Linaro Connect LONDON 2023

Linaro Connect 2023

Build Security & Privacy Compliant Embedded System

Lei Zhou
Loic Poulain



Why security & privacy compliance matters?

Security & privacy compliance is very important for device manufacturers nowadays to get their products into market.

Consequences of failing to comply:

Legal litigation, reputation damage, financial penalties...

A few of widely known **cybersecurity regulations such as**:



- US IoT Cybersecurity Improvement Act of 2020.
- EU Cyber Resilience Act, making manufacturers accountable for ensuring their products being digitally secure.
- EU General Data Protection Regulation(GDPR) for Data protection and Privacy.

Challenges to meet compliance requirements

- Regulations and standards vary across different industries and regions, **while technical (open-)source solutions & frameworks are fragmented.**
- **Fast evolving cyber-attacks (ever increasing offensive capabilities)** against all **connected** embedded systems. Where attack entry point can be in every layer. eg. *Spectra/Meltdown attack at Silicon's micro-architecture*, *DPA or EMI/ultrasound side channel based attacks at electric layer*, supply-chain attack at HW accelerators' microcode, secure enclave's firmware to OS, application frameworks etc.
- Fast evolving heterogeneous computing platforms and complex open-source software stacks (**the more complex system, the larger attack surface**).

Challenges come with Opportunities

Linaro has been actively innovating and contributing to important **open-source security & privacy oriented** projects with its technical know-how and full-spectrum exposure to the ARM ecosystem (*eg. Platform architecture/ISA*, *firmware*, *software framework internals*, *toolchain & compiler*)

A few active contributions from Linaro (**not exhaustive**):

- **OP-TEE and TrustFirmware A/M**
- **Android AOSP and Linux Security subsystems**
- **Trusted Substrate Project (TRS) to build device firmware**
- **ARM systemReady enablement**
- **ARM Morello** **(part of the UK government Digital Security by Design programme)**
- **U-boot/EFI & MCUboot - secure bootloader for microcontrollers**



Our HOLISTIC approach…

Systematic challenges calls for a systematic approach to tackle

- Zero trust and defense in depth, hardware based security as core principles
- Understand and analyse embedded system as a whole, from SoCs, hardware, firmware and OS, connectivity, infrastructure all together to evolve robust and holistic security model.
- To build security & privacy compliance into embedded system is a journey covering entire device life-cycle from supply chain and manufacturing to, device provisioning, on-boarding, deployment, operating, upgrading, decommissioning.

Thanks for being with us so far, Let's start our journey to see how to put these principles and approach into practice!

Foremost, understand regulations and standards

Which guides us for security & privacy **requirement discovery**. ***Select applicable one(s) for the device you plan to build depending on industry and regions***:

