

# System Security: Trusted Computing I

TDDE62 Information Security: Privacy, System and Network Security

Thomas Nyman

Ericsson Product Security

# Table of contents

- Getting to know each other

Extra Reading

- Quick introduction to cryptographic trust

- Why trusted computing?

- Hardware security mechanisms

- Trusted execution environments

- Processor secure environments

- Arm TrustZone (case study: Android)

- Intel SGX

Extra Reading

- (Intel TDX / AMD SEV / Arm RME)

Extra Reading

- Concerns with TEEs & Takeaways

Extra Reading



# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (Intel TDX / AMD SEV / Arm RME)
- Concerns with TEEs & Takeaways

Extra Reading

Extra Reading

Extra Reading

Extra Reading



# Cryptography: one piece of system security

## Usage examples

### Integrity

- I put this data here and want to **see if it has changed**
- Software signing: I want to know **who made this module** before I run it
- Attestation: I booted today with this software that version

### Identities

- Thomas' passport is officially signed by Suomi Finland AB, they **vouch for it**
- Digital signatures: I am this large number and **this message provably came from me**
- Hardware: This serial number device came from this corporation

### Confidentiality

- I want to send my letters in a sealed envelope and lock my door
- TLS connections: I want to **transmit to my peer without eavesdroppers**
- Protected blobs: My private secret is kept in this digital box

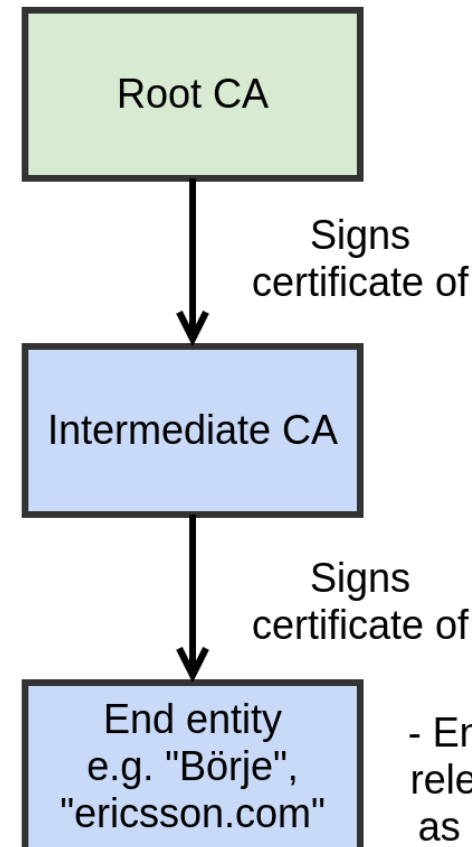
# Digital signatures carry human trust through time and space

- **Public Key Infrastructure (PKI)** = public and private keys chained together by signatures
- Built on a belief that a signature would be **hard to forge** without the private key
- The ultimate purpose of PKI is **transfer of trust**
  - Trusting party varies, e.g., vendor / user
  - In software signing, the user platform trusts software came from a software vendor
  - When a vendor control the full hardware platform, their **product protects themselves** against all outside influence (also from user)



# Trust chains of signed certificates

- Anyone can create a certificate (cf. text file) saying "I am Börje"
- Our trust comes from a **chain of signatures**
  - The root of the chain is the **trust anchor**
  - Signatures are validated a step at a time
- We call the top certificate authority "**Root CA**"
  - A certificate containing the CA's public key represents it in the cryptographic trust chain
- Intermediate CAs are used for manageability
  - Their certificates are also needed for trust chain validation
- "Ericsson Root CA signed this certificate for a Börje, I trust it, you(r private key) must be Börje"



- Trust anchor, root of trust
- Delivered "out of band"
- Installed in a safe integrity-protected place

- There may be many layers of intermediate CAs depending on distribution

- End entity needs to also provide its relevant intermediate CA certificates as evidence of its certificate validity

# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- **Why trusted computing?**
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (AMD SEV / Intel TDX / Arm CCA)
- Concerns with TEEs & Takeaways



# Current security challenges

Computing devices are increasingly **distributed**, **unsupervised**, and **physically exposed**

- Computers on the Internet (with untrusted owners)
- Embedded devices (cars, home appliances)
- Mobile devices (cell phones, PDAs, laptops)
- Base stations and wireless access points

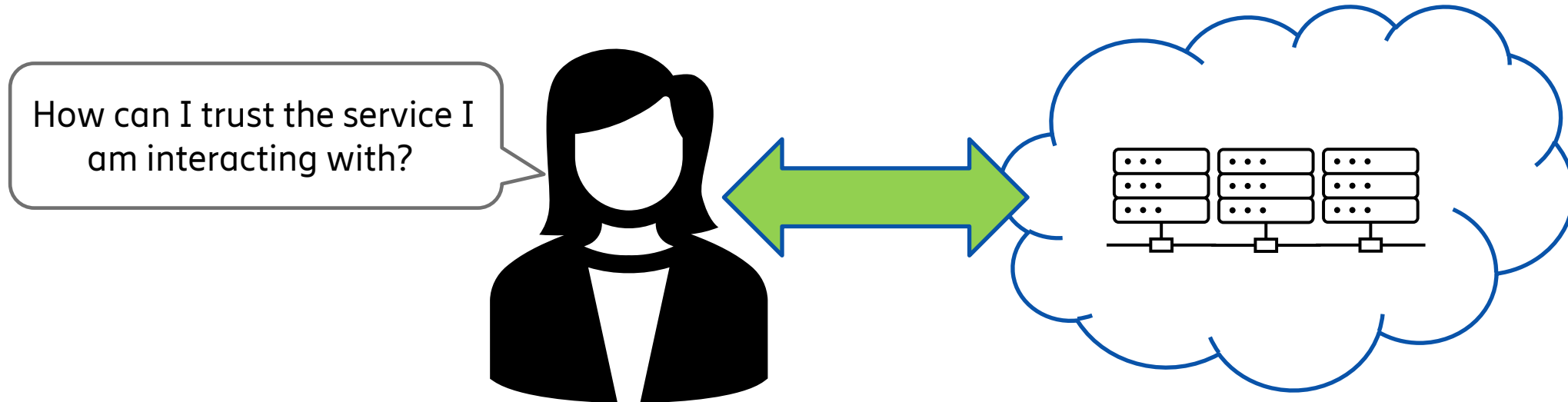
Cloud computing

- Virtualization, containers
- Web technologies - microservices

Attackers may **physically tamper with devices**

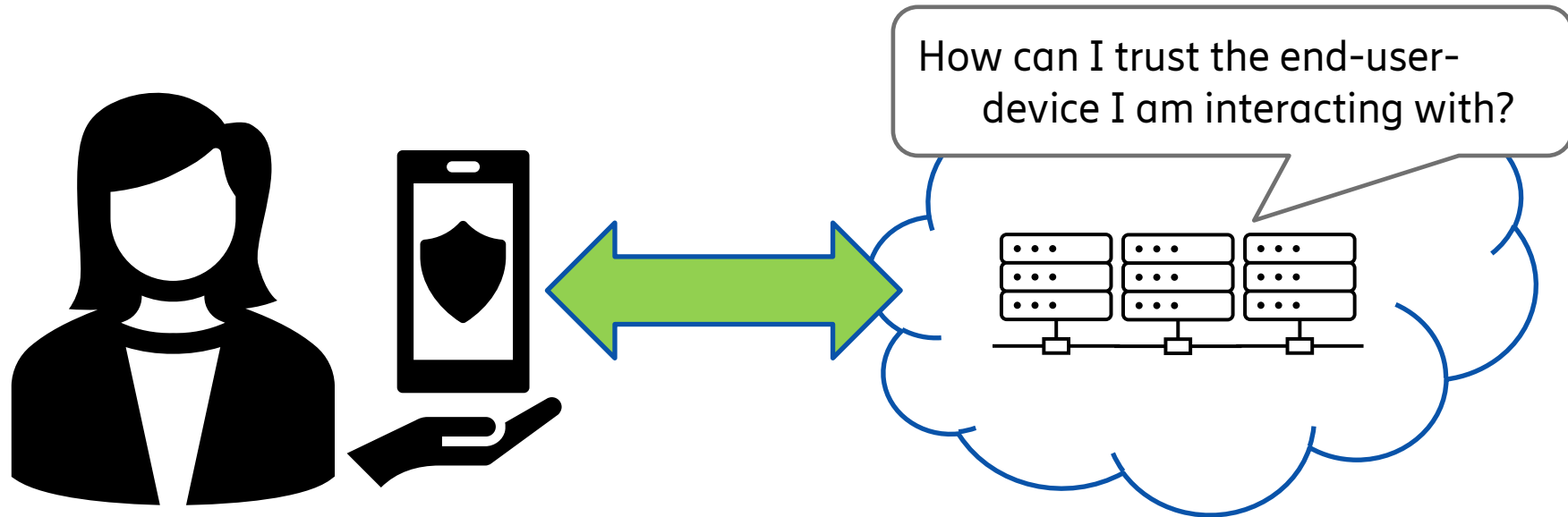
- Install malicious software
- Non-invasive measurement
- Invasive probing

# The main security question from a user's perspective



(we ignore here the questions related to the trustworthiness and semantics of data exchanged and processed)

# The main security question from the service's perspective?



(we ignore here the questions related to the trustworthiness and semantics of data exchanged and processed)

# Typical problems we want to address

1. How can we, inside a device (computer), protect sensitive data? **[secure storage]**
  - For example: cryptographic keys (most common), device identifiers, biometric information, etc.
2. How can we securely provision (set up) cryptographic keys in a remote device? **[key provisioning]**
  - Such that we can later use the key to bootstrap a secure communication, e.g., TLS connection.
3. How can we confidentially compute on sensitive information on a remote system? **[protection of data-in-use]**
  - For example: patient medical information, such that personally identifiable information (PII) remains protected from malicious actors, vendors and other partners, and even the compute provider?

# Trusted computing

Trusted computing is a notion of computing which can provide answers our first two problems

- A trusted computing system will consistently behave in expected ways
- The expected behaviors will be enforced by computer hardware and software

There are different approaches to trusted computing; there is no well-established precise definition

Other closely related concepts:

- Trusted Execution Environments (TEEs)
- Confidential Computing

# What is trust?

## Attention

The use of the word “**trusted**” here differs from the everyday use of the word (*deserving of trust, or able to be depended on*).

In the context of trusted computing, a *trusted entity* denotes an entity for which a *prediction regarding their behavior* has already been made.

# What is a TEE?

Ability to assess trustworthiness

Processor, memory, storage, peripherals

## Trusted Execution Environment

Isolated and integrity-protected

From the "normal" execution environment ("rich execution environment", REE)

# Assessing trustworthiness?

The system fulfils requirements defined by an assessment methodology, e.g., evaluation, compliance test

Questions:

1. Is the assessment methodology trustworthy? (and we get recursion)
2. Do we just have to trust the methodology?

Recall: Common Criteria (CC) is an ISO standard of a methodology to evaluate and certify products according an agreed target set of (security related) requirements (EAL levels)

- Used for smart cards, cryptographic libraries, cryptographic hardware, etc.
- CC certification is done by approved certification bodies, and a CC certificate holds in any country that accepts the CC scheme. In Sweden, [FMV/CSEC](#) is an accredited certification body

# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?

Extra Reading

- **Hardware security mechanisms**

- Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (Intel TDX / AMD SEV / Arm RME)

Extra Reading

Extra Reading

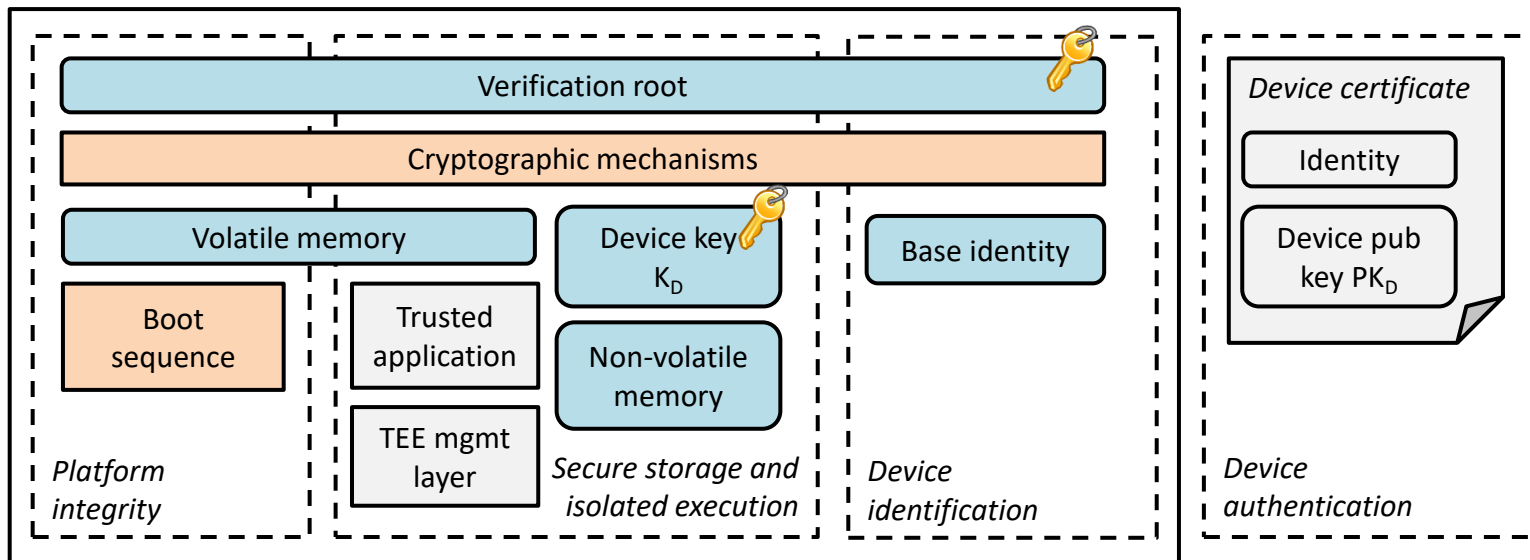
- Concerns with TEEs & Takeaways

Extra Reading



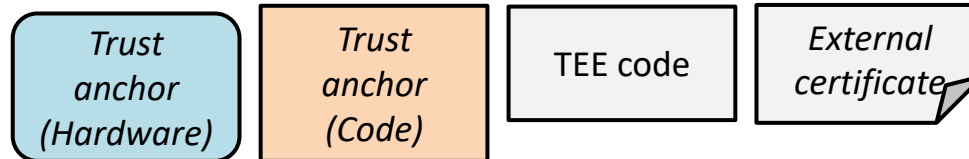
# Hardware security mechanisms

## Overview of common hardware security mechanisms



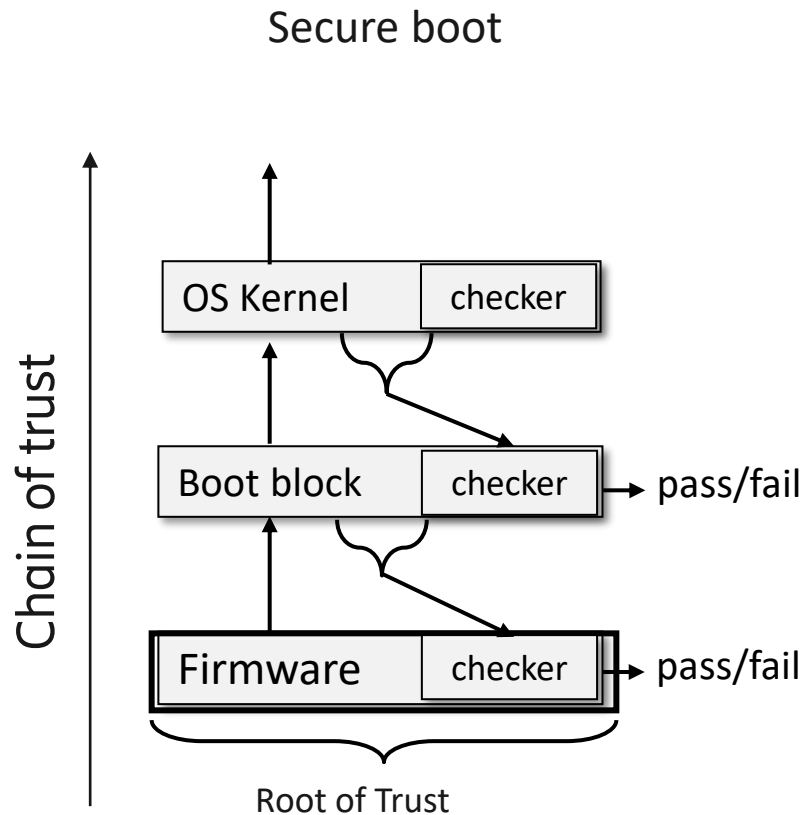
1. Platform integrity
2. Secure storage
3. Isolated execution
4. Device identification
5. Device authentication

### Legend



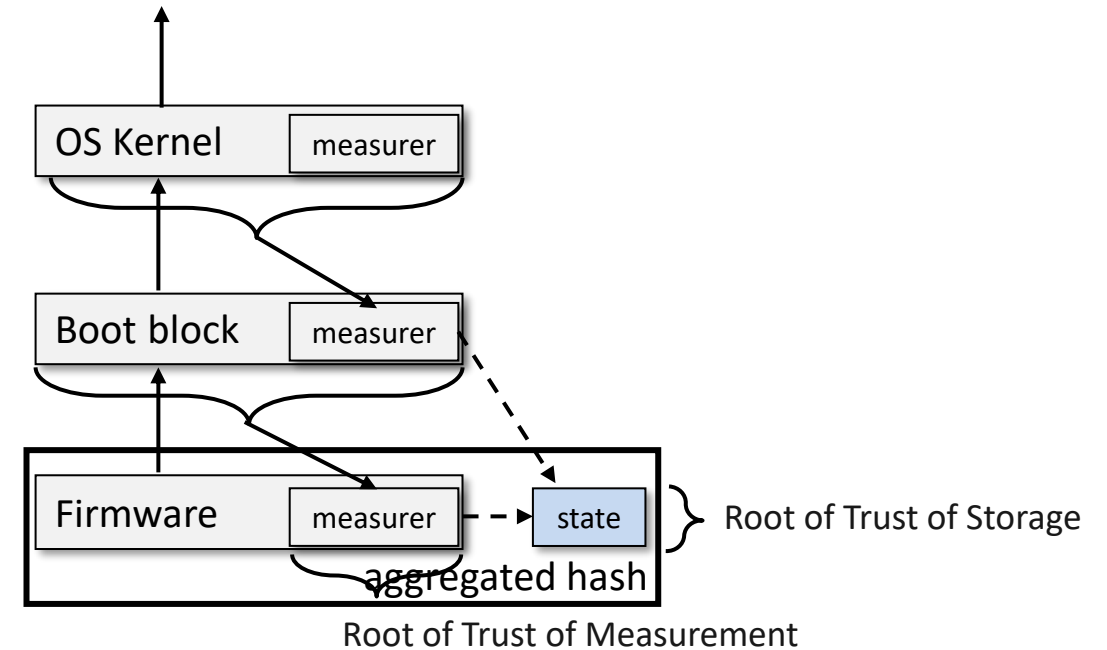
Figures adapted from: Jan-Erik Ekberg, Kari Kostianen, and N. Asokan "[Untapped Potential of Trusted Execution Environments on Mobile Devices](#)", 2014

# Secure vs. measured boot



How will you implement a checker?  
- hardcode  $H(\text{Boot block}|\text{checker})$  as reference value in checker (Firmware)?

