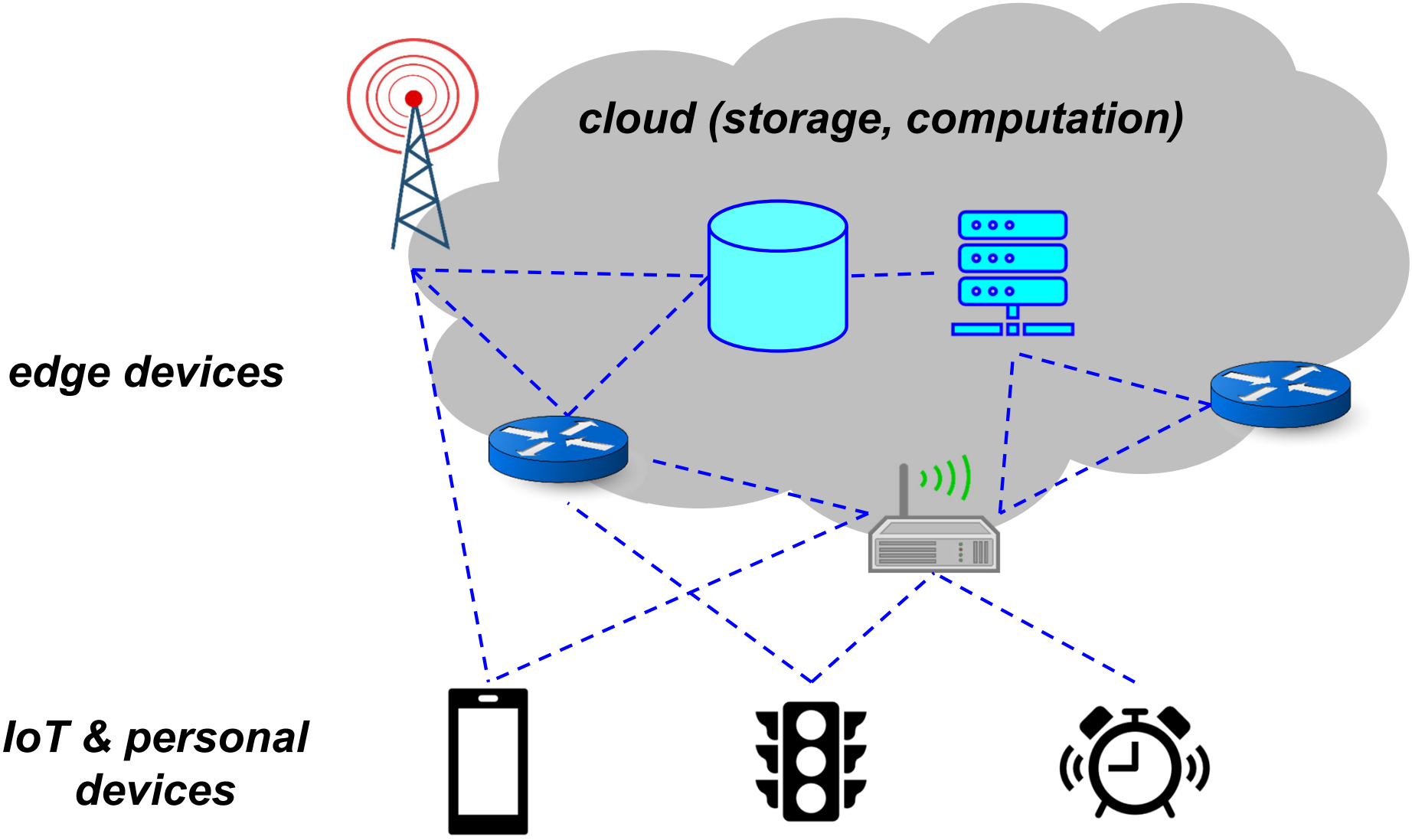
Integrity and attestation of distributed infrastructures (cloud, SDN, NFV, …)

Antonio Lioy
< antonio.lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica
Torino – Italy

Typical distributed infrastructure (e.g. smart-x)



Trend towards softwarization

- **SDN (software-defined networking)**
- **… but also**
  - SDR, NFV, …
  - AI (!)
- **as a consequence, more flexible but more vulnerable**
  - software more prone to bugs than hardware
  - software updates

- EASY (FAST, FLEXIBLE, …)
- CHEAP
- SECURE
… PICK TWO!

Can I trust this infrastructure?

- **trustworthy = will behave in the expected way**
- **problems:**
  - trust in the cloud provider(s)
  - trust in the network/edge provider(s)
  - low or no access control for edge- and end-devices
  - low cost IoT devices (typically it implies low security too)
  - personal devices (typically managed by “ignorant” users)
- **if possible, protect (i.e. avoid/block attacks)**
- **otherwise, at least monitor the “state” for early detection (and possibly reaction)**

INTEGRITY VERIFICATION

Integrity

- **hardware**
  - am I talking to the right (intended) node?
  - does it host the expected (physical) components?
- **software**
  - am I talking to the right (intended) software component?
  - is it correctly configured?
  - is the baseline software the expected one?

TEE
Trusted Execution Environment

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

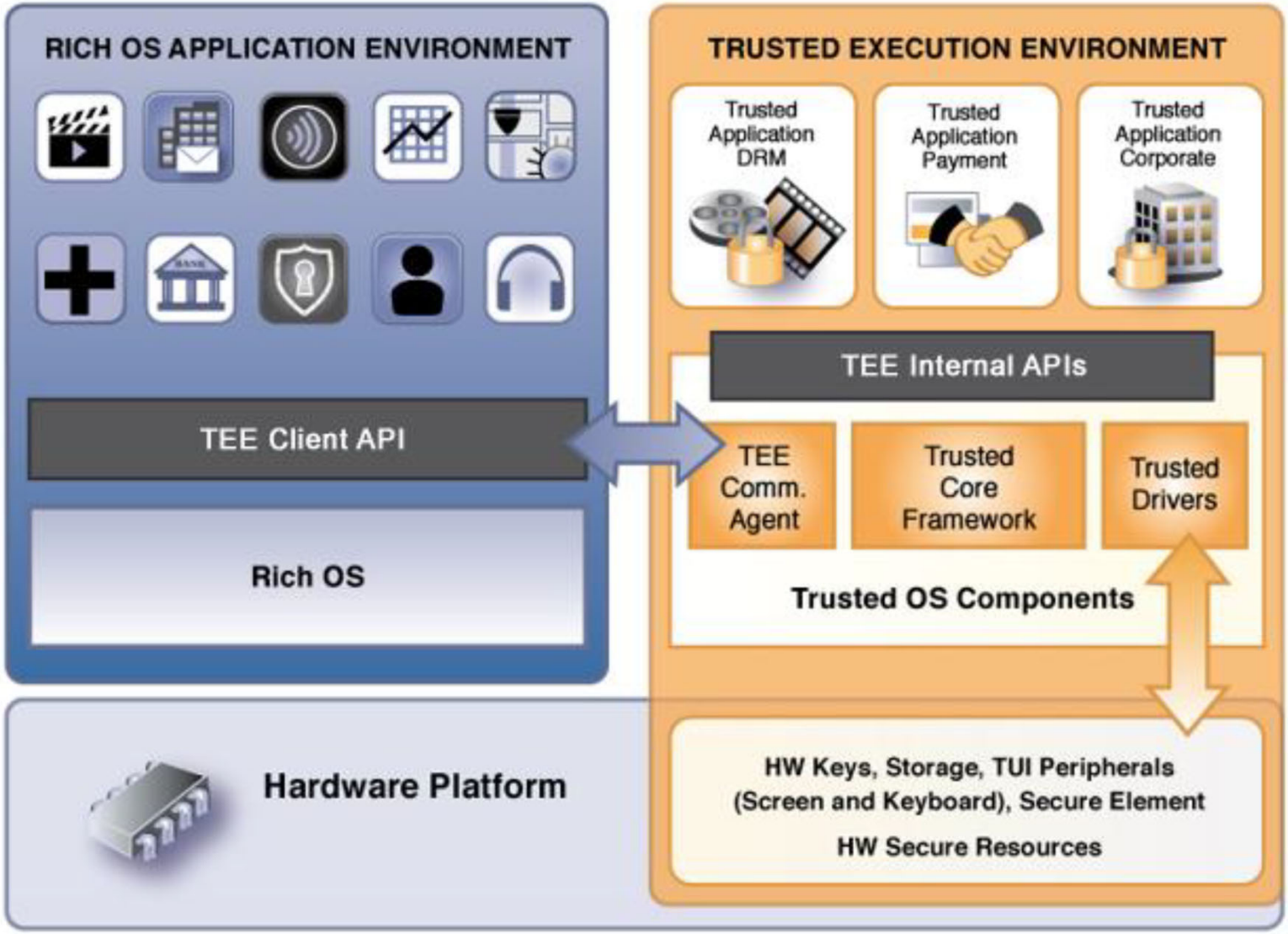
Trusted vs. Trustworthy

- **Trusted**
  - someone or something that you rely upon to not compromise your security
- **Trustworthy**
  - someone or something that will not compromise your security
- **Trusted is about how you use something**
- **Trustworthy is about whether it is safe to use something**
- **Trusted Execution Environment is what you may choose to rely upon to execute sensitive tasks**
  - the TA's (Trusted Applications)
- **hopefully it is Trustworthy too!**

TEE evolution

- **Open Mobile Terminal Platform (now within GSMA)**
  - 2006 TR0 specifications (security requirements)
  - 2008 TR1 specifications (TEE on top of TR0)
- **(2010) Global Platform launched to define interfaces and certifications (is the de-facto standardization body)**
- **2012**
  - ARM, Gemalto, and G+D formed Trustonic for an open TEE
  - GlobalPlatform and the Trusted Computer Group (TCG) founded a joint working group focusing on TEE specifications and use with the Trusted Platform Module (TPM)
- **first major business case: Netflix HR on smartphones and tablets**
- **then financial, enterprise, government, automotive, IoT, …**

TEE and REE


RICH OS APPLICATION ENVIRONMENT
TRUSTED EXECUTION ENVIRONMENT
Trusted Application DRM
Trusted Application Payment
Trusted Application Corporate
TEE Internal APIs
TEE Client API
TEE Comm. Agent
Trusted Core Framework
Trusted Drivers
Rich OS
Trusted OS Components
Hardware Platform
HW Keys, Storage, TUI Peripherals (Screen and Keyboard), Secure Element
HW Secure Resources

TEE – some key points

- **nowadays TEE is a hot topic for "confidential computing"**
  - fostered by the Confidential Computing Consortium (CCC)
  - protection of "data in use"
    - nobody else can read/write the data
    - processed only by an authorized application
  - compare to various cryptographic techniques to protect "data at rest" and "data in motion"
- **Root of Trust (RoT)**
  - element whose misbehaviour cannot be detected at runtime
  - should be Trusted and Trustworthy
  - part of the Trusted Computing base (TCB), i.e. the set of all HW, FW, and/or SW components critical to its security (any vulnerability inside the TCB jeopardizes the system security)

TEE security principles

- **be part of the device secure boot chain (based on a RoT) and verify code integrity during each device boot**
- **hardware-based isolation from the device's rich OS environment to execute sensitive code**
- **isolate TAs from each other**
- **secure data storage, using a hw-unique key accessible only by the TEE OS to prevent unauthorized access and modification and any possibility of exploiting the data in other devices**
- **(trusted path) privileged and secure access to peripherals**
  - peripherals (fingerprint sensors, displays, touchpads, …) can be hardware-isolated from the rich OS environment and controlled only by the TEE during specific actions (no visibility/access from the REE, malware included)

Intel IPT

- **Intel Identity Protection Technology**
- **runs Java applet on separate CPU**
  - Management Engine is part of the chipset, so it's bound to physical hardware
- **sample applications**
  - key generation and storage (integrated with Windows Cryptographic API)
  - OTP generation (VASCO MYDIGIPASS.COM)
  - secure PIN entry
    - possible because chipset also manages video

ARM TrustZone

- CPU buses extended to a “33rd bit”, signalling whether in secure mode or not
- signal exposed outside of the CPU to allow secure peripherals and secure RAM
- potentially could have indicator for which mode CPU is in
- open system and documented, but only allows *one* secure enclave

- current efforts of ARM to add a third mode

Trustonic

- **TrustZone is not very useful by itself due to only allowing one enclave**
  - Gemalto developed the Trusted Foundations system
  - G+D (Giesecke+Devrient) developed MobiCore
  - both split the one secure enclave into several ones, essentially through a smart-card operating system
- **Trustonic development based on MobiCore**
- **license fees required to implement code**
- **Trustonic's TEE OS "Kinibi"**
  - e.g. version 500 has 64-bit SMP for embedded systems
- **Samsung Knox is similar, but also introduces secure boot**

Intel SGX

- Intel Software Guard Extensions
- tightly integrated with CPU
- modifies memory management
- enclaves protected from other code, and vice versa
- as enclaves are built, the code is measured in a similar way to the TPM
- can be combined with IPT for trusted display
- SGX1 initially available on low and high-end CPUs, now SGX2 mostly restricted to server-oriented CPUs (Xeon)

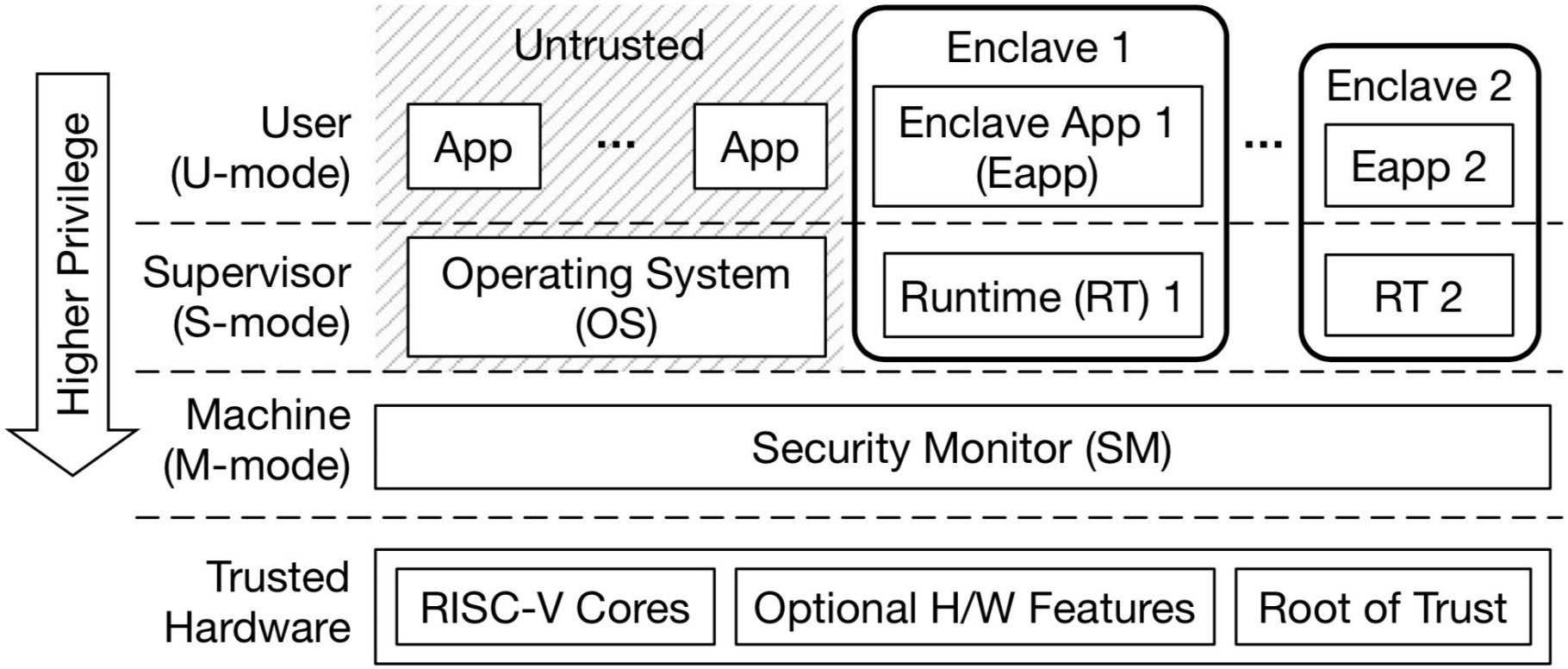
Keystone

- **open-source framework to build a TEE**
  - select only the features needed
  - minimize the TCB
  - untrusted environment (general-purpose OS) + N trusted segregated enclaves
- **on top of the RISC-V open-source customizable hardware**
  - FPGA or SOC
  - core + crypto extensions
  - various execution modes
    - M / S / U (Machine / Supervisor / User)
  - PMP (Physical Memory Protection)
    - for memory and I/O

