Trusted Execution Environments on Mobile Devices

ACM CCS 2013 tutorial

Jan-Erik Ekberg, Trustonic
Kari Kostiainen, ETH Zurich
N. Asokan, University of Helsinki and Aalto University

TRUSTONIC

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

UNIVERSITY OF HELSINKI

Aalto University

What is a TEE?

Processor, memory, storage, peripherals

Trusted Execution Environment

Isolated and integrity-protected

From the “normal” execution environment (Rich Execution Environment)

Chances are that:

You have devices with hardware-based TEEs in them!

But you don’t have (m)any apps using them

Outline

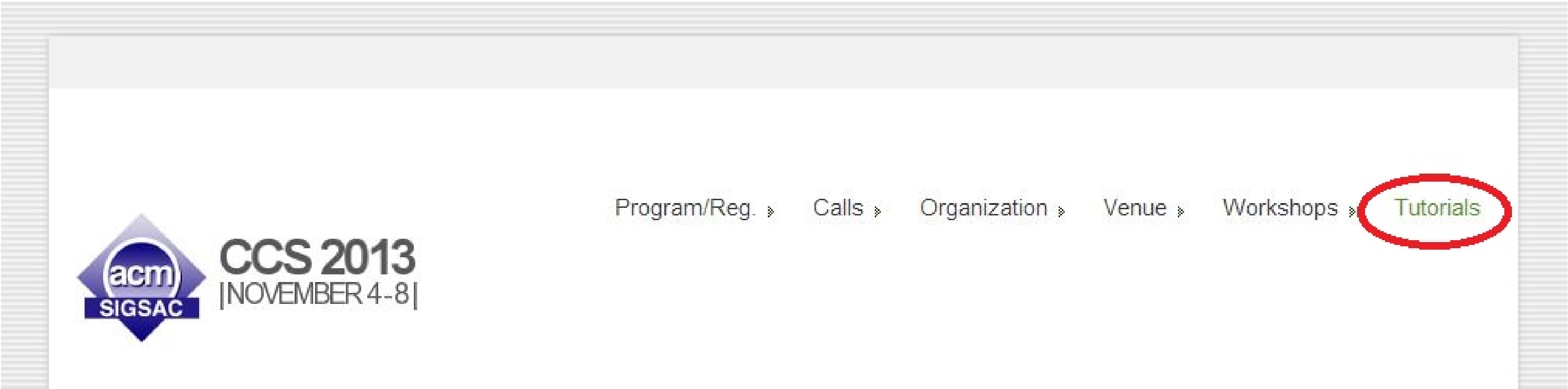
- A look back (10 min)
  - Why mobile devices have TEEs?
- Mobile hardware security (30 min)
  - What constitutes a TEE?
- Application development (30 min)
  - Mobile hardware security APIs + DEMO

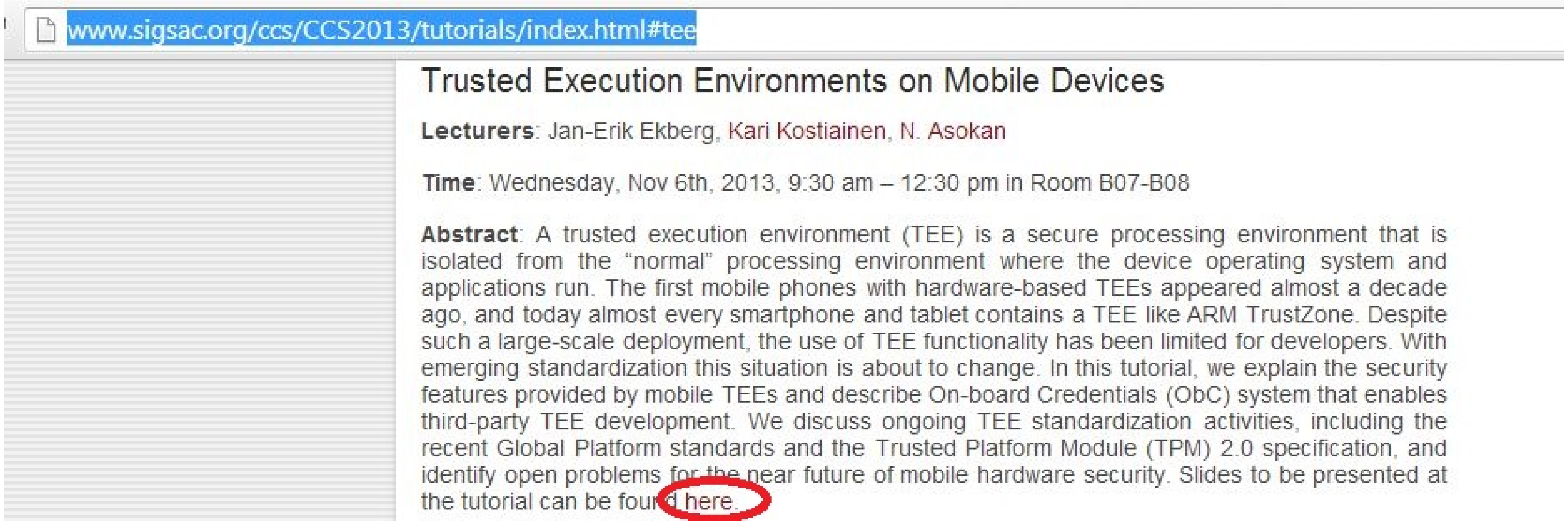
Break (10 min)

- Current standardization (60 min)
  - NIST, Global Platform, TPM 2.0
- A look ahead (10 min)
  - Challenges and summary

Tutorial based on: Ekberg, Kostiainen and Asokan. The Untapped Potential of Trusted Execution Environments on Mobile Devices. IEEE S&P magazine, (to appear). (author copy)

Tutorial slides





Trusted Execution Environments on Mobile Devices

Lecturers: Jan-Erik Ekberg, Kari Kostiainen, N. Asokan

Time: Wednesday, Nov 6th, 2013, 9:30 am – 12:30 pm in Room B07-B08

Abstract: A trusted execution environment (TEE) is a secure processing environment that is isolated from the “normal” processing environment where the device operating system and applications run. The first mobile phones with hardware-based TEEs appeared almost a decade ago, and today almost every smartphone and tablet contains a TEE like ARM TrustZone. Despite such a large-scale deployment, the use of TEE functionality has been limited for developers. With emerging standardization this situation is about to change. In this tutorial, we explain the security features provided by mobile TEEs and describe On-board Credentials (ObC) system that enables third-party TEE development. We discuss ongoing TEE standardization activities, including the recent Global Platform standards and the Trusted Platform Module (TPM) 2.0 specification, and identify open problems for the near future of mobile hardware security. Slides to be presented at the tutorial can be found here.

Why do most mobile devices today have TEEs?

A LOOK BACK

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 expectation
compared to PCs

Early adoption of platform security

GSM 02.09, 1993

```
Both IMSI and IMEI require physical protection.

Physical protection means that manufacturers shall take necessary and sufficient
measures to ensure the programming and mechanical security of the IMEI. The
manufacturer shall also ens
(where applicable) remains
```

3GPP TS 42.009, 2001

The IMSI is stored securely within the SIM.

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 1st June 2002.

Different starting points compared to PCs:
Widespread use of hardware and software platform security

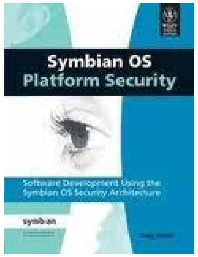
~2001

J2ME

~2002

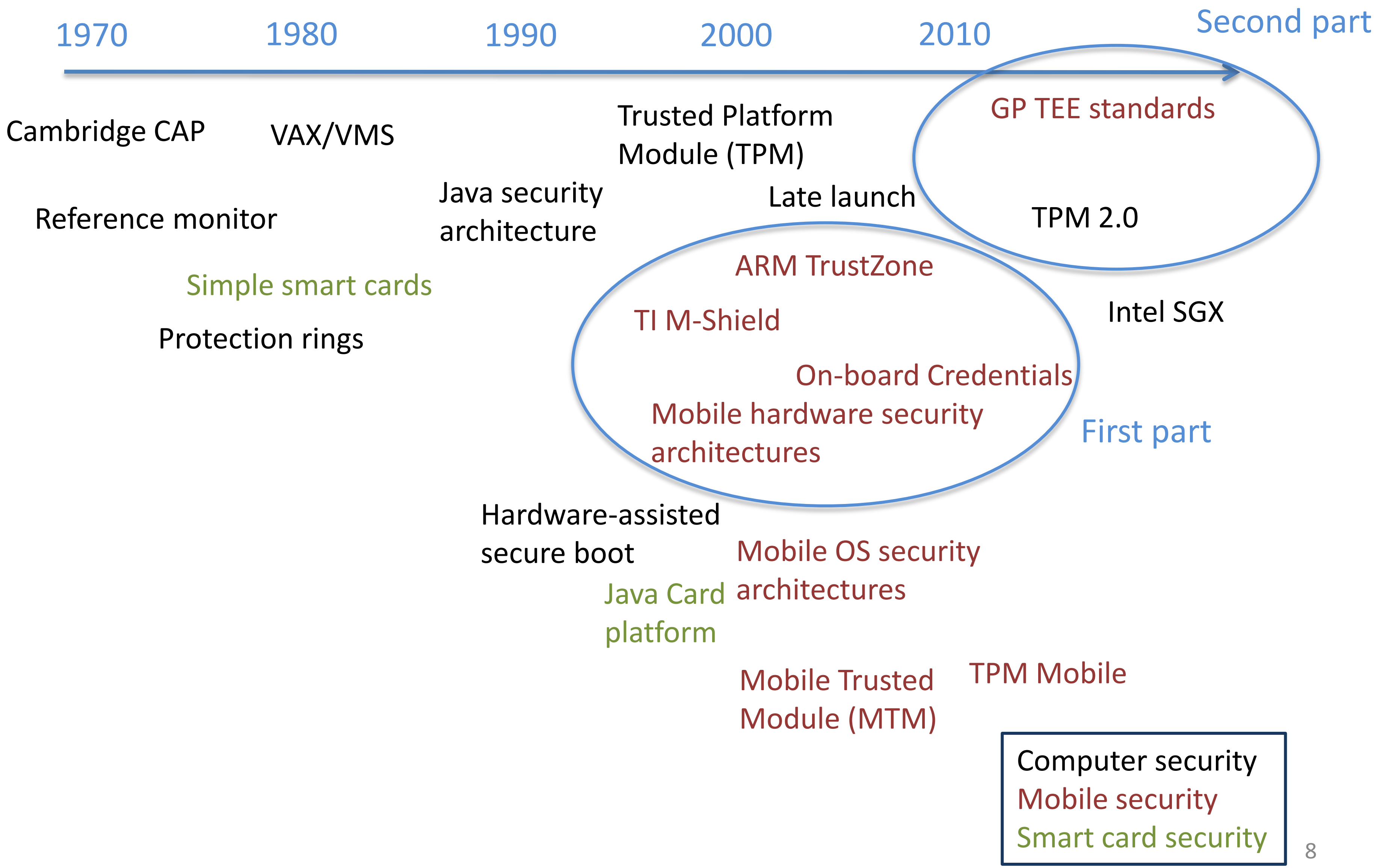
TrustZone®
Security Foundation by ARM®

~2005



~2008

Historical perspective

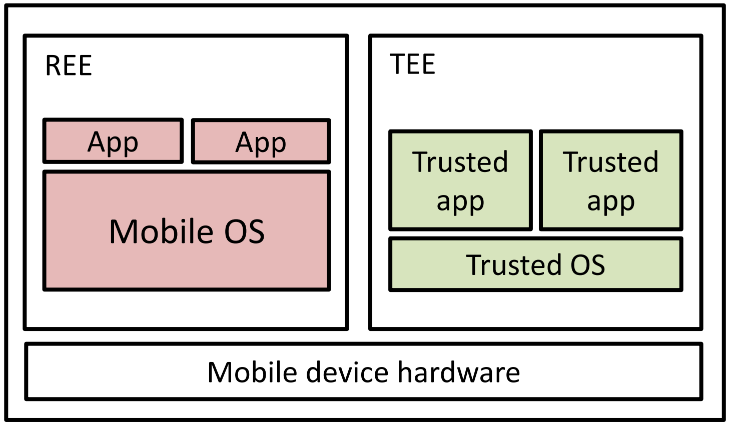


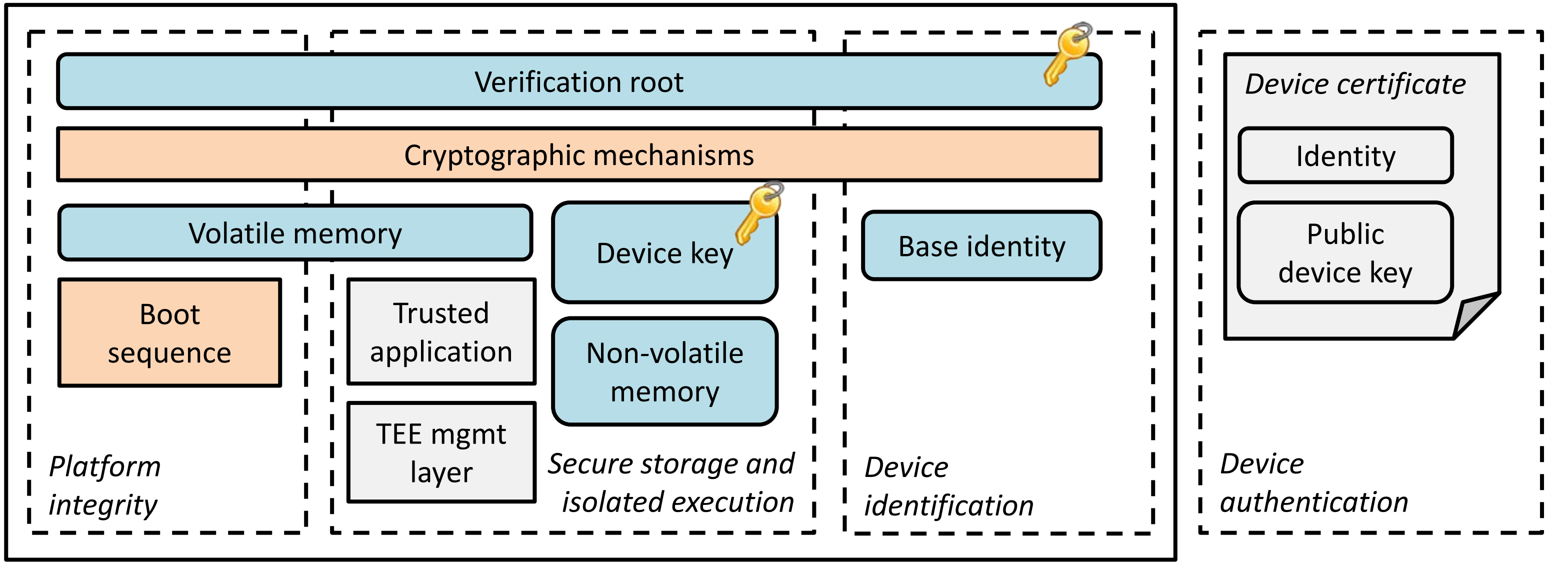
What constitutes a TEE?

