Mobile Platform Security

Trusted Execution Environments

TCE Summer School, 2014

N. Asokan
Aalto University and University of Helsinki

Jointly prepared with:
Jan-Erik Ekberg, Trustonic
Kari Kostiainen, ETH Zurich

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Aalto University

UNIVERSITY OF HELSINKI

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
  - Why do mobile devices have TEEs?
- Mobile hardware security
  - What constitutes a TEE?
- Application development
  - Mobile hardware security APIs

- Current standardization
  - UEFI, NIST, Global Platform
- Current standardization: TCG
  - TPM1.2 and TPM 2.0 extended authorization model

- A look ahead
  - Challenges and summary

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 expectations
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

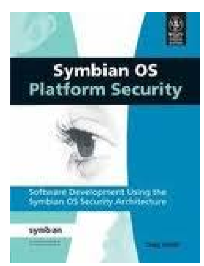
~2001

J2ME

~2002

TrustZone®
Security Foundation by ARM®

~2005



~2008

Historical perspective

1970 1980 1990 2000 2010

Second part

Cambridge CAP

VAX/VMS

Trusted Platform Module (TPM)

GP TEE standards

NIST

TPM 2.0

Reference monitor

Java security architecture

Late launch

Simple smart cards

ARM TrustZone

TI M-Shield

Intel SGX

Protection rings

On-board Credentials

Mobile hardware security architectures

First part

Hardware-assisted secure boot

Mobile OS security architectures

Java Card platform

Mobile Trusted Module (MTM)

TPM Mobile

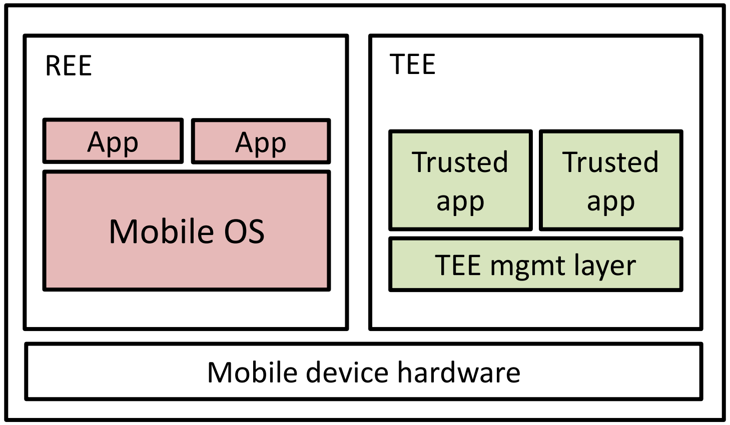
Computer security
Smart card security
Mobile security

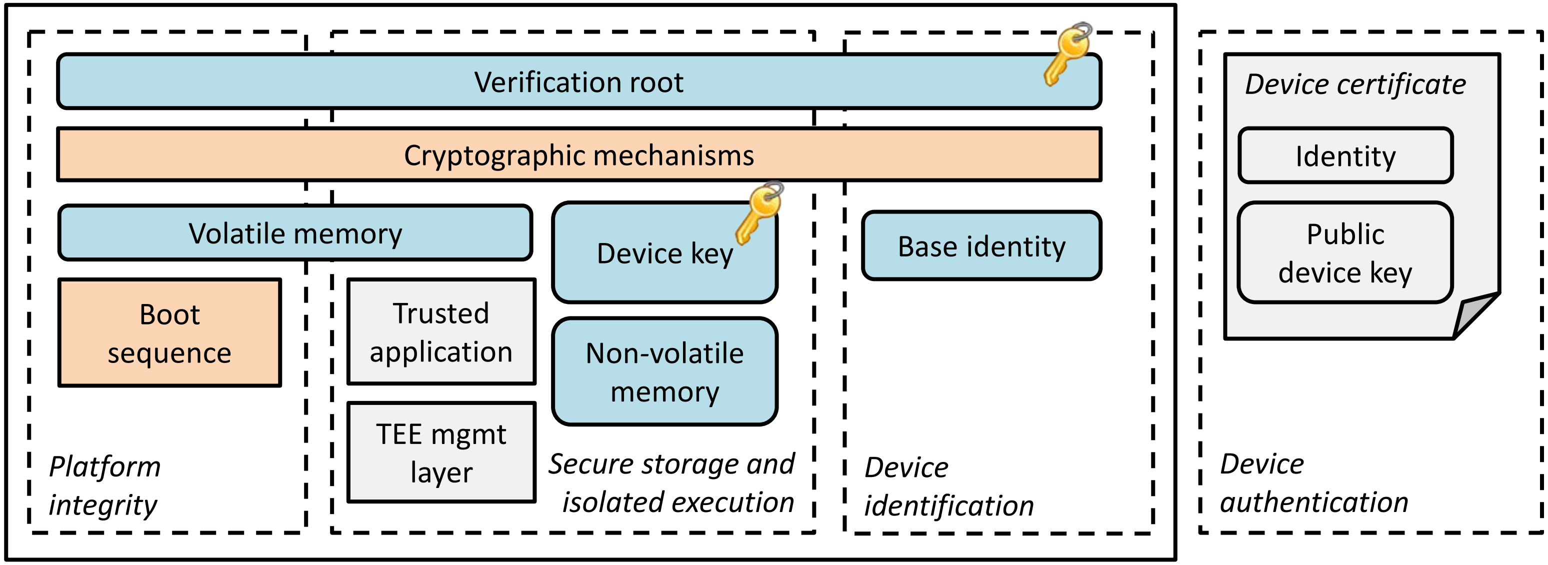
What constitutes a TEE?

MOBILE HARDWARE SECURITY

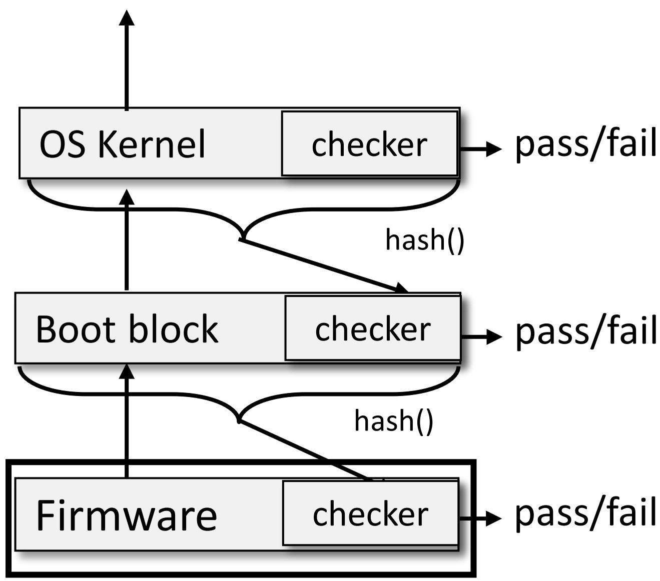
TEE overview

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



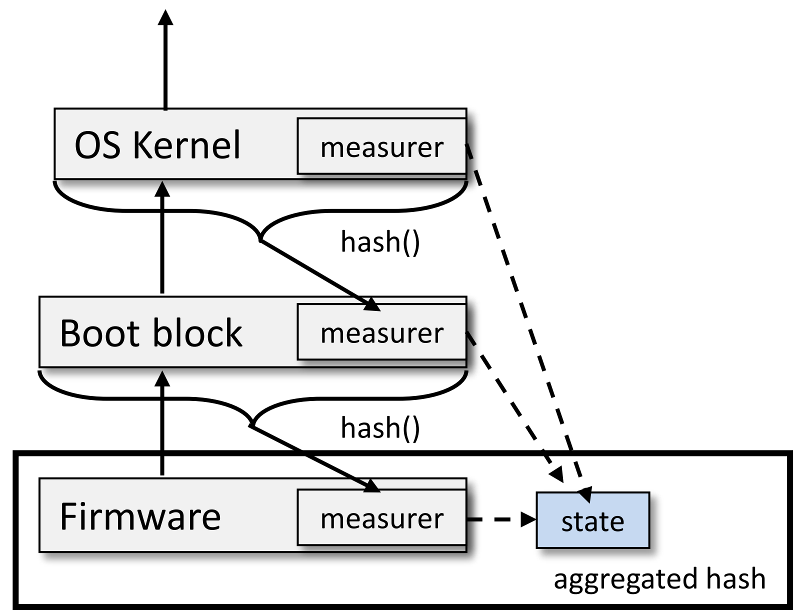


Secure boot vs. authenticated boot



Why? Secure boot

Start only known trusted configurations



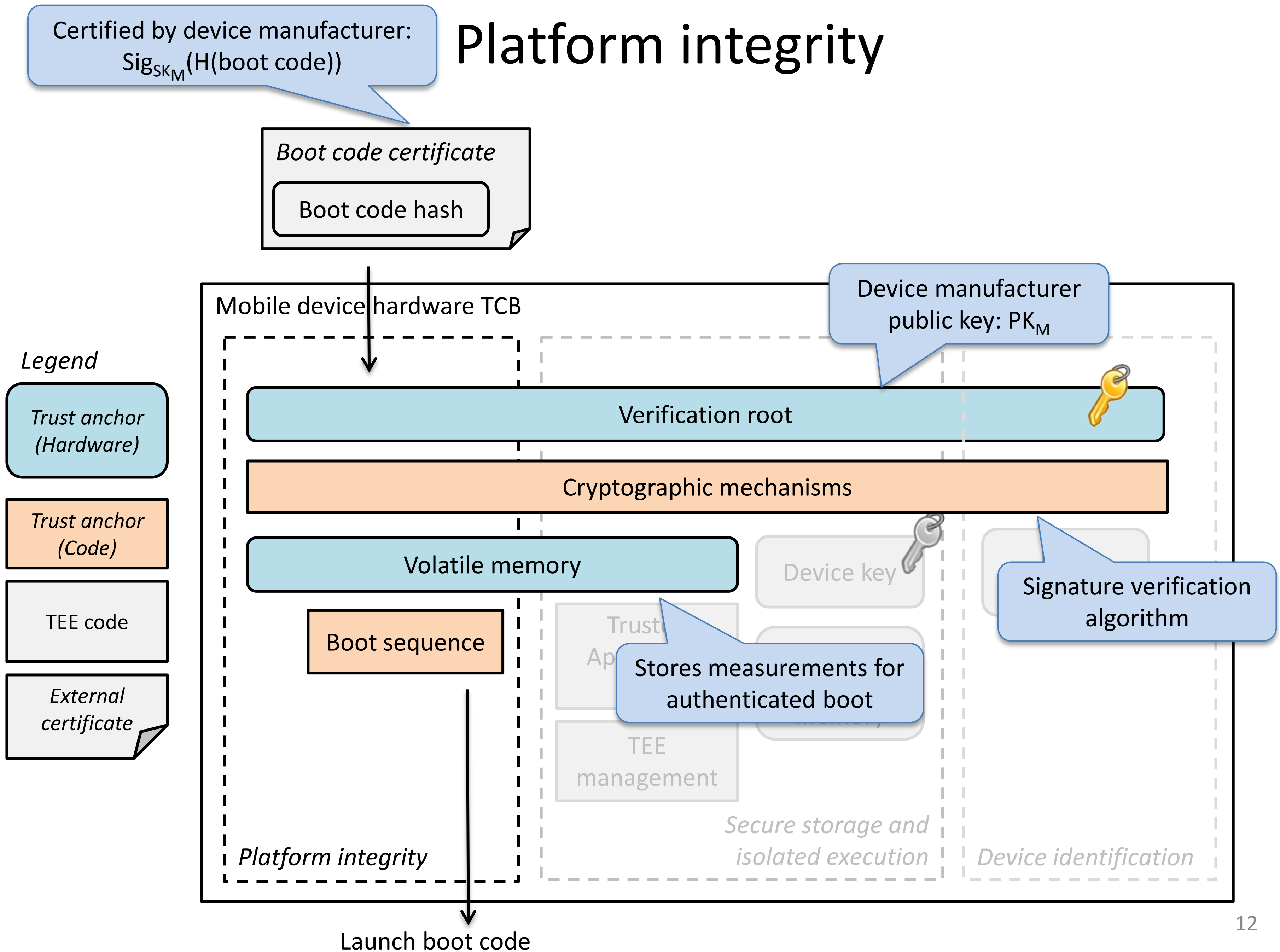
Why? Authenticated boot

Start any configuration, but remember the state

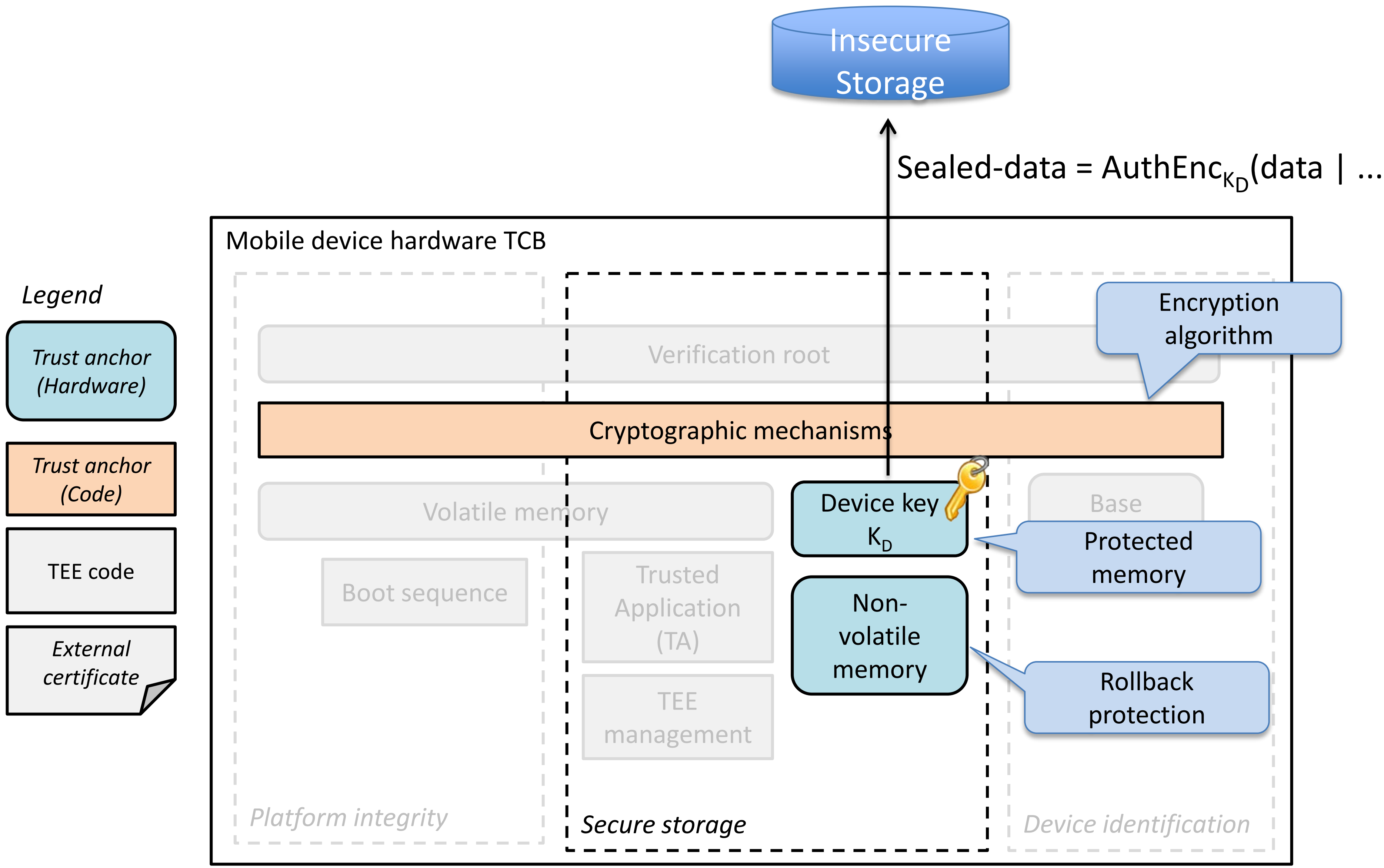
State can be:

- bound to stored secrets (sealing) or resources
- reported to external verifier (remote attestation)

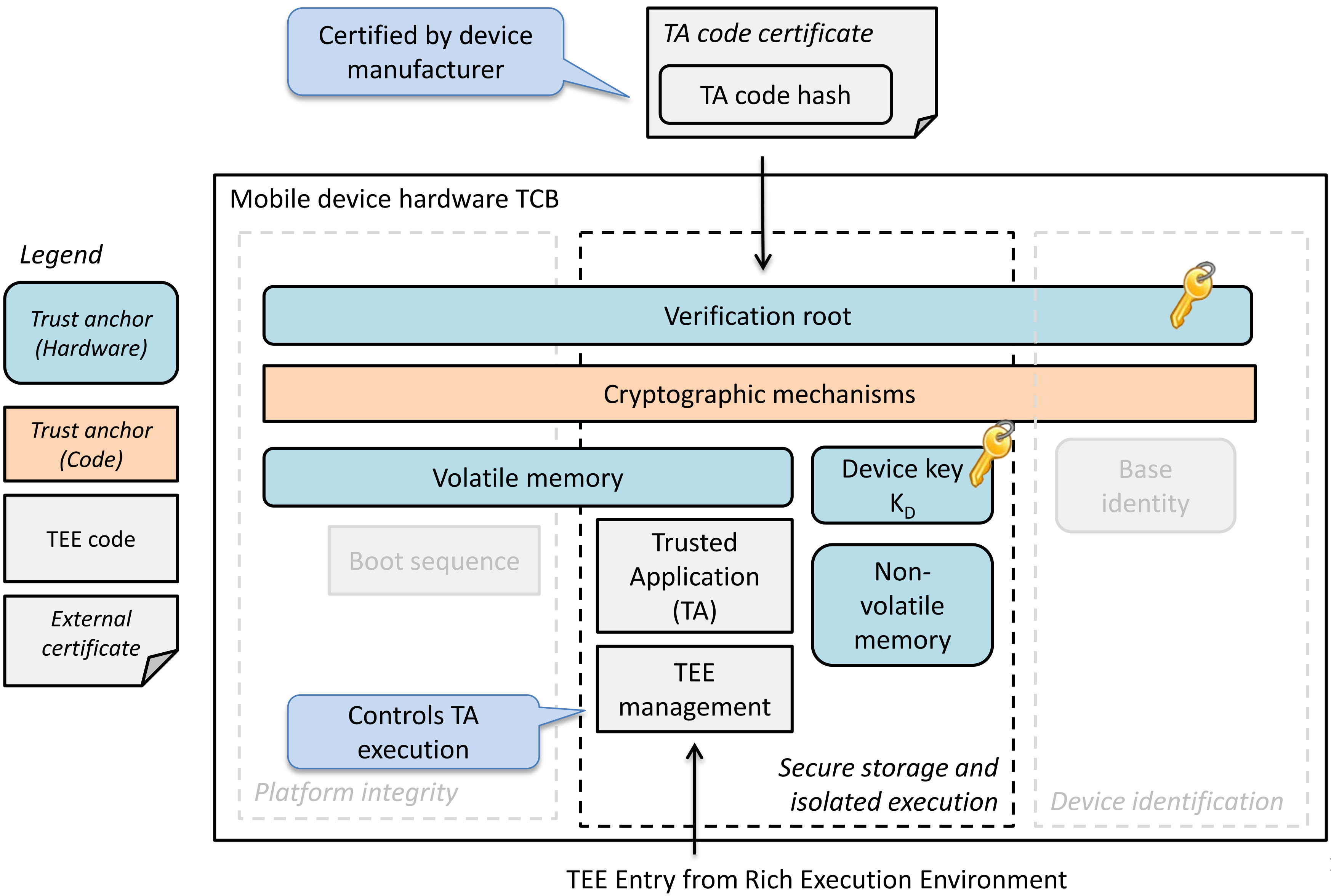
Platform integrity



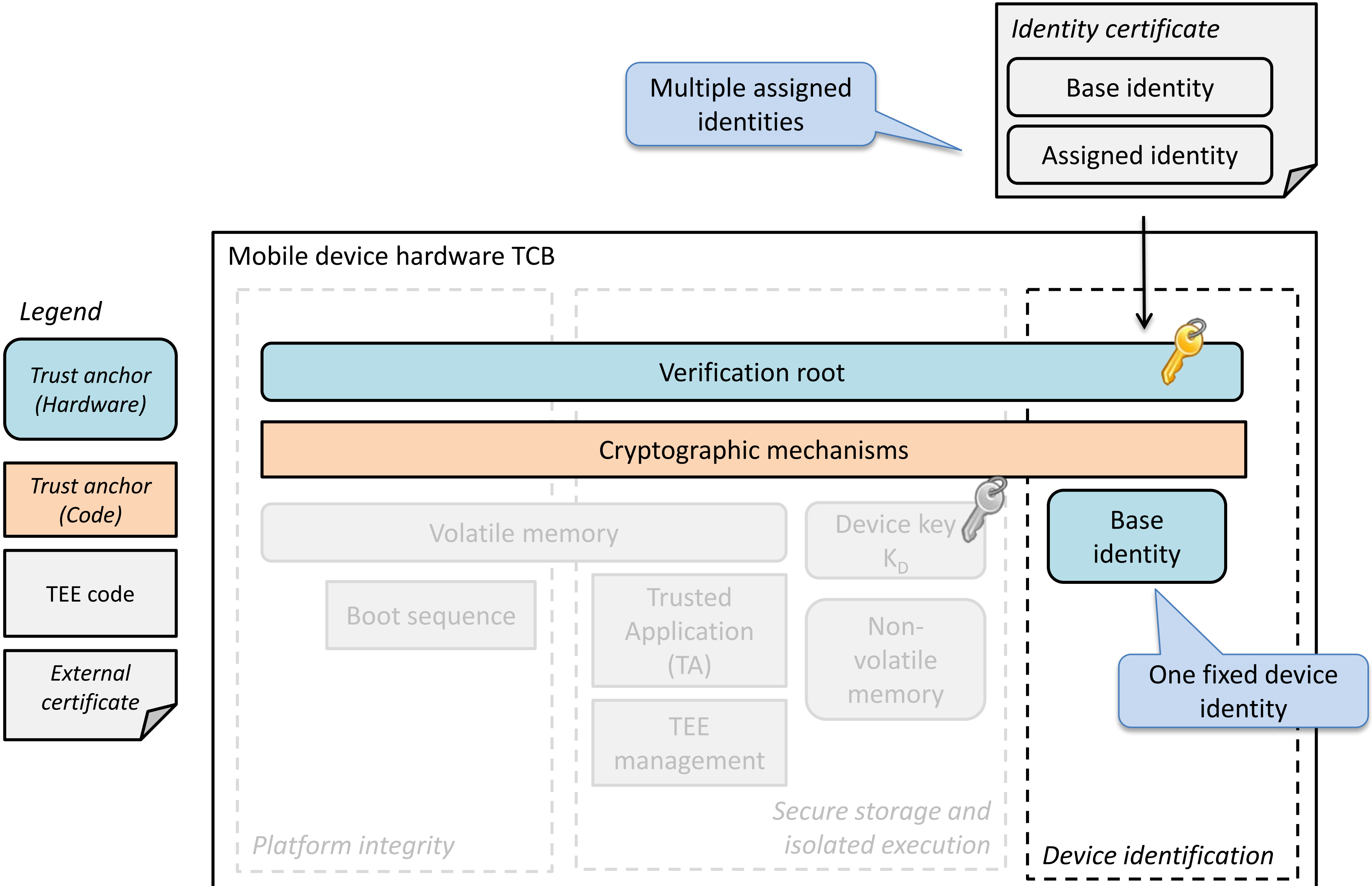
Secure storage



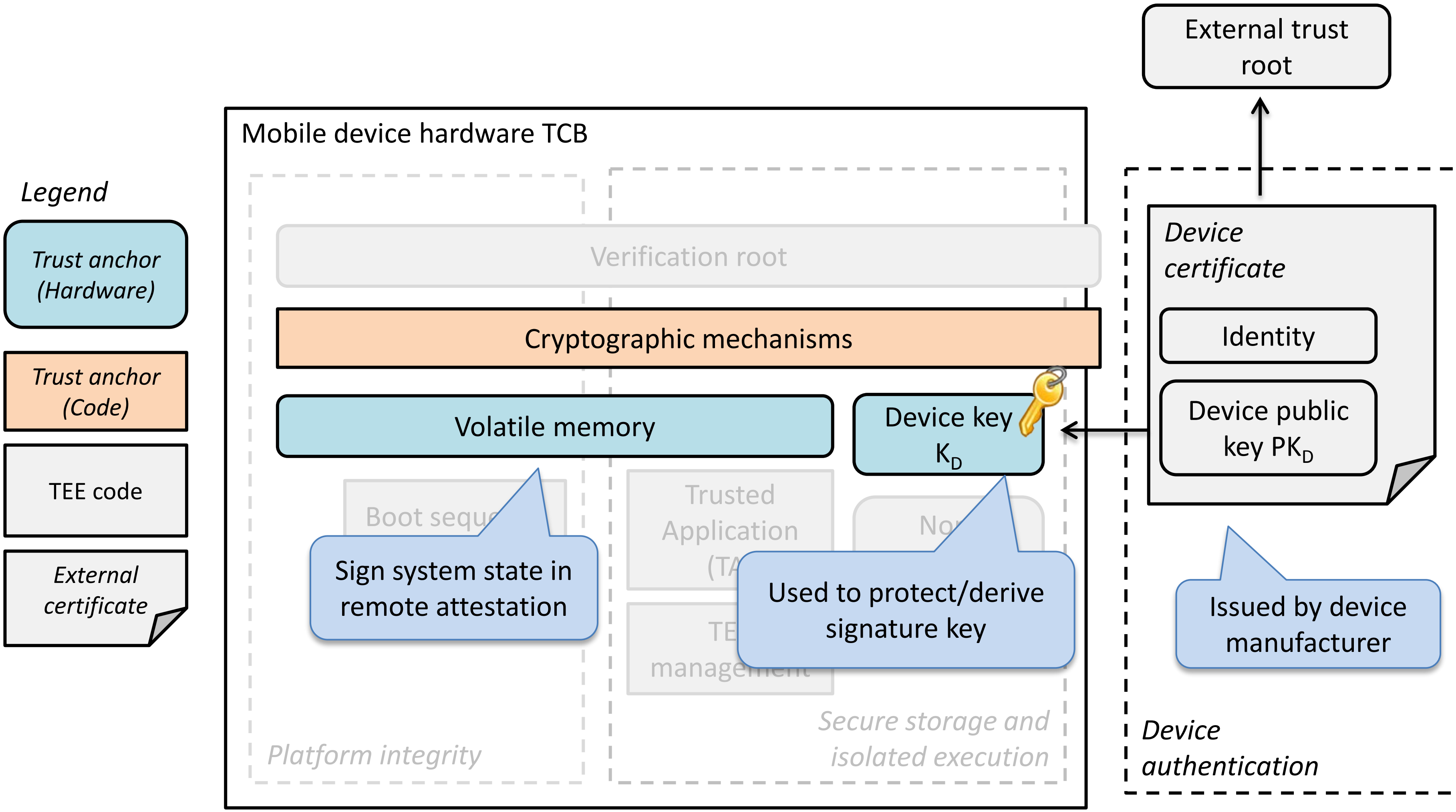
Isolated execution



Device identification

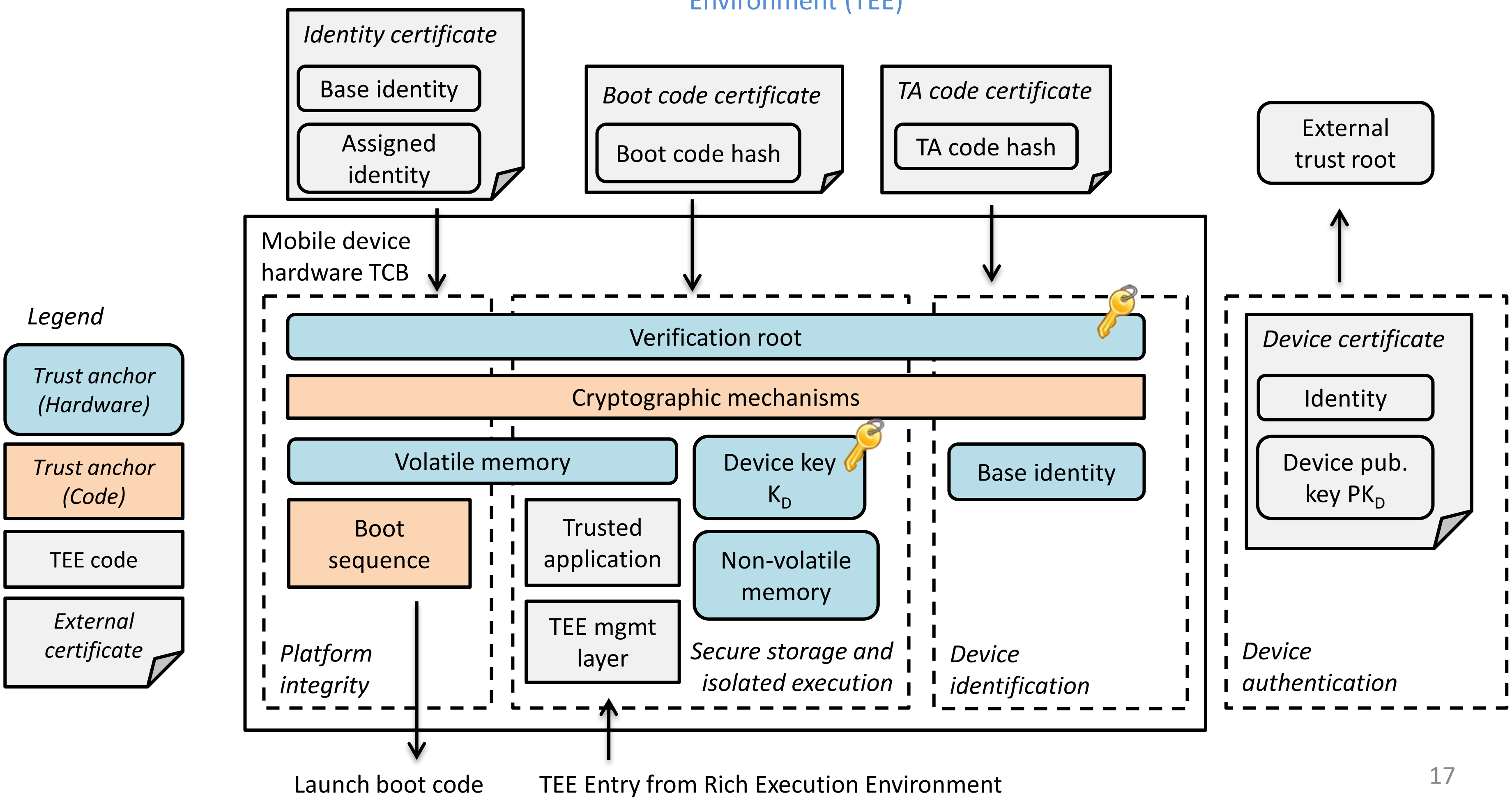


Device authentication (and remote attestation)

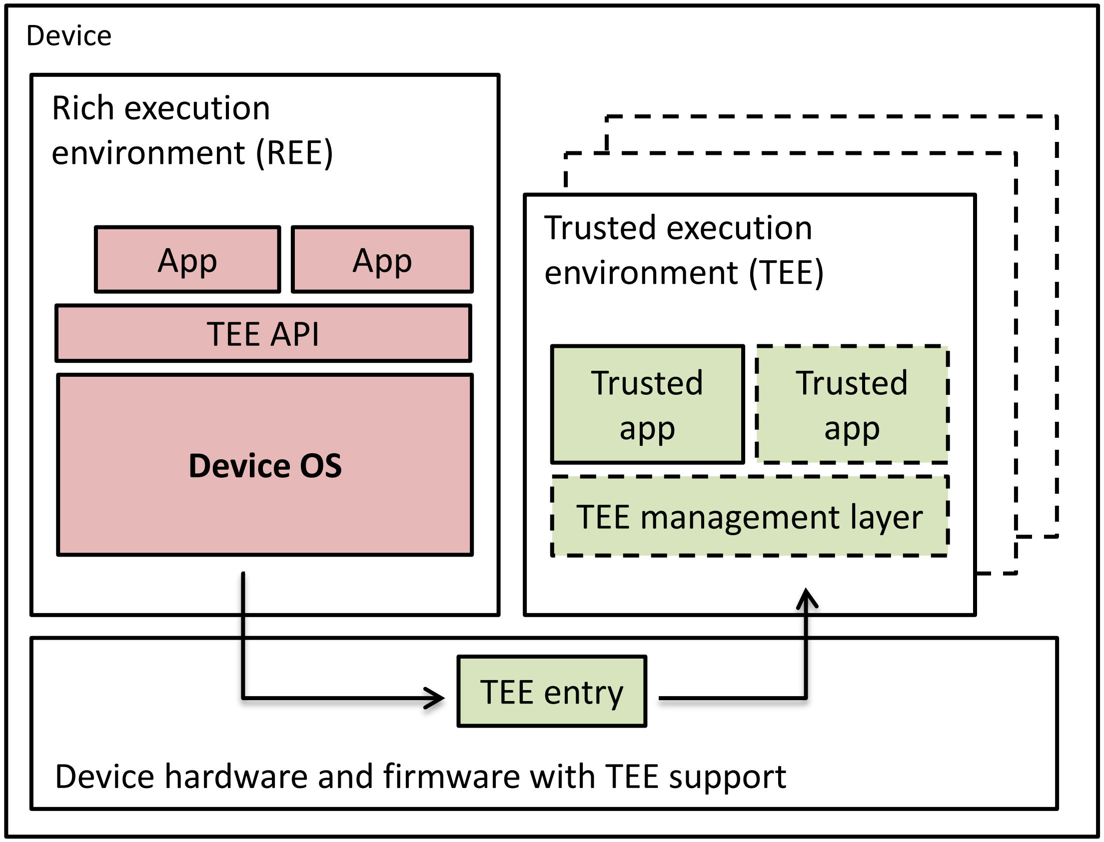


Hardware security mechanisms (recap)

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
- Trusted Platform Module (TPM)

Architectures with multiple TEEs

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

Figure adapted from: Global Platform. TEE system architecture. 2011.