Motivation for Keystone

- **TEEs are rigid and un-customizable.**
- **existing solutions inherit the underlying design limitations:**
  - Intel SGX: large software stack
  - AMD SEV: large TCB
  - ARM TrustZone: not enough domains

Keystone architecture



Keystone: further reading

- **Keystone presentation at CCC:**
  - video = https://www.youtube.com/watch?v=ITA3FfjKqNk
  - slides = CCCWebinar-Keystone.pdf (in course material)
- **Keystone presentation at EuroSys'20:**
  - slides = 233_lee_slides.pdf (in course material)
  - video = https://www.youtube.com/watch?v=S8MmKBCoPSg
  - https://n.ethz.ch/~sshivaji/publications/keystone_eurosys20.pdf (also in course material)

Trusted computing and remote attestation

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Baseline computer system protection

- **attackers try to inject malware at the lowest possible level to remain undetected and control the largest part of the system**
  - modify the OS
  - try to boot an alternative OS
  - modify the boot sequence or the boot loader
- **we need to protect the boot system and the OS**
- **once we had the BIOS**
  - very difficult to protect
- **now we have UEFI**
  - native support for firmware signature and verification
- **then the boot loader can verify the OS before activating it**

Rootkits

- **Firmware rootkits**
  - overwrite the BIOS/UEFI (or the firmware of other hardware!) so the rootkit can start before the OS
- **Bootkits**
  - replace the OS's bootloader so that the node loads the bootkit before the OS
- **Kernel rootkits**
  - replace a portion of the OS kernel so the rootkit can start automatically when the OS loads
- **Driver rootkits**
  - pretend to be one of the trusted drivers that the OS (e.g. Windows) uses to communicate with the hardware

Software to protect software?

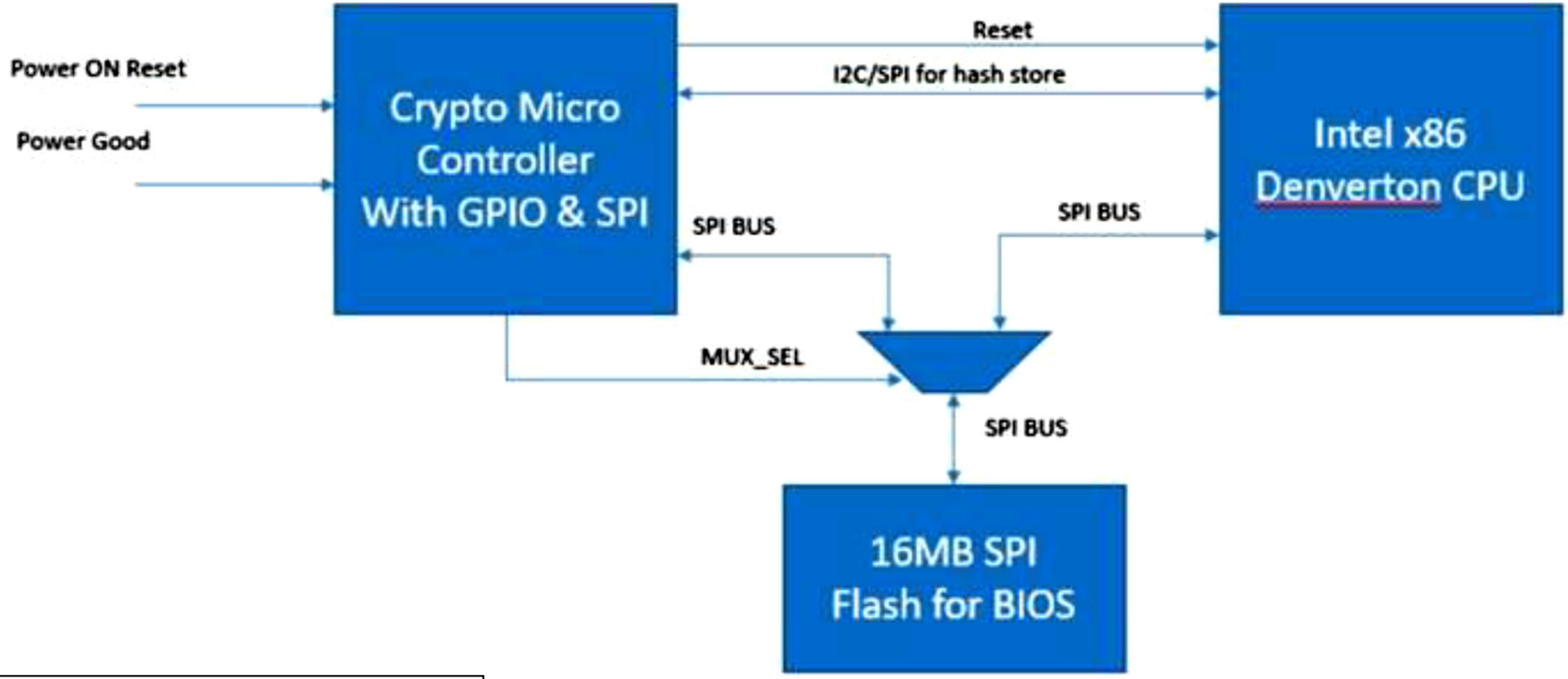
- **not a good idea (as software may fail …)**
- **need hardware support to protect software**
  - RoT (Root-of-Trust)
  - should be part of the TCB (Trusted Computing Base)
  - … because hardware is anyway operated by software or firmware
  - TCB should be minimal

Firmware self-protection (SW root-of-trust)

- example courtesy of HPE
- HPE has a “signature” region at a fixed location in the final BIOS image (16MB)
- after the BIOS build (i.e. at manufacturing time), the SHA256 of specific BIOS regions is calculated; these regions include static code, the BIOS version information and microcode
- the hash is sent to the HPE signing server which returns a signed hash image (32 bytes + signature and certificate size), which is copied into the “signature” region
- after power on, the early BIOS code calculates the combined hash of each of the specific valid regions in the BIOS image
- after verifying that the “signature” region contents are valid, the BIOS compares the stored hash and the calculated hash
- if both are same, boot continues, otherwise halt the system

HW root-of-trust for firmware protection (I)

- self-verification is based on the firmware itself (static portion verifies the part that can be updated)
- but verification of the firmware can be implemented by an external chip as well (picture below and text in next slide)



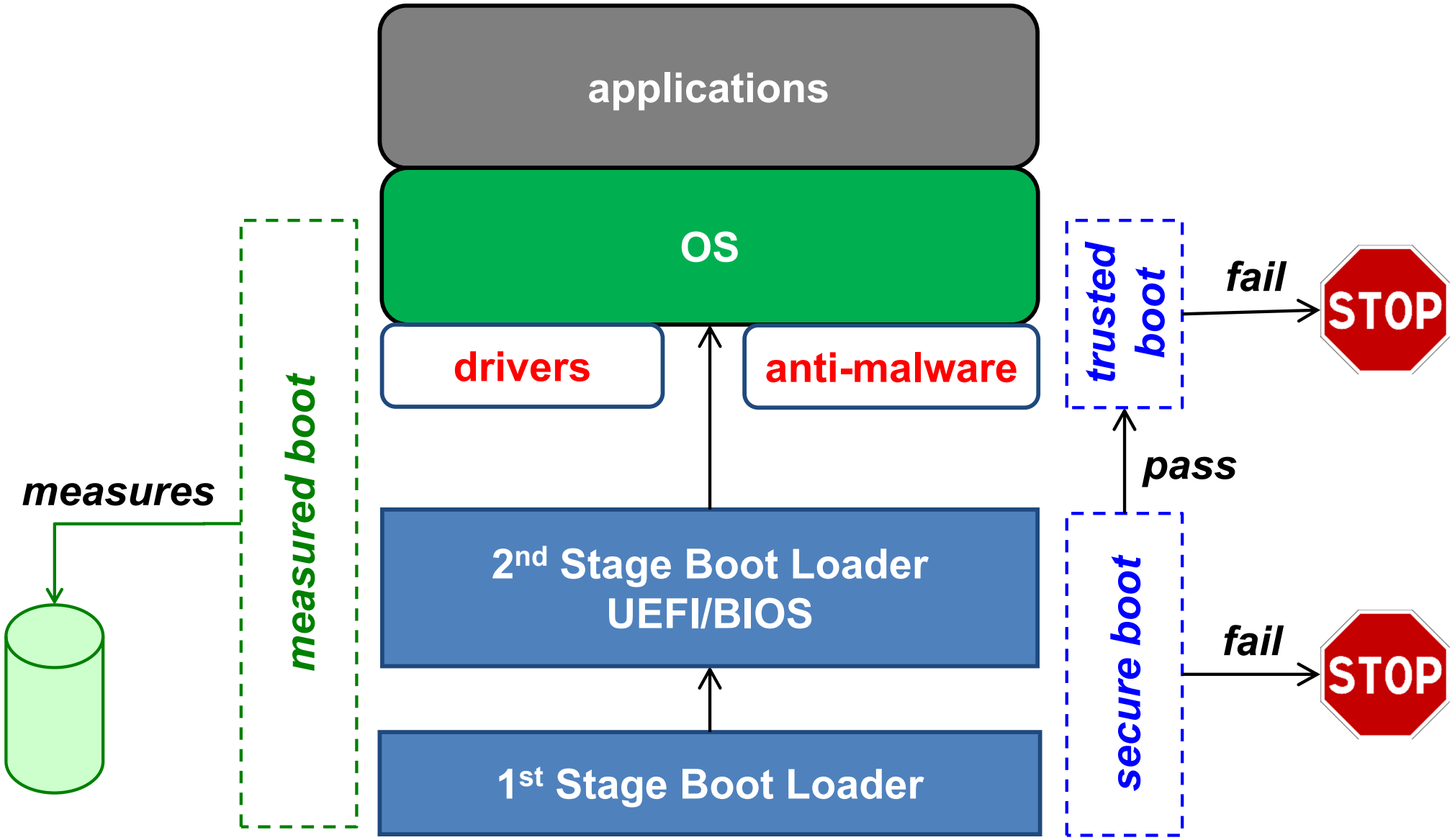
HW root of trust for firmware protection (II)

- the external crypto chip validates the BIOS in SPI flash post power ON
- once validation is successful, only then the x86 CPU will be out of reset, else remains in reset state
- this chip has a fusing option so that we can fuse one public key hash which will be used to verify the signature of hash file stored in the signature region
- validation flow is similar to the BIOS self-integrity check, except that that the external chip is doing the validation which makes it the real HW root of trust

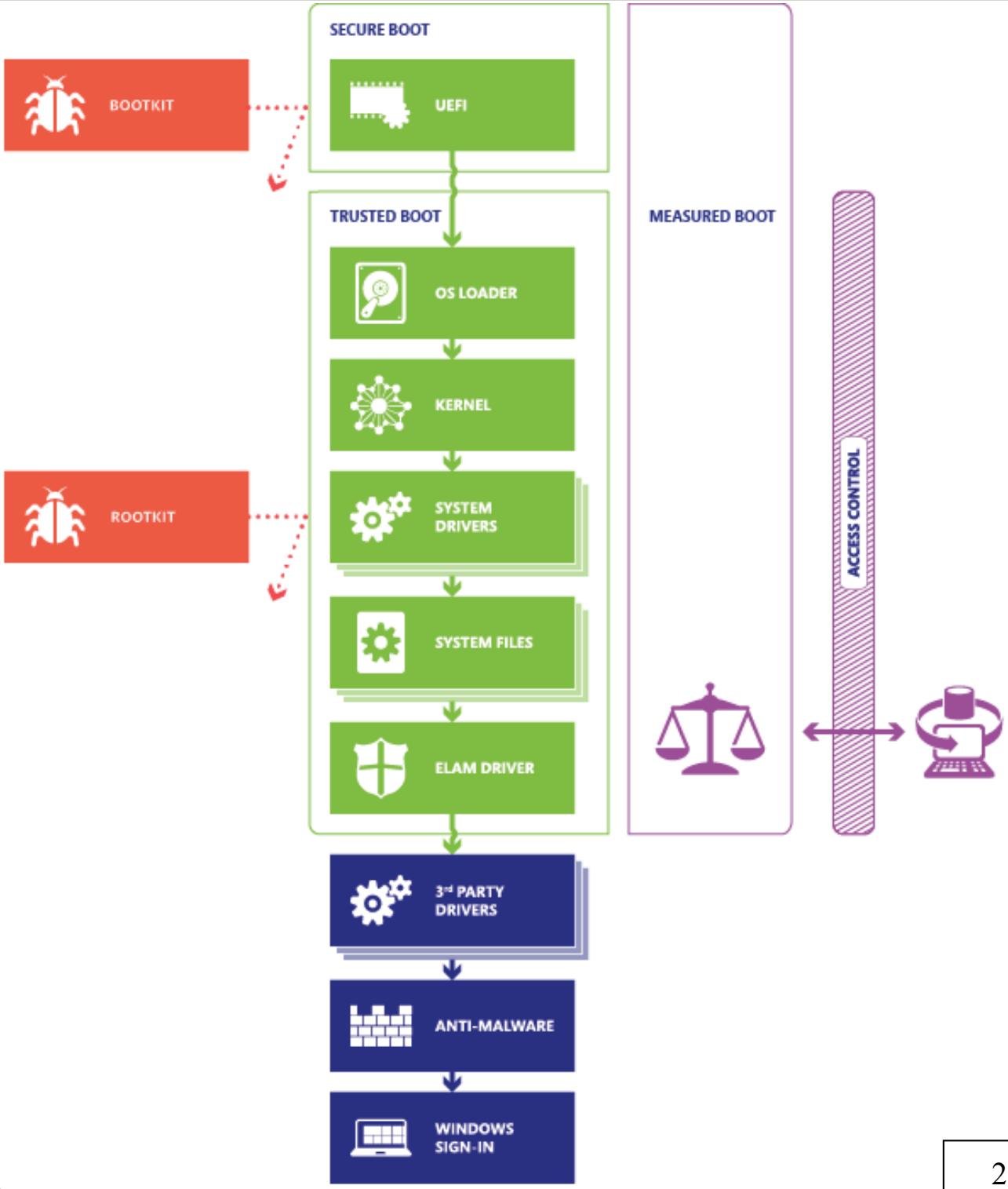
Boot types

- **plain boot … no security ☹**
- **secure boot**
  - firmware verifies signature and will halt platform if the verification fails
  - mostly hardware-based, verifies up to the OS-loader
- **trusted boot**
  - OS verifies the signature of OS components (e.g. drivers, antimalware) and will stop operations if the verification fails
  - mostly software-based, verifies up to OS operational state
- **measured boot**
  - measures all components executed from boot up to X
  - never stops operations but can securely report measures to an external verifier

Boot types


applications
OS
drivers
anti-malware
2nd Stage Boot Loader
UEFI/BIOS
1st Stage Boot Loader
measures
measured boot
trusted boot
fail
STOP
pass
secure boot
fail
STOP

Windows boot protection



What is Trusted Computing?