MOBILE HARDWARE SECURITY

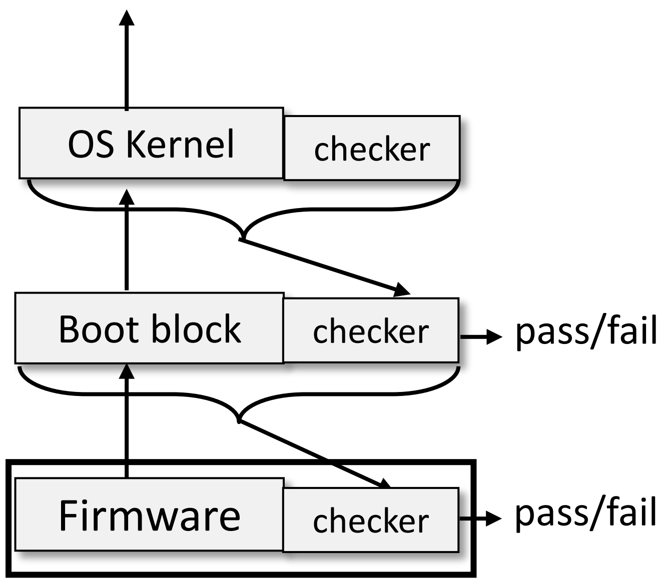
TEE overview

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

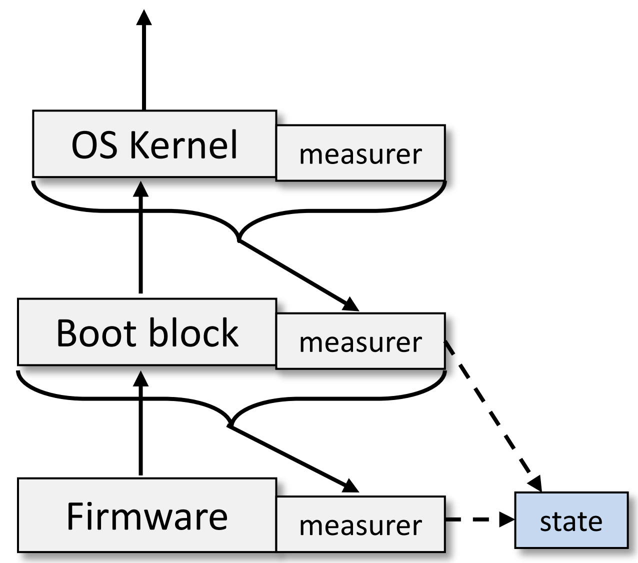




Secure boot vs. authenticated boot

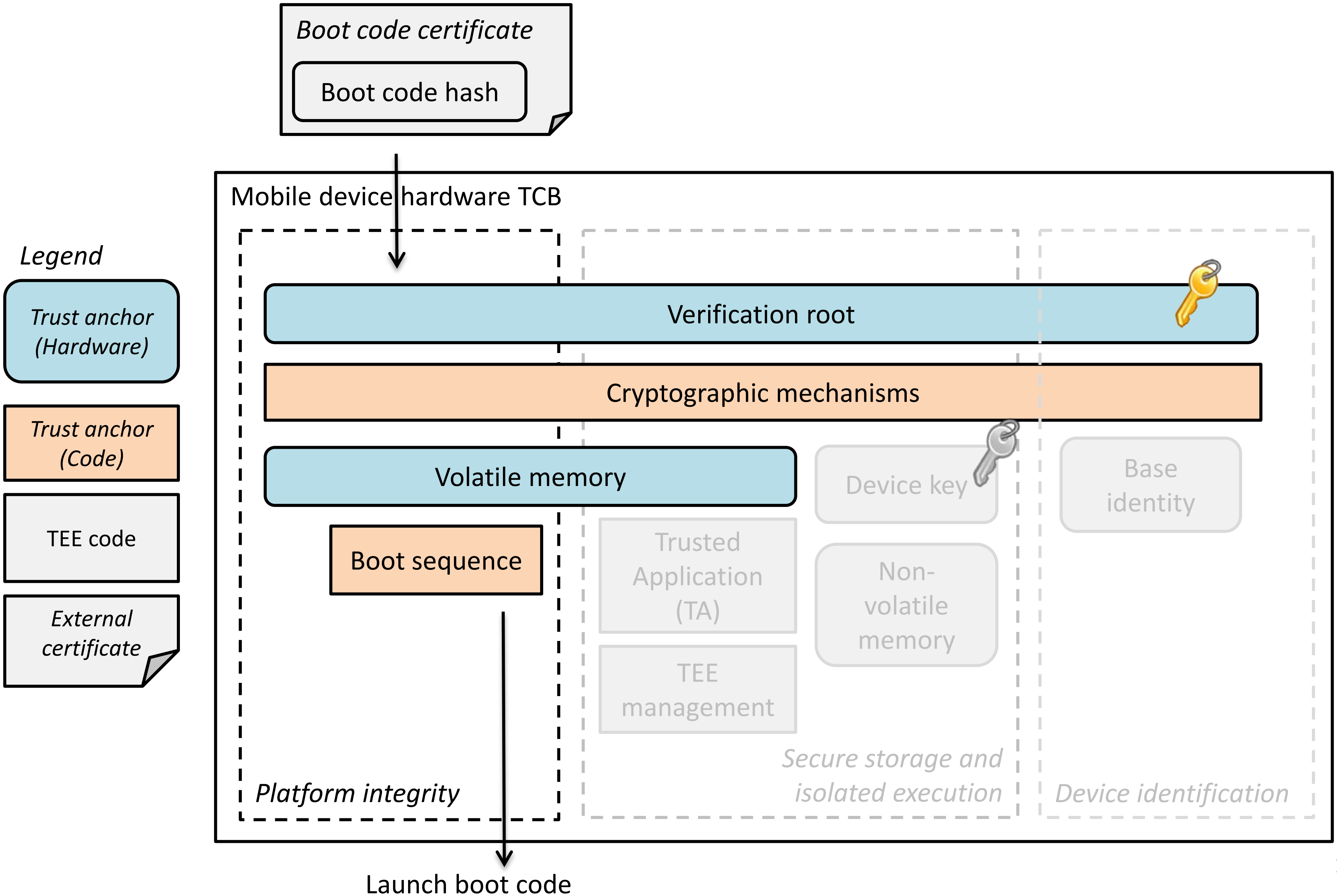


Secure boot

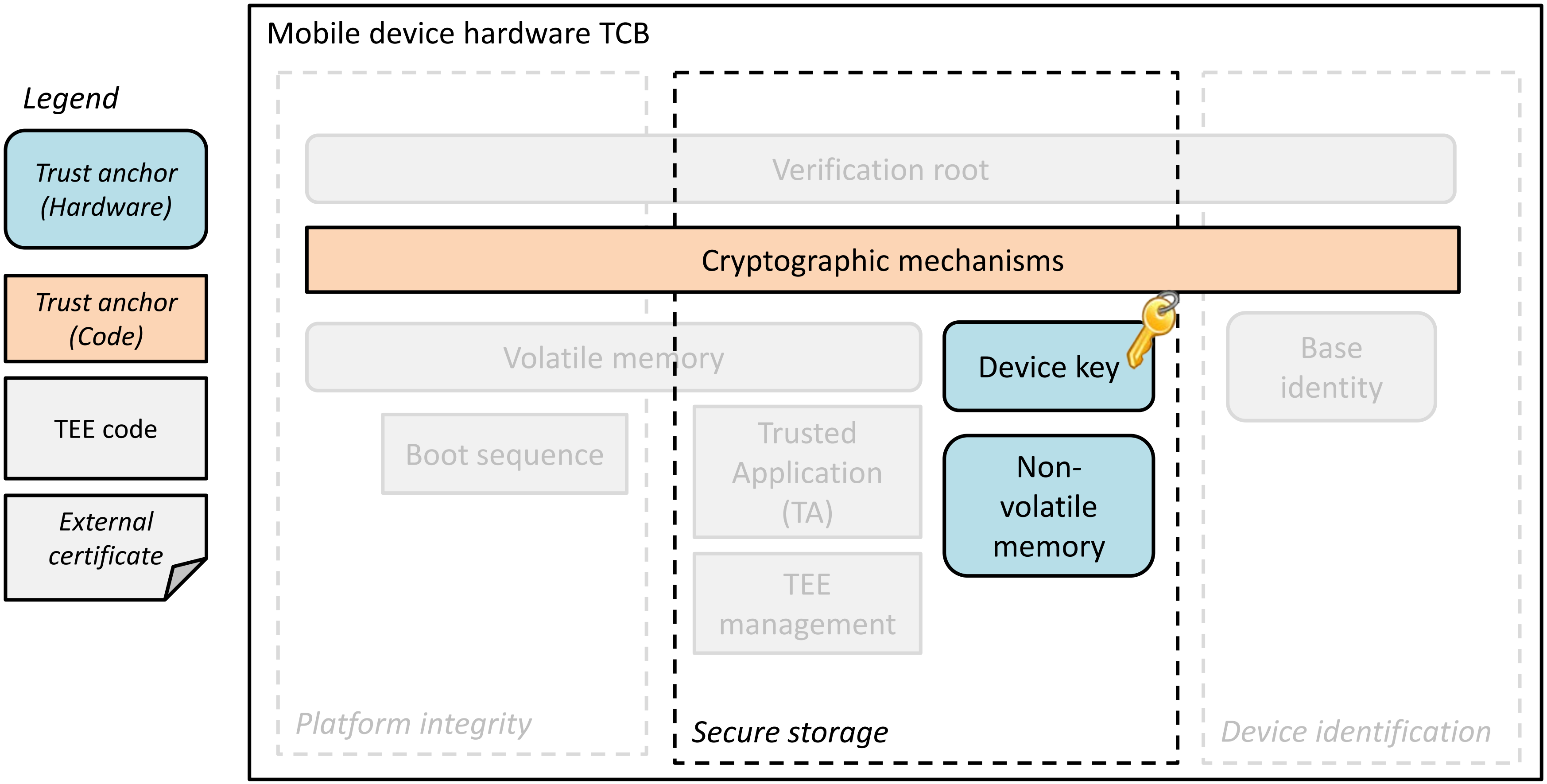


Authenticated boot

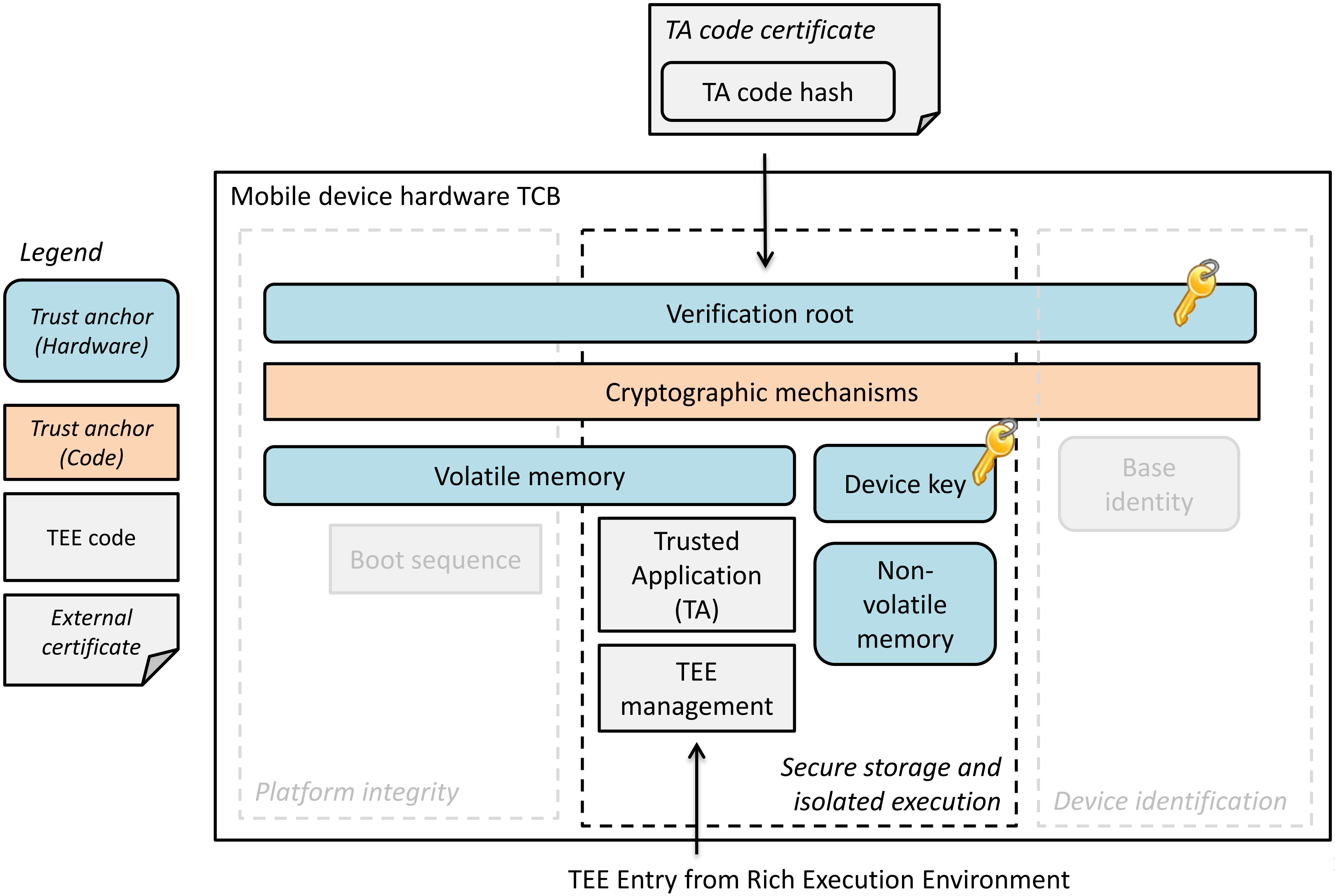
Platform integrity



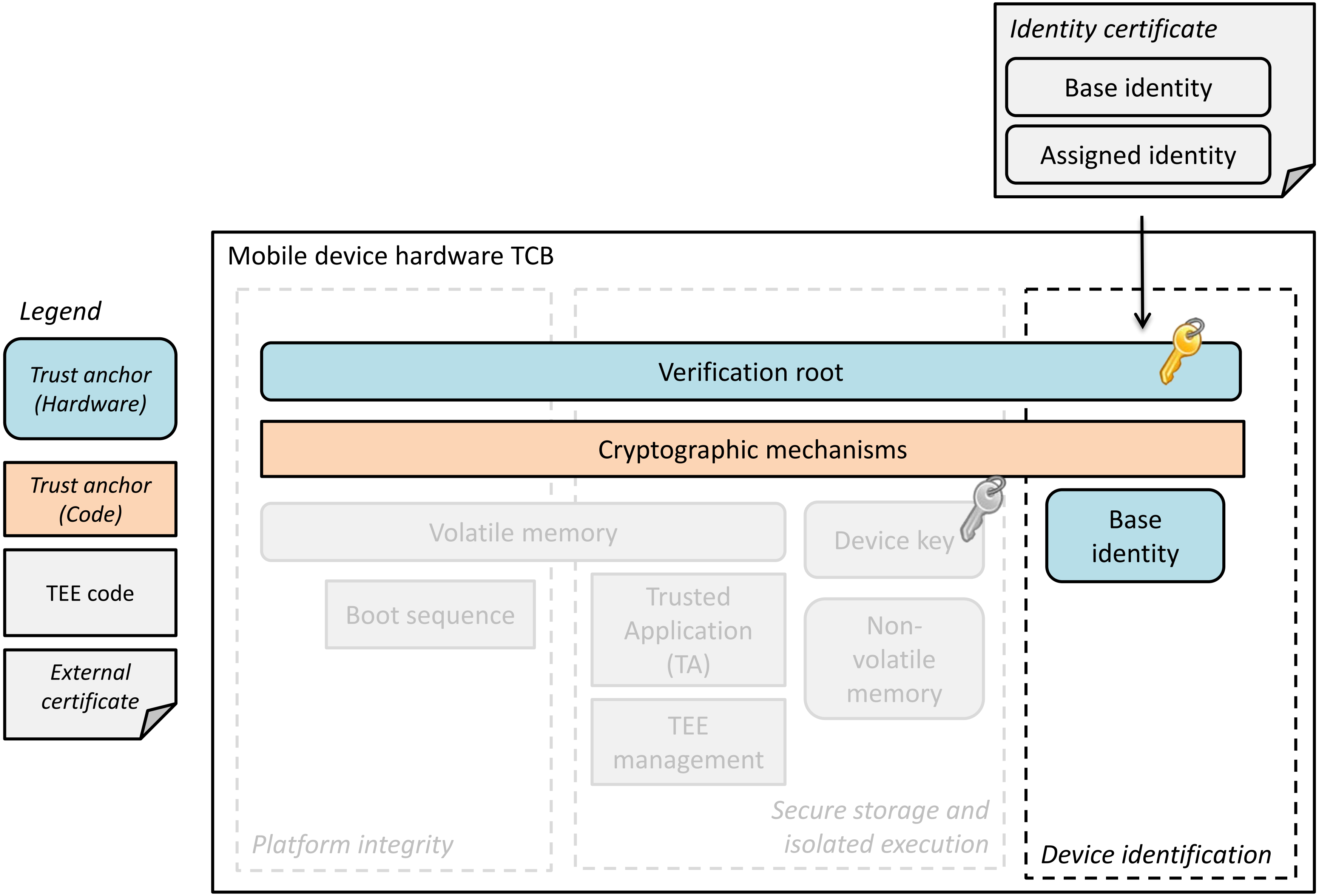
Secure storage



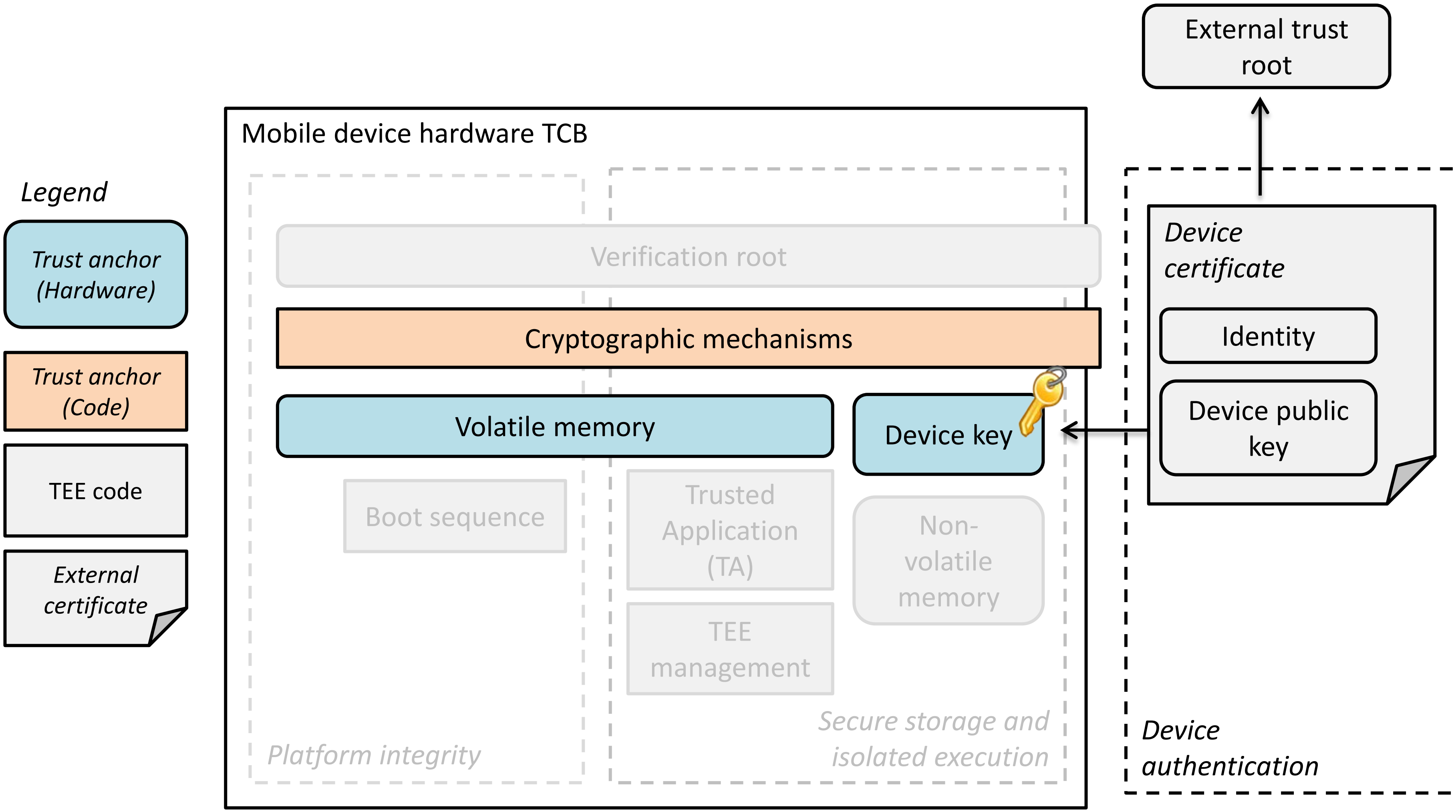
Isolated execution



Device identification

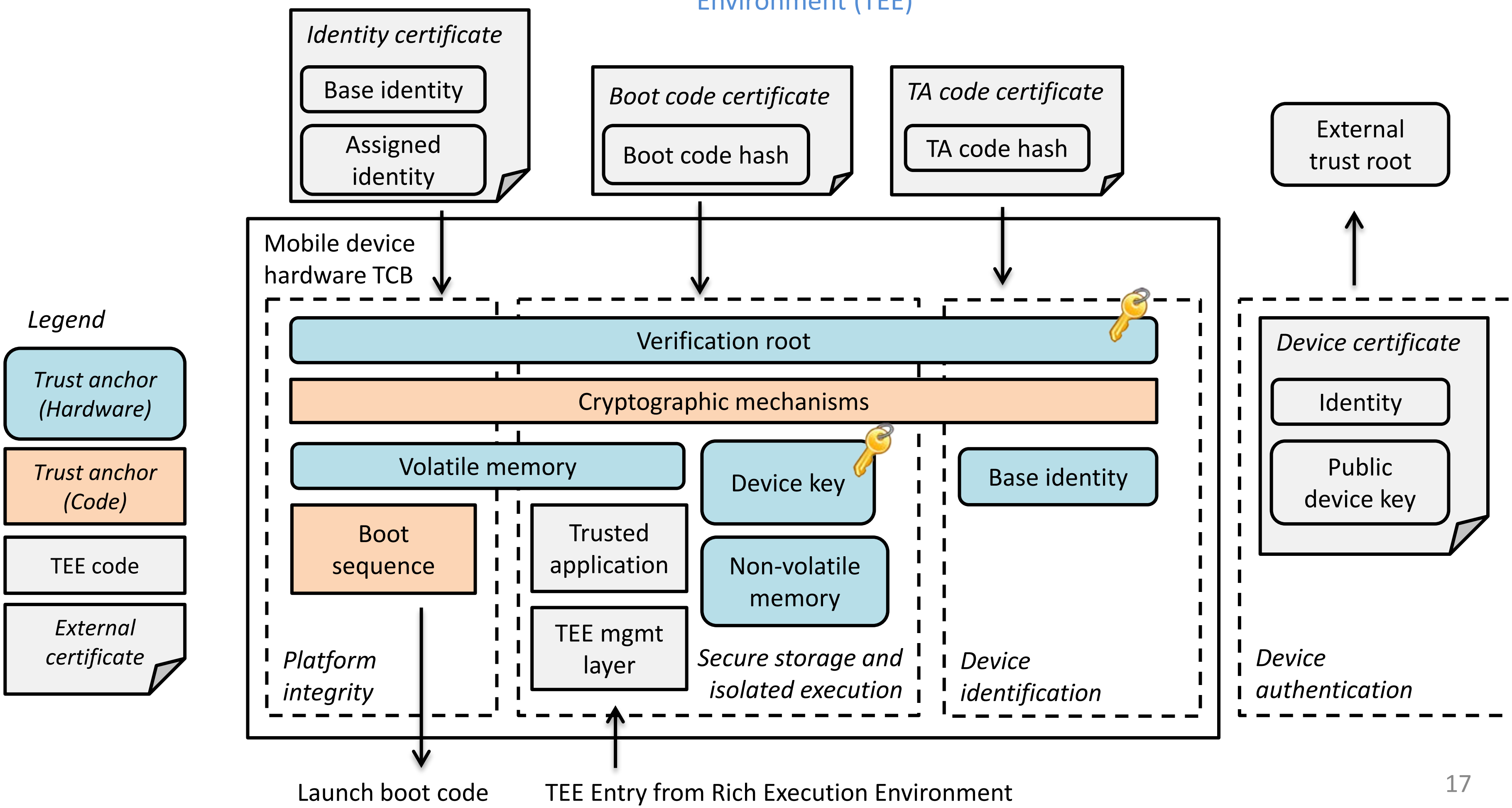


Device authentication (and remote attestation)

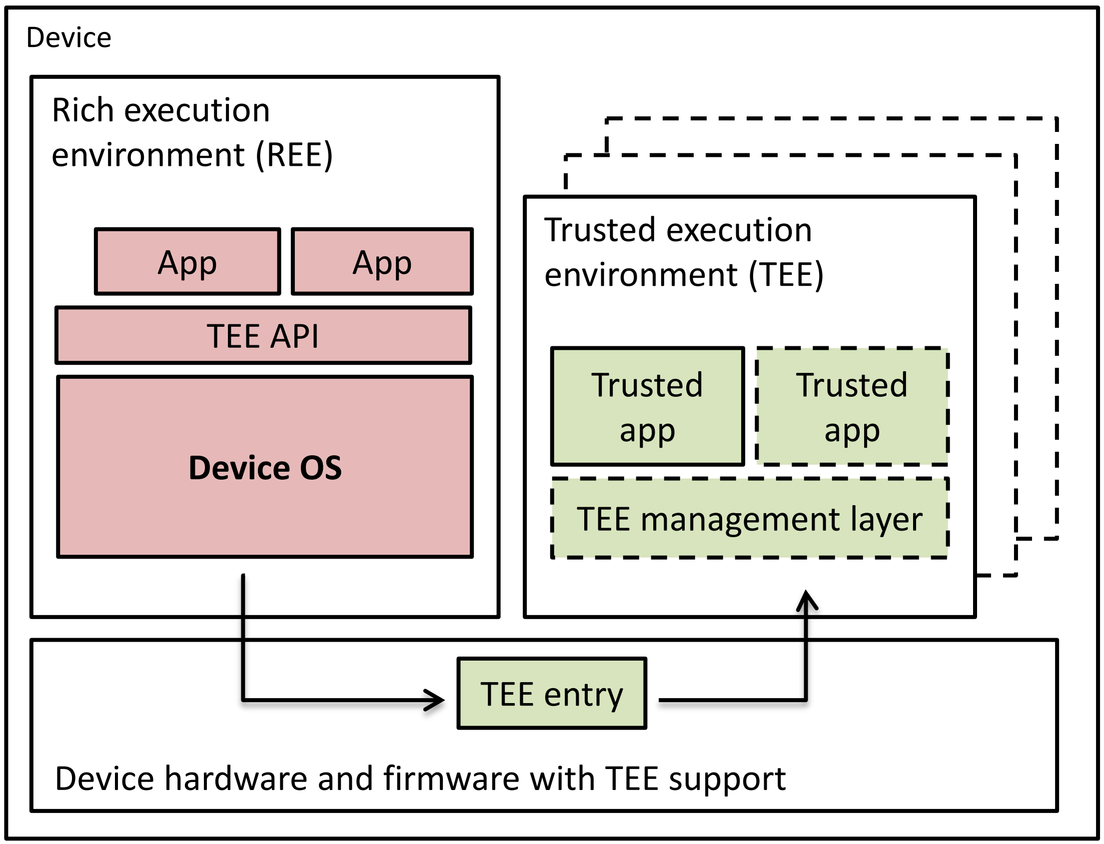


Hardware security mechanisms (recap)

1. Platform integrity
   - Secure boot
   - Authenticated boot
2. Secure storage
3. Isolated execution
   - Trusted Execution Environment (TEE)
4. Device identification
5. Device authentication
   - Remote attestation



TEE system architecture



Architectures with single TEE

- ARM TrustZone
- TI M-Shield
- Smart card
- Crypto co-processor
- TPM

Architectures with multiple TEEs

- Intel SGX
- TPM (and "Late Launch")
- Hypervisor

Figure adapted from: Global Platform. [TEE system architecture](). 2011.

TEE hardware realization alternatives

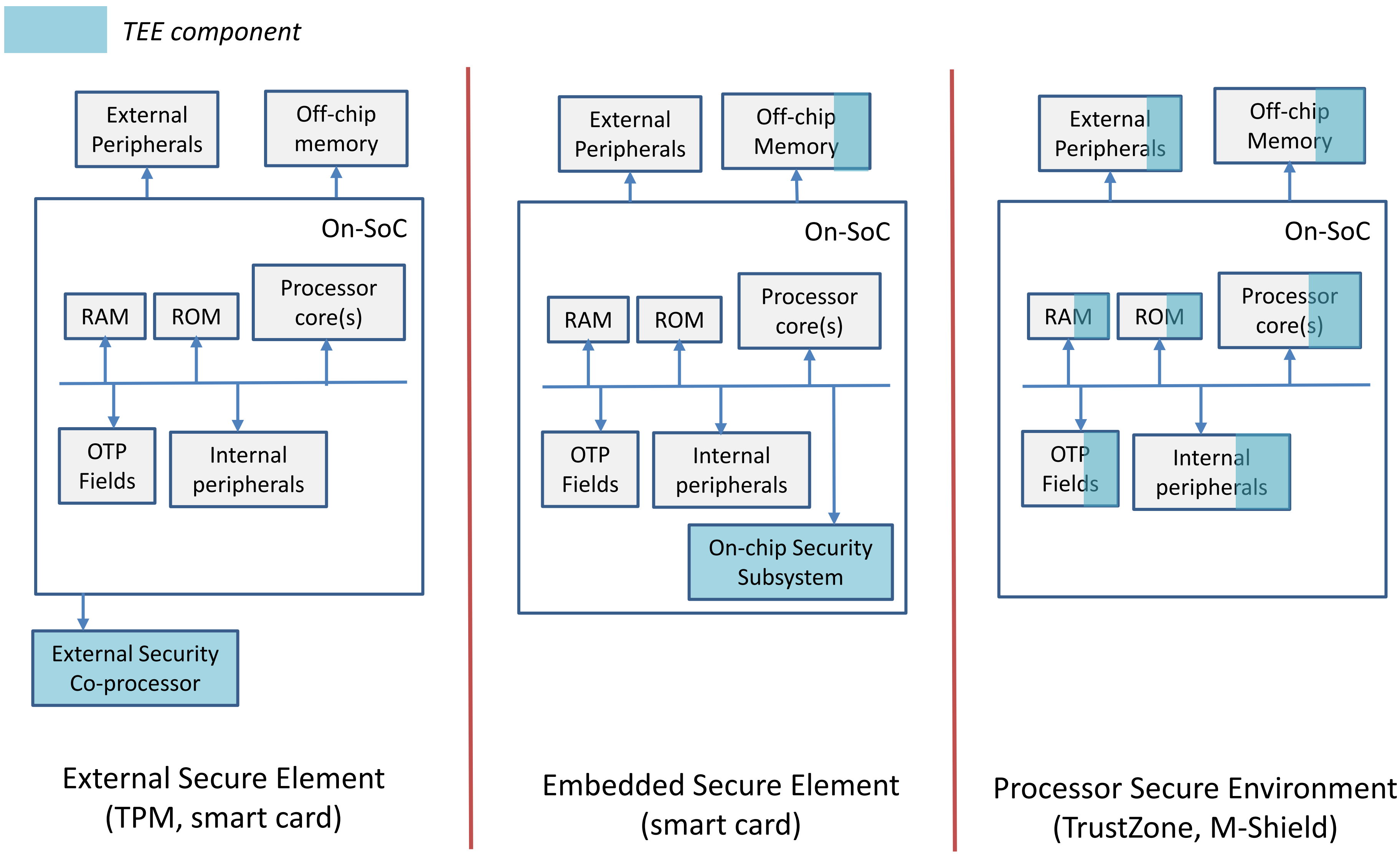
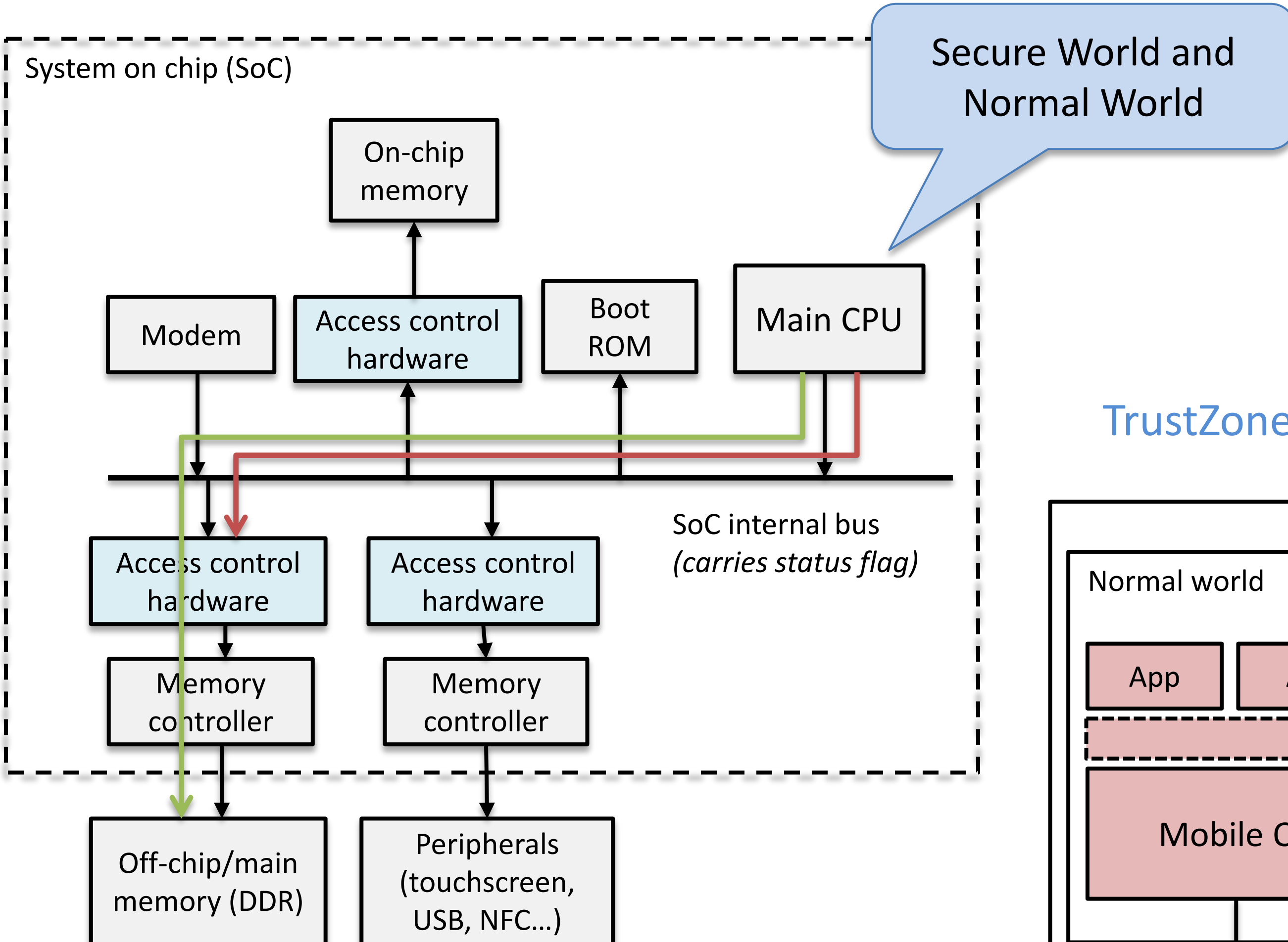


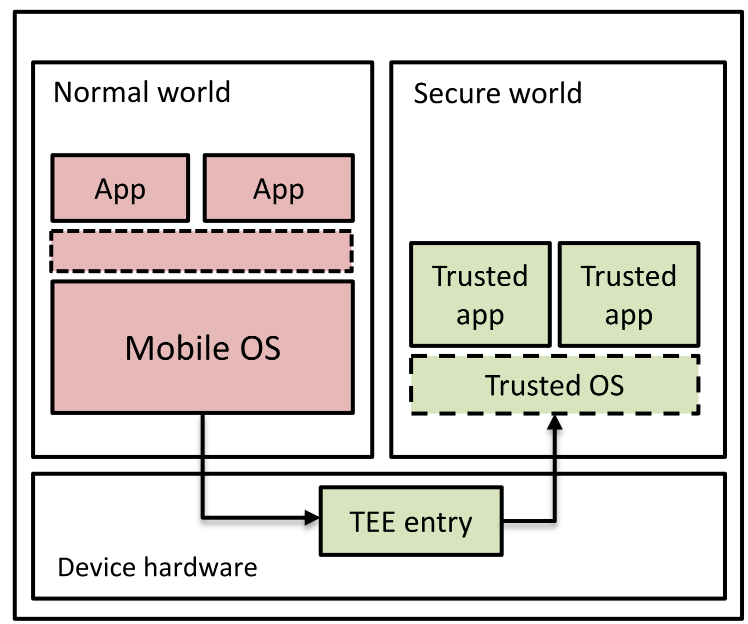
Figure adapted from: Global Platform. [TEE system architecture](). 2011.

