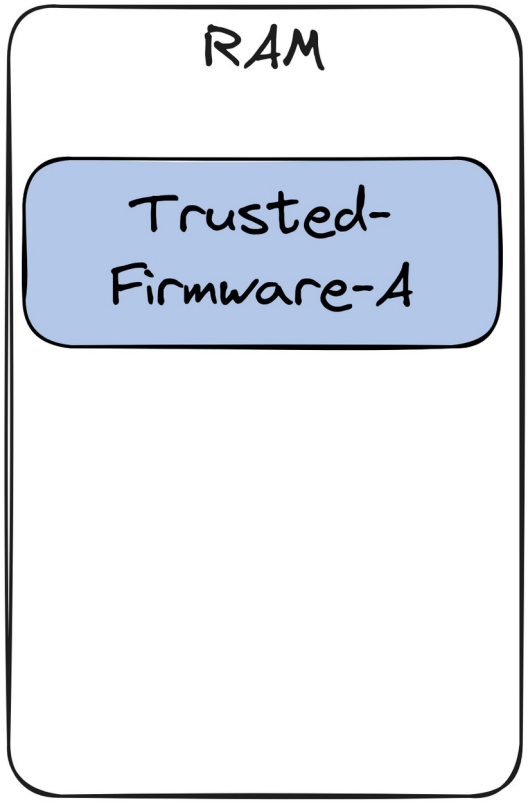
Verified Boot Chain (HAB – barebox)





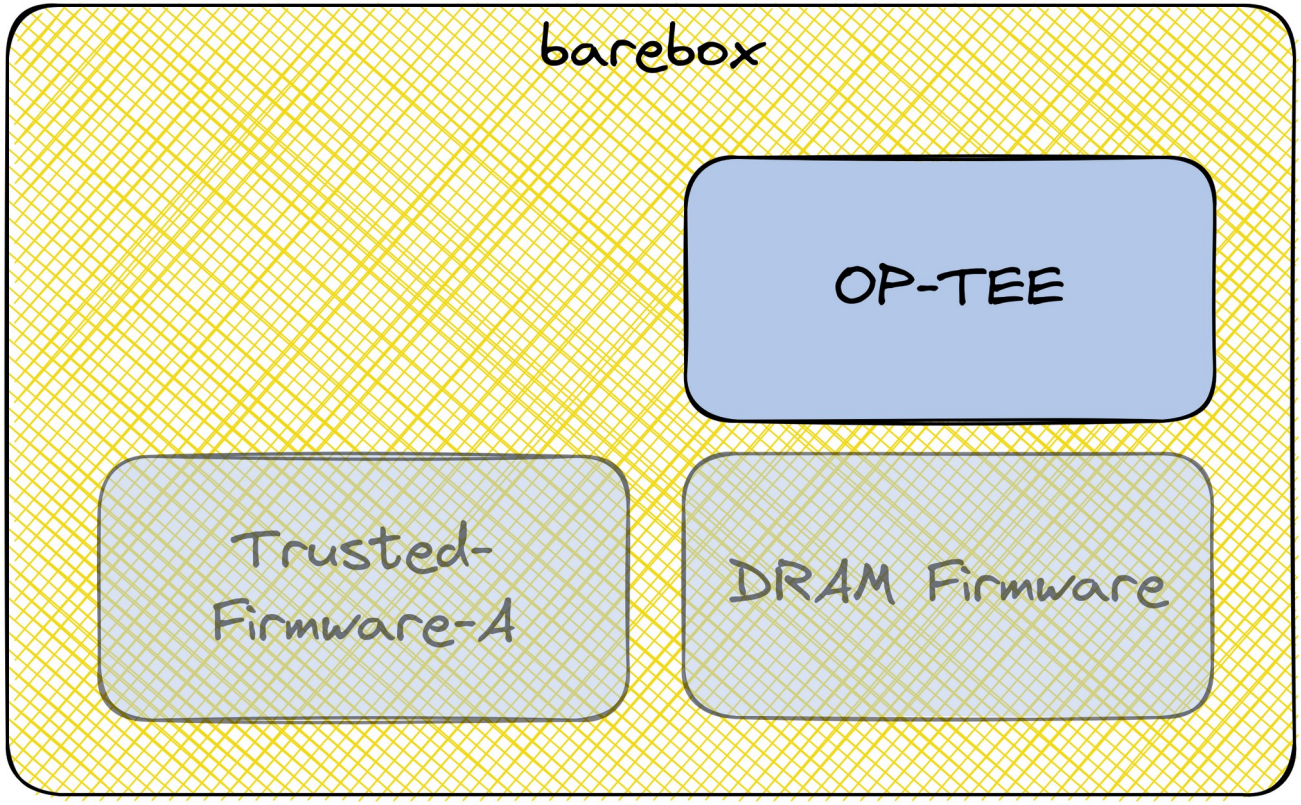
Verified Boot Chain (HAB – barebox)