- **a trusted component/platform is one behaving as expected**
  - trust is not the same as good/secure
    - the behaviour needs to be verified against the expected one
  - attestation: verifiable evidence of the platform's state
  - Root of Trust: inherently trusted component
- **Trusted Computing defines schemes for establishing trust in a platform that are based on identifying its hardware and software components**
- **the Trusted Platform Module (TPM) provides methods for collecting and reporting these identities**
- **a TPM used in a computer system reports on the hardware and software in a way that allows determination of expected behaviour and, from that expectation, establishment of trust**

Trusted Computing Base (TCB)

- **collection of system resources (hardware and software) that is responsible for maintaining the security policy of the system**
  - an important attribute of a TCB is its ability to prevent itself from being compromised by any hardware or software that is not part of the TCB
- **the TPM is *not* the TCB of a system**
  - rather, a TPM is a component that allows an independent entity to determine if the TCB has been compromised
  - in some uses, the TPM can help prevent the system from starting if the TCB cannot be properly instantiated

Root of Trust (RoT)

- **a component that must always behave in the expected manner because its misbehaviour cannot be detected**
  - building blocks for establishing trust in a platform
- **Root of Trust for Measurement (RTM)**
  - measure and send integrity measurement to RTS
  - usually the CPU executes the CRTM (Core Root of Trust for Measurement) at boot as the first piece of BIOS/UEFi code, to start the chain of trust
- **Root of Trust for Storage (RTS)**
  - shielded/secured storage
- **Root of Trust for Reporting (RTR)**
  - entity that securely reports the content of RTS

Chain of trust

- **component A measures component B**
  - stores the measurement in RTS
- **component B measures component C**
  - stores the measurement in RTS
- **and so on …**
- **note:**
  - component A is typically the CRTM, which is part of the TCB

- **by using RTR, a verifier can securely retrieve B's and C's measurements from the RTS**
  - B and C can only be trusted if A is trustworthy

Trusted Platform Module (TPM) overview

- **inexpensive (< $1)**
  - available on most servers, laptop, PC
- **tamper-resistant**
  - but not tamper-proof
- **it is NOT a high-speed cryptographic engine**
  - rather slow
- **certified Common Criteria EAL4+**
- **it's a “passive component”, needs to be driven by the CPU**
  - cannot prevent boot
  - … but can protect data and securely report them
  - so TPM is both RTS and RTR
  - … but it's not RTM

TPM features

- RTS ~ secure storage (extend-only)
- RTR ~ report content of RTS with digital signature
- hardware random number generator
- crypto algorithms (hash, MAC, symmetric and asymmetric encryption) … but it's NOT a crypto accelerator (slow!)
- secure generation of cryptographic keys for limited uses
- binding (data encrypted using the TPM bind key, a unique RSA key descending from a storage key)
- sealing (similar to binding, but in addition, specifies the TPM state for the data to be decrypted, i.e. unsealed)
- computer programs can use a TPM to authenticate hardware devices, since each TPM chip has a unique and secret Endorsement Key (EK) burned in as it is produced

TPM-1.2

- fixed set of algorithms (SHA-1, RSA, optionally AES)
- one storage hierarchy for platform user
- one root key (SRK, RSA-2048)
- hardware identity via the built-in Endorsement Key (EK)
- sealing to PCR value

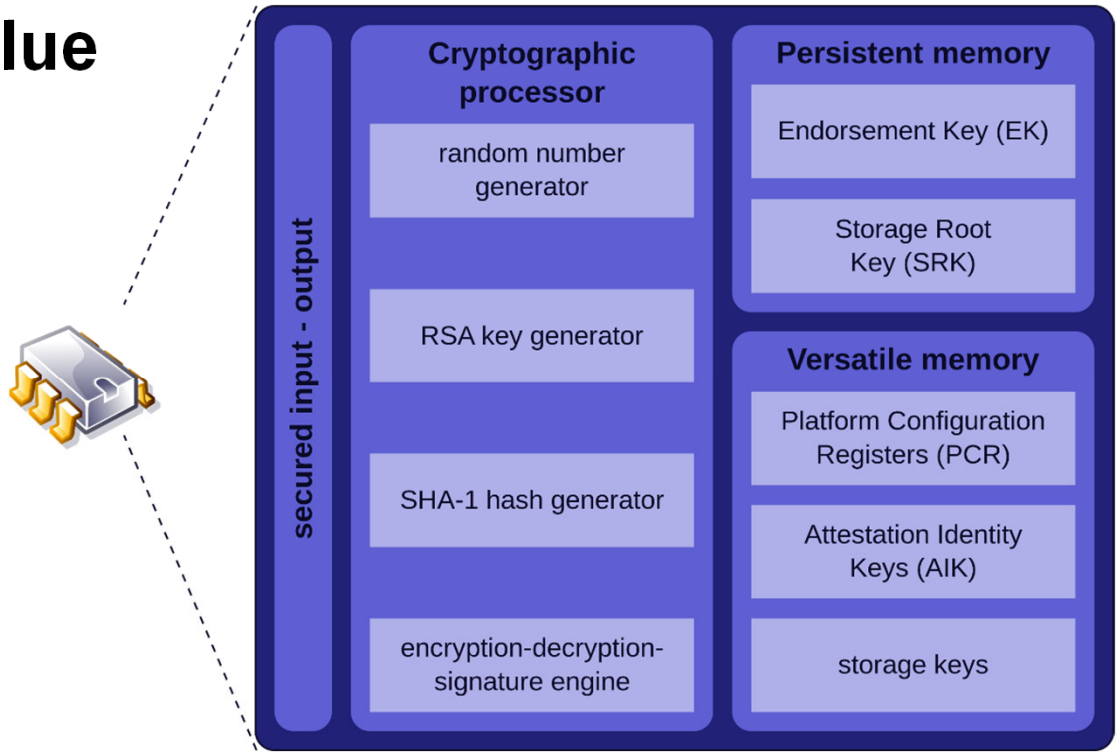


figure source = Wikipedia
author = Eusebius (Guillaume Piolle)

TPM-2.0

- **cryptographic agility (SHA-1 and SHA-256, RSA, ECC-256, HMAC, AES-128, …)**
- **three key hierarchies (platform, storage, and endorsement)**
- **multiple keys and algorithms per hierarchy**
- **policy-based authorization**
- **platform-specific specifications for**
  - PC client
  - mobile
  - automotive-thin

Implementations of TPM-2.0

- **Discrete TPM = dedicated chip**
  - implements TPM functionality in its own tamper resistant semiconductor package
- **Integrated TPM = part of another chip**
  - not required to implement tamper resistance (Intel has integrated TPMs in some of its chipsets)
- **Firmware TPM = software-only solution**
  - runs in a CPU's trusted execution environment (AMD, Intel, and Qualcomm have implemented firmware TPM)
- **Hypervisor TPM = virtual TPM provided by a hypervisor**
  - runs in an isolated exec. env. (comparable to a firmware TPM)
- **Software TPM = software emulator of TPM**
  - useful for development purposes.

TPM-2.0 three hierarchies

- **Platform Hierarchy**
  - for platform's firmware
  - NV storage for keys and data
- **Endorsement Hierarchy**
  - for the privacy administrator
  - keys and data
- **Storage Hierarchy**
  - for the platform's owner (usually also the privacy administrator)
  - NV storage for keys and data
- **each hierarchy has:**
  - dedicate authorization (password) and policy
  - specific seed for generating the primary keys

Using a TPM for securely storing data

- **physical isolation**
  - storage in the TPM (i.e. in the NVRAM)
    - primary keys
    - permanent keys
  - very limited space
  - Mandatory Access Control
- **cryptographic isolation**
  - storage outside of the TPM (i.e. in the platform HDD/SSD)
    - keys or data
    - blob needs to be protected !!!
  - encrypted with a key controlled by the TPM
  - Mandatory Access Control

TPM objects

- **primary keys:**
  - endorsement keys, storage keys
  - derived from one of the primary seeds
    - the TPM does not return the private value
  - can be re-created by using the same parameters
    - assuming the primary seed has not been changed
- **keys & sealed data objects (SDO)**
  - protected by a Storage Parent Key (SPK)
    - SPK needed in the TPM to load/create a key/SDO
  - randomness comes from the TPM RNG
  - the TPM returns the private part – protected by the SPK
    - the private part needs to be stored somewhere

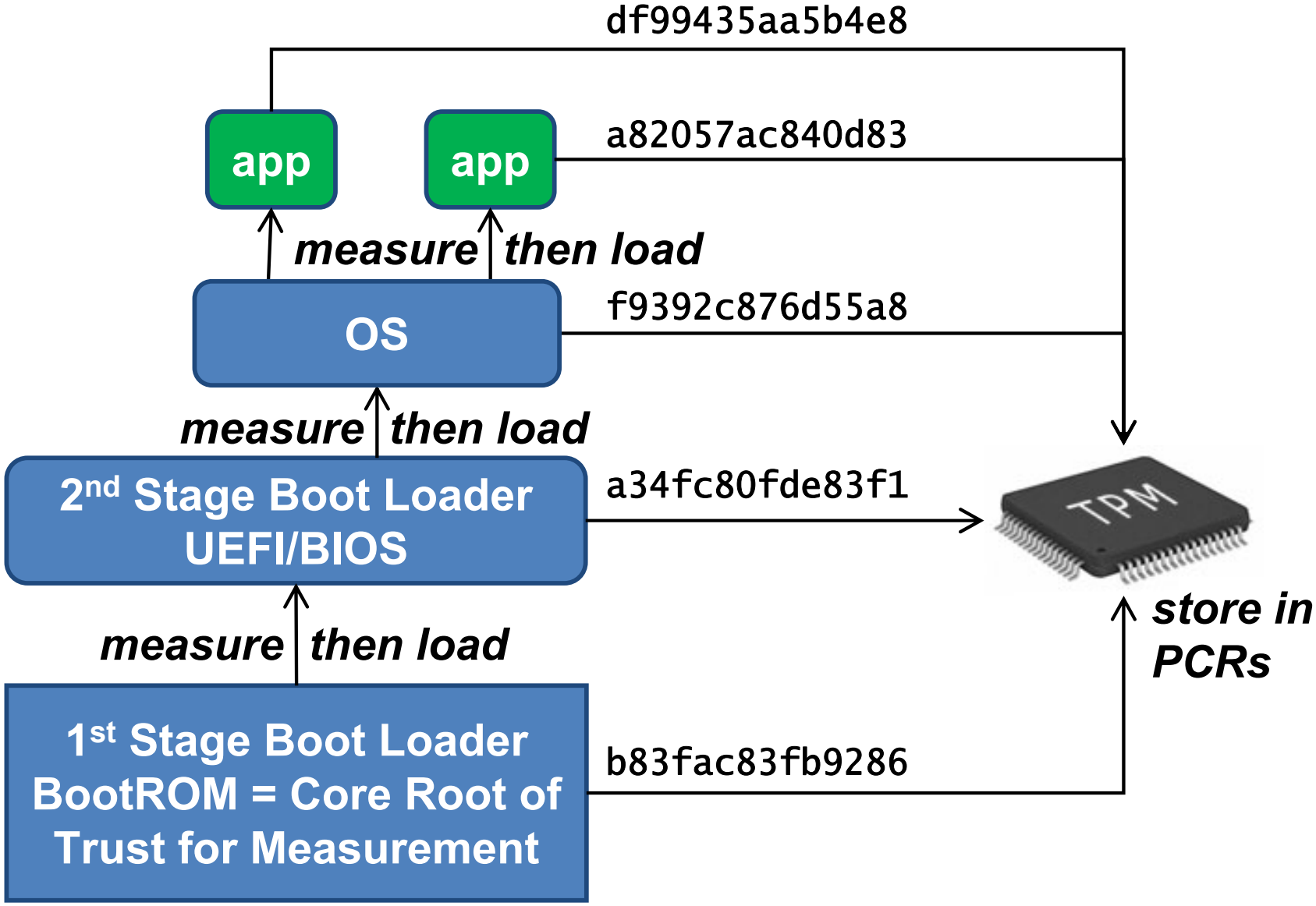
TPM object's area

- **public area**
  - used to uniquely identify an object
- **private area:**
  - object's secrets
  - only exists in the TPM
- **sensitive area:**
  - encrypted private area
  - use for storage outside of the TPM

TPM Platform Configuration Register (PCR)

- **TPM's implementation of RTS**
- **core mechanism for recording platform integrity**
  - only reset at platform reset (or with hardware signal)
  - malicious code cannot take its measurement back
- **PCRs are extended using a cumulative hash**
  - PCR_new = hash(PCR_old || digest_of_new_data)
  - in short this is the EXTEND operation
- **can be used to gate access to other TPM objects**
  - e.g. BitLocker seals disk encryption keys to PCR values

Measured boot

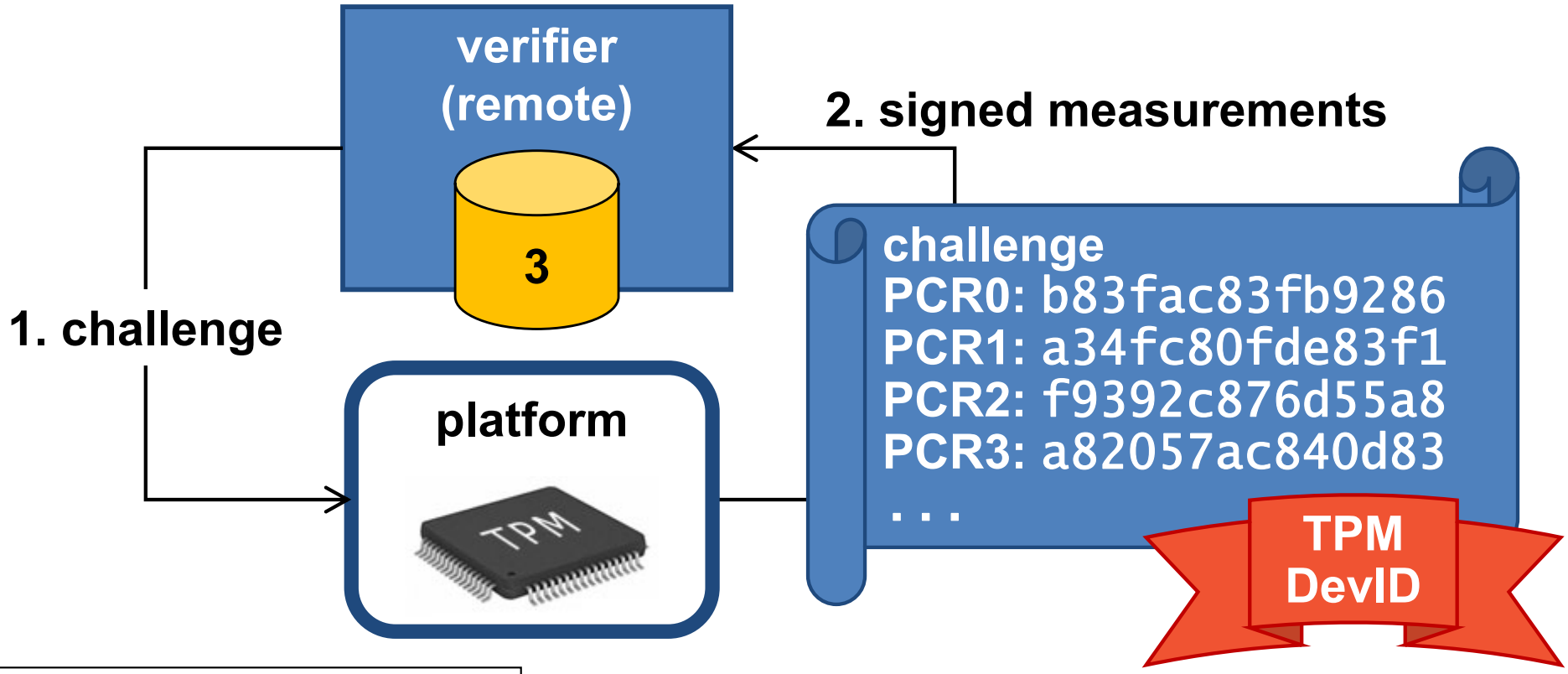


Remote attestation procedure

(1) challenge (=nonce)