ARM TrustZone architecture

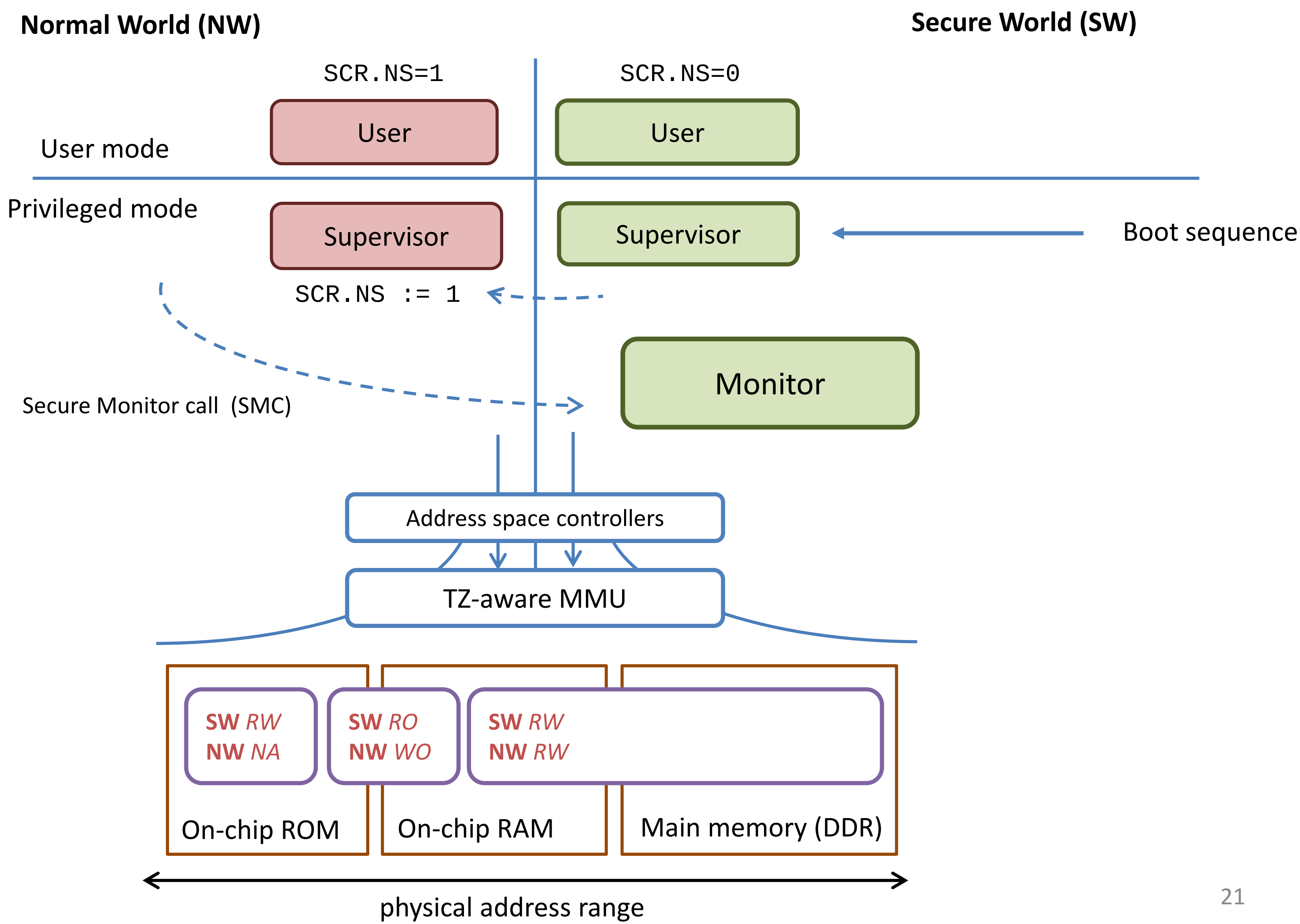


TrustZone hardware architecture

TrustZone system architecture



TrustZone overview



TrustZone example (1/2)

1. Boot begins in Secure World Supervisor mode (set access control)

Boot vector → Secure World Supervisor

2. Copy code and keys from on-chip ROM to on-chip RAM

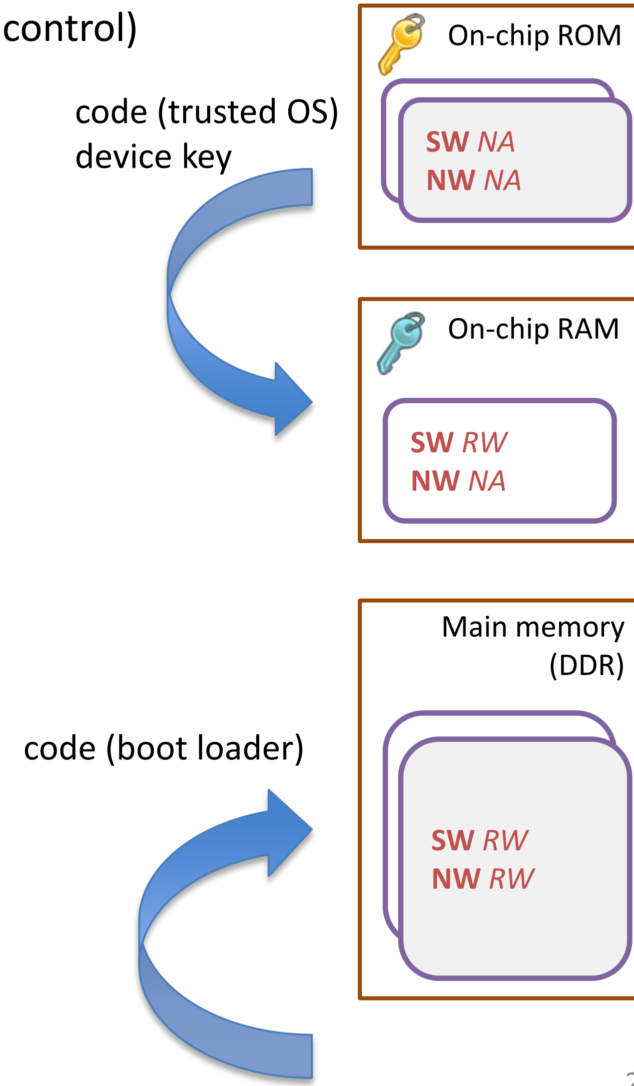
Secure World Supervisor

3. Configure address controller (protect on-chip memory)

Secure World Supervisor

4. Prepare for Normal World boot

Secure World Supervisor



TrustZone example (2/2)

5. Jump to Normal World Supervisor for traditional boot

Secure World Supervisor → Normal World Supervisor

An ordinary boot follows: Set up MMU, load OS, drivers…

6. Set up trusted application execution

Normal World User

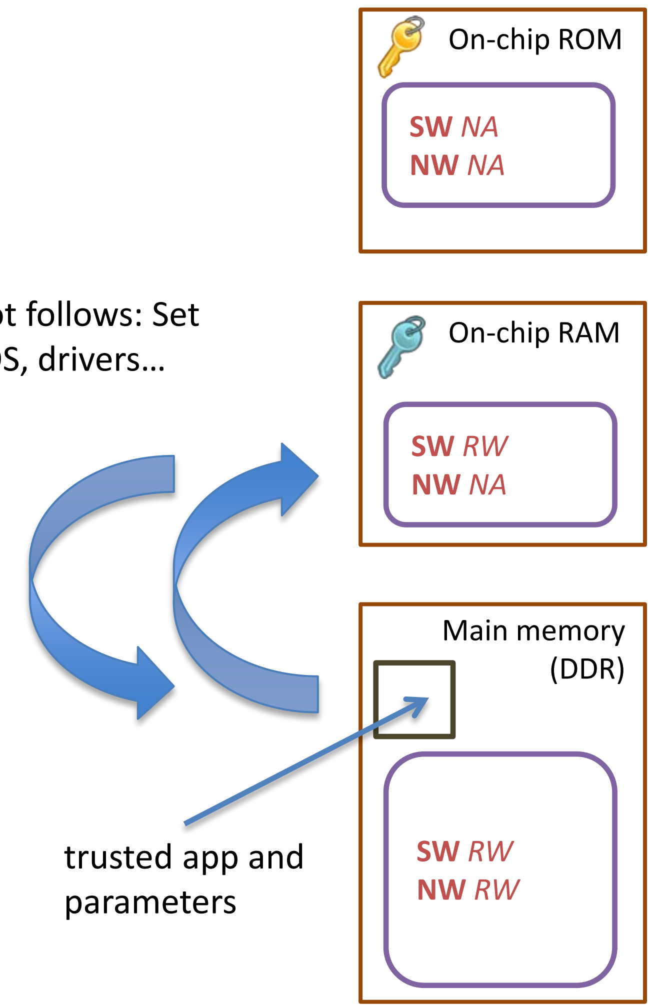
Supervisor

7. Execute trusted application

Normal World Supervisor

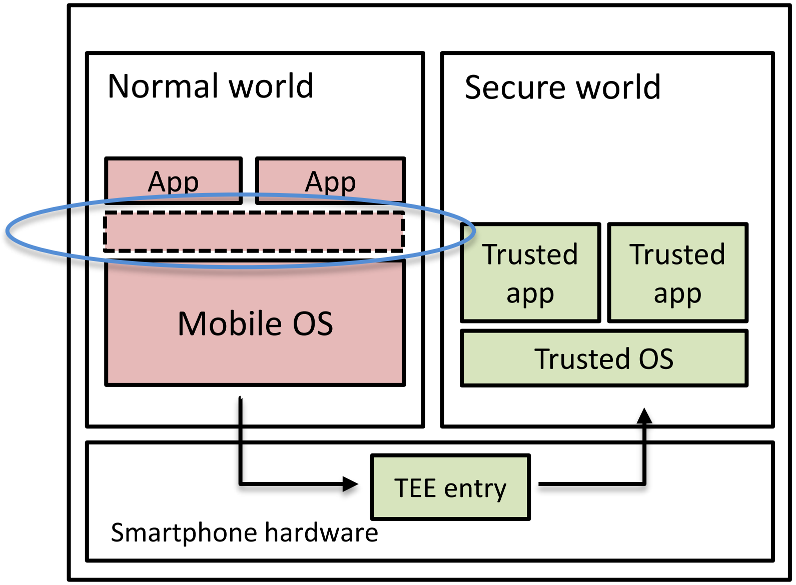
Secure World Monitor

SMC, NS→0



Mobile TEE deployment

- TrustZone support available in majority of current smartphones
- Mainly used for manufacturer internal purposes
  - DRM, Subsidy lock…
- *Third-party APIs emerging…*



Mobile hardware security APIs

APPLICATION DEVELOPMENT

Mobile hardware security APIs

1. Standardized key stores:

JSR 177

PKCS #11

2. Proprietary hardware key stores:

iOS Key Store

Android Key Store

3. Programmable TEE "credential platforms":

On-board Credentials

Trustonic TEE API

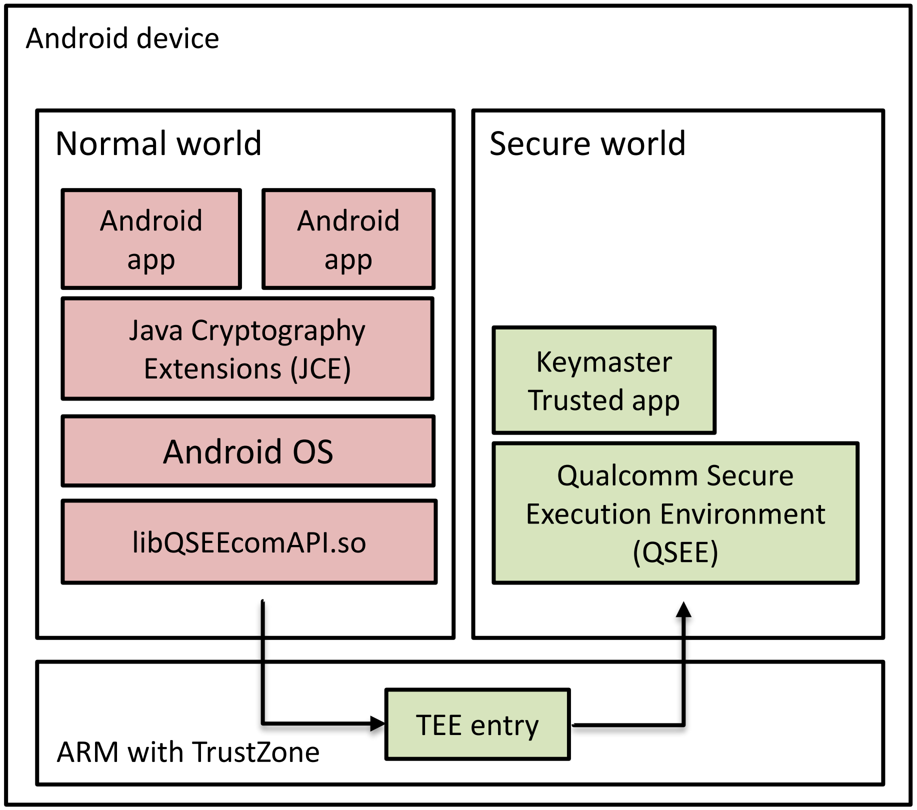
Android Key Store API

Android Key Store example

```
// create RSA key pair
Context ctx;
KeyPairGeneratorSpec spec = new KeyPairGeneratorSpec.Builder(ctx);
spec.setAlias(”key1")
…
spec.build();

KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
gen.initialize(spec);
KeyPair kp = gen.generateKeyPair();


// use private key for signing
AndroidRsaEngine rsa = new AndroidRsaEngine("key1", true);
PSSSigner signer = new PSSSigner(rsa, …);
signer.init(true, …);
signer.update(signedData, 0, signedData.length);
byte[] signature = signer.generateSignature();
```

Elenkov. Credential storage enhancements in Android 4.3. 2013.

Android Key Store implementation



Selected devices

- Android 4.3
- Nexus 4, Nexus 7

Keymaster operations

- GENERATE_KEYPAIR
- IMPORT_KEYPAIR
- SIGN_DATA
- VERIFY_DATA

Persistent storage on Normal World

Elenkov. Credential storage enhancements in Android 4.3. 2013.

Android Key Store

- Available operations
  - Signatures
  - Encryption/decryption
- Developers cannot utilize programmability of mobile TEEs
  - Not possible to run arbitrary trusted applications
- Different API abstraction and architecture needed…

On-board Credentials goal

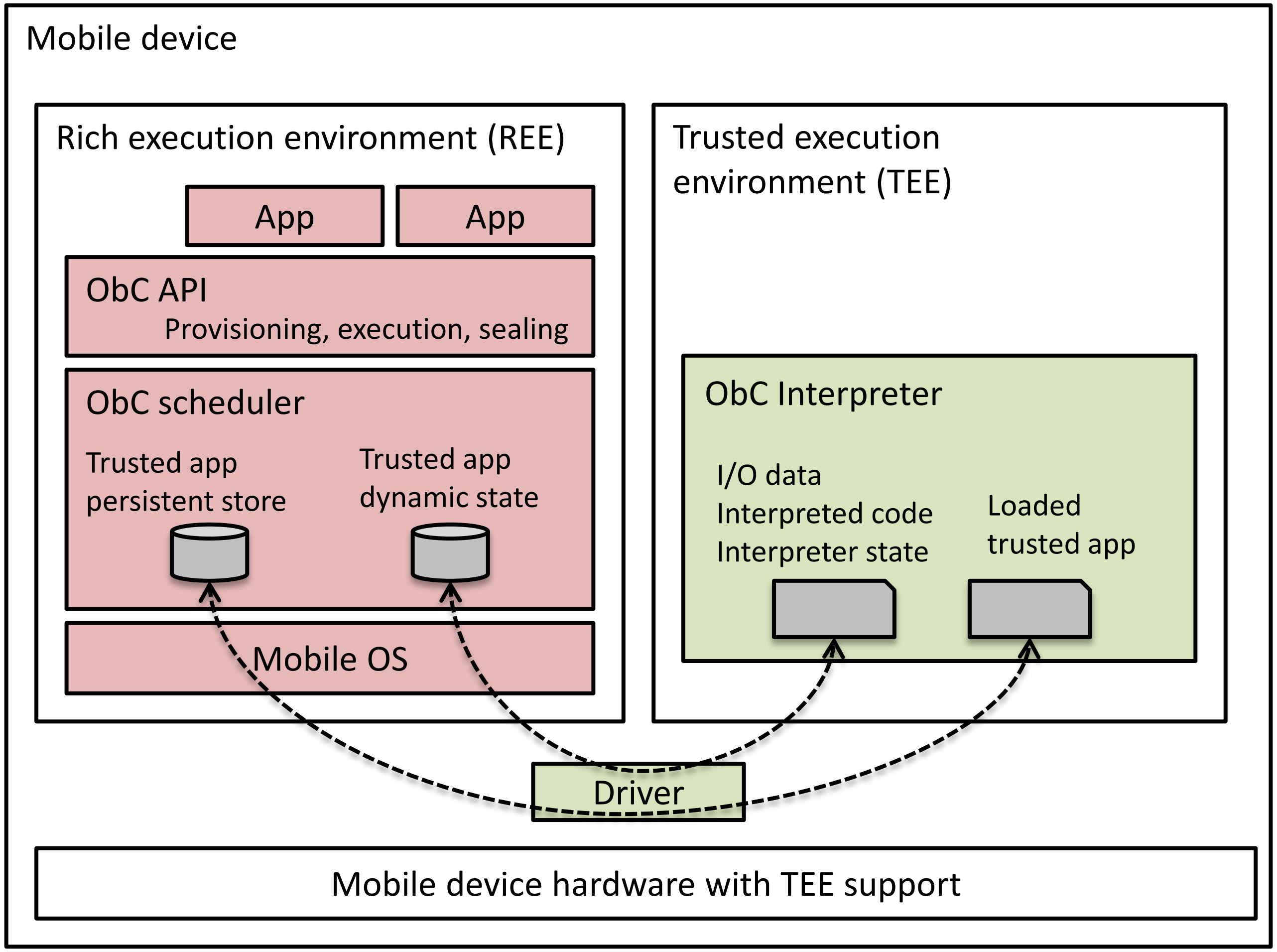
An open credential platform that enables existing mobile TEEs

Secure yet inexpensive

Design constraints:

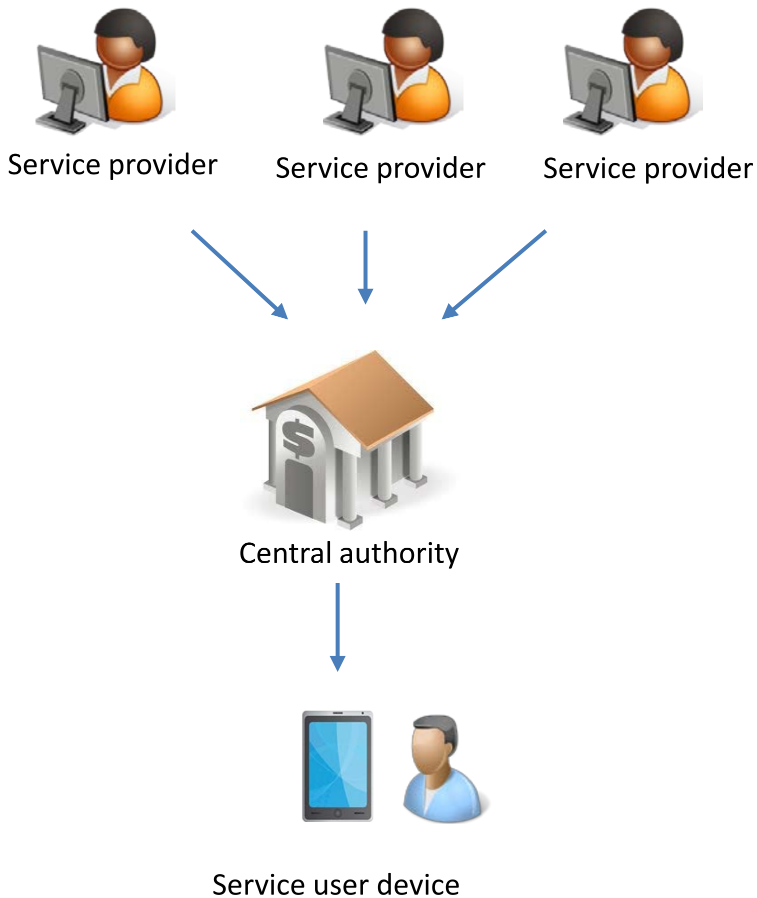
- Open provisioning model
- Limited secure (on-chip) secure memory
- No access control architecture within TEE

On-board Credentials (ObC) architecture

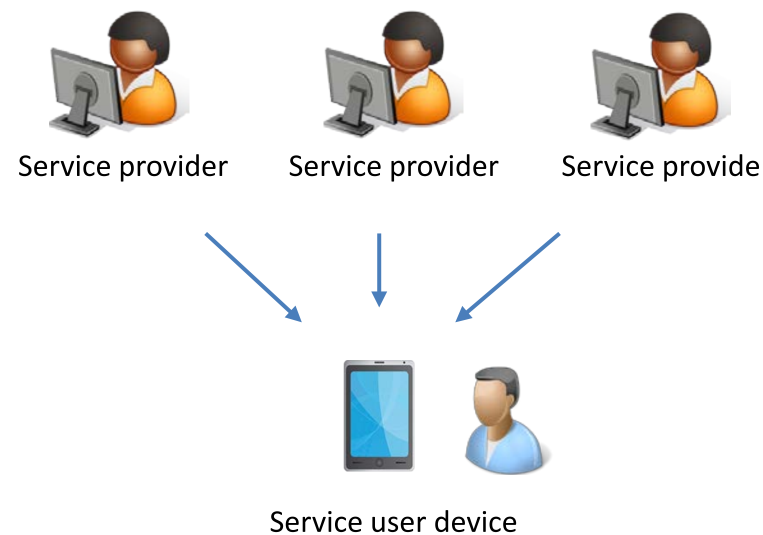


Ekberg. [Securing Software Architectures for Trusted Processor Environments](). Dissertation, Aalto University 2013.
Kostiainen. [On-board Credentials: An Open Credential Platform for Mobile Devices](). Dissertation, Aalto University 2012.

Centralized provisioning vs. open provisioning

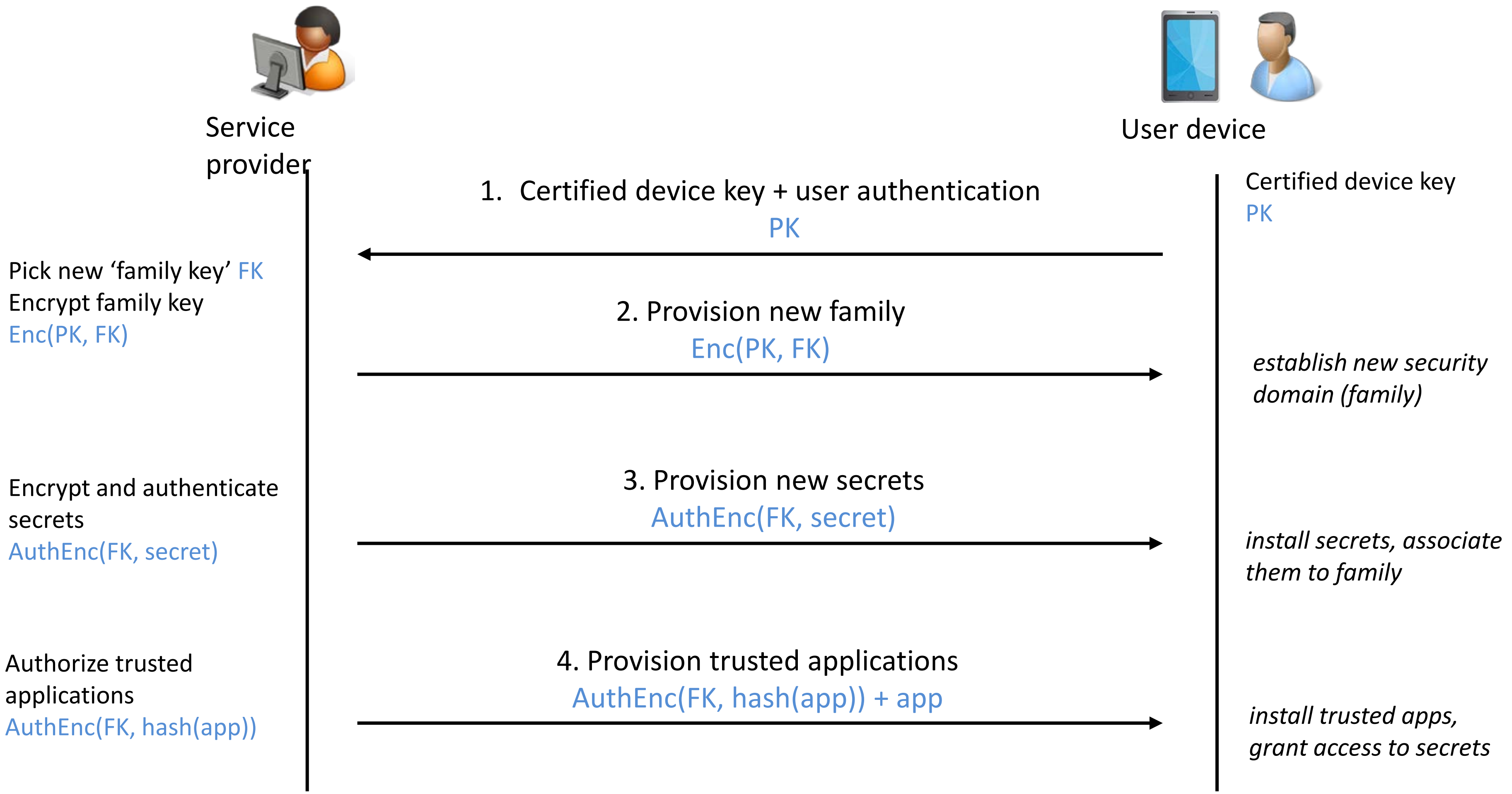


Centralized provisioning
(smart card, Trustonic)



Open provisioning
(On-board Credentials)

Open provisioning model



Principle of same-origin policy

Kostiainen, Ekberg, Asokan and Rantala. [On-board Credentials with Open Provisioning](). ASIACCS 2009.

On-board Credentials development

Service provider

- Trusted application development
  - BASIC like scripting language
  - Common crypto primitives available (RSA, AES, SHA)
- REE application counterpart
  - Standard smartphone app (Windows Phone)
  - ObC API: provisioning, trusted application execution

ObC counterpart application pseudo code

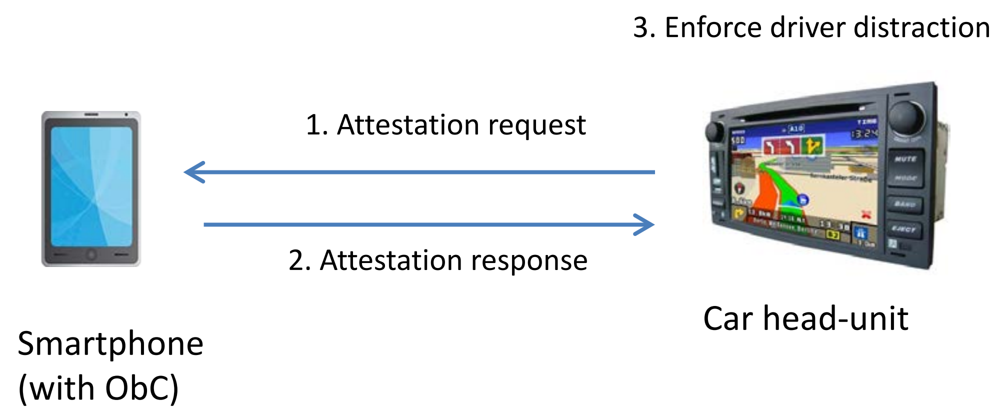
```
// install provisioned credential
secret = obc.InstallSecret(provSecret)
app = obc.InstallCode(provApplication)
credential = obc.CreateCredential(secret,
            app, authData)

// run installed credential
output = obc.RunCredential(credential, input)
```

ObC trusted application extract

```
rem --- Quote operation
if mode == MODE_QUOTE
  read_array(IO_SEALED_RW, 2, pcr_10)
  read_array(IO_PLAIN_RW, 3, ext_nonce)

rem --- Create TPM_PCR_COMPOSITE
pcr_composite[0] = 0x0002    rem --- sizeOfSelect=2
pcr_composite[1] = 0x0004    rem --- PCR 10 selected (00 04)
pcr_composite[2] = 0x0000    rem --- PCR selection size 20
pcr_composite[3] = 0x0014
append_array(pcr_composite, pcr_10)
sha1(composite_hash, pcr_composite)

rem --- Create TPM_QUOTE_INFO
quote_info[0] = 0x0101    rem --- version (major/minor)
quote_info[1] = 0x0000    rem --- (revMajor/Minor)
quote_info[2] = 0x5155    rem --- fixed (`Q' and `U')
quote_info[3] = 0x4F54    rem --- fixed (`O' and `T')

append_array(quote_info, composite_hash)
append_array(quote_info, ext_nonce)
write_array(IO_PLAIN_RW, 1, pcr_composite)

rem --- Hash QUOTE_INFO for MirrorLink PA signing
sha1(quote_hash, quote_info)
write_array(IO_PLAIN_RW, 2, quote_hash)
```

Example application: MirrorLink attestation

- MirrorLink system enables smartphone services in automotive context
- Car head-unit needs to enforce driver distraction regulations
- Attestation protocol
  - Defined using TPM structures (part of MirrorLink standard)
  - Implemented as On-board Credentials trusted application (deployed to Nokia devices)



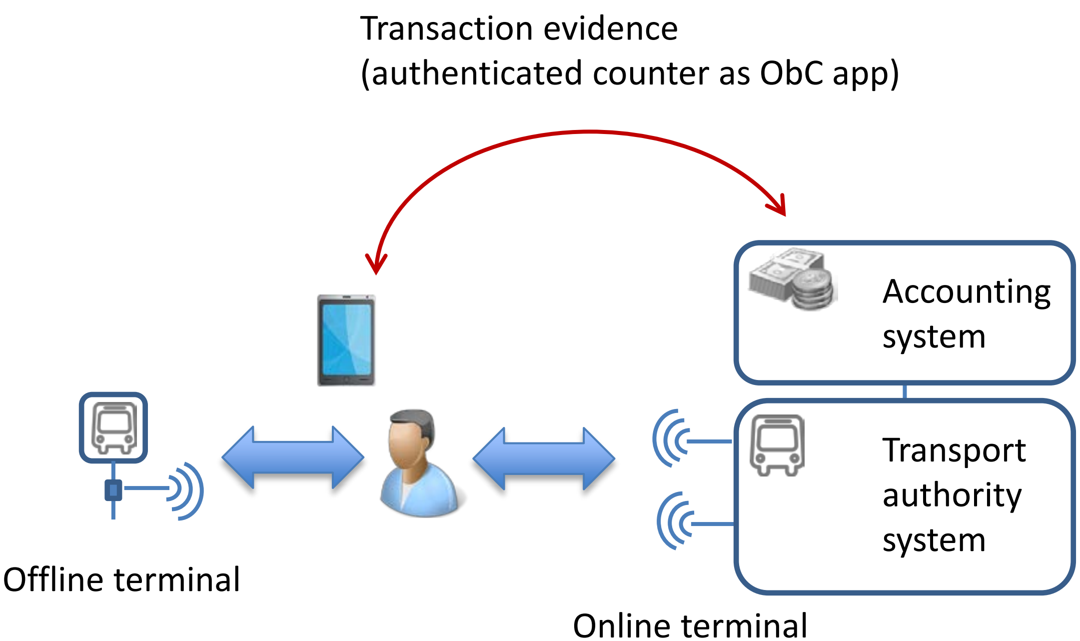
http://www.mirrorlink.com

CARCONNECTIVITY consortium

Kostiainen, Asokan and Ekberg. [Practical Property-Based Attestation on Mobile Devices](). TRUST 2011.