Legend:
SoC : system-on-chip
OTP: one-time programmable

TEE hardware realization alternatives

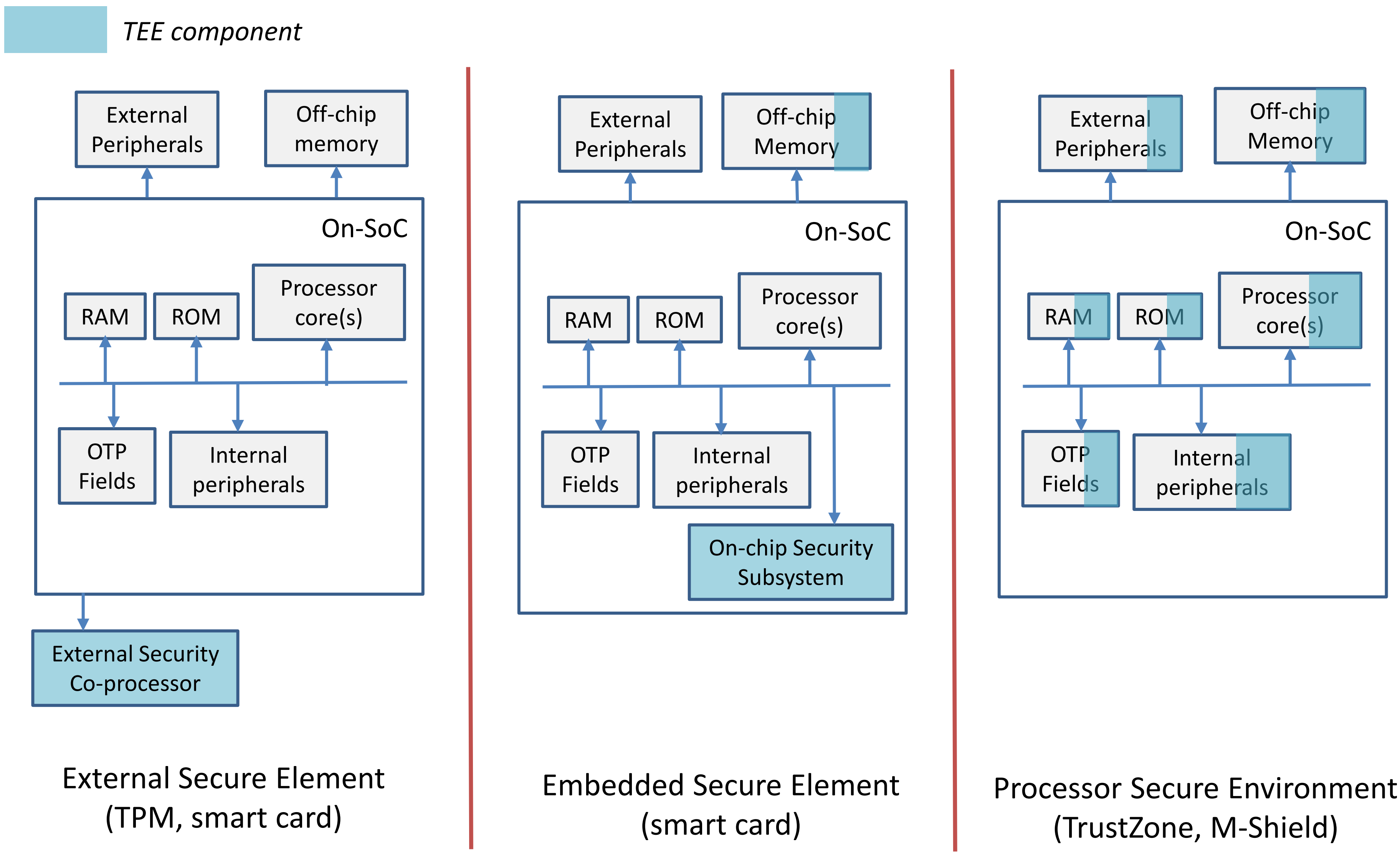
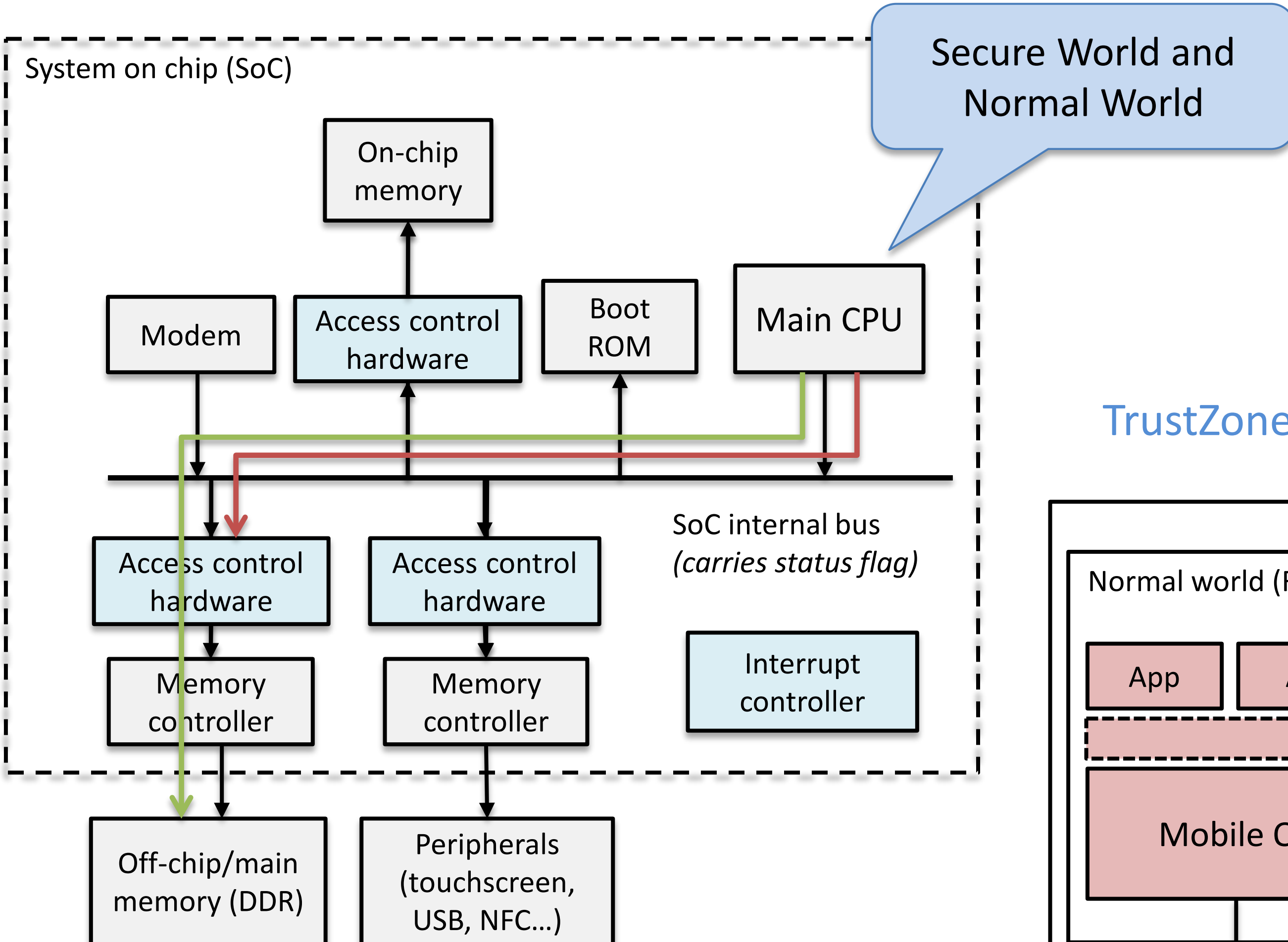


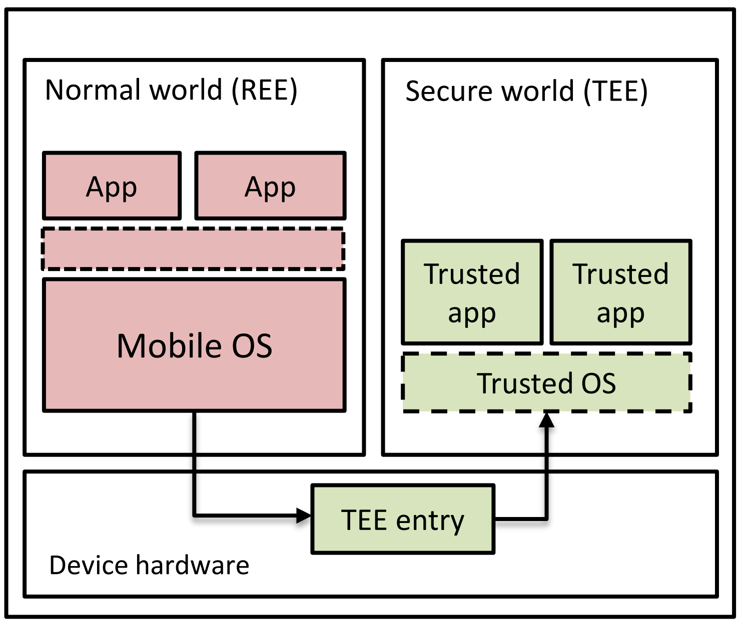
Figure adapted from: Global Platform. [TEE system architecture](). 2011.

ARM TrustZone architecture

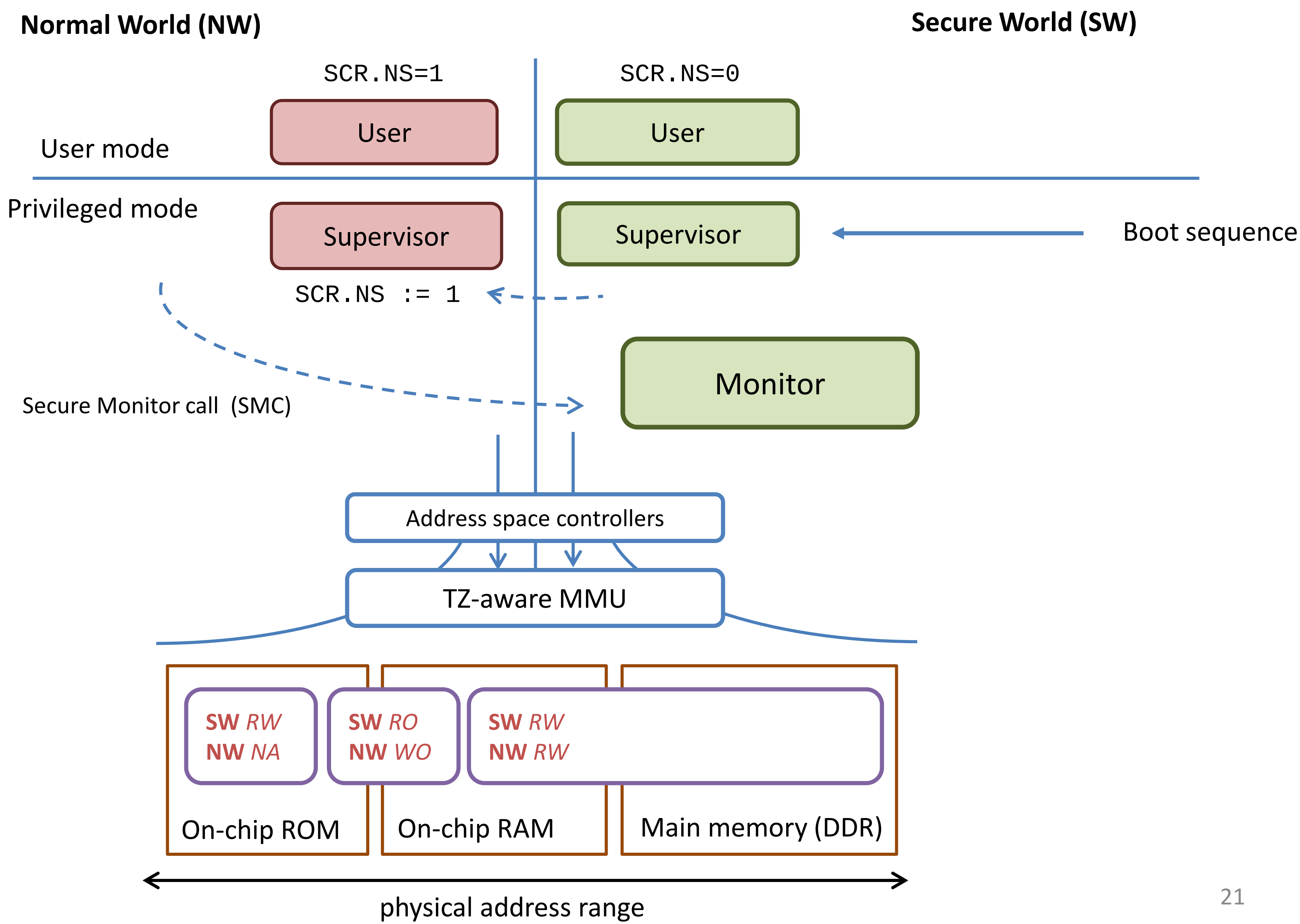


TrustZone hardware architecture

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 derive keys from on-chip ROM to on-chip RAM

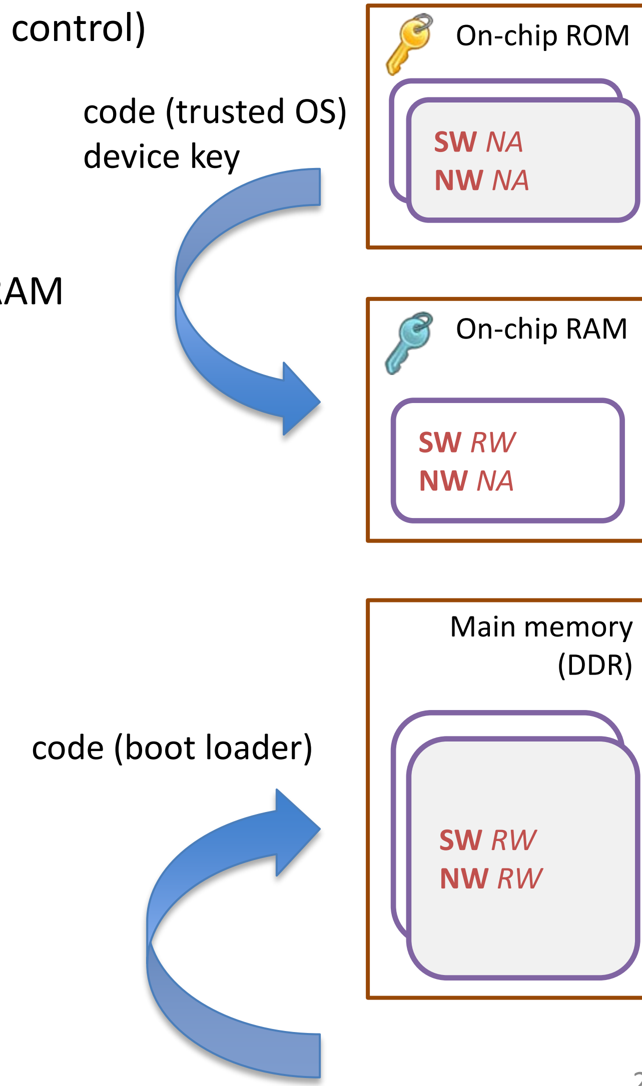
Secure World Supervisor

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

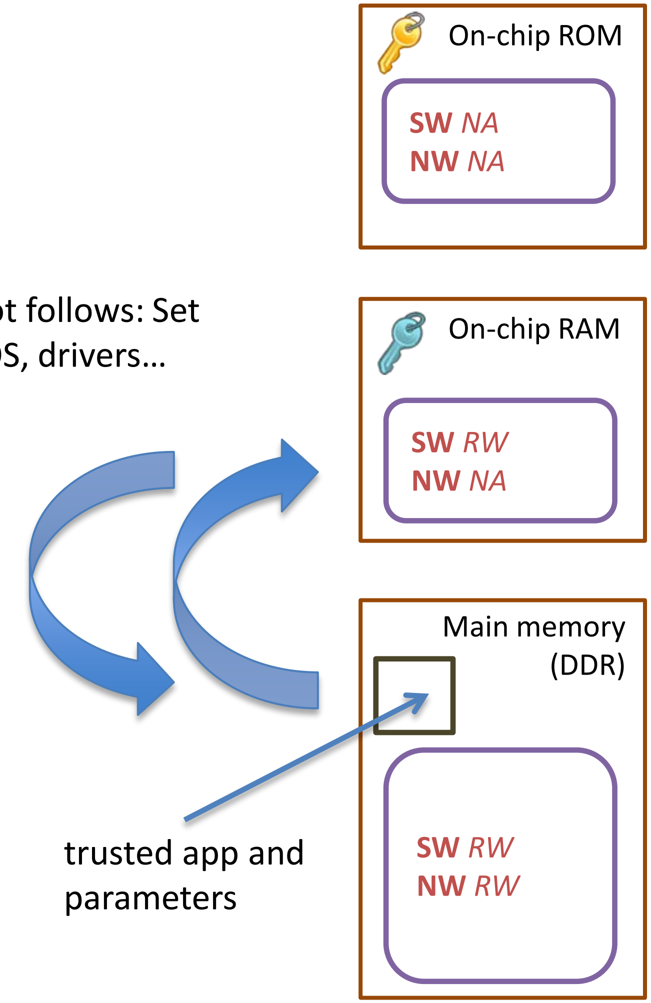
Supervisor

7. Execute trusted application

Normal World Supervisor

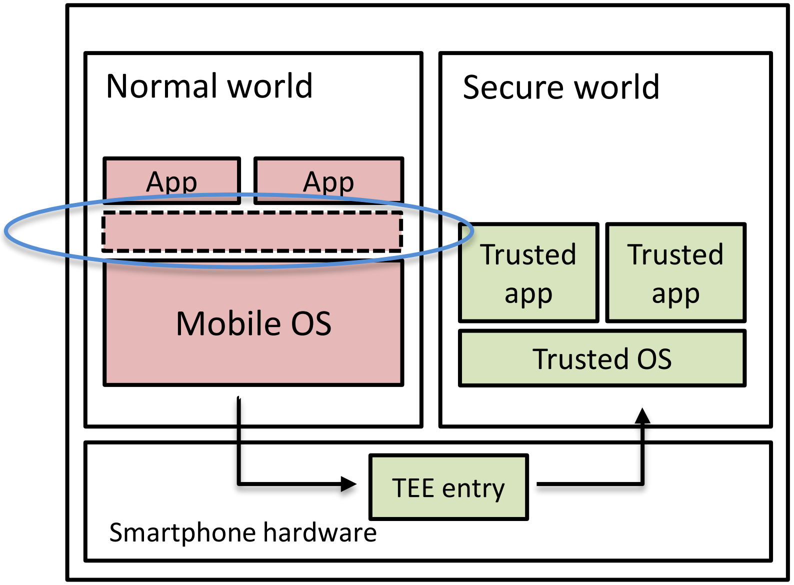
Secure World Monitor

SMC, NS→0



Mobile TEE deployment

- TrustZone support available in **majority** of current smartphones
- *Are there any APIs for developers?*



Mobile hardware security APIs

APPLICATION DEVELOPMENT

Mobile hardware security APIs

1. Secure element APIs:
(smart cards)

JSR 177

PKCS #11

2. Mobile hardware key stores:

iOS Key Store

Android Key Store

3. Programmable TEE
"credential platforms":

On-board Credentials

Trustonic TEE API

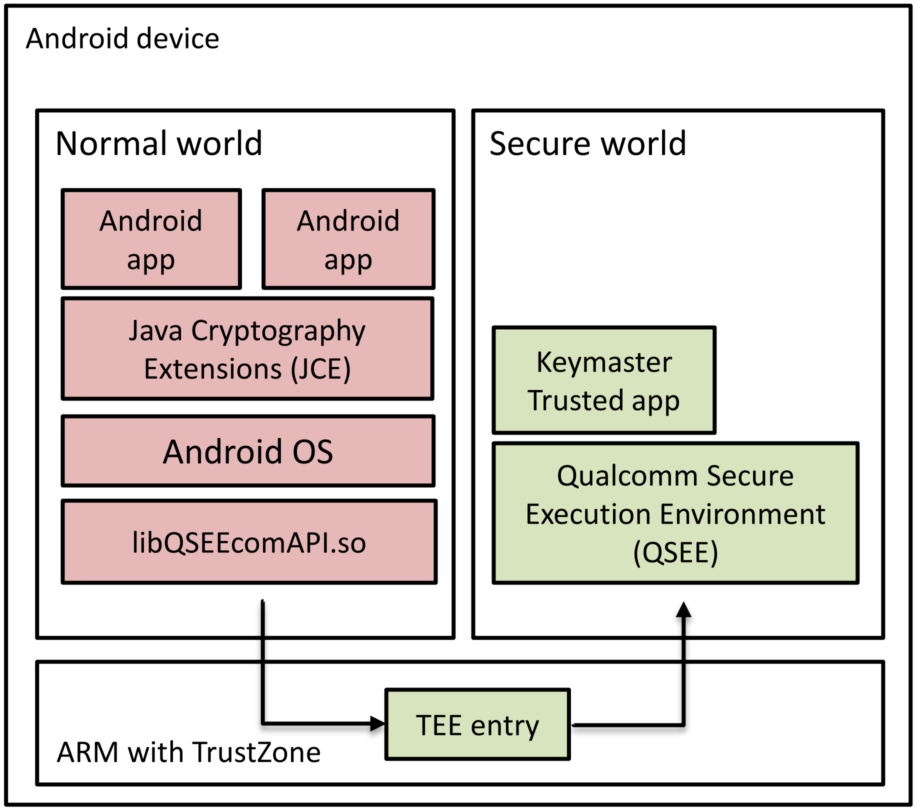
Android Key Store API

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.

Example 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.

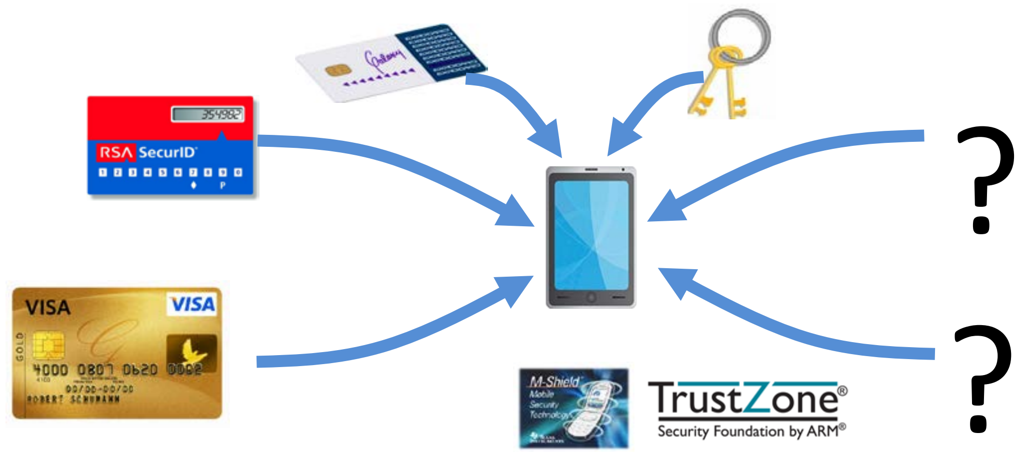
Android Key Store

- Only predefined operations
  - Signatures
  - Encryption/decryption
- Global Platform is standardizing TEE APIs
- Developers cannot utilize **programmability** of mobile TEEs
  - Not possible to run arbitrary trusted applications
  - (Same limitations hold for hardware protected iOS key store)
- Different API abstraction and architecture needed…
  - Example: On-board Credentials

Skip ObC

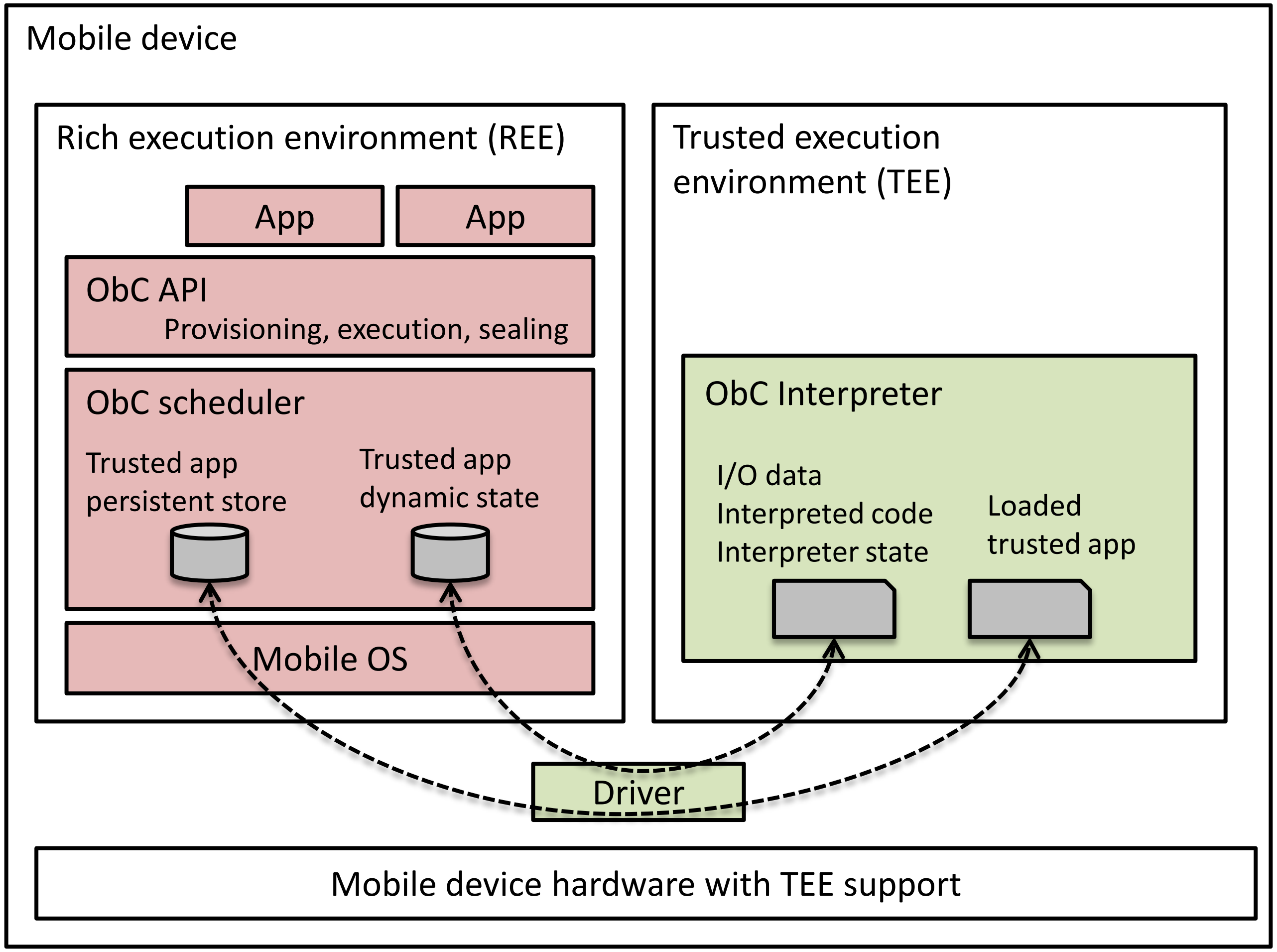
On-board Credentials goal

An open credential platform that enables existing mobile TEEs



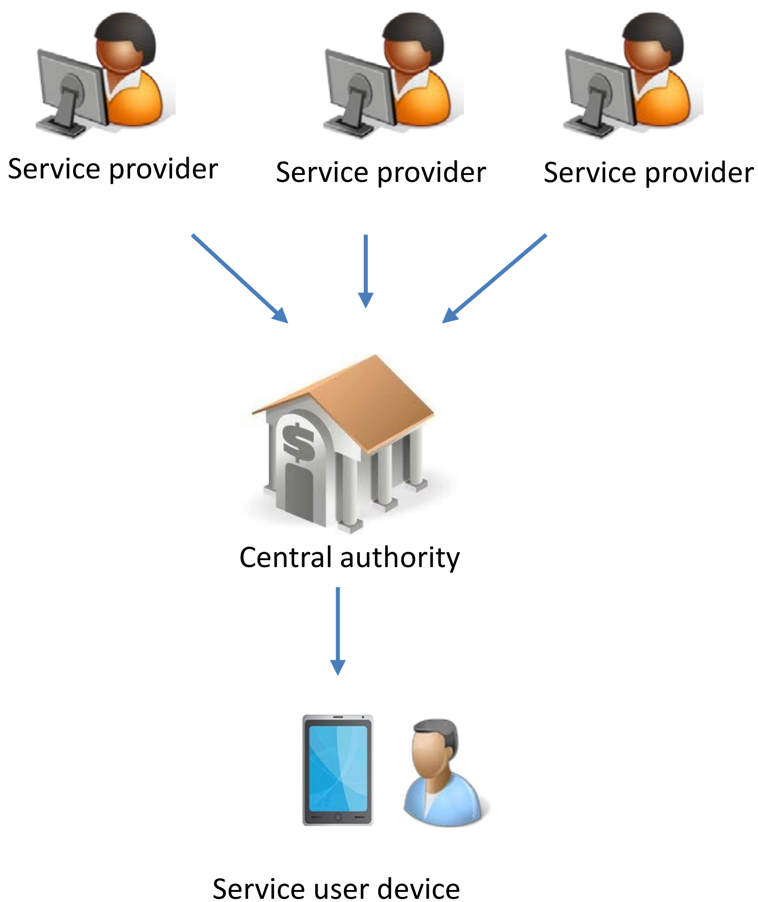
Secure yet inexpensive

On-board Credentials (ObC) architecture

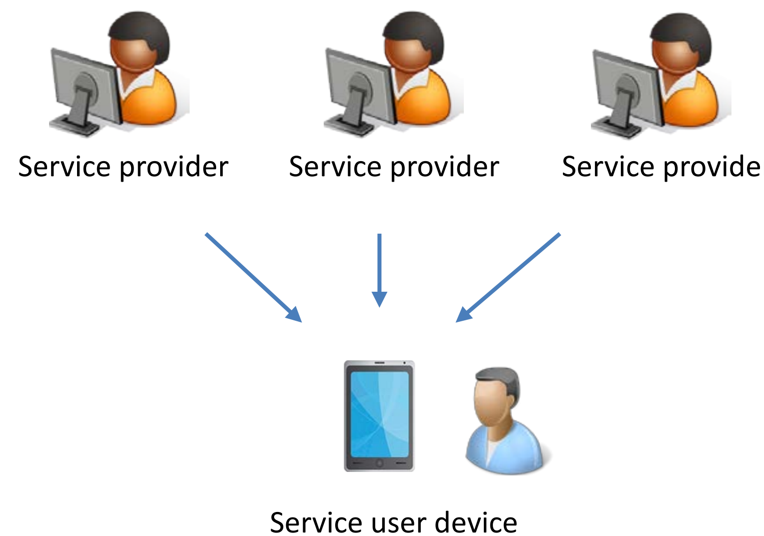


Ekberg. [Securing Software Architectures for Trusted Processor Environments](). Dissertation, Aalto University 2013.
Kostiainen. [On-board Credentials: An Open Credential Platform for Mobile Devices](). Dissertation, Aalto University 2012.