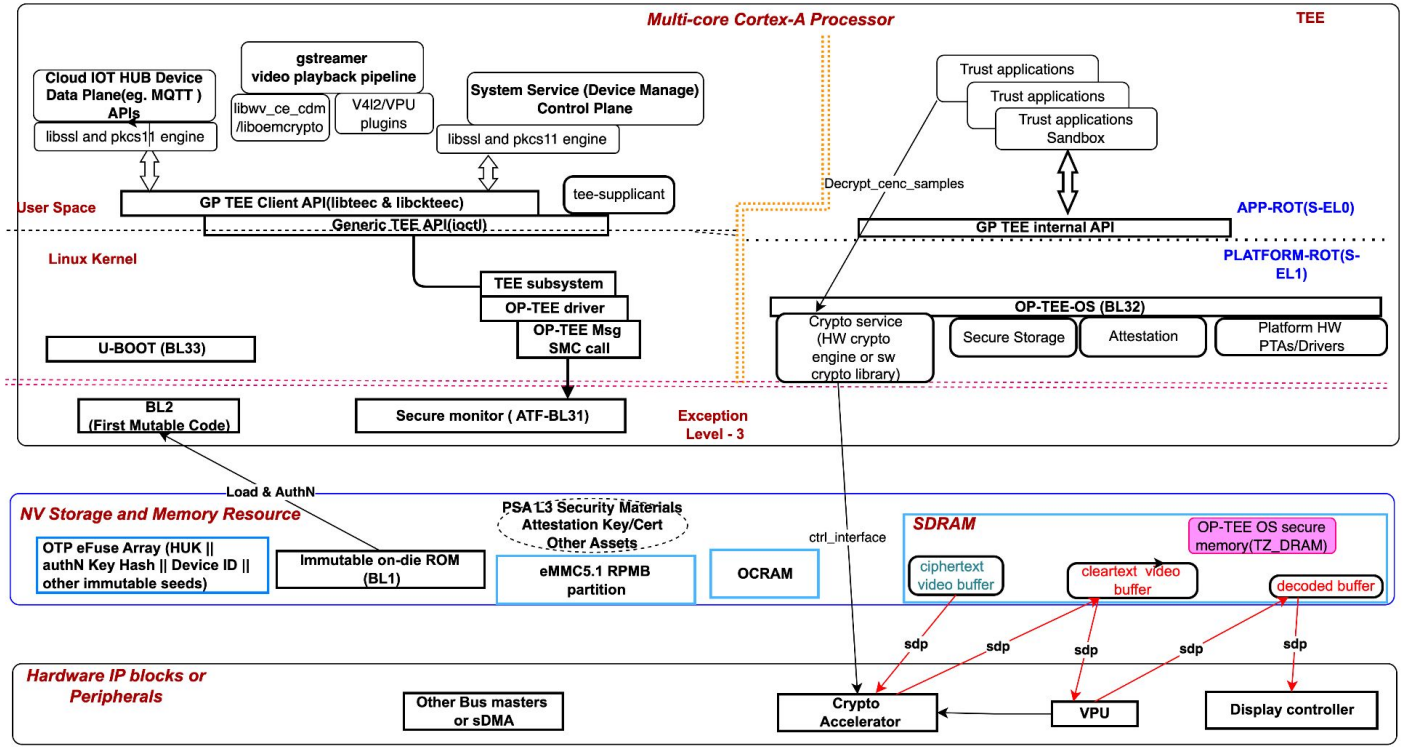
- Common Criteria EAL+ evaluation, GDPR for Data protection in EU
- ISO/IEC 27001:2013 Or NIST Cybersecurity Framework(CSF)
- FIPS 140-2
- For banking/Payment industry, PCI-DSS and PCI-PTS
- ISO/SAE 21434 (Road vehicles – Cybersecurity Engineering)
- IEC 62443 for industrial automation and control systems
- HIPAA Privacy and Security Rules for medical devices.



Eg. PCI-PTS standard requires ***Defence in depth and at least two independent defence schemes for any secure sensitive functions/assets.***

Eg. ***FIPS 140-2 level-3 requires keystore need to be Physical and side-channel attack resistance, (eg. Android StrongBox.)***

Understand your system (assets, TCB & DF)



Note: *Diagram here illustrates some typical data flow and trust boundaries for use cases eg.*

1. DRM secure data path protection.
2. Cloud based secure endpoint management(PSA-L3 compliance).

Analyze threats to your system

Threat modelling systematically guides us:

- Identify security assets and data flow, Trust boundaries and interfaces between different domains/components, which serve as a logical high-level view to define the system security model.
- Identify potential threat vectors (remote, local and physical attacks)
- Define security features and protocols required to mitigate these risks by applying our ZERO-TRUST/Defense-in-Depth mitigating approach.

Some threat modelling frameworks, eg.:

- **STRIDE model.**
- **OWASP top 10.**
- **MITTE ATT&CT framework or other equivalent.**

List out your threat vectors and mitigations

Threat vector	STRIDE	Asset	Mitigation
Un-authorized software injection, such as FW and kernel, rootfs or system configuration	*Spoof & tamper*	*FW binaries & meta-data*	*Secure boot (Extend CoT from HW RoT by via code authentication)*
Device counterfeit and cloning	*Spoof*	*Device Identity (HW & FW)*	*Immutable silicon ROT, TPM, or DICE device attestation to verify authenticity*
0-day Attack or unsafe REE based remote/local attacks to connected devices	*DoS & Elevating Privilege*	*Sensitive operations*	*Secure by Isolation (HW or/and SW enforced sandbox) to contain potential compromise.*
Secret or Credential extraction User privacy data & IP leaking	*Info leak*	*Key store/ User Privacy DB*	*1. Defense in Depth 2. CIA for lifecycle of Data (DAR, DIT and DIU)*
**Run-time kernel memory tampering *Software not fully patched for public CVE.*	*Tamper*	*Kernel or FW text/RO sections*	*Continuous monitoring. (Response and Remediation) eg. remote attestation & FOTA secure update)*