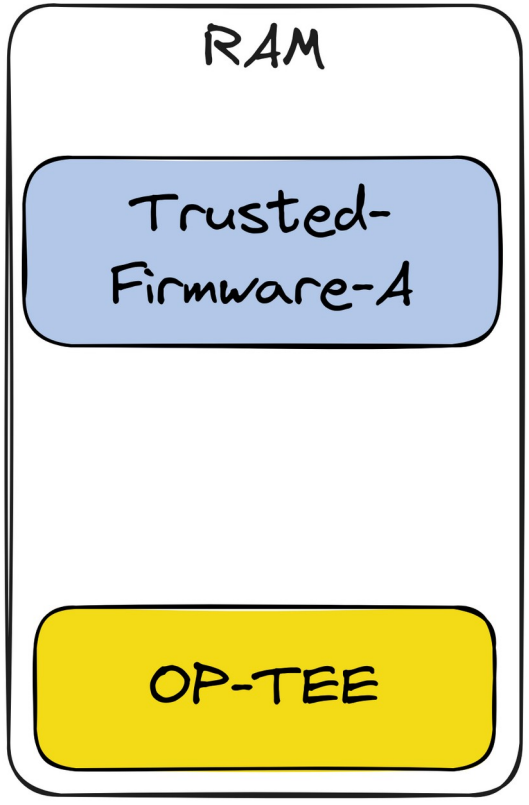




Randnotiz: Die TF-A wird in den SRAM des SoC geladen

Verified Boot Chain (HAB – barebox)



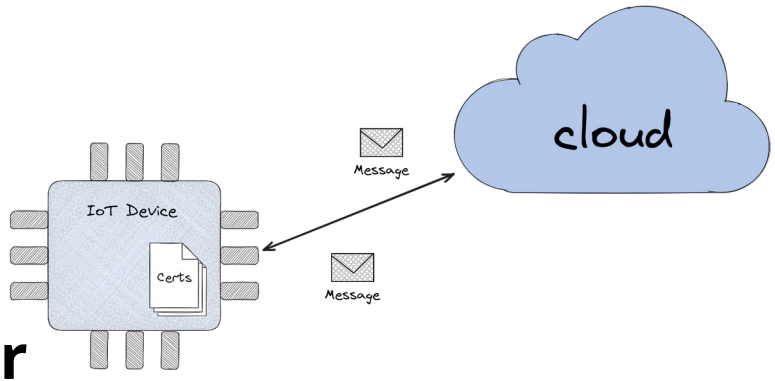


Verified Boot Chain (HAB – barebox)

- Nachdem alle Komponenten geladen und gestartet wurden, kann Linux gestartet werden
- Die Client-Zertifikate werden in der TrustZone mittels OP-TEE gespeichert

Einführung – Problembeschreibung

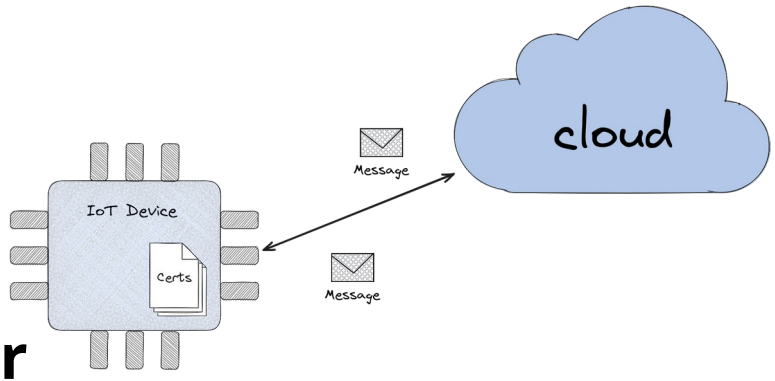
- (I)IoT Geräte müssen mit einer Cloud kommunizieren
- Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren
- Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren



Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

Einführung – Problembeschreibung

- (I)IoT Geräte müssen mit einer Cloud kommunizieren
- Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren

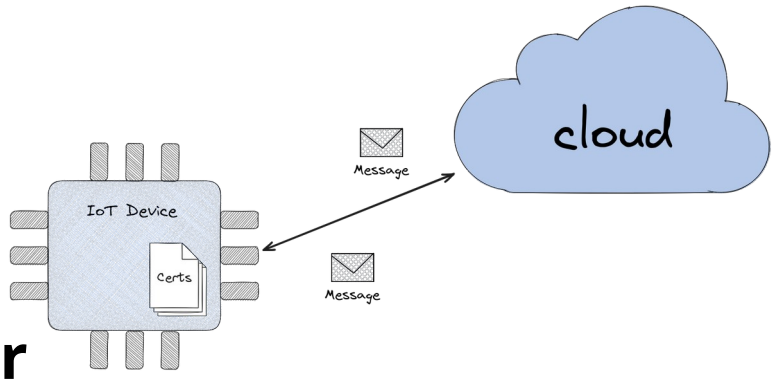


- ✅ Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren

Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

Einführung – Problembeschreibung

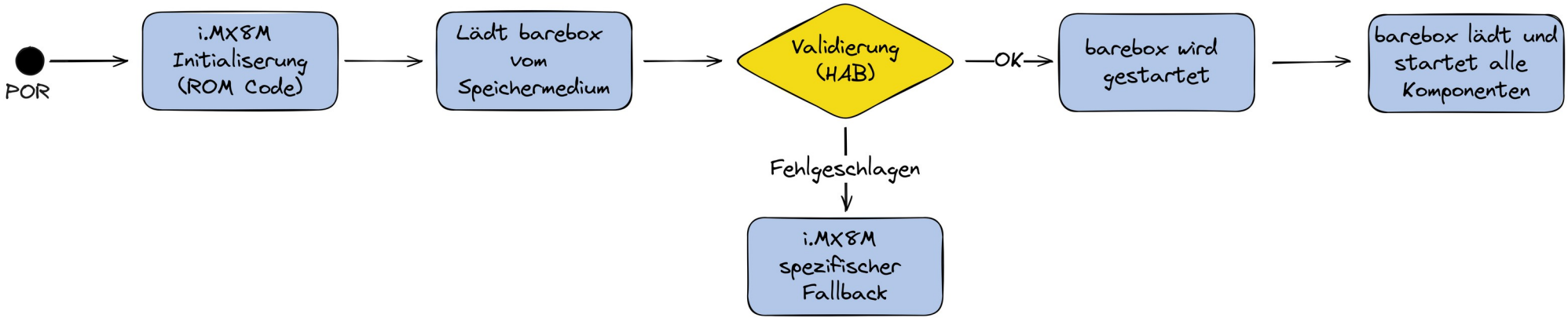
- (I)IoT Geräte müssen mit einer Cloud kommunizieren



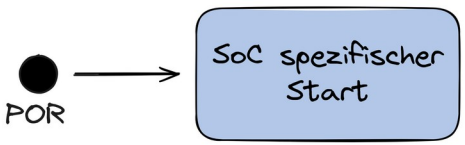
- ❌ Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren
- ✅ Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren

Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

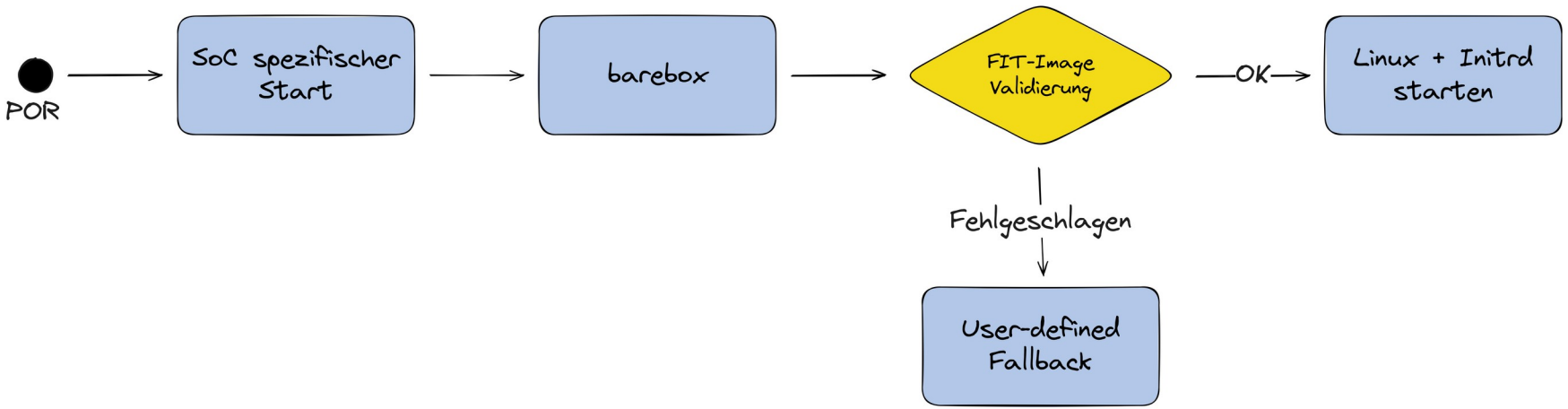
Verified Boot Chain (FIT)



Verified Boot Chain (FIT)



Verified Boot Chain (FIT)



Verified Boot Chain (FIT)

- FIT – Flat Image Tree
- Ursprung U-Boot[1], jetzt wird die Spezifikation innerhalb der Open-Source-Firmware Foundation fortgeführt[2]
- Erstellung mittels mkimage und dtc
- Devicetree Source Format zur Beschreibung der Image Bestandteile -> image tree source (.its)
- Finales Image ein Devicetree Blob, welcher Images beinhaltet -> flatten image tree blob (.itb)

