



9<sup>th</sup> December 2025

# Secure Boot: Challenges and attestation of run-time integrity of full systems

Jeremy O'Donoghue, Qualcomm Inc.



# Disclaimer: Phil Lapczynski of Renesas did a great job on secure boot basics last year...

Assuming you all did your homework, but if not...

[https://globalplatform.org/wp-content/uploads/2025/01/9\\_Realizing-Secure-Boot-1.pdf](https://globalplatform.org/wp-content/uploads/2025/01/9_Realizing-Secure-Boot-1.pdf)

# Quick recall – some challenges with Secure Boot

## SECURE BOOT CHALLENGES – MANY HARDWARE VARIATIONS

Examples of different automotive hardware architectures. Sometime multiple types on same ECU.

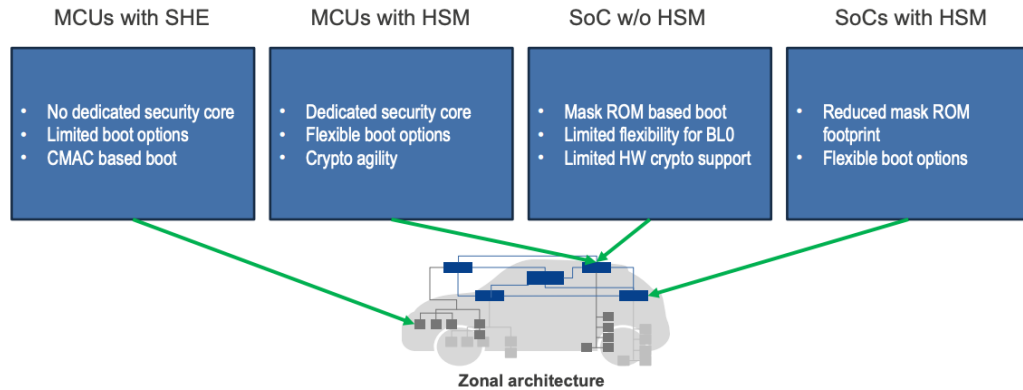


Image credit: Phil Lapczynski, Renesas, CSVF December 2024

- Significantly differing device capabilities
- Many manufacturers, often providing basically similar capability in different ways.
- Mix of symmetric and asymmetric crypto-based secure boot.

## SECURE BOOT CHALLENGES – BALANCING REQUIREMENTS

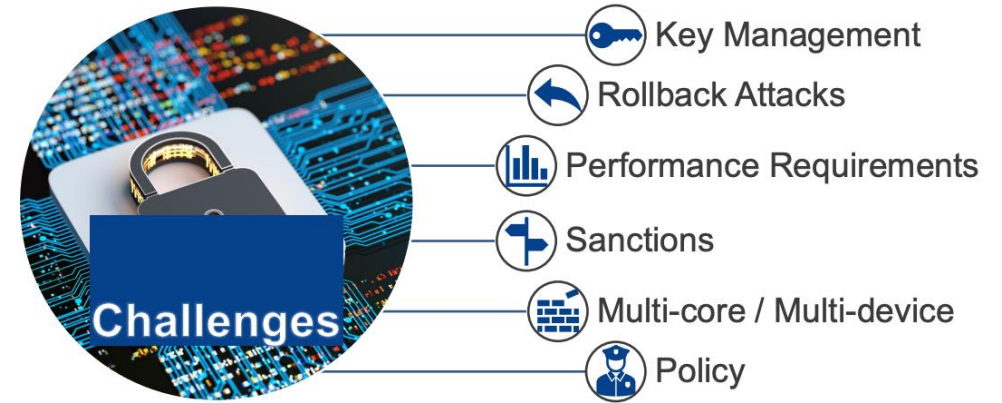


Image credit: Phil Lapczynski, Renesas, CSVF December 2024

- Need to balance of often conflicting requirements.
- Performance and security often conflict: e.g. use of asymmetric crypto (especially PQC) requires hardware crypto accelerators to meet performance goals - but not cost-compatible with small ECUs.
- System is as secure is its weakest component

# Secure Boot Requirements

Basic requirements to prevent unauthorized code from running at boot:

- The system SHALL only execute software that has been authorized by the OEM.
- The system SHALL ensure that the software image has not received unauthorized modification.
- The system SHALL ensure that rollback to older software after update can be prevented.