(2) measurements (and nonce) signed with the device's key

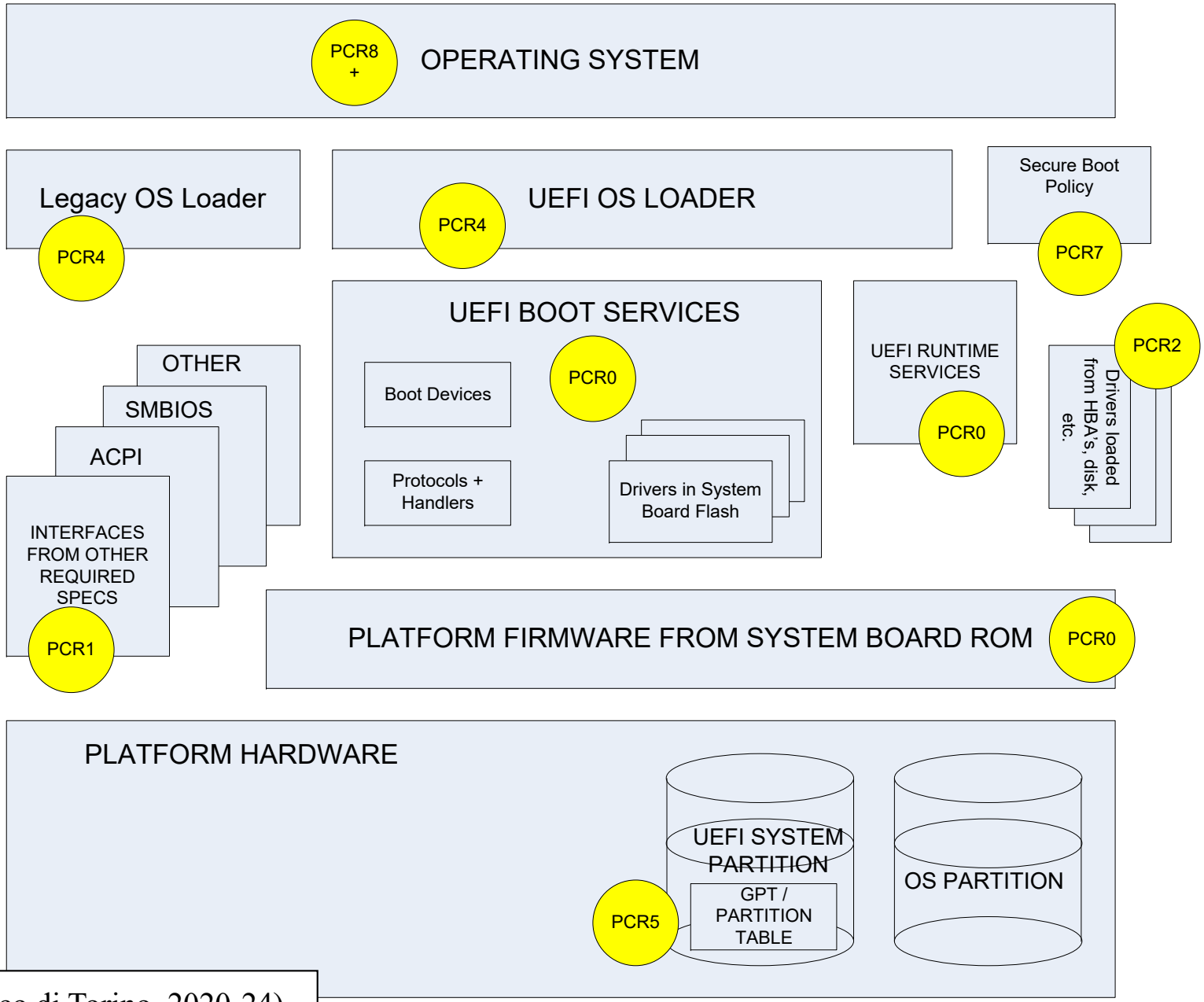
(3) validate signature (crypto + ID) and check measurements against Reference Measurements (golden values)



Management of Remote Attestation

- **only boot attestation (static) or periodic (dynamic) too?**
  - consider the attack model (runtime vulnerabilities)
- **periodicity of the operation**
  - consider the speed of attack
  - implementation limits (signature + protocol + DB lookup)
  - currently in the range of some seconds (due to TPM slowness)
- **whitelist generation**
  - difficult in general, not so difficult for limited environments
    - IoT, edge device, SDN, NFV, …
  - labels (good, old, buggy, vulnerable, …)
  - include configurations too (e.g. from MANO, netman)
    - easy if file-based, difficult if memory-based

TCG PC Client PCR use (architecture)

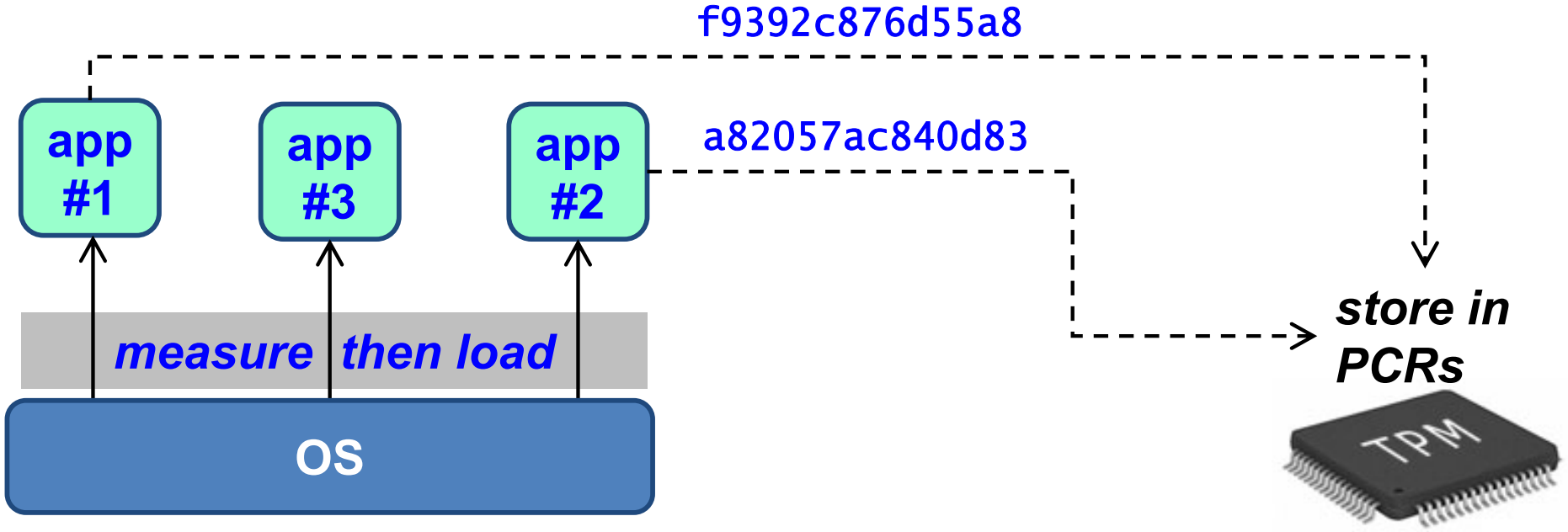


TCG PC Client PCR use (detail allocation)

PCR Index	PCR Usage
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration
2	UEFI driver and application Code
3	UEFI driver and application Configuration and Data
4	UEFI Boot Manager Code (usually the MBR) and Boot Attempts
5	Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
8-15	Defined for use by the Static OS
16	Debug
23	Application Support

Measured execution

- **problem:**
  - not enough PCRs
- **solution:**
  - use just one PCR!
- **… but now the PCR value depends on the execution order (!!!)**



Linux's IMA

- **Integrity Measurement Architecture (IMA)**
- **extends attestation to dynamic execution (e.g. applications)**
- **Collect**
  - measure a file before it is accessed
- **Store**
  - add the measurement to a kernel-resident list (ML, Measurement List) and extend the IMA PCR (PCR 10)
- **(Appraise)**
  - enforce local validation of a measurement against a "good" value stored in an extended attribute of the file
- **(Protect)**
  - protect a file's security extended attributes (including appraisal hash) against off-line attacks

Linux's IMA details

- **extension of UEFI measured boot to the OS and applications**
  - Linux IMA uses PCR10
  - 1st measurement: boot_aggregate
    - hash of TPM's PCR 0 to 7 (i.e. UEFI-related PCRs)
- **measurement configuration through IMA template**
  - mostly ima-ng, but can be customized
- **exposed in the kernel's securityfs**
  - /sys/kernel/security/ima/ascii_runtime_measurements

PCR	template-hash	template	filedata-hash	filename-hint
10	91f34b5[…]ab1e127	ima-ng	sha1:1801e1b[…]4eaf6b3	boot_aggregate
10	8b16832[…]e86486a	ima-ng	sha256:efdd249[…]b689954	/init
10	ed893b1[…]e71e4af	ima-ng	sha256:1fd312a[…]6a6a524	/usr/lib64/ld-2.16.so
10	9051e8e[…]4ca432b	ima-ng	sha256:3d35533[…]efd84b8	/etc/ld.so.cache

Verification of the IMA ML

- **with IMA enabled, the attestation report contains not only nonce and PCR values but also the ML (Measurement List)**
- **… but the PCR10 value is variable, as it depends on (A) the applications executed, and (B) their order of execution**
- **… so the verifier computes the correct value by using the ML**
  - myPCR10=0
  - myPCR10 = extend (boot_aggregate)
  - foreach measure M of a component C
    - if (C not authorized) then alarm
    - if (M different from gold_measure(C)) then alarm
    - myPCR10 = extend (M)
- **if (myPCR10 == PCR10) then OK else alarm**

Size and variability of the TCB

- **the Trusted Computing Base (TCB) is the smallest amount of code (and hardware, people, processes, ...) to be trusted to meet the security requirements**
- **confidence in the TCB can be increased through**
  - static verification
  - code inspection
  - testing
  - formal methods
- **all these methods are expensive, so reducing the complexity of the TCB is important but it is not sufficient**
- **the TPM tries to create a TCB via the CRTM … but the TCB has become too large and too much dynamic**
  - two “identical” computers could have different measures

Dynamic Root of Trust for Measurement

- **rather than trust everything since BIOS, reset CPU and start measuring from that point on**
- **TPM v1.2 added dynamic PCRs (17–23)**
  - set to -1 on boot
  - can be reset by OS to 0
- **PCR17 is special**
  - only set by calling SKINIT (AMD-V) or SENTER (Intel TXT)
    - disable DMA, interrupts, debugging
    - measure and execute Secure Loader Block

Dynamic Root of Trust for Measurement

- **Dynamic Root of Trust for Measurement (DRTM)**
- **special processor command**
  - SENTER (Intel TXT, executes the SINIT binary module)
  - SKINIT (AMD SVM)
- **stops all processing on the platform**
- **DRTM hashes contents of memory region**
  - stores measurement in dynamic PCR
- **transfer control to specified location in memory**
- **also called Late Launch**
- **helps to avoid the problem with PCR values incorrect when firmware is updated (and the consequent problem with sealed data)**

Hypervisor TEE

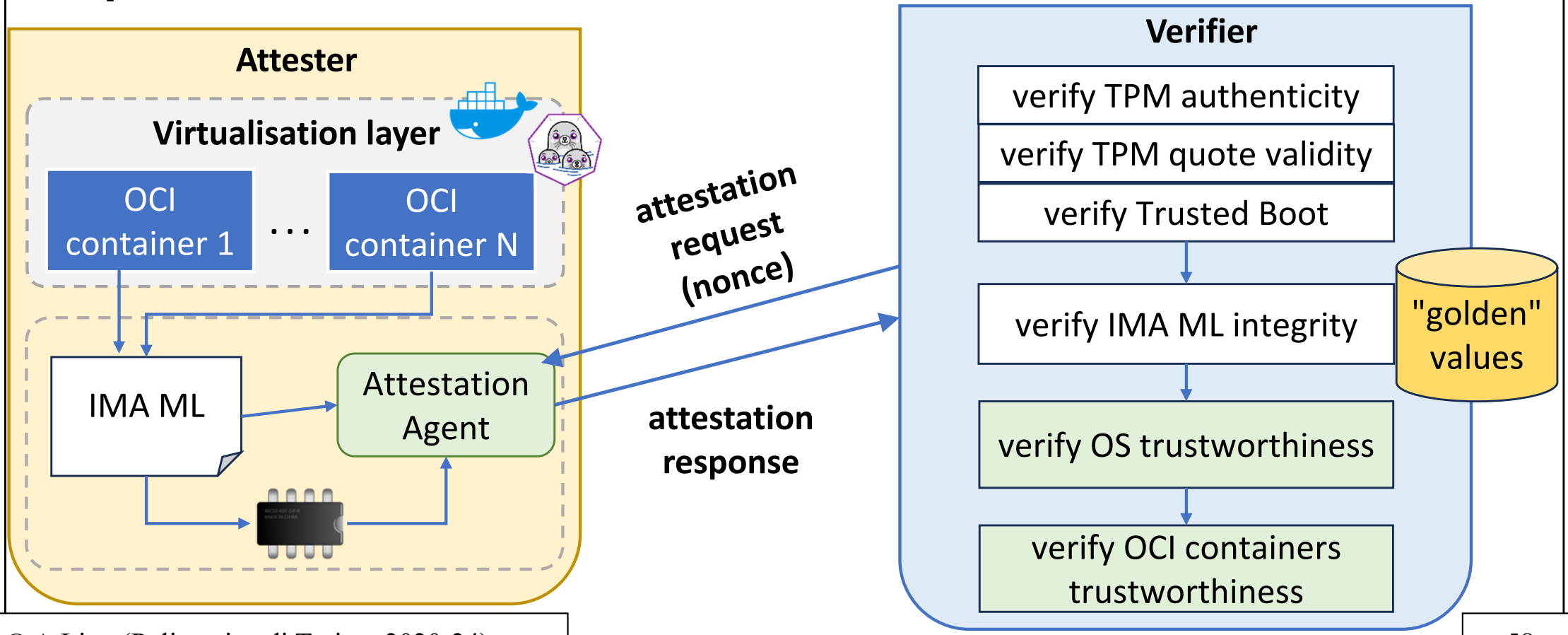
- **DRTM was intended to allow loading of a hypervisor**
  - e.g. Xen or VMWare ESX
  - hypervisor loads and isolates VMs
  - TPM can attest the hypervisor
  - TPM sealed storage can be released only to hypervisor once it has been loaded properly
- **may be useful for cloud computing**
- **hypervisor is still a huge amount of code to validate (Xen contains a full copy of Linux; VMware is of similar size)**