## Measured boot (a.k.a. authenticated boot)

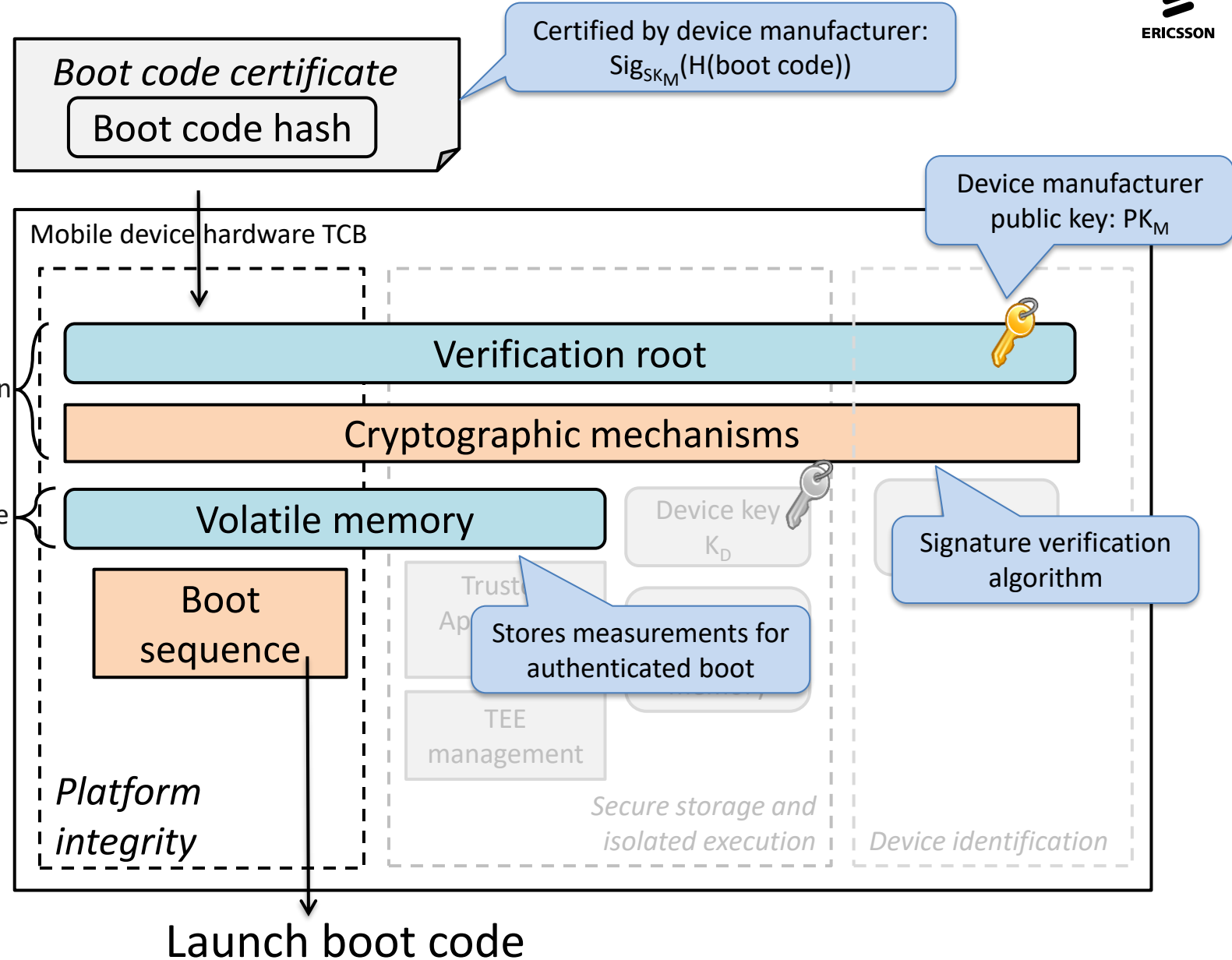
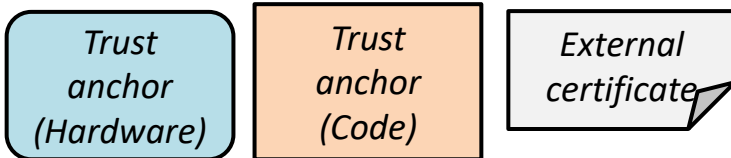


State can be:  
- bound to stored secrets (sealing)  
- reported to external verifier (remote attestation)

# Boot integrity

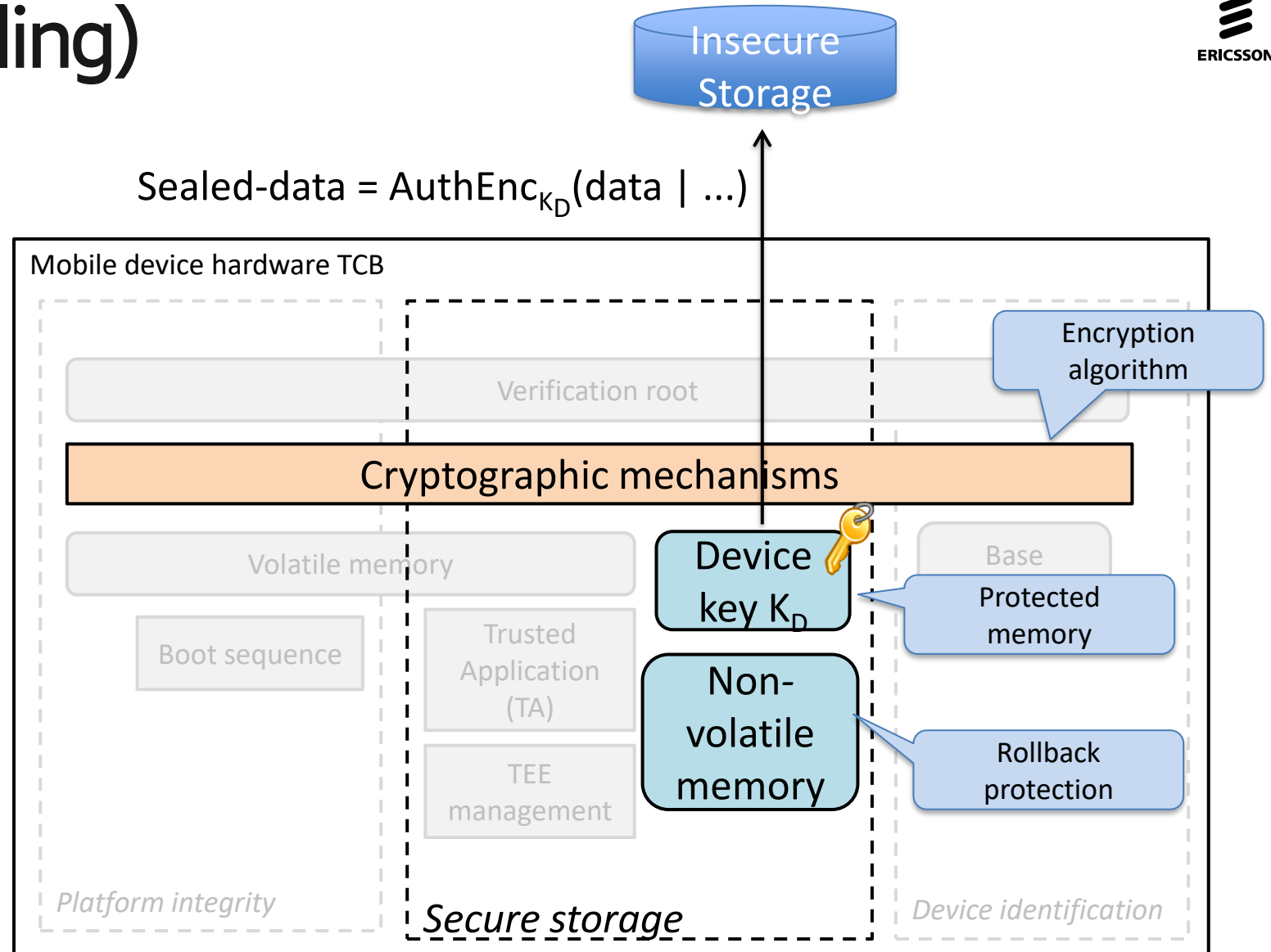
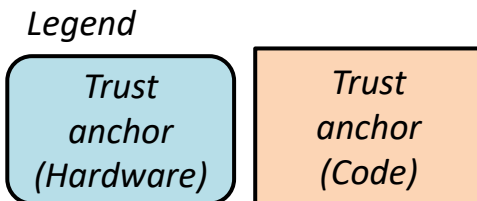
- In practice, secure boot is typically implemented via software signing
- The trust anchor, i.e., public key (PK) used for verification is fused (permanently and immutably stored) in hardware

Legend



# Secure storage (sealing)

- Bind sensitive data to a specific device (and system state) through encryption against a hardware-fused, device-specific secret key
- Sealed data can only be decrypted ("unsealed") on the device it was sealed on

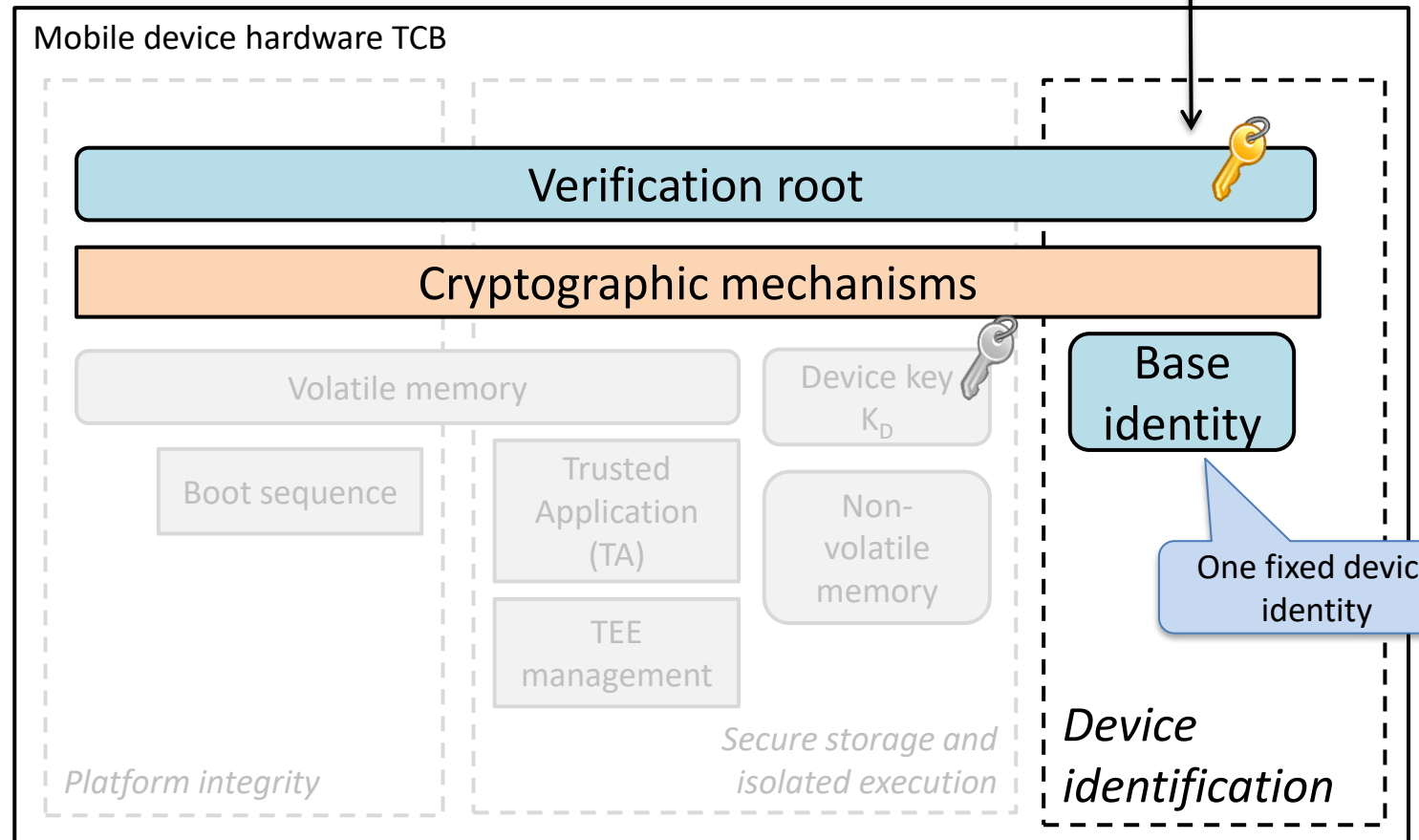
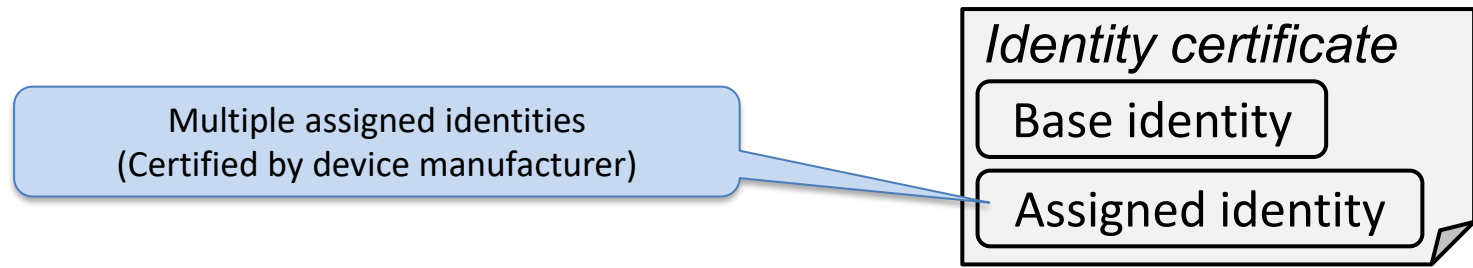
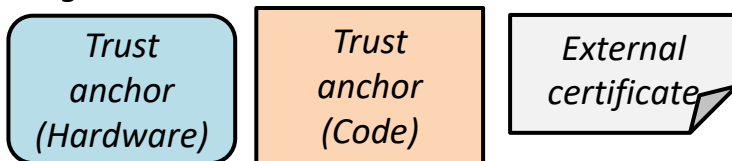




# Device identification

- Dynamic identifier assignment gives flexibility during and after manufacturing
- For example: delay assigning IMEI to ASIC until device manufacture, as opposed to doing it at chip manufacture

*Legend*



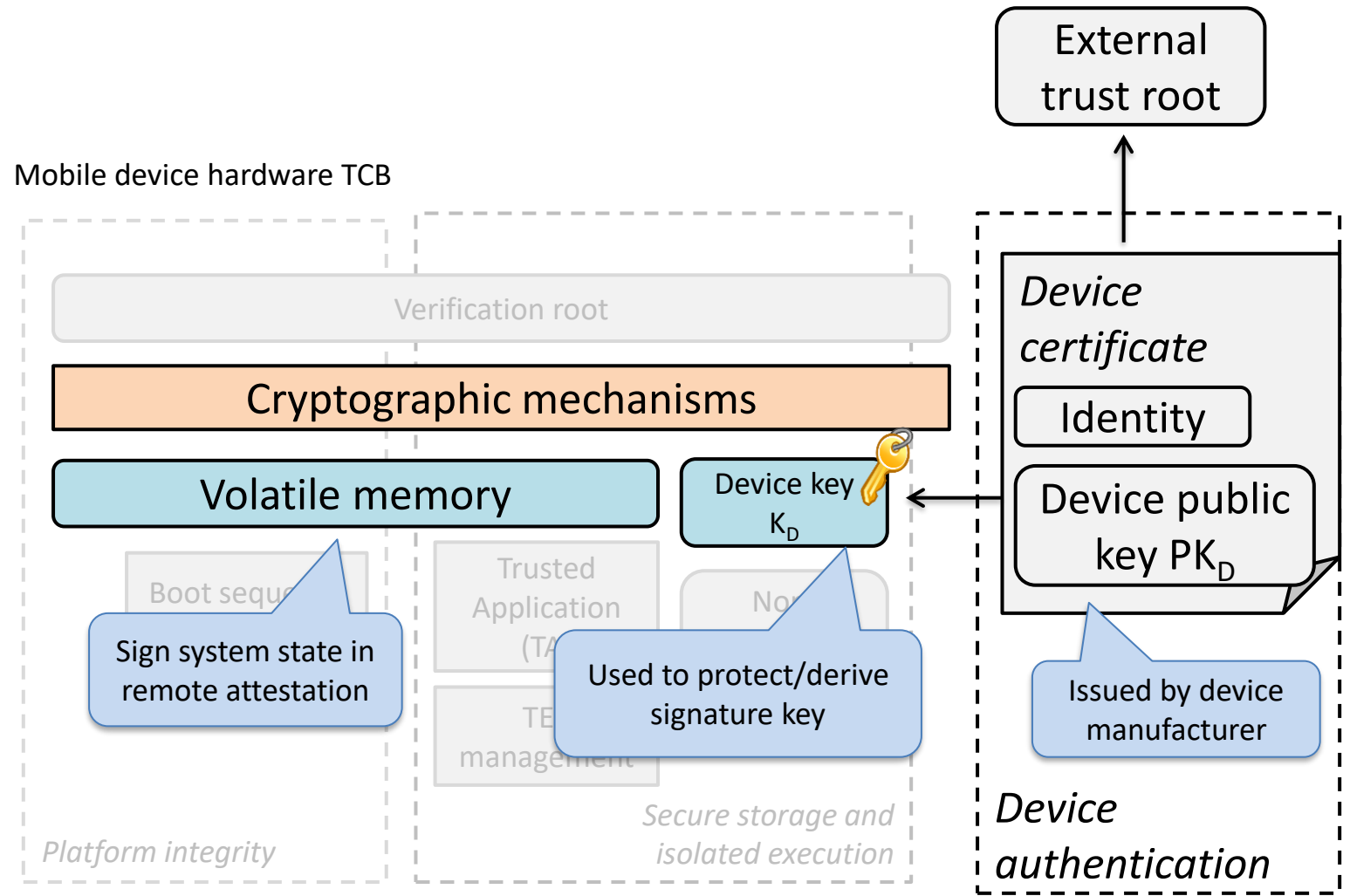
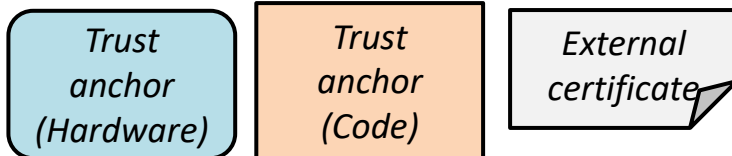
**ASIC** Application-Specific Integrated Circuit  
**IMEI** International Mobile Equipment Identifier

# Device authentication (remote attestation)

Security mechanisms described so far are all local to device

- Communication with external entities also needed (typically service providers)
- Verify identity or origin / manufacturer of a device