As Phil explained last year, surprisingly hard, even for a single system.

What about 50 or more systems that must all boot securely – i.e. the ECUs and other systems in a typical mid-range vehicle?



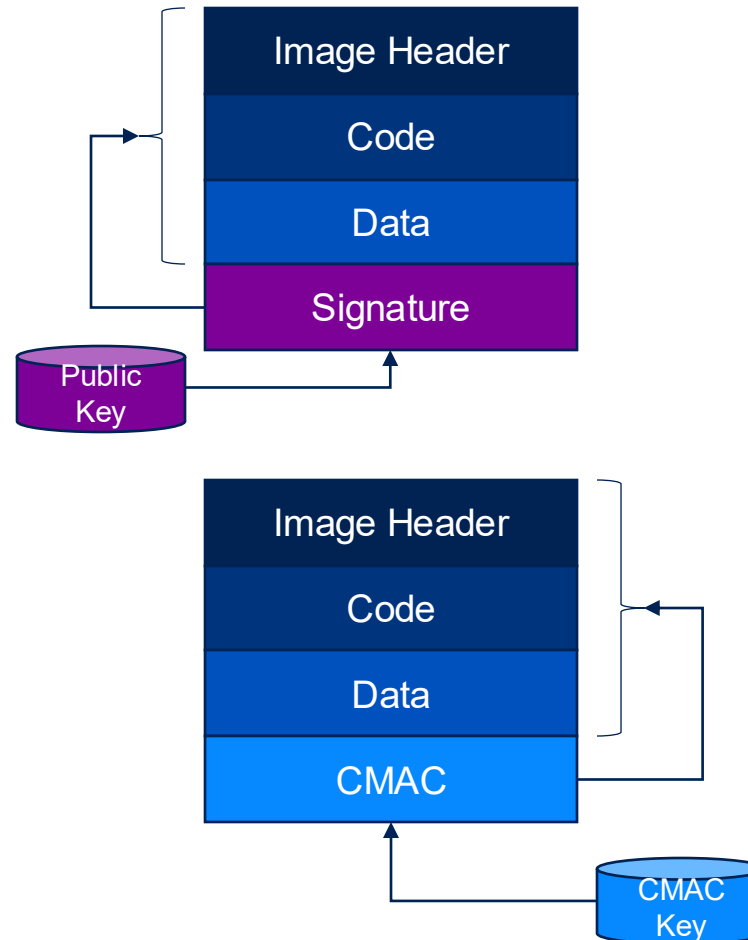
# Secure Boot Challenges at scale

- Diverse set of suppliers.
  - Each supplier is likely to have its own tooling and signing systems.
- Diverse set of underlying devices.
  - A car has everything from large SoCs containing multiple controllers based on asymmetric (including PQC) technologies... to very small controllers that barely support symmetric secure boot mechanisms.
- Complex orchestration and KPIs
  - Dependencies between devices when booting, e.g. ECU A cannot do anything until ECU B is running.
  - Customers don't want to wait for their cars to "boot". Expectation is "press start button, select "D" and go.
- Complexity of updates
  - When you have 50+ devices to maintain, how to be sure that all of them are running compatible software?
- Systemic vulnerabilities
  - Systems based on shared symmetric keys are vulnerable to key extraction attack unless keys are strongly protected. SE does this, but not appropriate in many cases.

# Verified Boot vs Measured Boot

## Verified Boot: Upper Image

- Verified Boot using Public Key Cryptography.
- Signature created by taking hash of image header + code + data and cryptographically signing with private key.
- Public key (stored e.g. in ROM) can be used to verify



## Verified Boot: Lower Image

- Verified boot using AES + CMAC
- CMAC calculated by performing AES+CMAC using known key and length of data.
- CMAC can be verified by device recomputing AES-CMAC using a key stored in the device.

## Measured Boot

- Same process is performed for all image types: perform a cryptographic hash of all of the data of interest and store in a protected location.

## Measurement extension

- A chain of trust can be established by hash extension:
- $H' = H \parallel HASH(M)$
- TPM defines this as PCR\_Extend

NB: Unrelated to measuring time to perform verification (see J3101)

# What is “Evidence”?

“A set of trustworthy Claims generated by an Attester to be appraised by a Verifier.”