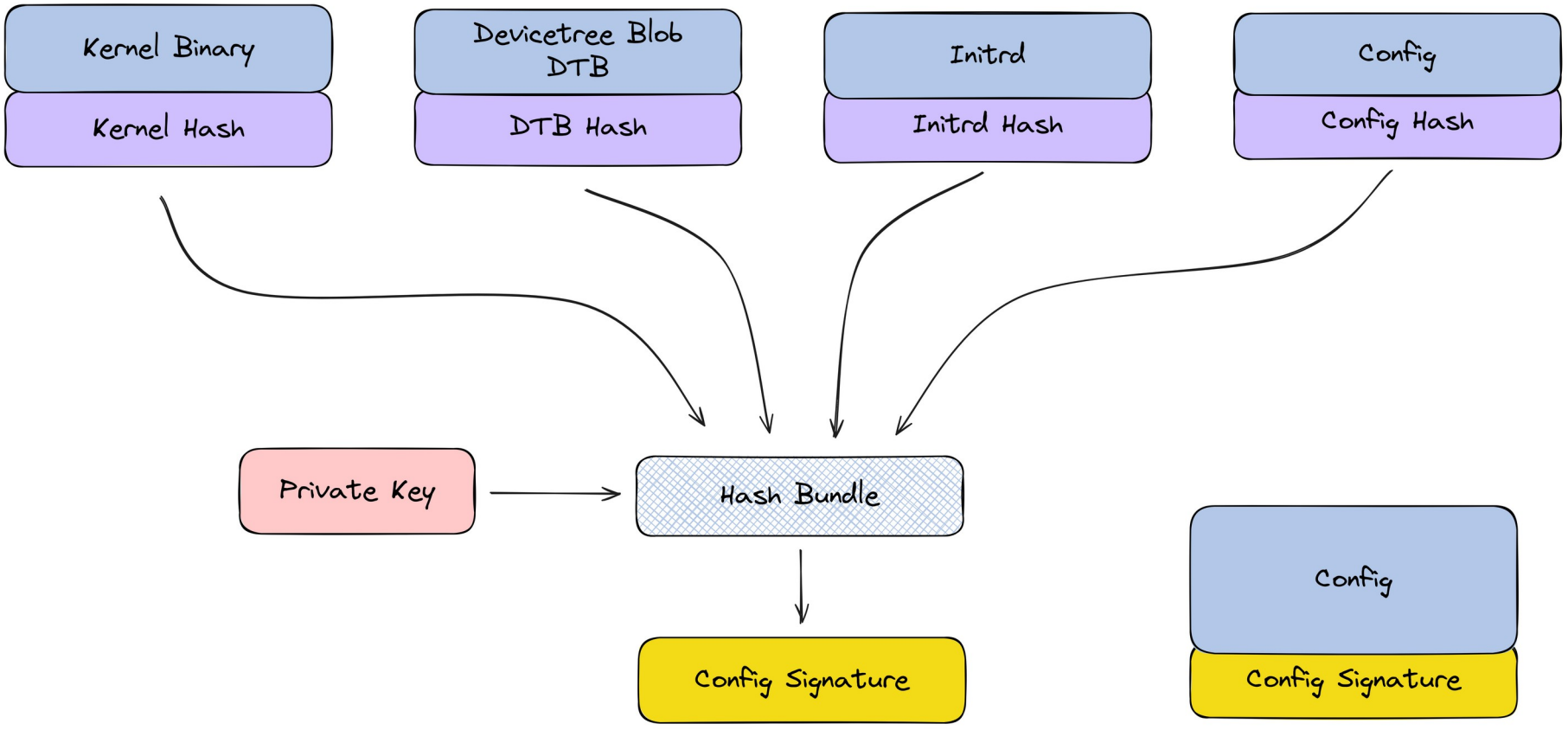
(1) https://docs.u-boot.org/en/latest/usage/fit/index.html
(2) https://github.com/open-source-firmware/flat-image-tree

Verfied Boot Chain (FIT)