RA in virtualized environments

- **having a hardware RoT is an important point for security**
- **full virtualization (i.e. VM)**
  - often offers just a software version of the RoT (e.g. vTPM by Xen, Google, VMware)
  - we need a strong link between the vTPM and the pTPM
    - deep attestation (hardware-based)
    - sealed objects rooted in pTPM to protect the vTPM
    - requires extension of the usual TCG-defined interfaces (on-going work)
- **… but if we adopt light virtualization (e.g. Docker containers)**
  - a different solution is possible because hardware (including TPM) is shared

RA for OCI containers

- not tied to a specific containerization technology
- transparent to the container runtime and to the containerized workloads
- provides RA of host + containers, based on a hardware RoT



RA for OCI containers: implementation

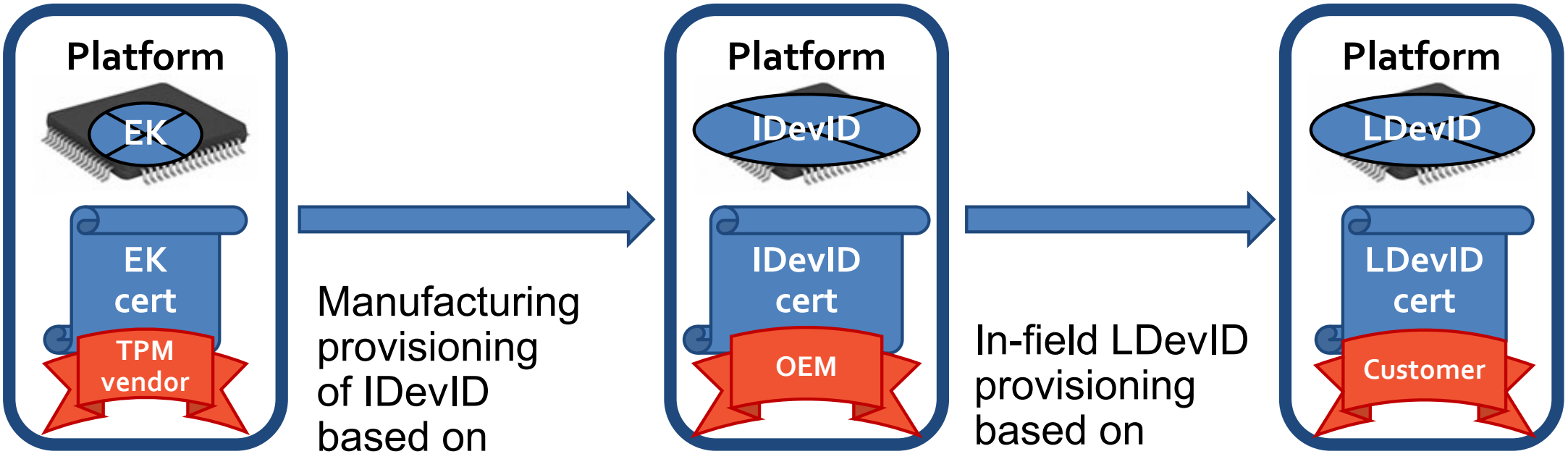
- **new IMA template, ima-dep-cgn**
  - dependencies: entry belongs to the host or to a container?
  - control-group-name: identifies the specific container
  - template-hash: digest calculated with an algorithm other than sha1 (sha256, sha512, ...)

Container ID

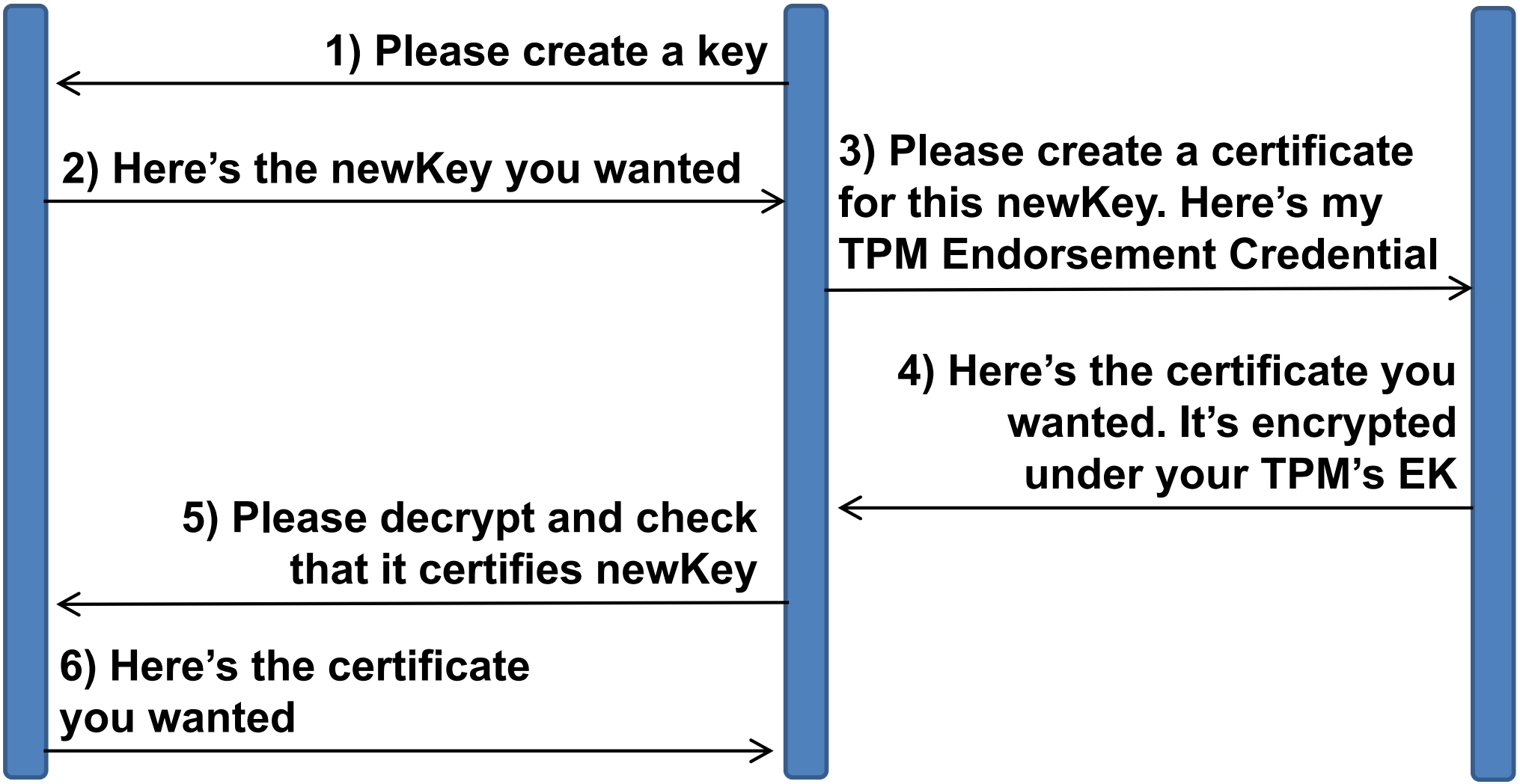
PCR	template-hash	template-name	dependencies	cgroup-name	filedata hash	filename hint
10	**sha256**:8af8cf[...]	ima-dep-cgn	runc:/usr/bin/containerd-shim-runc-v2:...	8b2ad985209b51aea87[...]	[...]	/usr/bin/bash
10	**sha256**:1590d[...]	ima-dep-cgn	kworker/u8:3:kthreadd:swapper/0	/	[...]	/usr/bin/kmod
10	**sha256**:01c73[...]	ima-dep-cgn	/usr/bin/bash:/usr/bin/containerd-shim-runc-v2:...	5cbc6f873774aa67fcfa[...]	[...]	/usr/lib/[...]/ld-2.31.so

Credentials chain of trust

- **from the TPM vendor to a customer-usable certificate**
- **IEEE 802.1AR – Secure Device Identity using TPM**
  - allows Zero-Touch management of a platform



TPM-2.0 Make/Activate Credentials

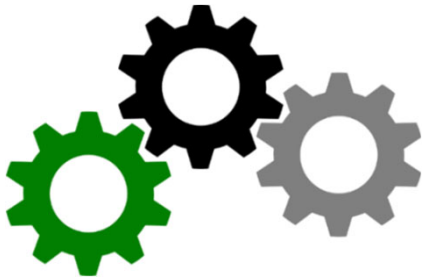


TPM basic authorization mechanism

- **direct password-based authorization**
  - for single commands
- **password-based HMAC to authenticate commands and responses**
  - with caller_nonce and TPM_nonce (to prevent replay)
- **but the TPM knows a lot about the platform states and can be configured to:**
  - prevent object usage unless selected PCRs have specific values
  - prevent object usage after a specific time
  - prevent object usage unless authorized by multiple entities (i.e. key holders)

Which is your trust perimeter?

- **installation (or download) time**
  - check signature
- **load time**
  - measure components when loaded for execution
    - what is "executable"?
- **run time (components that change their behaviour while running)**
  - measure configuration files (when loaded or re-loaded)
    - beware of caching!
  - measure in-memory configuration (e.g. filtering or forwarding rules modified by CLI or network protocol)
    - needs appropriate firmware/host

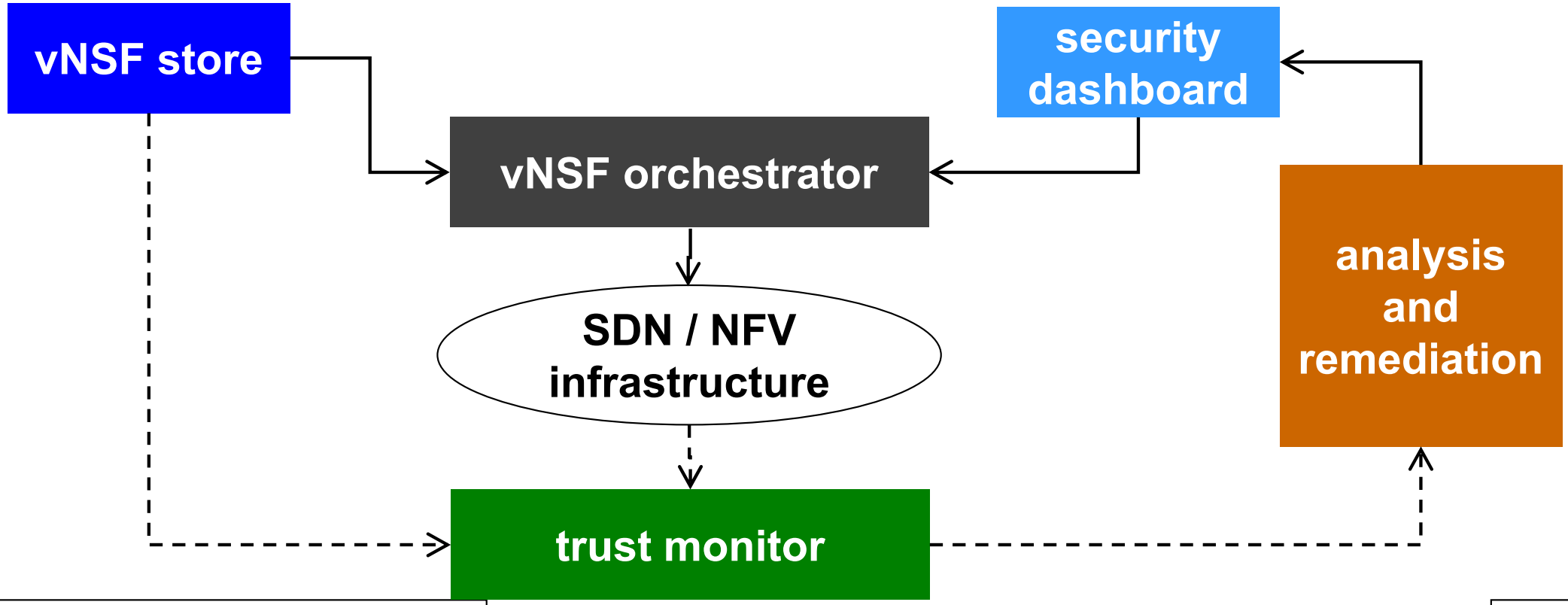


Audit and forensic analysis

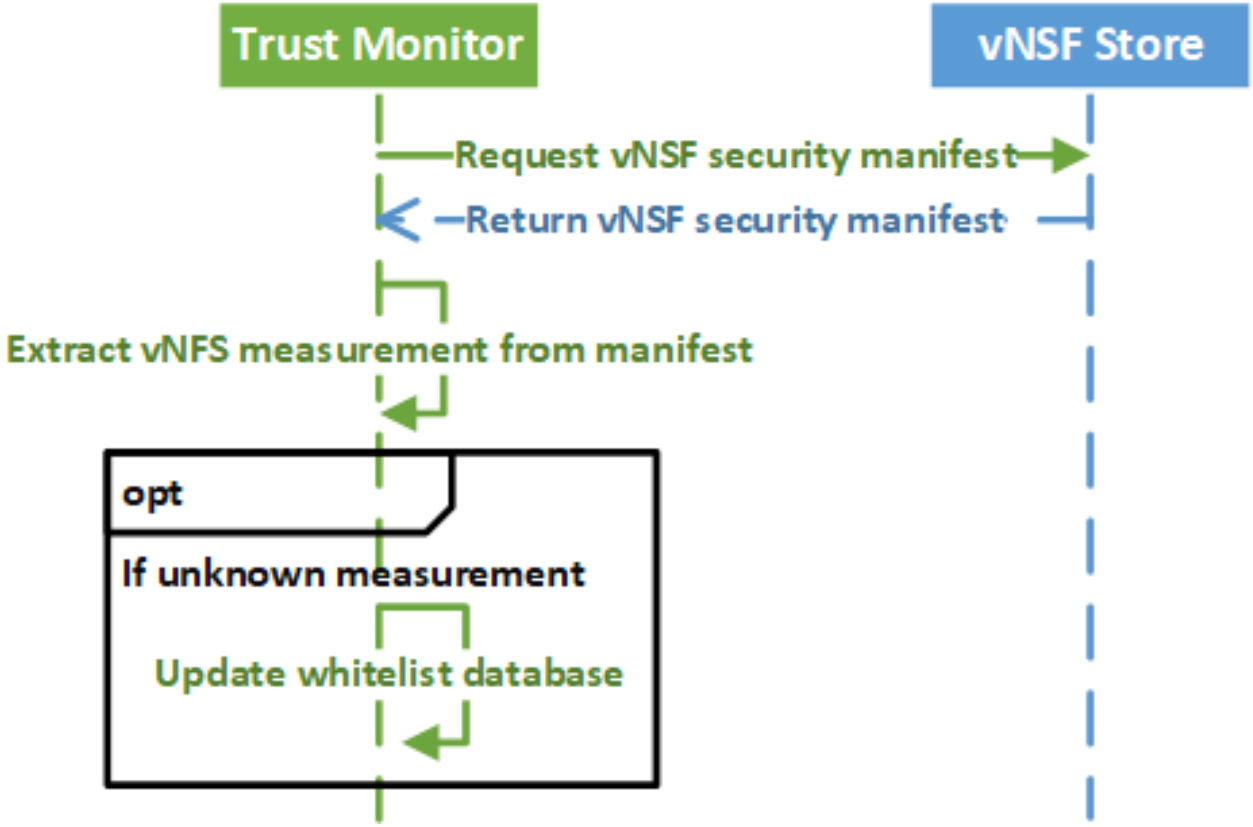
- **node (e.g. IoT, ECU) / network behaviour cannot be given for granted any more**
- **increasingly important as more intelligence / computation is moved into the edge nodes / network**
- **open questions:**
  - system state at time T?
  - network path + processing for user U at time T?