- “trustworthy” – cryptographically endorsed by the Attester (which is Trusted –usually a Root of Trust)
- “Claims”: a piece of asserted information, usually in the form of a name-value pair.

In practice (RFC9711) an Entity Attestation Token containing Claims about the device.

- Evidence format: COSE envelope over CBOR claims
  - Why this matters: COSE and CBOR are copact, both over the wire and as implementations.
  - COSE allows (asymmetric) signed or (symmetric) MAC as the protecting envelope.
- Format is widely used (and gaining increasing traction) in other verticals.

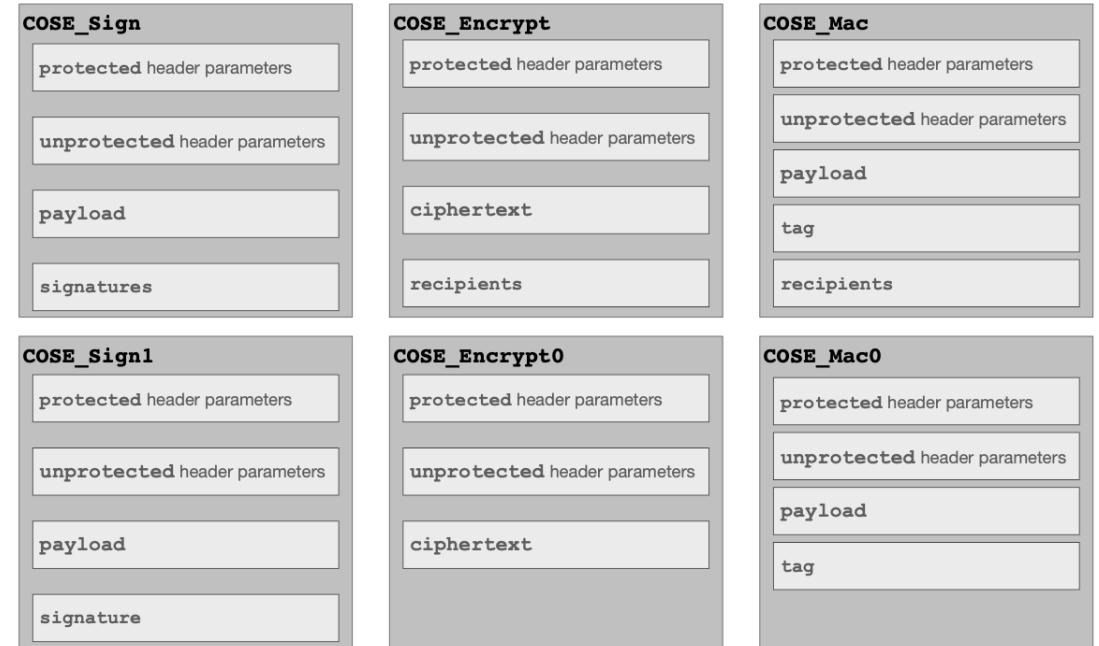


Image credit: Laurence Lundblade

Headers: describe content (e.g. algorithms used)

Payload: the Evidence

# Claims Format: CBOR map (key, value pairs)

- Basic structure defined in RFC9711
- Example claims:
  - Nonce (for freshness / anti-replay)
  - UEID: unique, non-secret device identifier
  - HW model: OEM model identifier
  - HW version
  - SW name, version
  - OEM Boot: used to indicate that only OEM signed SW can run.
  - Debug status: used to indicate if debug is possible/enabled
  - SBOM information (CoSWID)
  - DLOAs: information about approvals and certifications
  - Measurements: anything in principle, usually FW hashes in practice

Designed to support common profiles: could define e.g.:

- Small ECU (COSE\_MAC, limited claims)
- Large ECU (COSE\_Sign1, limited claims)
- Large SoC (COSE\_Sign1, full claims)

```

/ This is an EAT payload that describes a simple TEE. /
{
  / eat_nonce /      10: h'48df7b172d70b5a18935d0460a73dd71',
  / oemboot /       262: true,
  / dbgstat /       263: 2, / disabled-since-boot /
  / manifests /     272: [
    [
      258, / CoAP Content ID for CoSWID /
      / This is a byte-string-wrapped /
      / payload CoSWID. It gives the TEE /
      / software name, the version, and /
      / the name of the file it is in. /
      / {0: "3a24", /
      / 12: 1, /
      / 1: "Acme TEE OS", /
      / 13: "3.1.4", /
      / 2: [{31: "Acme TEE OS", 33: 1}, /
      /      {31: "Acme TEE OS", 33: 2}], /
      / 6: { /
      /   17: { /
      /     24: "acme_tee_3.exe" /
      /   } /
      / } /
      / } /
      h' a60064336132340c01016b
        41636d6520544545204f530d65332e31
        2e340282a2181f6b41636d6520544545
        204f53182101a2181f6b41636d652054
        4545204f5318210206a111a118186e61
        636d655f7465655f332e657865'
    ]
  ]
}

```

# Endorsing Attestation Tokens

Endorse: IETF terminology for “Cryptographic Protection”

- COSE envelope is very flexible.
- ECDSA using most curves and key lengths
- EdDSA
- ML-DSA, HSS/LMS (for PQC signature)
- AES-HMAC, AES-CBC-MAC (\*not\* CMAC)
- Many others
- Key management / storage
  - Key confidentiality required to protect attestation.
  - TEE, strong RoT, Secure Element all good options.
  - DICE signing keys (next slide) an interesting option for some cases.
- Choices have implications for verifier performance and scaling.