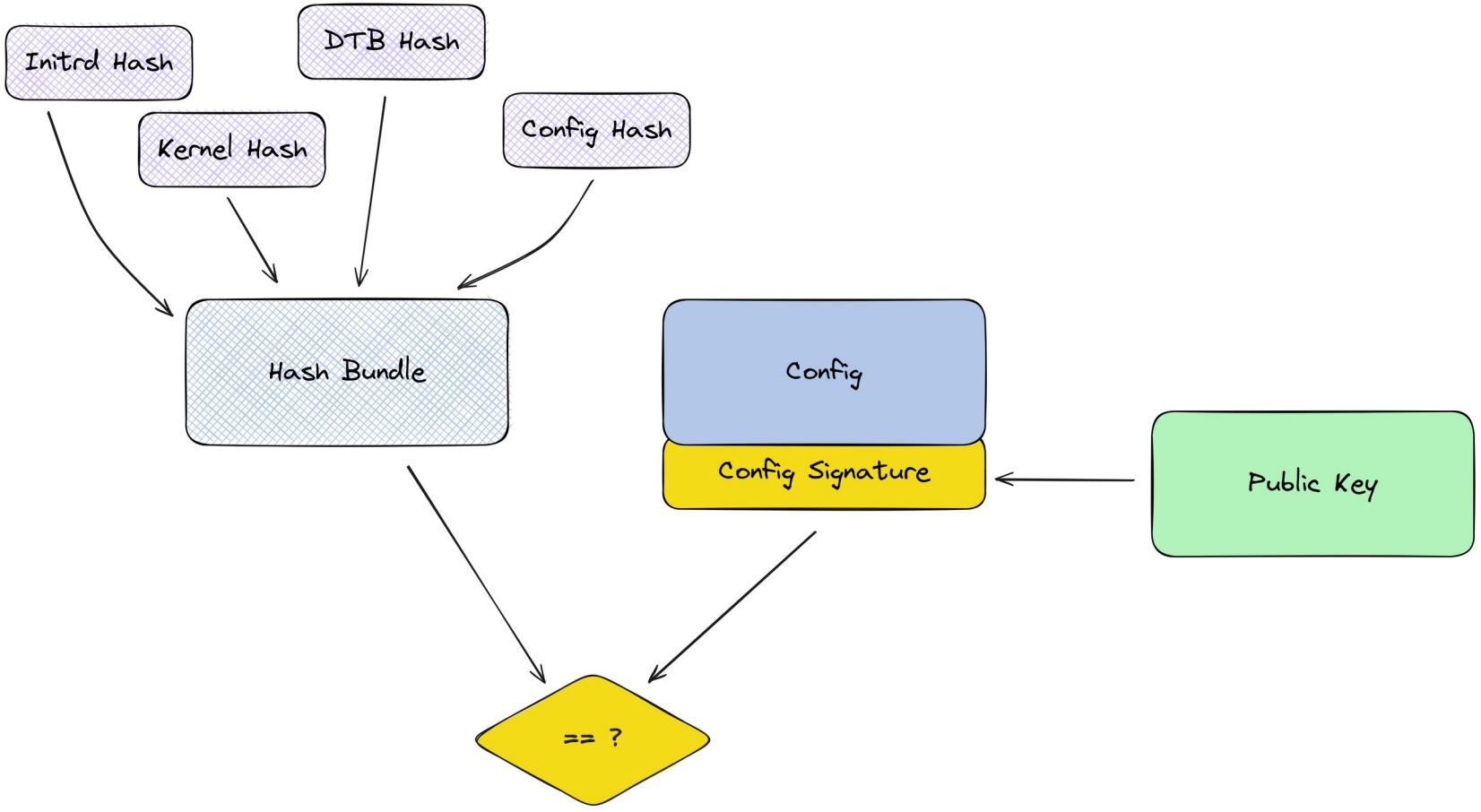
- Der Bootloader verifziert vor dem Start des Kernels alle benötigten Artefakte
  - z.B.: Kernel, Initrd, DTB, DTBO, FPGA Bitstream, ...
- Verifikation basierend auf der digitalen Signatur des FIT Images (FIT config Knoten)

→ Sicherstellung das ausschließlich der Code vom Hersteller ausgeführt wird

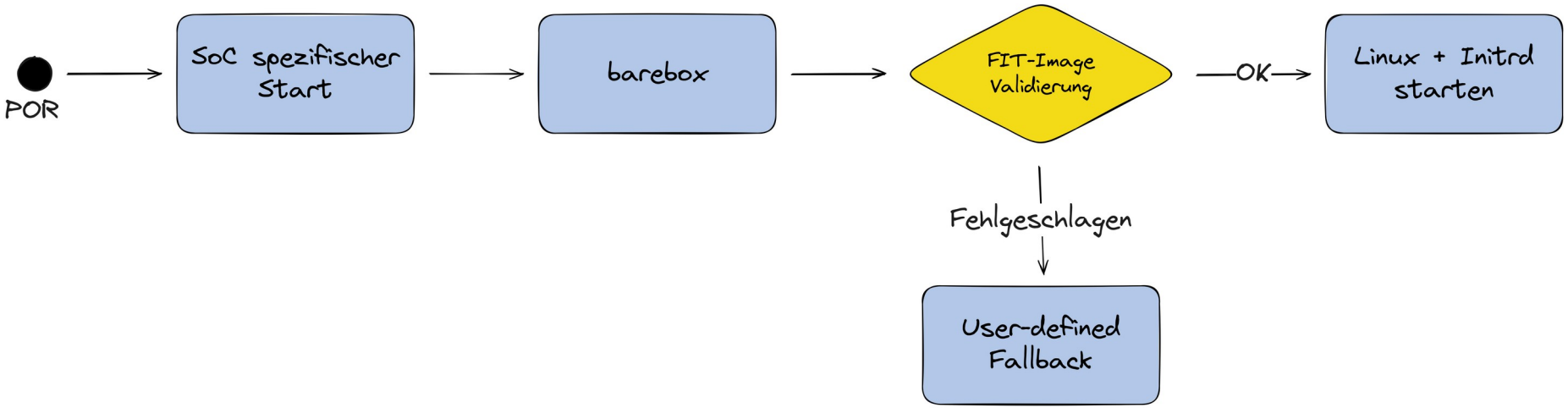
Verified Boot Chain (FIT Host Creation)