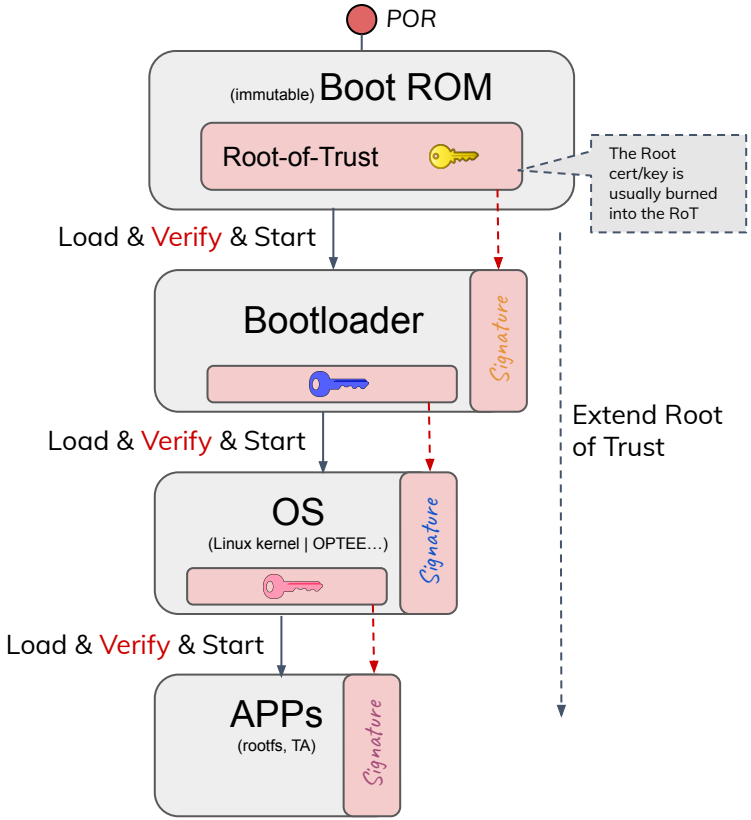
non-exhaustive examples

Authenticate from POR - Secure Boot & RoT

Secure Boot (or Verified Boot) strives to ensure executed code is **trusted and legitimate** by verifying the **integrity** and **authenticity** of the software.

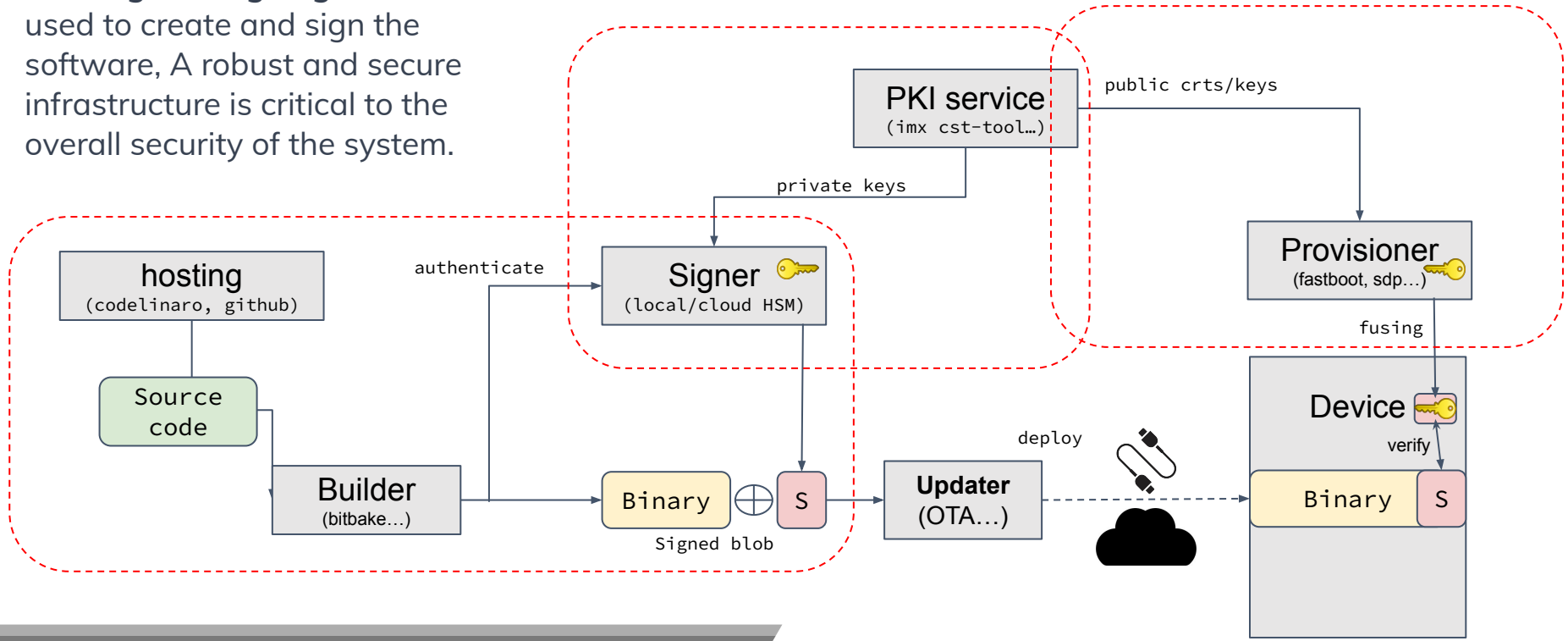
Secure Boot is defined as a boot sequence in which each software image to be executed is authenticated (asymmetric signatures) by software that was previously verified. This the **secure boot chain**.