Centralized provisioning vs. open provisioning



Centralized provisioning
(smart card)



Open provisioning
(On-board Credentials)

Open provisioning model

Service provider

User device

Certified device key
PK

1. Certified device key + user authentication
PK ←

Pick new 'family key' FK
Encrypt family key
Enc(PK, FK)

2. Provision new family
Enc(PK, FK) →

establish new security domain (family)

Encrypt and authenticate secrets
AuthEnc(FK, secret)

3. Provision new secrets
AuthEnc(FK, secret) →

install secrets, associate them to family

Authorize trusted applications
AuthEnc(FK, hash(app))

4. Provision trusted applications
AuthEnc(FK, hash(app)) + app →

install trusted apps, grant access to secrets

Principle of same-origin policy

Skip to App. Development summary

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 trusted application extract

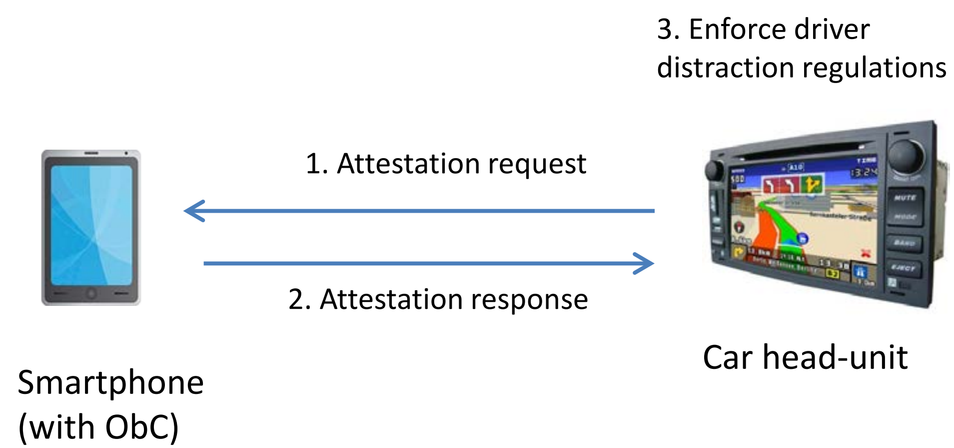
```
rem --- Quote operation
if mode == MODE_QUOTE
  read_array(IO_SEALED_RW, 2, pcr_10)
  read_array(IO_PLAIN_RW, 3, ext_nonce)

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)
```

ObC counterpart application pseudo code

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

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

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)



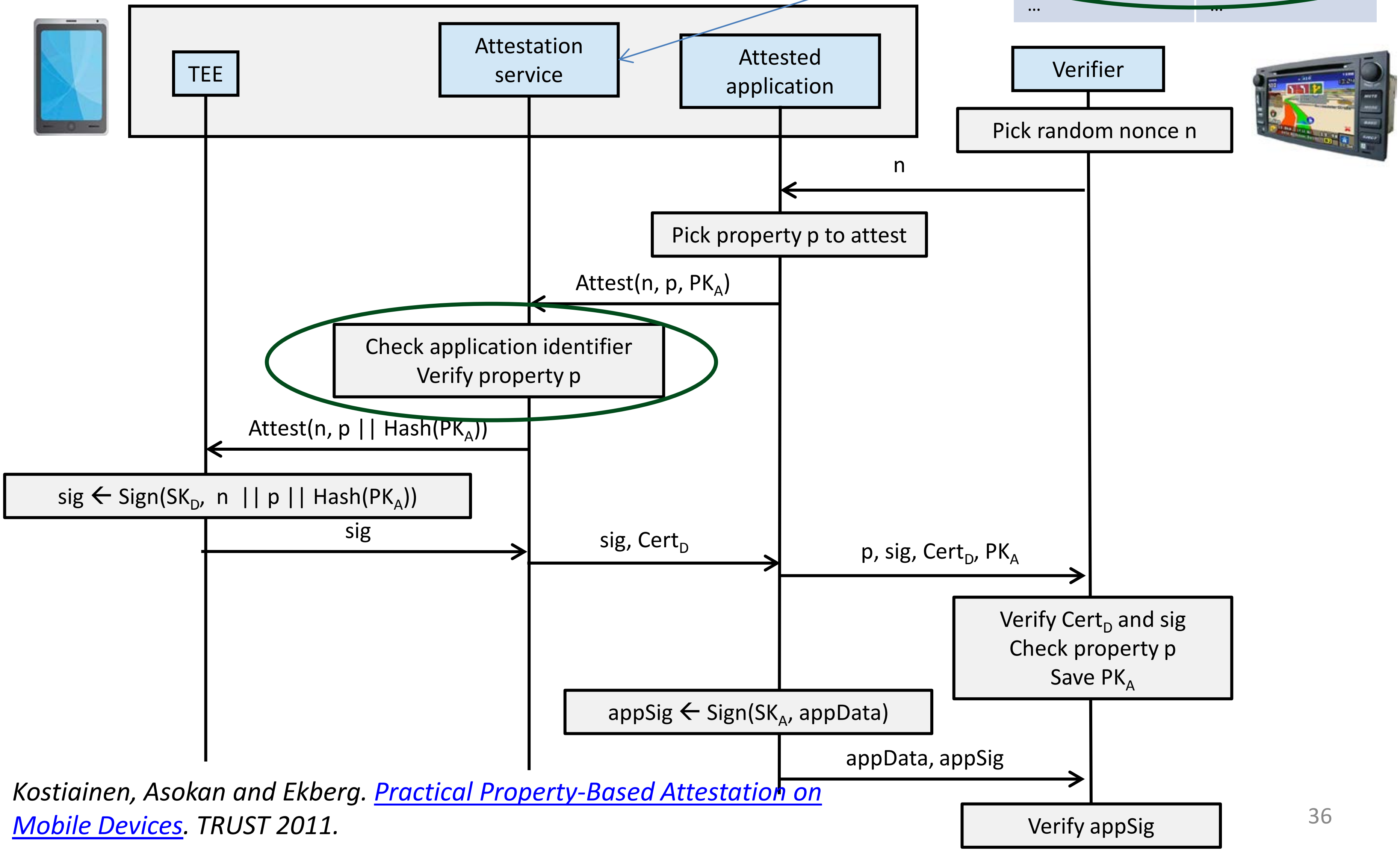
http://www.mirrorlink.com

CARCONNECTIVITY consortium

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

Attestation protocol

Application Identifier	Property
App1	P1, P2
App2	P3
…	…



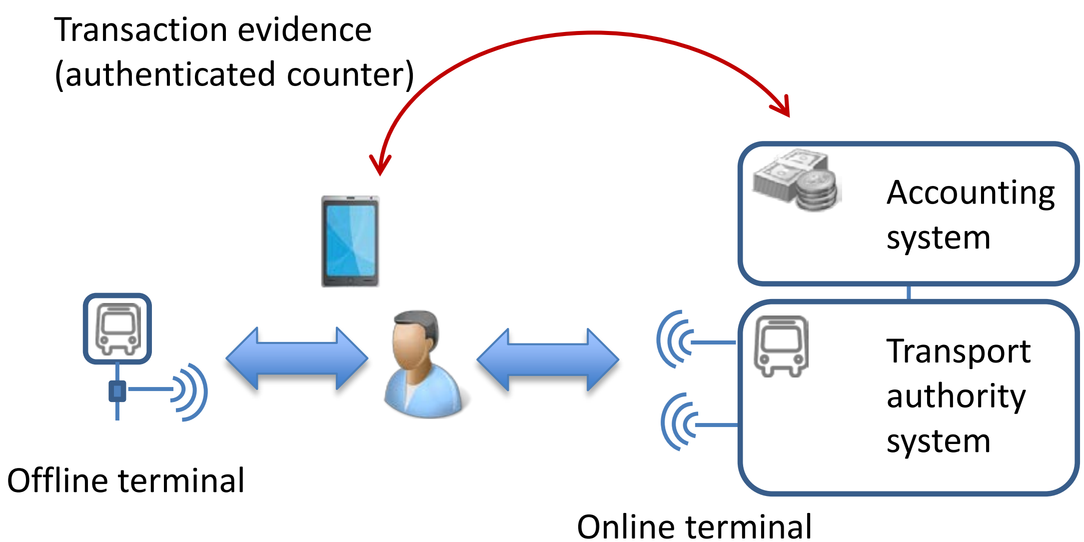
Kostiainen, Asokan and Ekberg. [Practical Property-Based Attestation on Mobile Devices](). TRUST 2011.

Example application: Public transport ticketing

- Mobile ticketing with NFC phones and TEE
  - Offline terminals at public transport stations
  - Mobile devices with periodic connectivity

    → Such **use case requires** ticketing protocol with **state keeping** (authenticated counters)

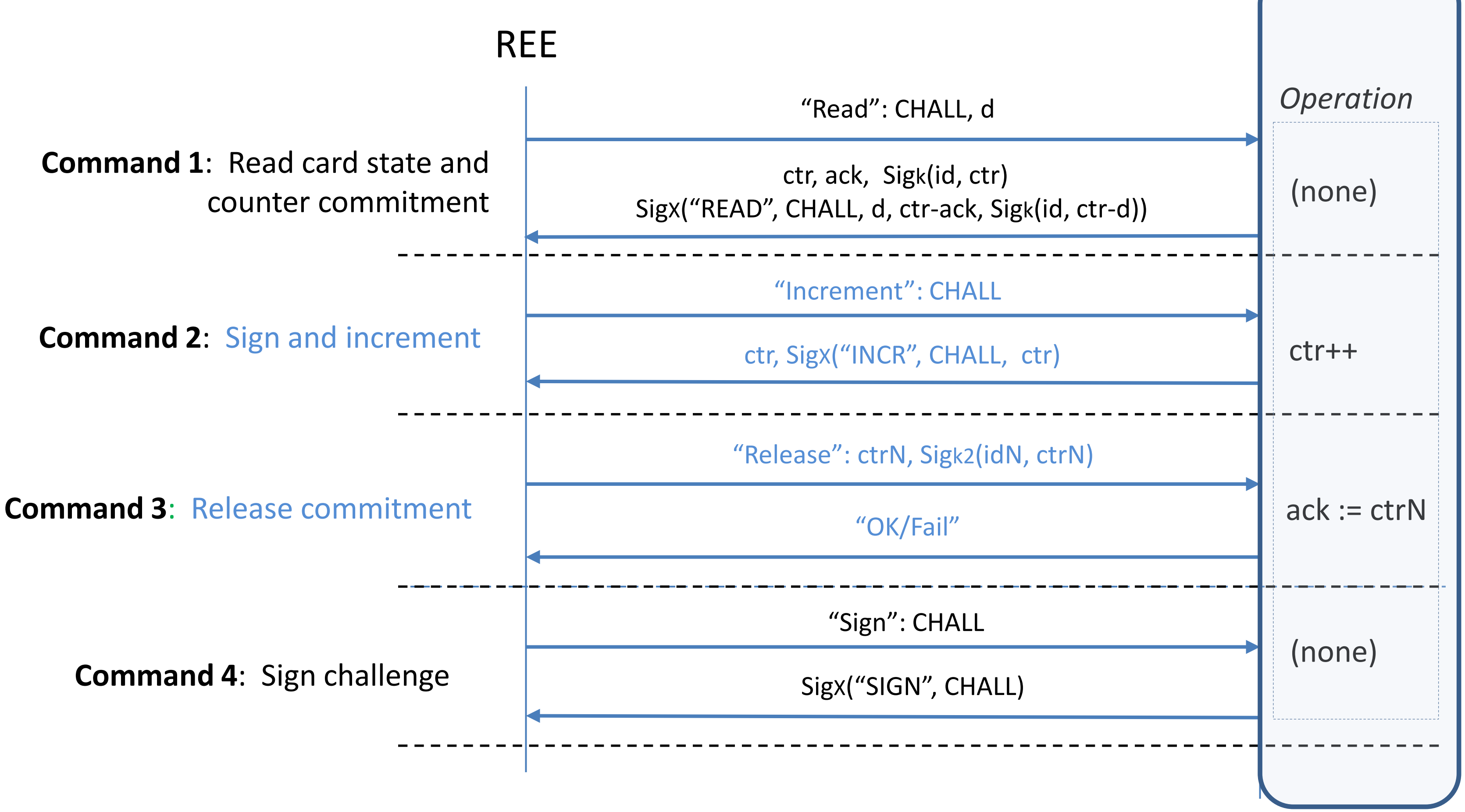
- 110 traveler trial in New York (summer 2012)
  - Implemented as On-board Credentials trusted application



Ekberg and Tamrakar. [Tapping and Tripping with NFC](). TRUST 2013

Transport ticketing protocol

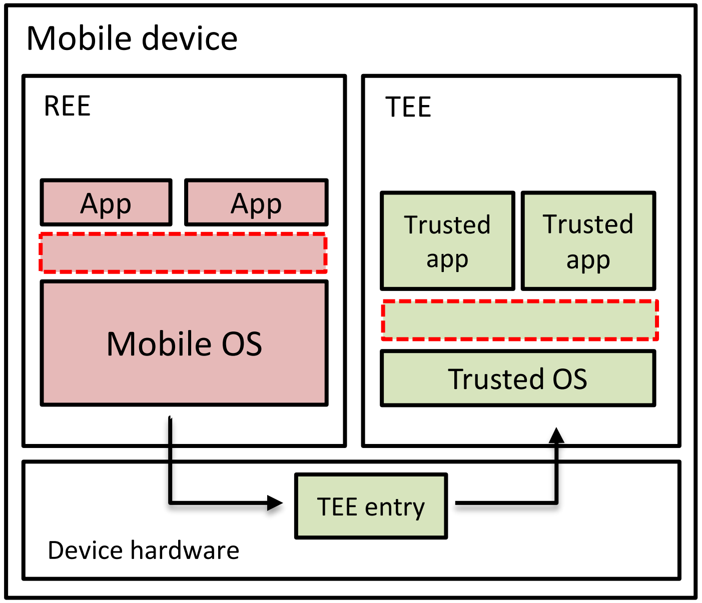
Authenticated counters implemented as an ObC program



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

Application development summary

- Mobile TEEs previously used mainly for internal purposes
  - DRM, subsidy lock
- Currently available third-party APIs enable only limited functionality
  - Signatures, decryption
  - Android key store
  - iOS key store
- Programmable TEE platforms
  - On-board Credentials
  - Demonstrates that mobile TEEs can be safely opened for developers



UEFI, NIST, Global Platform, Trusted Computing Group

STANDARDIZATION

TEE standards and specifications

- First versions of standards already out
- Goal: easier development and better interoperability

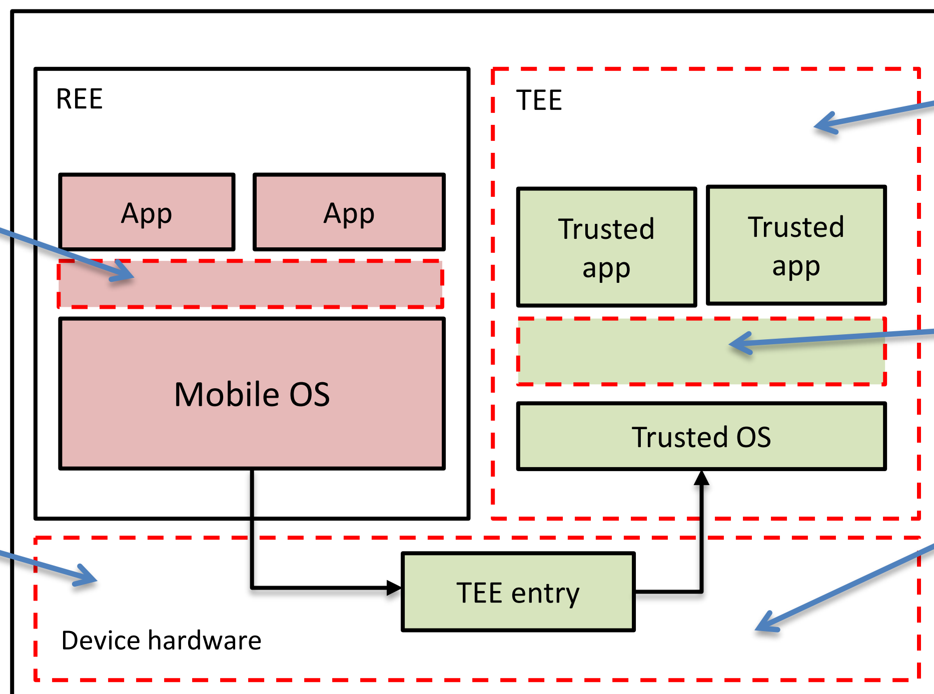
TEE environment

TrustZone
Security Foundation by ARM

REE app API

TRUSTED COMPUTING GROUP

GLOBALPLATFORM



TEE app API

GLOBALPLATFORM

Secure Boot

TRUSTED COMPUTING GROUP

NIST

Hardware trust roots

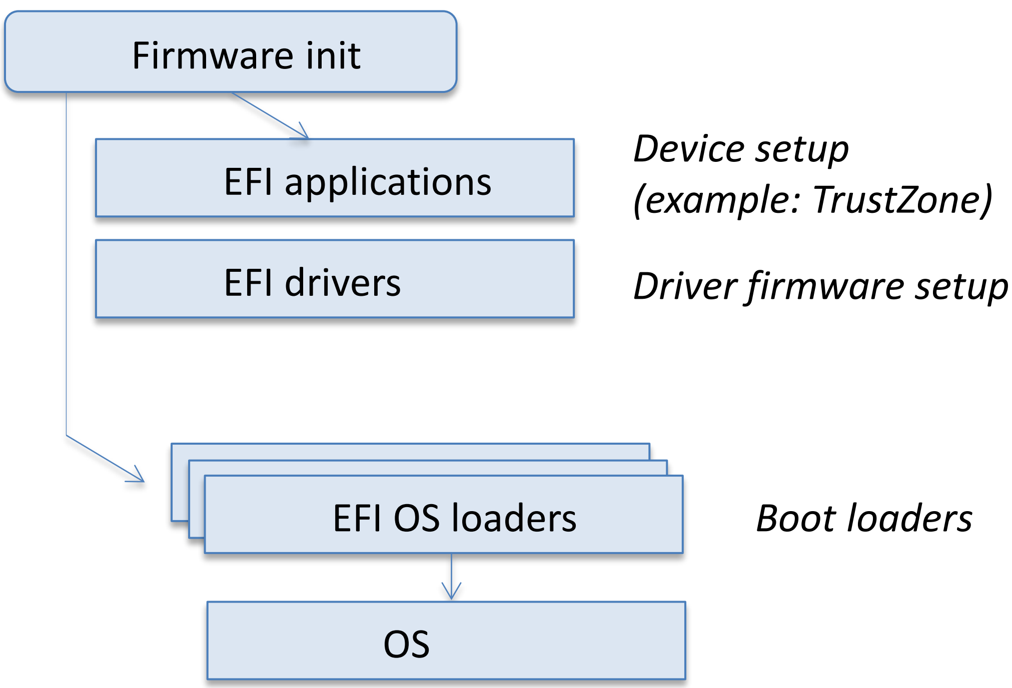
[Skip to Global Platform]

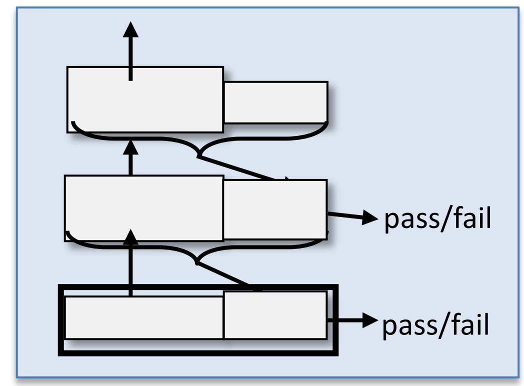
Secure Boot

UEFI

UEFI –boot principle

- UEFI standard intended as replacement for old BIOS
- Secure boot an optional feature





[Unified Extensible Firmware Interface Specification]()
[Nyström et al: UEFI Networking and Pre-OS security (2011)]()

UEFI – secure boot

Signature Database (s)

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

Key management for update

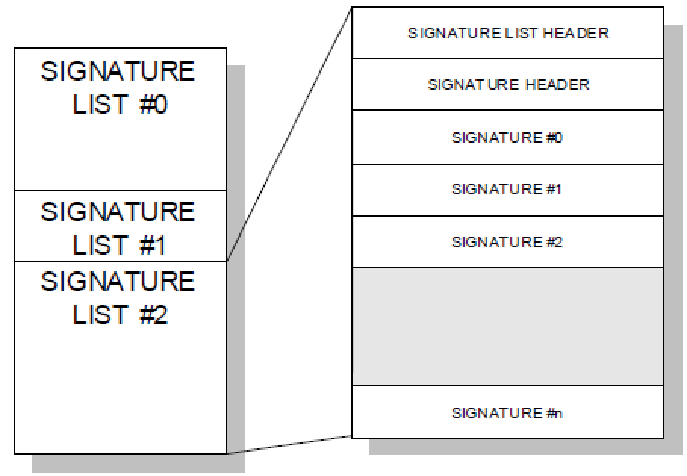
Platform Firmware Key Storage

→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

UEFI secure boot

- Thus far primarily used in PC platforms
  - Also applicable to mobile devices
- Can be used to limit user choice?
  - The specification defines user disabling
  - Policy vs. mechanism

Hardware-based Trust Roots for Mobile Devices

NIST

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

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

"RoTs are **preferably** implemented in hardware"

"the APIs **should** be standardized"

Roots of Trust (RoTs)

Root of Trust for Storage (RTS): repository and a protected interface to store and manage keying material

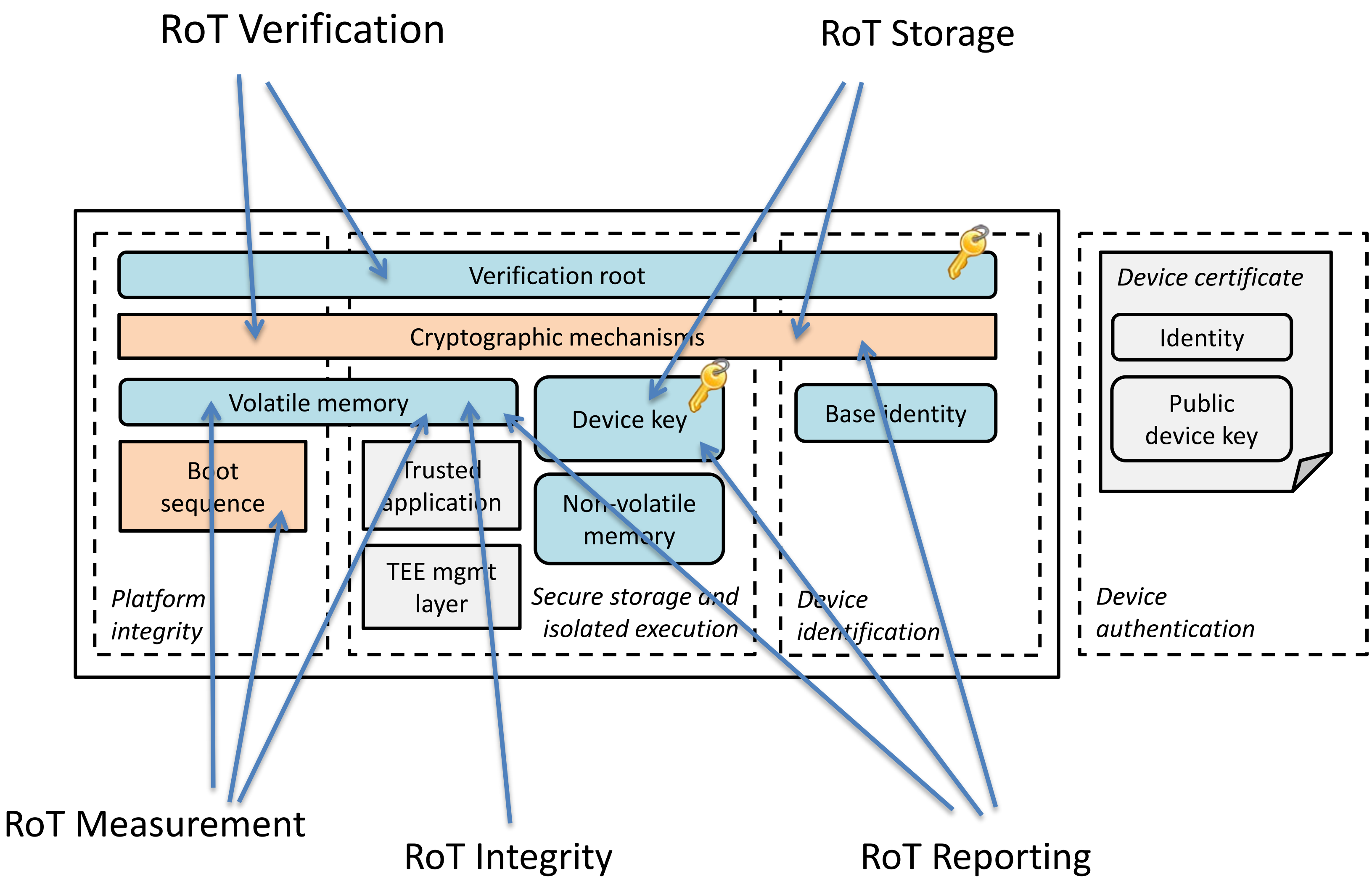
Root of Trust for Measurement (RTM): reliable measurements and assertions

Root of Trust for Verification (RTV): engine to verify digital signatures associated with software/firmware

Root of Trust for Integrity (RTI): run-time protected storage for measurements and assertions

Root of Trust for Reporting (RTR): environment to manage identities and sign assertions

Root of Trust mapping



Trusted Execution Environment (TEE) specifications

GLOBAL PLATFORM

Global Platform (GP)

GP standards for smart card systems used many years

- Examples: payment, ticketing
- Card interaction and provisioning protocols
- Reader terminal architecture and certification