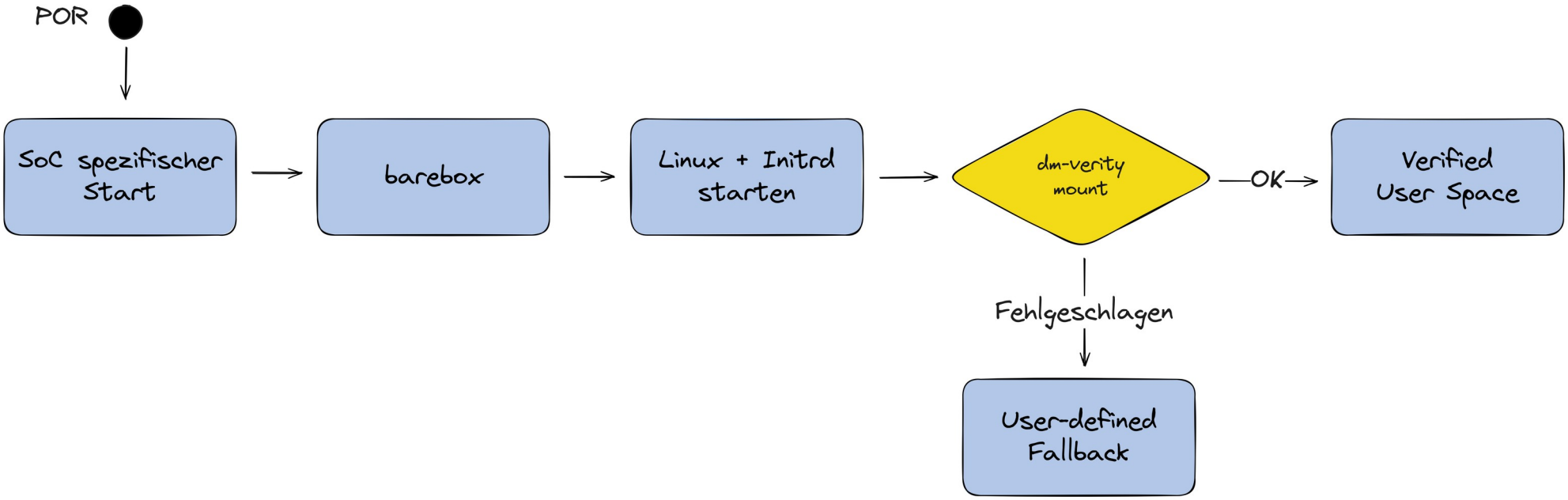
Verified Boot Chain (FIT Target Verifikation)



Verified Boot Chain (FIT)



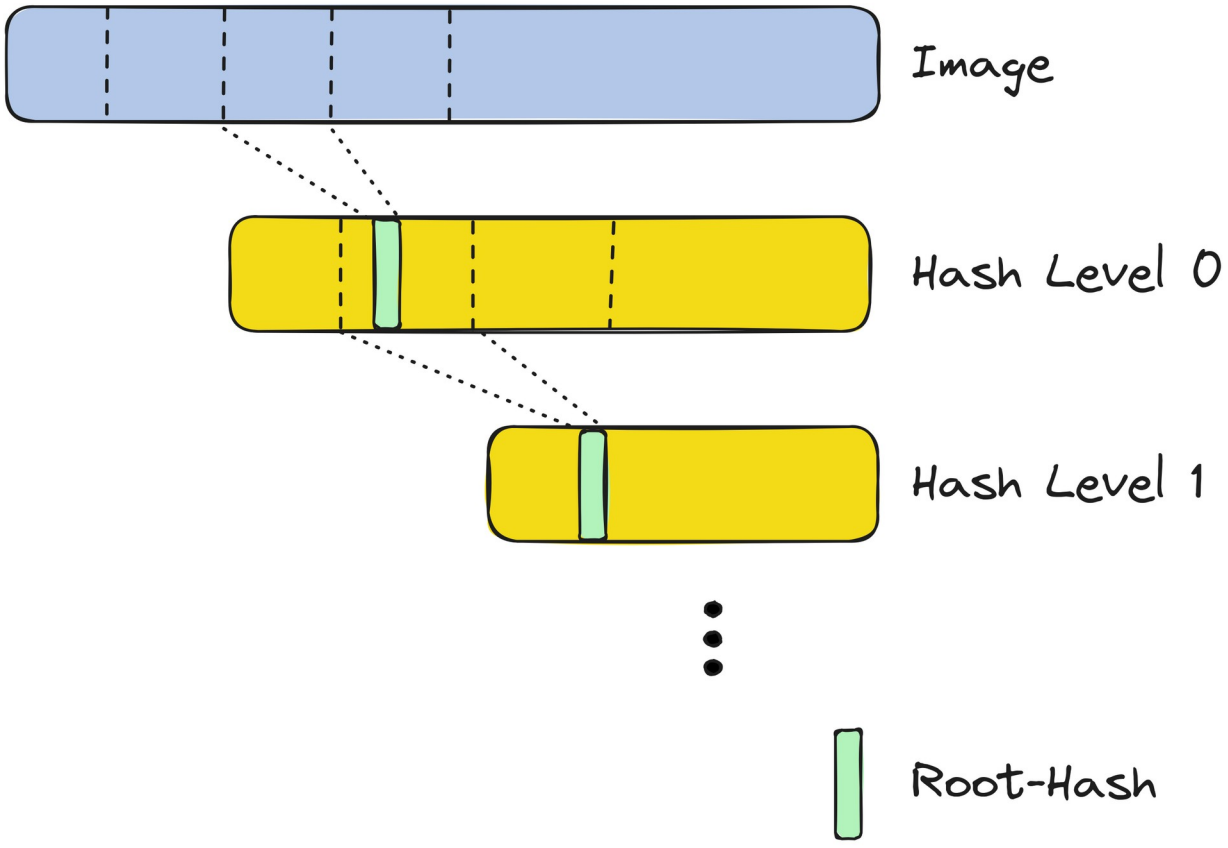
Verified Boot Chain (dm-verity)