The very first (root) verification relies on a hardware/silicon **Root-of-Trust** (RoT), which is inherently trusted and secure by design.



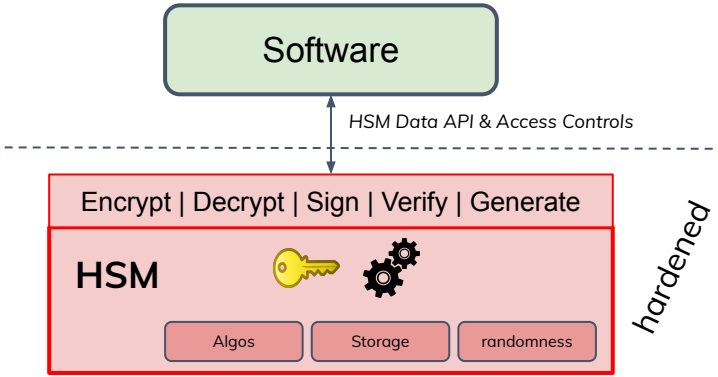
Secure boot deployment & provisioning

Secure Boot is only as secure as the **building and signing infrastructure** used to create and sign the software, A robust and secure infrastructure is critical to the overall security of the system.



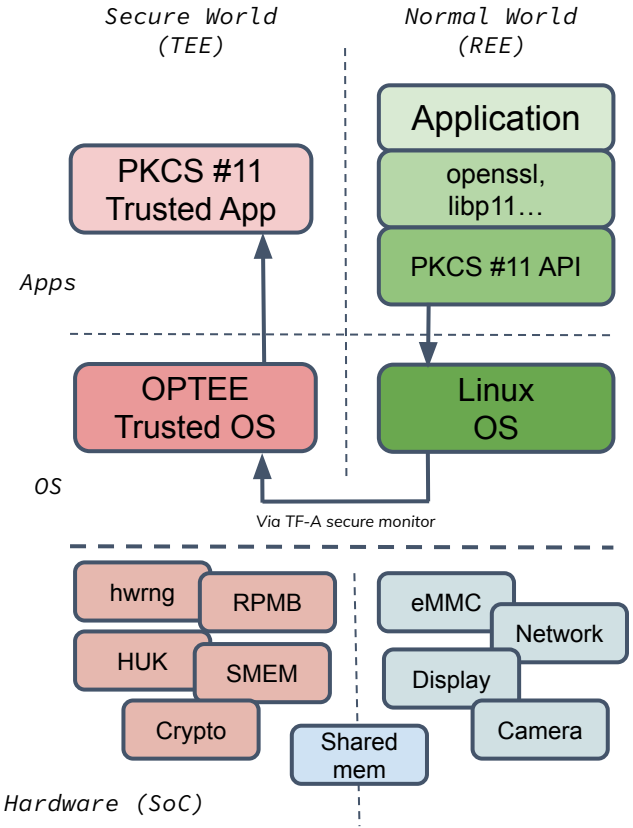
Protect your sensitive assets and operations

- Store and manage **secrets** such as **digital keys** or tokens securely.
- Perform **cryptographic operations** such as `encrypt`, `decrypt`, `sign` and `verify` in **trusted context/environment**.
- **Hardened** and **tamper resistant** to **protect** critical PKI root and CA signing keys, trusted identity credentials, TLS/SSL keys, etc.
- **T**rusted **P**latform **M**odule (TPM, fTPM) and **H**igh **S**ecure **M**odule (HSM, fHSM).
- FIPS 140-2, GDPR, eIDAS, PCI DSS, HIPAA...