## Legend



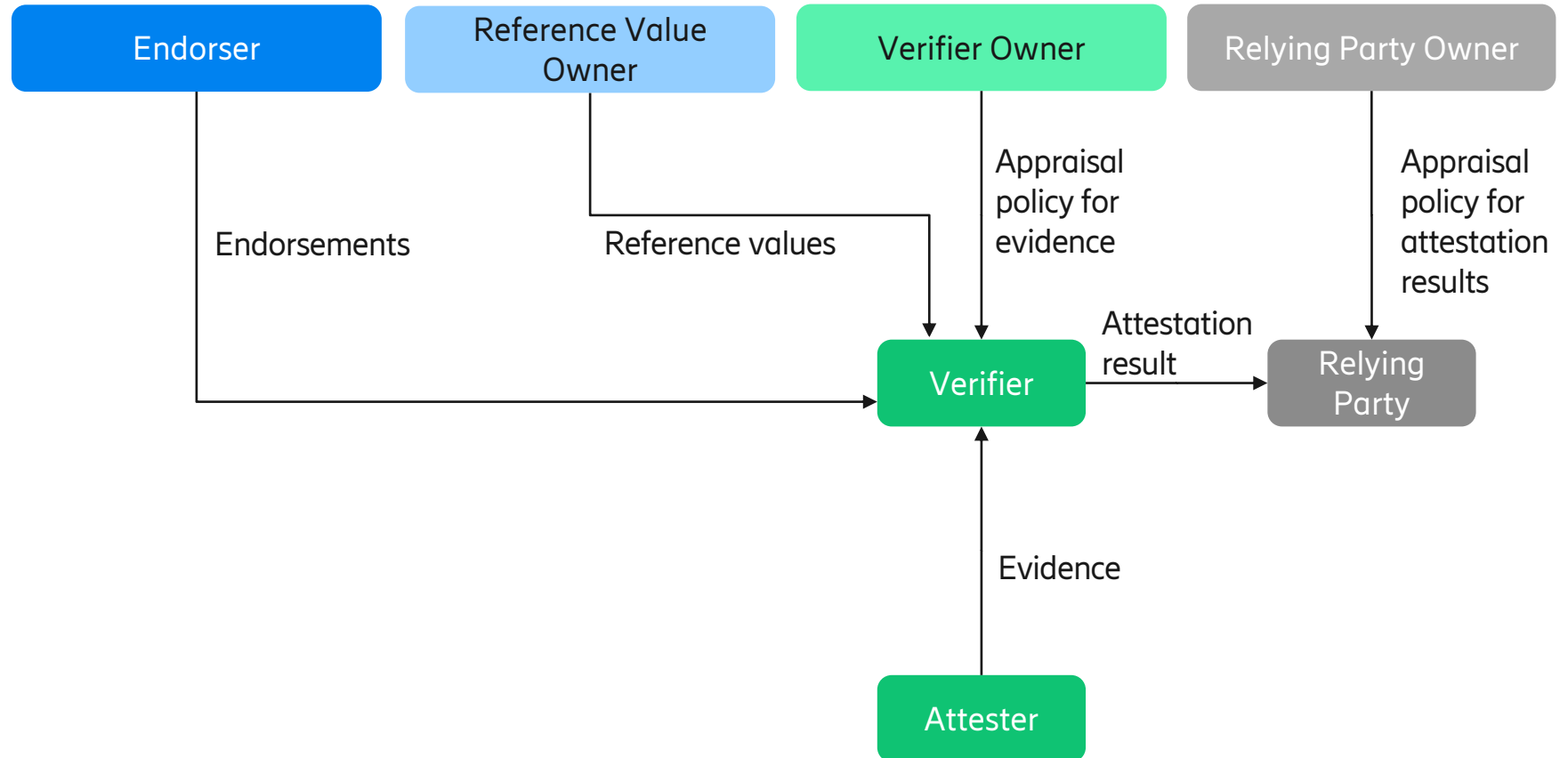
# Remote attestation procedures

## IETF [RATS](#)

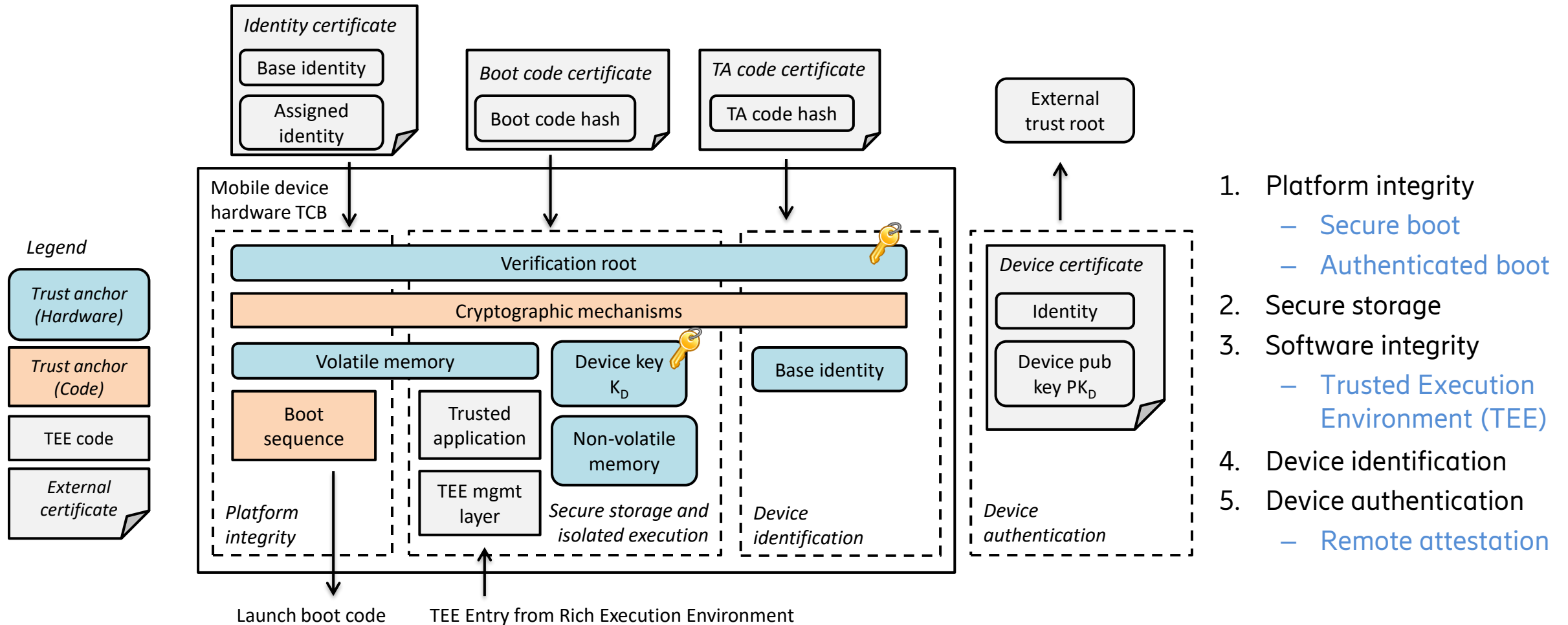
- *In network protocol exchanges, it is often the case that one entity (a Relying Party) requires evidence about the remote peer [...], in order to assess the trustworthiness of the peer.*

### In practice:

- Did remote system boot up in a known and trustworthy state?



# Hardware security mechanisms (recap)



# Table of contents

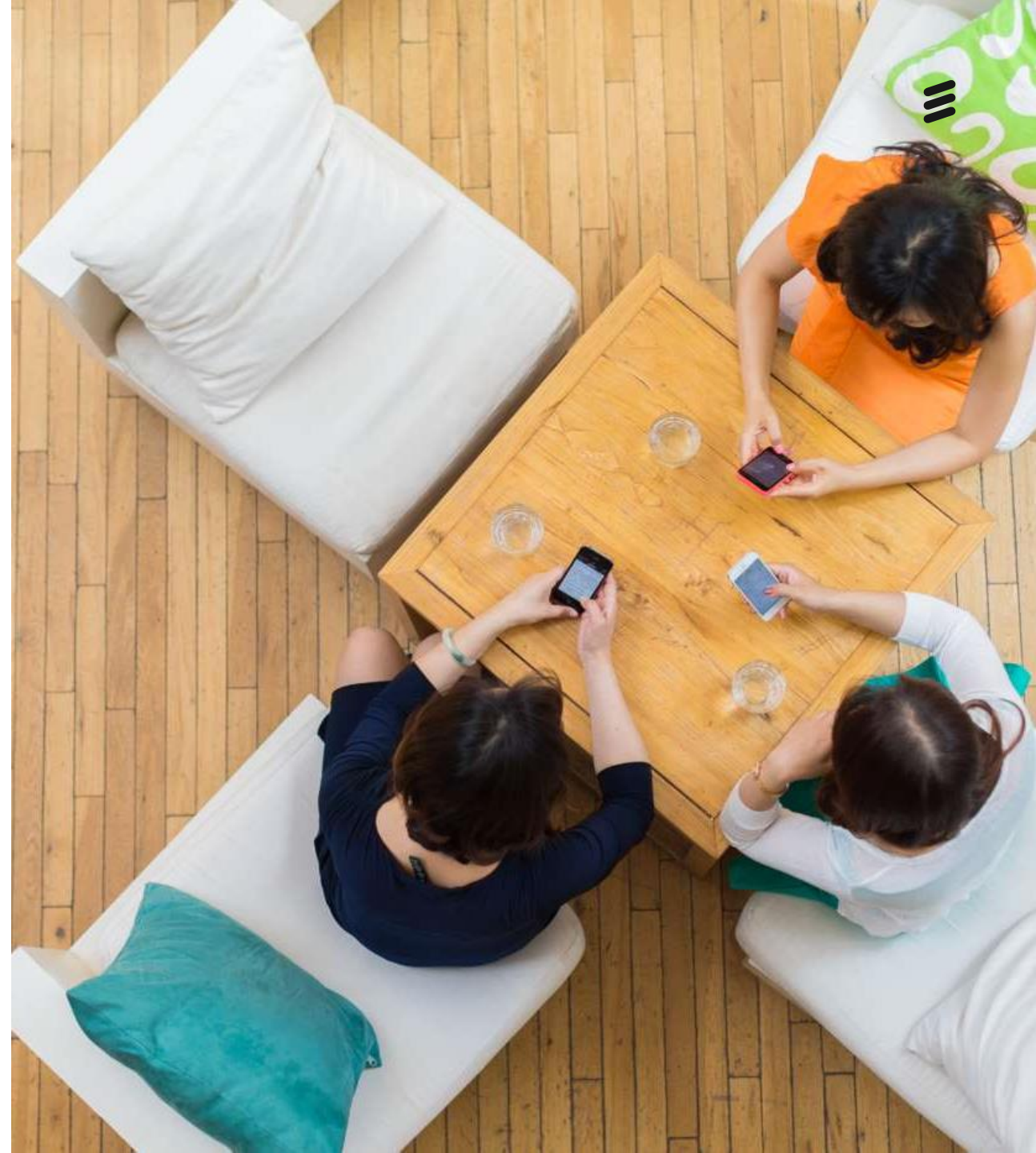
- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (Intel TDX / AMD SEV / Arm RME)
- Concerns with TEEs & Takeaways

Extra Reading

Extra Reading

Extra Reading

Extra Reading

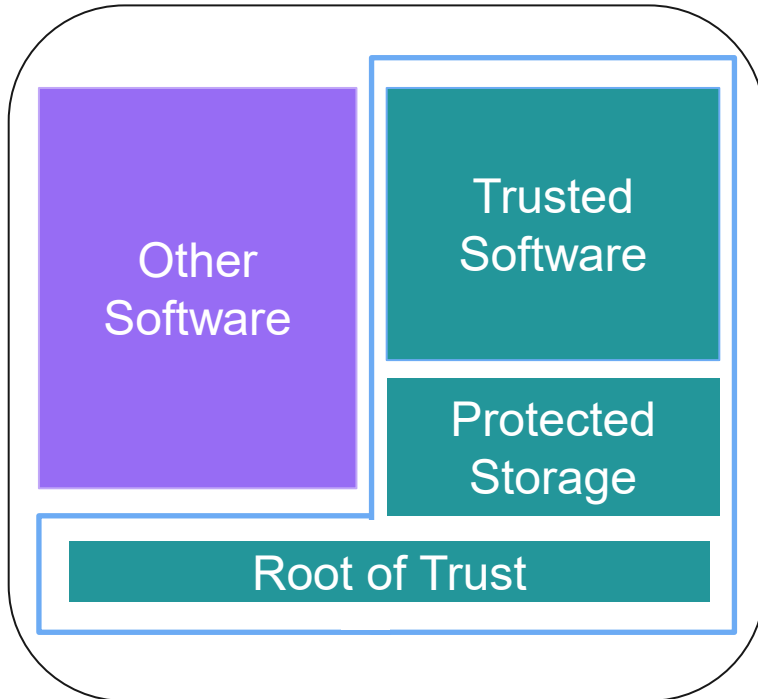


# Trusted Execution Environments (TEEs)

An environment in a computer system capable of running trusted software while providing the following :

1. **Isolation:** the ability to run trusted software strongly isolated from software belonging to other security domains so that untrusted software cannot influence or infer computation or outputs by trusted software
2. **Secure storage:** the ability for trusted code to store persistent data, even across reboots of the platform, with guarantees of the integrity and confidentiality of the data with respect to untrusted code belonging to other security domains
3. **Attestation:** the ability to convince a (typically a remote) party of the presence of the above attributes

# Hardware-assisted TEEs are pervasive



Hardware support for

- Isolated execution: [Isolated Execution Environment](#)
- Protected storage: [Discrete non-volatile memory](#) or based on [sealing](#)
- Ability to convince remote verifiers: [\(Remote\) Attestation](#)

Cryptocards



<https://www.ibm.com/security/cryptocards/>

Trusted Platform Modules



<https://www.infineon.com/tpm>

ARM TrustZone



<https://www.arm.com/products/security-on-arm/trustzone>

Intel Software Guard Extensions



<https://software.intel.com/en-us/sgx>

# Platform security for mobile devices

## Mobile network operators:

1. Subsidy locks → immutable ID
2. Copy protection → device authentication, app. separation
3. ...

## Regulators:

1. RF type approval → secure storage
2. Theft deterrence → immutable ID
3. ...



## End users:

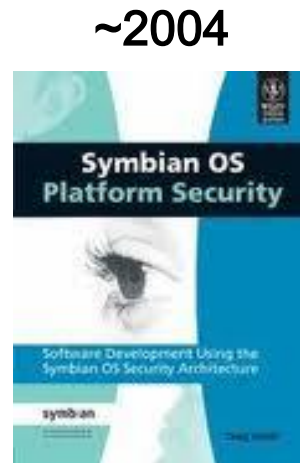
1. Reliability → app. separation
2. Theft deterrence → immutable ID
3. Privacy → app. separation
4. ...



Closed → Open

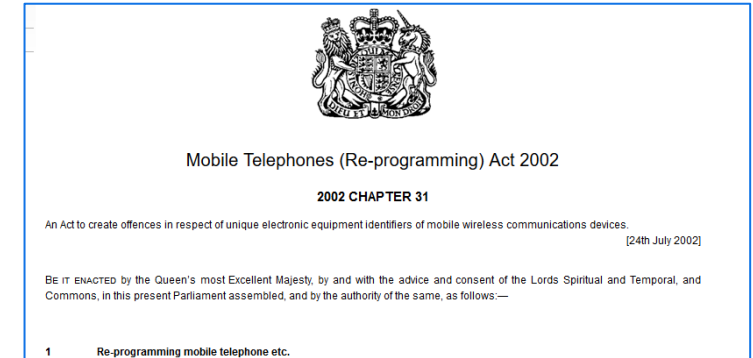
Different expectations than for PCs!

# Early adoption of software platform security



Mobile software platform security  
is now **widely deployed**

# Example: regulatory compliance



The IMEI shall not be changed after the ME's final production process. It shall resist tampering, i.e. manipulation and change, by any means (e.g. physical, electrical and software).

NOTE: This requirement is valid for new GSM Phase 2 and Release 96, 97, 98 and 99 MEs type approved after 1<sup>st</sup> June 2002.

**3GPP TS 42.009, 2001**



**Early TEEs for mobile phones**  
Nokia Radio Application Processor (RAP), ca. 2001

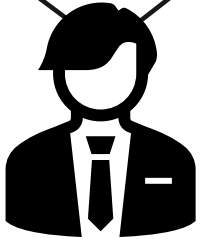


Saara Matala & Thomas Nyman, "[Historical insight into the development of Mobile TEEs](#)", Aalto SSG research group blog (2019)

# Mobile TEEs: Motivation

## Business requirements:

- mobile payments
- subsidy locks



## Regulatory requirements:

- tamper-resistant IMEIs
- secure storage for RF



# Mobile TEEs: Motivation

**THE WALL STREET JOURNAL.**  
U.S. Edition | September 16, 2019 | Print Edition | Video

Home World U.S. Politics Economy Business Tech Markets Opinion Life & Arts Real Estate

## Motorola Announces Plans To Offer Smart-Card Phone

*By Kimberley A. Strassel Staff Reporter of The Wall Street Journal*  
Updated April 23, 1998 12:01 am ET

LONDON -- Hoping to cash in on one of Europe's hottest technology markets, Motorola Corp. announced a new mobile telephone that features a slot for smart cards.

<https://www.wsj.com/articles/SB893268045342680500>

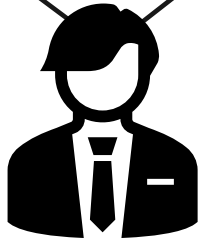


<https://www.mobilephoneduseum.com/phone-detail/startac-d-200>

# Mobile TEEs: Motivation

## Business requirements:

- mobile payments
- subsidy locks



## Regulatory requirements:

- tamper-resistant IMEIs
- secure storage for RF



## Engineering constraints:

Cost of discrete security chip too high on bill of materials!

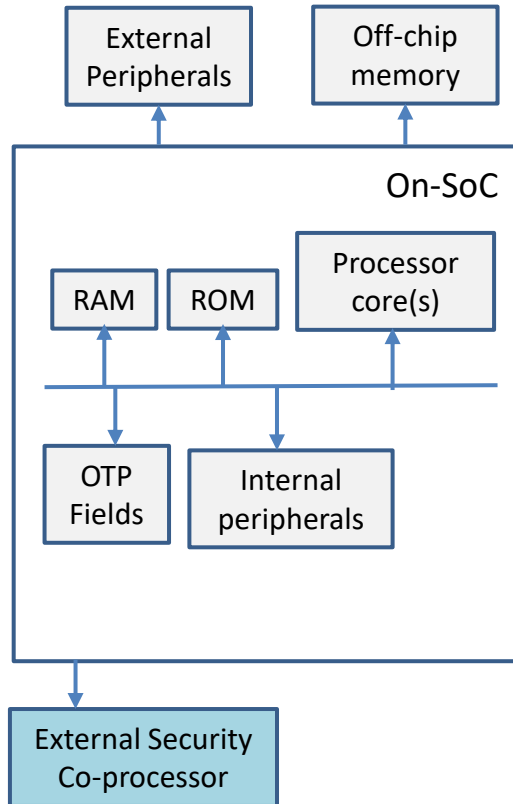


 New approach: "*processor secure environments*"

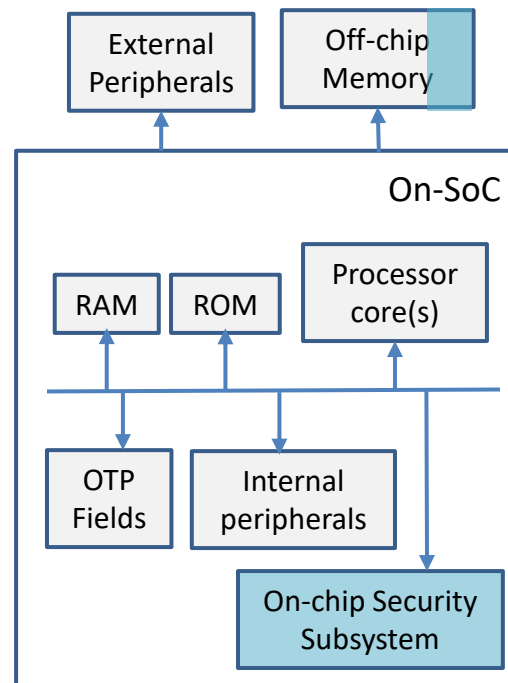
Generic low-cost enabler (rather than point solutions for particular use cases)

# TEE hardware realization alternatives

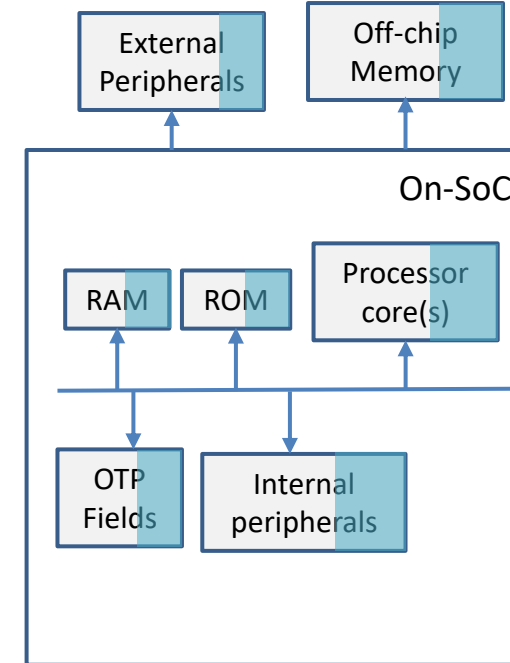
TEE component



External Secure Element  
(TPM, smart card)



Embedded Secure Element  
(secure enclave processor)



Processor Secure Environment  
(TrustZone, M-Shield, SGX)

# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- **Processor secure environments**
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (AMD SEV / Intel TDX / Arm CCA)
- Concerns with TEEs & Takeaways

Extra Reading

Extra Reading

Extra Reading

Extra Reading



# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments

Extra Reading

- Arm TrustZone (case study: Android)