SHIELD project: trust monitor

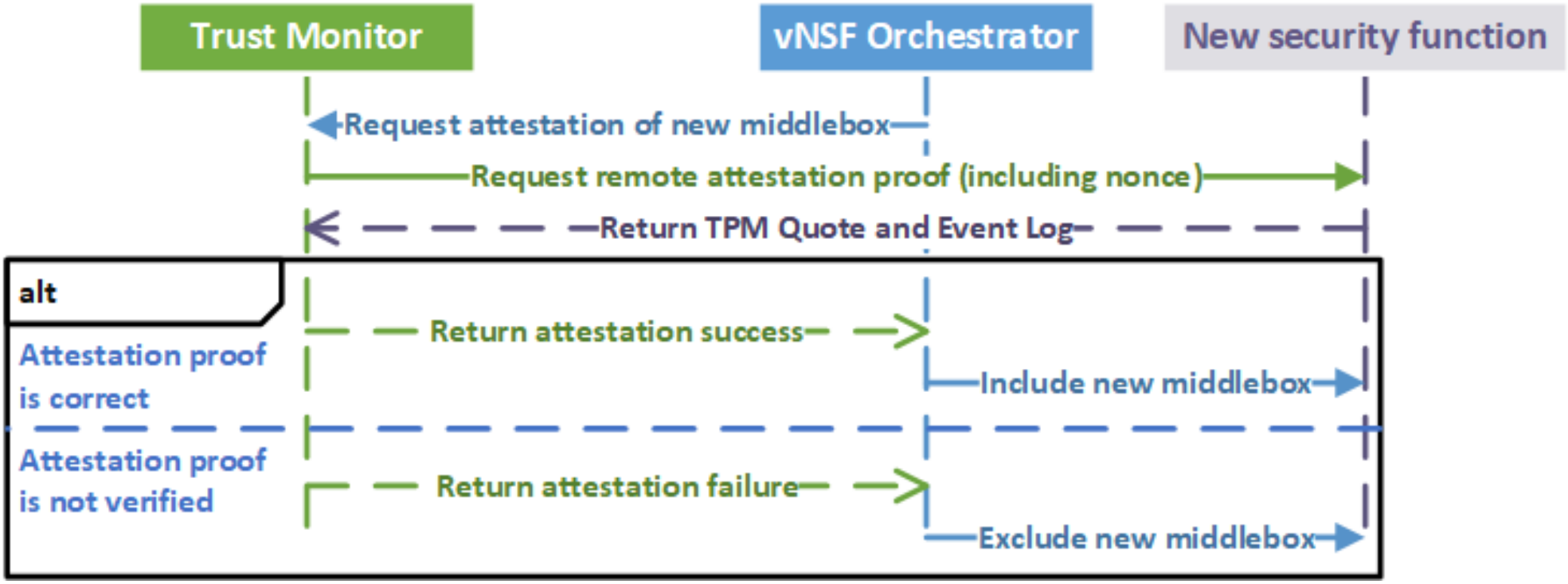
- **SHIELD H2020 project = NFV-based security infrastructure**
  - vNSF (security controls for monitoring, action, and reaction)
  - remote attestation to verify integrity state of the infrastructure



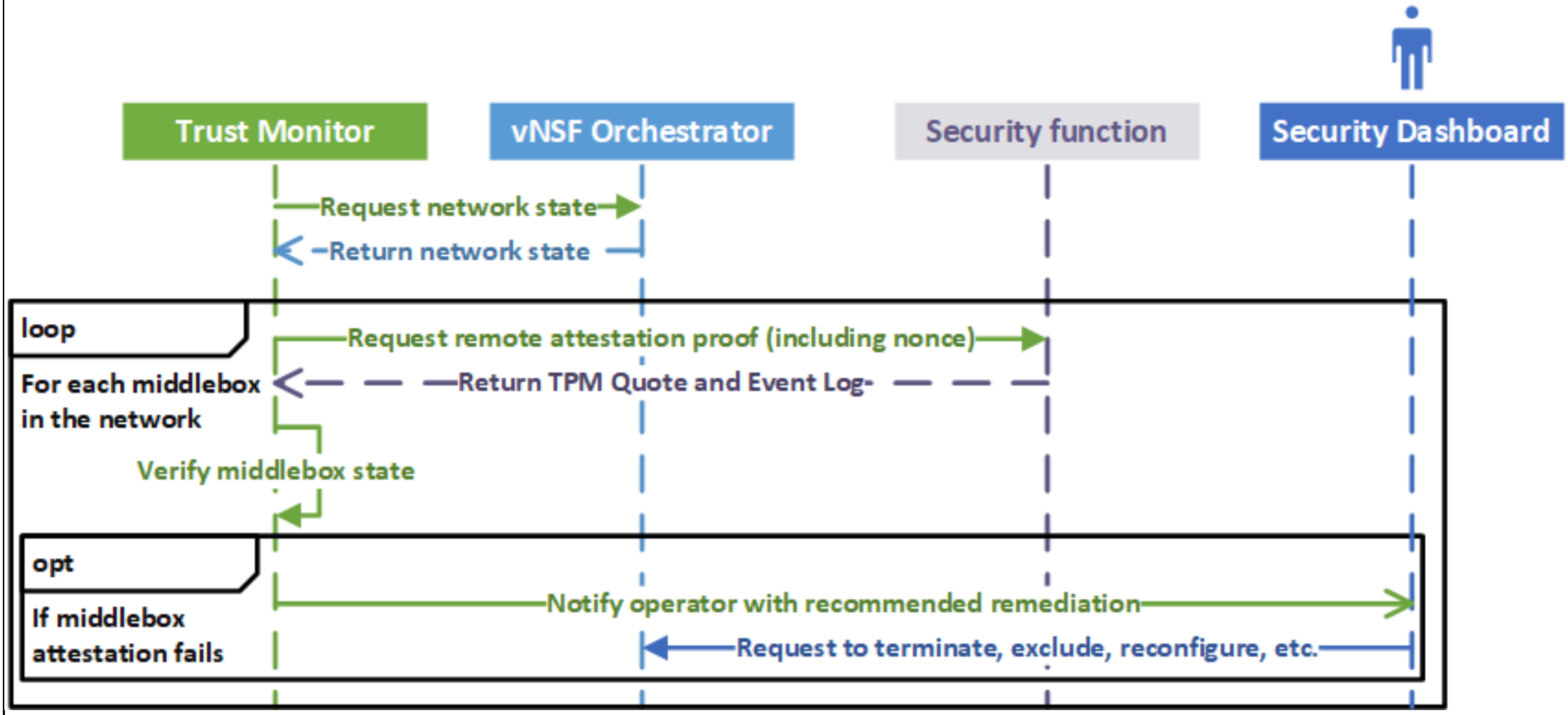
Golden value creation



Initial deployment of a security function



Periodic attestation of security functions



Implementing remote attestation

Antonio Lioy
< antonio.lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

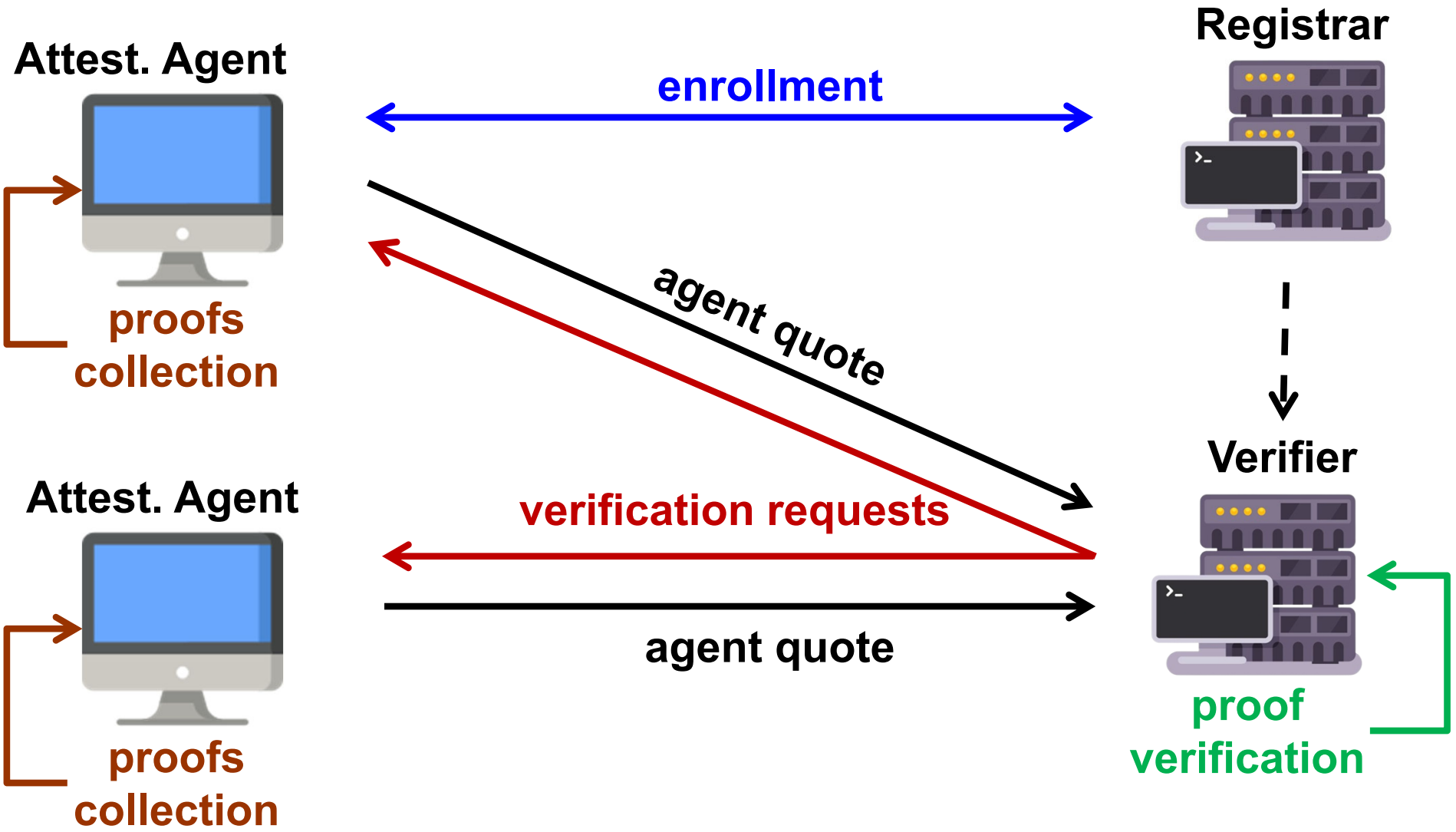
Keylime

- **open-source remote attestation project**
  - hosted at CNCF (Cloud-Native Computing Foundation)
  - cloud oriented
  - TPM based
  - highly scalable RA framework
  - straightforward architecture
- **features**
  - remote boot attestation
  - Linux IMA support (periodic runtime attestation)
  - registration of multiple agents to a Verifier
  - [certificate infrastructure]

Keylime: structure

- **Attester Agent**
  - retrieve TPM quote
  - collects other necessary data (e.g. IMA list)
  - [listen for revocation messages]
- **Registrar**
  - manages agent enrollment, providing a UUID
  - handle keys (Agent Key AK, Endorsement Key EK)
- **Verifier**
  - attests the platforms (by talking to the agents)
  - [sends revocation messages (agent leaves trusted state)]
- **tenant**
  - CLI management tool for agent

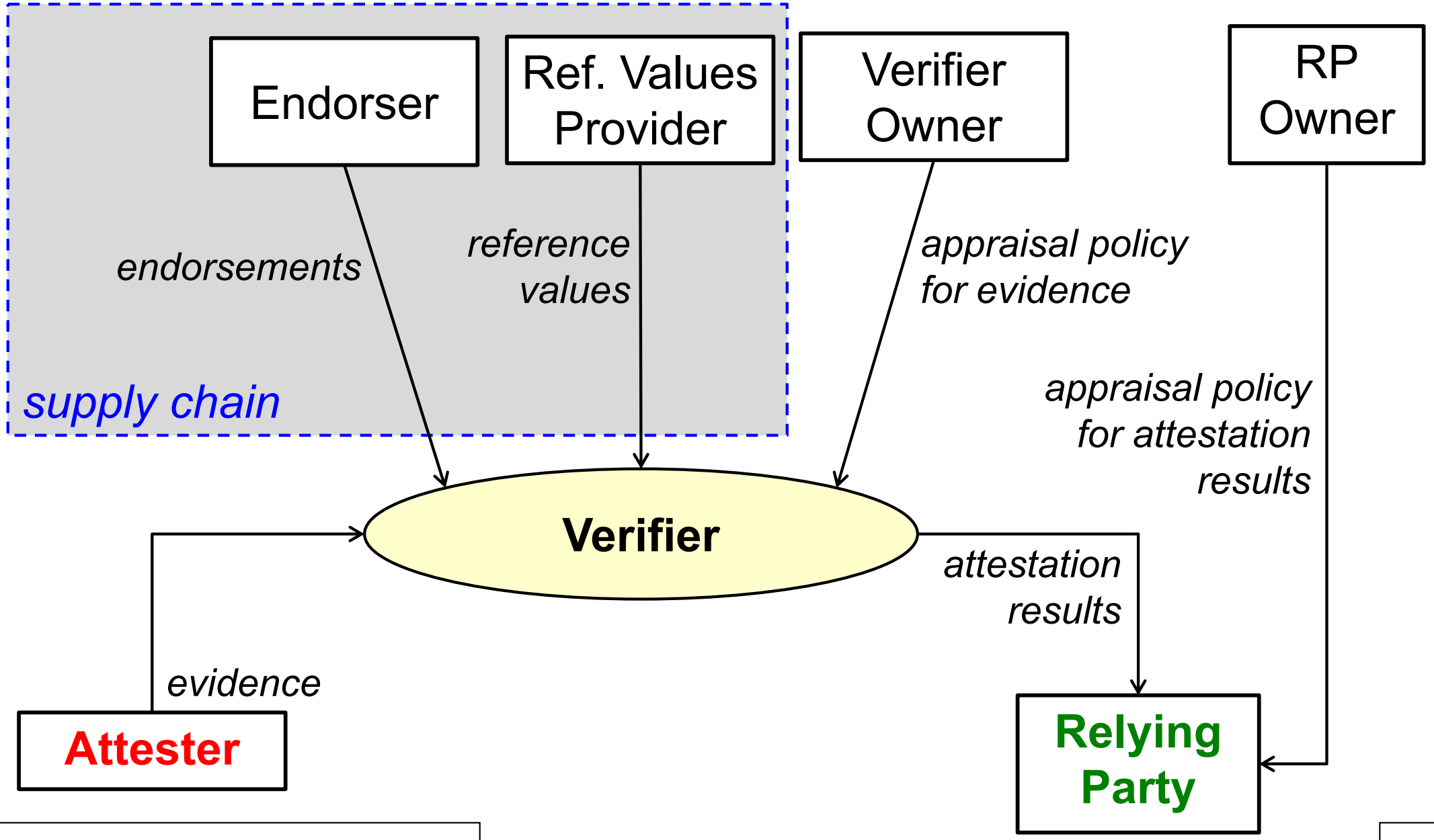
Keylime: schema



Remote ATestation procedureS – RATS (I)

- **proposed by IETF**
  - support for different platforms at load time
- **defines the actors in RA procedures**
  - Attester, Relying Party, Verifier, Relying Party Owner, Verifier Owner, Endorser, Reference Value Provider
- **defines topological patterns**
  - Background Check Model
  - Passport Model
- **RFC-9334 "RATS Architecture"**
  - many other rfc-drafts going-on, about various aspects (data formats, procedures, attestation models, …)

Remote ATestation procedureS – RATS (II)