Protect your sensitive assets and operations

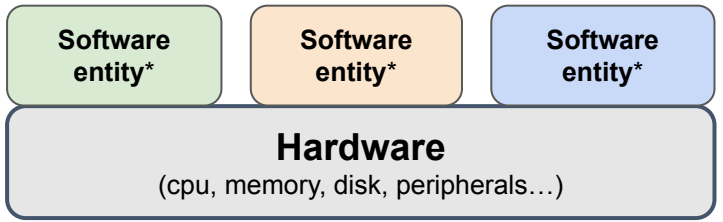
- Store and manage **secrets** such as **digital keys** or tokens securely.
- Perform **cryptographic operations** such as `encrypt`, `decrypt`, `sign` and `verify` in **trusted context/environment**.
- **Hardened** and **tamper resistant** to **Protect** critical PKI root and CA signing keys, trusted identity credentials, TLS/SSL keys, etc.
- **T**rusted **P**latform **M**odule (TPM, fTPM) and **H**igh **S**ecure **M**odule (HSM, fHSM).
- FIPS 140-2, GDPR, eIDAS, PCI DSS, HIPAA...



example: PKCS #11 fHSM with OPTEE

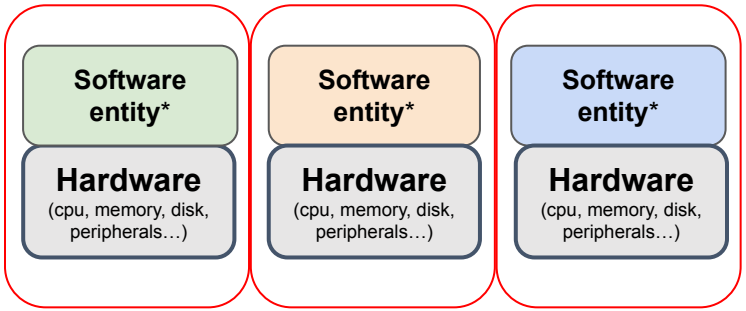
Secure by isolation

On a system, **multiple** logical **software component instances** run onto the same **unique** underlying **hardware**...



Ideally, each component should run onto its own dedicated and **isolated** hardware to prevent unwanted resource sharing between the **software** or **hardware** components.

This can be achieved in practice using and combining OS/Platform facilities such as **threads**, **processes**, **containers/cgroups**, **virtual machines**, **secure enclave, privileges, access controls**....



* an OS, app, process, thread...

Secure by isolation

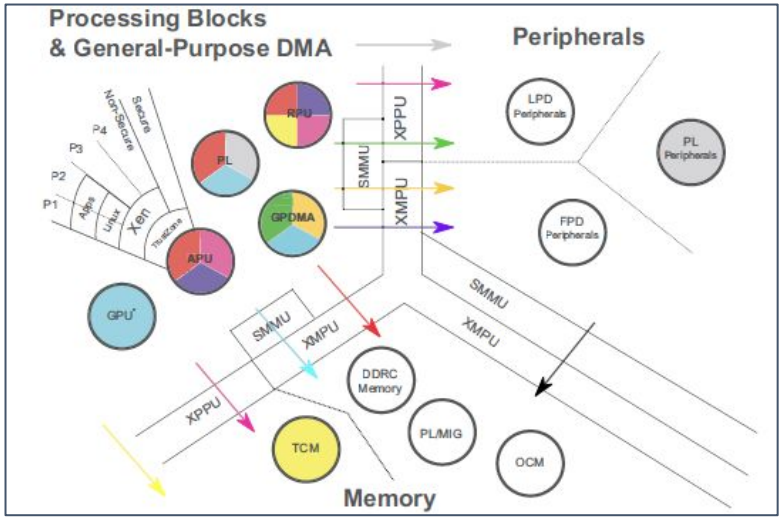
The main idea is to **separate duties** between the components, especially for security mechanisms, with minimal functionalities to **reduce attack surface**.