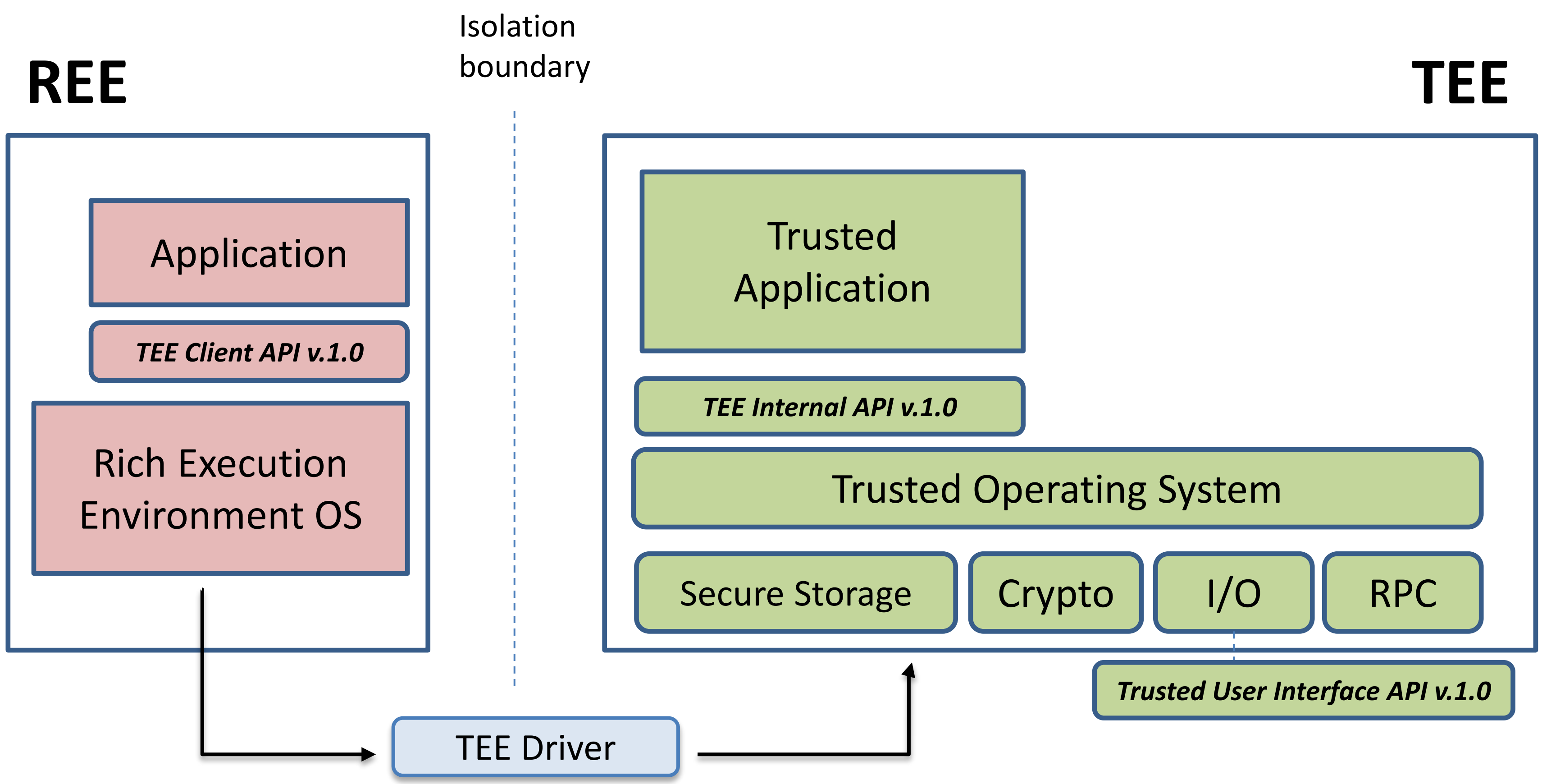
Recently GP has released standards for mobile TEEs

- Architecture and interfaces

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

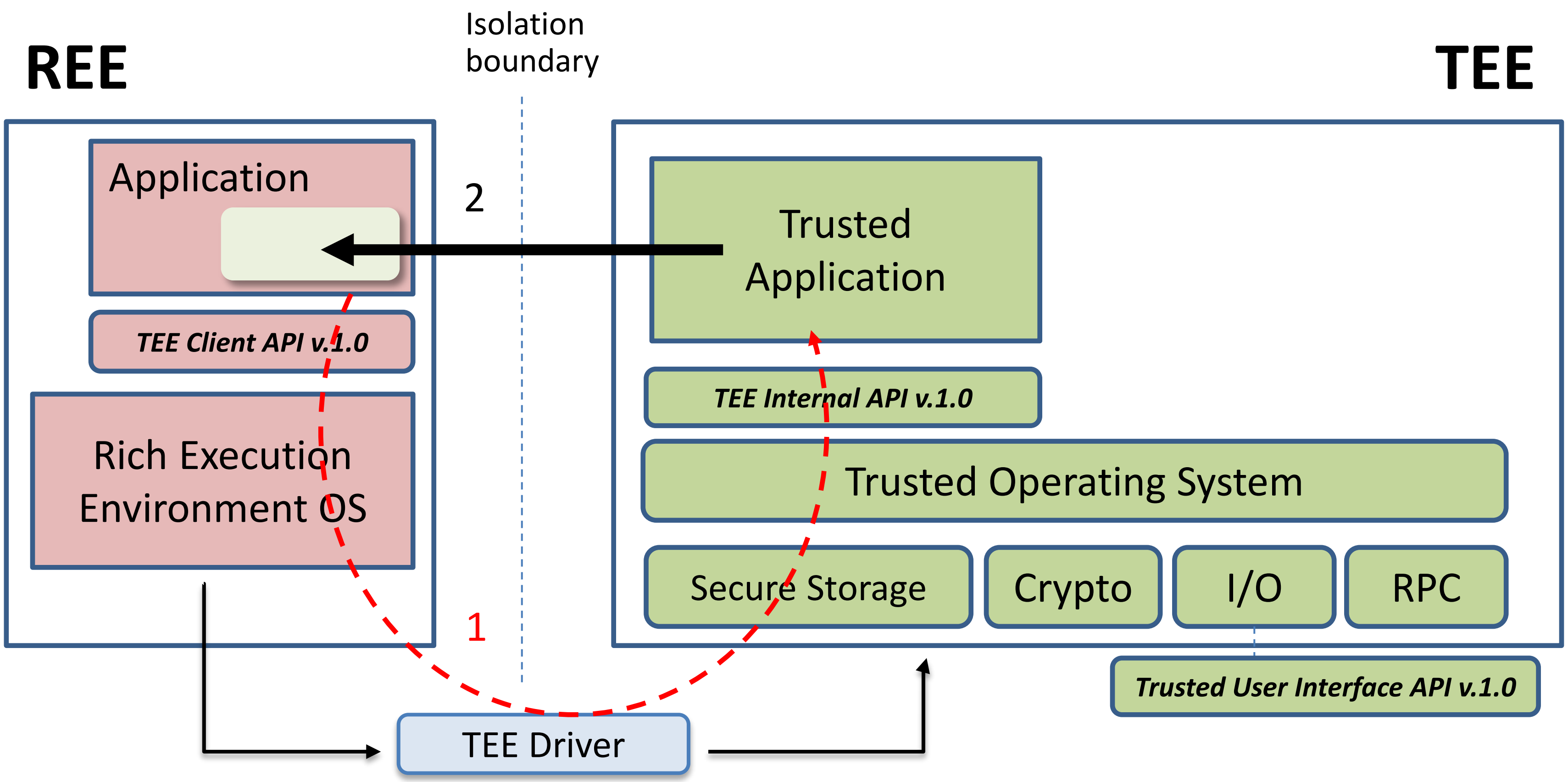
GP TEE System Architecture



Interaction with Trusted Application

REE App provides a pointer to its memory for the Trusted App

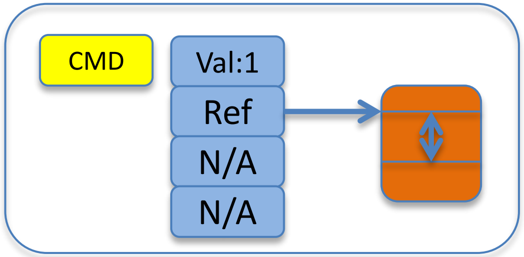
- Example: Efficient in place encryption



TEE Client API example

```
// 1. initialize context
TEEC_InitializeContext(&context, …);

// 2. establish shared memory
sm.size = 20;
sm.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
TEEC_AllocateSharedMemory(&context, &sm);

// 3. open communication session
TEEC_OpenSession(&context, &session, …);

// 4. setup parameters
operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, …);
operation.params[0].value.a = 1;           // First parameter by value
operation.params[1].memref.parent = &sm;   // Second parameter by reference
operation.params[1].memref.offset = 0;
operation.params[1].memref.size = 20;

// 5. invoke command
result = TEEC_InvokeCommand(&session, CMD_ENCRYPT_INIT, &operation, NULL);
```

Parameters:



TEE Internal API example

```
// each Trusted App must implement the following functions…

// constructor and destructor
TA_CreateEntryPoint();
TA_DestroyEntryPoint();

// new session handling
TA_OpenSessionEntryPoint(uint32_t param_types, TEE_Param params[4], void **session)
TA_CloseSessionEntryPoint (…)

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

In Global Platform model Trusted Applications are command-driven

Storage and RPC (TEE internal API)

Secure storage: Trusted App can persistently store memory and objects

```
TEE_CreatePersistentObject(TEE_STORAGE_PRIVATE, flags, ..., handle)

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

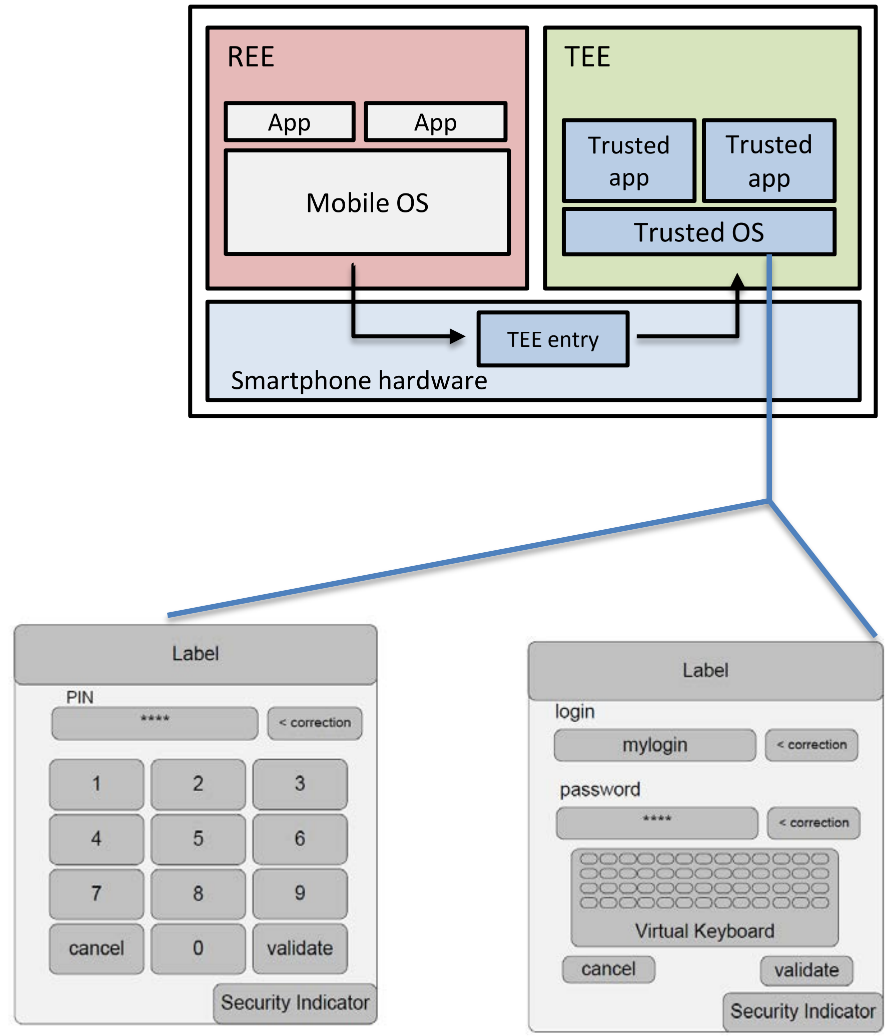
```
TEE_OpenTASession(TEE_UUID* destination, …,  paramTypes, params[4], &session);
TEE_InvokeTACommand(session, …, commandId, paramTypes, params[4]);
```

Also APIs for **crypto**, **time**, and **arithmetic** operations…

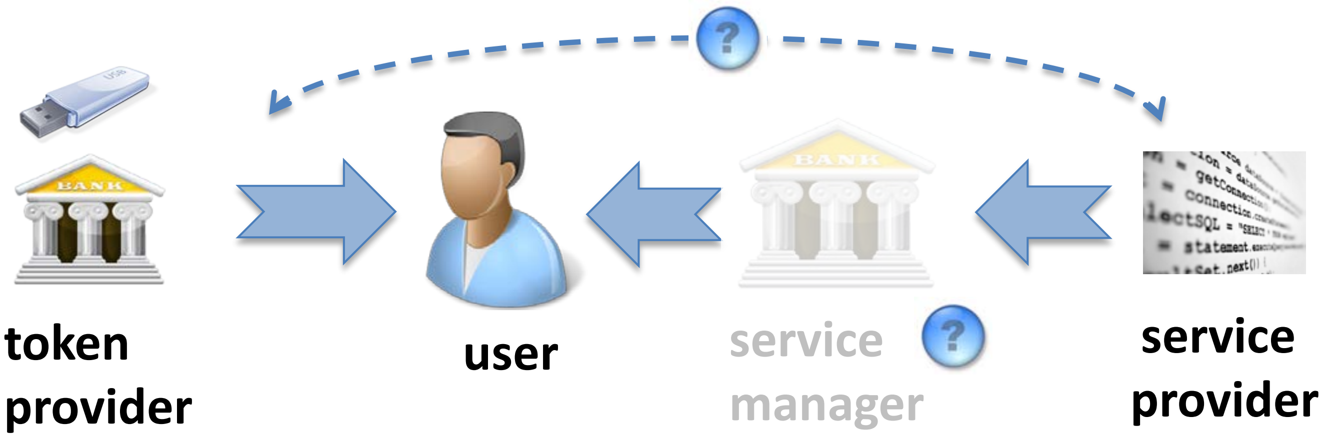
Trusted User Interface API

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

- Trusted User Interface API 1.0:
  - TEE_TUIDisplayScreen



Global Platform User-centric provisioning



GP device committee is working on a TEE provisioning specification

[User-centric provisioning white paper]

GP standards summary

- Specifications provide sufficient basis for TA development

- Issues
  - Application installation (provisioning) model not yet defined
  - Access to TEE typically controlled by the manufacturer
  - User interaction

- Open TEE
  - Virtual TEE platform for prototyping and testing
  - Implements GP TEE interfaces
  - https://github.com/Open-TEE

TPM 1.2 and TPM 2.0 EA

TRUSTED COMPUTING GROUP

Trusted Platform Module (TPM)

- Collects state information about a system
  - separate from system on which it reports

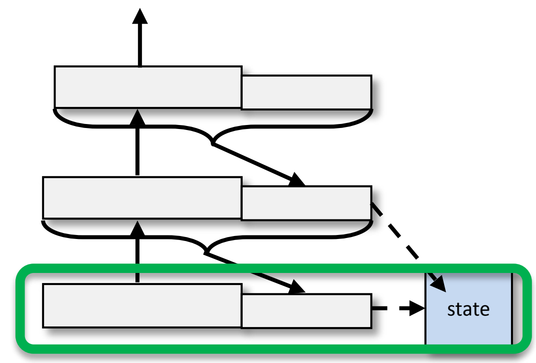
- For remote parties
  - **Remote attestation** in well-defined manner
  - **Authorization** for functionality provided by the TPM

- Locally
  - **Key generation** and **key use** with TPM-resident keys
  - **Sealing:** Secure **binding** with **non-volatile storage**
  - **Engine** for cryptographic operations

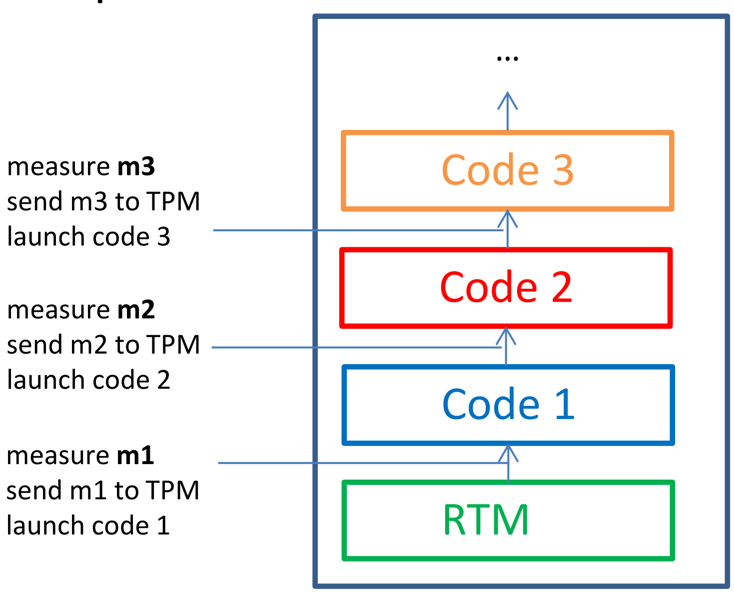
Platform Configuration Registers (PCRs)

- Integrity-protected registers
  - in volatile memory
  - represent current system configuration

- Store aggregated platform "state" measurement
  - a given state reached ONLY via the correct extension sequence
  - Requires a root of trust for measurement (RTM)



Authenticated boot



$H_{new}=H(H_{old} \mid new)$

$H_0= 0$

$H_3=H (H (H (0|m1) |m2)|m3)$

Use of platform measurements (1/2)

Remote attestation

- verifier sends a challenge
- attestation is SIG_{AIK}(challenge, PCRvalue)

- AIK is a unique key specific to that TPM ("Attestation Identity Key")

- attests to current system configuration

Use of platform measurements (2/2)

Sealing

- bind secret data to a specific configuration

- E.g.,: Create RSA key pair PK/SK when PCR_X value is Y

- Bind private key: $Enc_{SRK}(SK, PCR_X=Y)$
  - SRK is known only to the TPM
  - "Storage Root Key"

- TPM will "unseal" key **only** if PCR_X value is Y
  - Y is the "reference value"

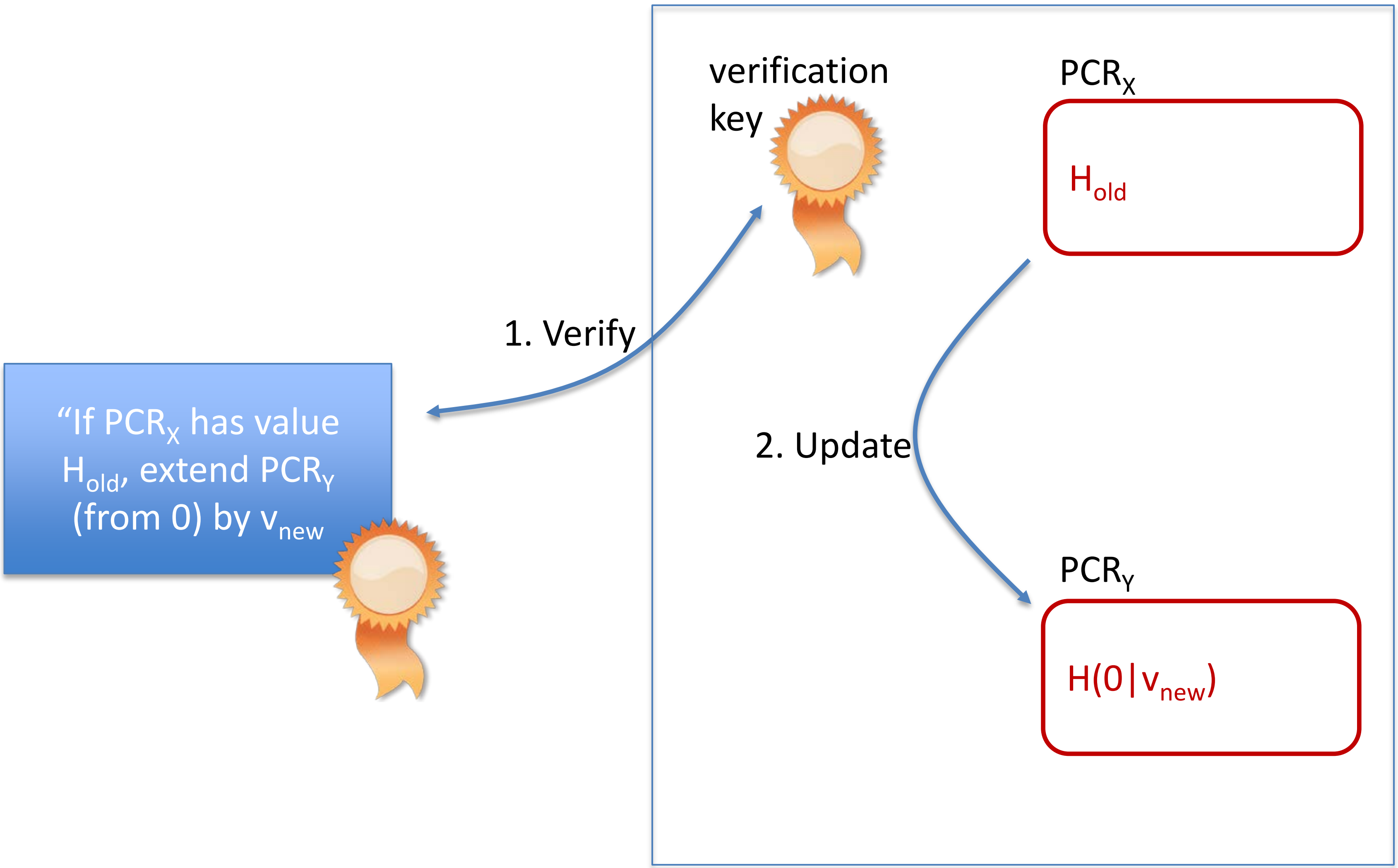
TPM Mobile (Mobile Trusted Module)

A TPM profile for Mobile devices that adds mechanisms for

- **Adaptation to TEEs:**
  - New roots of trust definitions and requirements

- **Multi-Stakeholder Model (MSM):**
  - ”Certified boot”: Secure boot with TCG authorizations
    - Reference Integrity Metric (RIM) certificates:
      - ”if PCR_X matches *reference*, extend PCR_Y by *target*”

RIM Certificate



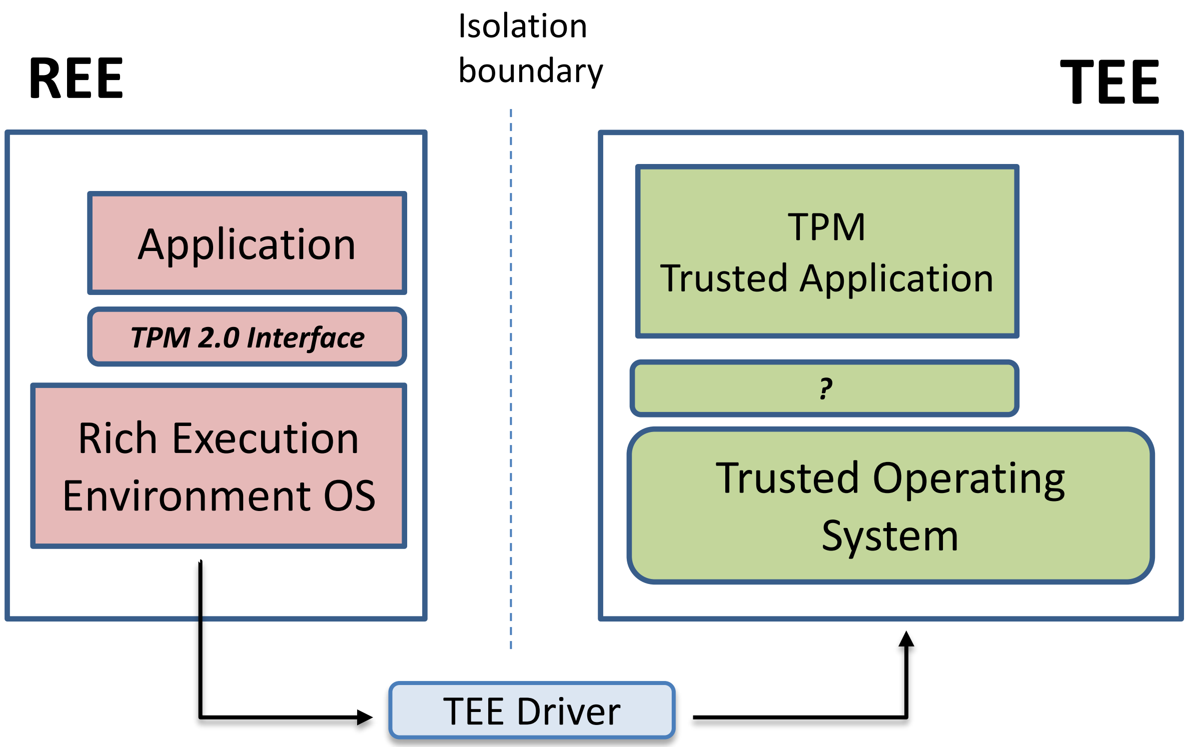
TPM 2.0

- Recent specification, in public review
  - Algorithm agility
  - New enhanced authorization model
  - "Library specification"
    - → Defines interface, not physical security chip
    - → Intended for various devices (not only PCs)

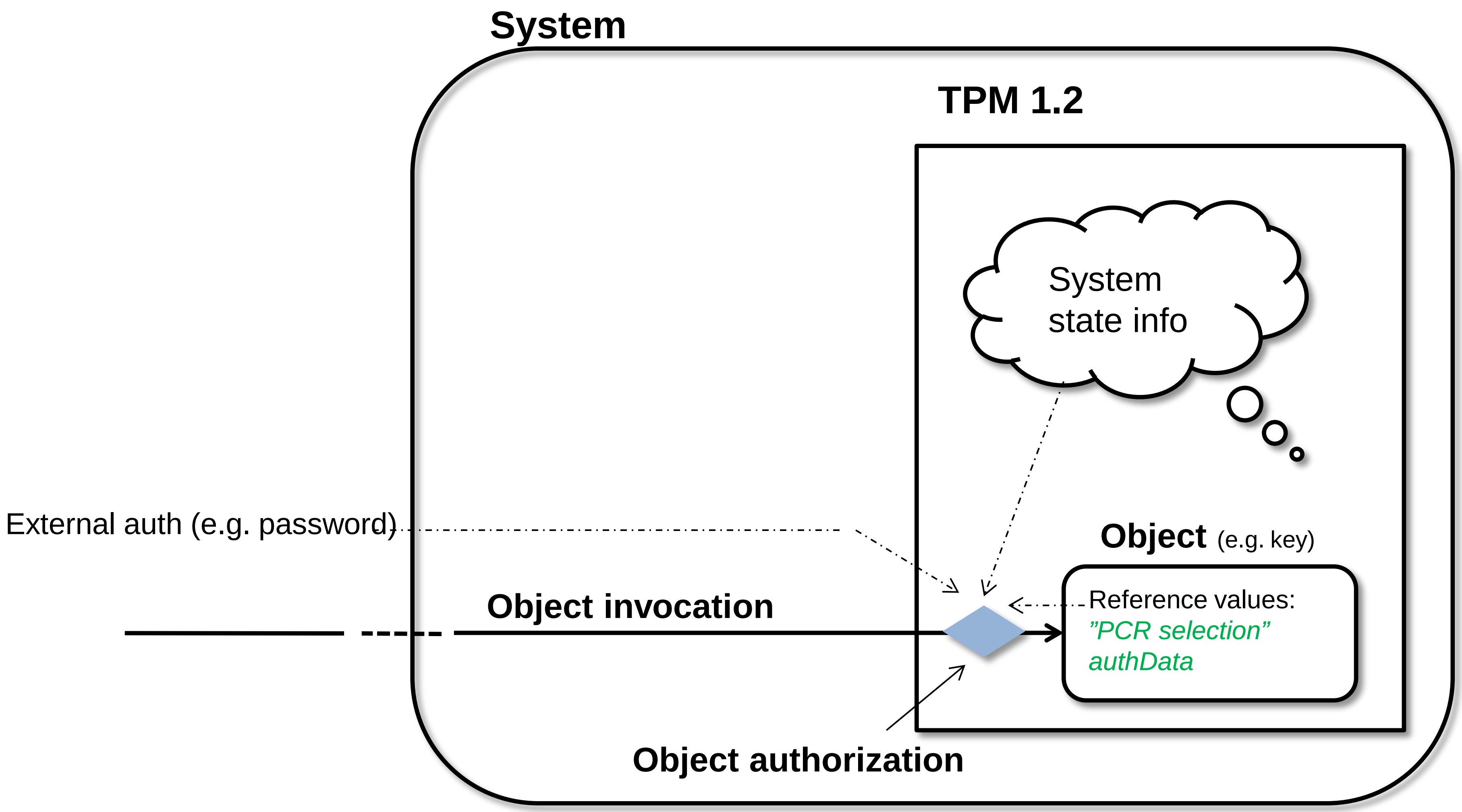
TPM 2.0 Mobile Reference Architecture

"Protected Environment"

- "the device SHALL implement Secure Boot"
- "the Protected Environment SHALL provide isolated execution"