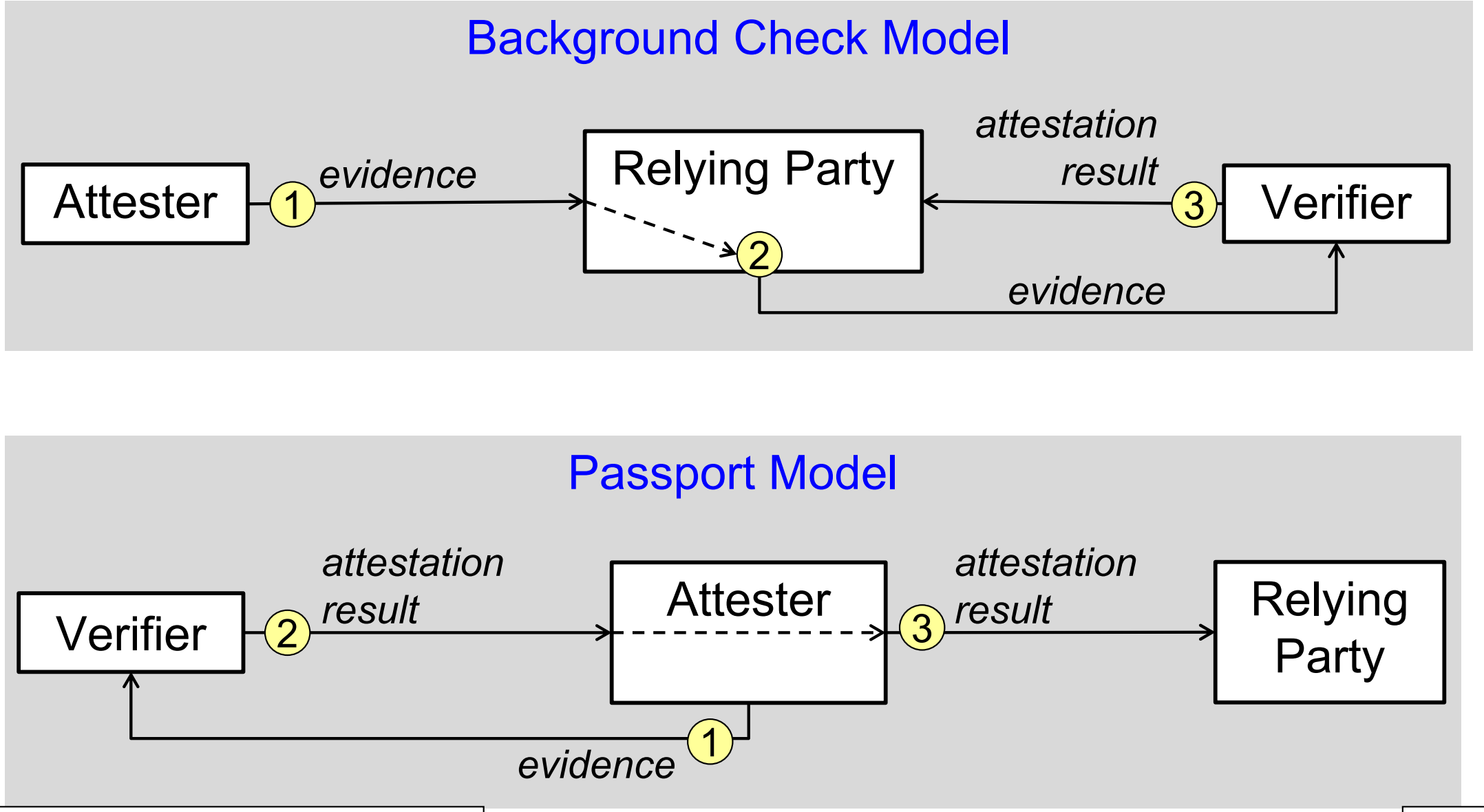


RATS – main roles

- **Attester**
  - generates Evidence when attestation needed
    - requested by the Relying Party
    - needed for accessing a service (e.g. OAuth)
- **Verifier**
  - compares Evidence against Reference Values (using Appraisal Policy)
  - uses Endorsements to identify valid attesters
- **Relying Party**
  - evaluate Attestation Results using RP Policy

- **note: RP and Verifier may be part of the same service**

Remote ATestation procedureS – RATS (III)



Attestation-related IETF working groups

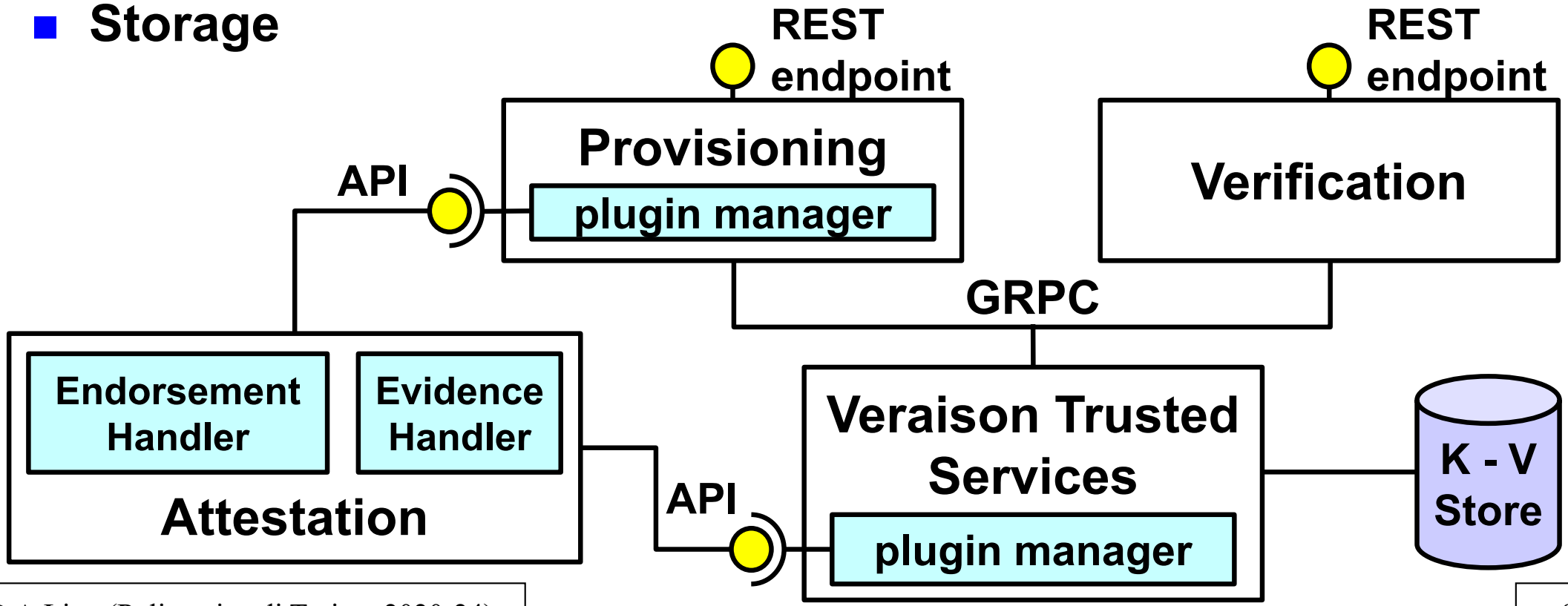
- **RATS = https://datatracker.ietf.org/wg/rats/about/**
- **Trusted Execution Environment Provisioning (TEEP)**
  - https://datatracker.ietf.org/wg/teep/about/
- **Software Update for Internet of Things (SUIT) =**
  - https://datatracker.ietf.org/wg/suit/about/
  - manifest format defined in SUIT used in TEEP
- **Limited Additional Mechanisms for PKIX and SMIME (LAMPS)**
  - https://datatracker.ietf.org/wg/lamps/about/
- **Web Authorization Protocol (OAuth)**
  - https://datatracker.ietf.org/wg/oauth/about/
- **Privacy Pass**
  - https://datatracker.ietf.org/wg/privacypass/about/

Veraison (VERificAtIon of atteStatiON)

- **open-source project**
  - enhance consistency for Verification Service
  - implementation of various standards
  - set of library for customizations
  - generated by ARM ATG, then adopted by the Confidential Computing Consortium in the Linux Foundation
- **support for different architectures and RoT implementations**
  - no standard agent implementation
  - flexible structure for evidence provisioning
- **high customizability**
  - choose only necessary features
  - easy development of custom features

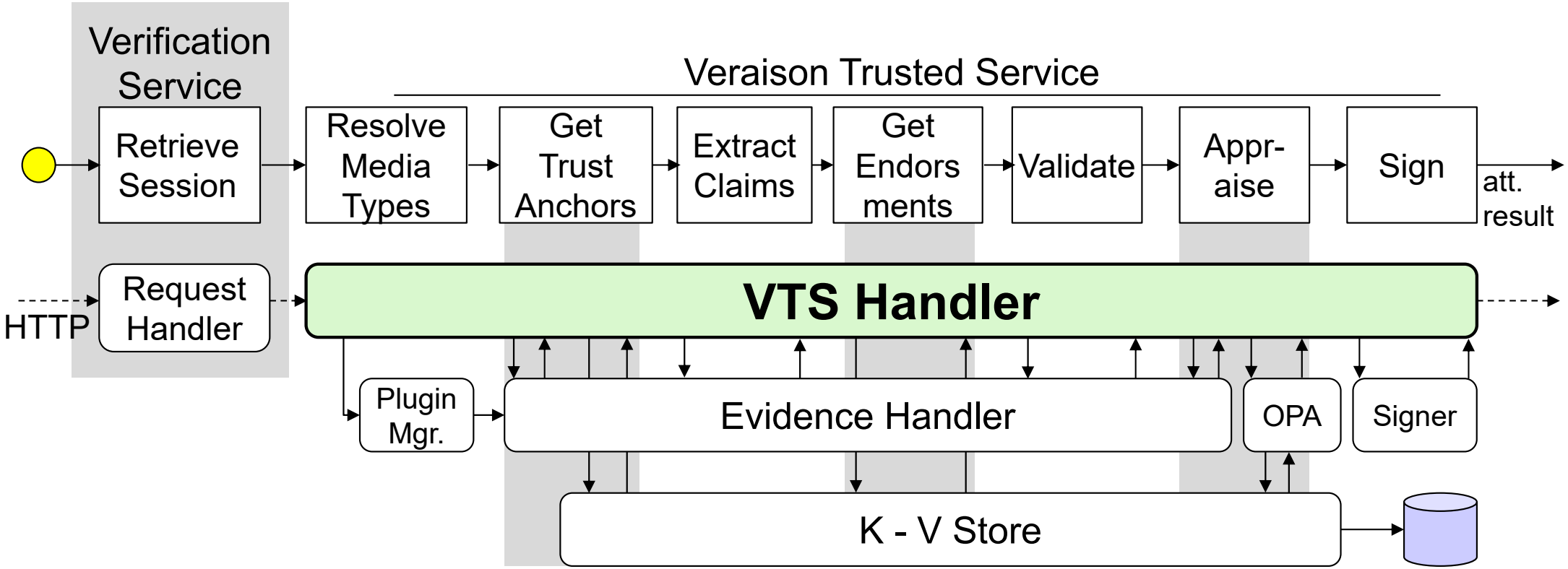
Veraison: architecture

- Provisioning service
- Verification service
- Attestation scheme
- Veraison Trusted Service (VTS)
- Storage



Veraison: verification

- **Verification service**
  - Policy and API calls
- **Veraison Trusted Service (VTS)**
  - data extraction, validation, sign of the attestation result



Veraison: data formats (I)

- **native support for various attestation types, by ingestion and production of various formats**
- **Entity Attestation Token (EAT) in CBOR or JSON format**
- **Evidence**
  - EAT PSA (by Platform Security Architecture, a security certification scheme for IoT)
  - EAT CCA (by ARM Confidential Compute Architecture)
  - TCG TPM
  - TCG DICE
  - AWS Nitro (by AWS for Nitro secure enclaves)

Veraison: data formats (II)

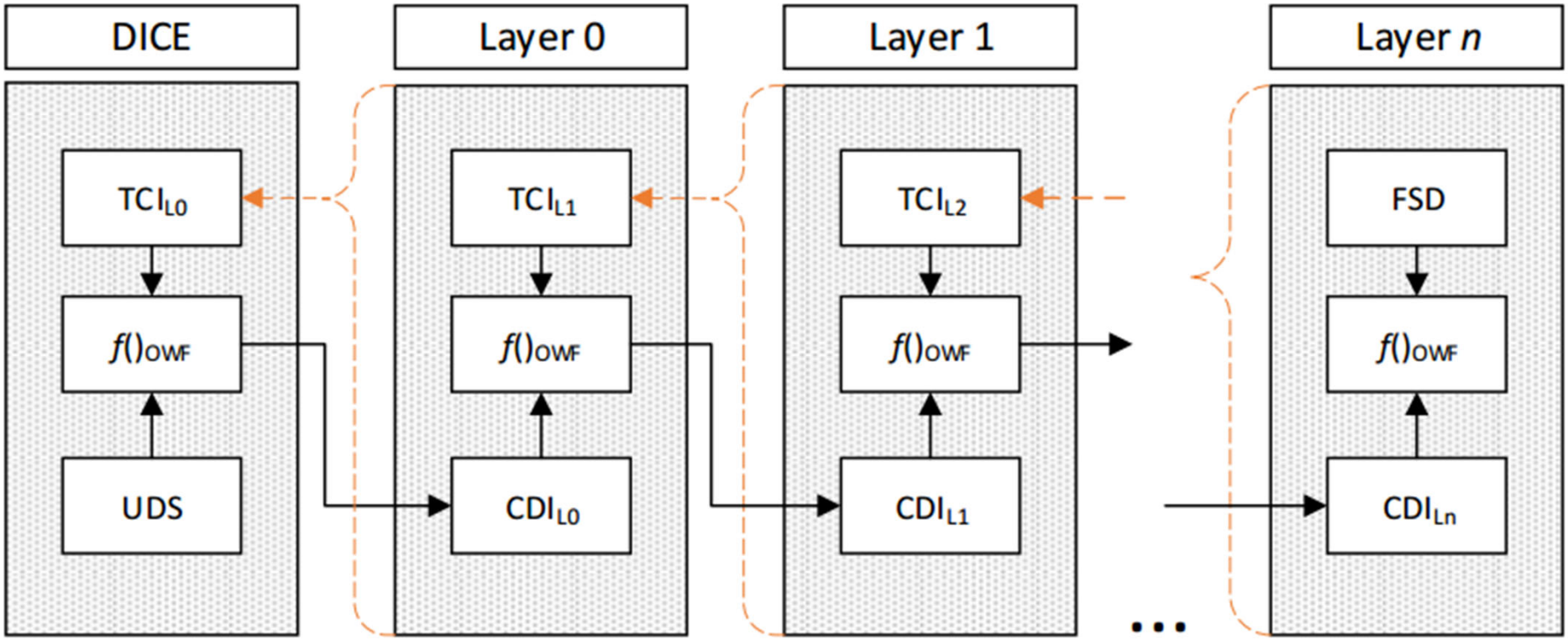
- **Endorsements and Reference Values**
  - CoRIM (Concise Reference Integrity Manifest)
    - CoMID (Concise Module ID) for hardware/firmware modules
    - CoSWID (Concise Software ID) software components
    - CoBOM (Concise Bill of Material) = active CoMID/CoSWID
    - CoTS (Concise Trust Anchor Stores)
- **Appraisal Policy for Evidence**
  - OpenPolicyAgent
- **Attestation Results**
  - EAR (EAT Attestation Results)
  - AR4SI (Attestation Results for Secure Interactions)