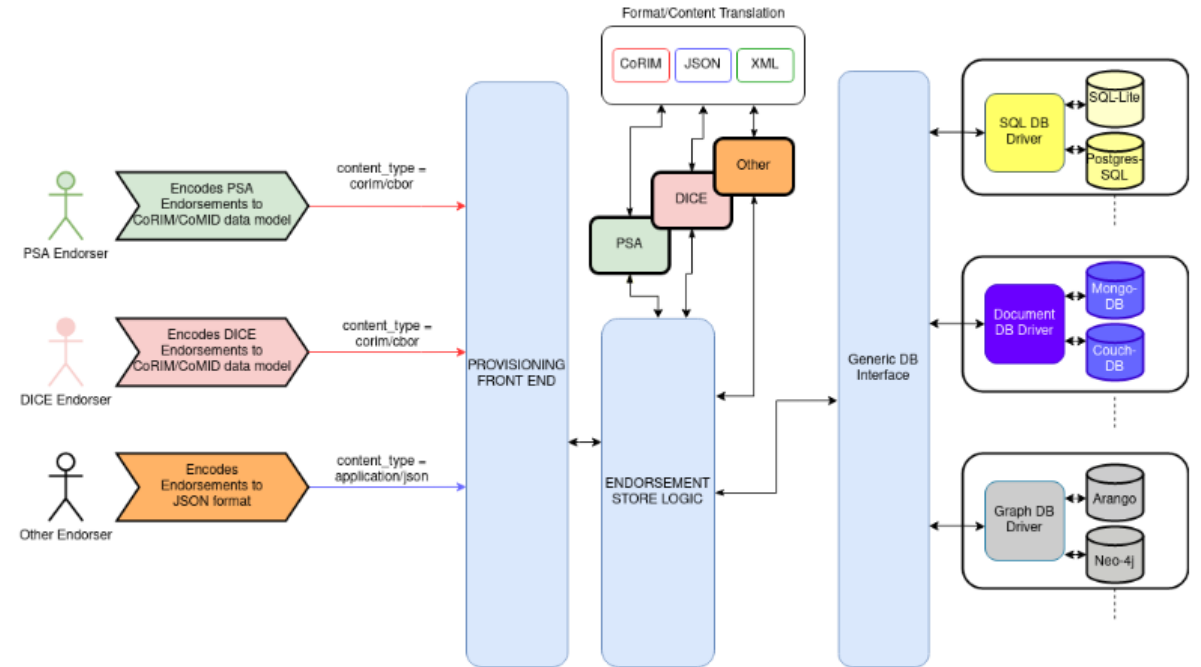


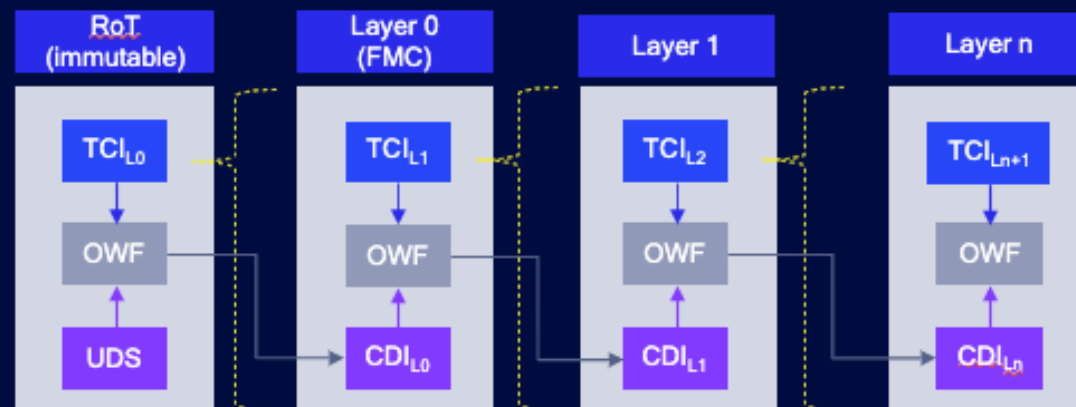
Image credit: Veraison project (Confidential Computing Consortium)

# DICE in a slide

Each DICE layer measures the next layer and derives an identity based on it, all previous measurements and a random value

## • Foundations:

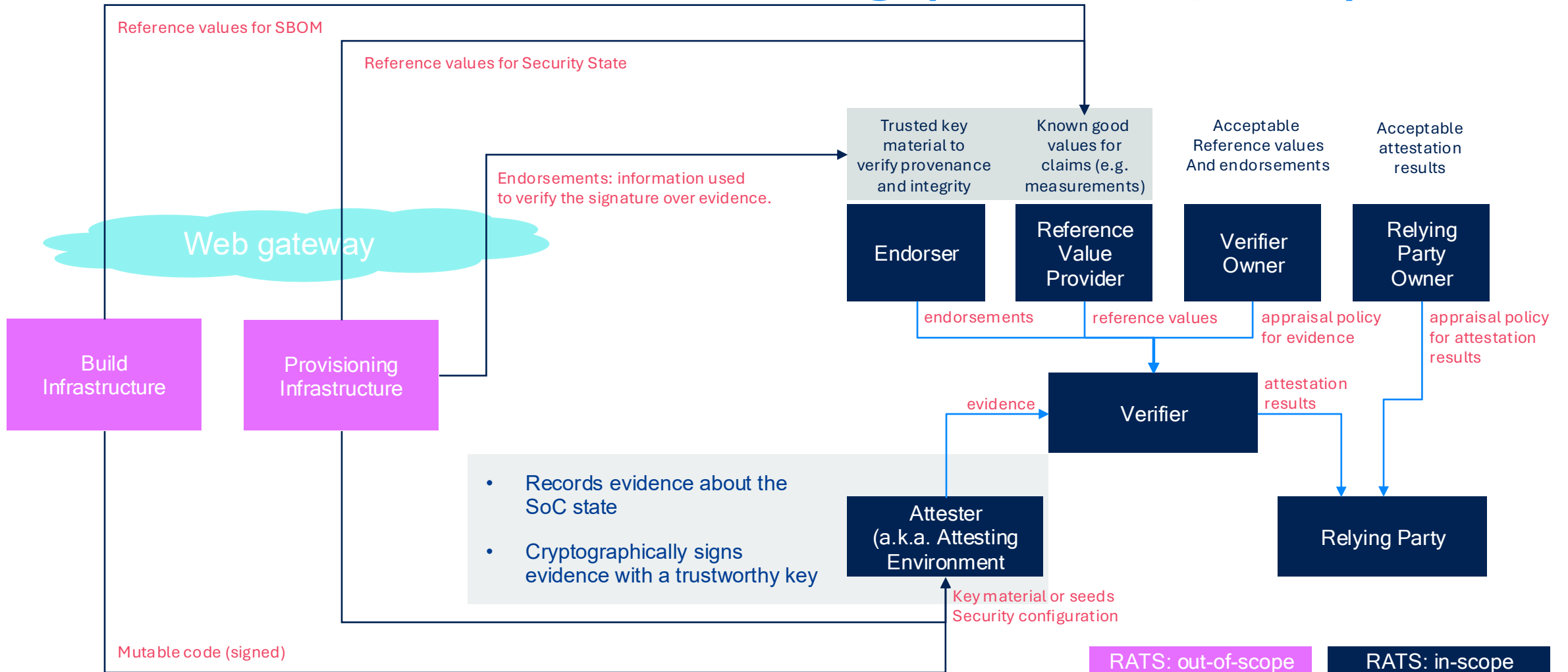
- DICE “Identity” is not a hardware identity: it is the identity of the HW along with all the software that has been loaded onto it.
- The Root of Trust (RoT) of a system is implicitly trusted and cannot be measured – this means it should be immutable.
- Good security design: the RoT should be as small as possible.
- If an image is “measured” as “good” by a previously loaded trustworthy image, it is also good.
  - “Measure” above is TCI in DICE terminology
- The immutable RoT measures the first mutable code (FMC). Since RoT is implicitly trusted, RoT must be able to “measure” whether FMC is “good”.
- Eventually, a chain of trusted images is loaded. The last image is trustworthy because all previous images were trustworthy.
  - The compounded measure is called CDI in DICE terminology
- Each image loaded, starting with FMC, is a new layer.