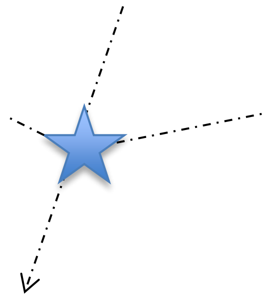


Authorization (policy) in TPM 1.2

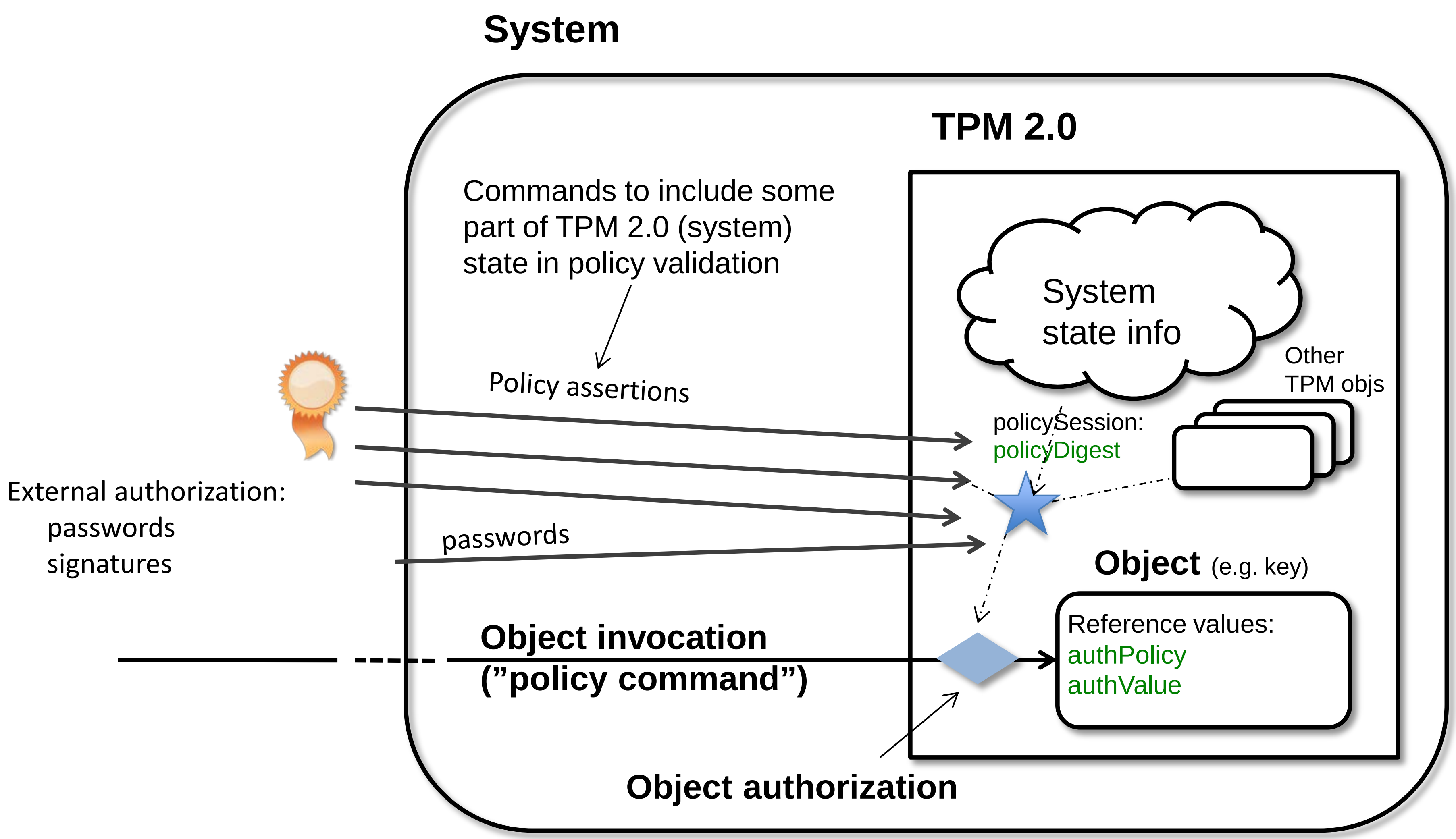


TPM 2.0

- More expressive policy definition model
- Various policy preconditions
- Logical operations (AND, OR)
- A policy session accumulates all authorization information



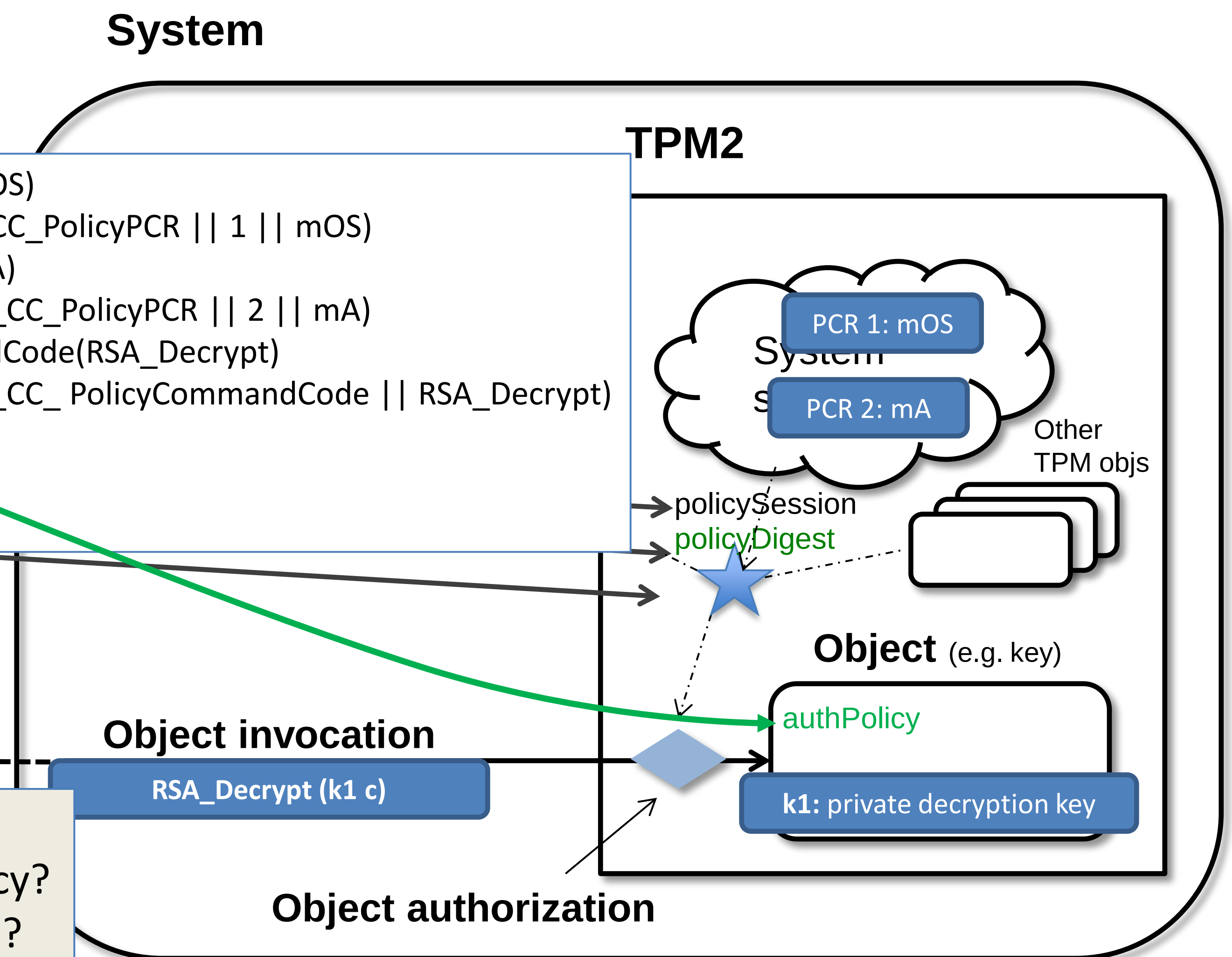
Authorization (policy) in TPM 2.0



An Example

Command sequence

v11 <- TPM2_PolicyPCR(1, mOS)
    // v11 = h (0 || TPM_CC_PolicyPCR || 1 || mOS)
v12 <- TPM2_PolicyPCR(2, mA)
    // v12 = h (v11 || TPM_CC_PolicyPCR || 2 || mA)
v13 <- TPM2_PolicyCommandCode(RSA_Decrypt)
    // v13 = h (v12 || TPM_CC_ PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)



Checks:
- policyDigest == authPolicy?
- deferred checks succeed?
    - command == RSA_Decrypt?
    - PCR 1 == mOS?
    - PCR 2 == mA?

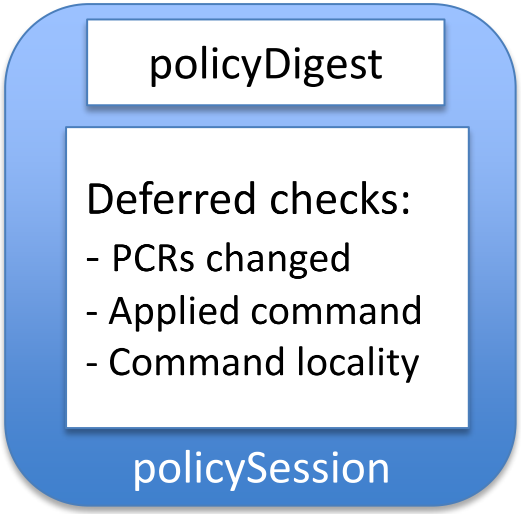
TPM2 Policy Session Contents

- Contains accumulated session policy value: **policyDigest**

newDigestValue := H(oldDigestValue ||
commandCode || state_info)

- Some policy commands **reset** the value

IF condition THEN
newDigestValue := H(0 || commandCode
|| state:info)



- Can contain optional assertions for **deferred policy checks** to be made at object access time.

TPM2 Policy Command Examples

- **TPM2_PolicyPCR:** Include PCR values in the authorization

  update *policyDigest* with *[pcr index, pcr value]*

  newDigest := H(oldDigest || TPM_CC_PolicyPCR || pcrs || digestTPM)

- **TPM2_PolicyNV:** Include a reference value and operation (<, >, eq) for non-volatile memory area

  e.g., *if counter5 > 2 then*
  update *policyDigest* with *[ref, op, mem.area]*

  newDigest := H(oldDigest || TPM_CC_PolicyNV || args || nvIndex->Name)

TPM2 Deferred Policy Example

- **TPM2_PolicyCommandCode:** Include command code for later checking during ”object invocation” operation:

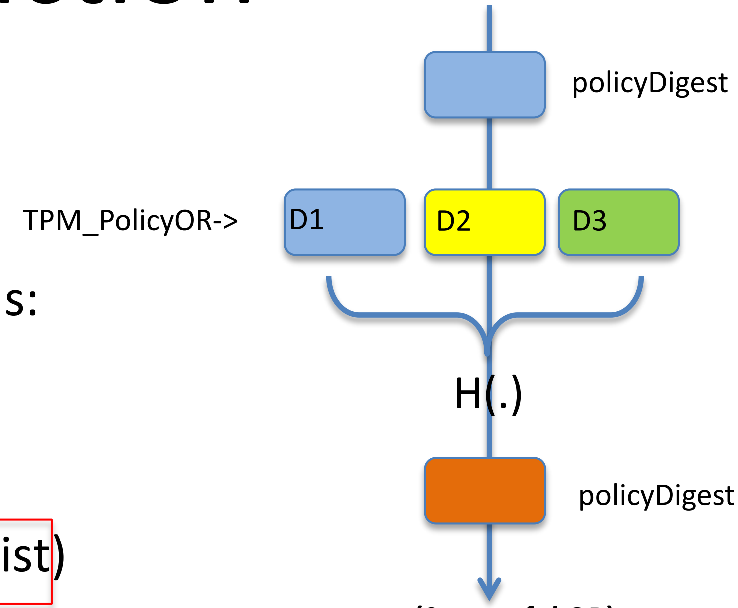
  update *policyDigest* with *[command code]*

  newDigest := H(oldDigest || TPM_CC_PolicyCommandCode || code)

  additionally save *policySession->commandCode := command code*

policySession->commandCode checked at the time of object invocation!

Policy disjunction



(Successful OR)

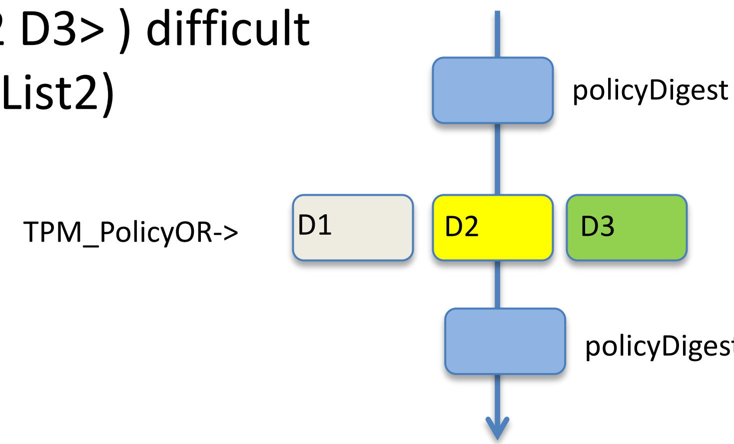
TPM2_PolicyOR: Authorize one of several options:

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

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

newDigest := H(0 || TPM2_CC_PolicyOR || List)

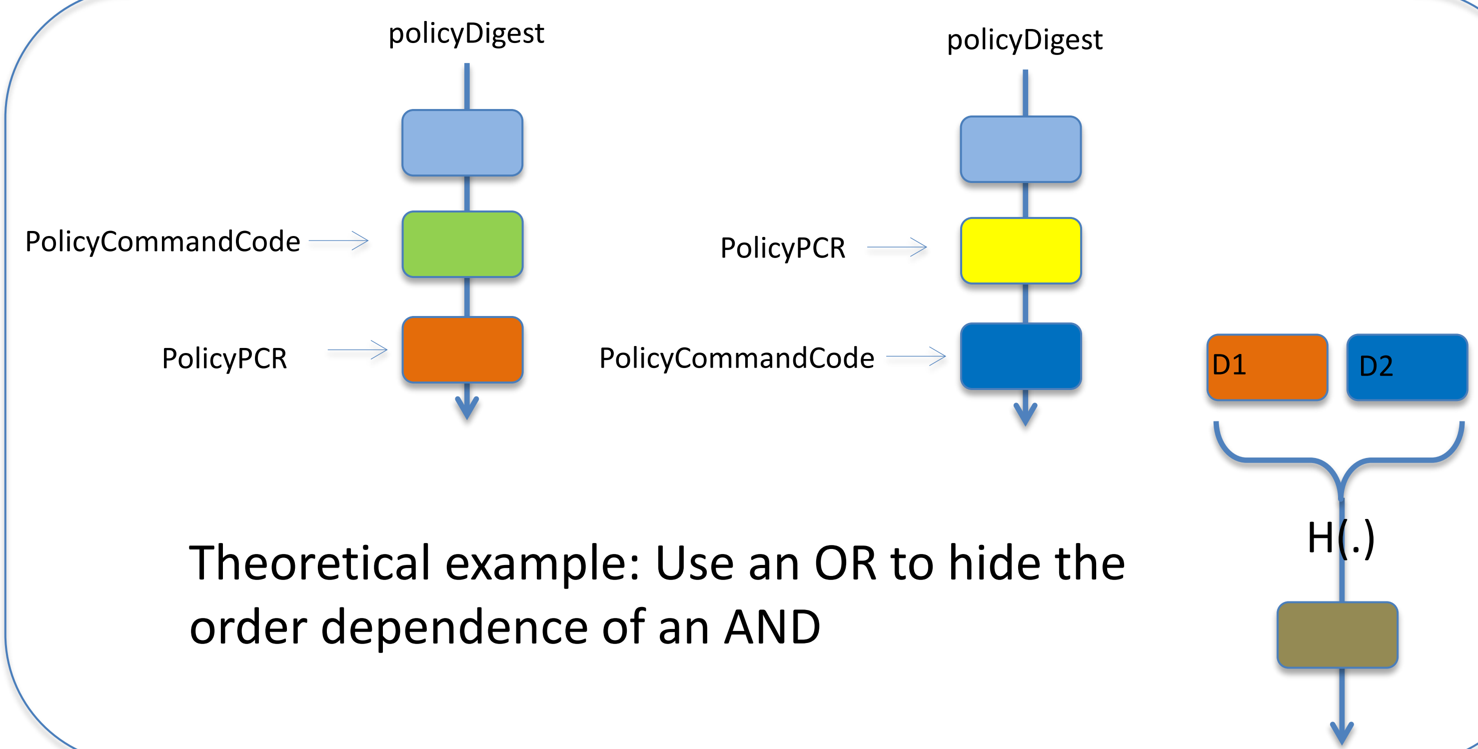
Reasoning: For a wrong digest Dx (not in <D1 D2 D3>) difficult to find *List2* = <Dx Dy, Dz, .. > where H(List) == H(List2)



(Failing OR)

Policy conjunction

- No explicit AND command
- AND achieved by consecutive authorization commands → order dependence

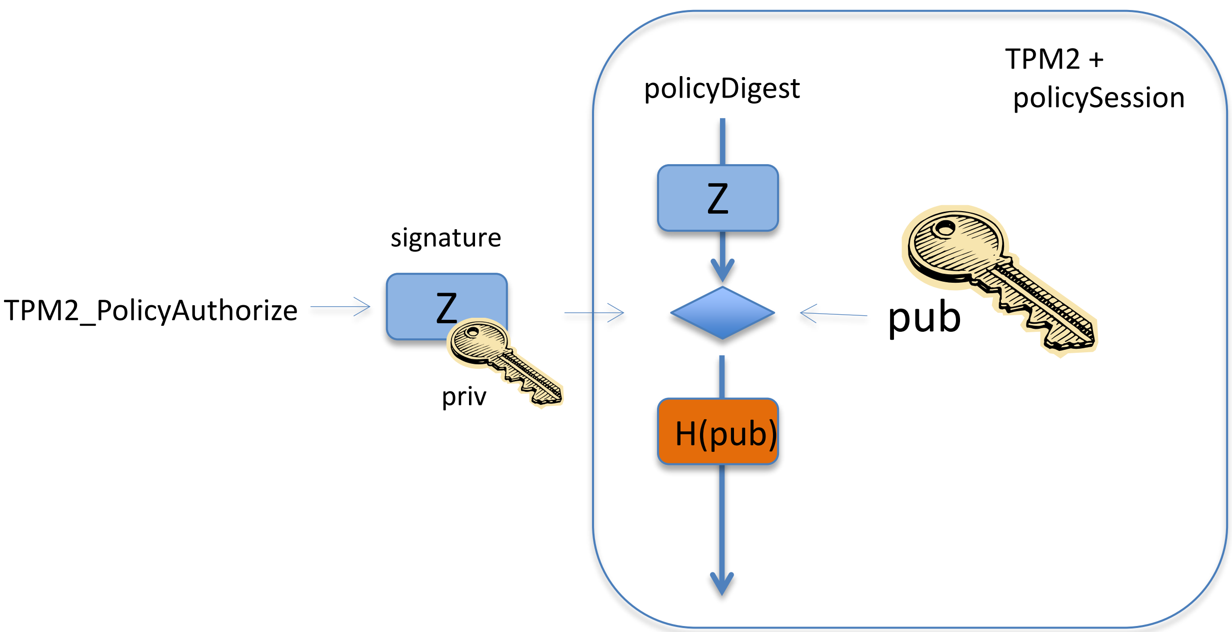


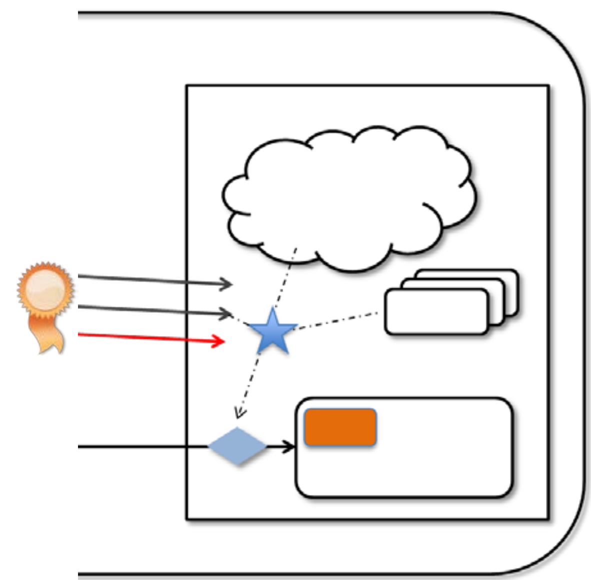
Theoretical example: Use an OR to hide the order dependence of an AND

External Authorization

TPM2_PolicyAuthorize: Validate a signature on a policyDigest:

IF signature validates **AND** *policySession->policyDigest* in *signed content*
THEN
newDigest := H(0 || TPM2_CC_PolicyAuthorize || **H(pub)** || ..)





Let’s try this out: example 1

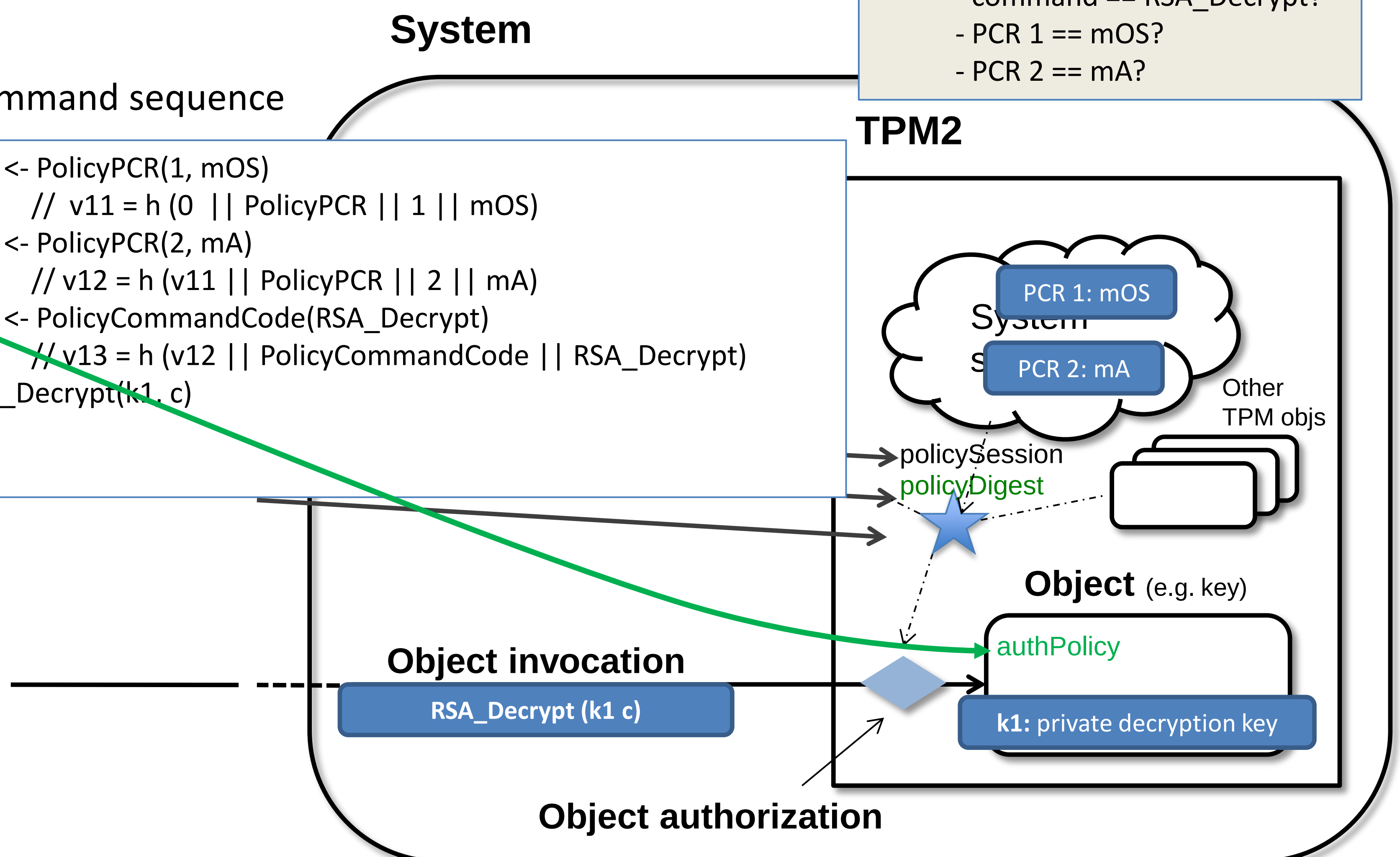
- Developer D
  - Has TPM2-protected keypair *k1* and Application A
  - Wants **only A** can use *k1* via
    - TPM2_RSA_Decrypt (key, ciphertext)
- Assume that
  - OS measured into PCR1 (if correct OS: PCR1 = mOS)
  - Foreground app into PCR2 (if A: PCR2 = mA)
- What should authPolicy of *k1* be?

Example 1

Two checks:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == RSA_Decrypt?
  - PCR 1 == mOS?
  - PCR 2 == mA?

Command sequence

```
v11 <- PolicyPCR(1, mOS)
        //  v11 = h (0  || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
        // v12 = h (v11 || PolicyPCR || 2 || mA)
v13 <- PolicyCommandCode(RSA_Decrypt)
        // v13 = h (v12 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)
```



NOTE: We drop "TPM2_" and "TPM_" prefixes for simplicity…

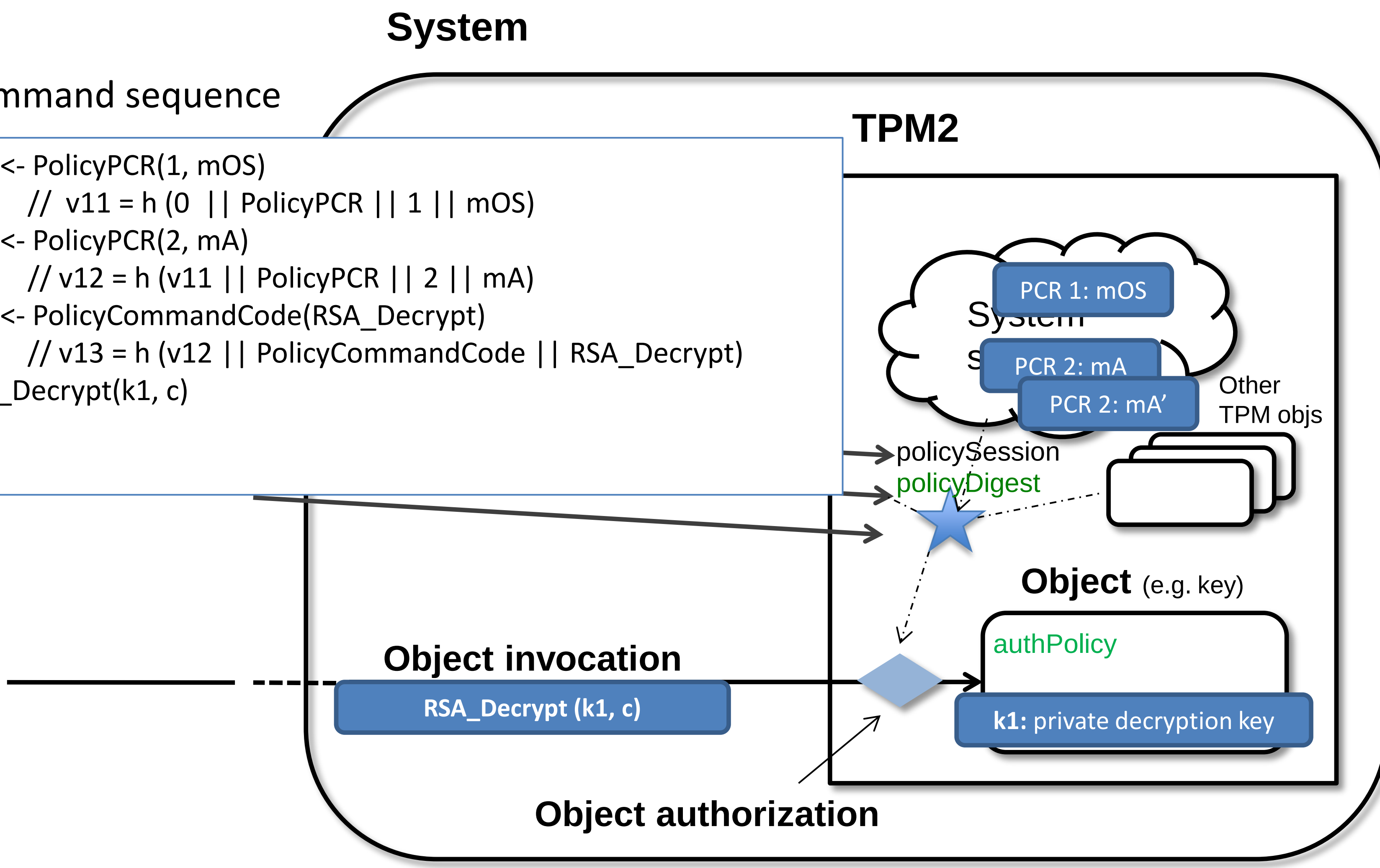
Example 2

- What if D wants to authorize app A (PCR2=mA) ***or*** app A’ (PCR2=mA’)

Example 2

Command sequence

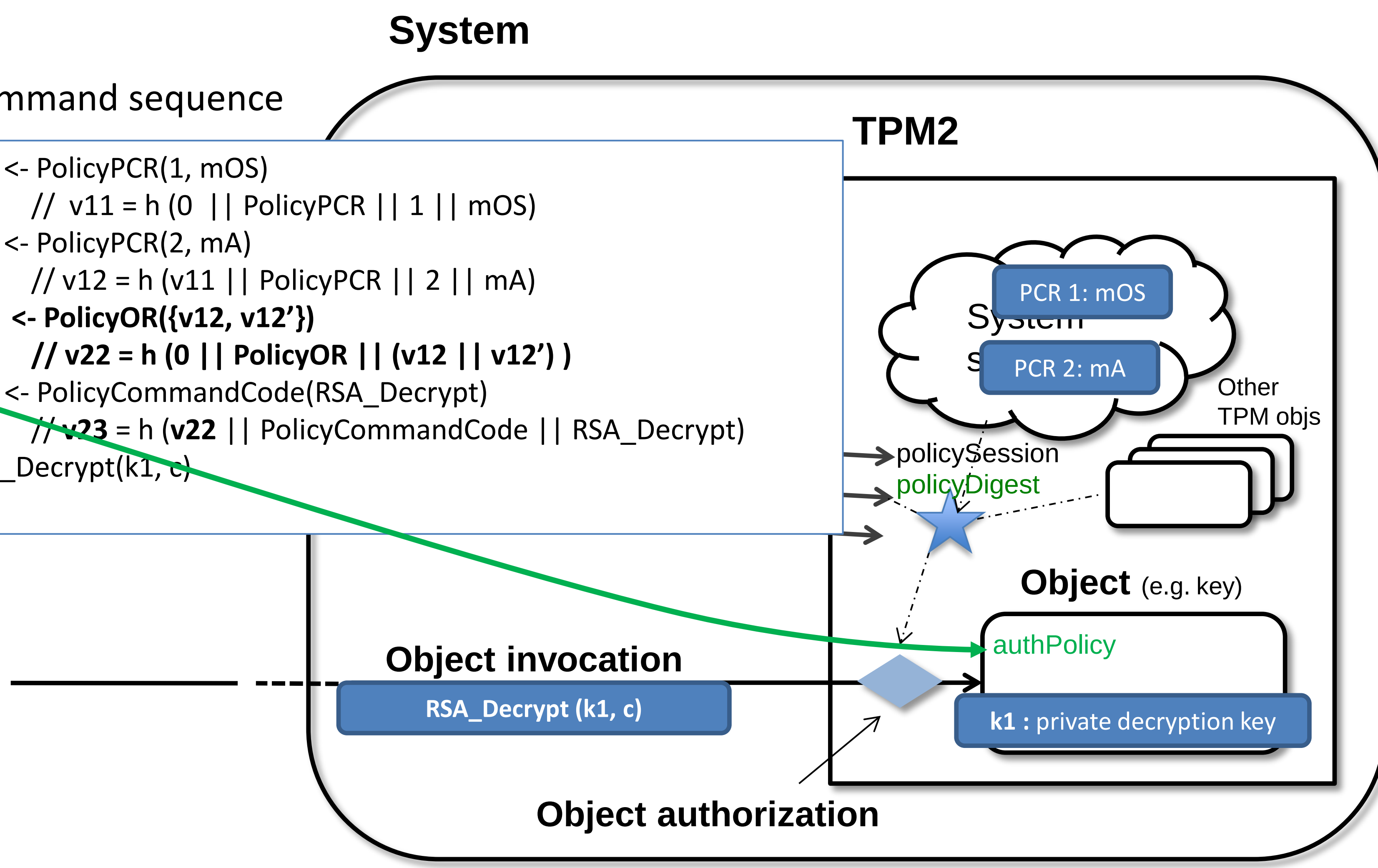
```
v11 <- PolicyPCR(1, mOS)
        // v11 = h (0 || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
        // v12 = h (v11 || PolicyPCR || 2 || mA)
v13 <- PolicyCommandCode(RSA_Decrypt)
        // v13 = h (v12 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)
```