Device Identifier Composition Engine (DICE)

- **DICE provides a secure identity to a device and to all its software components constituting its TCB**
- **Compound Device Identifier (CDI)**
  - secret value resulting from the application of a cryptographic one-way function to a combination of a DICE Layer's secret value and the measurement of the next DICE Layer
- **TCB Component Identifier (TCI)**
  - measurement of a system layer
- **Unique Device Secret (UDS): secret value of a specific platform used to compute the first CDI value**
  - statistically unique: randomly generated with low possibilities to have the same value in another device
  - not correlated: impossible to determine UDS of other devices

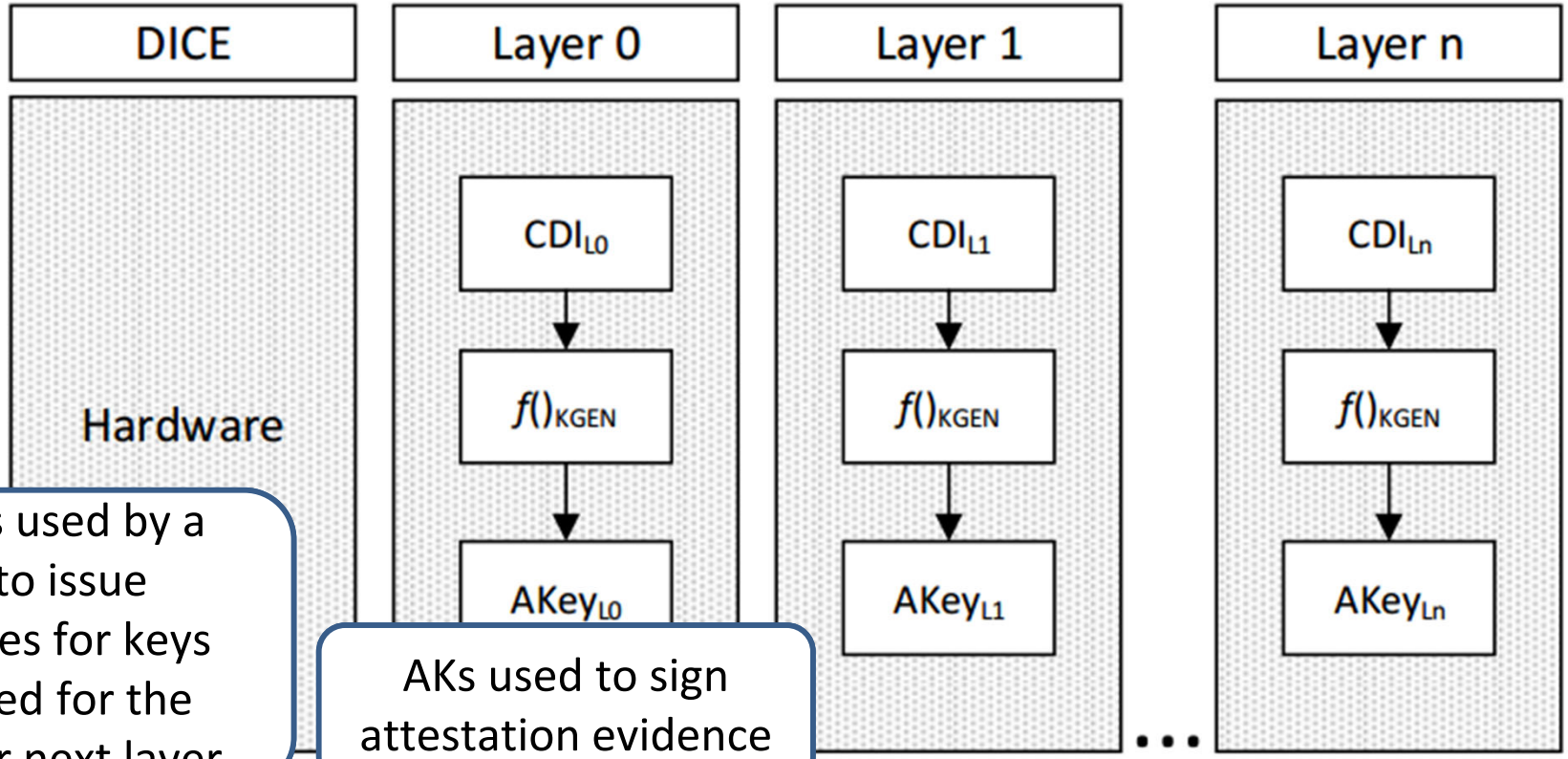
DICE layered architecture

- each layered TCB component combines its CDI secret with the TCI of the next layer to generate the next layer CDI



"DICE Layering Architecture", https://trustedcomputinggroup.org/resource/dice-layering-architecture/

DICE keys and certificates



ECA keys used by a layer to issue certificates for keys generated for the current or next layer

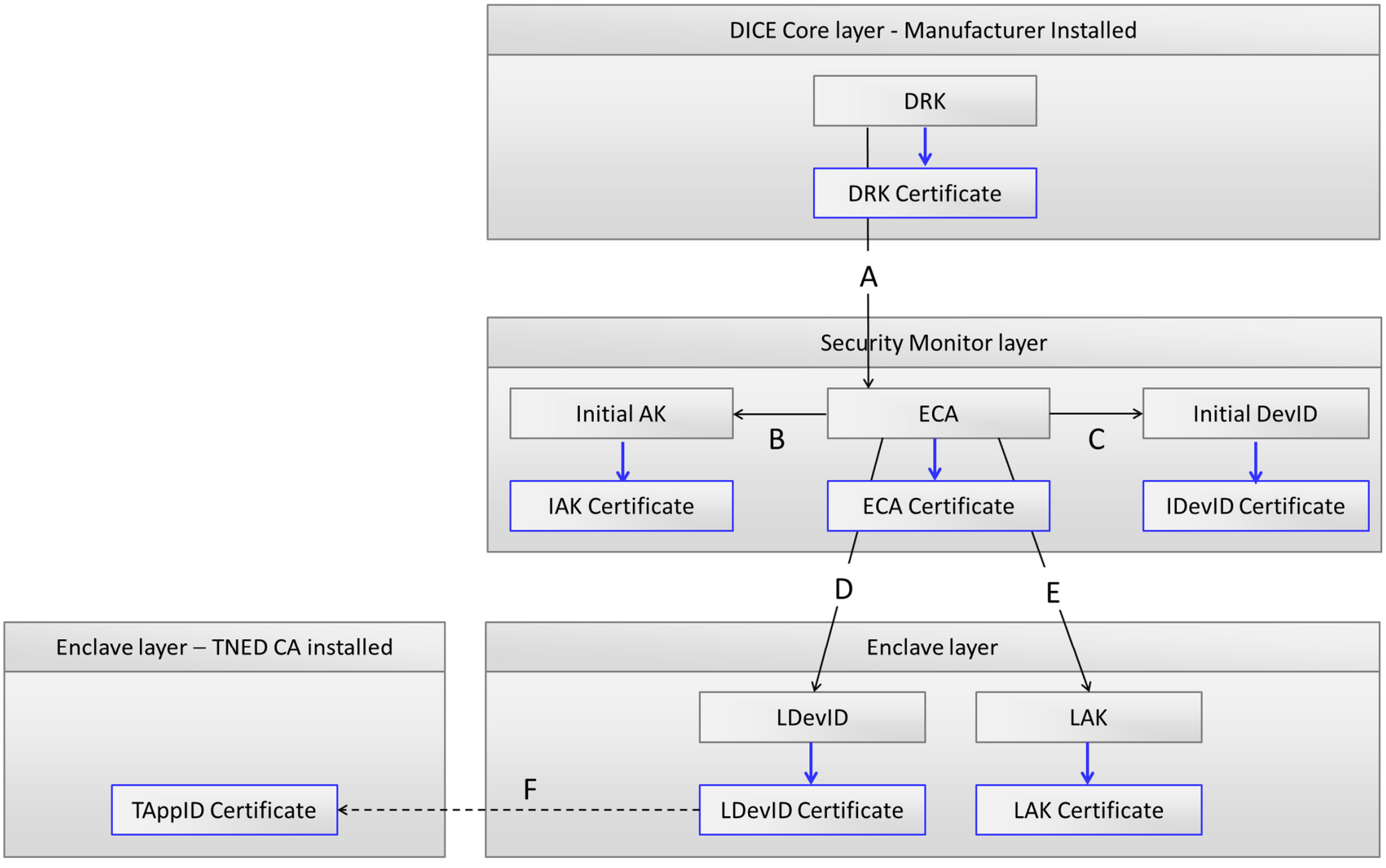
AKs used to sign attestation evidence

	CERTIFICATION	ATTESTATION	IDENTITY	NOTES
ECA Key	✓	✓	✓	Only sign known data
Attestation Key		✓	✓	Only sign known data
Identity Key			✓	May sign opaque challenge

Identity keys used to sign authentication challenges

DICE layered certification

- **each layer can act as an Embedded CA (ECA) to create a hierarchy based on the manufacturer (Root CA)**
- **each ECA can produce two different types of certificate**
  - Device Identity cert: used to embed a cryptographic identity
  - Attestation cert: used to authenticate evidence
- **each certificate must contain the extension DiceTcbInfo**
  - its OID is 2.23.133.5.4.1 = joint-iso-itu(2).international-org(23).tcg(133).tcg-platformClass(5).dice(4).TcbInfo(1)
  - it's a SEQUENCE of information about the target level
    - names (e.g. vendor. model, layer)
    - measurements (e.g. version, svn, fwidlist={hashAlg+digest})



Open profile for DICE

- **Google specification for implementing DICE**
- **each layer has two CDIs**
  - Attestation CDI (mandatory)
  - Sealing CDI (optional)
- **the CDI of the next level is computed using also a set of specific input values (depending on the type of the CDI)**
  - configuration data (information on the integrity of the system)
  - authority data (hash of information about verified boot trusted authority)
  - mode decision (operating mode of the device)
  - hidden inputs (values not included in any certificate)
- **each layer has an Attestation keypair derived from the Attestation CDI**

DICE and RIOT

- **RIOT (Robust Internet Of Things) is Microsoft specification for implementing DICE**
- **there is only one CDI for the entire device**
  - it is computed combining together the UDS of the platform with the measure of the RIOT core (the only part of the sw that can access the CDI)
- **each layer has an Alias keypair used for attestation**
  - the layer N computes the Alias keypair for the layer N+1 starting from the measure of the layer N+1 (for the RIOT core it is derived from the CDI)

Google Cloud

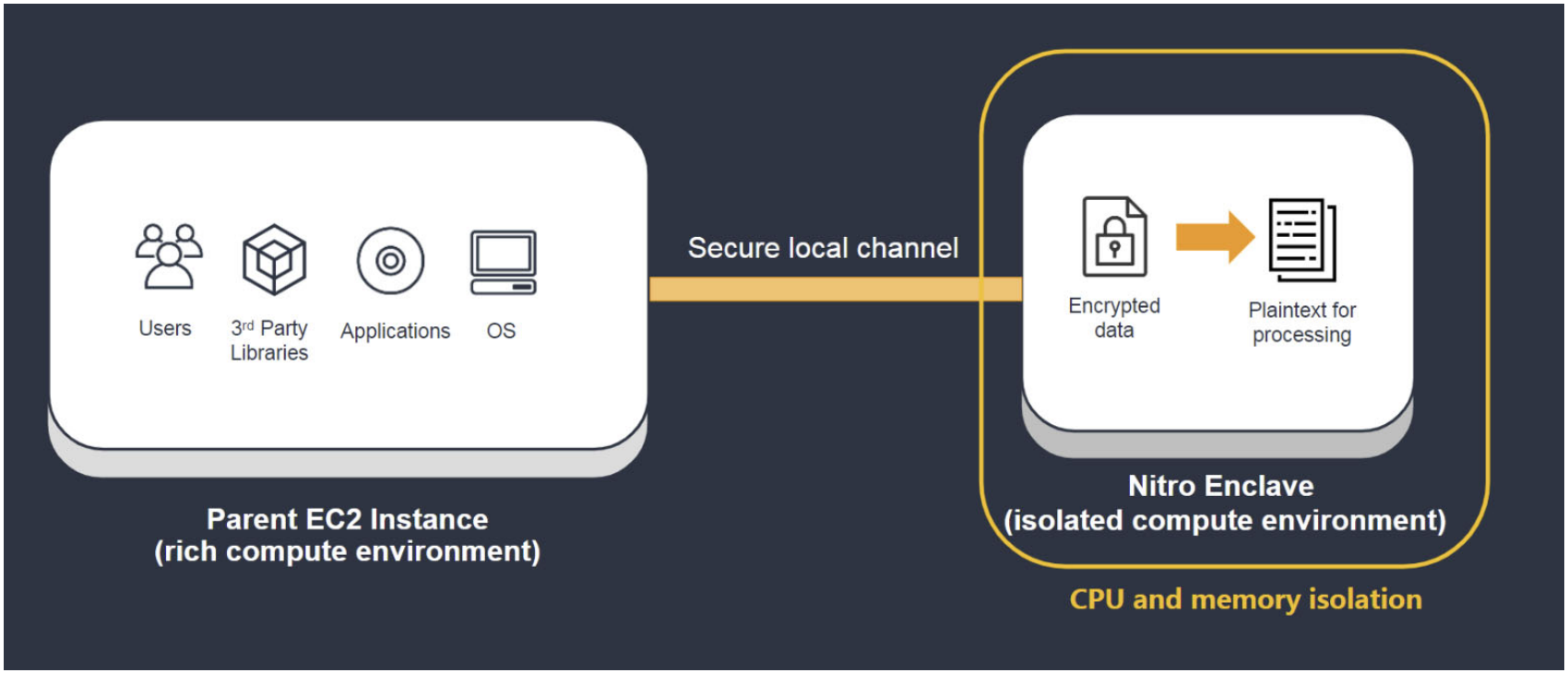
- **optional attestation on platforms via policies**
  - Google Kubernetes Engine (GKE)
  - Cloud Run
- **binary authorization on container image**
  - different level of scans and analysis
  - vulnerability scan, regression test
  - signature of the container image hash
- **container deploy-time security control**
  - blocks container deployment if no valid signature
  - allows deployment of container if matching policies

Amazon Web Services (AWS)

- **Elastic Container Registry (ECR)**
  - store containers' images
- **Amazon Inspector scan of container images in ECR**
  - new pushed images
  - new vulnerability inserted
- **Enhanced scanning**
  - OS vulnerability
  - programming language package vulnerability
  - service vulnerability

AWS Nitro

- Nitro enclave = isolated and constrained execution environment (VM) that can talk only to parent
- enclave can request attestation (to the Nitro manager) for proving some properties to request services (e.g. access cryptographic keys via KMS)



AWS Nitro's PCRs

- **PCR0 (Enclave image file)**
  - measurement of the image file, without the section data
- **PCR1 (Linux kernel and bootstrap)**
  - measurement of the kernel and boot ramfs data
- **PCR2 (Application)**
  - measurement of the user applications, without the boot ramfs
- **PCR3 (IAM role assigned to the parent)**
  - attestation succeeds only when the parent has the correct role
- **PCR4 (Instance ID of the parent)**
  - attestation succeeds only when the parent has a specific ID
- **PCR8 (Enclave image file signing certificate)**
  - attestation succeeds only when the enclave was booted from an enclave image file signed by a specific certificate

Azure Confidential Containers

- **on container deployment, a token is generated**
  - signed by the cloud node
  - verifiable by a remote entity
- **token contains information**
  - correct deployment of the container in a TEE
- **with a VM TEE**
  - hardware based and attested TEE
  - full guest attestation
  - additional data and code protection
- **no need for**
  - specialized programming model
  - special management