Example application: Public transport ticketing

- Mobile ticketing with NFC and TEE
- 110 traveler trial in New York (summer 2012)
  - Implemented as On-board Credentials trusted application
  - Deployed to Nokia devices

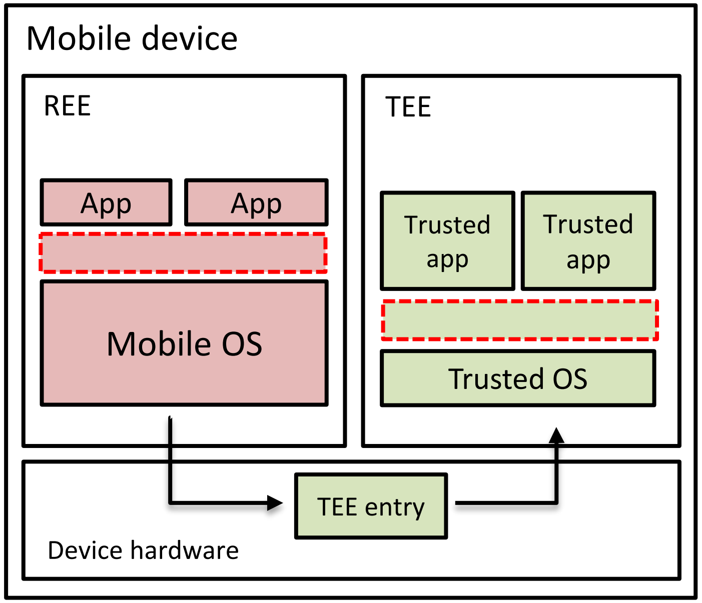


Skip to <tBase

Ekberg and Tamrakar. Tapping and Tripping with NFC. TRUST 2013

Application development summary

- Previously mainly internal purposes
  - DRM, subsidy lock
- Third-party APIs have started to emerge
  - Android KeyStore (TrustZone)
  - Trustonic security API
- Research for open TEEs
  - On-board Credentials with open provisioning
- *Standardization would help developers…*

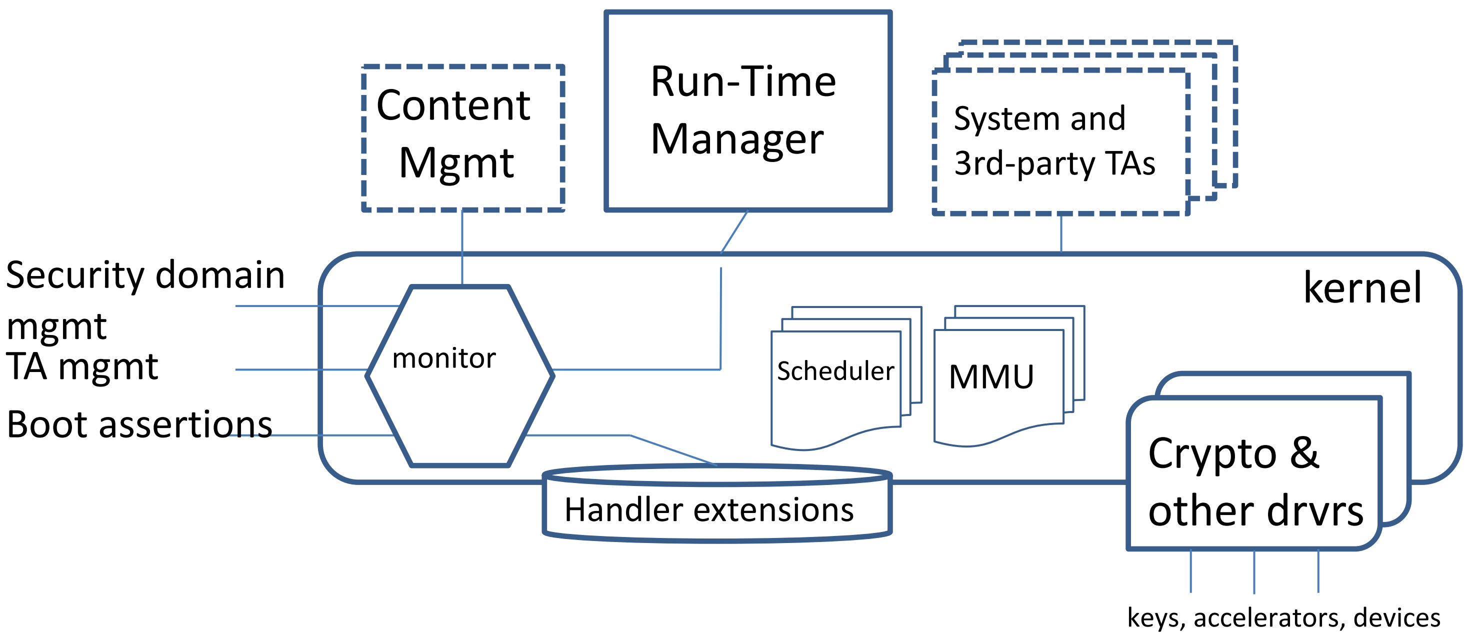


See you in 10 minutes…

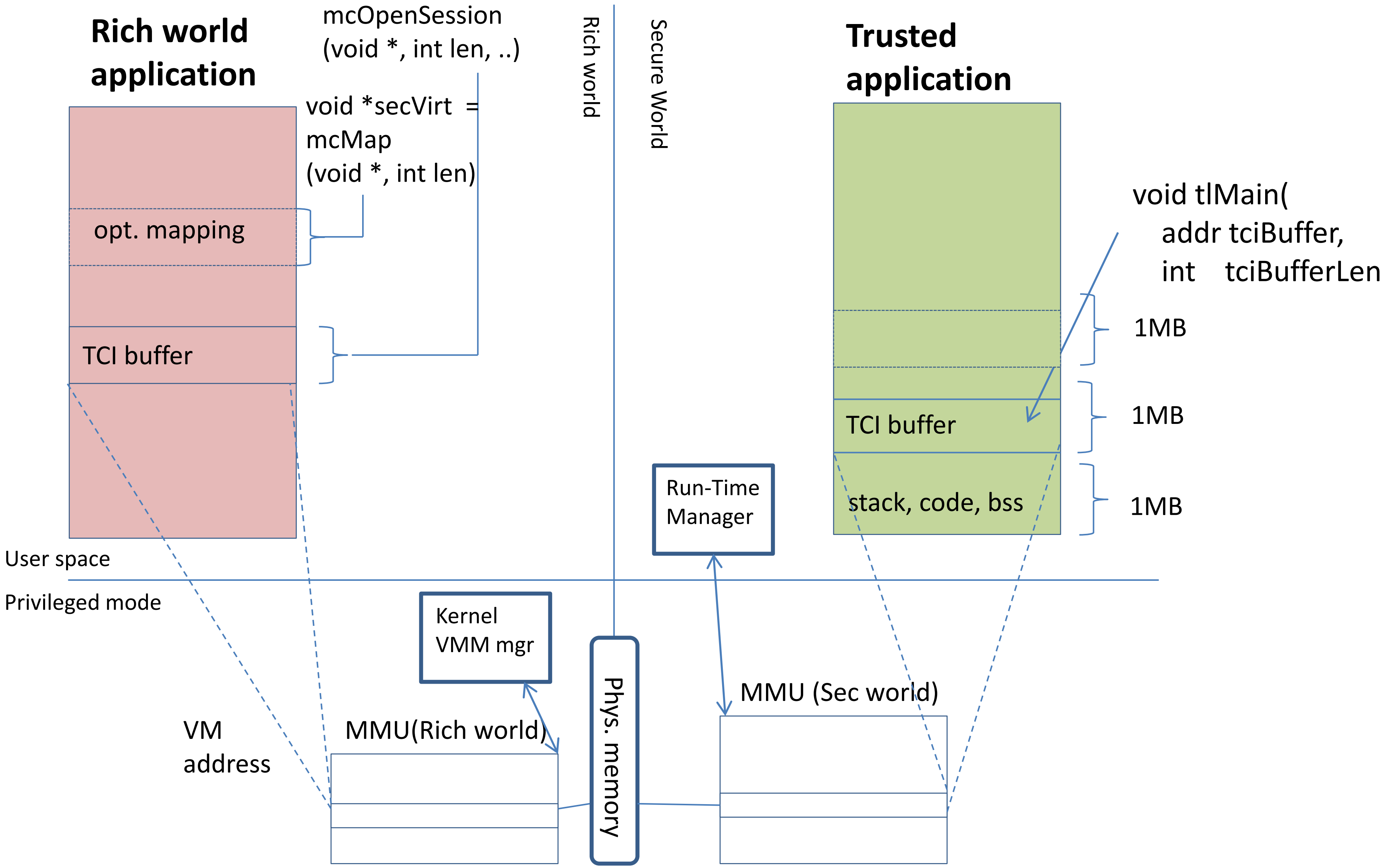
BREAK

Trustonic <t-base TEE

- **L4:** minimized kernel: IPC, scheduling, MMU
- **Run-Time Manager:** Installation, I/O.
- **Crypto driver:** key access, crypto, RNG, secure storage
- Smart-card like provisioning and life-cycle model for TAs
- Global Platform compatibility



<t-base TA invocation



Code Example: Rich World

1. Open connection to TEE

2. Open session
   - provide TA
   - Opt: provide shared mem.

3. *Communicate*

4. Terminate session and connection

```
static TEEC_Result Run (TEEC_Session *session,
                        unsigned char *pData)
{
  TEEC_Result nError;
  TEEC_Operation sOperation;

  memset(&sOperation, 0, sizeof(TEEC_Operation));
  sOperation.paramTypes = TEEC_PARAM_TYPES(
    TEEC_MEMREF_TEMP_INOUT, TEEC_NONE,
    TEEC_NONE, TEEC_NONE);
  sOperation.params[0].tmpref.buffer = pData;
  sOperation.params[0].tmpref.size = 512;

  nError = TEEC_InvokeCommand(session,
    CMD_GENKEY, &sOperation,  NULL);

  return nError;
}
```

#define CMD_GENKEY 1

Code Example: Secure World

1. **Provide handlers for**
   - **instantiation / unload**
   - **session open / close**

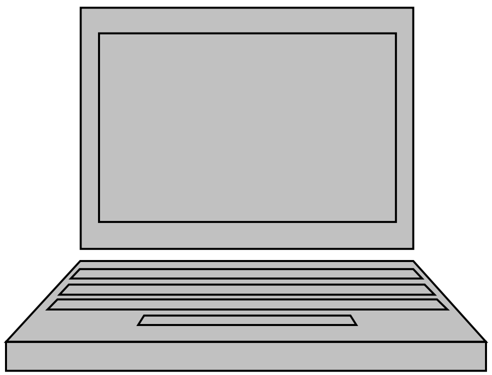
2. **Provide code for**
   - **function that is called**

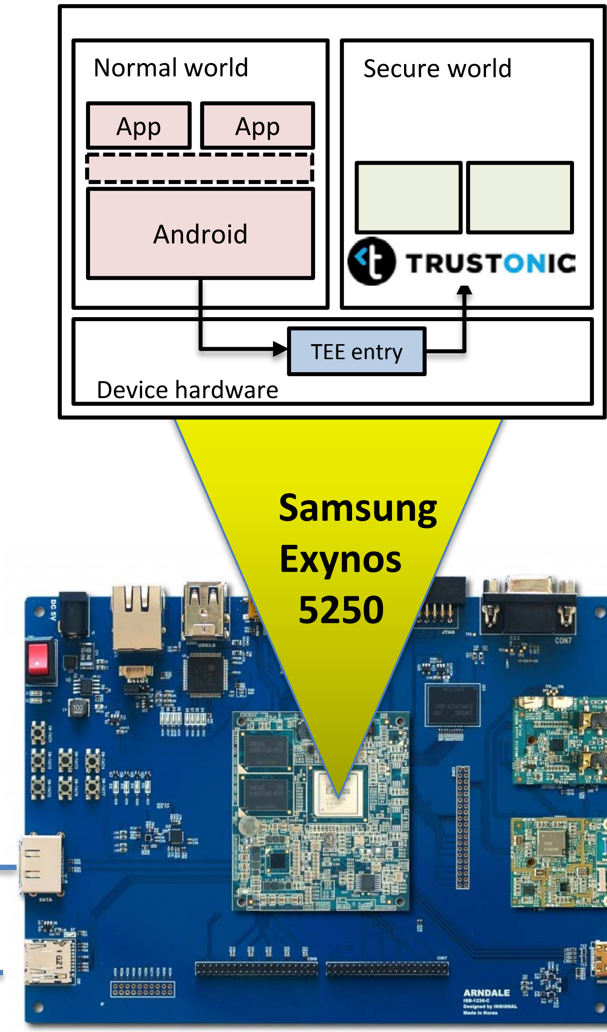
```
TA_InvokeCommandEntryPoint(void* pSessionContext,
  uint32_t nCommandID,
  uint32_t nParamTypes, TEE_Param pParams[4])
{
...
 switch(nCommandID)
 {
   case CMD_GENKEY:
    if (nParamTypes != CMD_GENKEY_PTYPES) {…}
    pInput = pParams[0].memref.buffer;
    size = (uint32_t)pParams[0].memref.size;
    if (TEE_CheckMemoryAccessRights( … ) { … }
   TEE_AllocateTransientObject(TEE_TYPE_RSA_KEYPAIR,
           maxObjectSize, &keyObj))
   TEE_GenerateKey(keyObj, 2048, NULL, 0);
   TEE_GetObjectBufferAttribute(keyObj,
        TEE_ATTR_RSA_MODULUS, …);
   TEE_FreeTransientObject(keyObj);
   return TEE_SUCCESS;
...
```

```
#define CMD_GETKEY 1
```

<tbase demo

- Run a dev-board so that we can see the activity





”Google Nexus 10”

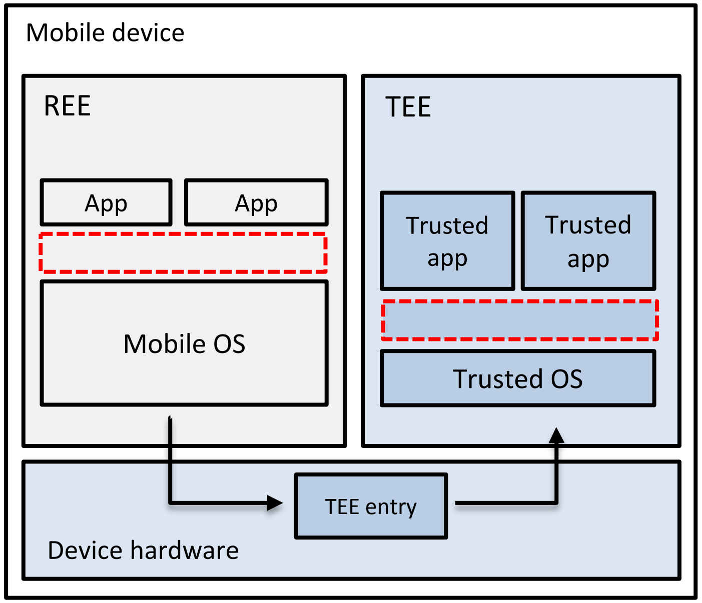
Android command line

Console

www.arndaleboard.org

Application development summary

- Previously mainly internal purposes
  - DRM, subsidy lock
- Third-party APIs have started to emerge
  - Android KeyStore (TrustZone)
  - Trustonic <tbase
- Research for open TEEs
  - On-board Credentials with open provisioning
- *Standardization would help developers…*



Skip to Outline

See you in 10 minutes…

BREAK

Outline

- A look back (10 min)
  - Why mobile devices have TEEs?
- Mobile hardware security (30 min)
  - What constitutes a TEE?
- Application development (30 min)
  - Mobile hardware security APIs + DEMO

Break (10 min)

- Current standardization (60 min)
  - NIST, Global Platform, TPM 2.0
- A look ahead (10 min)
  - Challenges and summary

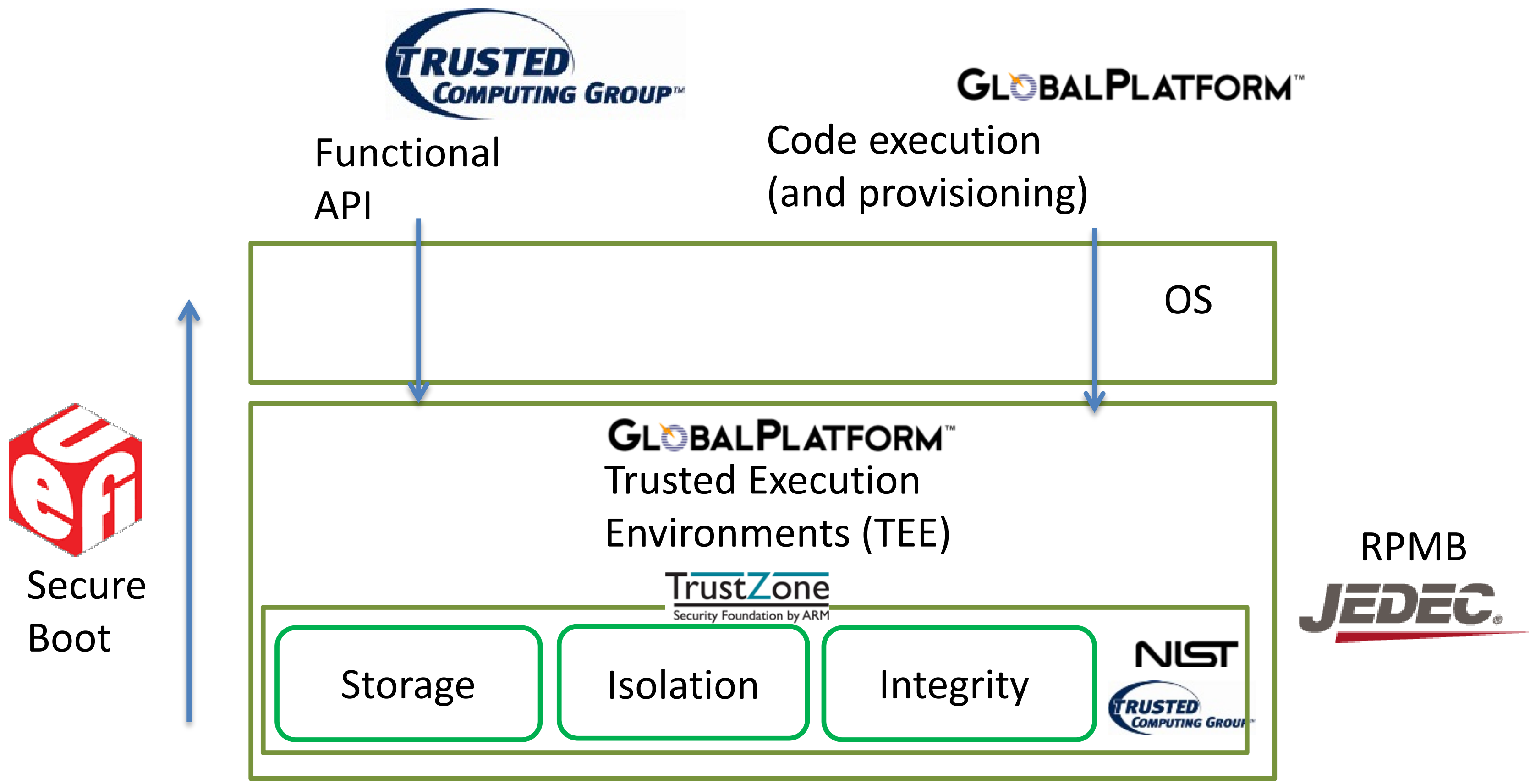
Tutorial based on: Ekberg, Kostiainen and Asokan. The Untapped Potential of Trusted Execution Environments on Mobile Devices. IEEE S&P magazine, 2013.

NIST guidelines, Global Platform, Trusted Computing Group, Jedec

STANDARDIZATION

TEE-related standards and specifications

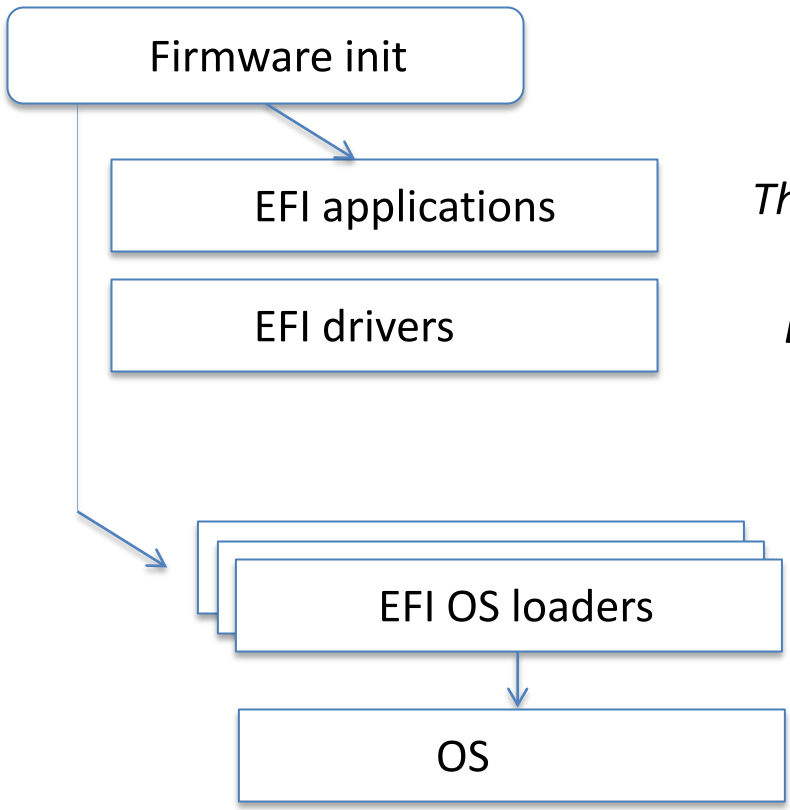
- First versions of standards already out
- Needed for compliance/interoperability
- Enables app developers to leverage TEEs



55

EFI SECURE BOOT

UEFI –boot principle



Things that e.g. sets up the device (like TZ)

Driver firmware setup

Boot loaders

Replacement for BIOS
Secure Boot is an optional feature

[Unified Extensible Firmware Interface Specification]()

[Nyström et al: UEFI Networking and Pre-OS security (2011)]()

UEFI – secure boot

Key management for updates

Platform Firmware Key Storage

→tamper-resistant
→updates governed by platform key

Key Exchange Keys

Platform Key (Pub/Priv)

UEFI – secure boot

Signature Database (s)

→tamper-resistant
(rollback prevention)
→updates governed by keys

Key management
for updates

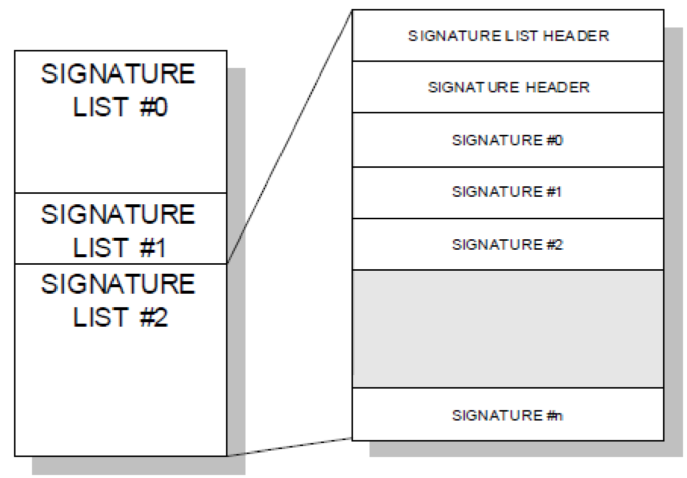
Platform Firmware
Key Storage

→tamper-resistant
→updates governed by
platform key

Keys allowed to
update

Key Exchange Keys

Platform Key (Pub/Priv)



(ref: UEFI spec)

UEFI – secure boot

Signature Database (s)

→tamper-resistant
(rollback prevention)
→updates governed by keys

Key management for update

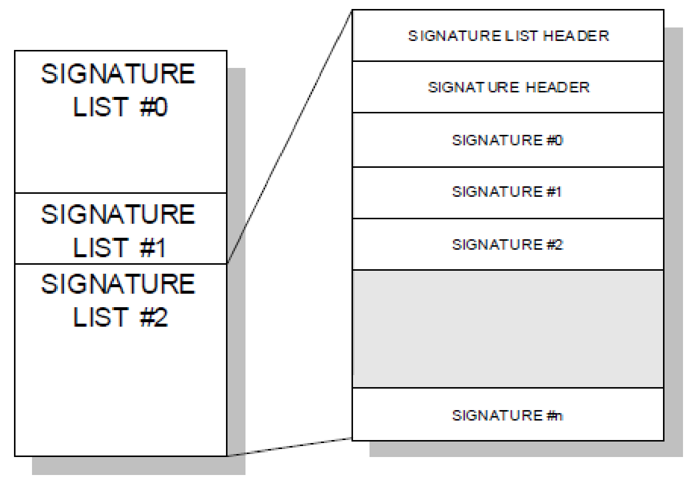
Platform Firmware Key Storage

→tamper-resistant
→updates governed by platform key

Key Exchange Keys

Platform Key (Pub/Priv)

Keys allowed to update



(ref: UEFI spec)

Successful & failed authorizations

Image Information Table
→hash
→name, path
→ Initialized / rejected

White list + *Black list* for database images

ROOTS OF TRUST (HARDWARE ANCHORS)

Guidelines on Hardware-Rooted Security in Mobile Devices (SP800-164, draft)

•

Required security components are

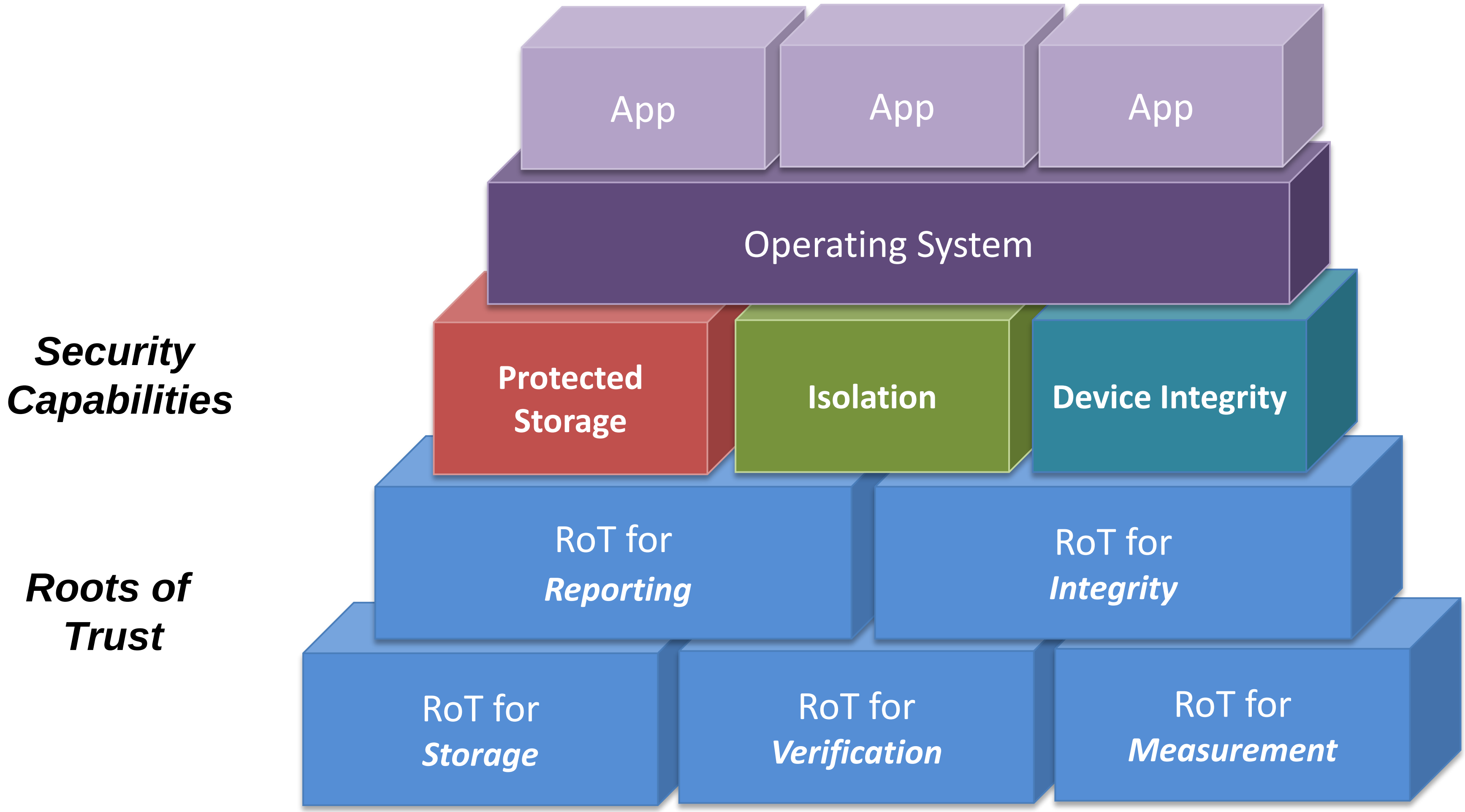
a) **Roots of Trust** (RoT)

b) an **application programming interface** (API) to expose the RoT to the platform

c) a **Policy Enforcement Engine** (PEnE)"

"RoTs are preferably implemented in hardware"

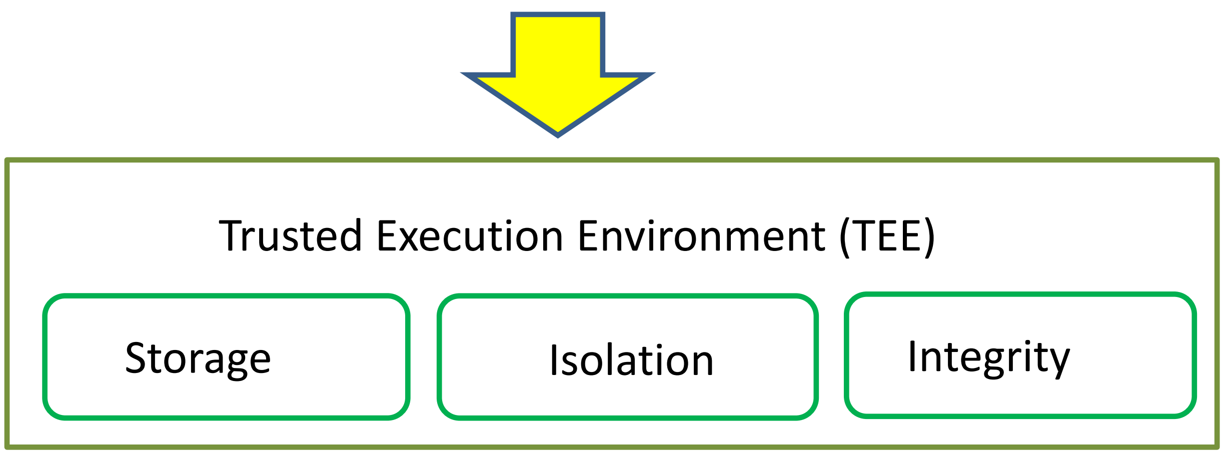
Secure Capabilities built from Roots-of-Trust



Picture: Andrew Regenshield: NIST/Computer Security Division

ARM TrustZone + Secure Boot + Secrets = RoT?