Example 2

Command sequence

v11 <- PolicyPCR(1, mOS)
    // v11 = h (0 || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
    // v12 = h (v11 || PolicyPCR || 2 || mA)
v22 <- PolicyOR({v12, v12'})
    // v22 = h (0 || PolicyOR || (v12 || v12'))
v23 <- PolicyCommandCode(RSA_Decrypt)
    // **v23** = h (**v22** || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)



Example 2

Command sequence

v11 <- PolicyPCR(1, mOS)

    // v11 = h (0 || PolicyPCR || 1 || mOS)

v12' <- PolicyPCR(2, **mA'**)

    // v12' = h (v11 || PolicyPCR || 2 || **mA'**)

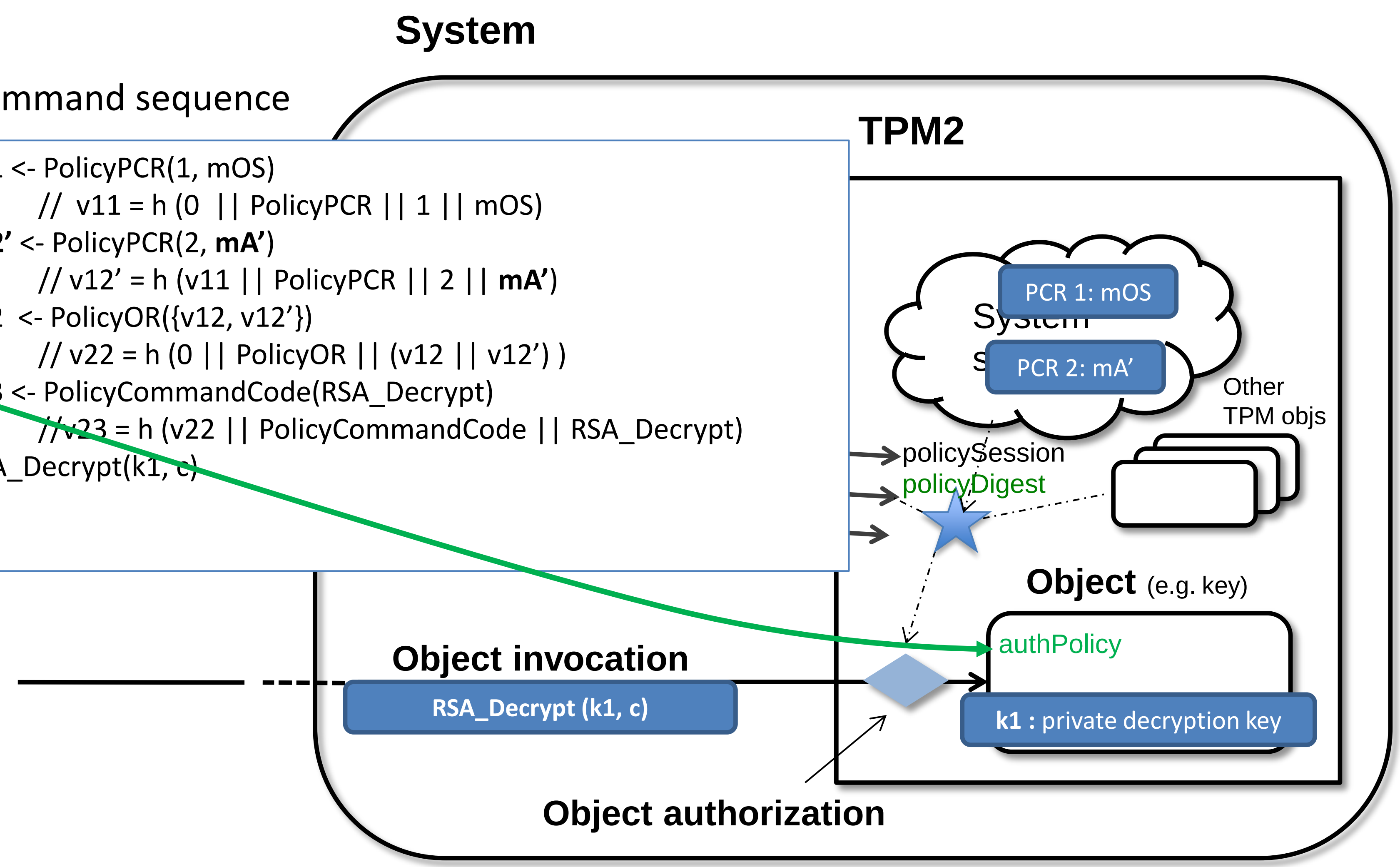
v22 <- PolicyOR({v12, v12'})

    // v22 = h (0 || PolicyOR || (v12 || v12'))

v23 <- PolicyCommandCode(RSA_Decrypt)

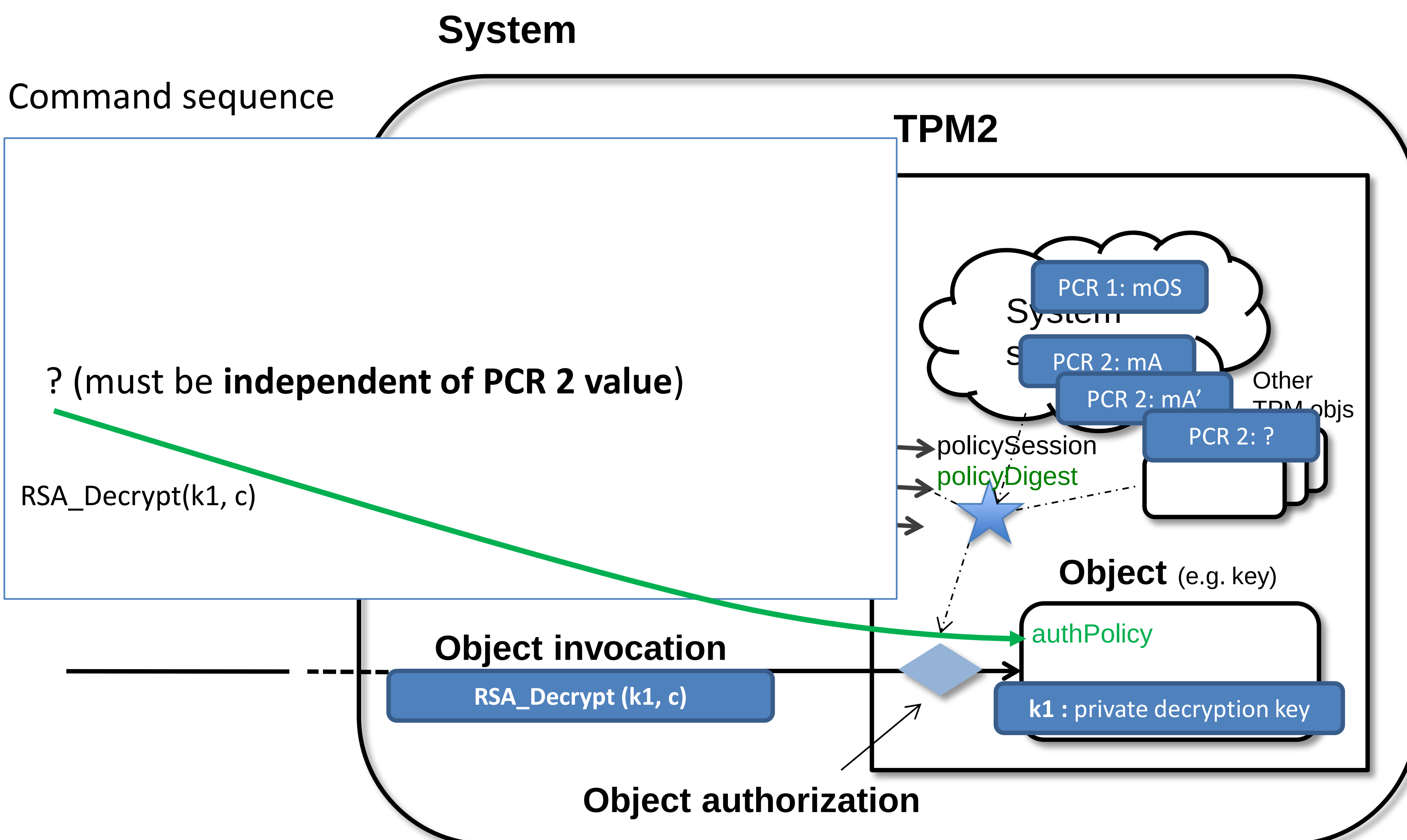
    // v23 = h (v22 || PolicyCommandCode || RSA_Decrypt)

RSA_Decrypt(k1, c)



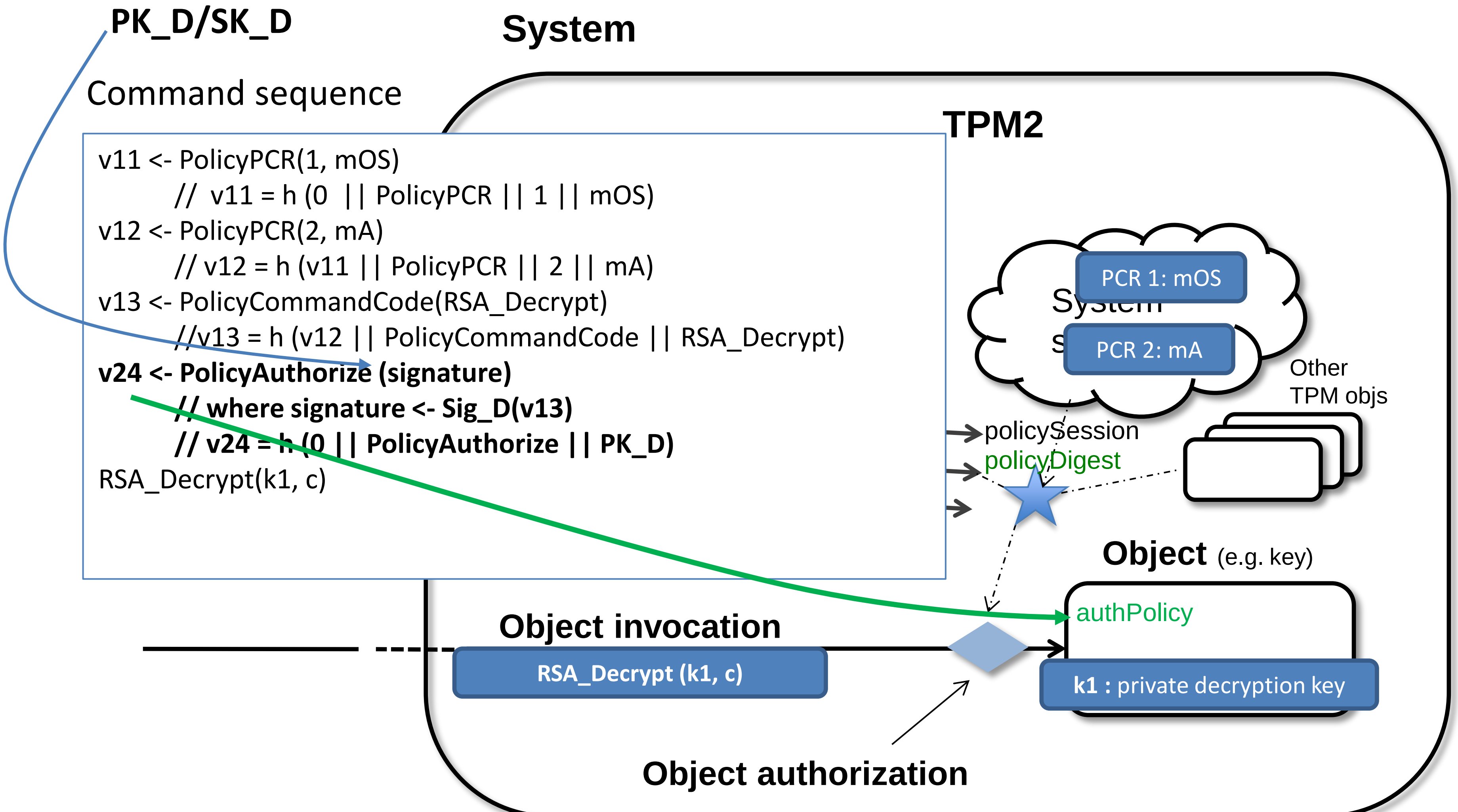
*Allow **any** app by D*

Example 2’



Example 2'

*Allow **any** app by D*



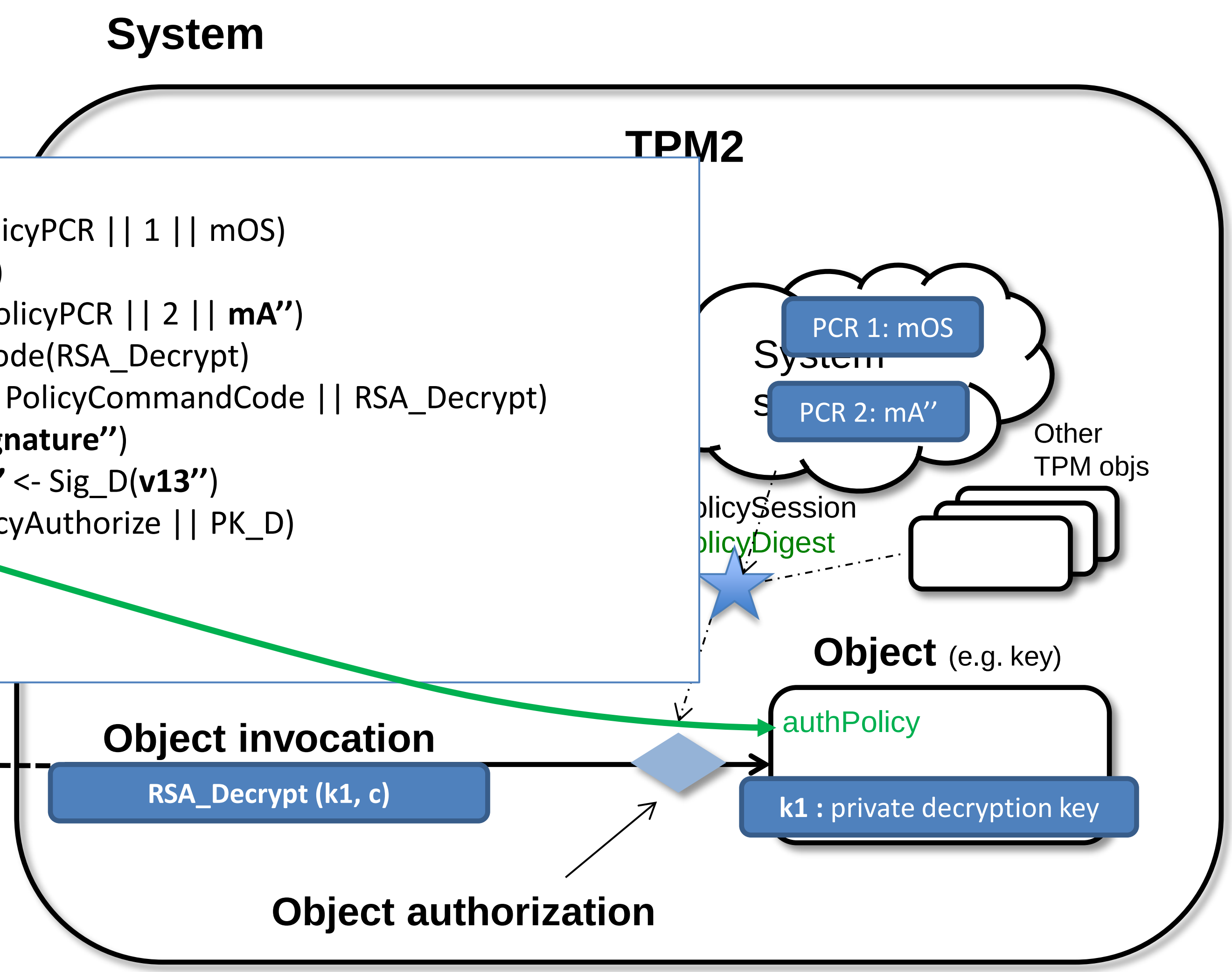
*Allow **any** app by D*

Example 2’

PK_D/SK_D

Command sequence

v11 <- PolicyPCR(1, mOS)
        // v11 = h (0 || PolicyPCR || 1 || mOS)
v12’’ <- PolicyPCR(2, **mA’’**)
        // **v12** = h (v11 || PolicyPCR || 2 || **mA’’**)
v13’’ <- PolicyCommandCode(RSA_Decrypt)
        //**v13’’** = h (**v12’’** || PolicyCommandCode || RSA_Decrypt)
v24 <- PolicyAuthorize (**signature’’**)
        // where **signature’’** <- Sig_D(**v13’’**)
        // v24 = h (0 || PolicyAuthorize || PK_D)
RSA_Decrypt(k1, c)



Example 2’

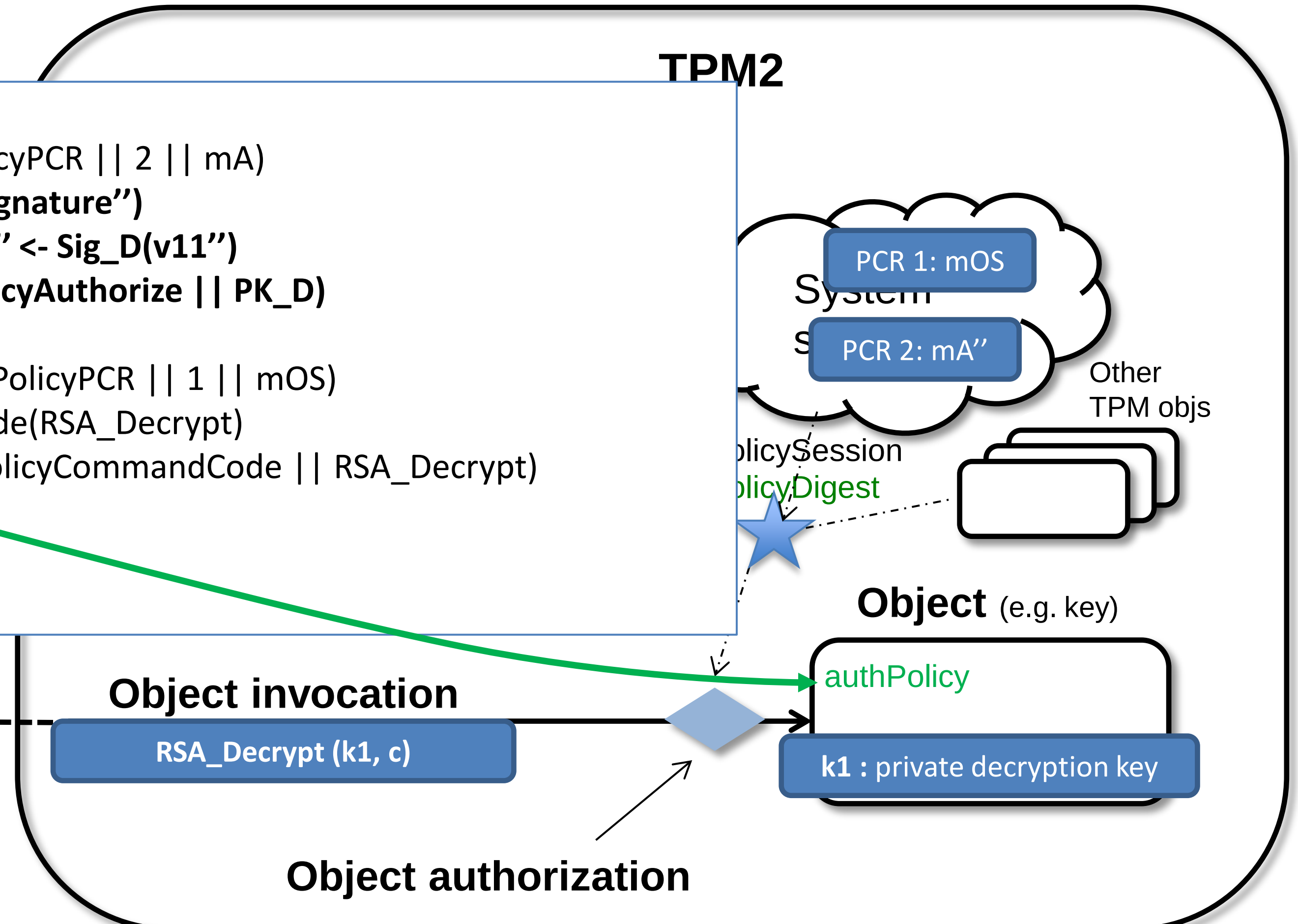
But we don’t want to allow any OS or any policyCommand!

PK_D/SK_D

System

Command sequence

v11 <- PolicyPCR(2, mA’’)
    // v11 = h (0 || PolicyPCR || 2 || mA)
v12 <- PolicyAuthorize (signature’’)
    // where signature’’ <- Sig_D(v11’’)
    // v12 = h (0 || PolicyAuthorize || PK_D)
v13 <- PolicyPCR(1, mOS)
    // v13 = h (v12 || PolicyPCR || 1 || mOS)
v14 <- PolicyCommandCode(RSA_Decrypt)
    //v14 = h (v14 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)



[Skip to Secure Boot example](#)

More Examples

- Example 3
  - D wants to license the use of k1 to any app of another developer D1
  - D1's app signing keypair PK_D1/SK_D1
- Example 4
  - D wants to license use of k1 to any app of any developer that he later authorizes!

Skip to Secure Boot example

Example 3

- D wants to license the use of k1 to any app of another developer D1
  - D1’s app signing keypair PK_D1/SK_D1

Example 3

*Allow **any** app by D **or** D1*

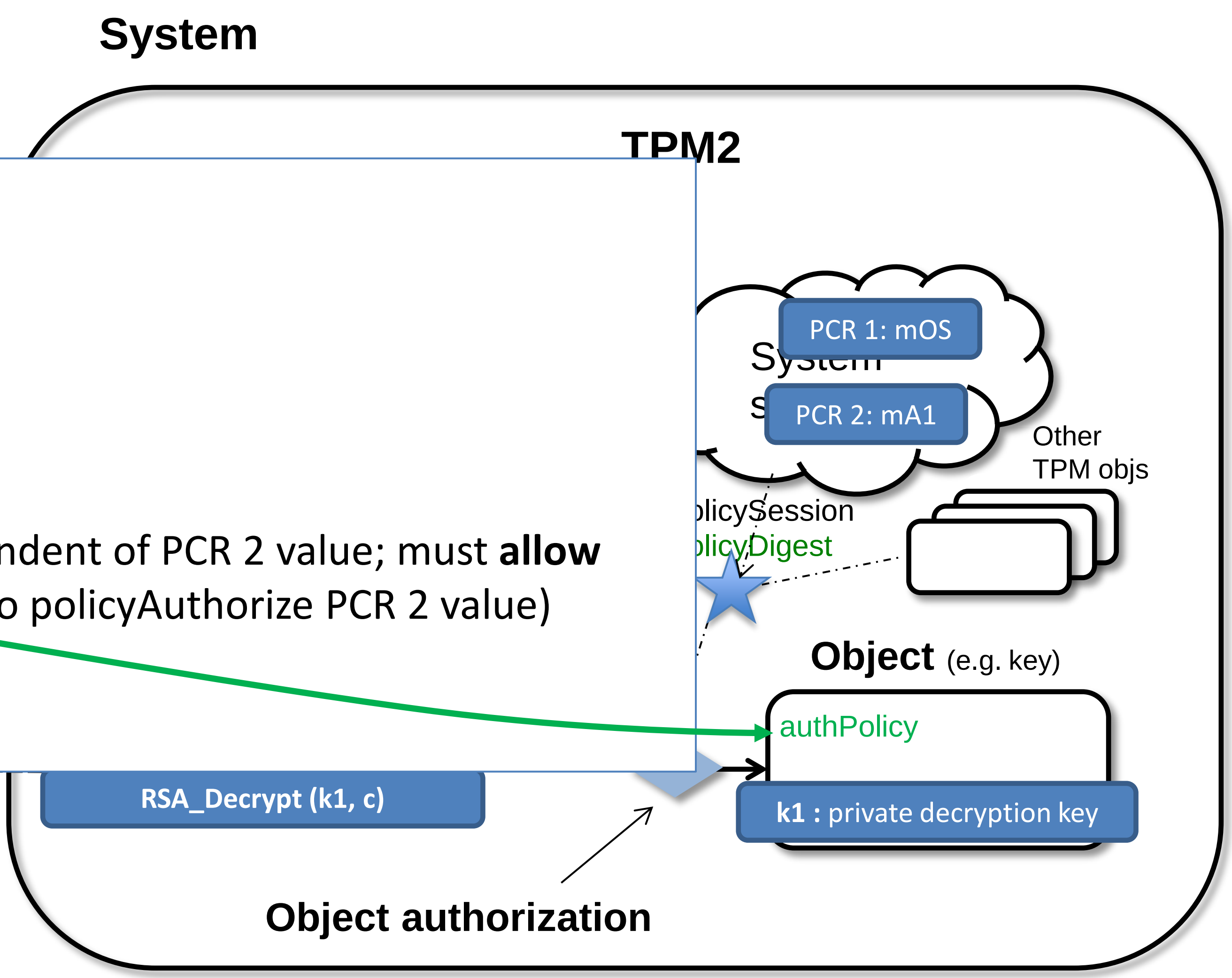
PK_D/SK_D

PK_D1/SK_D1

Command sequence

? (must be independent of PCR 2 value; must **allow either public** key to policyAuthorize PCR 2 value)

RSA_Decrypt(k1, c)



Example 3

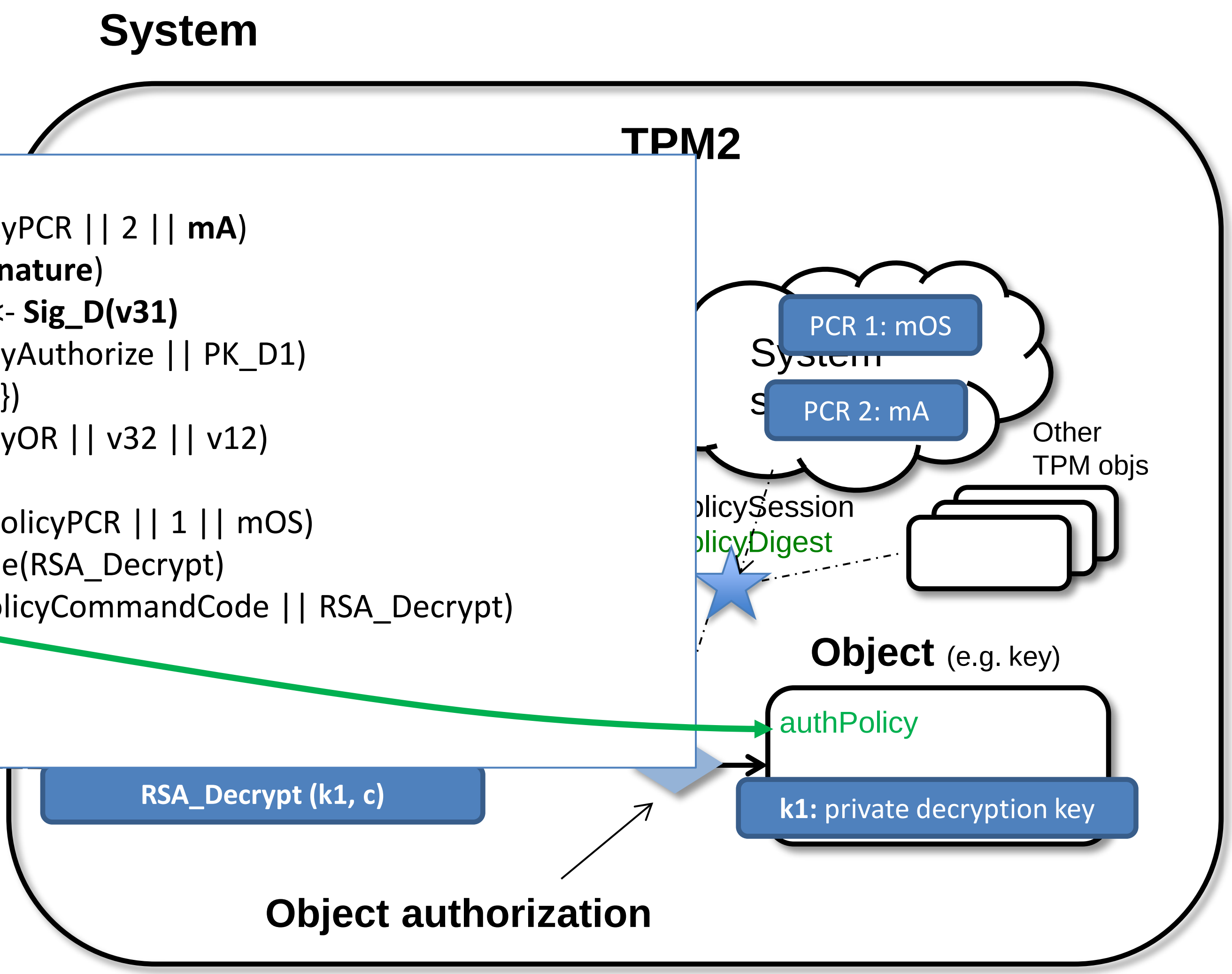
*Allow **any** app by D **or** D1*

PK_D/SK_D

PK_D1/SK_D1

Command sequence

v31 <- PolicyPCR(2, **mA**)
    // **v31** = h (0 || PolicyPCR || 2 || **mA**)
v32 <- PolicyAuthorize (**signature**)
    // where **signature** <- **Sig_D(v31)**
    // v32 = h (0 || PolicyAuthorize || PK_D1)
v33 <- PolicyOR ({v32, v32'})
    // v33 = h (0 || PolicyOR || v32 || v12)
v34 <- PolicyPCR(1, mOS)
    // v34 = h (v33 || PolicyPCR || 1 || mOS)
v35 <- PolicyCommandCode(RSA_Decrypt)
    // v35 = h (v34 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)



Example 4

- D wants to license use of k1 to any app of any developer that he later authorizes!