Each component should follow the Principle of **least privilege** and be allowed to access only the absolutely needed resources, for everything else => **deny by default**.

The silicon/SoC allows to implement and enforces this with:

MMU/MPU, IOMMU/SMMU, Exception levels, NoC, Hypervisor controls, Trustzone, Normal/Secure/Realm worlds…

Example SoC hardware firewall:



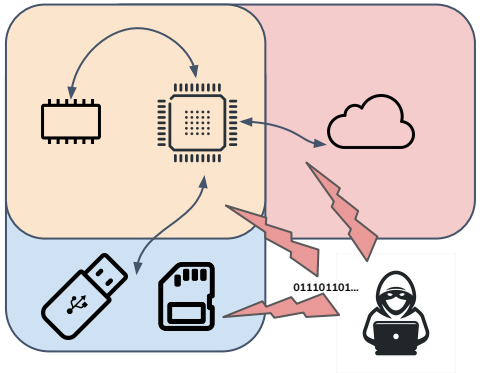
Zynq ZCU-104's Device Resources and Protection among different AXI bus masters and memory, peripherals.

* an OS, app, process, thread...

Protect user/device private data

Embedded systems collect, store, and transmit **sensitive data** => must be protected against **unauthorized access, theft**, or **misuse**.

- **Data at rest** - Protect data stored persistently in the device (e.g. SSD, eMMC...).
  - ⇨ File, Disk, Hardware encryption...
- **Data in transit** - Protect data going in and out of the device (e.g. Network, radio...).
  - ⇨ End to End encryption
- **Data in use** - Data manipulated by the device (e.g. RAM, CPU caches, registers...).
  - ⇨ Confidential computing, Tamper detection, secure enclave...

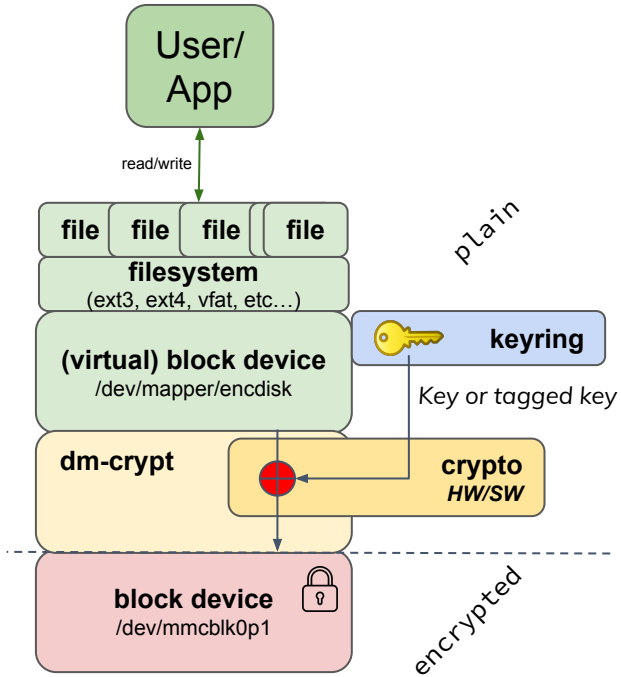


Probing, tampering, MITM...

Protect user/device private data

Embedded systems collect, store, and transmit **sensitive data** => must be protected against **unauthorized access, theft**, or **misuse**.

- **Data at rest** - Protect data stored persistently in the device (e.g. SSD, eMMC...).
  - ⇨ File, Disk, Hardware encryption...
- **Data in transit** - Protect data going in and out of the device (e.g. Network, radio...).
  - ⇨ End to End encryption
- **Data in use** - Data manipulated by the device (e.g. RAM, CPU caches, registers...).
  - ⇨ Confidential computing, Tamper detection, secure enclave...