- Intel SGX
- (AMD SEV / Intel TDX / Arm CCA)
- Concerns with TEEs & Takeaways

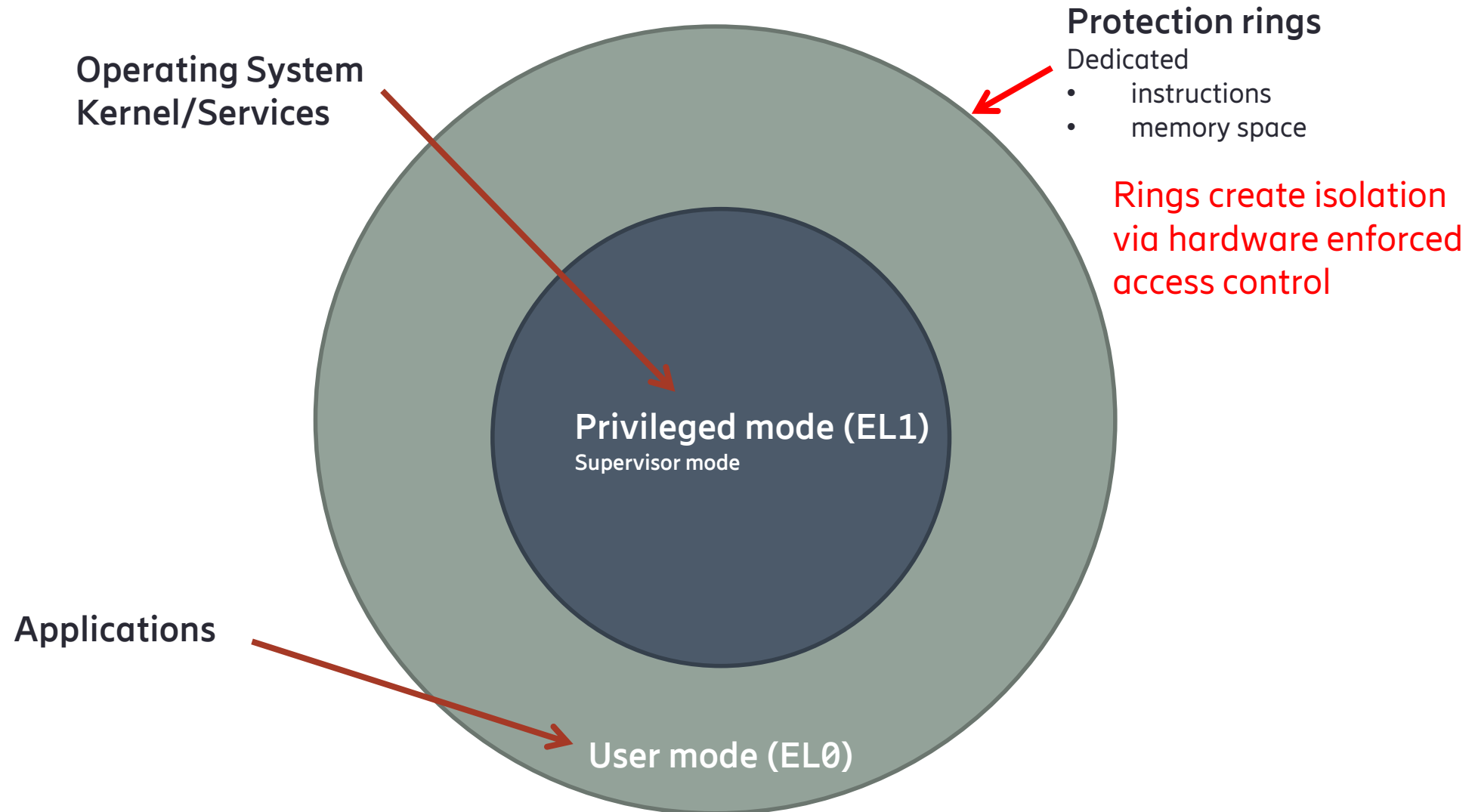
Extra Reading

Extra Reading

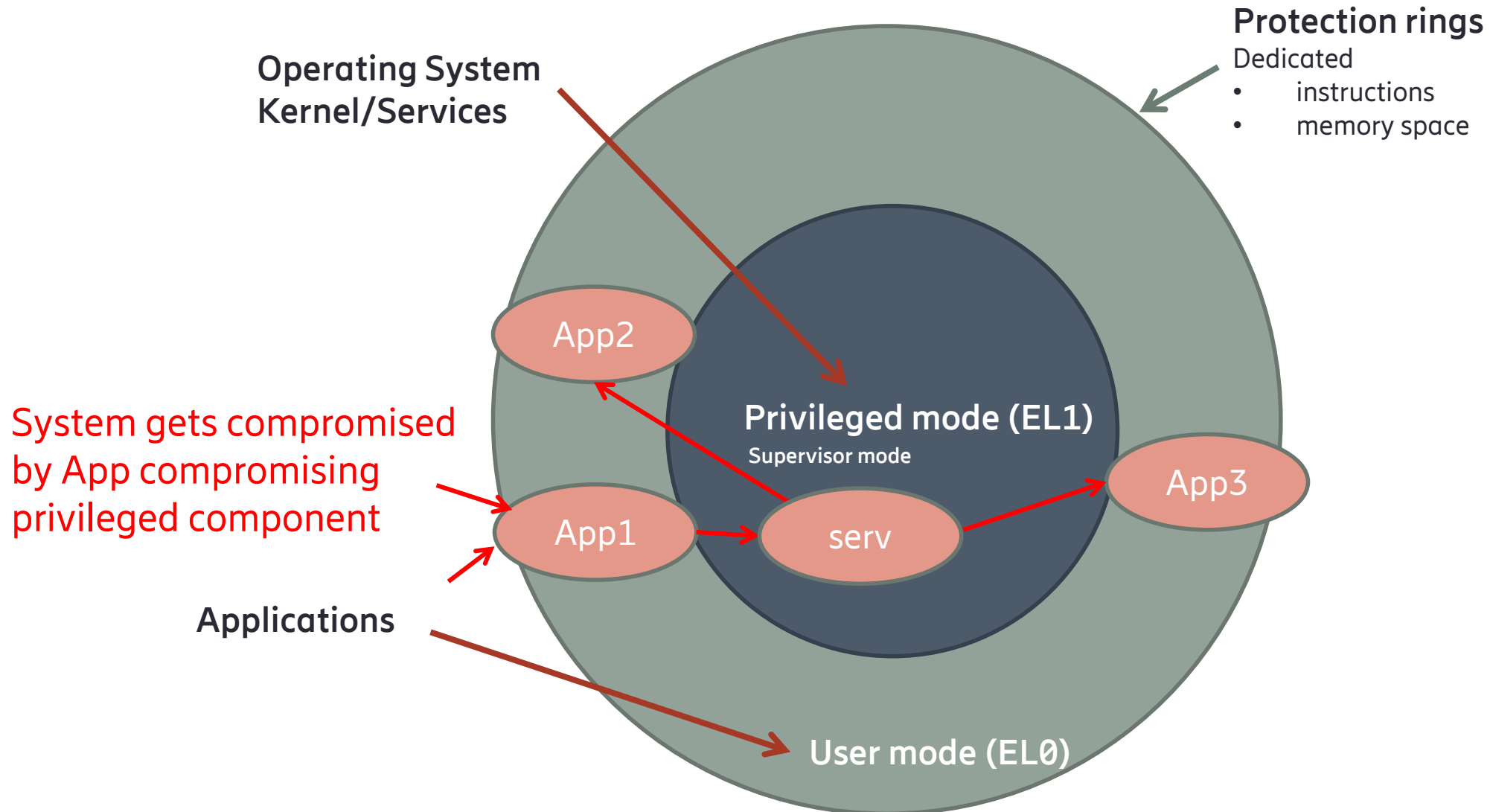
Extra Reading



# Recall: Protection rings in Arm architecture



# Kernel exposes large attack surface

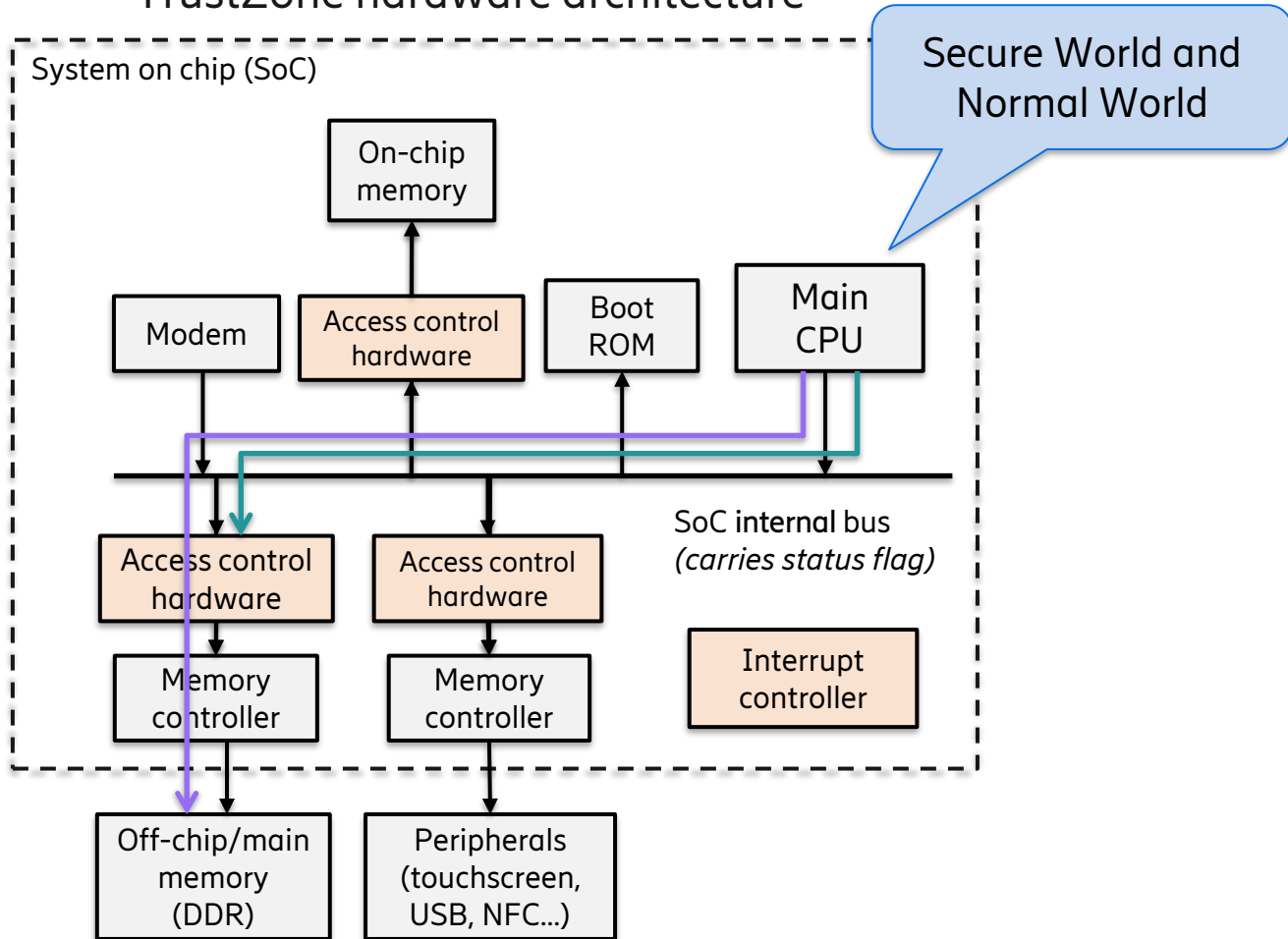


# High-Level Idea: Arm TrustZone

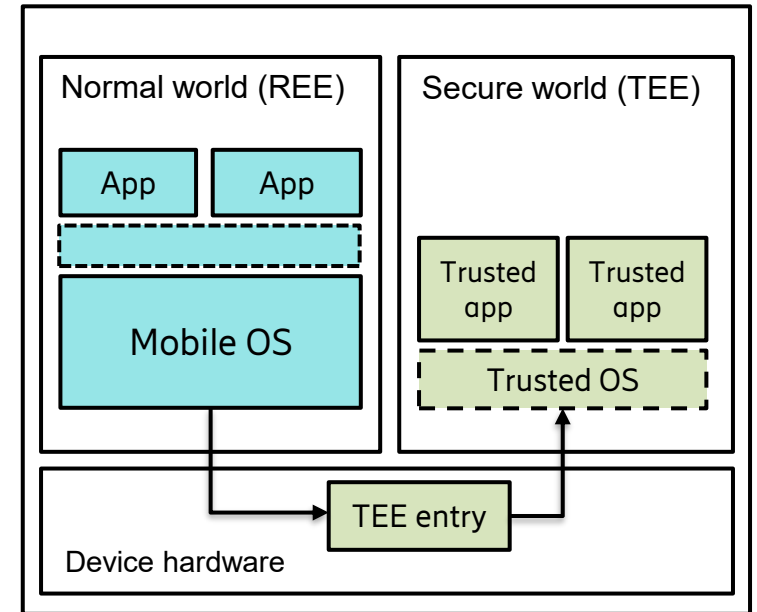
- **Introduce a Not-Secure-bit (NS-bit)**
  - Use this bit to *tag* architectural data as it moves throughout system: buses, cache, pages
- **Introduce new privilege level (EL3) and Secure Monitor which executes in EL3**
  - manages the NS-bit and transition in and out of secure world
  - small fixed API (making assurance of monitor code easier)
- **Hardware-enforced isolation**
  - Code in normal world cannot access resources that are tagged as belonging to the secure world
  - Code in secure world can access normal world but ring protection is still present
- **Secure interrupt**
  - Can force execution to proceed in secure world

# Arm TrustZone Architecture

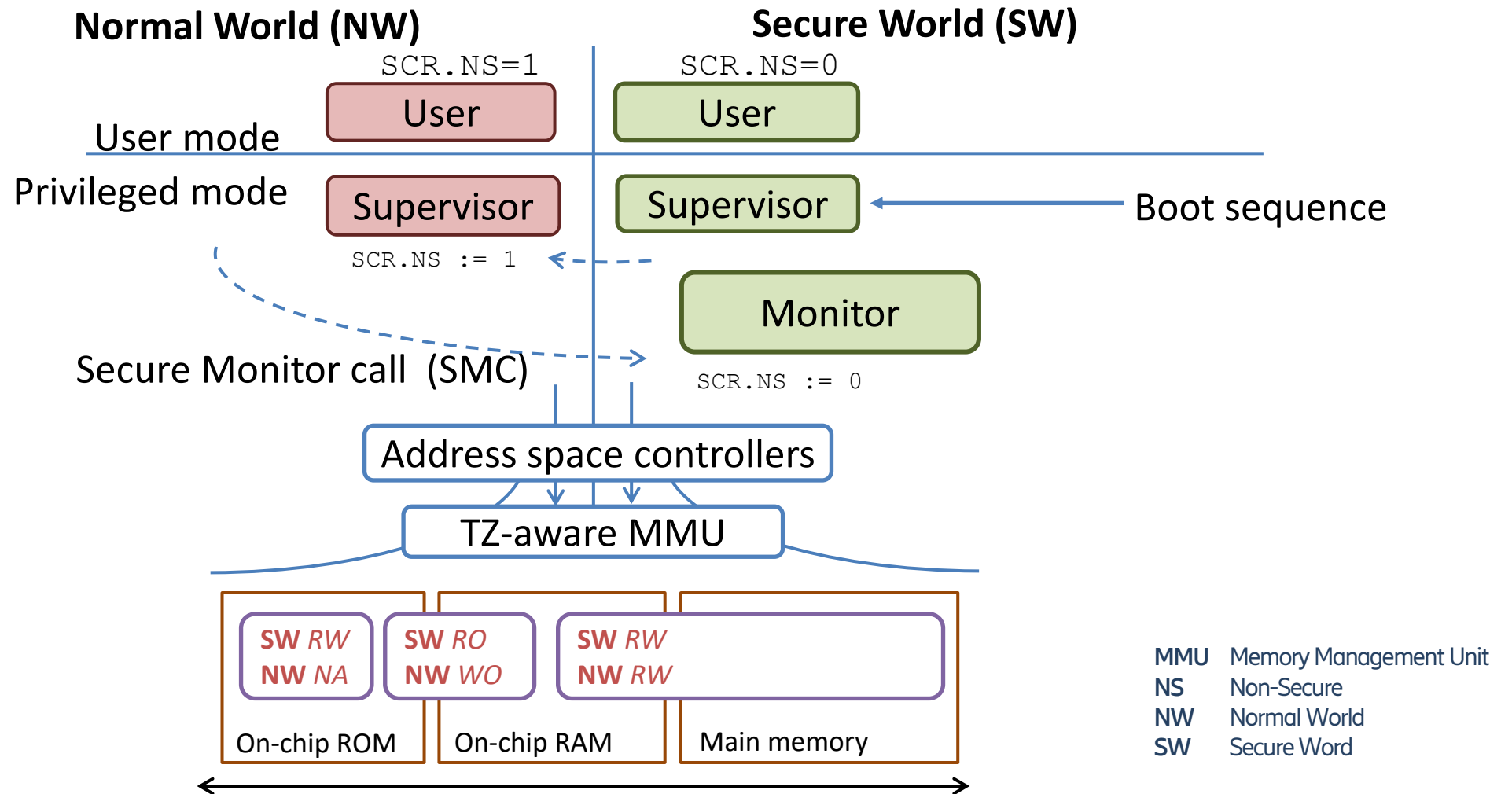
## TrustZone hardware architecture



## TrustZone system architecture



# TrustZone overview



# Shortcomings of TrustZone

- Since the TZ system is not an isolated part on the ASIC it is practically impossible to get high EAL levels in the Common Criteria framework or the US NIST security levels for HW , FIPS 140-3, Security Requirements For Cryptographic Modules
- Isolation of multiple apps in secure world and handling of multiple threads up to Trusted OS
- Secure boot of system and thus the setup of the TZ system is not part of the TZ solution and must be addressed by the chip maker that used TZ in his ASICs and the final device vendor (e.g. Samsung, Google)

# Case Study: Android

# Hardware-based security in Android

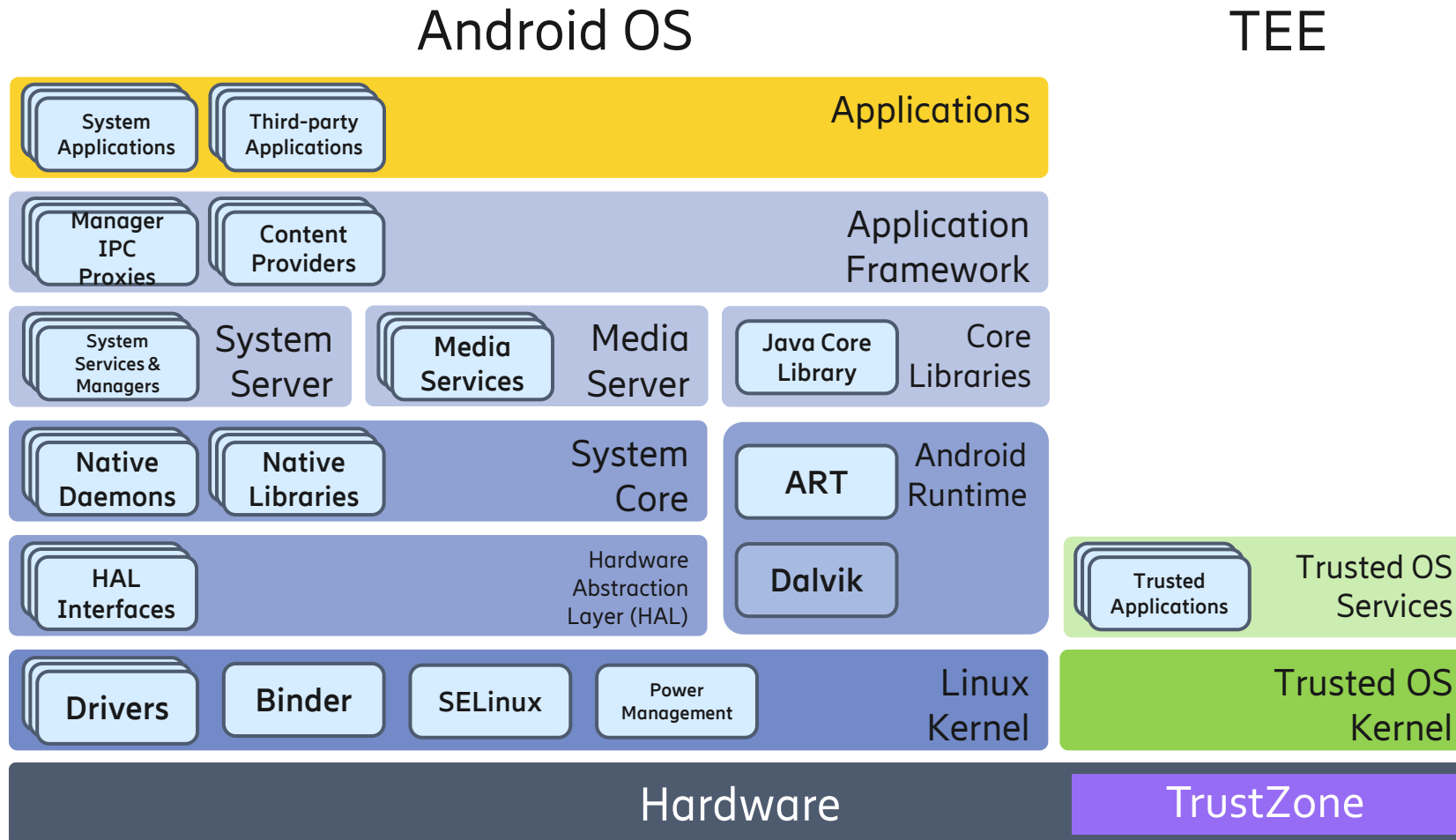
## Goals:

- Strong but easy to use user authentication
- Secure storage
- Platform integrity

## Implementation on Android:

- Fingerprint / Biometrics / Gatekeeper
- Keychain / Keystore
- Full-disk / File-Based encryption
- Verified boot

# Android Platform Architecture



TrustZone TEE operating in parallel with "rich execution environment" (Android)

# Android Keystore

Protects cryptographic keys from extraction even if application or OS is compromised

- [KeyChain API](#) for system credentials, e.g., Wi-Fi and VPN certificates (since 4.0)
- [KeyStore API](#) (Java CE) for application-bound keys

```
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(  
    KeyProperties.KEY_ALGORITHM_EC, "AndroidKeyStore");
```

**Hardware-backed keystore binds keys to device**

- Includes operating system and patch level of system image
- Keys not exposed unencrypted outside secure hardware
- `KeyInfo.isInsideSecureHardware()` (since Android 6.0) indicates if key is stored in hardware keystore