Example 4

*Allow any app certified by **any** developer authorized by D*

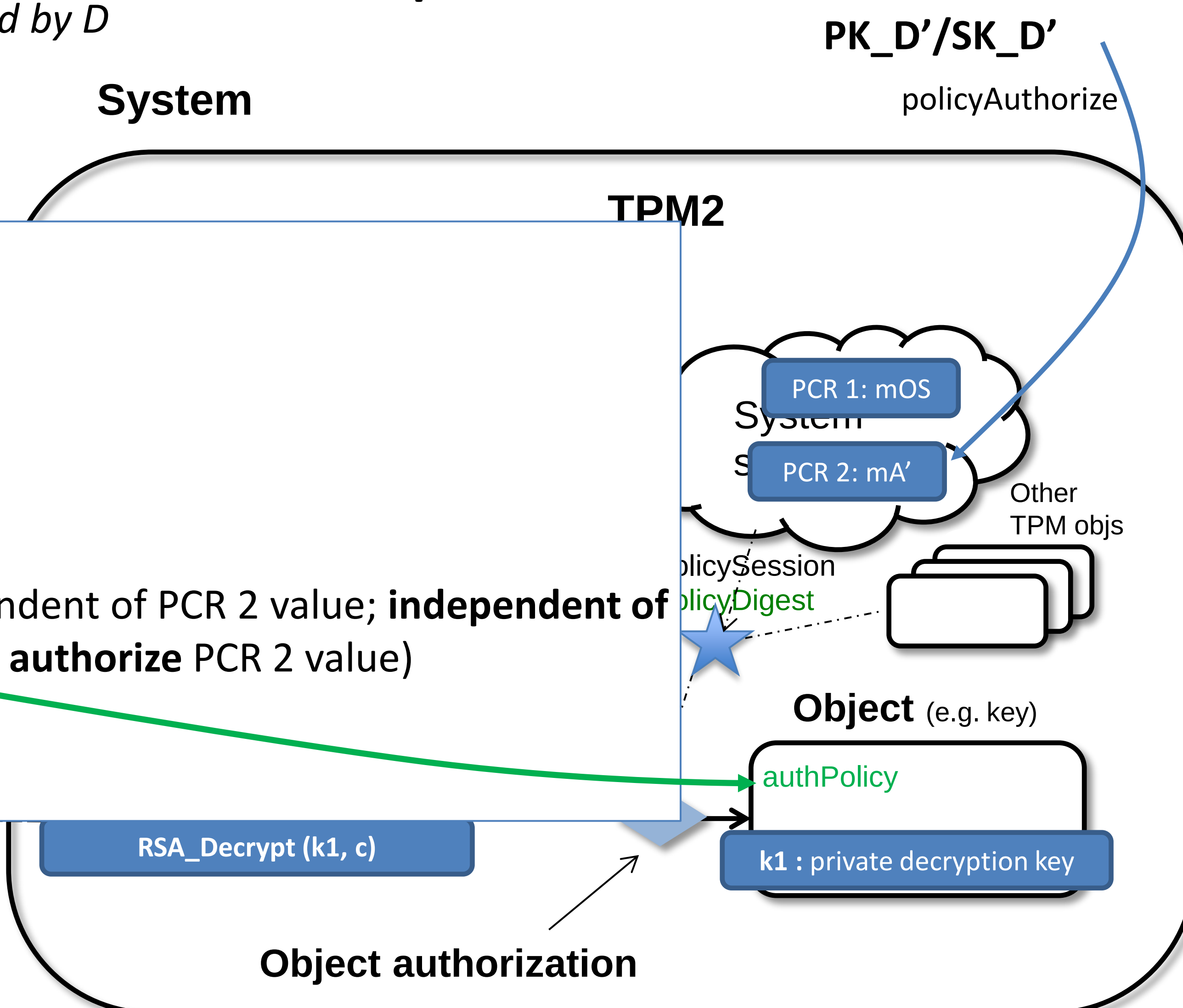
PK_D/SK_D

System

Command sequence

? (must be independent of PCR 2 value; **independent of public key used to authorize** PCR 2 value)

RSA_Decrypt(k1, c)



Example policy: Simple Secure Boot

Continue boot only if Platform A kernel has been loaded

measurement mA → PCR5

IF

measurement mA → PCR 2

Platform A kernel

Object invocation for authorization

- Suppose PCR 2 has value mA when Platform A kernel loaded

- Sequence of commands to ensure secure boot
  - [PCRExtend(2, measurement value); Start new authorization session]
  - V1 <- PolicyPCR (2, mA)
  - V2 <- PolicyCommandCode (PCRExtend)

  → **PCRExtend(5, mA)**

- authPolicy for PCR 5 is V2
  - V1 = h (0 || PolicyPCR || 2 || mA)
  - V2 = h (V1 || PolicyCommandCode || PCR_Extend)

Skip to Standards Summary

Simple secure boot not always enough

Secure boot **can** have the following properties

A) Extend to start up of applications

B) Include platform-dependent policy

C) Include optional or complementary boot branches

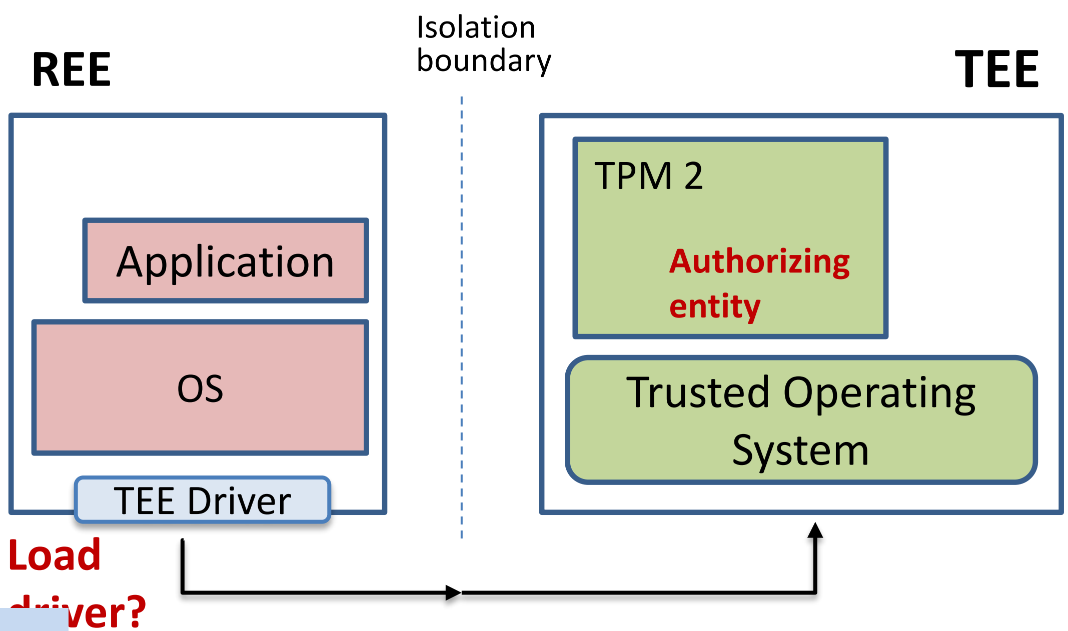
D) Order in which components are booted may matter

Skip to Standards Summary

Advanced Secure Boot example

1. Root-of-Trust-for-Measurement starts Boot Loader and boot process
2. It loads the TEE and TPM (PCR 1)
3. It loads the REE OS (PCR 2)
4. We want to verify **loading of the OS TEE driver** (PCR 3)

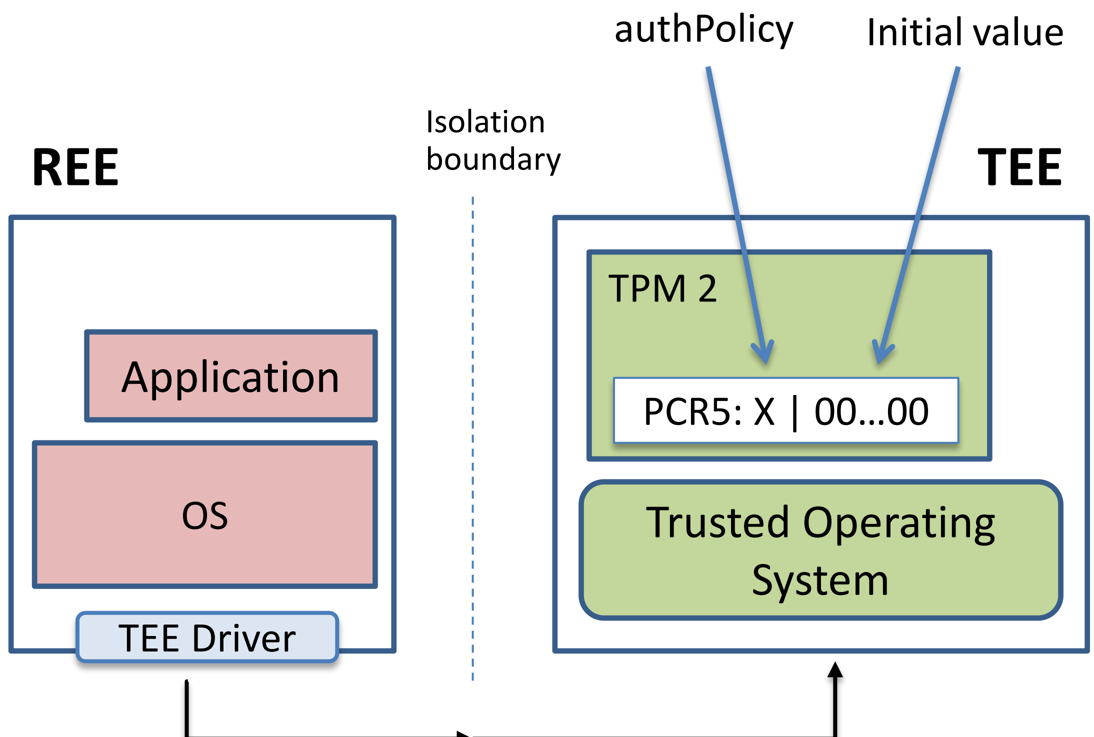
Authorization policy conditional to correct execution of previous steps



Skip to Standards Summary

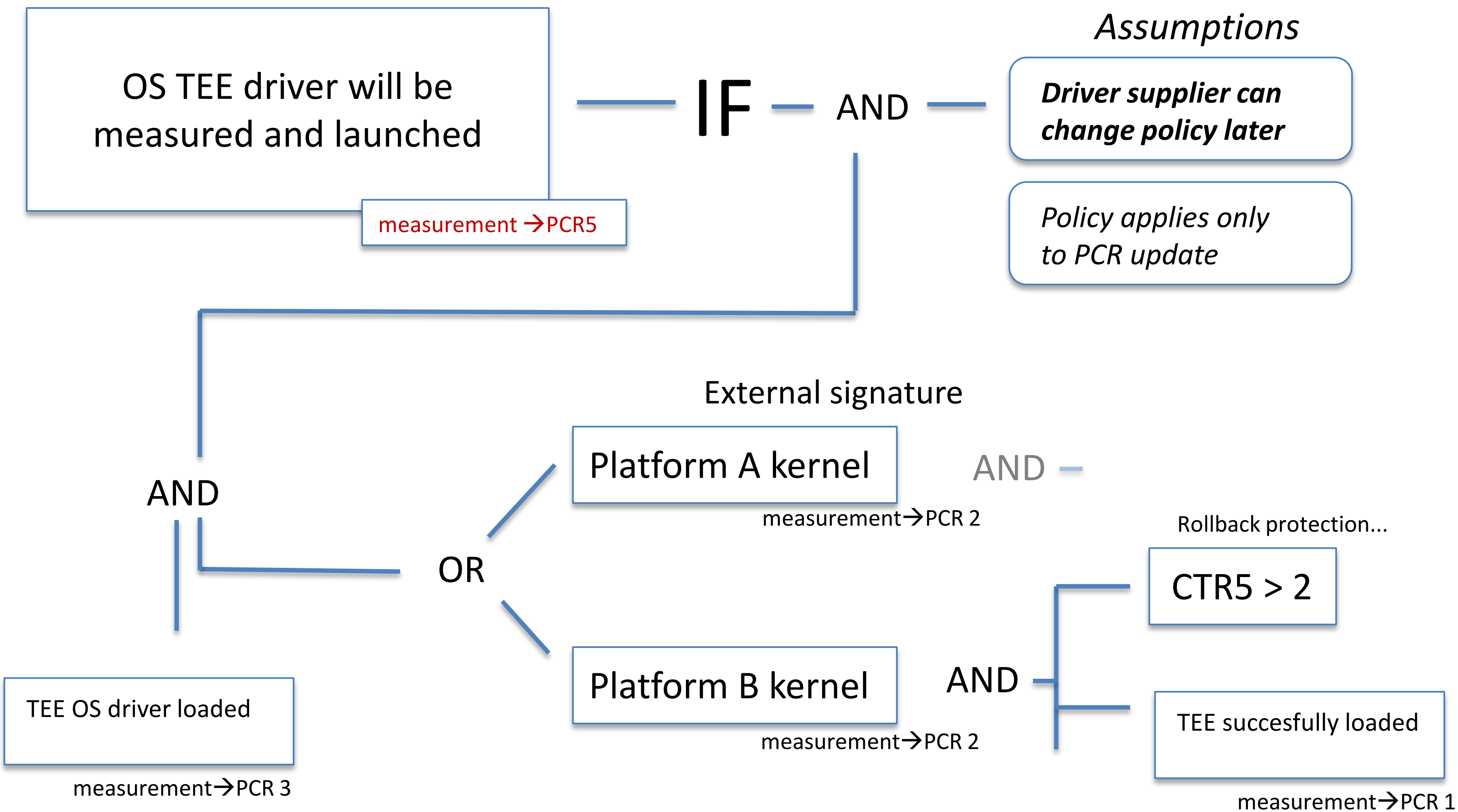
Advanced Boot: example policy

- Policy applies to extending of PCR5 (authPolicy = X)
- Create policy session with policyDigest = X



[Skip to Standards Summary]

Advanced Boot Policy

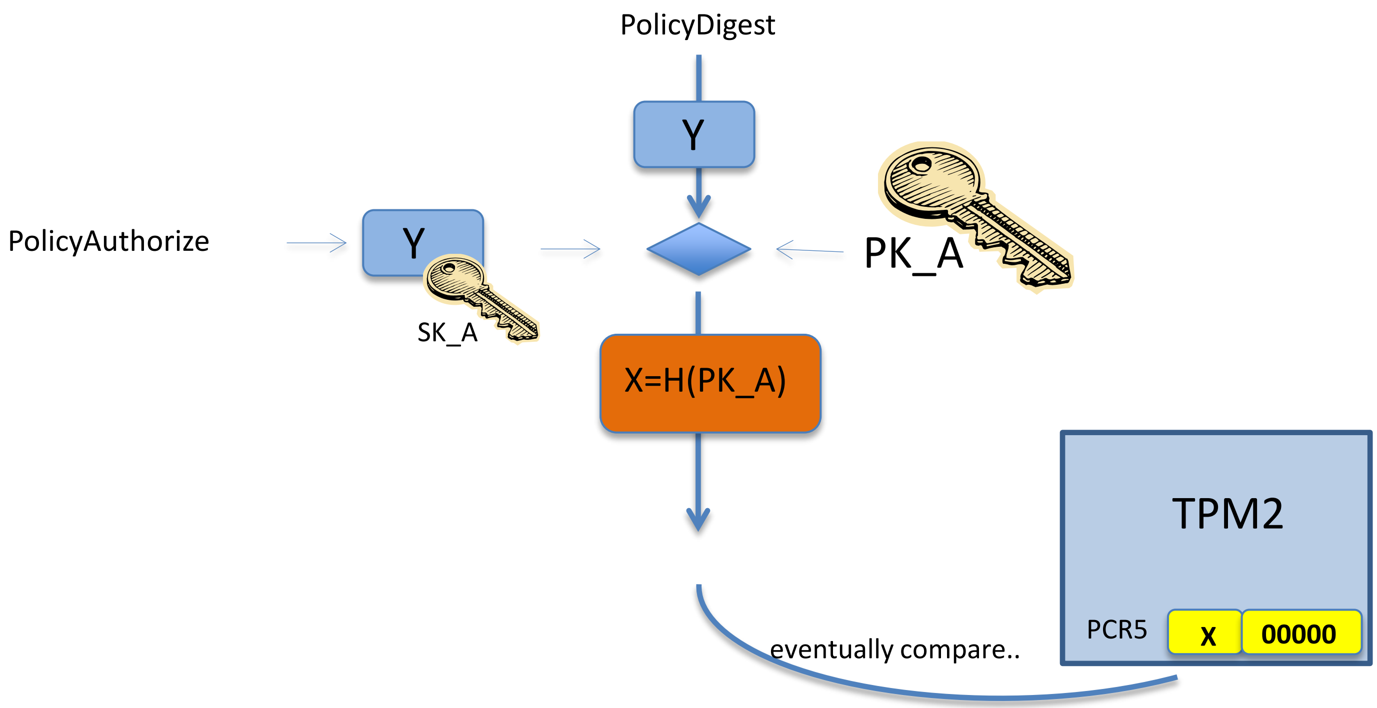


Skip to Standards Summary

Advanced Boot Policy

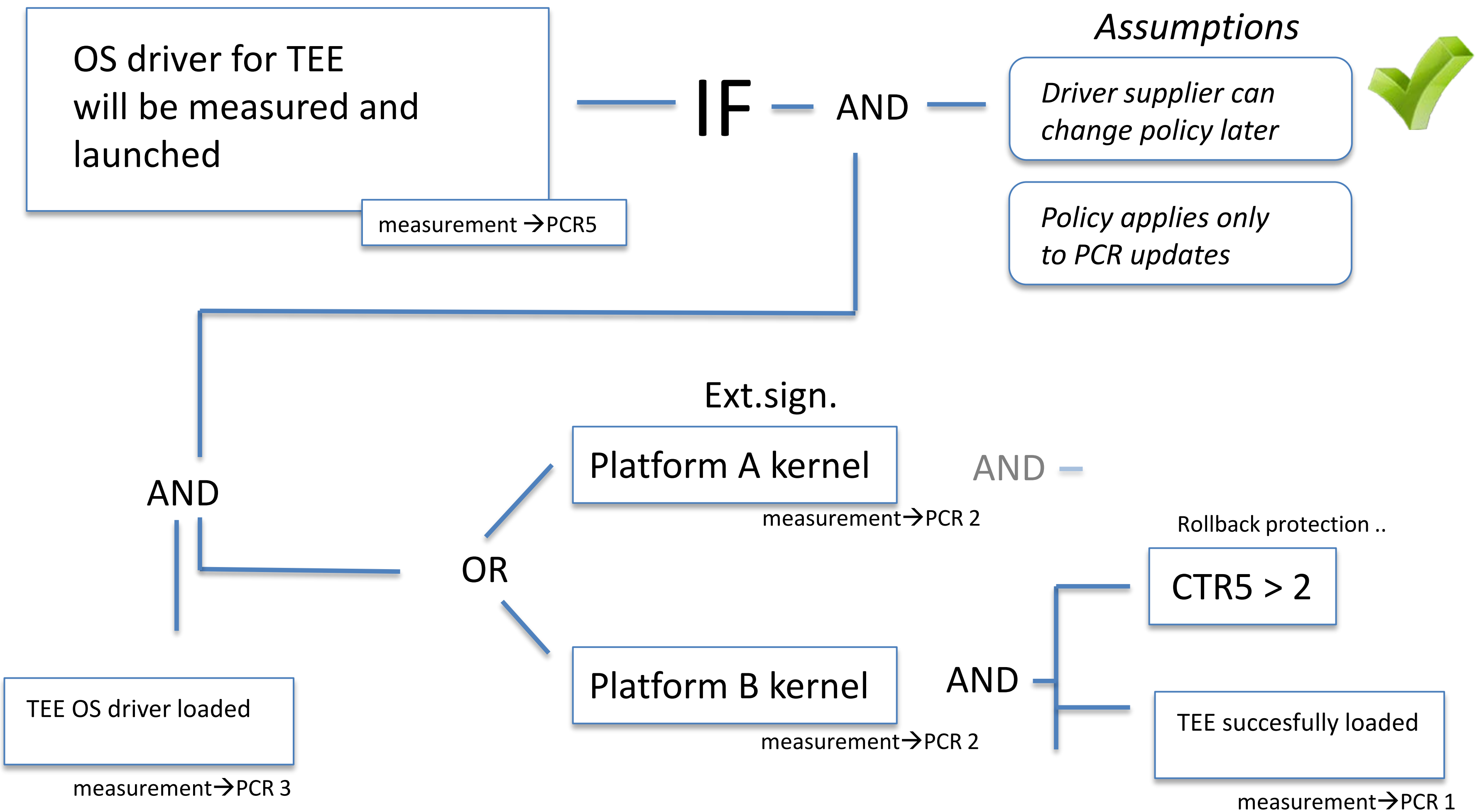
- authPolicy X = (PK_A)*
- driver supplier **A** can authorize any value Y as policy for PCR 5

* more precisely H(0 || PolicyAuthorize || **PK_A** || …)



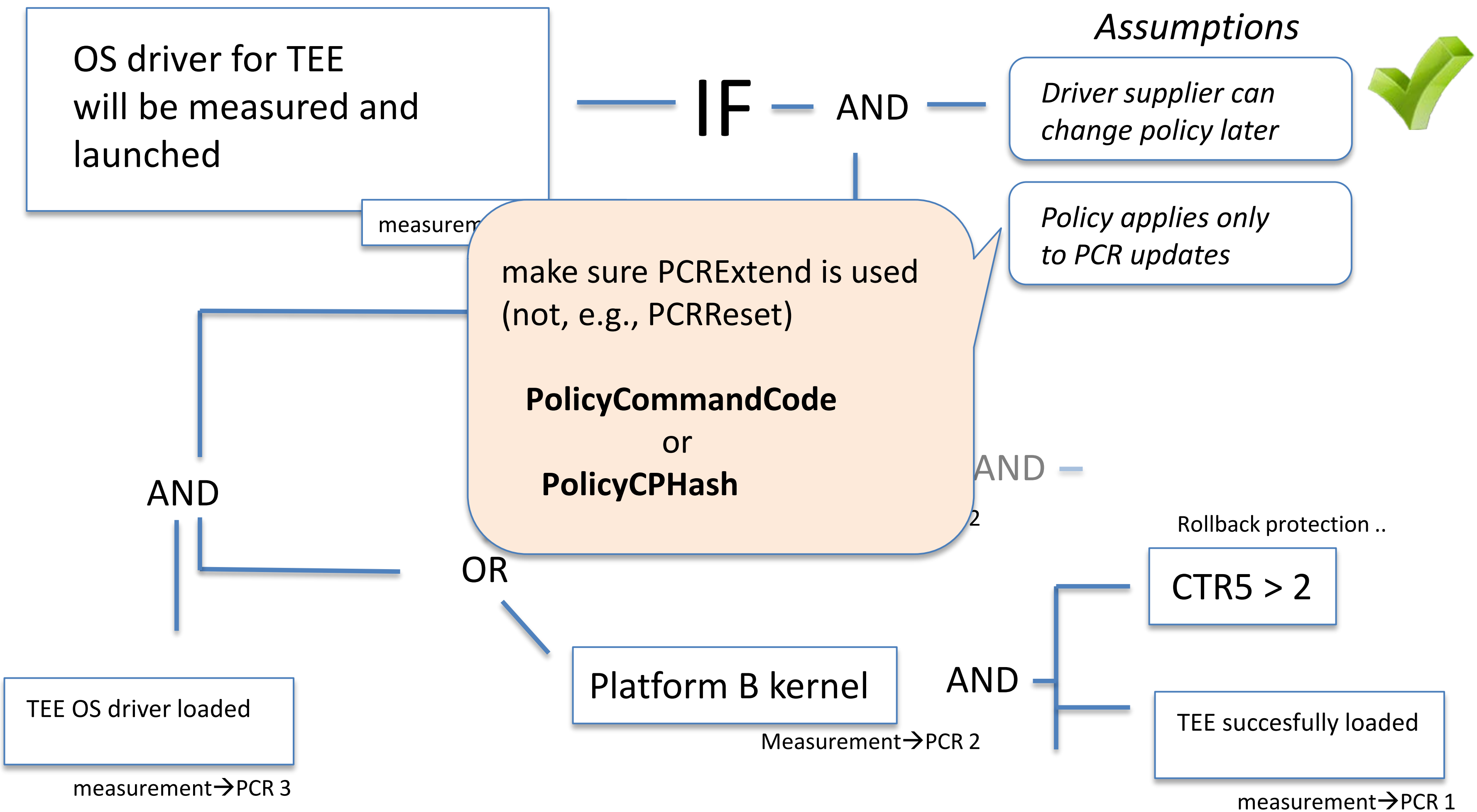
Y → PolicyAuthorize(Sig_A(Y)) → **X**

Example policy



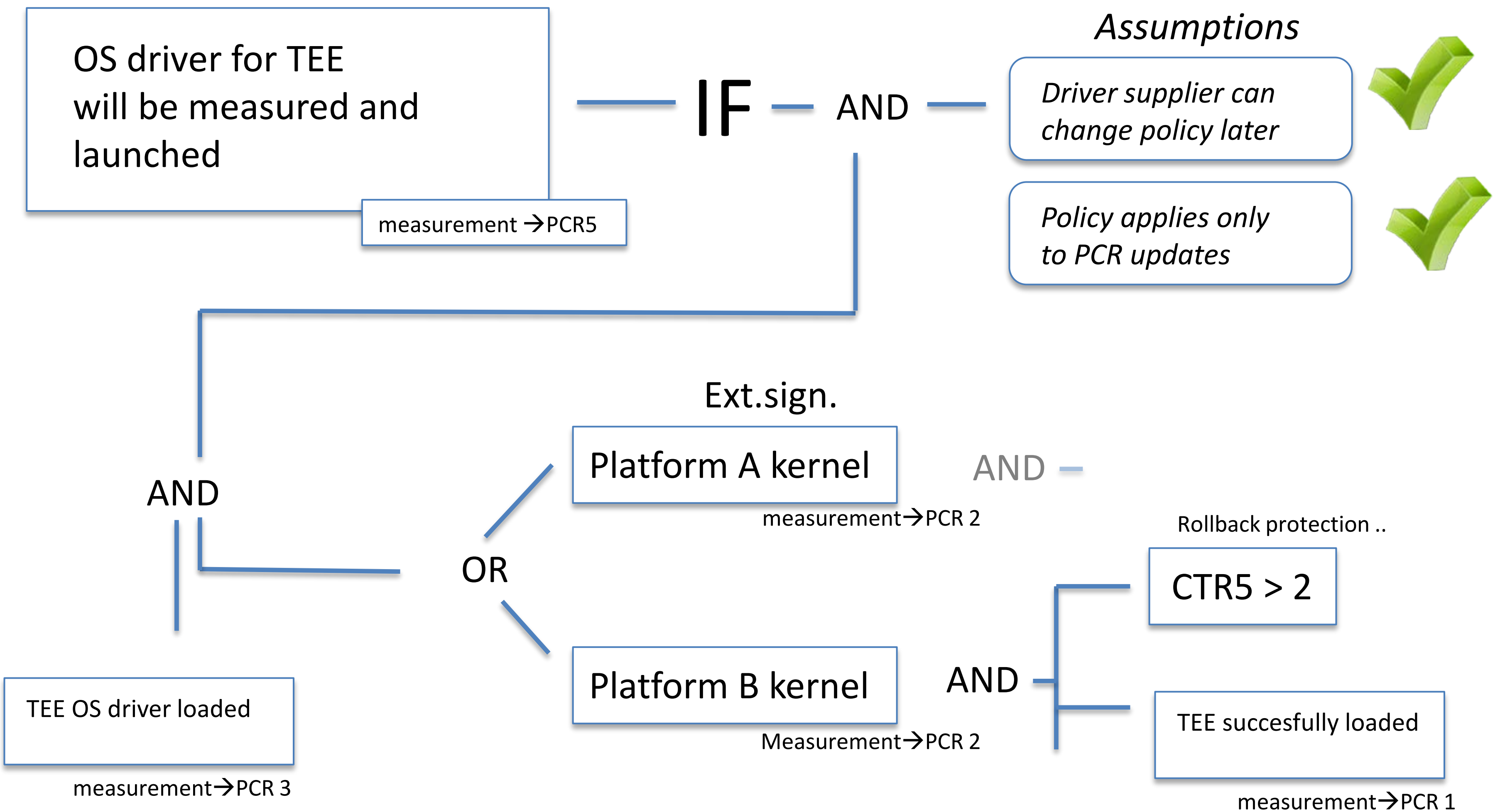
Y → PolicyAuthorize(Sig$_A$ (Y)) → **X**