1. Secure boot → **Root of Trust for Verification**
2. Measuring in secure boot → **Root of Trust for Measurement**
3. Device key + code in TZ TEE → **Root of Trust for Reporting**
4. TEE secure memory → **Root of Trust for Integrity**
5. Device key + TEE → Most of **Root of Trust for Storage**. No easy rollback protection.



Specifications: www.globalplatform.org

GLOBALPLATFORM ™

Global Platform

Most of the smart-card based ecosystems around authentication, payment and ticketing make use of Global Platform standards:

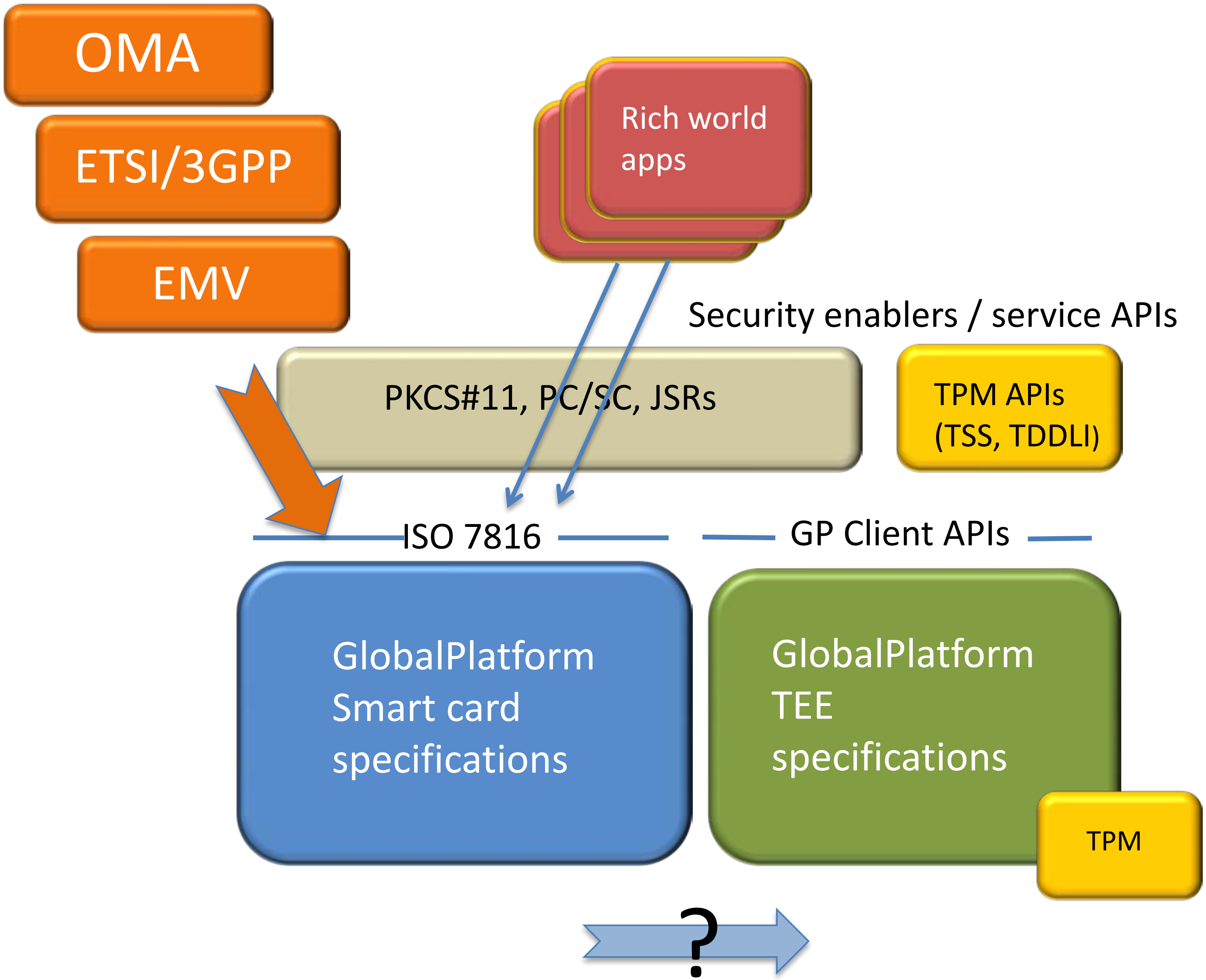
- For card interaction and provisioning protocols
- For reader terminal architecture and certification

The Global Platform Device Committee specifies architecture and interfaces for a **trusted operating system** in a TEE

References:
http://www.globalplatform.org/specificationsdevice.asp

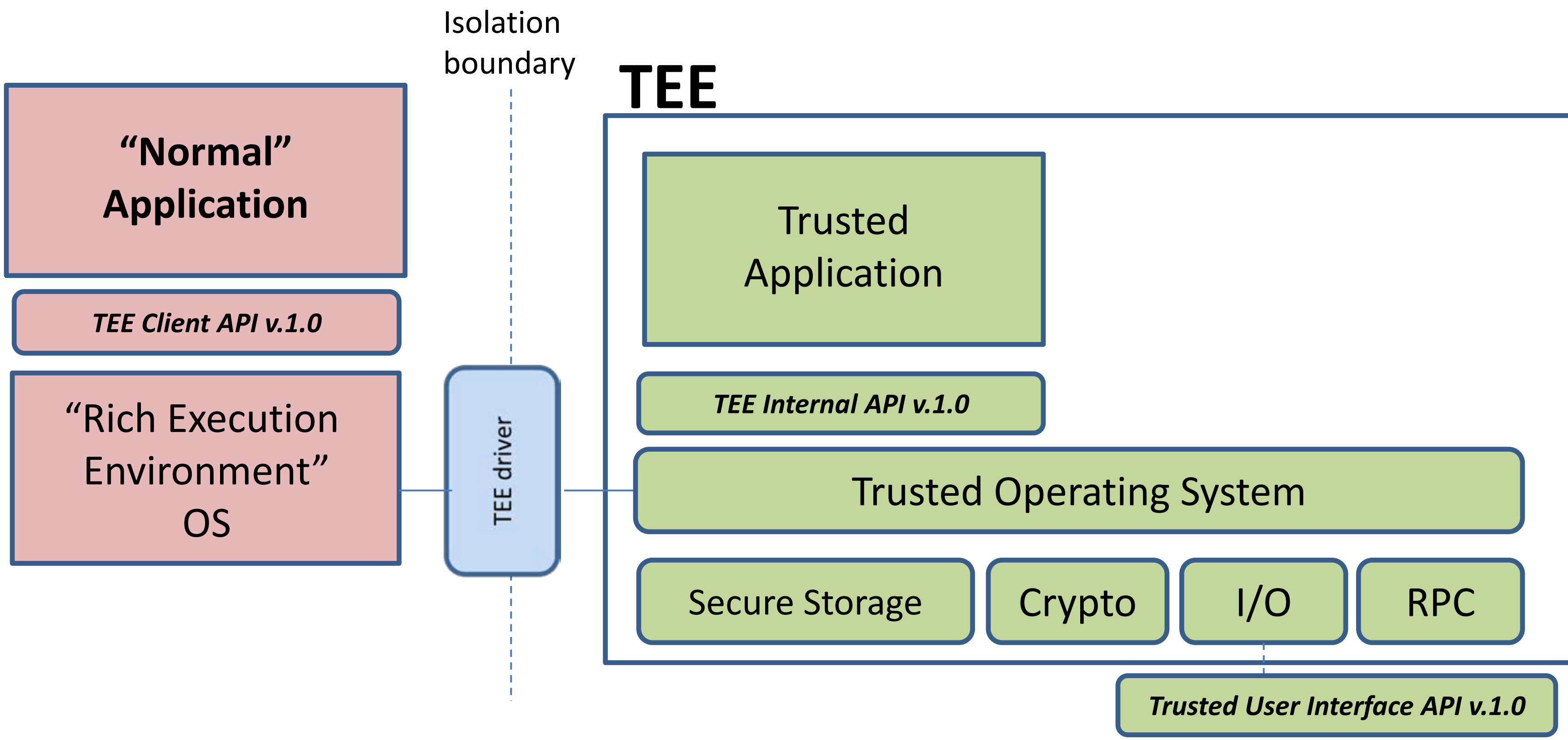
- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal API Specification v1.0
- Trusted User Interface API v 1.0

Global Platform in industry



Global Platform Device Architecture

- API to communicate with the TEE
- System interface library (libc ..) for Trusted Applications with RPC, crypto and necessary I/O functions



Eventually, these APIs may become the reference model for writing code for and interacting with a TEE. Missing pieces still include **provisioning** and **compliance** aspects

Interaction with a TEE (GP) -- caller

(adapted from example in TEE Client API specification)

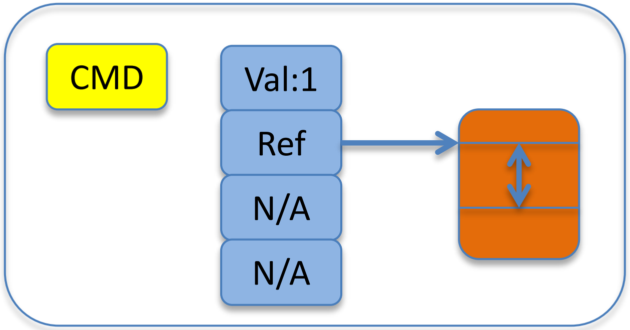
```
result = TEEC_InitializeContext( NULL, &context);
result = TEEC_OpenSession(&context, &session, &cryptoTEEApp, TEEC_LOGIN_USER,
                          NULL,  NULL, NULL);
commsSM.size = 20; commsSM.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
result = TEEC_AllocateSharedMemory(&context, &commsSM);
```

```
// omitted: registration of additional shared memory for in-place encryption of data
operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_MEMREF_PARTIAL_INPUT,
                                        TEEC_NONE, TEEC_NONE);
ivPtr = (uint8_t*)commsSM.buffer; memset(ivPtr, 0, 16);  // Set input (IV)
operation.params[0].value.a = 1;                         // Set input (key handle=1)
operation.params[1].memref.parent = &commsSM;
operation.params[1].memref.offset = 0;
operation.params[1].memref.size = 20;
```

Setting up parameters

```
result = TEEC_InvokeCommand(&session, CMD_ENCRYPT_INIT, &operation, NULL);
```

Parameters:



Interaction with a TEE (GP) -- callee

Mandatory handler functions:

Constructor / Destructor

TA_CreateEntryPoint(void); / **TA_DestroyEntryPoint(void)**;

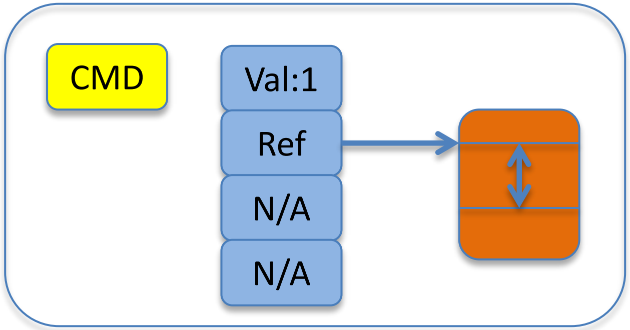
TA_OpenSessionEntryPoint(uint32_t param_types, TEE_Param params[4], void **session)

May point to any memory chosen by TA

TA_CloseSessionEntryPoint (..)

```
TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
                           uint32_t param_types, TEE_Param params[4])
{
       switch(cmd)
       {
         case CMD_ENCRYPT_INIT:
              ....
       }
}
```

Parameters:

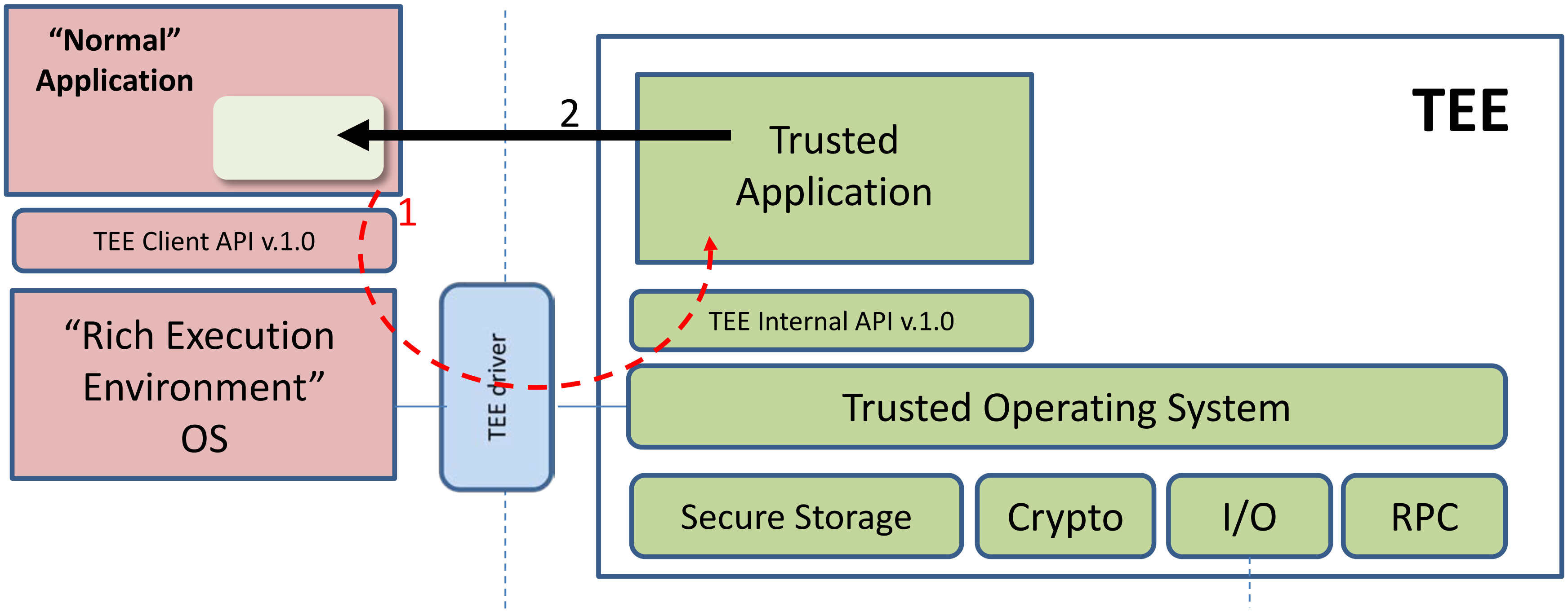


Interaction with a TEE (GP)

TA pointer to shared memory in the callers' context.

Efficient mechanism for in-place encryption / decryption etc.

The TA **programmer must be aware of differences in memory references**.
Ekberg et al, Authenticated Encryption Primitives for Size-Constrained Trusted Computing, TRUST 2012



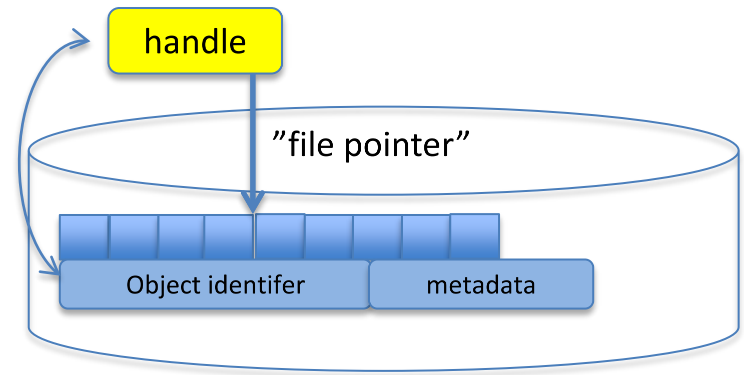
Storage and RPC (GP TEE internal API)

Secure storage: Memory / objects in a TA can be persistently stored

TEE_CreatePersistentObject(TEE_STORAGE_PRIVATE, objID, objIDLen, flags, attributes, .., **handle**)

bytes read

TEE_ReadObjectData(handle, buffer, size, count);
TEE_WriteObjectData(handle, buffer, size);
TEE_SeekObjectData(handle, offset, ref);
TEE_TruncateObjectData(handle, size);



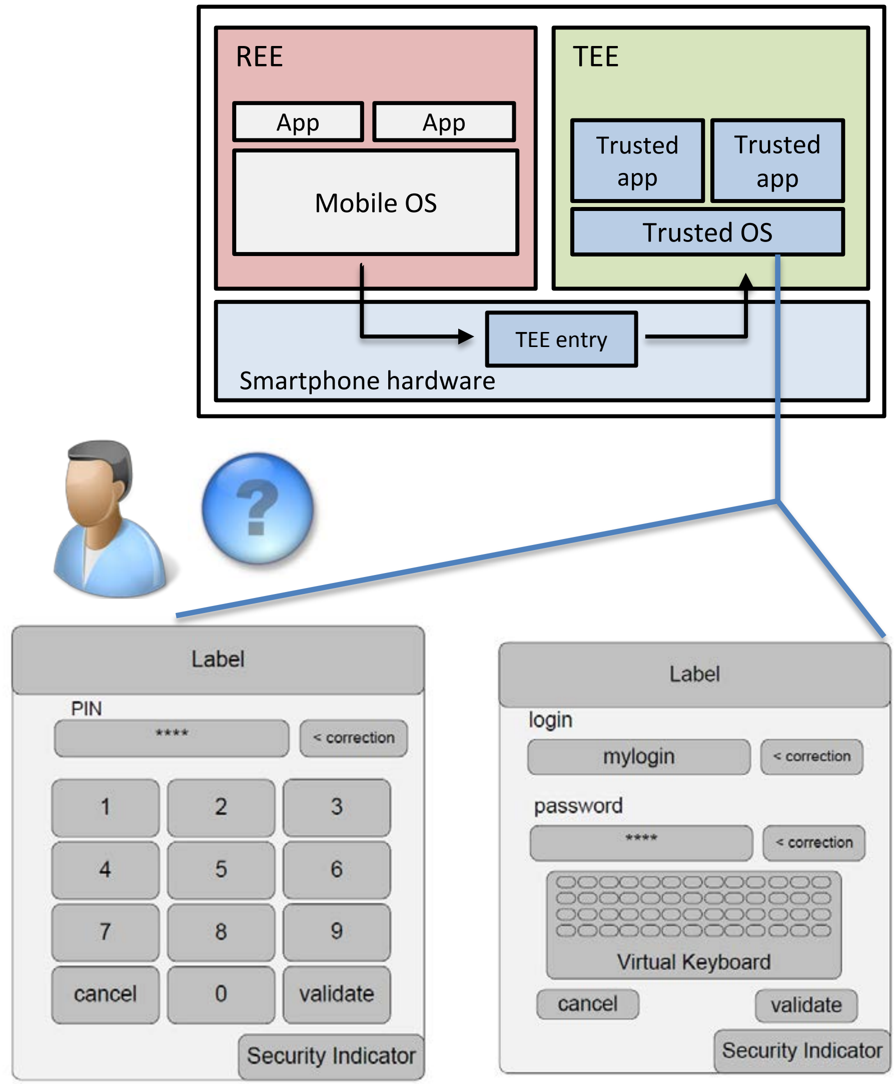
RPC: Communication with other TAs

TEE_OpenTASession(TEE_UUID* destination, …, paramTypes, params[4], &session);
TEE_InvokeTACommand(session, …, commandId, paramTypes, params[4]**);**

(The invocation calls the same interface as the one used for external calls)

Trusted path to user (GP)

- Trustworthy user interaction needed
  - Provisioning
  - User authentication
  - Transaction confirmation
- Trusted User Interface API 1.0:
  - Set up widget structures
  - Call TEE_TUIDisplayScreen
  - Collect results
- Only for I/O directly wired to to the trusted OS

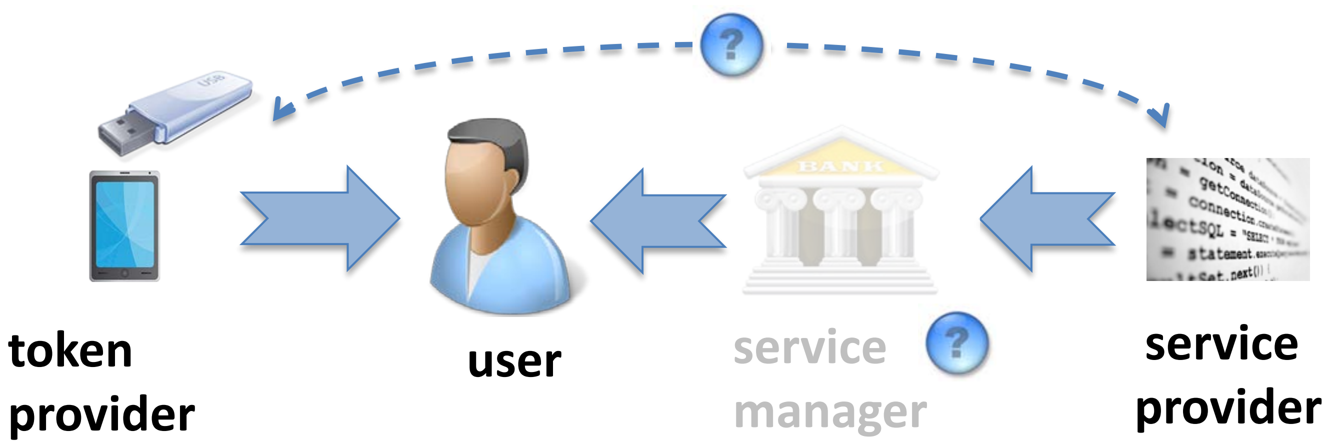


GP User-Centric provisioning model

[User-centric provisioning white paper]

Trad:

New:



GP device committee is working on a TEE provisioning specification

Specifications: www.jedec.org

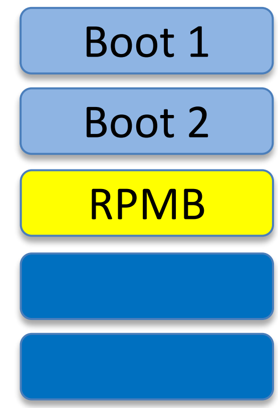
JEDEC ™

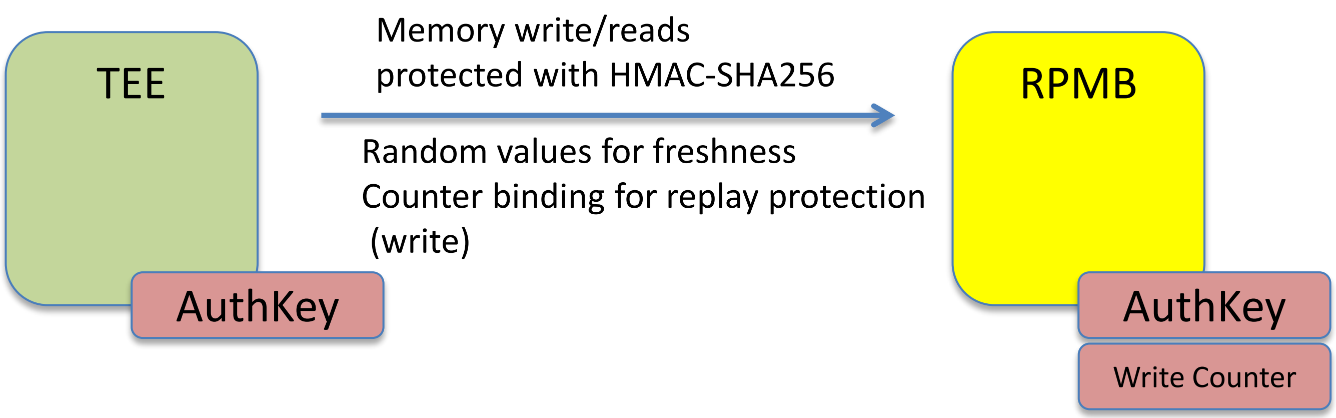
JEDEC RPMB in e·MMC v4.41 and v4.5

Jedec is primarily known for standards like DDR, MMC , UFS, but is important esp. in microelectronics.