# Keymaster / KeyMint

## KeymasterHAL provides OS access to hardware-backed keystore

- Asymmetric key generation, signing and verification (since Android 4.1)
- Binder IKeyStoreInterface (since Android 4.3)
- Symmetric key support, access control, public key import and private / symmetric key import (since Android 6.0)
- Key attestation (since Android 7.0)
- ID attestation (since Android 8.0)
- Secure key import (since Android 9)
- Protected Confirmation (since Android 9)
- Keys that are only usable when the device is unlocked or during early boot stages (since Android 10)
- Optional support for hardware-wrapped storage keys and device-unique attestation in StrongBox (since Android 10)
- Keys with a limited number of uses and user-specified attestation keys (since Android 12 renamed to KeyMint HAL)

## Implementation alternatives:

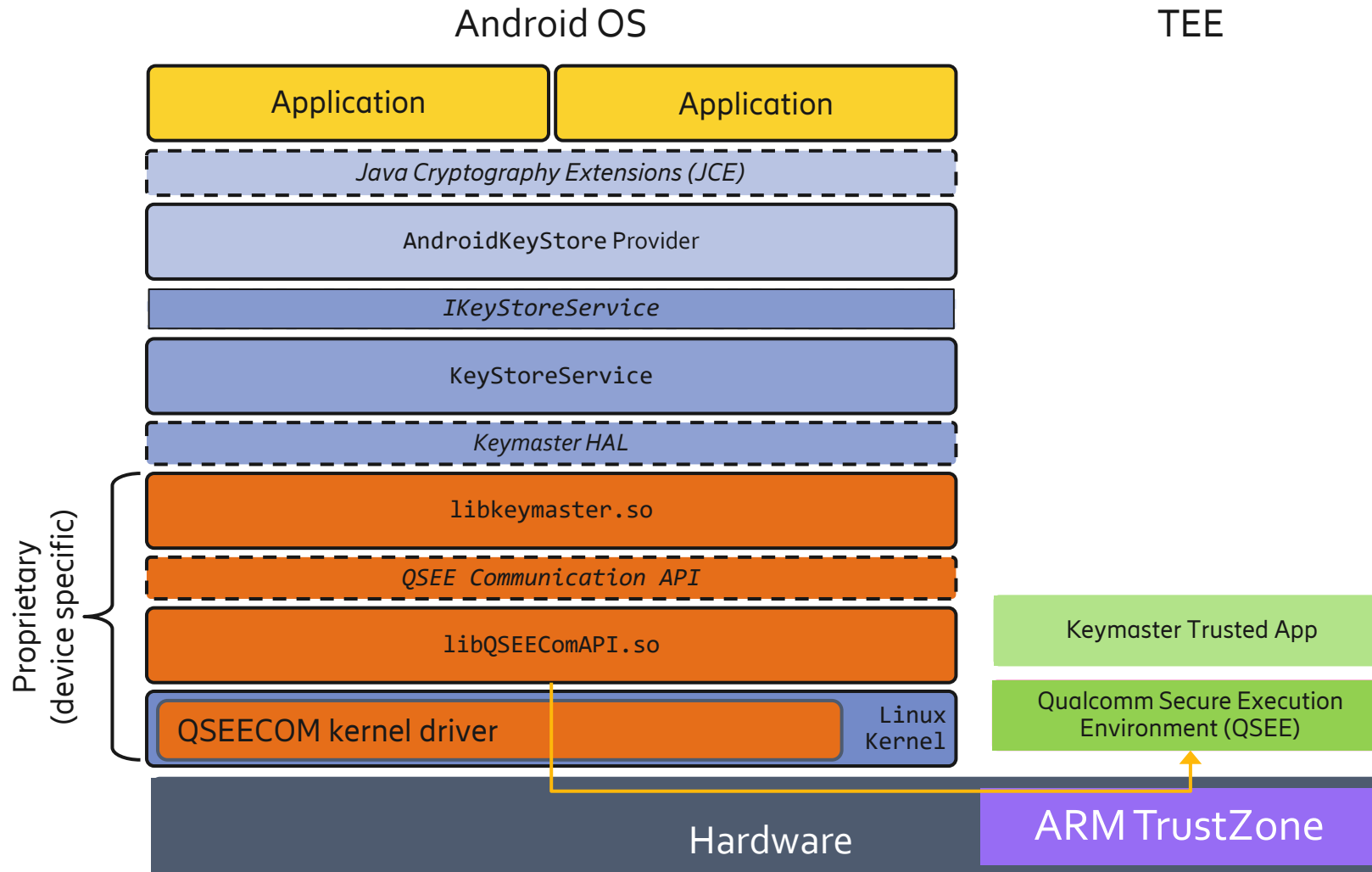
### Keymaster / KeyMint Trusted App

- backed-by processor secure environment (mandatory since 7.0)

### Strongbox Keymaster

- backed-by hardware security module with discrete CPU, secure storage, true RNG, tamper protection (option since 9.0)

# Example: Qualcomm Keymaster architecture



# Key / ID Attestation

Signed statement that improves confidence that application's keys [stored in hardware-backed keystore](#)

- attestation certificate chain root cert signed with Google attestation root key
- accessed via `getCertificateChain()` for `KeyStore` object's
- must be validated on separate trusted server!

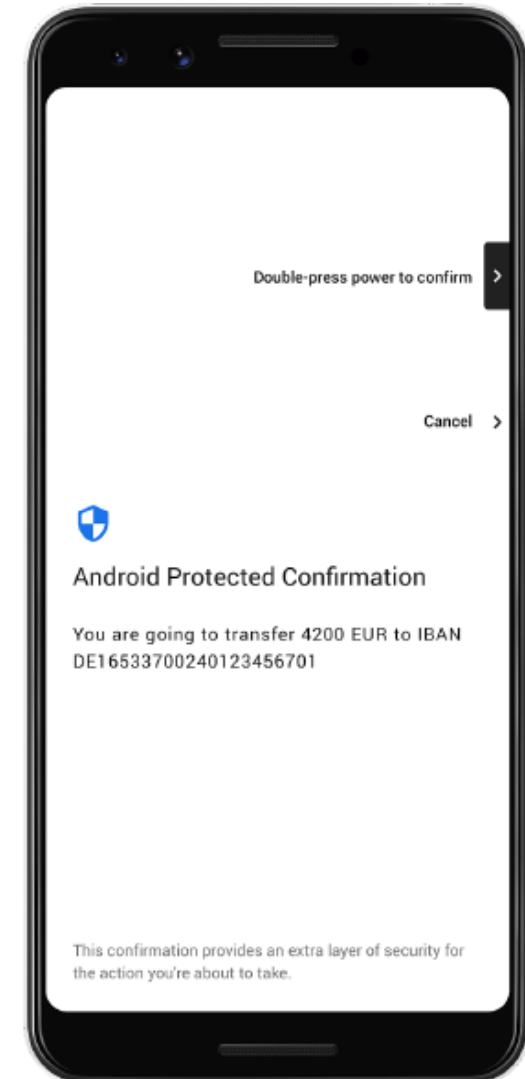
ID attestation provides proof of hardware identifiers

- e.g., brand, manufacturer, model, serial number or IMEI

# Protected Confirmation

Cryptographic keys in Keymaster can have usage requirement to assert user's approval using [Trusted UI](#) and [trusted path](#)

- protected [confirmation dialog](#) and [input method](#) outside the control of Android OS and Linux kernel
- upon confirmation, Keymaster generates [confirmation token](#) that can be conveyed and verified by remote party
- hardware implications: components controlled by kernel must not be able to drive a wire connected to physical input button



# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - **Intel SGX**
  - (AMD SEV / Intel TDX / Arm CCA)
- Concerns with TEEs & Takeaways

Extra Reading

Extra Reading

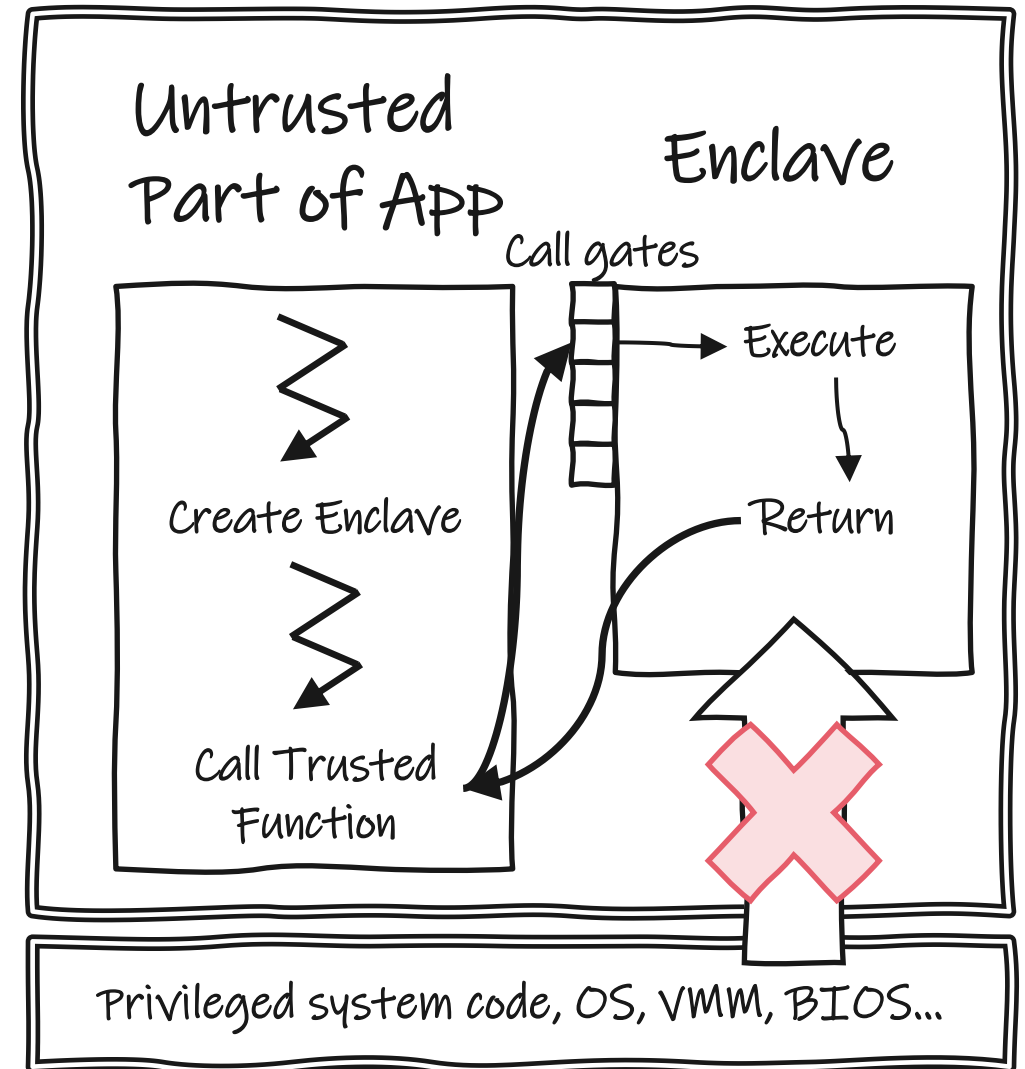
Extra Reading

Extra Reading



# Overview of Intel SGX

- Software Guard Extension (SGX)
- Hardware extension to Intel 64-bit x86 architecture
- Protects integrity and confidentiality of security-sensitive computation and data
- Designed to operate securely even when privileged software (OS, hypervisor, etc.) is potentially malicious
- Uses discrete TEEs called **enclaves**



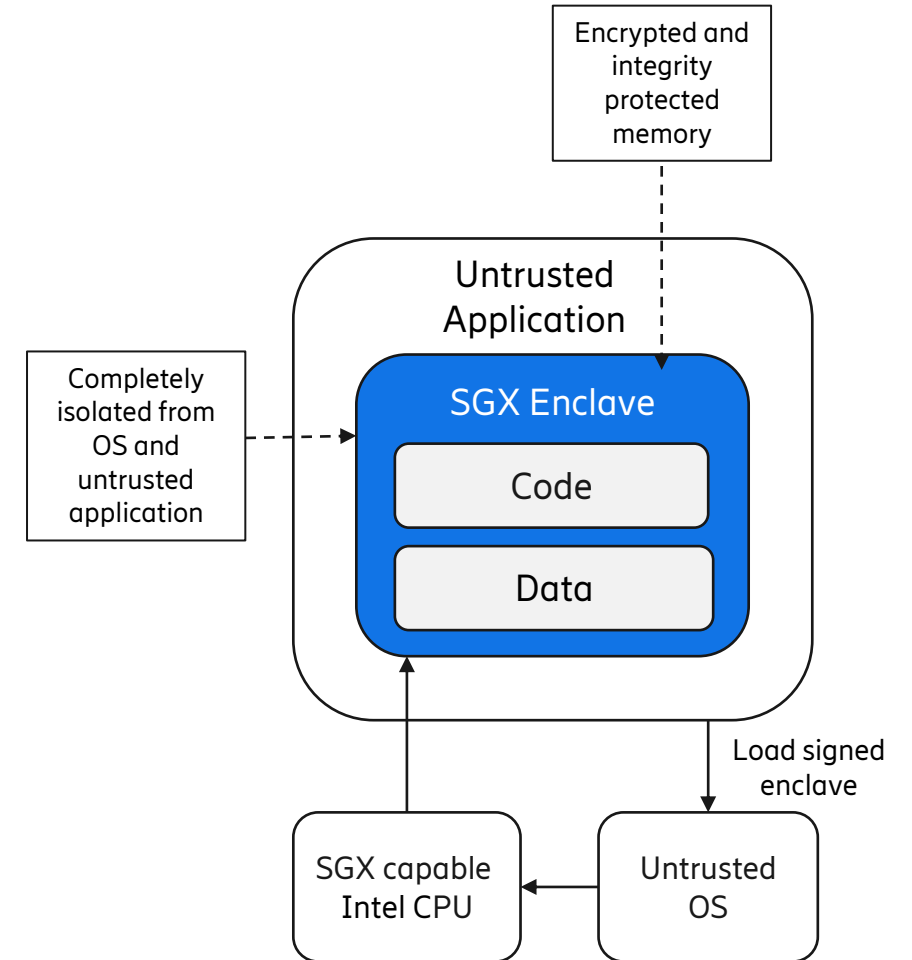
# Enclaves

Isolated and encrypted portions of a process's address space:

- Protected from host process, OS, hypervisor, firmware, and hardware devices
- OS still manages enclave resources, but cannot access protected data or code

Enclave identity and measurement:

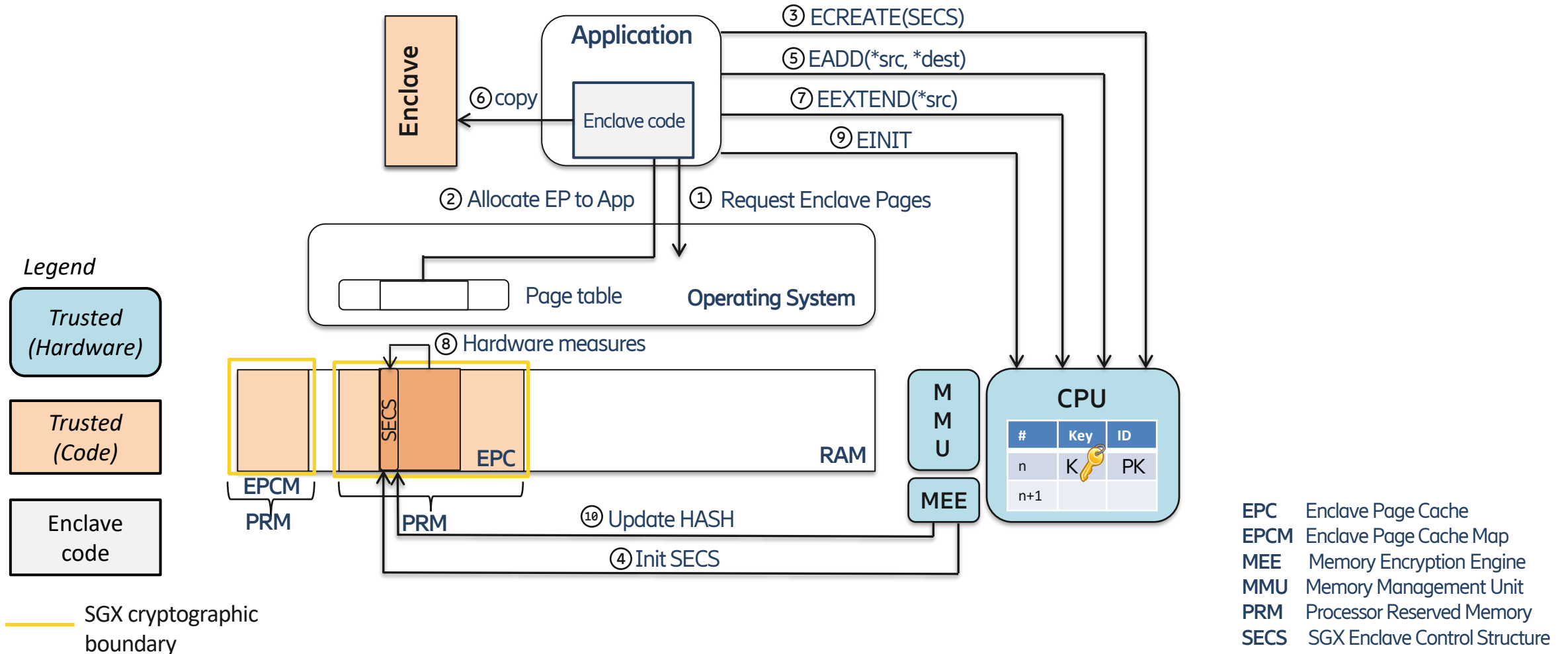
- Initial state measured via cryptographic digest (MRENCLAVE) of memory pages and security attributes (SECS)
- MRENCLAVE used for:
  - Cryptographic key derivation
  - Attestation



# Memory encryption

- Enclave memory encrypted (mainly against coldboot attack)
- Original SGX1 implementation uses dedicated MEE (Memory Encryption Engine) to handle memory (DRAM) encryption and integrity protection
- SGX2 (since Ice Lake microarchitecture) uses MKTME (Multi-Key Total Memory Encryption) with AES-XTS block-cipher for encryption, sacrificing integrity guarantees for ability to dynamically extend enclave memory

# Enclave creation



# ECALLs and OCALLs

Interactions with enclaves happens via ECALLs and OCALLs:

- **Enclave Calls (ECALLs)** : calls from applications **into the enclave**
  - The application can invoke a pre-defined function inside the enclave, passing input parameters and pointers *to shared memory within the application*.
- **Outside Calls (OCALLs)**: calls from **enclave to its application**
  - When an enclave executes, it can perform an OCALL to a pre-defined function in the application. Contrary to an ECALL, an OCALL cannot share enclave memory with the application, *so it must copy the parameters into the application memory* before the OCALL.