Example policy



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

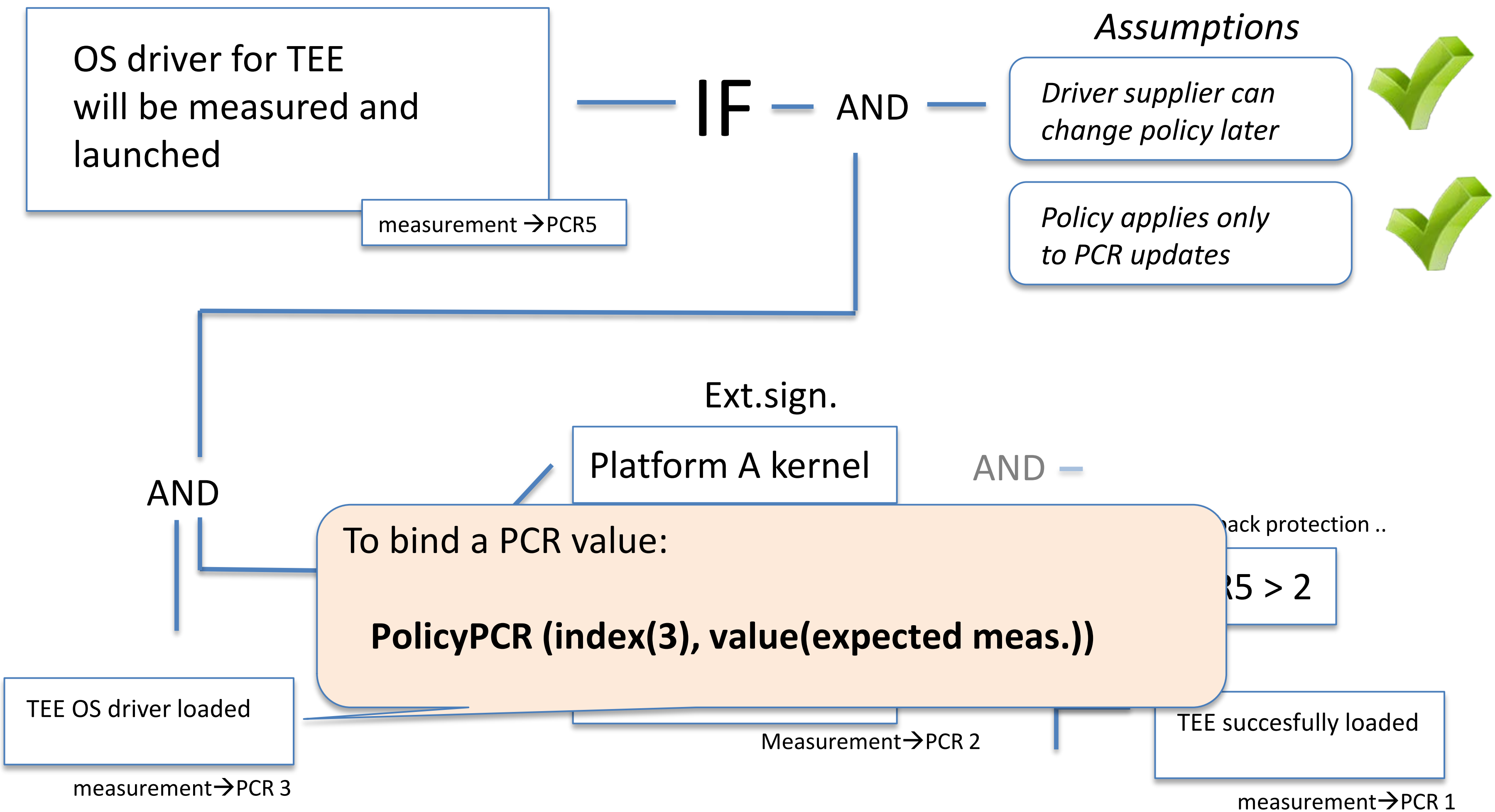
Example policy



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

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

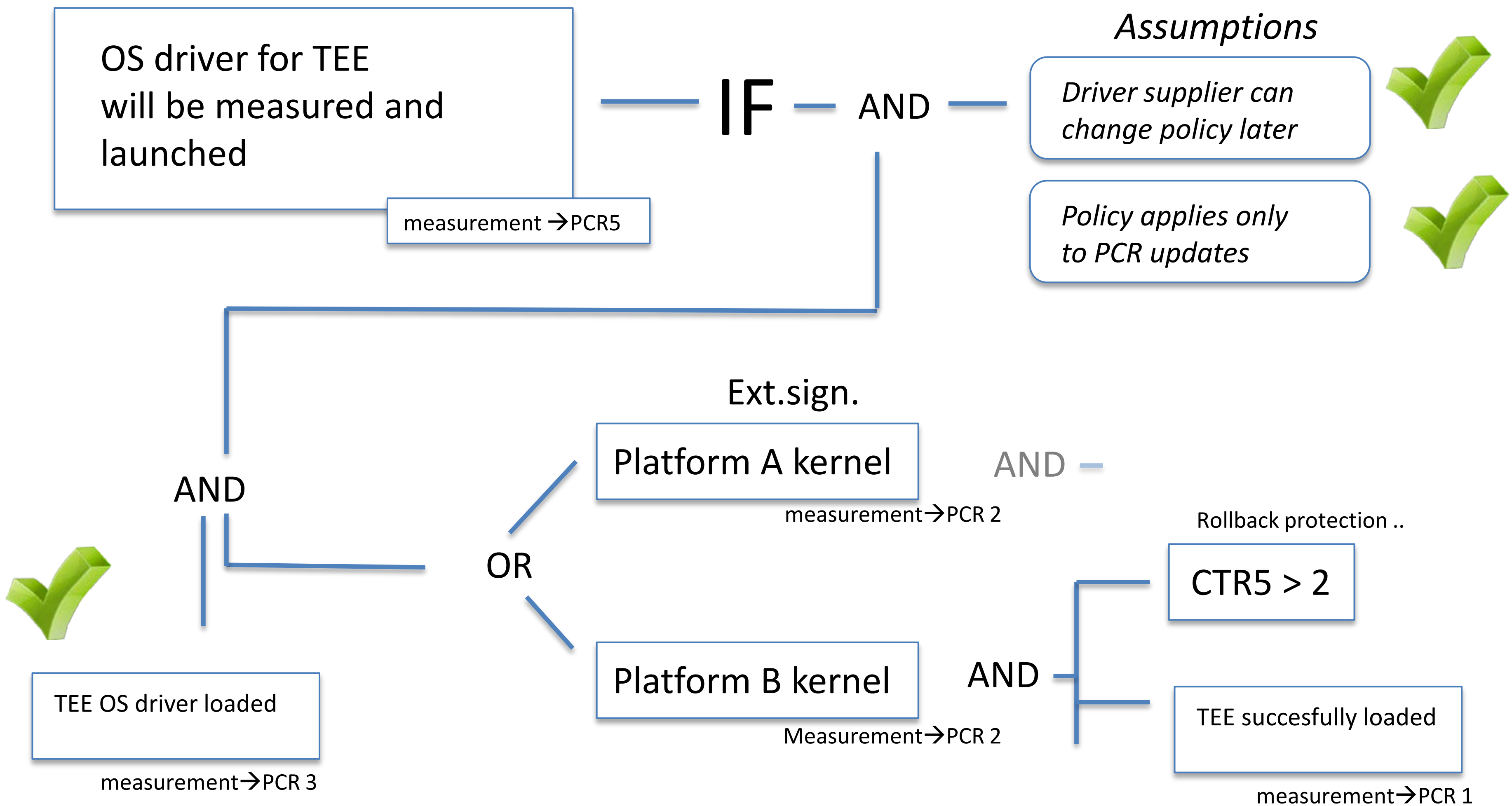
Example policy



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

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

Example policy

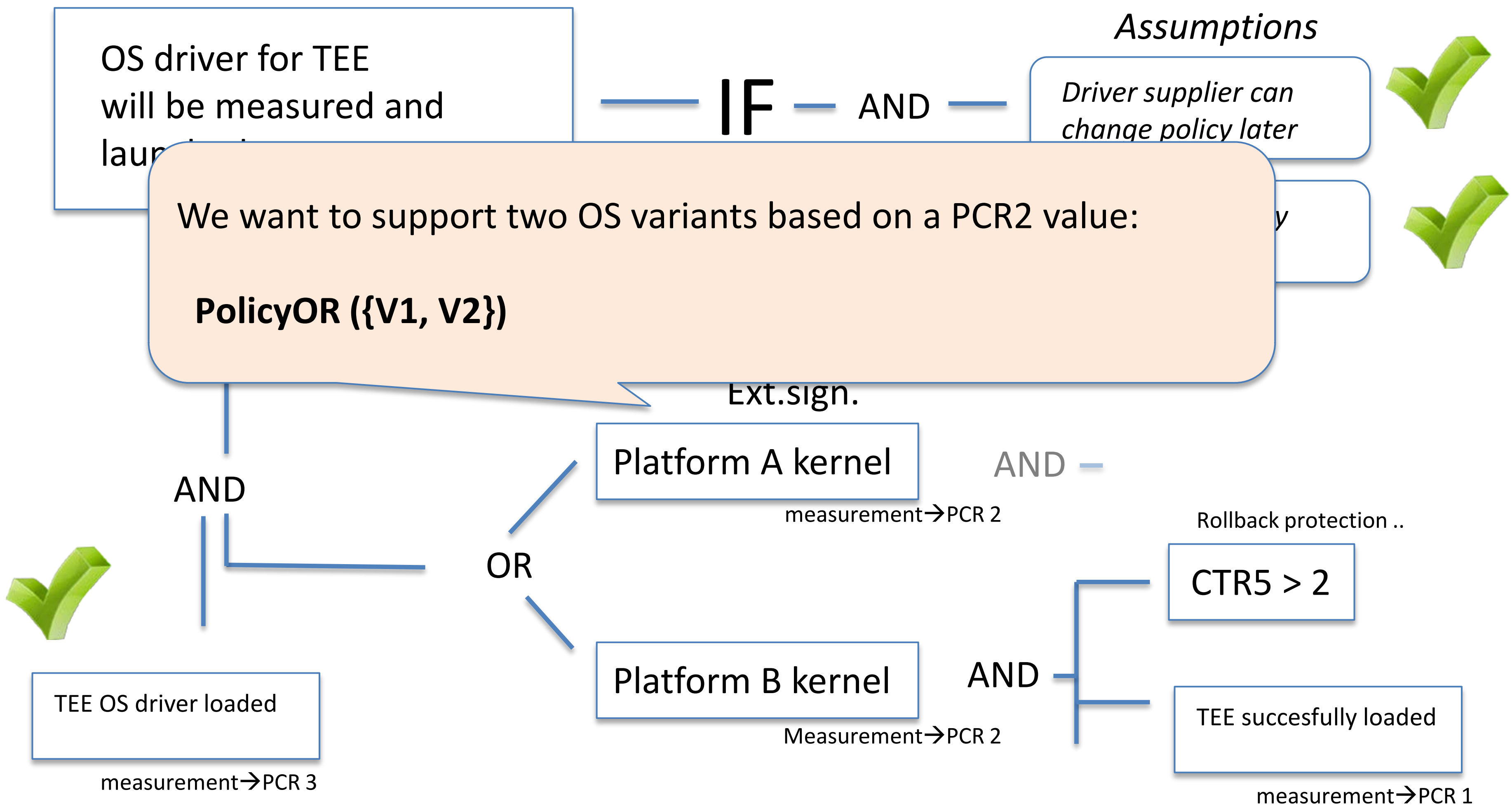


W → PolicyPCR(3, meas.) → **Z**

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

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

Example policy

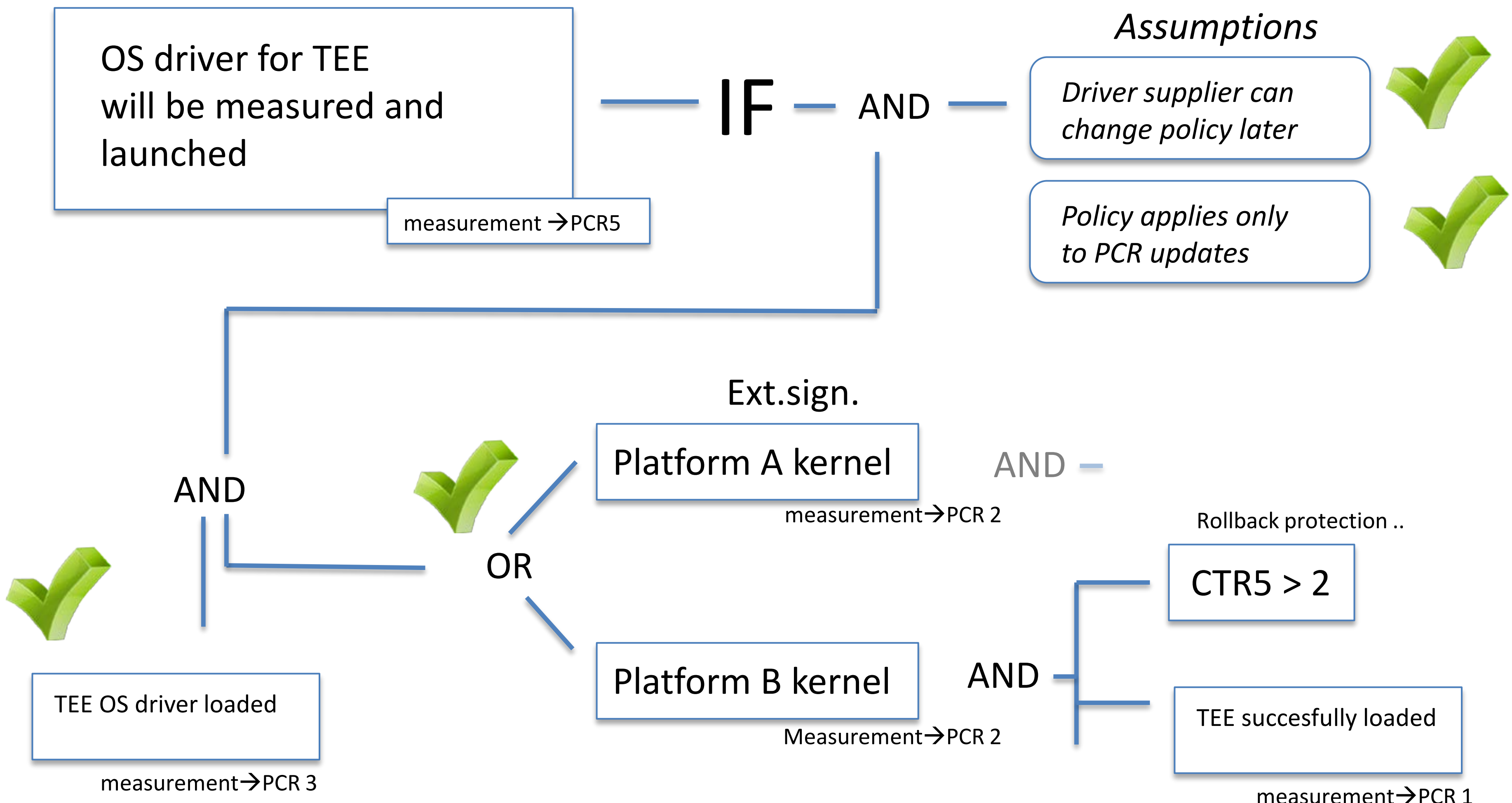


W → PolicyPCR(3, meas.) → **Z**

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

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

Example policy

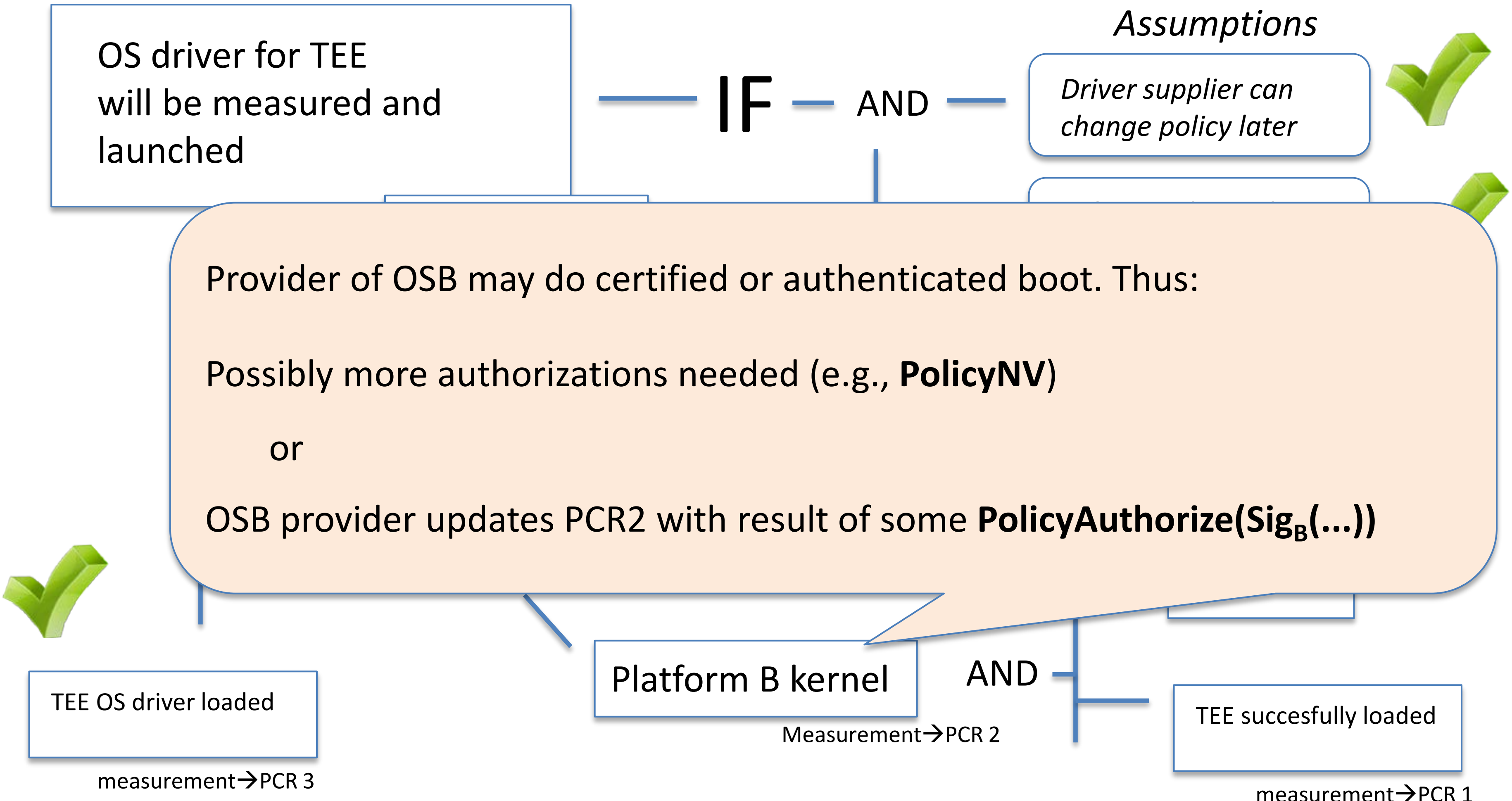


V1 →
V2 → PolicyOr({V1,V2}) →**W** → PolicyPCR(3, meas.) → **Z**

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

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

Example policy

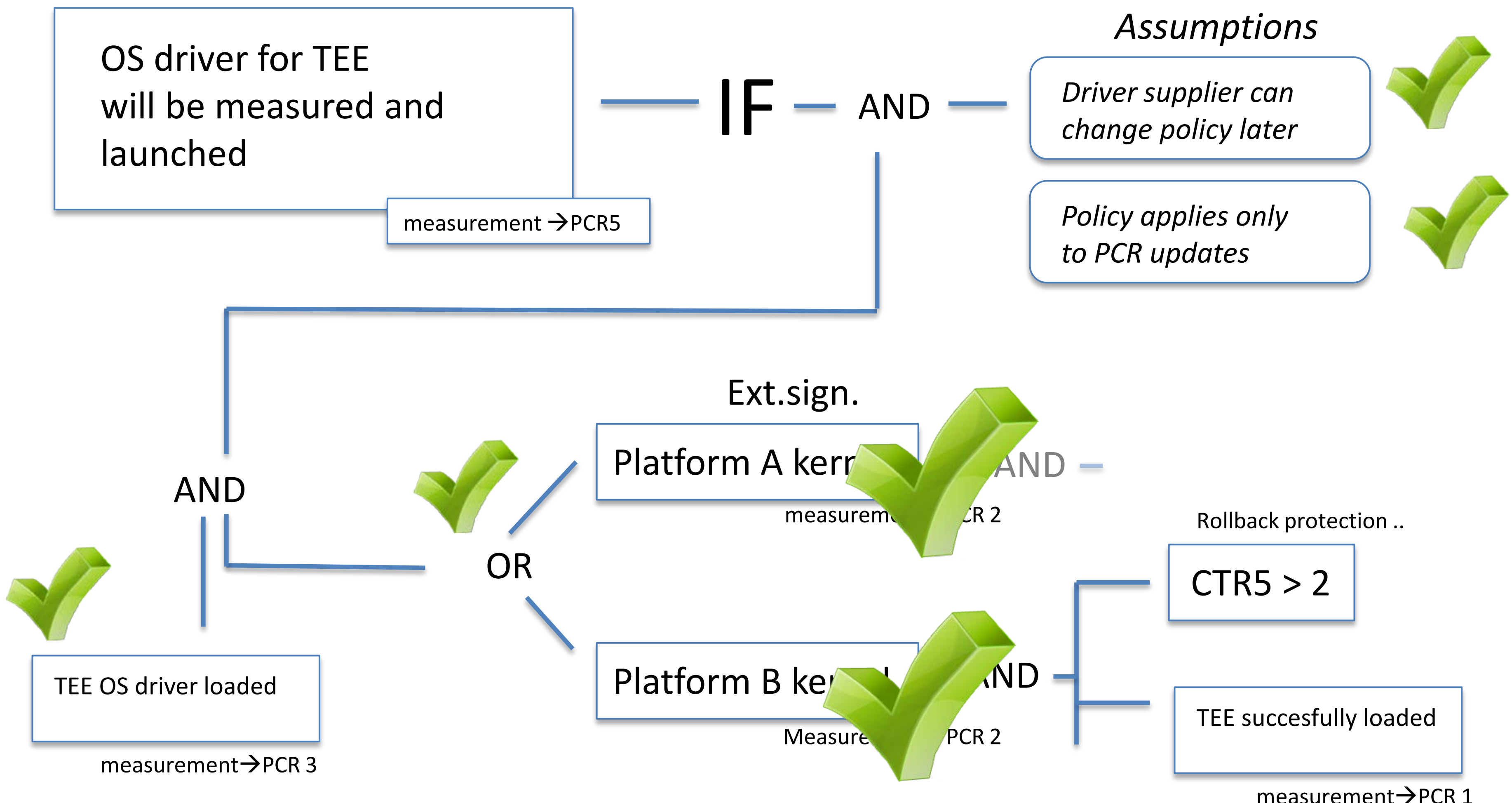


V1 →
V2 → PolicyOr({V1,V2} →**W** → PolicyPCR(3, meas.) → **Z**

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

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

Example policy



PolicyPCR(3, H(...)) →**V1** →
PolicyPCR(3, H(...)) →**V2** →
PolicyOr({V1,V2} →**W** → PolicyPCR(3, meas.) → **Z**

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

Sequence of TPM commands (1/2)

Assume PCR2 will have value mB if a kernel authorized by provider B (such as platform B kernel was booted, and PCR1 will have mN if the correct TEE dríver N was loaded

- V1 <- PolicyPCR (2, mB)
- W <- PolicyOR ({V1, V2})
- Z <- PolicyPCR (1, mN)
- Y <- PolicyCommandCode (PCRExtend)
- X <- PolicyAuthorize (sig), where sig = Sig_A (Y)

→PCRExtend(5, measurement value)

authPolicy for PCR5 is X

Sequence of TPM commands (2/2)

V1 = h (0 || PolicyPCR || 2 || mB)

W = h(0 || PolicyOR || (V1 || V2))

Z = h (W || PolicyPCR || 1 || mN)

Y = h (Z || PolicyCommandCode || PCR_Extend)

X = h (0 || PolicyAuthorize || PK_A)

Standards summary

- Global Platform Mobile TEE specifications
  - Sufficient foundation to build trusted apps for mobile devices
- TPM 2.0 library specification
  - TEE interface for various devices (also Mobile Architecture)
  - Extended Authorization model is (too?) powerful and expressive
- Mobile deployments can combine UEFI, NIST, GP and TCG standards
- Developers do not yet have full access to TEE functionality

Challenges ahead and summary

A LOOK AHEAD

Open issues and research directions

1. Novel mobile TEE architectures

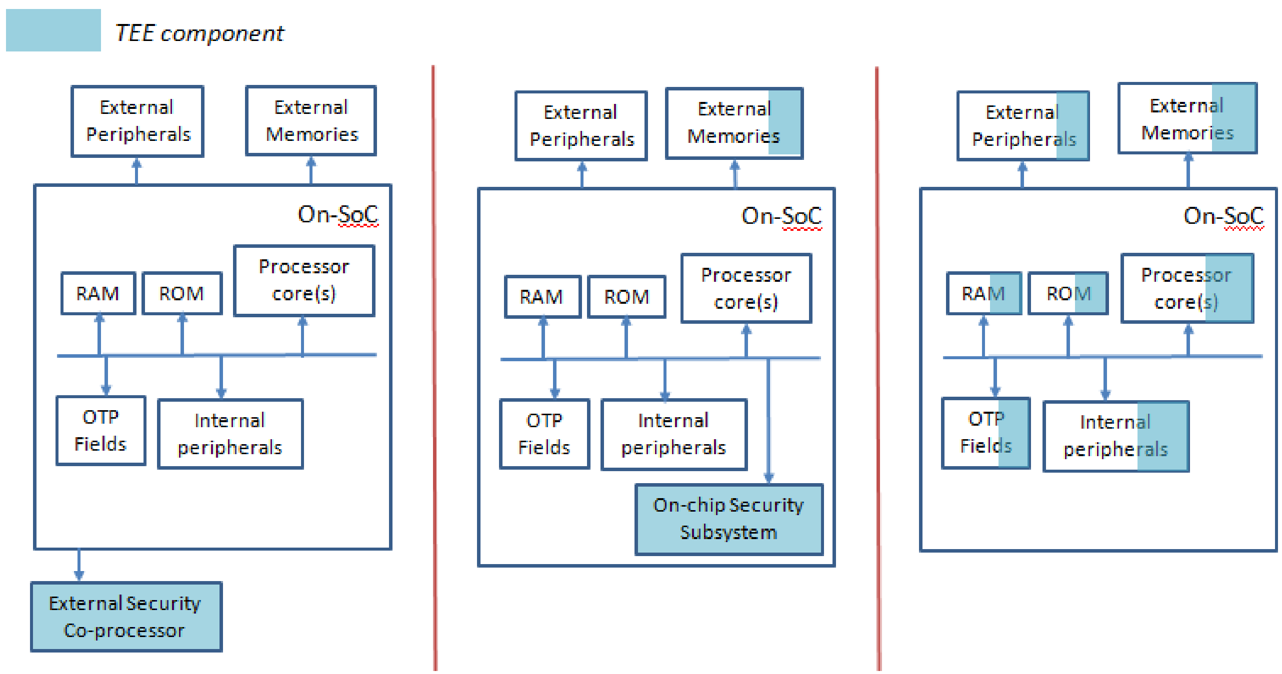
2. Issues of more open deployment

3. Trustworthy TEE user interaction

4. Hardware security and user privacy

Skip to Summary

Novel mobile TEE architectures



- Multiple cores?
- Low-cost alternatives?

TEE architectures for multi-core

- Issues to resolve
  - Possible to have separate TEEs for each core?
  - Can other cores run REE, while TEE active on one?

- SICE
  - Architecture for x86
  - Assigns **one or more cores for each TEE**
  - Other cores can run REE simultaneously
  - Azab et al. SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-core Platforms. CCS'11.

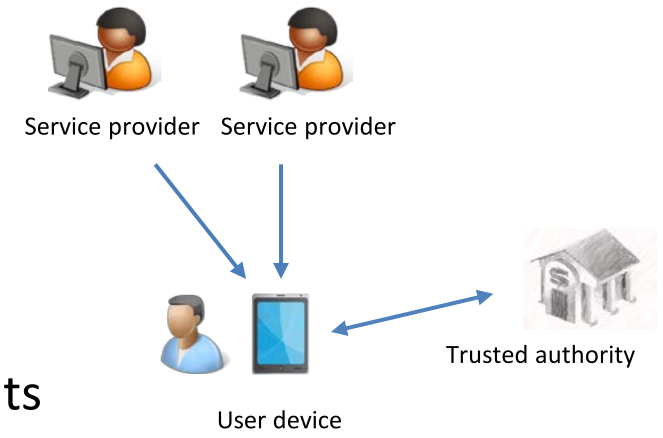
Low-cost mobile TEE architectures

- Can mobile TEEs made cheaper?
  - Low-end phones and embedded mobile devices

- TrustLite
  - Execution aware memory protection
  - Modified CPU exception engine for interrupt handling
  - Koeberl et al. TrustLite: A Security Architecture for Tiny Embedded Devices. EuroSys'14.

- SMART
  - Attestation and isolated execution at minimal hardware cost
  - Custom access control enforcement on memory bus
  - Defrawy et al. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. NDSS'12.

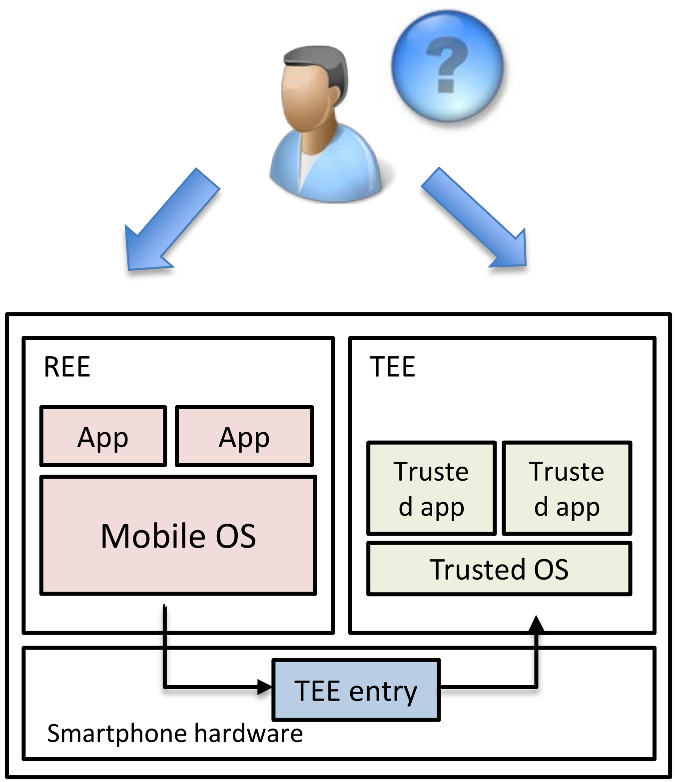
Issues of open deployment



- Certification and liability issues?
  - Especially application domains like payments
- Credential lifecycle management
  - Device migration becomes more challenging in open model
  - Hybrid approach: open provisioning and centralized assisting entity
  - Kostiainen et al. Towards User-Friendly Credential Transfer on Open Credential Platforms. ACNS’11.

Trustworthy user interaction

- Trustworthy user interaction needed
  - Provisioning
  - User authentication
  - Transaction confirmation
- Technical implementation possible
- But how does the user know?
  - Am I interacting with REE or TEE?



Hardware security and user privacy?

- Secure boot **can** be used to limit user choice
  - Common issue of mechanism vs. policy
- Allows new opportunities for attackers
  - Vulnerabilities in TEE implementation → rootkits
  - Thomas Roth. Next Generation rootkits. Hack in Paris 2013.

Summary

- Hardware-based TEEs are widely deployed on mobile devices
  - But access to application developers has been limited
- TEE functionality and interfaces are being standardized
  - Might help developer access
  - Global Platform TEE architecture
  - TPM 2.0 Extended Authorization and Mobile Architecture
- Open research problems remain

Further reading

- **Mobile Trusted Computing.** Proceedings of the IEEE 102(8): 1189-1206 (2014)
- **The Untapped Potential of Trusted Execution Environments on Mobile Devices**. IEEE Security & Privacy Magazine 12(4):29-37 (2014)
- **Citizen Electronic Identities using TPM 2.0**, To appear in ACM CCS TrustED workshop (2014) (arXiv:1409.1023)

Slides of this talk: http://asokan.org/asokan/TCE2014

Contact info: http://asokan.org/asokan/