RPMB: Replay-Protected Memory Block

- Separate partition in the MMC
- Authenticated channel





Specifications: www.trustedcomputinggroup.org

TRUSTED COMPUTING GROUP TPM / TPM2 / TPM MOBILE

TCG Trusted Platform Module (TPM)

- an application interface to secure services

- deployed to hundreds of millions of PCs and laptop (v1.2. chip + drivers)

- potential way applications and OS services interact with platform security

TPM

- Component that collects state and is separate from system on which it reports

- **Relies on Roots of Trust**

- For remote parties
  Remote attestation in well-defined manner
  Authorization for functionality provided by the TPM

- Locally
  - **Key generation** and **key use** with TPM-resident keys
  - Secure **binding** with encryption, as well as **non-volatile storage**
  - An **engine** for encryption / decryption and signing, also for hash algorithms and symmetric ciphers

A TPM is NOT

- An enforcing component or mechanism for services outside the TPM
- An eavesdropping channel for remote monitoring

HOWEVER

Secure Boot + (GP TEE OR TPM)

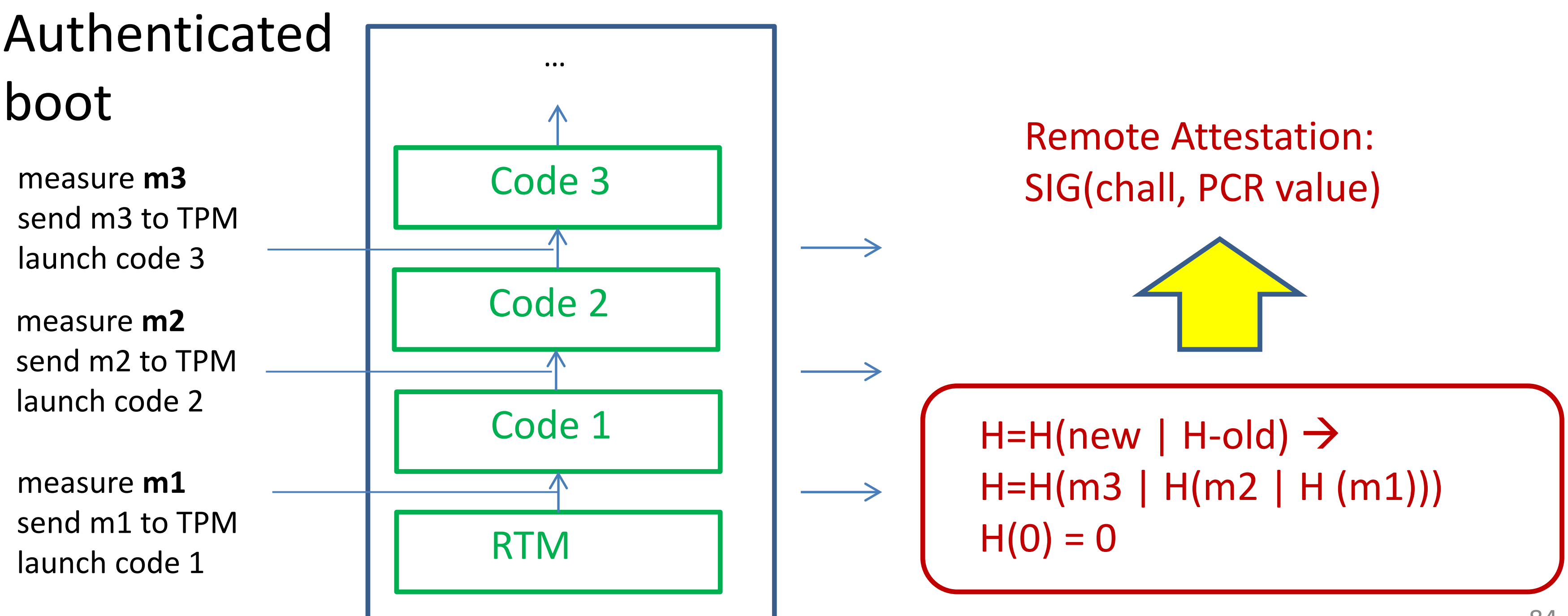
can potentially be used to violate privacy
alternatively, it can be used to protect user privacy

Platform Configuration Register (PCR)

... Measurement aggregation for eventual binding or attestation

... A given expected PCR value can ONLY be reached by a correct extension sequence

... In an aggregate with a trustworthy root, any divergence in reported events causes an irrevocable change in the eventual PCR value.



TPM Mobile (Mobile Trusted Module)

A TPM profile for Mobile devices (v 1.2. & v.2) that adds mechanisms for

Adaptation to TEEs:

New RoT definitions and requirements for TEE adaptation

Multi-Stakeholder Model (MSM):

Rich Application – Trusted Application – TPM relation

Measurements, lifecycle models

Relations between different ”types” of TPM mobiles

”Certified boot”:

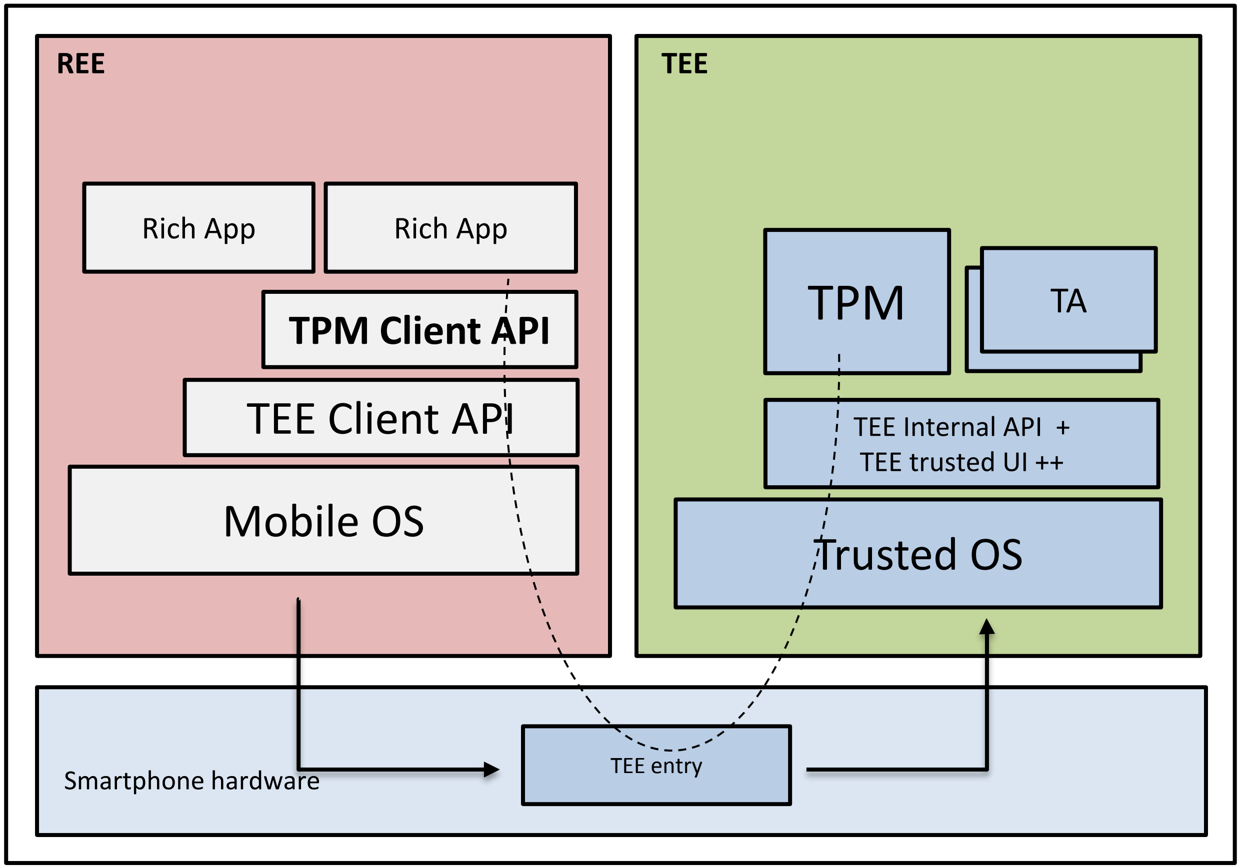
Secure boot with TCG authorizations

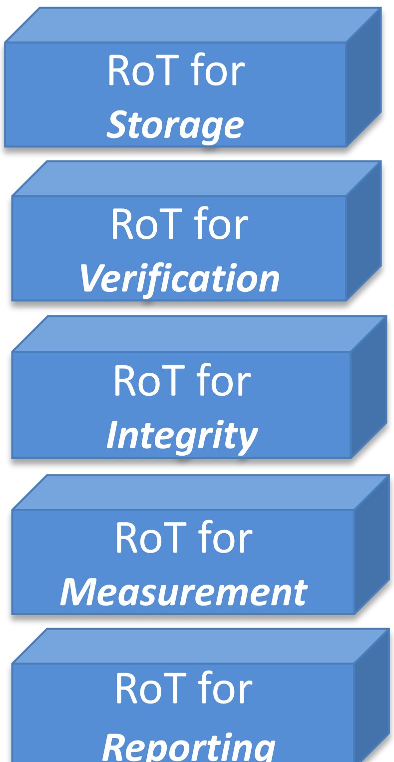
(RIM Certificates → TPM2 authorization)

TPM Mobile on GP TEE

([Whitepaper: TPM on GP TEE])

- Do GP TEEs provide needed functionality?
- Do GP TEEs provide needed security assurance?

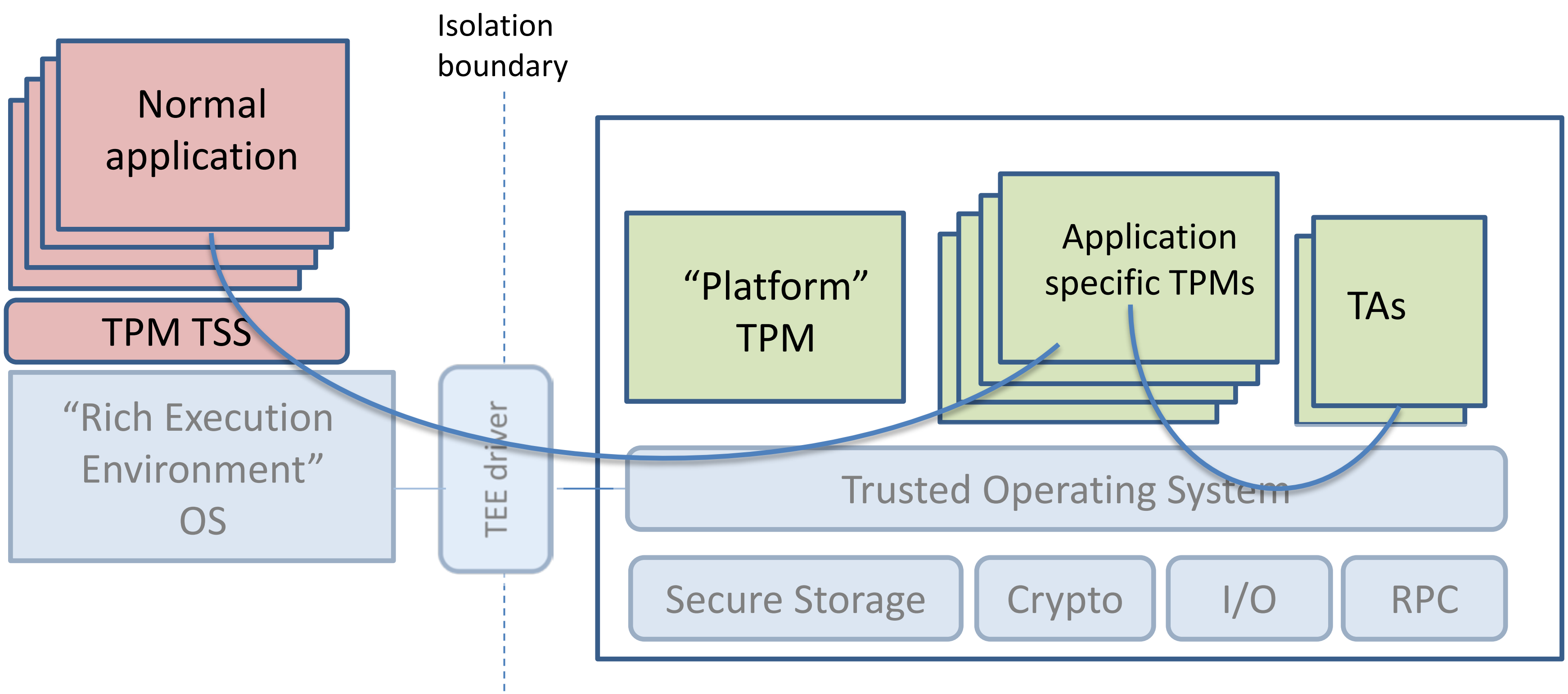




TPM Mobile Multi-Stakeholder Model (MSM)

A TEE **can** host a mumber of ”simultaneous” TPMs
One TPM (platform) is needed for OS services – say secure boot

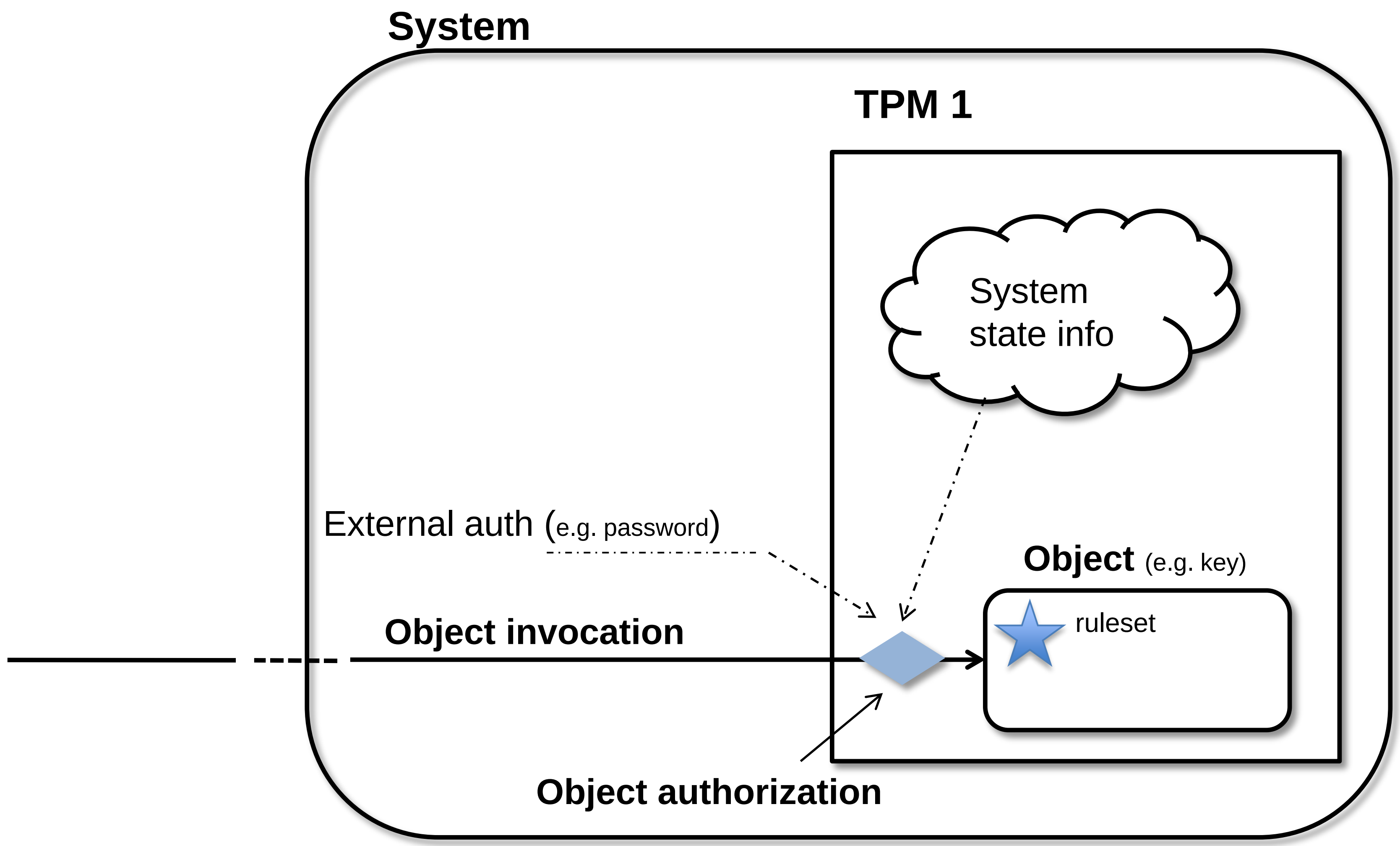
Most applications do not need dedicated code (a TA) in the TEE. But they may need secure storage, state-aware keys, and attestation for those



TPM authorization

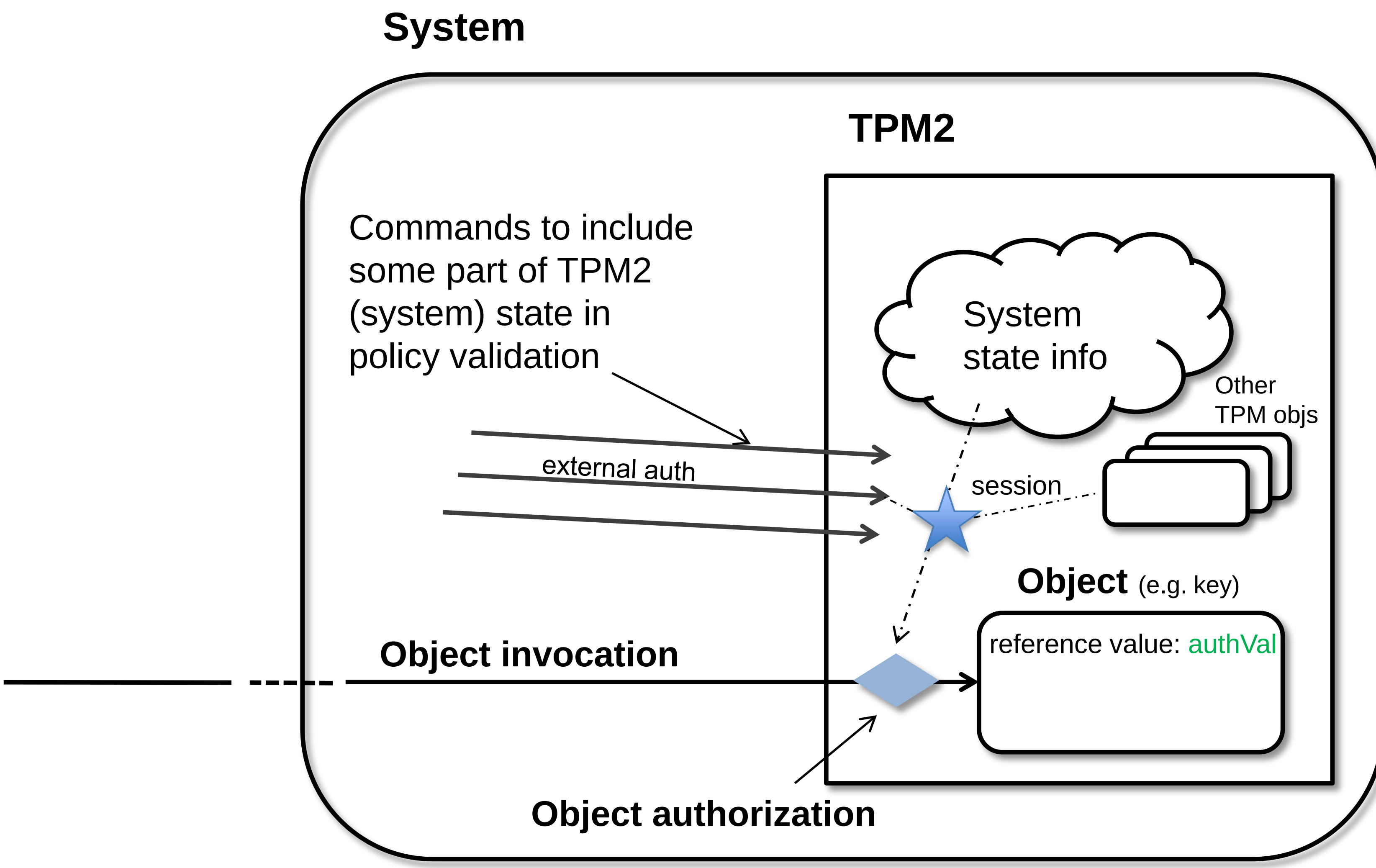
- Many users of varying security levels
- System state awareness is a fundamental to TPMs – sets TPMs apart from e.g. removable smartcards.
- To implement any TPM service that **enforces control**, authorization is essential

Authorization (policy) TPM 1



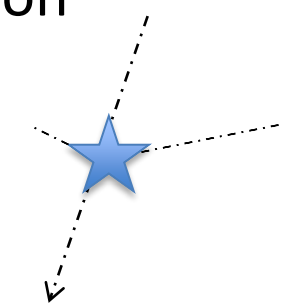
MTM added key authorization, but only for PCRs

Authorization (policy) TPM2



TPM2 Policy Session

- different types of preconditions can be part of an authorization policy (session)
- In addition, logical relations should be applicable on the set of atomic preconditions that constitutes the policy (AND, OR)
- A policy session accumulates all policy information needed to make the authorization decision.



TPM2 Policy Session Contents

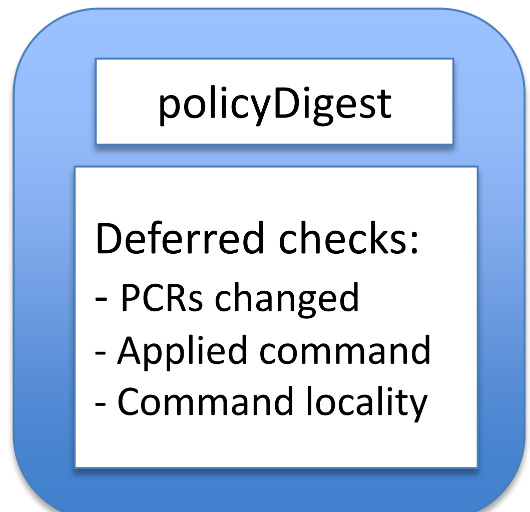
- An accumulated session policy value called **policyDigest**

newDigestValue := H(oldDigestValue ||
policyCommand || stateinfo)

- Some policy commands reset the value

IF condition THEN
newDigestValue := H(0 || policyCommand
|| stateinfo)

- Session also contains optional **assertions** to be made at object access.



TPM2 Policy Command Examples

- **TPM2_PolicyPCR:** Include a set of PCR values in the authorization

  sessionUpdate.state_info := [pcr value, pcr index}

- **TPM2_PolicyNV:** Include a reference value and operation index in case a comparison (<, >, eq) of a non-volatile memory area with the reference succeeds.

  e.g., *if counter5 > 2 then*
  sessionUpdate.state_info := [ref, op, mem.area]

TPM2 Deferred Policy Examples

- **TPM2_PolicyCommandCode:** Include the command code specification in session:

  sessionUpdate.*state_info := command code*

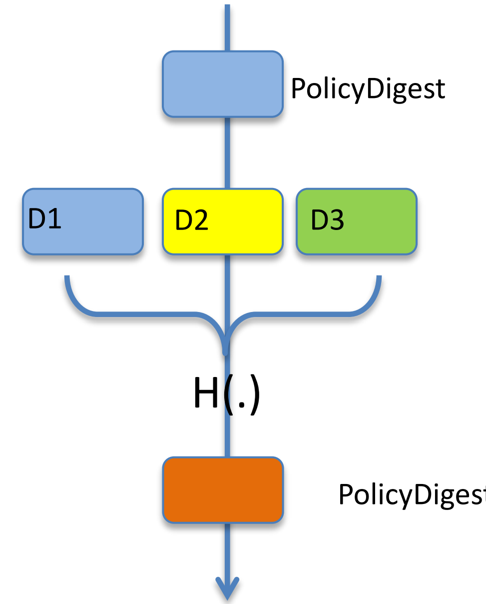
  deferred : policySession->commandCode := command code

- **TPM2_PolicyLocality:** Restrict the operation to a given locality:

  sessionUpdate.*state_info := locality*

  deferred : policySession->commandLocality := locality

TPM2 PolicyOR



(Successful OR)

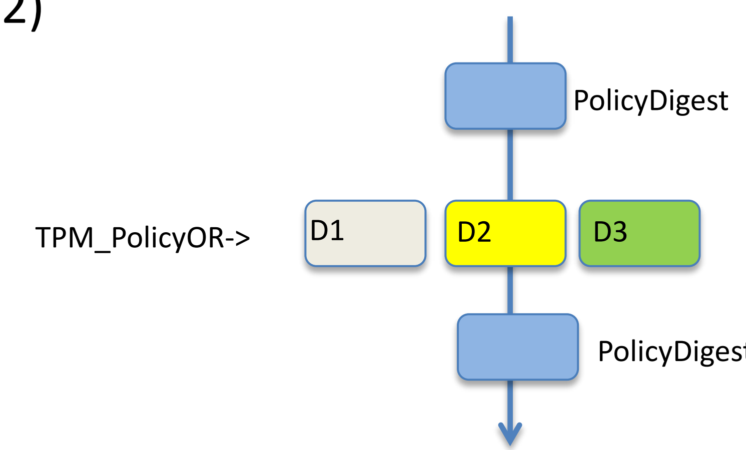
TPM2_PolicyOR: Authorize one of several options:

Input: *List* of digest values <D1, D2, D3, .. >

IF *policySession->policyDigest* in *List* **THEN**

newDigestValue := H(0 || policyCommand || List)

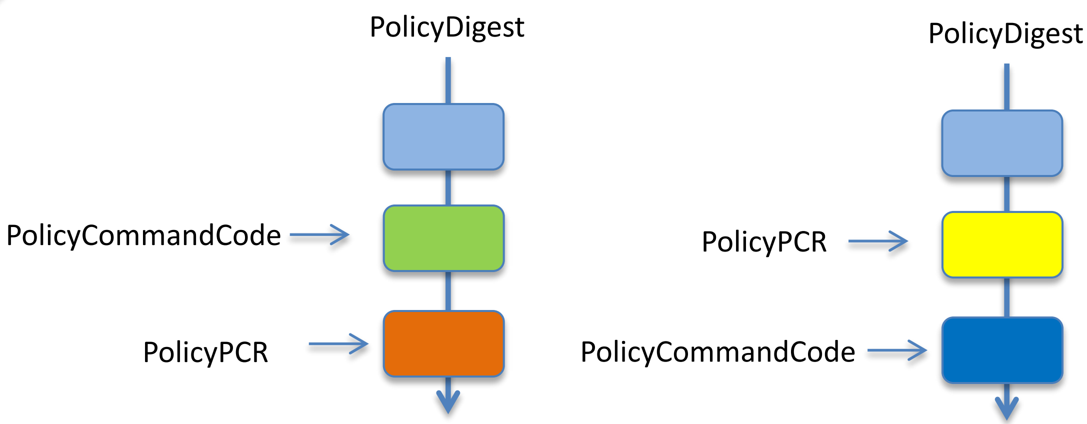
Reasoning: H(List) is known (fixed) policy. For a wrong digest Dx (not in set <D1 D2 D3>) it is difficult to find another *List2* = <Dx Dy, Dz, .. > where H(List) == H(List2)



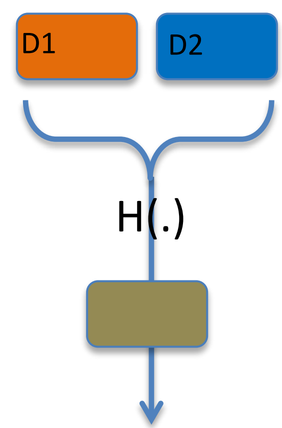
(Failing OR)

”TPM2 PolicyAND”

- There is no explicit AND command
- AND is achieved by to consecutive policy commands → order dependence



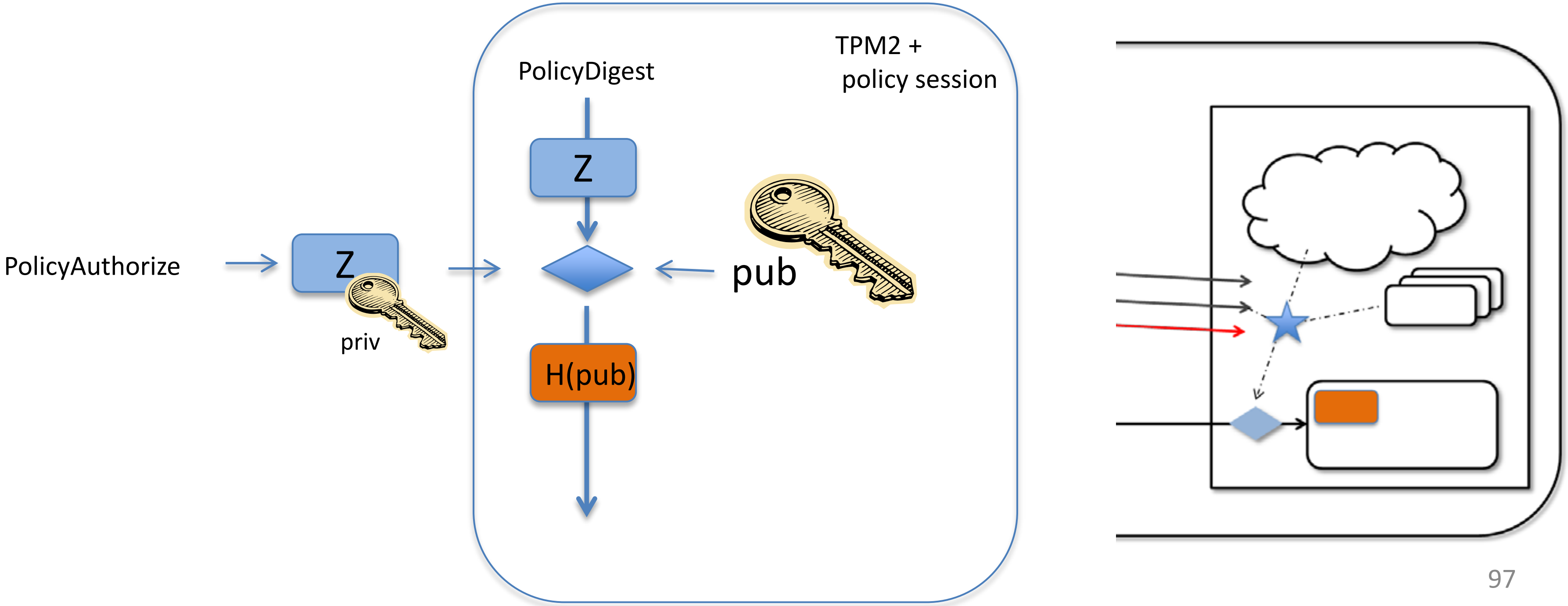
Theoretical example: Use an OR to hide the order dependence of an AND



External Authorization

TPM2_PolicyAuthorize: Validate a signature on a policyDigest:

IF signature validates
AND *policySession->policyDigest* in *signed content*
THEN
newDigestValue := H(0 || policyCommand || **pub**|| ..)