# Signing and identity

- Enclave binaries (.dll / .so) must be digitally signed (RSA 3072 with PKCS#1 v1.5 padding)
- Signature structure (SIGSTRUCT) includes:
  - MRENCLAVE (enclave identity)
  - MRSIGNER (digest of signer's RSA public key)
  - ISVPRODID (product identifier)
  - ISVSVN (security version number)
- Combination {MRSIGNER, ISVPRODID, ISVSVN} uniquely identifies an enclave and forms its *signer-assigned identity*

# Key derivation

Each SGX-capable platform has two root cryptographic keys burned into **one-time programmable fuses** during manufacturing:

- **Root Provisioning Key (RPK)**: known to Intel; used to prove to remote verifiers that enclaves run on genuine Intel SGX hardware
- **Root Sealing Key (RSK)**: generated inside the processor, unique per platform and known only to that processor

Enclaves cannot access RPK/RSK directly; the hardware uses them for deterministic, one-way key derivation to limit exposure

- Key derivation inputs include the root keys plus *signer-assigned identity*

Enclave retrieves derived keys via a dedicated EGETKEY instruction

# Sealing

Enclaves can derive **sealing keys** to encrypt data persisted outside the enclave and to decrypt previously sealed data across restarts

- **Enclave-specific sealing keys** tied to MRENCLAVE  
(unique to a particular enclave image running on a particular platform)
- **Signer-bound sealing keys** tied to {MRSIGNER, ISVPRODID} available to any enclave signed by the same signer/product (on a particular platform)

ISVSVN (security version) semantics:

- Newer enclave versions (higher ISVSVN) can obtain fresh sealing keys and also access sealing keys from prior versions

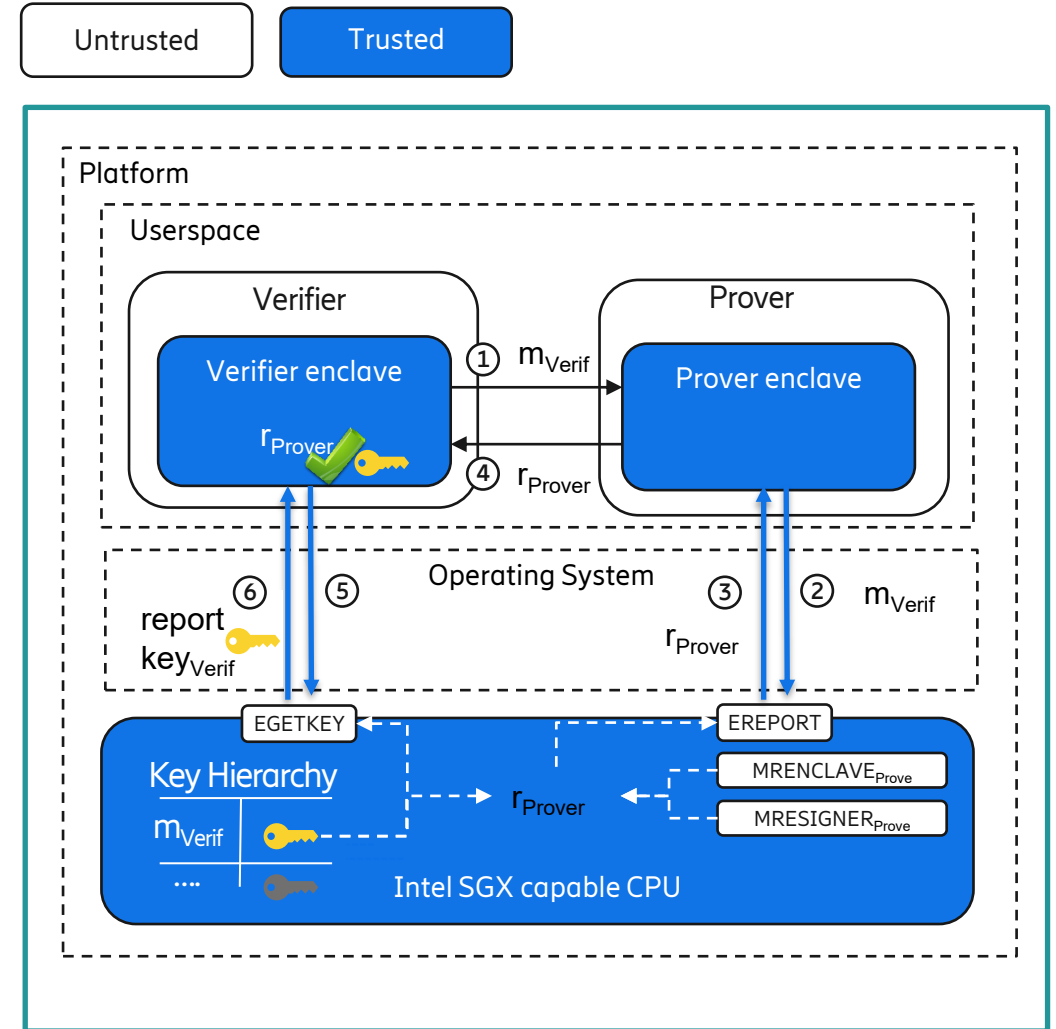
# Local attestation

Local attestation allows a verifier enclave (verif) to verify another prover enclave (prov) on the same SGX-enabled platform:

1. Verifier sends measurement ( $m_{\text{Verif}}$ ) to prover
2. Prover calls *EReport*, with  $m_{\text{Verif}}$  as parameter, to create report
3. Prover's report (ID and MAC generated using the verifier's *report key*) returned

$$r_{\text{Prover}} := \{ \text{ID}_{\text{Prover}}, \text{MAC}(\text{ID}_{\text{Prover}}, \text{RepKey}_{\text{Verifier}}) \}$$

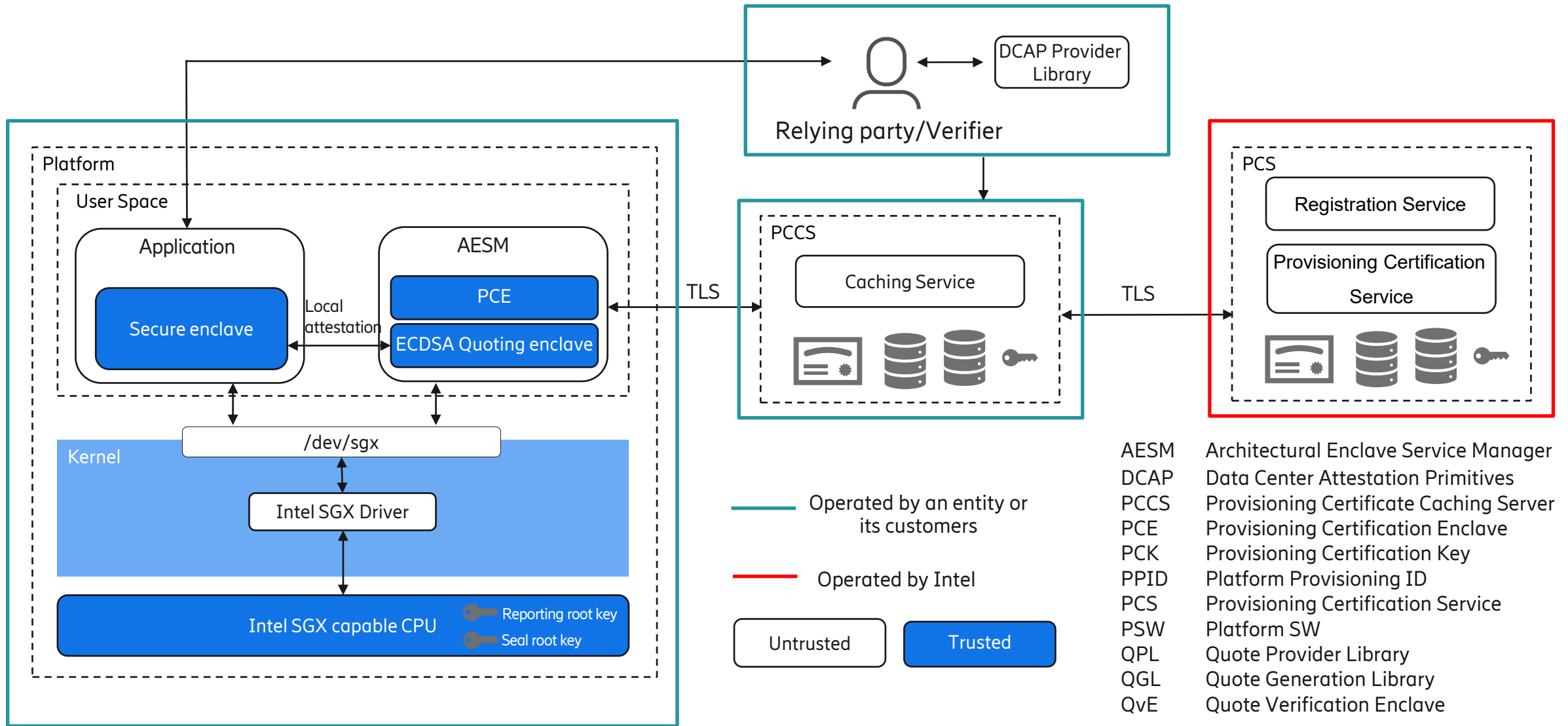
4. Report transferred to verifier
5. Verifier calls *EGETKEY* (for reports)
6. Verifier's *report key* is returned
7. MAC included in Report verified using received *report key*



# OS services required for running enclaves

- [Architectural Enclave Service Manager](#) (AESM)
  - manages Architectural Enclaves (AEs)
  - acts as a system service management agent between the AEs and SGX enabled applications
  - reference implementation provided in PSW
    - runs as a daemon process `aesmd`
    - provides an untrusted API to communicate with AEs via Unix socket at `/var/run/aesmd/aesm.socket`
- [Provisioning Certificate Caching Service](#) (PCCS)
  - requests collaterals from Intel's Provisioning Certificate Service for ECDSA attestation (PCS)
  - requires API key ([instruction for how to subscribe](#))
  - reference implementation provided as part of [DCAP](#)
    - written in `node.js`
    - applications access PCCS via URL configured in `/etc/sgx_default_qcn1.conf`
    - config directory in `/opt/intel/sgx-dcap-pccs/config`

# Remote attestation: DCAP



- AESM Architectural Enclave Service Manager
- DCAP Data Center Attestation Primitives
- PCCS Provisioning Certificate Caching Server
- PCE Provisioning Certification Enclave
- PCK Provisioning Certification Key
- PPID Platform Provisioning ID
- PCS Provisioning Certification Service
- PSW Platform SW
- QPL Quote Provider Library
- QGL Quote Generation Library
- QvE Quote Verification Enclave

# Intel SGX DCAP attestation data

- Attestation report body includes the following information (see [sgx\\_report\\_body\\_t](#))

```
typedef struct _report_body_t
{
    sgx_cpu_svn_t      cpu_svn;      /* ( 0) Security Version of the CPU */
    sgx_misc_select_t  misc_select;  /* ( 16) Which fields defined in SSA.MISC */
    uint8_t            reserved1[SGX_REPORT_BODY_RESERVED1_BYTES]; /* ( 20) */
    sgx_isvext_prod_id_t  isv_ext_prod_id; /* ( 32) ISV assigned Extended Product ID */
    sgx_attributes_t    attributes;  /* ( 48) Any special Capabilities the Enclave possess */
    sgx_measurement_t   mr_enclave;  /* ( 64) The value of the enclave's ENCLAVE measurement */
    uint8_t            reserved2[SGX_REPORT_BODY_RESERVED2_BYTES]; /* ( 96) */
    sgx_measurement_t   mr_signer;   /* (128) The value of the enclave's SIGNER measurement */
    uint8_t            reserved3[SGX_REPORT_BODY_RESERVED3_BYTES]; /* (160) */
    sgx_config_id_t     config_id;    /* (192) CONFIGID */
    sgx_prod_id_t       isv_prod_id;  /* (256) Product ID of the Enclave */
    sgx_isv_svn_t       isv_svn;     /* (258) Security Version of the Enclave */
    sgx_config_svn_t    config_svn;   /* (260) CONFIGSVN */
    uint8_t            reserved4[SGX_REPORT_BODY_RESERVED4_BYTES]; /* (262) */
    sgx_isvfamily_id_t  isv_family_id; /* (304) ISV assigned Family ID */
    sgx_report_data_t   report_data;  /* (320) Data provided by the user */
} sgx_report_body_t;
```

- Intel SGX DCAP attestation quote. (see [sgx\\_quote3\\_t](#))

```
typedef struct _sgx_quote3_t {
    sgx_quote_header_t  header;
    sgx_report_body_t   report_body;
    uint32_t            signature_data_len;
#ifdef _MSC_VER
#pragma warning(push)
#pragma warning ( disable:4200 )
#endif
    uint8_t             signature_data[];
#ifdef _MSC_VER
#pragma warning(pop)
#endif
} sgx_quote3_t;
```

# Shortcomings of SGX

- **Security vulnerabilities:** History of flaws, including side-channel attacks that can leak keys or other confidential data, often requiring microcode patches
- **Complexity:** Developing applications leveraging SGX requires specialized experience and tools
- **Performance overhead:** Due to encryption and decryption when moving enclave pages between EPC and CPU, and data copied from/to the enclave and its host application
- **Hardware dependencies and deprecation:** SGX requires actively supported hardware from Intel
  - SGX has been deprecated in 11<sup>th</sup> generation and later Core processor, currently only available in Xeon server platform

# Notable OSS: Gramine



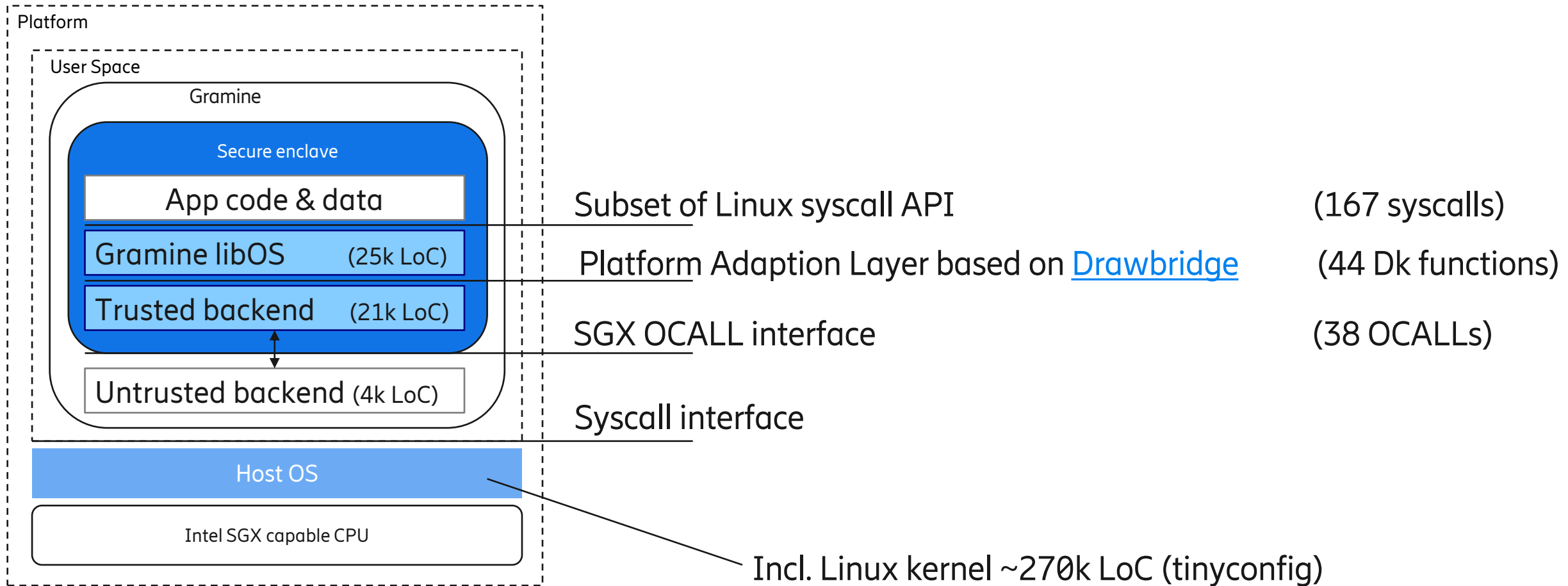
Gramine (formerly Graphine): <https://gramineproject.io/>

- Library OS for running unmodified applications within Intel SGX (and TDX)
- “Production ready” 1.0 release September 2021 – adopted to Confidential Computing Consortium

Developer-friendly interfaces for attestation:

- low-level interface: /dev/attestation
- Mid-level interface: RA-TLS library (TLS certificates with SGX Quote embedded)
- High-level interface: secret provisioning library (automatic RA-TLS channel)

# Gramine architecture



# Table of contents

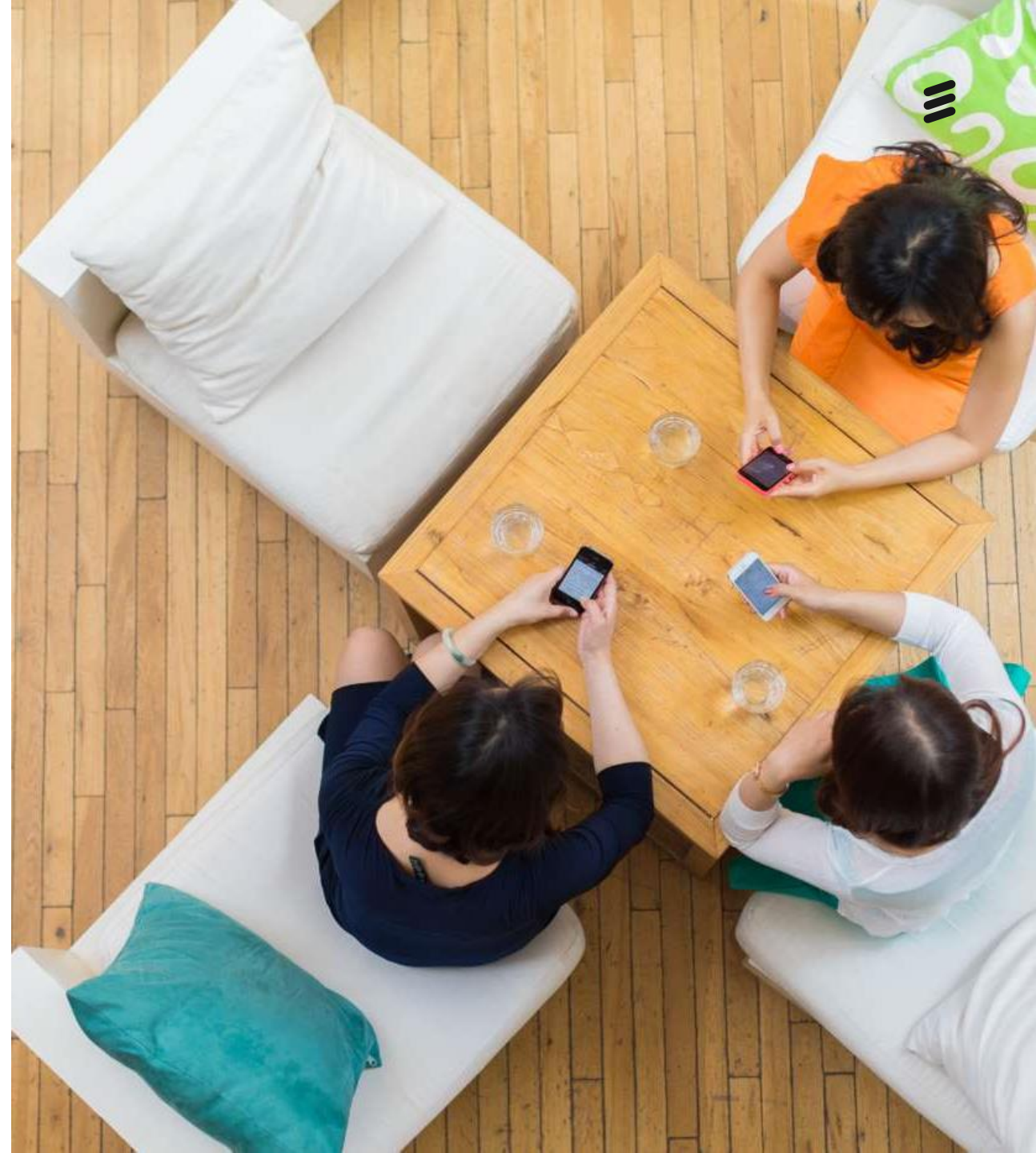
- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (AMD SEV / Intel TDX / Arm CCA)
- Concerns with TEEs & Takeaways

Extra Reading

Extra Reading

Extra Reading

Extra Reading



# Processor vendors moving towards confidential VM-based technology for cloud

All major processor vendors deploying VM-based confidential compute technology:

- AMD SEV-SNP in AMD EPYC processors
- ARM CCA RME in ARMv9 architecture
- AWS Nitro Enclaves on EC2 instances
- IBM PEF in POWER9
- Intel TDX in 4<sup>th</sup> gen Intel Xeon processors

Confidential computing on PC platforms discontinued:

- AMD Ryzen family [does not support any flavor of SEV](#)
- [11<sup>th</sup> gen Intel Core series processor](#) (Alder Lake) no longer supports Intel SGX
  - Mainly impacts DRM solutions, e.g., [UHD Blue-ray and certain PC games unplayable on Alder Lake](#)
- [4<sup>th</sup> gen Intel Xeon server](#) (Sapphire Rapids) onward support both [SGX](#) and [TDX](#)

# Alternatives to HW-based TEEs: Cryptographic Computing

- Homomorphic Encryption (HE) refers to an encryption scheme “[that allows computation to be directly on encrypted data, without requiring any decryption in the process](#)”
- Invented in 2009
  - but the origins go back to a paper (titled as “[On Data Banks and Privacy Homomorphisms](#)”) published by Ronald Linn Rivest and Len Adleman in 1978
  - the existence of a Full HE scheme was demonstrated in 2009 by [Craig Gentry](#)
- Publicly available SW implementations are available: [Microsoft SEAL](#) , [HELib](#) (IBM) and [PALISADE](#)
- FHE is far from being practical due to [massive overhead in computation and memory](#) (except for some special cases)