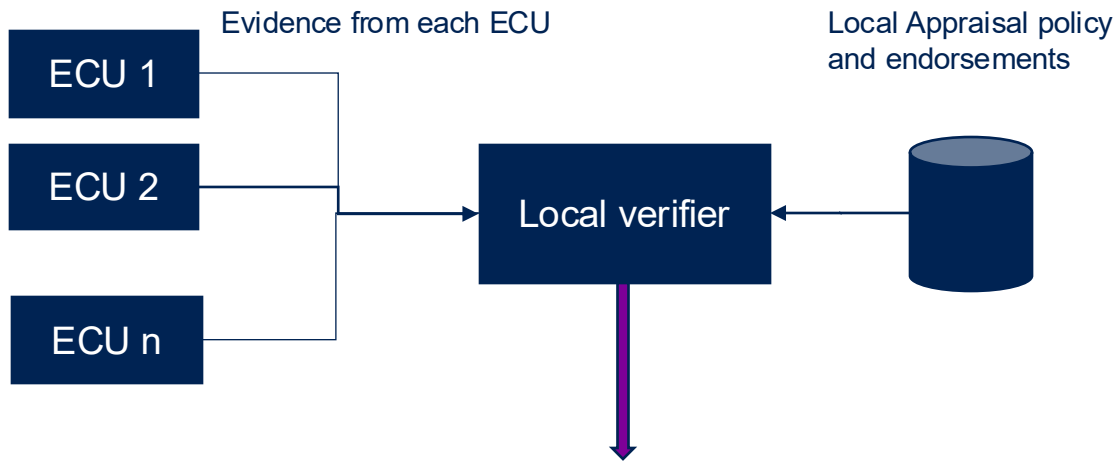
## • Cryptographic Identity:

- **UDS**: Universal Device Secret – an unchanging random value uniquely identifying an instance of a platform
- **TCI**: TCB Component Identifier – a unique identifier of code or image. Often a hash of contents but may also include other information. Any change to a component **MUST** change the TCI.
- **CDI**: Compound Device Identifier – A value obtained by combining at least two TCI values. CDI is a secret which **MUST NOT** leave its owing DICE layer.
- **OWF**: A cryptographic OWF e.g. *HKDF (seed, data)*

# Attestation and Provisioning (IETF-inspired)



# Attestation at scale in the vehicle



Local verifier: a subsystem in the vehicle that obtains and verifies attestations from every CPU.