Simple secure boot is not always enough

Secure boot **can** have the following properties

A) Extend all the way into OS / application booting

B) Can include platform-dependent policy

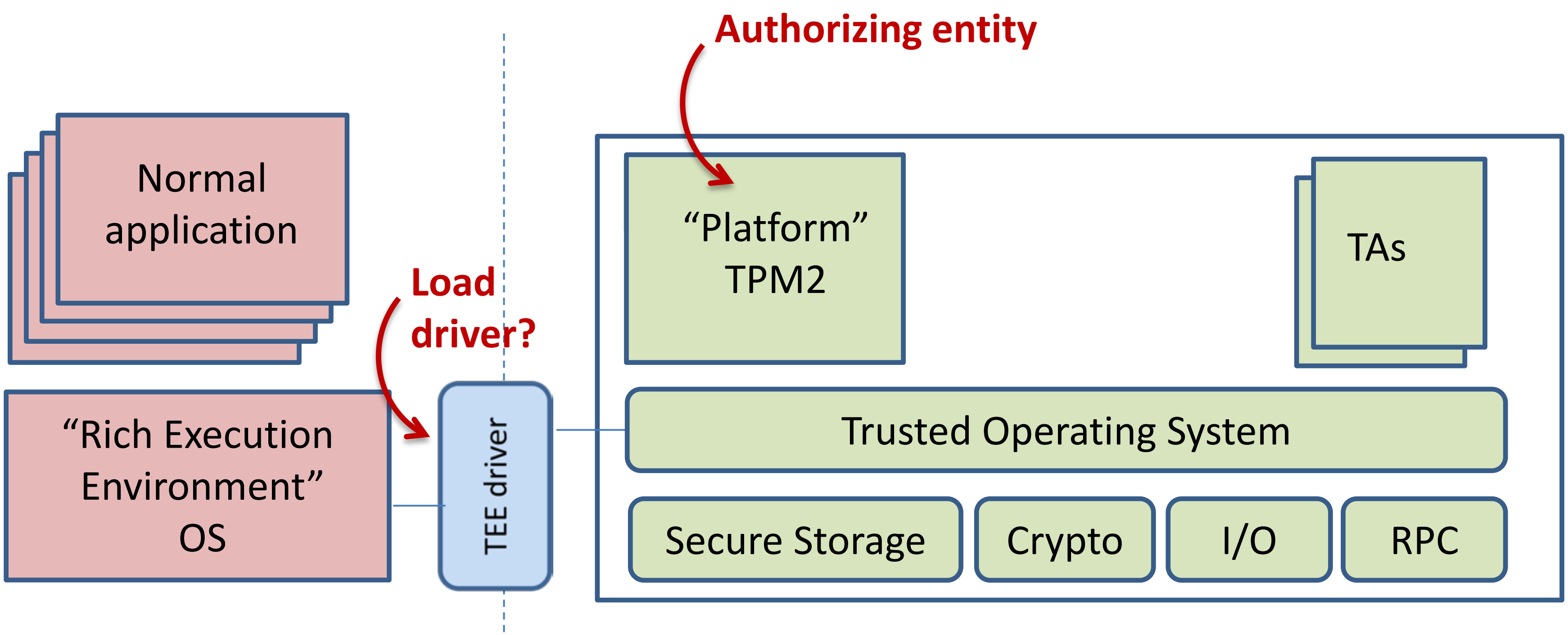
C) Can include optional / complementary boot branches

D) Order in which components are booted may matter

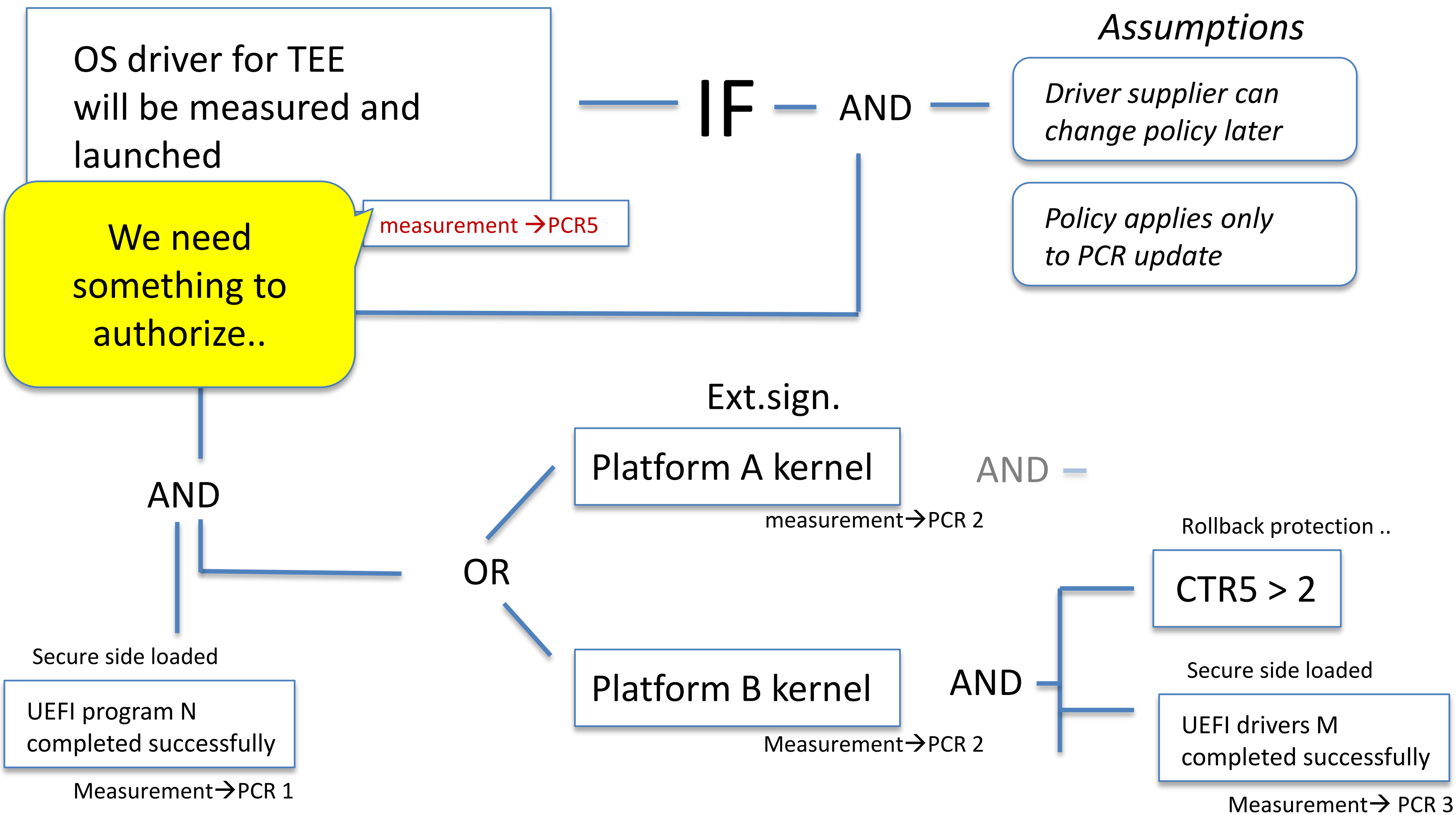
TPM2 authorizations can be used for secure boot:
Example follows

Secure boot "constructed example"

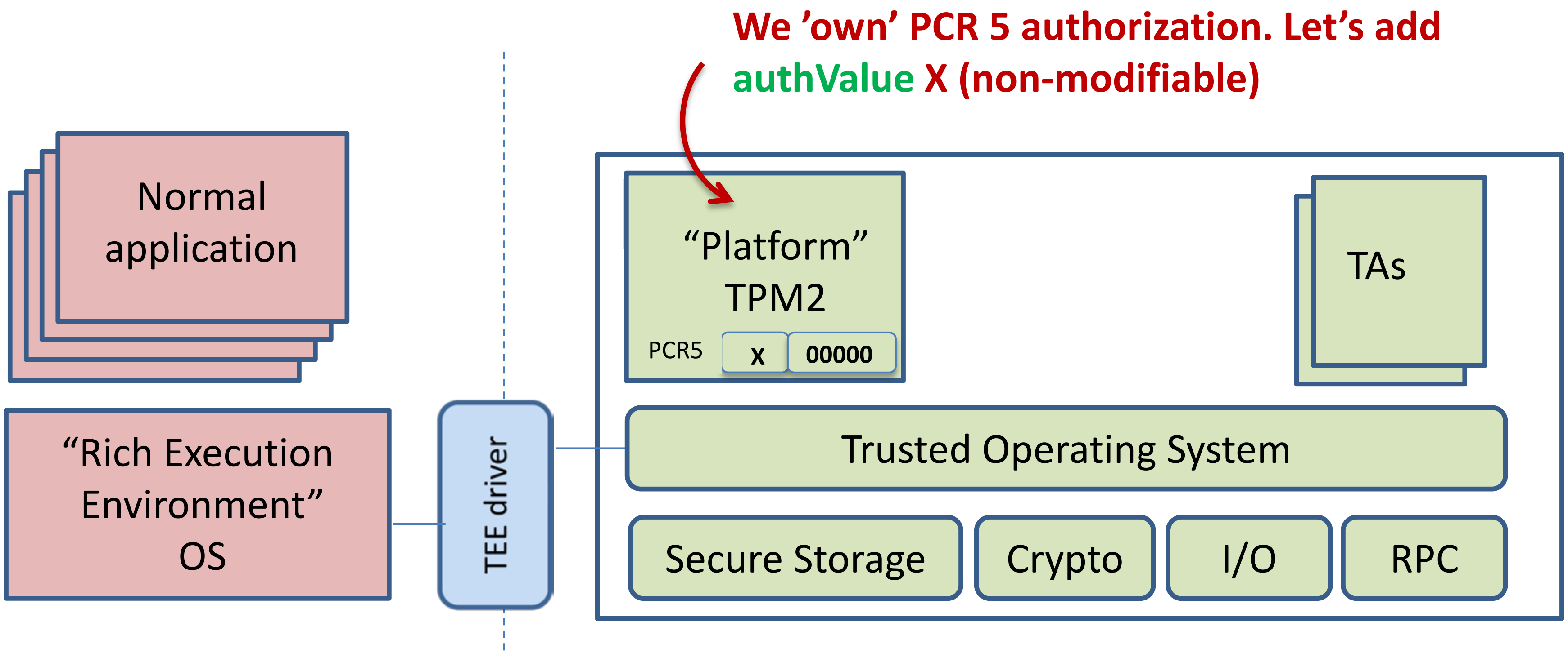
1. UEFI started the boot process
2. A UEFI program loads the TEE, TPM etc (PCR 1)
3. A UEFI OS loader loads the OS (PCR 2)
4. The OS boots
5. We want to (dynamically) **load the driver** that communicates with some aspect of the TEE --- the TPM must of course be accessible



Example policy



Example policy

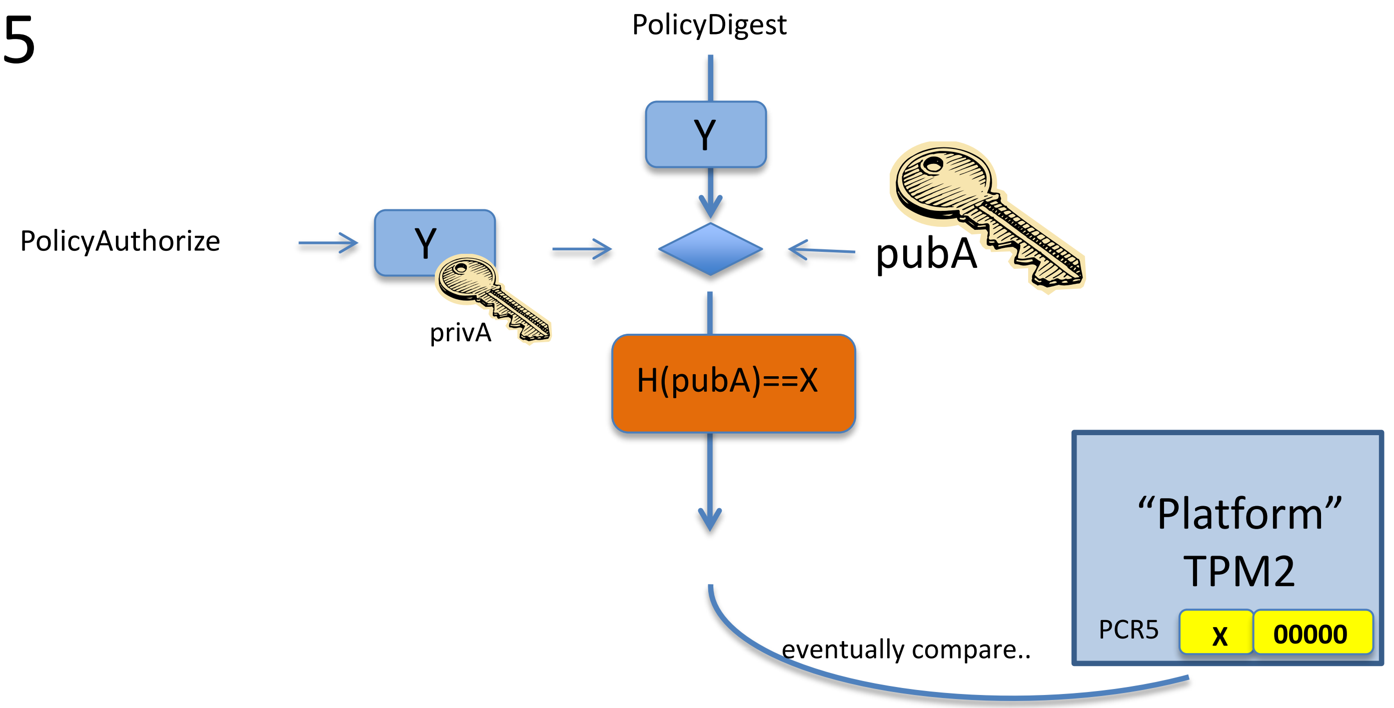


What is a good value for X?

Example policy

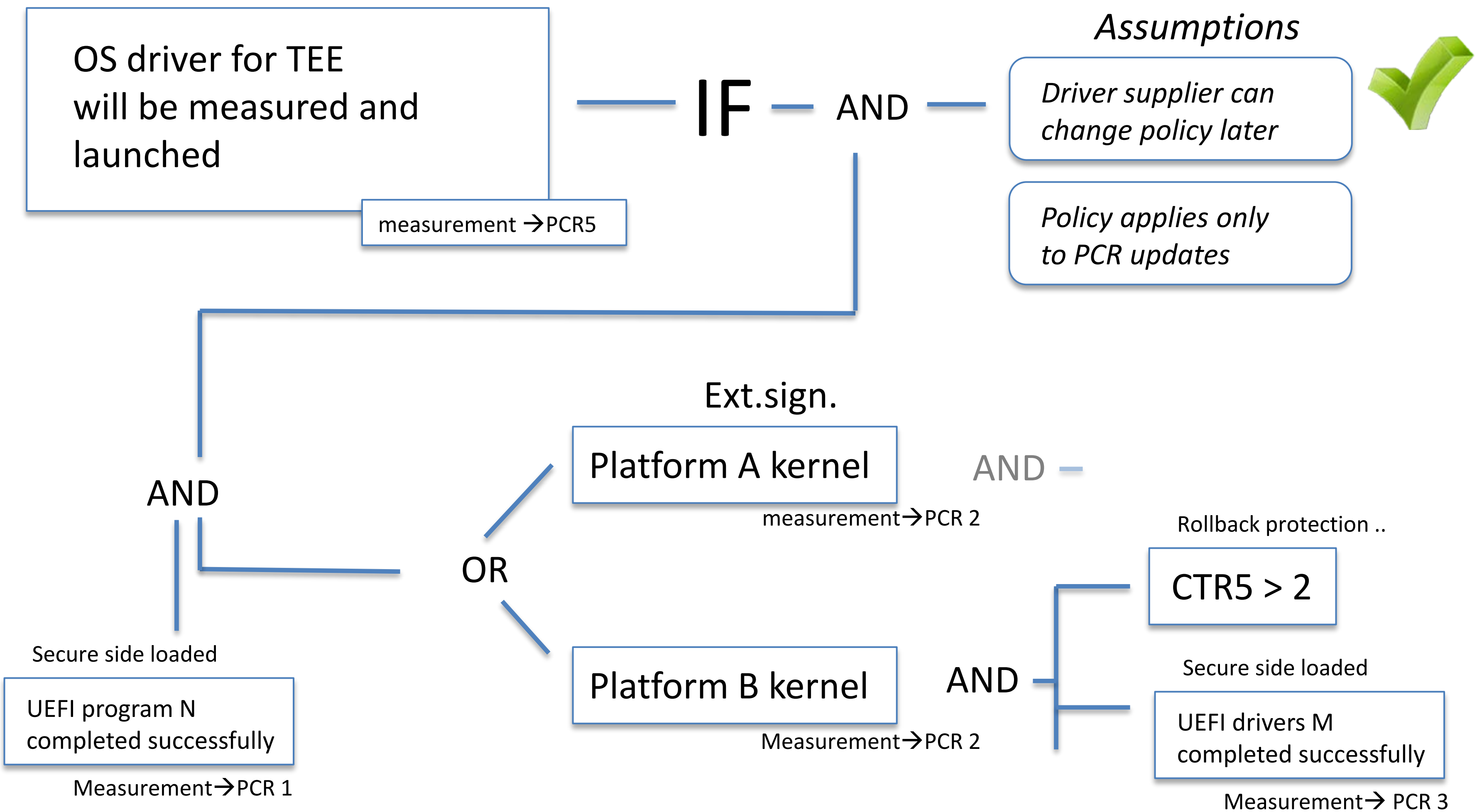
If X is H(pubA)
we can authorize
any value Y as policy
for PCR 5

[actually H(0 || PolicyAuthorize || **pubA** || ..)]



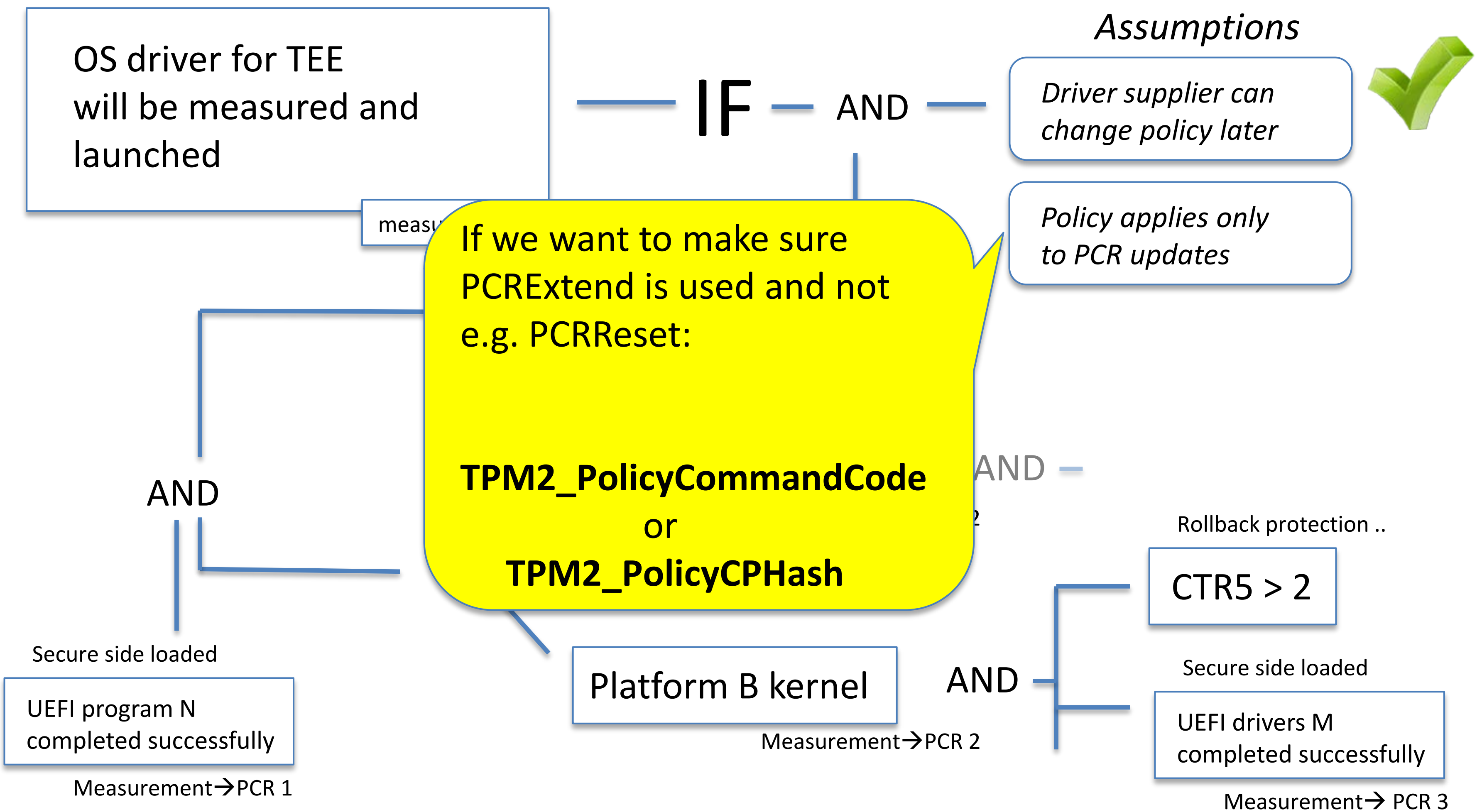
Y → PolicyAuthorize(Sig_A(Y)) → **X**

Example policy



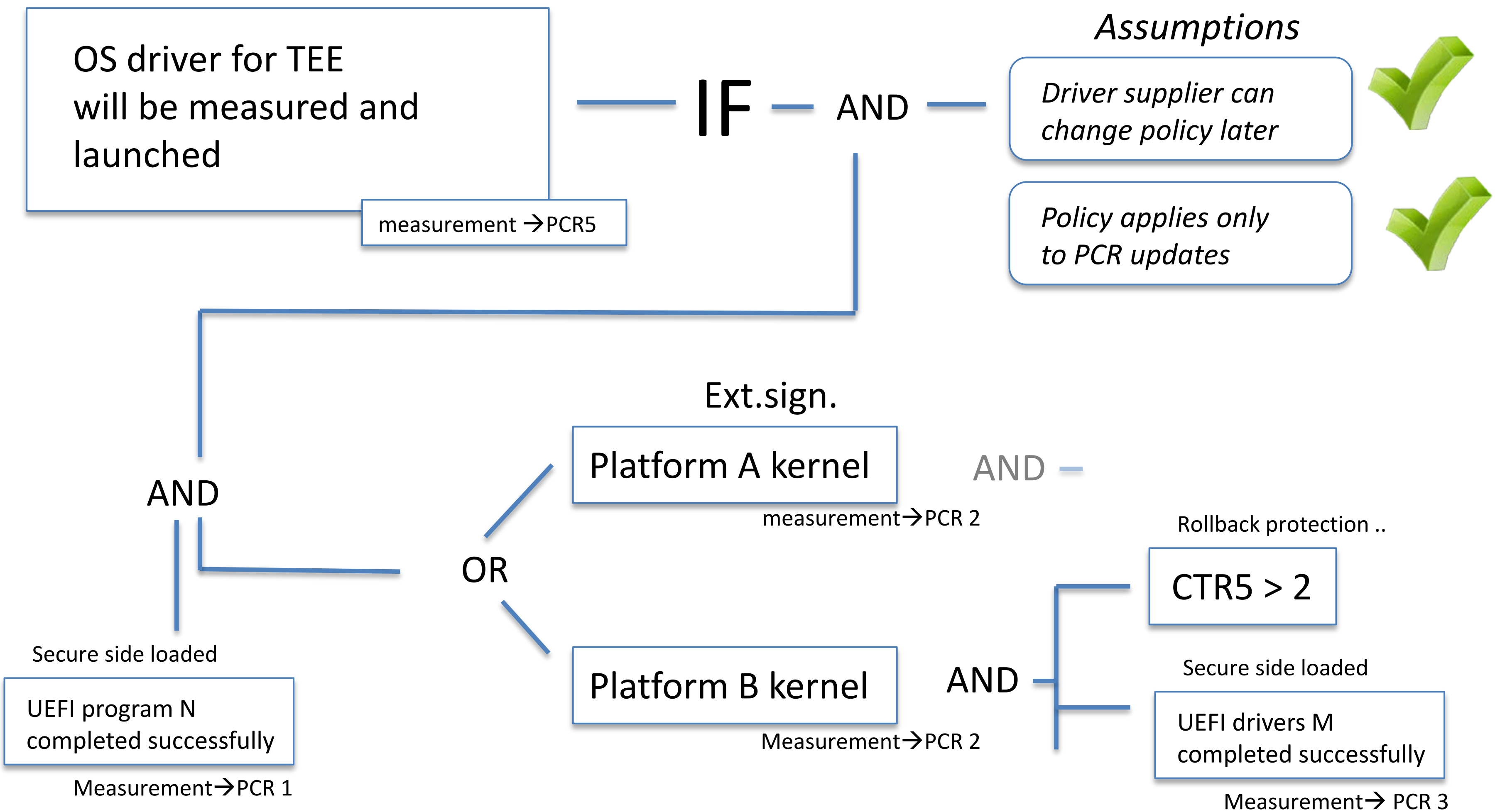
Y → PolicyAuthorize(Sig_A (Y)) → **X**

Example policy



Y → PolicyAuthorize(Sig_A (Y)) → **X**

Example policy



Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig$_A$(Y)) → **X**
{**Check:** Eventual command == TPM_PCRExtend}

Example policy

OS driver for TEE
will be measured and
launched

measurement → PCR5

IF — AND —

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

Ext.sign.

AND

To bind a PCR value:

TPM2_PolicyPCR (index(1), value(expected meas.))