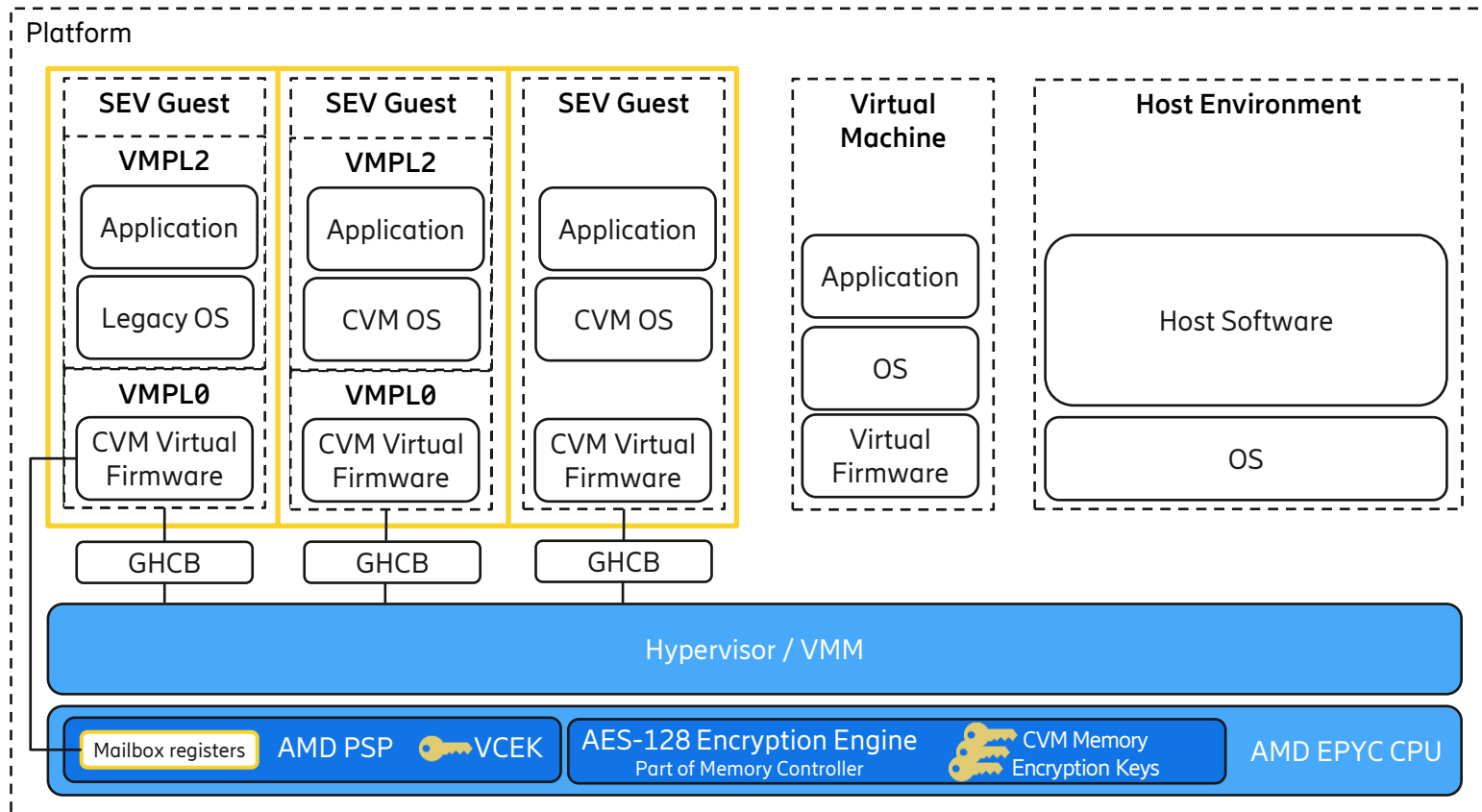
# AMD SEV-SNP Overview

CVM cryptographic boundary

Untrusted

Trusted

Manages resources



CVM  
GHCP  
PSP  
SEV  
SNP  
VCEK  
VMM  
VMPL

Confidential Virtual Machine  
Guest-Hypervisor Communication Block  
Platform Security Processor  
Secure Encrypted Virtualization  
Secure Nested Paging  
Versioned Chip Endorsement Key  
Virtual Machine Monitor  
Virtual Machine Privilege Level

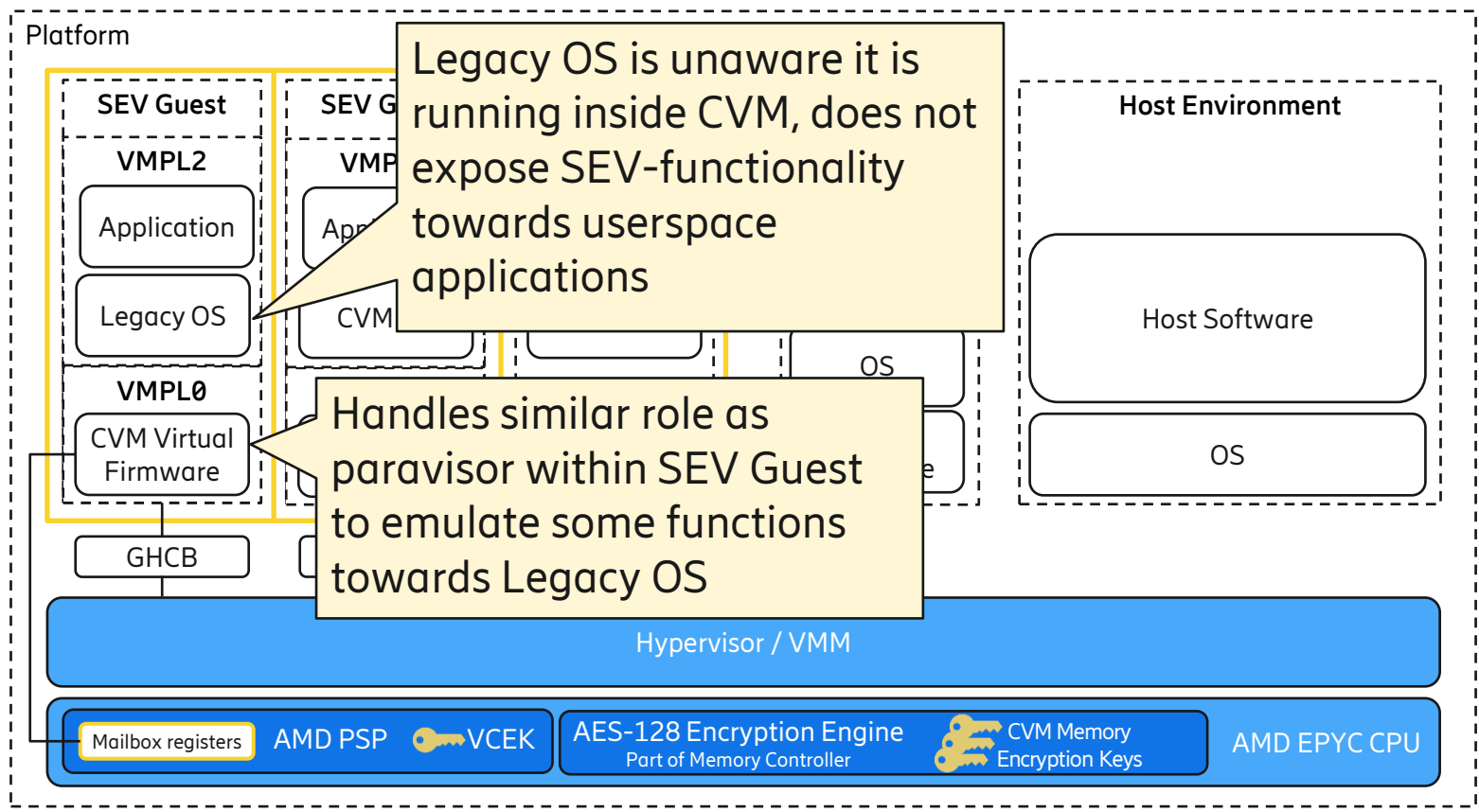
# AMD SEV-SNP Overview

— CVM cryptographic boundary

Untrusted

Trusted

Manages resources



- CVM Confidential Virtual Machine
- GHCP Guest-Hypervisor Communication Block
- PSP Platform Security Processor
- SEV Secure Encrypted Virtualization
- SNP Secure Nested Paging
- VCEK Versioned Chip Endorsement Key
- VMM Virtual Machine Monitor
- VMPL Virtual Machine Privilege Level

# Briefly about Intel TDX

- [Intel TDX](#) is an instruction set architecture extension that enables hardware-isolated Virtual Machines (VMs) called [Trust Domains](#) (TDs) (similar to AMD SEV-ES Secure Encrypted Virtualization)
- TDX capabilities are built on several, pre-existing Intel technologies, including Intel [TXT](#), [MKTME](#), [SGX](#) and instruction set extension for [VT-x](#)
- Provides memory encryption using AES-128- XTS and integrity using 28-bit MAC and a TD-ownership bit.
- Remote attestation designed to provide evidence of TD executing on a genuine, Intel TDX system and its TCB version.

# Intel TDX Overview



SEAM is a new CPU mode to support Intel-provided, digitally signed security service modules. The CPU only allows access to the SEAM-memory range to software executing inside the SEAM-memory range.

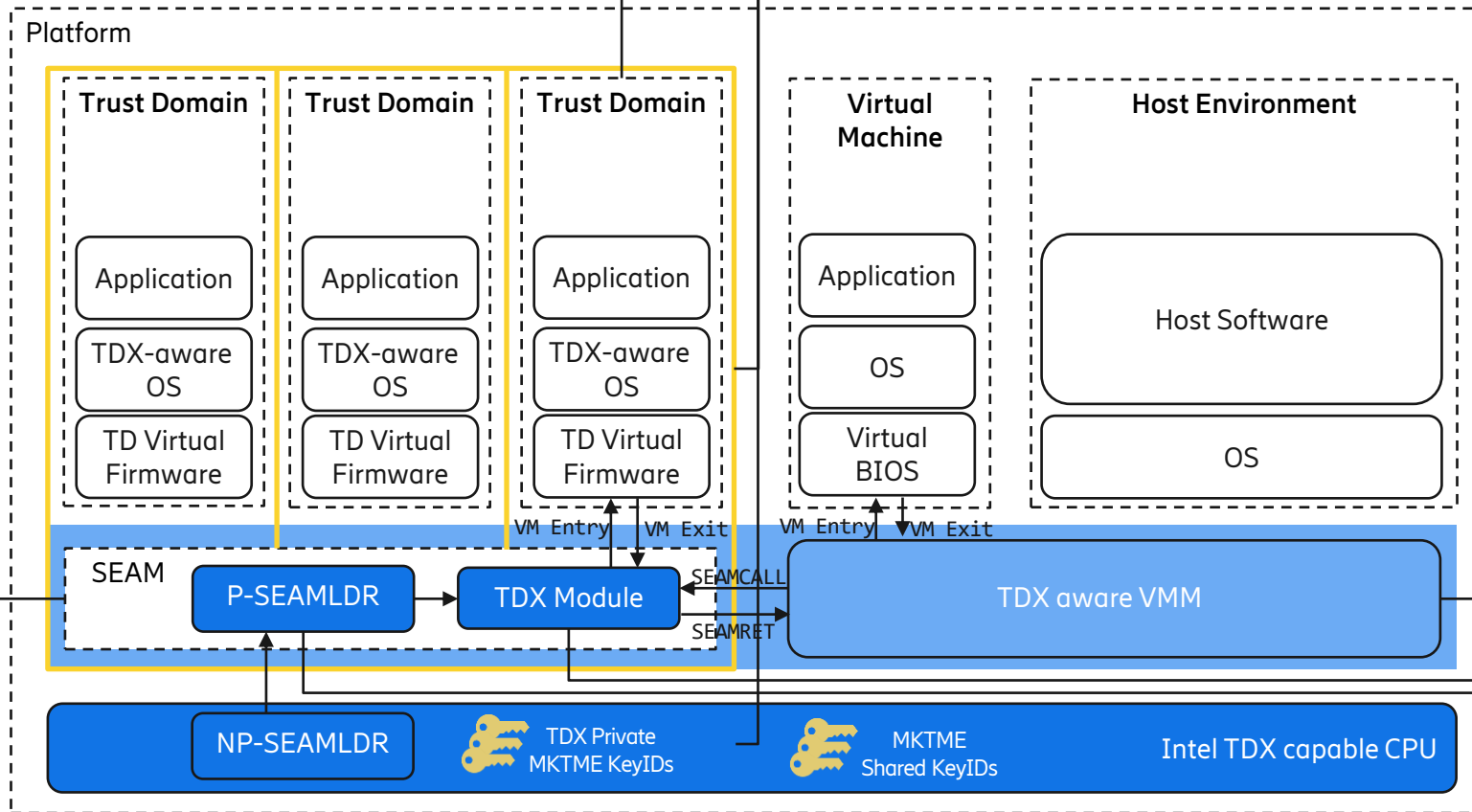
The TDs are VMs that are isolated from the VMM, and other non-TD software on the host platform.

TDX uses the **MKTME** engine to enable memory encryption for TD memory. TDX has access to a set of private MKTME KeyID(s) that are only accessible to the TDX Module

The **VMM** continues to be the resource manager, and TDs do not have privileges to deny service to VMM. DoS by VMM outside TDX threat model.

The **TDX Module** provides an interface to the VMM to create, delete, and schedule execution of TDs. It also enforces that the execution controls active for a TD do not allow the VMM (or other untrusted entities) to intercept TD accesses to protected resources.

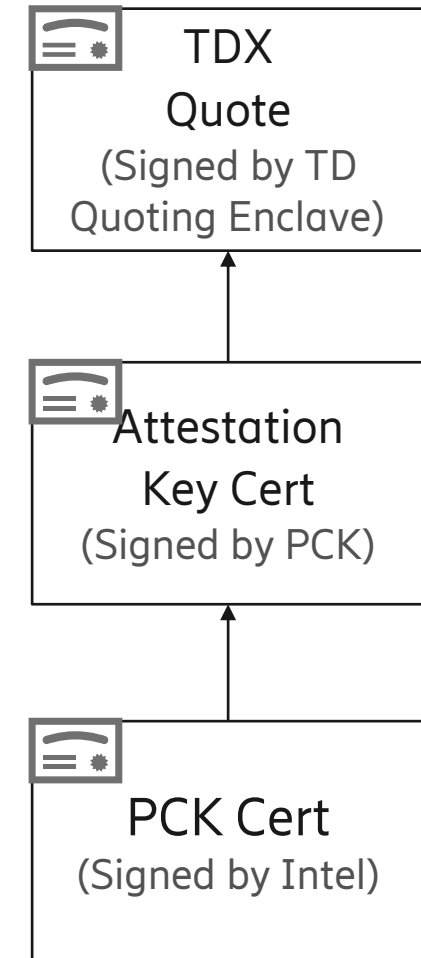
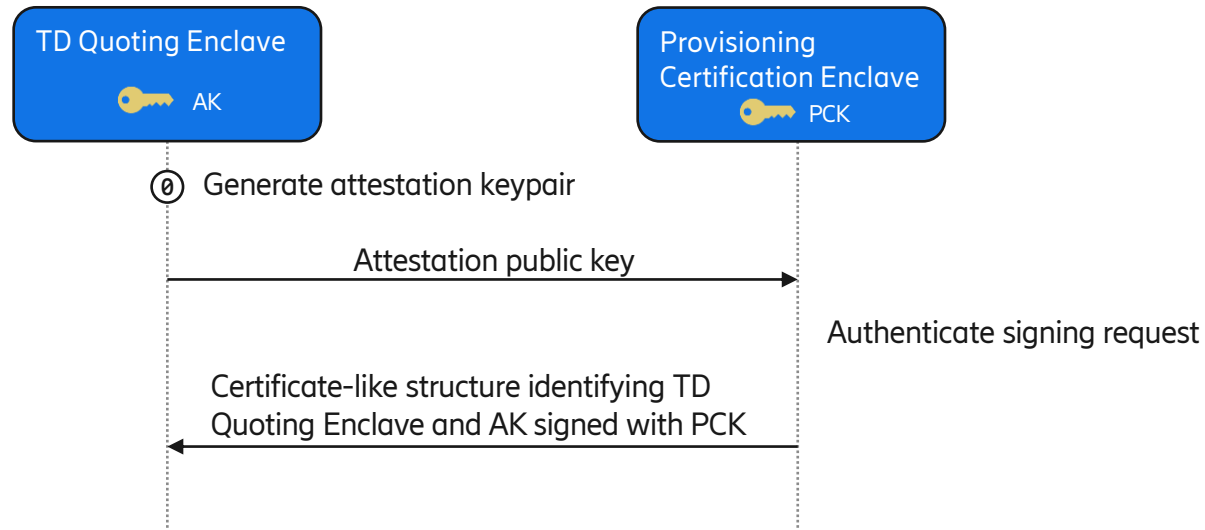
**SEAMLDR** verifies the digital signature on the Intel TDX module and loads it into the SEAM-memory range. The persistent SEAMLDR in SEAM is itself loaded during boot by a non-persistent SEAMLDR Intel TXT ACM.



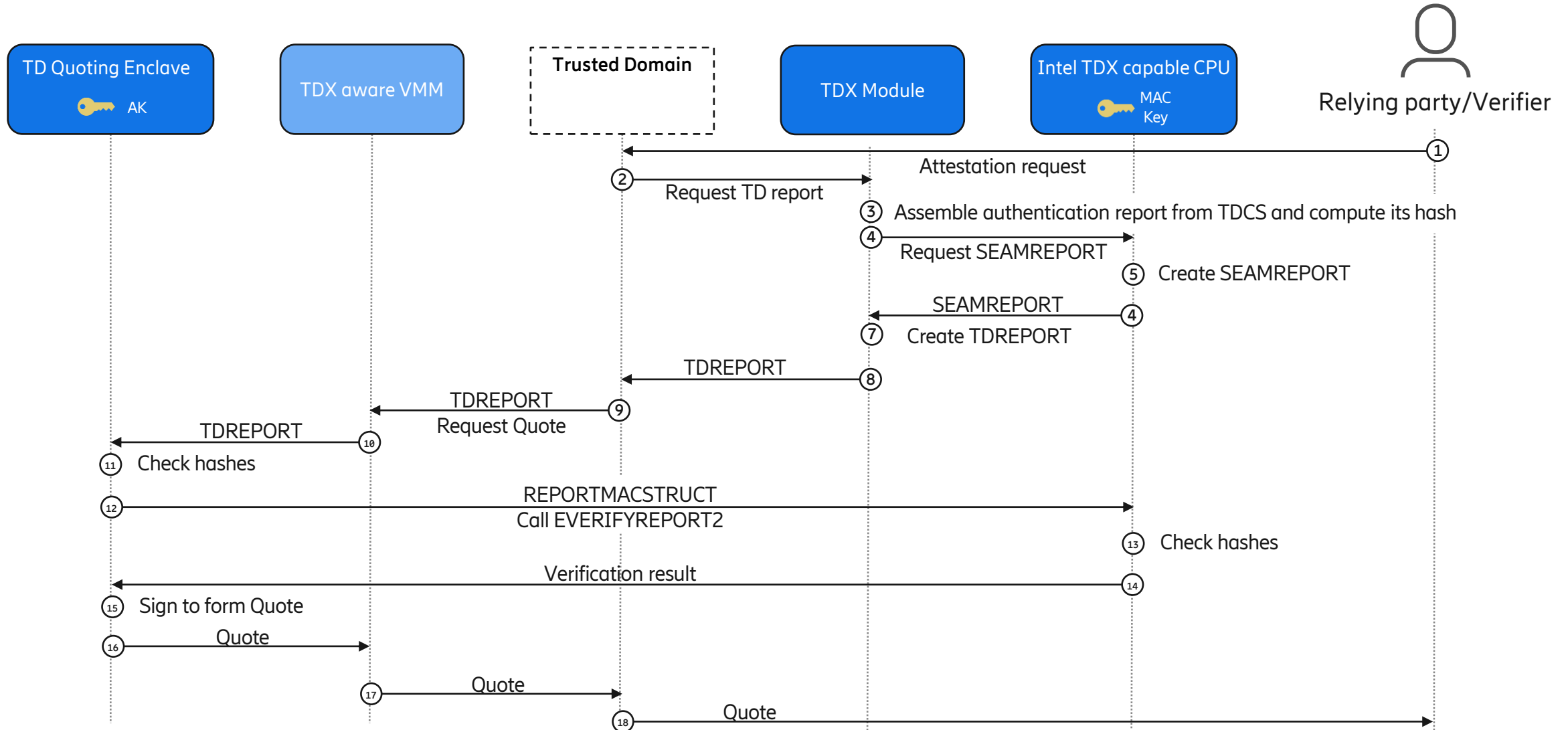
ACM	Authenticated Code Module
AK	Attestation Key
MKTME	Multi-Key Total Memory Encryption
NP-SEAMLDR	Non-Persistent SEAM Loader
SEAM	Secure Arbitration Mode
P-SEAMLDR	Persistent SEAM Loader
SEAM	Secure Arbitration Mode
TDX	Trust Domain Extension
TXT	Trusted Execution Technology
VMM	Virtual Machine Monitor

# Intel TDX Chain of Trust

- The TDX certification infrastructure builds upon the [SGX Data Center Attestation Primitives \(DCAP\)](#)
- The [SGX Provisioning Certification Enclave \(PCE\)](#) acts as local certificate authority for TD Quoting Enclave(s)