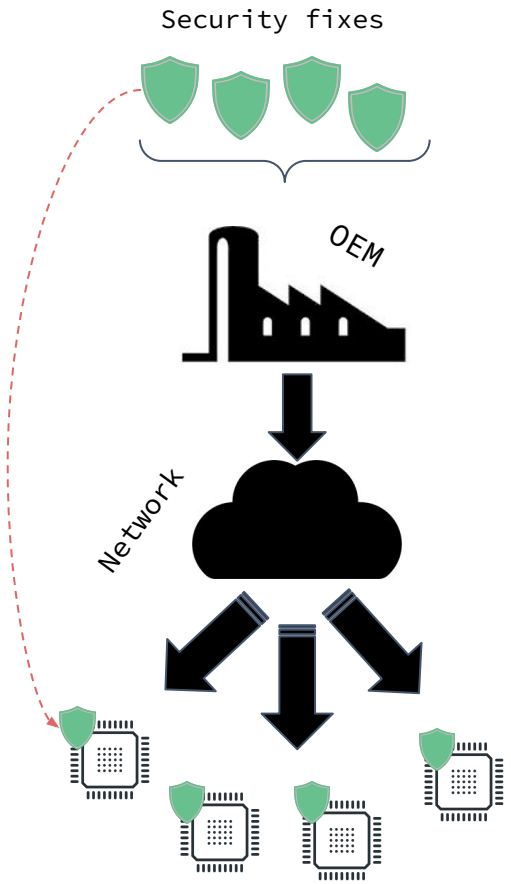
example: Disk Encryption with dmcrypt

Maintain in-field devices security

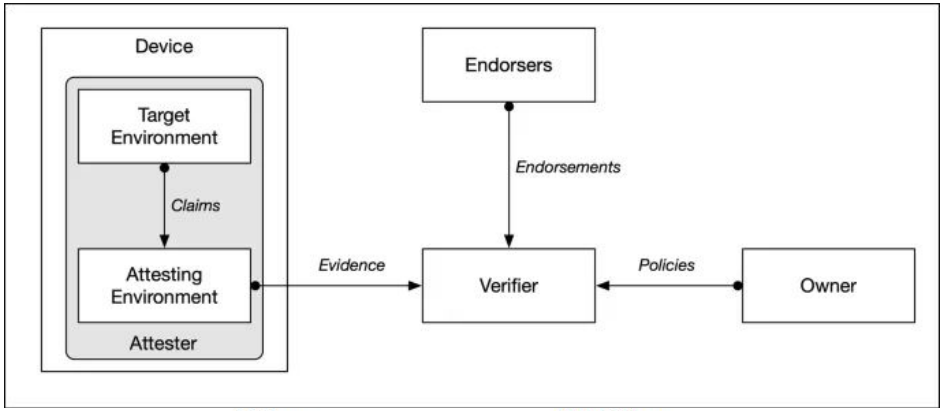
OTA updates help to reduce the risk of exploits and attacks on the system, especially in cases where critical **security vulnerabilities** are discovered after the system has been deployed.

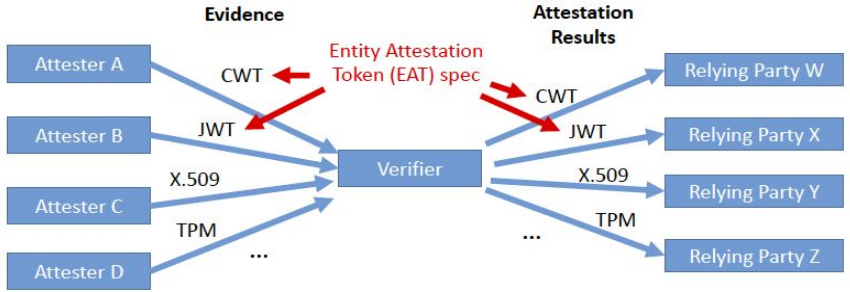
It provides a mechanism for deploying **security patches** and updates **without physical access** to the device (no recall), enforcing security policies and configurations, and ensuring that the **system remains secure** and **compliant** with industry standards and regulations.

OEM must ensure Regular **tracking security bulletins** (CVE...) for the various integrated components as well as provide a **secure communication** channels between the update server and the device.



Continuous monitoring and response





IETF RATS(Remote ATtestation procedures)

One of core principle for Zero Trust Architecture is continuously monitoring and report / response during run time.

Monitor system security status and device/run-time system integrity measurement. Attesting environment typical is hosted in TEE, where endorsing key will be signing the measurement report.

Remote attestation eg. Android SafetyNet Attestation API is used for continuously attesting current device platform/BSP integrity and generating measurement report during any sensitive operations are performed. Eg. online bank transaction


Linaro Connect
LONDON 2023
Thank you