(actually an aggregate PCR hash)

Secure side loaded

UEFI program N completed successfully

Measurement → PCR 1

Measurement → PCR 2

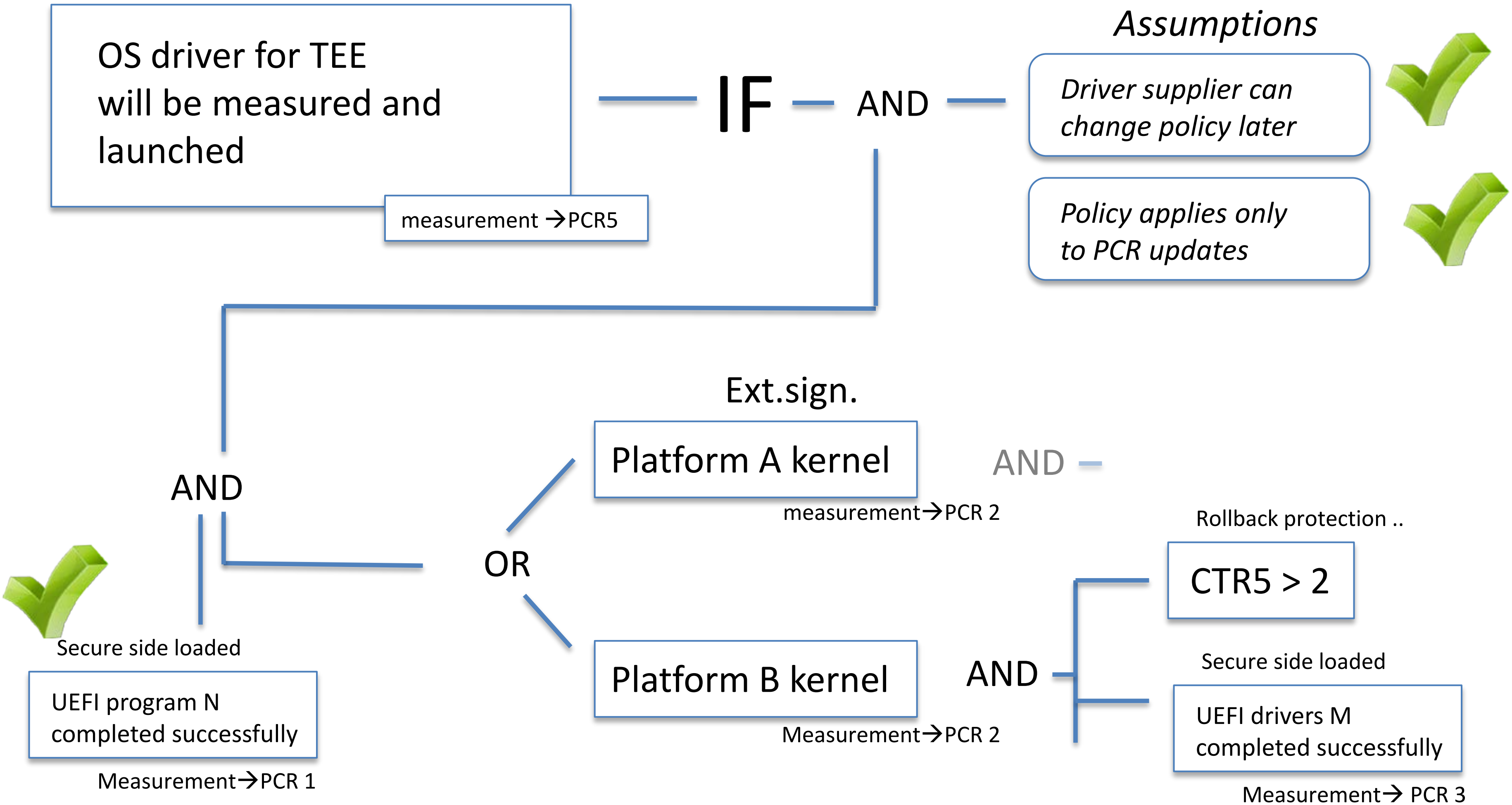
UEFI drivers M completed successfully

Measurement → PCR 3

Z → PolicyCommandCode(TPM_PCRExtend) → **Y →** PolicyAuthorize(Sig$_A$(Y)) **→ X**

{**Check:** Eventual command == TPM_PCRExtend}

Example policy

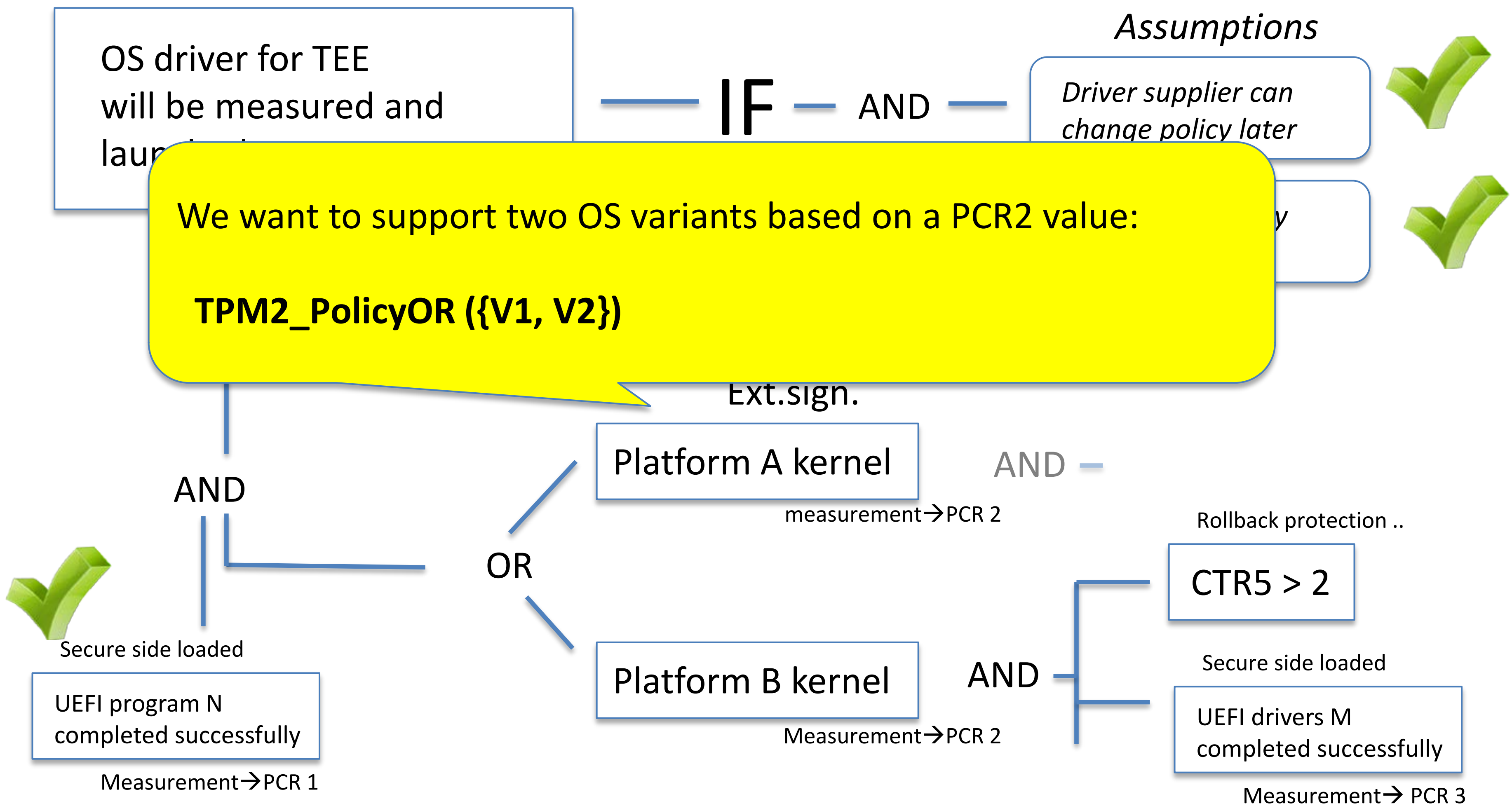


W → PolicyPCR(1, meas.) → **Z**

Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig$_A$(Y)) → **X**

{**Check:** Eventual command == TPM_PCRExtend}

Example policy

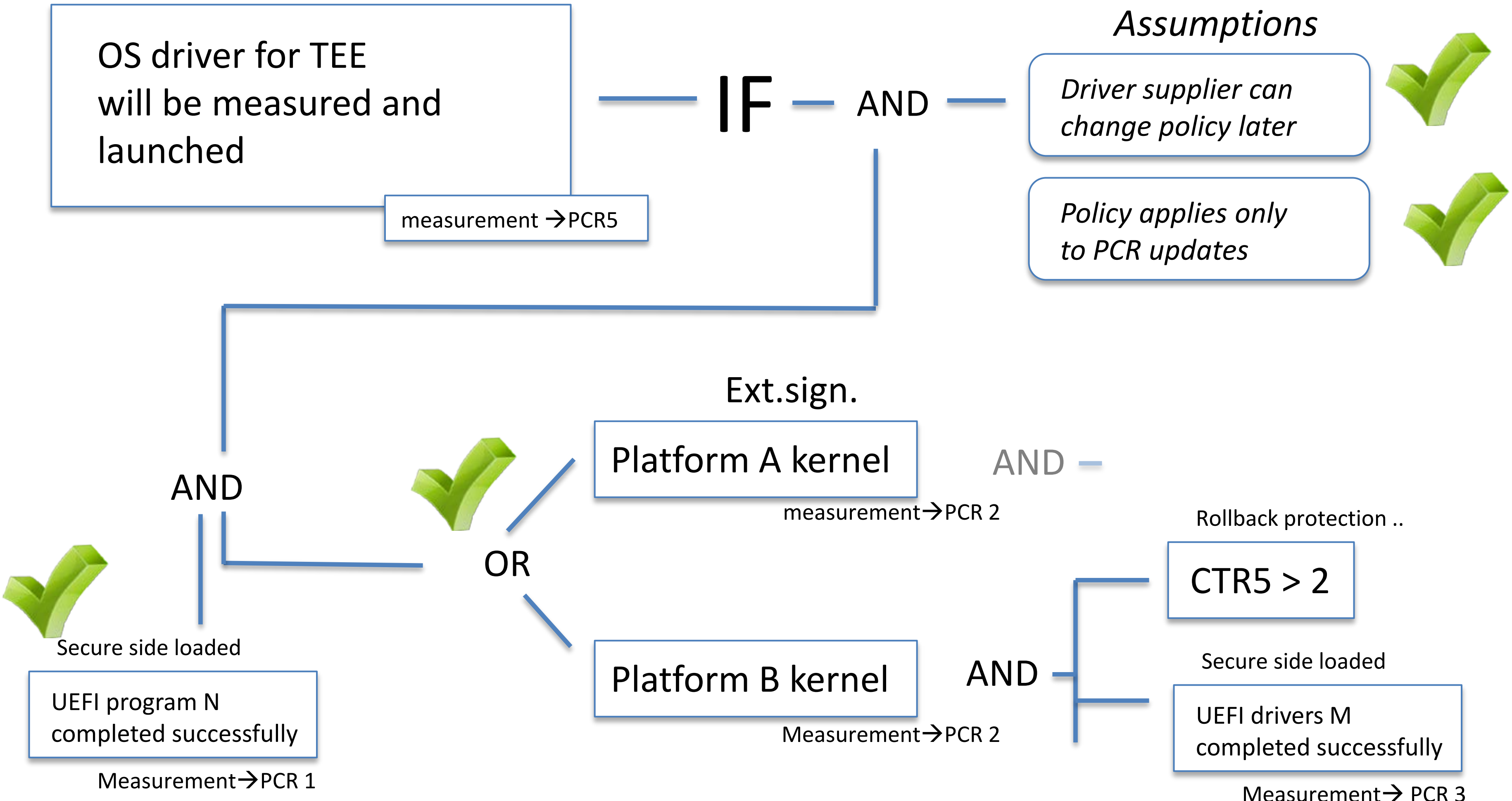


W → PolicyPCR(1, meas.) → **Z**

Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig$_A$(Y)) → **X**

{**Check:** Eventual command == TPM_PCRExtend}

Example policy



V1 →
V2 → PolicyOr({V1,V2} →**W** → PolicyPCR(1, meas.) → **Z**

Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig$_A$(Y)) → **X**

{**Check:** Eventual command == TPM_PCRExtend}

Example policy

Provider of OSB may do certified or authenticated boot. Thus:

Possibly there are many more authorizations needed (like a **PolicyNV**)

or

The OS provider updates PCR2 with result of some **PolicyAuthorize(Sig_{OSB}(...))** and guarantees its own freshness

OS driver for T
will be measur
launched

AND

Secure side loaded

UEFI program N completed successfully

Measurement→PCR 1

OR

Platform A kernel

measurement→PCR 2

AND

Platform B kernel

Measurement→PCR 2

AND

Rollback protection ..

CTR5 > 2

Secure side loaded

UEFI drivers M completed successfully

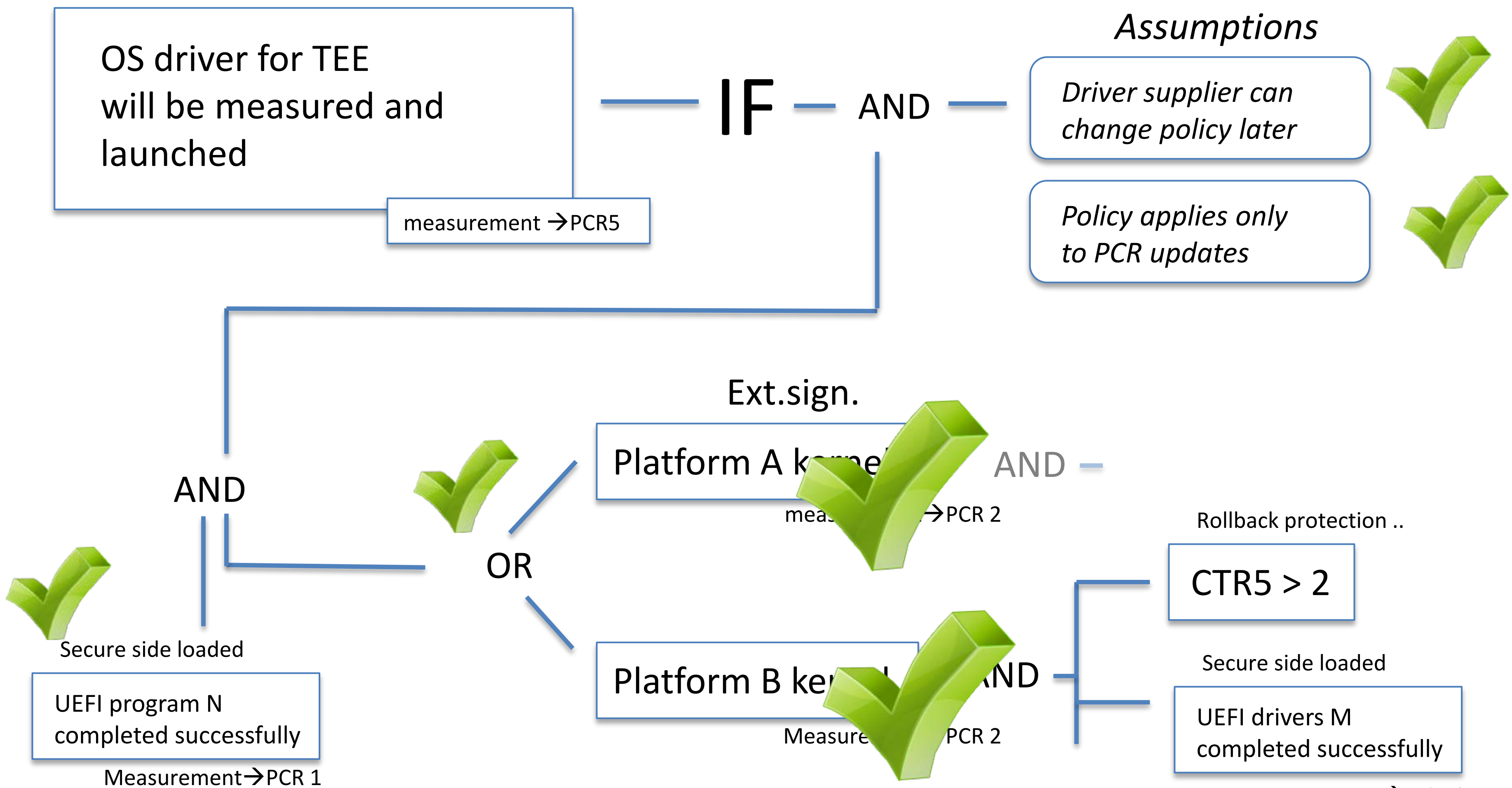
Measurement→ PCR 3

V1 →
V2 → PolicyOr({V1,V2} →**W** → PolicyPCR(1, meas.) → **Z**

Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig_A(Y)) → **X**

{**Check:** Eventual command == TPM_PCRExtend}

Example policy



PolicyPCR(2, H(...)) →**V1** →
PolicyPCR(2, H(...)) →**V2** →
PolicyOr({V1,V2} →**W** → PolicyPCR(1, meas.) → **Z**

Z → PolicyCommandCode(TPM_PCRExtend)→**Y** → PolicyAuthorize(Sig_A(Y)) → **X**

{**Check:** Eventual command == TPM_PCRExtend}

Recap: Example “boot sequence”

- **UEFI** starts TEE and lauches OS → PCR1 updated
- Operating System boots up
  - **TPM PolicyAuthorize** →OS manufacturer PCR2 updated

Policy Session

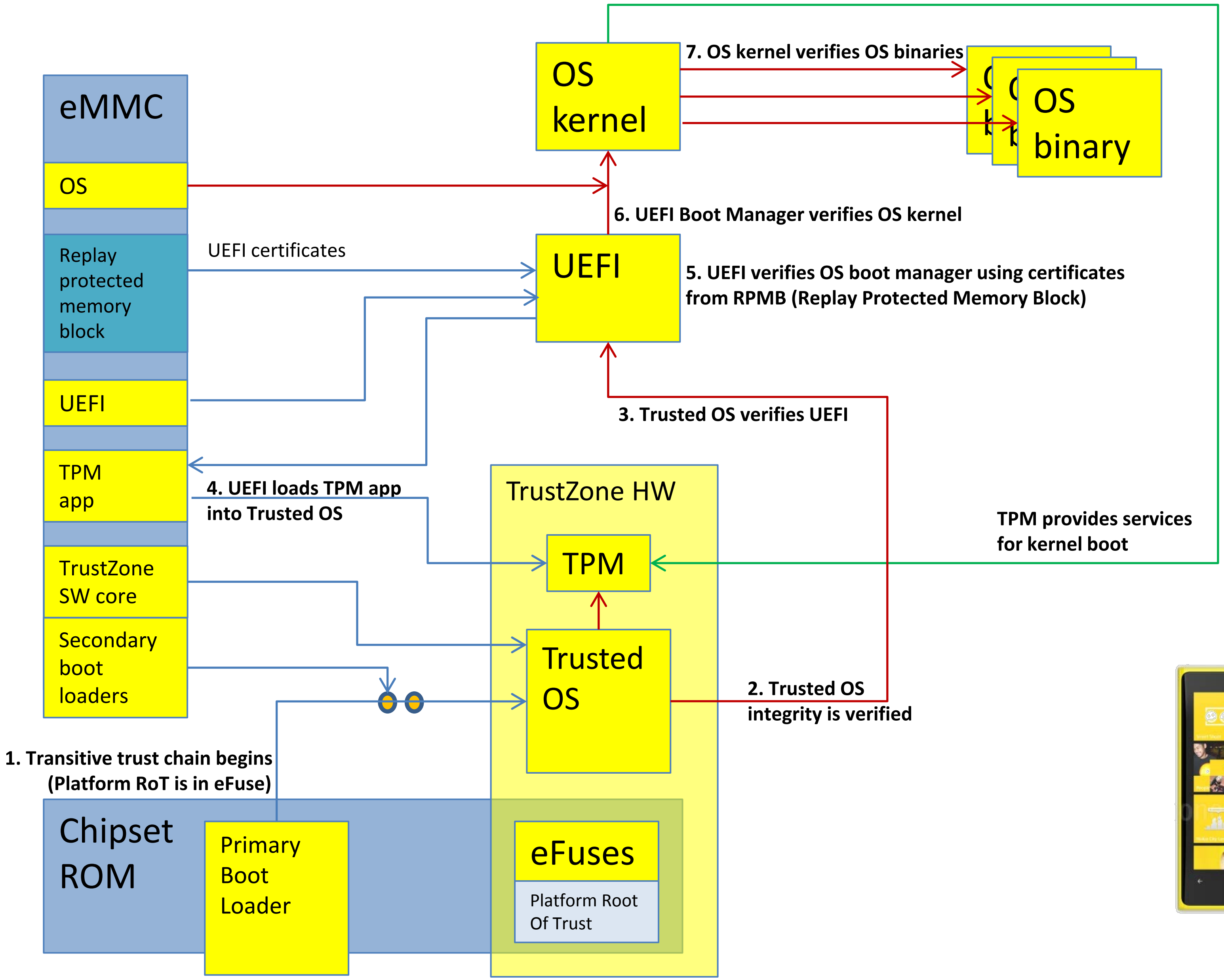
- **TPM_PolicyPCR** (PCR 2 ”Sign of OS provider”), → OS OK
- **TPM_PolicyOR** → One of two OSs values accepted
- **TPM_PolicyPCR** (PCR 1, ”H(TEE meas.)”) → TEE version correct
- **TPM_PolicyCommand** (PCRExtend) → Only authorize a PCRExtend command
- **TPM PolicyAuthorize** →”I” authorize the collected state

X=X

→TPM_PCRExtend(PCR 5, measurement value)

{**Check:** Eventual command == TPM_PCRExtend}

Deployed standards:Nokia Lumia Secure Boot Flow



Challenges ahead and summary

A LOOK AHEAD

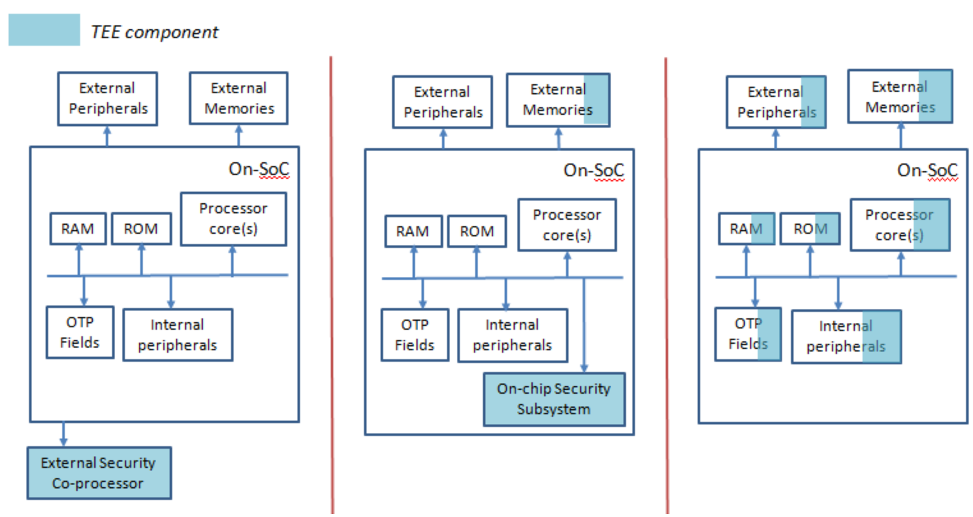
Skip to summary

Challenges ahead

- What is the right TEE architecture?
  - Processor secure environments vs. Separate secure elements vs ...?
- Hardware security and privacy
  - Secure boot and control points, TEE rootkits
- Provisioning
  - Does ‘open provisioning’ emerge as viable alternative for centralized model?
- Trusted user interaction
  - How to establish a secure channel between TEE and the user?
- Certification / verification
  - How to gain confidence in TEE designs?

What is the right TEE architecture?

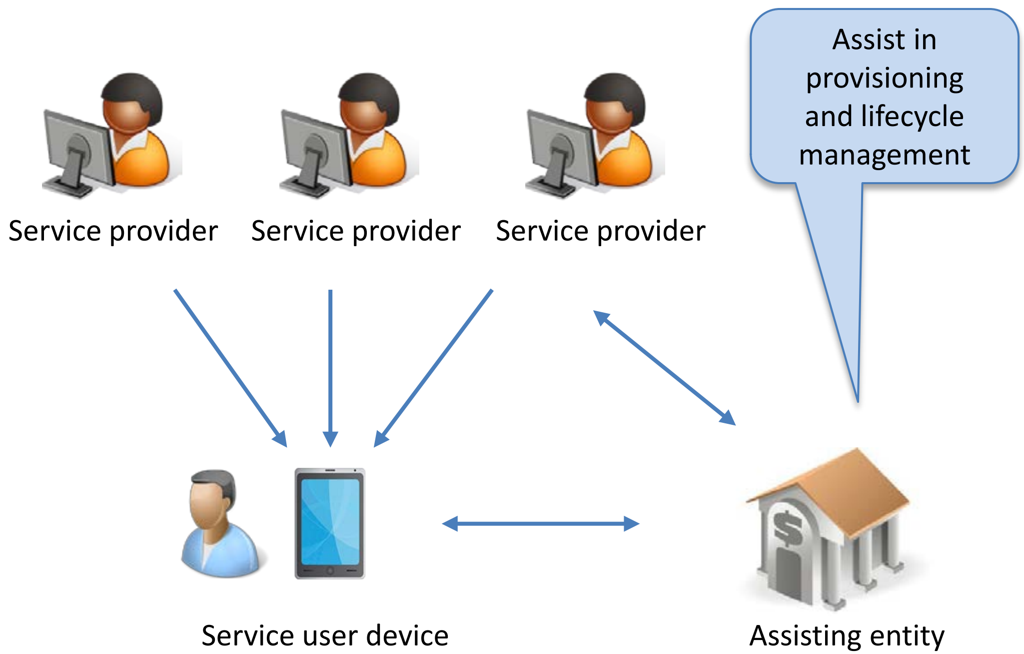
- Processor security architecture vs. embedded secure element vs. some combination?
- New designs like Intel SGX
- Multiple cores multiple TEEs
- Dealing with peripherals (UI, sensors, NFC, ...)



Hardware security and user privacy?

- Secure boot can be used to limit user choice
- Vulnerabilities in TEE implementation → rootkits

What is the right provisioning model?



- Open provisioning
  - Easy service deployment
  - But challenging lifecycle management

- Hybrid model?

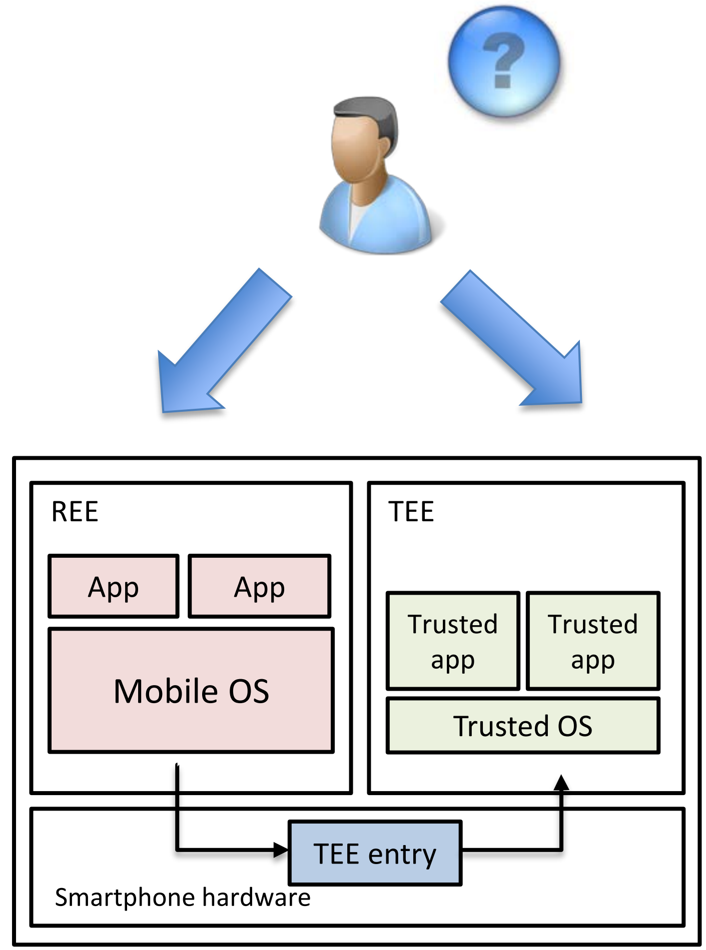
Kostiainen, Asokan and Afanasyeva. *[Towards User-Friendly Credential Transfer on Open Credential Platforms](#). ACNS 2011.*

How to provide trusted path to the user?

- Trustworthy user interaction needed
  - Provisioning
  - User authentication
  - Transaction confirmation

- Technical implementation possible

- But how does the user know?
  - Secure attention key (ctrl-alt-del)
  - Security indicator



Verification and certification?

- Common Criteria model may not be suitable for TEEs
  - too slow
  - too inflexible (cannot efficiently deal with software upgrades)

- Alternatives may/will emerge
  - UK: CPA
    http://www.cesg.gov.uk/servicecatalogue/CPA/Pages/CPA.aspx

Summary

- Hardware-based TEEs are widely deployed on mobile devices
  - But access to application developers has been limited
  - This is about to change
- TEE functionality and interfaces are being standardized
  - Promise of better third-party developer access
  - GlobalPlatform TEE architecture
  - Trusted Computing Group: TPM 2.0 specification
- Many open issues lie ahead…
- Thank you for any feedback (contact info in author copy)

Ekberg, Kostiainen and Asokan. The Untapped Potential of Trusted Execution Environments on Mobile Devices. IEEE S&P magazine, (to appear). (author copy)

Forthcoming e-book

"Mobile Platform Security"
(to be published by Morgan-Claypool)

Draft version at publisher stand (lobby)

Publisher offers to give you a free copy!