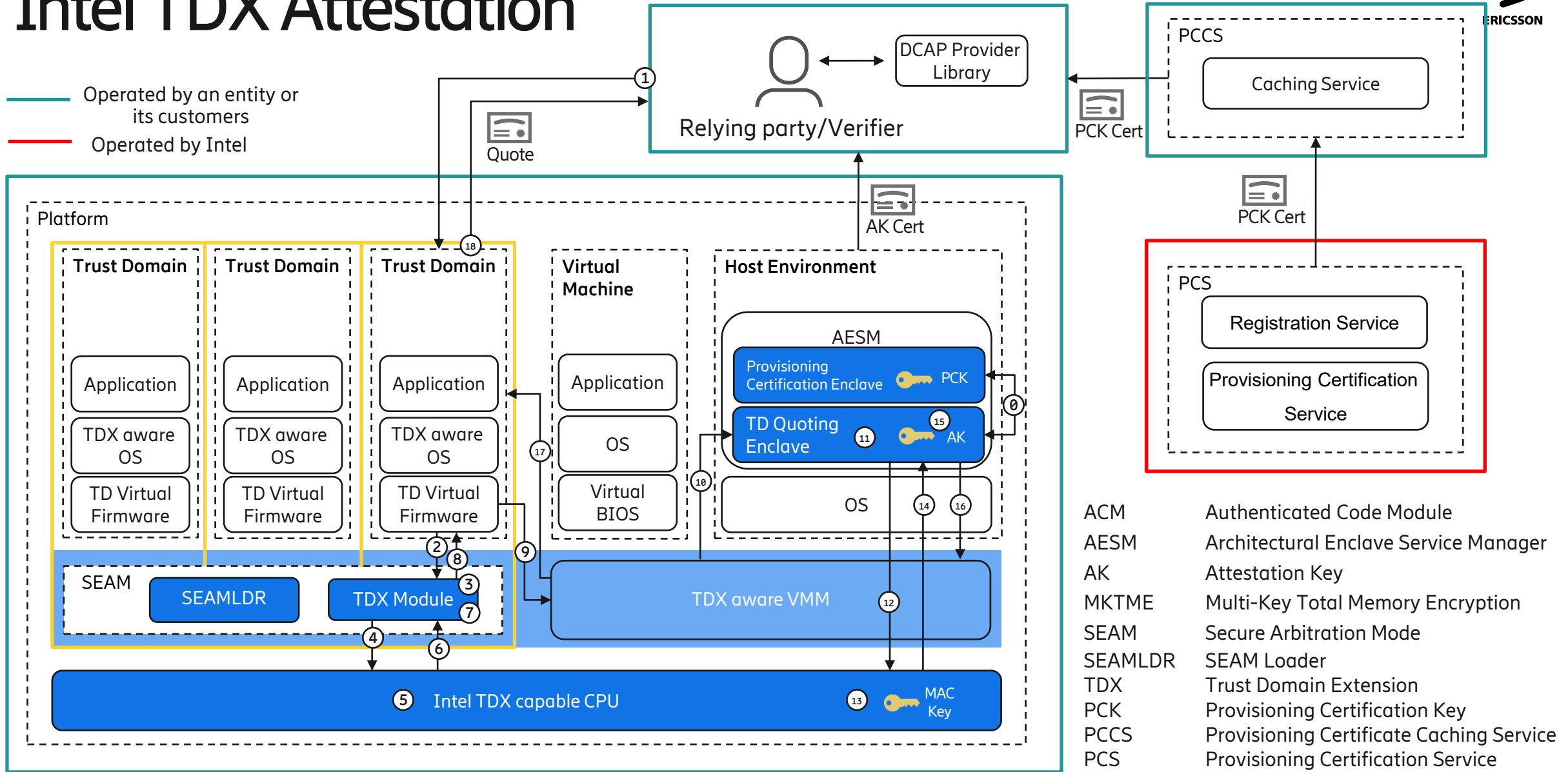
# Intel TDX Attestation Flow



# Intel TDX Attestation



- Operated by an entity or its customers
- Operated by Intel



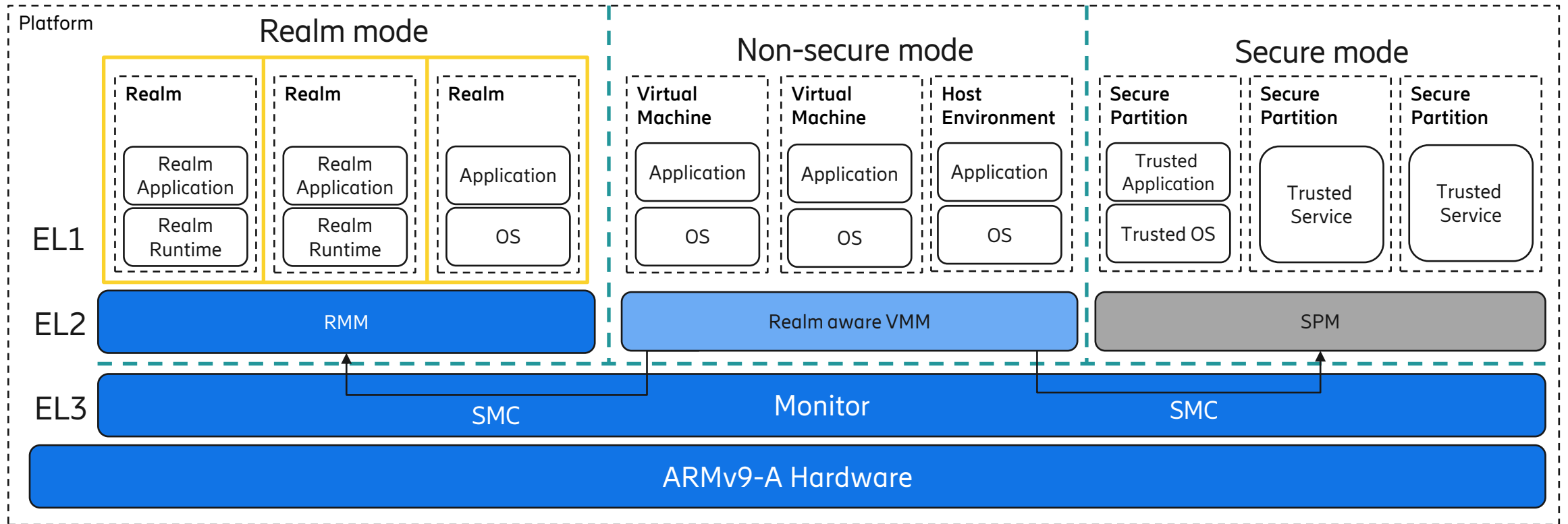
# Briefly about ARM CCA

- [Armv9-A Realm Management Extension](#) introduces [Realms](#) and new isolation boundaries at VM level (very similar to AMD SEV-SNP / Intel TDX)
- Realms cannot be accessed by [other Realms](#), [host OS / hypervisor](#), or by [TrustZone](#)
- Memory is protected in two ways:
  - Granule Protection Check (GPC): new Granule Protection Table (GPT) data structure determines the current physical address space of a page
  - Encryption: mainly for coldboot attacks
- Hypervisor still manages the resources of a Realm VM (memory, scheduling)

# ARM CCA



- CCA Confidential Computing Architecture
- EL Exception Level
- RMM Realm Management Monitor
- SMC Secure Monitor Call
- SPM Secure Partition Manager
- VMM Virtual Machine Monitor



# Table of contents

- Getting to know each other
- Quick introduction to cryptographic trust
- Why trusted computing?
  - Hardware security mechanisms
  - Trusted execution environments
- Processor secure environments
  - Arm TrustZone (case study: Android)
  - Intel SGX
  - (AMD SEV / Intel TDX / Arm CCA)

Extra Reading

Extra Reading

Extra Reading

Extra Reading

• Concerns with TEEs & Takeaways



# Concerns with TEEs: hardware flaws

## TPM Reset Attack

50,012 views



Evan Sparks

Published on Jun 18, 2007

A demonstration of a vulnerability in the TCG arch running TPM without restarting the platform.

<http://www.cs.dartmouth.edu/~pkilab/sparks/> (2007)

## CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management

Authors:

Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo, *Columbia University*

*Distinguished Paper Award Winner!*

<https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang> (2017)

## Foreshadow (security vulnerability)

From Wikipedia, the free encyclopedia

*This article is about the security vulnerability. For other uses, see Foreshadow (disambiguation).*

**Foreshadow** is a vulnerability that affects modern microprocessors that was first discovered by two independent teams of researchers in January 2018, but was first disclosed to the public on 14 August 2018.<sup>[1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16]</sup> The vulnerability is a speculative execution attack on Intel processors that may result in the loss of sensitive information stored in personal computers, or third party clouds.<sup>[1]</sup> There are two versions: the first version (original/Foreshadow) (CVE-2018-3615<sup>[4]</sup>) targets data from SGX enclaves; and the second version (next-generation/Foreshadow-NG <sup>[8]</sup>) (CVE-2018-3620<sup>[8]</sup> and CVE-2018-3646<sup>[8]</sup>) targets Virtual Machines (VMs), hypervisors (VMM), operating system (OS) kernel memory, and System Management Mode (SMM) memory.<sup>[1]</sup> Intel considers the entire class of speculative execution side channel vulnerabilities as "L1 Terminal Fault" (L1TF).<sup>[1]</sup> A listing of affected Intel hardware has been posted.<sup>[10][11]</sup>

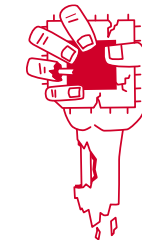
Foreshadow is similar to the Spectre security vulnerabilities discovered earlier to affect Intel and AMD chips, and the Meltdown vulnerability that also affected Intel.<sup>[6]</sup> However, AMD products, according to AMD, are not affected by the Foreshadow security flaws.<sup>[6]</sup> According to one expert, "[Foreshadow] lets malicious software break into secure areas that even the Spectre and Meltdown flaws couldn't crack".<sup>[15]</sup> Nonetheless, one of the variants of Foreshadow goes beyond Intel chips with SGX technology, and affects "all [Intel] Core processors built over the last seven years".<sup>[2]</sup>

Foreshadow may be very difficult to exploit,<sup>[2][6]</sup> and there seems to be no evidence to date (15 August 2018) of any serious hacking involving the Foreshadow vulnerabilities.<sup>[2][6]</sup> Nevertheless, applying software patches may help alleviate some concern(s), although the balance between security and performance may be a worthy consideration.<sup>[5]</sup> Companies performing cloud computing may see a significant decrease in their overall computing power; individuals, however, may not likely see any performance impact, according to researchers.<sup>[9]</sup> The real fix, according to Intel, is by replacing today's processors.<sup>[9]</sup> Intel further states, "These changes begin with our next-generation Intel Xeon Scalable processors (code-



FORESHADOW

A logo created for the vulnerability, featuring a lock with a shadow



(CCS 2019)

[https://en.wikipedia.org/wiki/Foreshadow\\_\(security\\_vulnerability\)](https://en.wikipedia.org/wiki/Foreshadow_(security_vulnerability)) (2018)

# Concerns with TEEs: software flaws

## ATTACKING YOUR TRUSTED CORE: EXPLOITING TRUSTZONE ON ANDROID

PRESENTED BY

Di Shen

For years fingerprint scanning has been supported in many Android devices. Fingerprint scanning on ARM always needs an implementation of TrustZone. While we enjoy unlocking devices and paying by fingerprint, we also figure out these new features bring out some new attack surfaces. Attacking the kernel of Android or the secure world of TrustZone may be not impossible.

<https://www.blackhat.com/us-15/briefings.html#attacking-your-trusted-core-exploiting-trustzone-on-android> (2015)

02/05/2016

### QSEE privilege escalation vulnerability and exploit (CVE-2015-6639)

In this blog post we'll discover and exploit a vulnerability which will allow us to gain code execution within Qualcomm's Secure Execution Environment (QSEE). I've responsibly disclosed this vulnerability to Google and it has been fixed - for the exact timeline, see the "Timeline" section below.

<https://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html> (2016)

30/06/2016

### Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption

After covering a TrustZone kernel vulnerability and exploit in the [previous blog post](#), I thought this time it might be interesting to explore some of the implications of code-execution within the TrustZone kernel. In this blog post, I'll demonstrate how TrustZone kernel code-execution can be used to effectively break Android's Full Disk Encryption (FDE) scheme. We'll also see some of the inherent issues stemming from the design of Android's FDE scheme, even without any TrustZone vulnerability.

<https://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html> (2016)

# Concerns with TEEs: suspicion of motives

Software

## MS Palladium protects IT vendors, not you – paper

Anderson gives us the FAQs

By [John Lettice](#) 28 Jun 2002 at 10:27

SHARE ▼

[https://www.theregister.co.uk/2002/06/28/ms\\_palladium\\_protects\\_it\\_vendors/](https://www.theregister.co.uk/2002/06/28/ms_palladium_protects_it_vendors/) (2002)

## Trusting Intel – Next Generation of Backdooring?

We have seen that SGX offers a number of attractive functionality that could potentially make our digital systems more secure and 3<sup>rd</sup> party servers more trusted. But does it really?

The obvious question, especially in the light of recent revelations about NSA backdooring everything and the kitchen sink, is whether Intel will have backdoors allowing “privileged entities” to bypass SGX protections?

<http://theinvisiblethings.blogspot.fi/2013/09/thoughts-on-intels-upcoming-software.html> (2013)

**Problem: Third-party uncertainty about your software environment is normally a feature, not a bug**

<https://www.eff.org/wp/trusted-computing-promise-and-risk> (2003)

# Challenge: Dealing with TEE compromise

## Hardware attacks pose a serious threat to TEEs

- No longer reasonable to assume hardware security to be inviolable

## Feature creep leads to larger software footprint

- Increases risk of software vulnerabilities that may compromise TEE security

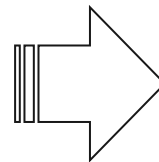
## Abandon hardware-assisted TEEs altogether?

- Instead rely only on cryptographic computing techniques?
  - e.g., multi-party computation, homomorphic encryption

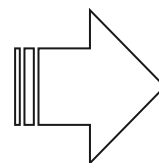
## TEEs still hold the promise of efficient solutions

- Hardware-assistance and cryptography are **not mutually exclusive!**
- **Defense-in-depth** is desirable
- Novel approaches for dealing with TEE compromise may be feasible

**Established Processor  
Secure Environments**

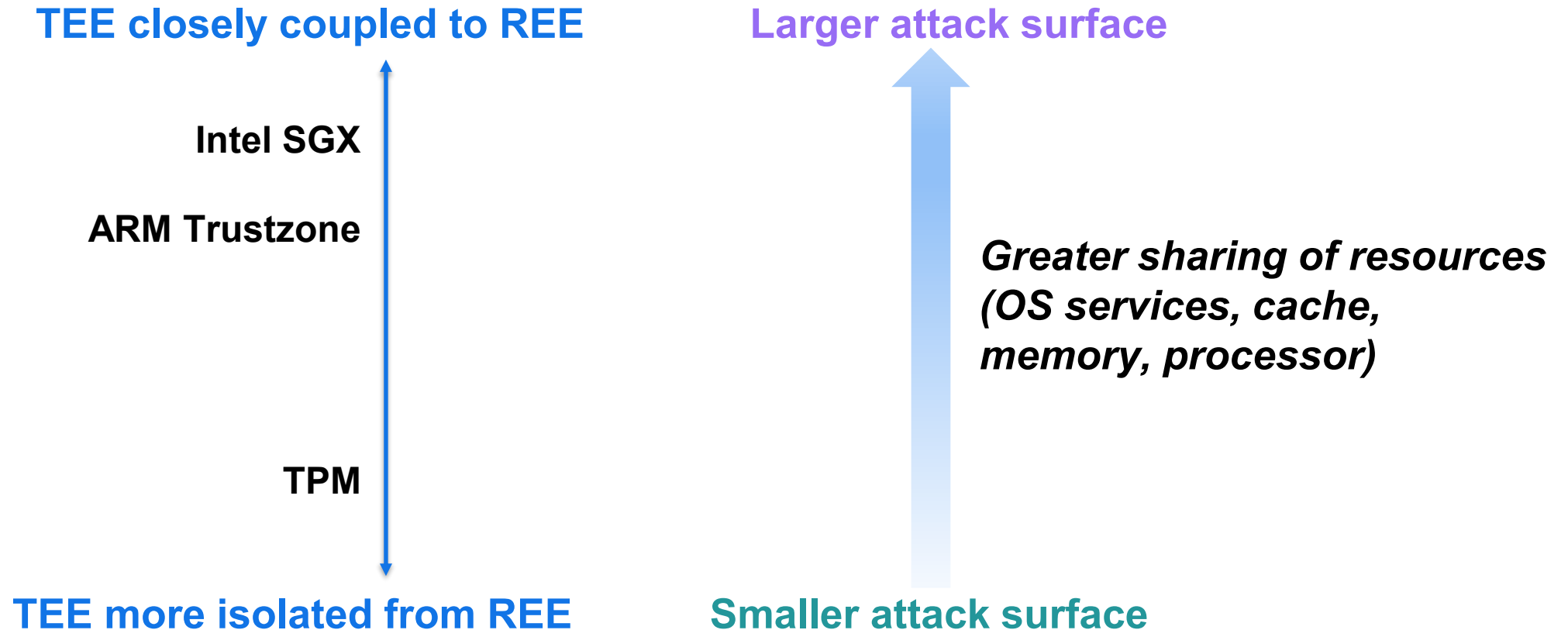


**Defense-in-depth using discrete  
security processors**



**Hardware-isolated,  
attestable virtual machines**

# Dealing with TEE compromise: minimize coupling



# Dealing with TEE compromise: defense-in-depth

## Discrete security processors in modern smartphones

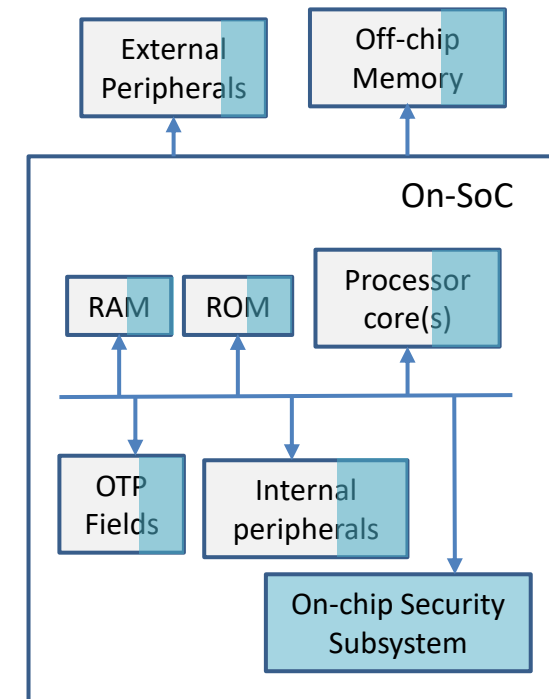
- Apple Secure Enclave Processor (SEP)  
Apple, [iOS Security](#) Whitepaper, May 2019
- Google Titan M  
Google Device Security Group, [Building a Titan](#), Android Developers Blog, October 2018
- Qualcomm Secure Processing Unit (SPU)  
Qualcomm. [How can Snapdragon 845 guard your smartphone data like a vault?](#) QnQ Blog. March 2018.

## Physical isolation **mitigates against entire classes** of hardware exploits

- Processor, caches, memory, and persistent storage are not shared with main OS

## Division of tasks (and secrets) between two (or more) elements

## Layered defense using multiple TEEs



Processor Secure Environment + Embedded Security Element

# Takeaways and acknowledgements

## TEEs have been around for more than two decades

- Dominant design choices informed by cost and usability considerations

## Unconditional trust in hardware-TEEs is no longer acceptable

- Emerging TEE architectures can provide
  - stronger defense-in-depth, or
  - enhanced ease of software deployment

## Acknowledgements:

- *N. Asokan. [Hardware-assisted Trusted Execution Environments – Look Back, Look Ahead](#). Invited keynote at the [ACM CCS 2019](#), London, November 2019 [[Video](#)]*
- Mobile Systems Security course taught at Aalto University 2014 – 2020  
*With content from: Jan-Erik Ekberg, Kari Kostinen, N. Asokan, Sini Ruohomaa, Ferdinand Brasser, Luca Davi, Ahmad-Reza Sadeghi*

# Did you learn?

- To understand trusted and confidential computing and its purpose
- Identify threats to computing hardware/infrastructure
- Get a basic insight in technologies to achieve trusted computing in devices, servers, and cloud infrastructure
- Recognize technical approaches for building trustworthy ICT systems

# Self-Study questions

- Explain the difference between “trust” and “trustworthiness” with respect to compute platforms?
- What is the purpose of a remote attestation w.r.t to trustworthiness?
- What is a RoT and give three different types of RoTs.
- What is the purpose of an Root of Trust of Measurement?
- What can Common Criteria be used for wrt. to the trustworthiness of a platform?
- To what extend can I make all parts of an ICT system trustworthy?
- How are processes in a running TrustZone-enabled system that are located in the normal or secure world isolated.



<https://www.ericsson.com/en/security>