- On demand (usually just after boot)
- Updateable policy and endorsements (connected)
- Produces a simple good/bad result (e.g. to inform driver of fault)
- Needs access to keys for verifying Evidence

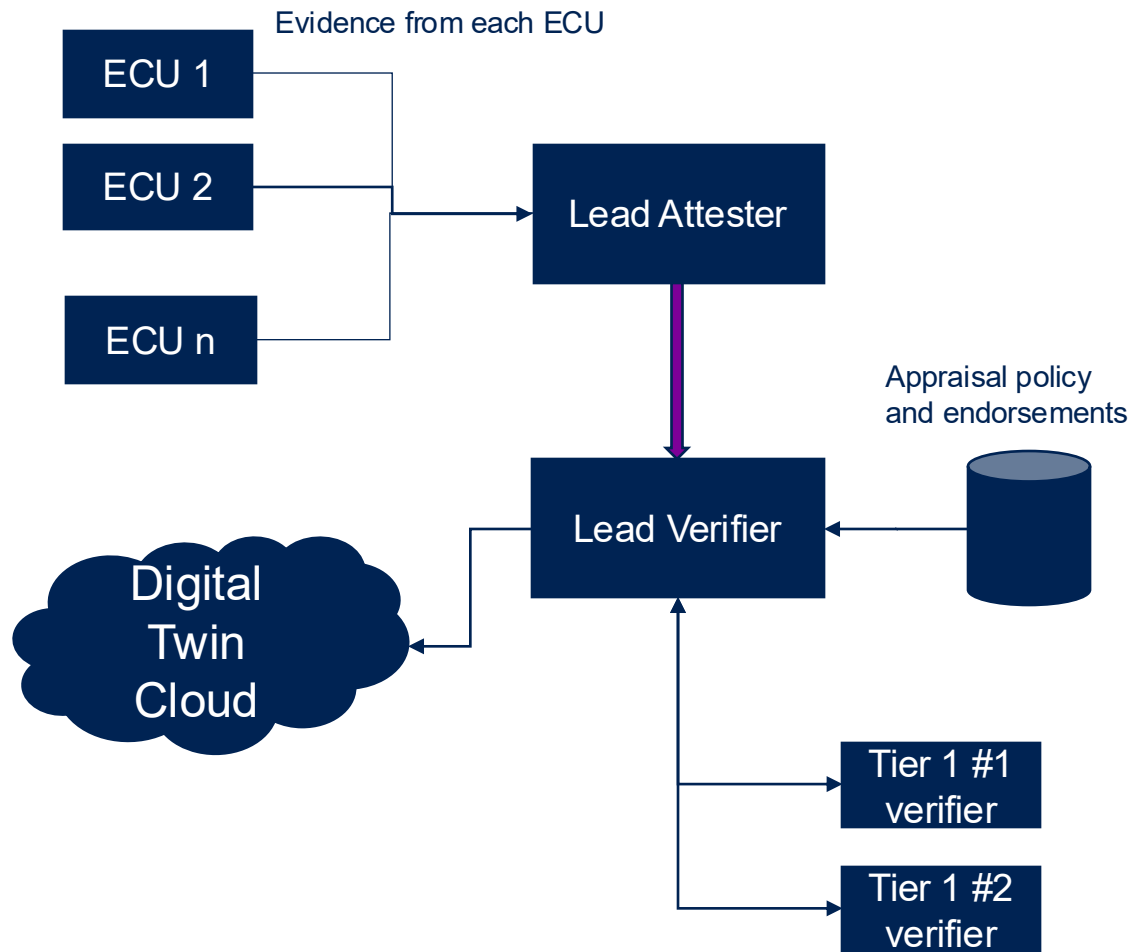
Attestation can be used internally:

- A verifier in the vehicle has all of the information needed to check the Evidence:
  - Signing / MAC keys.
  - Set of “golden” expected values.

Probably only simple checks on-board

- For reasons of speed and simplicity (Verifiers can get complicated).
- Likely based on the “expected” software installation.

# Attestation at scale in the cloud



Use-cases: digital twins, data mining, recalls

- Lead Attester in vehicle (probably same subsystem as local verifier) collects fresh attestations from all ECUs
- Submits them to a lead Verifier controlled by vehicle OEM which applies appraisal policy and endorsements.
- Optionally can make use of Verifier instances provided by Tier 1s and/or Silicon vendors.
- Attestation Results can be fed into Digital Twins and used for various types of data mining.



# Global Platform™

The standard for  
secure digital services  
and devices

→[globalplatform.org](https://globalplatform.org)