Verified Boot Chain (dm – verity)

- Kein Filesystem!
- Target rootfs ausschließlich read-only
- Basierend auf Hash Tree / Merkle Tree

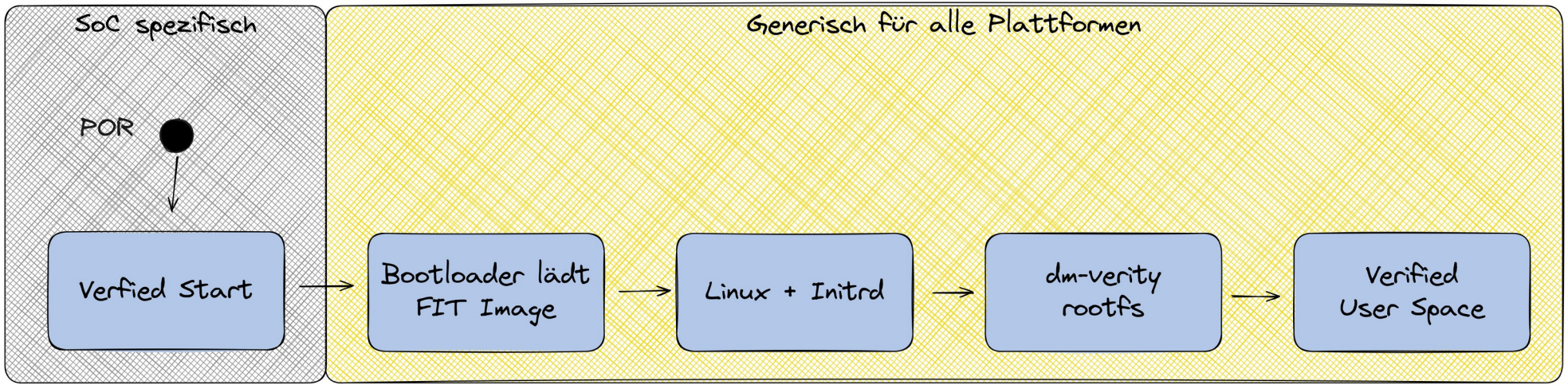
Verified Boot Chain (dm – verity)



Verified Boot Chain (dm – verity)

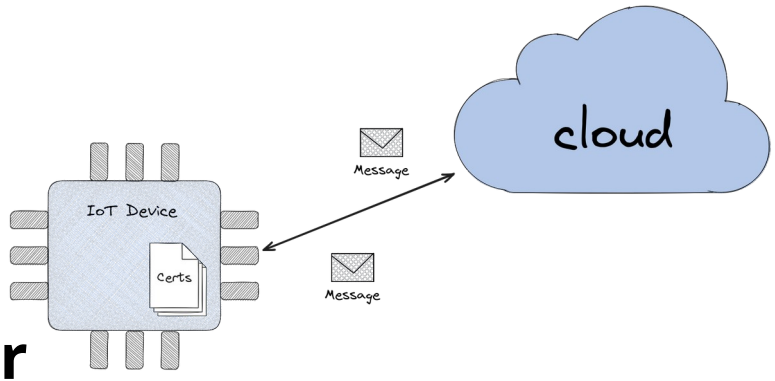
- Das dm-verity Image wird auf dem Host erstellt → der root-hash wird in die Initrd eingebaut
- Die Initrd setzt das dm device auf und mounted es als rootfs
- Die Initrd ist Bestandteil des FIT Images
  → root-hash vor böswilliger Veränderung geschützt

Verified Boot Chain



Einführung – Problembeschreibung

- (I)IoT Geräte müssen mit einer Cloud kommunizieren

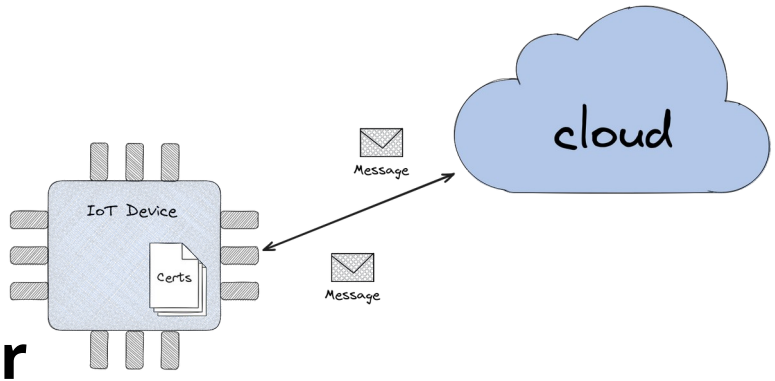


- ❌ Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren
- ✅ Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren

Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

Einführung – Problembeschreibung

- (I)IoT Geräte müssen mit einer Cloud kommunizieren



- ☑ Hersteller müssen sicherstellen das ausschließlich Geräte mit **verifizierter Software** mit der Cloud kommunizieren
- ☑ Geräte müssen sich mittels Client-Zertifikaten an der Cloud authentifizieren

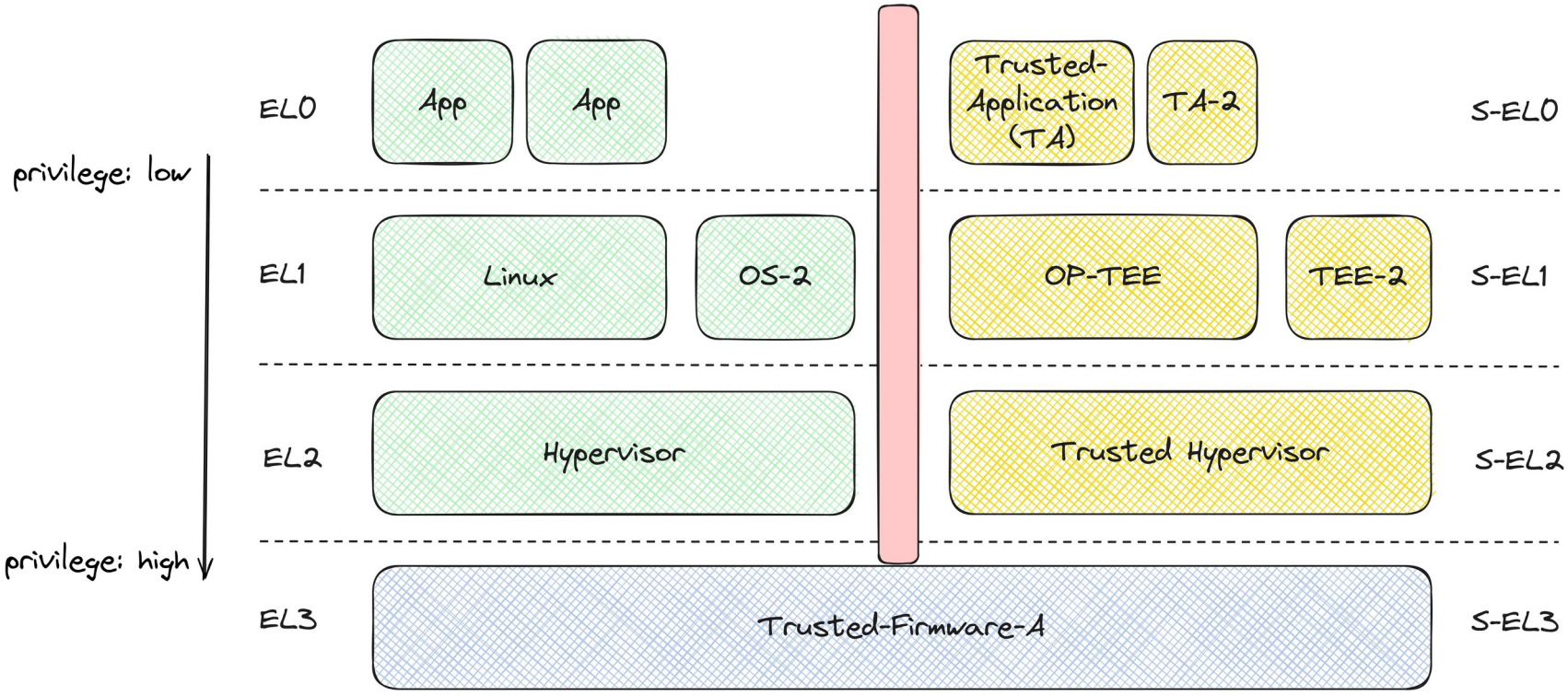
Wie sensible Daten sicher auf Millionen Geräten abgespeichert werden?

Umsetzung mittels Yocto

- Aktuelle FIT Image Generierung ist sehr U-Boot und Kernel gebunden
  - Abhilfe schafft die **`fitimage.bbclass`*** von `meta-phytec`
- Die Generierung des dm-verity Images erfolgt über eine eigens erstellte **`image_types_verity.bbclass`***
- Die Signierung der einzelnen Artefakte erfolgt über die `meta-oe` **`signing.bbclass`**

* Pengutronix ist am Mainlining dieser Komponenten

Speicherung der Zertifikate (TEE)



EL = Exception Level, S-EL = Secure-Exception Level