Lecture 4

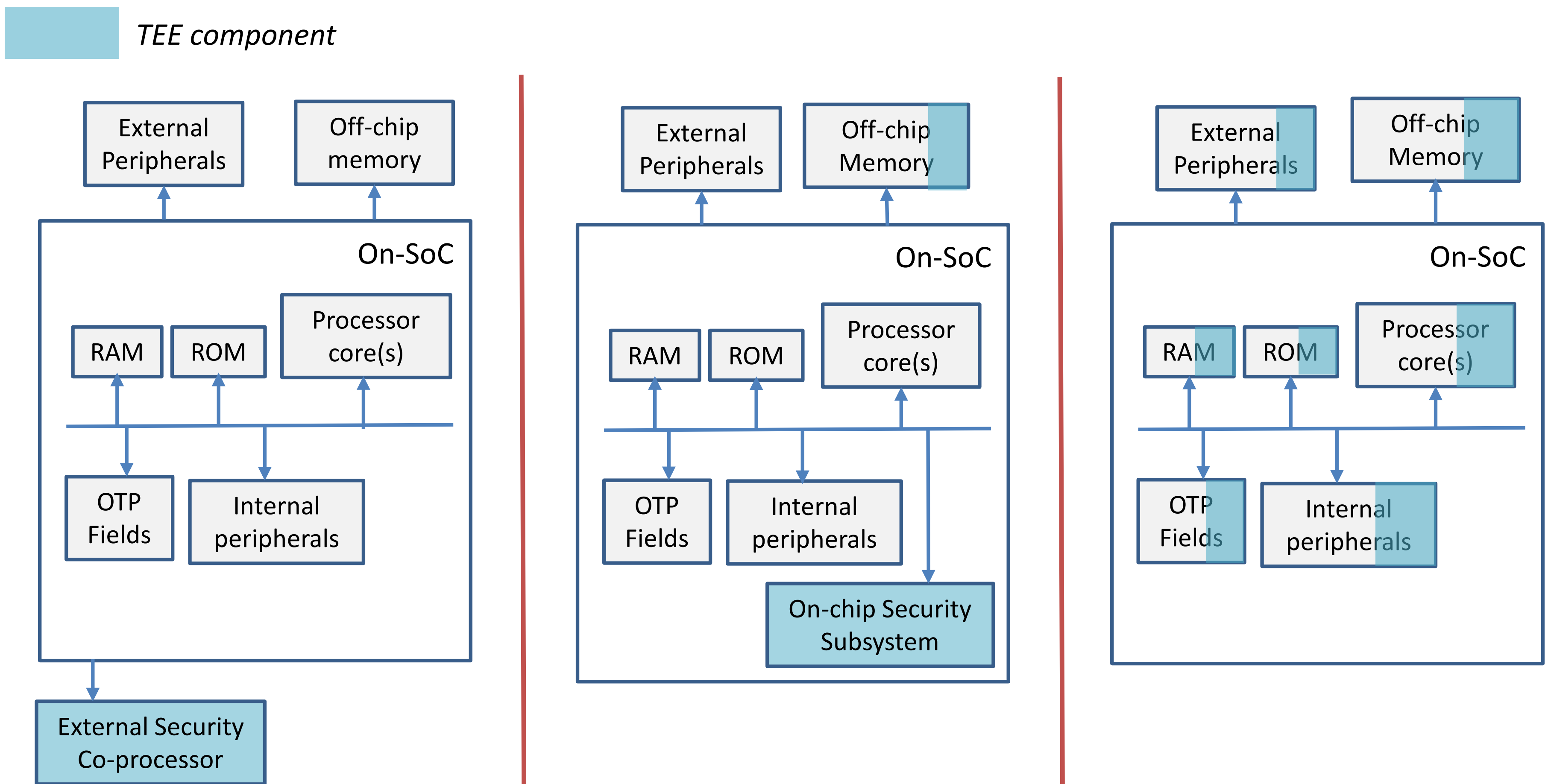
HARDWARE SECURITY ENABLERS

You will be learning:

- What are example instances of hardware platform security?
  - Fixed function TEEs: Trusted Platform Module (TPM)
  - Programmable TEEs:
    - ARM TrustZone
    - Intel Software Guard Extensions (SGX)
  - Standardized interfaces for using TEEs

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

TEE hardware realization alternatives



External Secure Element (TPM, smart card)

Embedded Secure Element (smart card)

Processor Secure Environment (TrustZone, M-Shield)

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

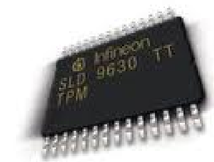
TEE Specifications: www.trustedcomputinggroup.org

TRUSTED COMPUTING GROUP TPM / TPM2

Trusted Platform Module (TPM)

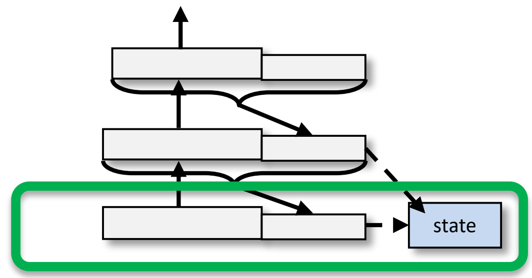
- Collects state information about a system
  - separate from system on which it reports

- For remote parties
  - well-defined **remote attestation**
  - **Authorization** for functions/objects in TPM



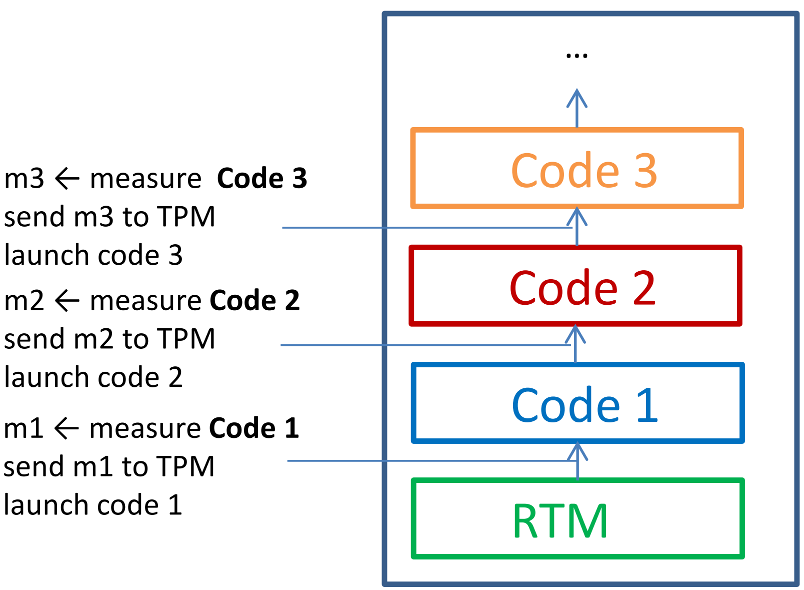
- Locally
  - **Generation/use** of TPM-resident keys
  - **Sealing:** Securing data for **non-volatile storage** (w/ binding)
  - **Engine** for cryptographic operations

Platform Configuration Registers (PCRs)



Authenticated boot

- 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)



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

$H_0 = 0$

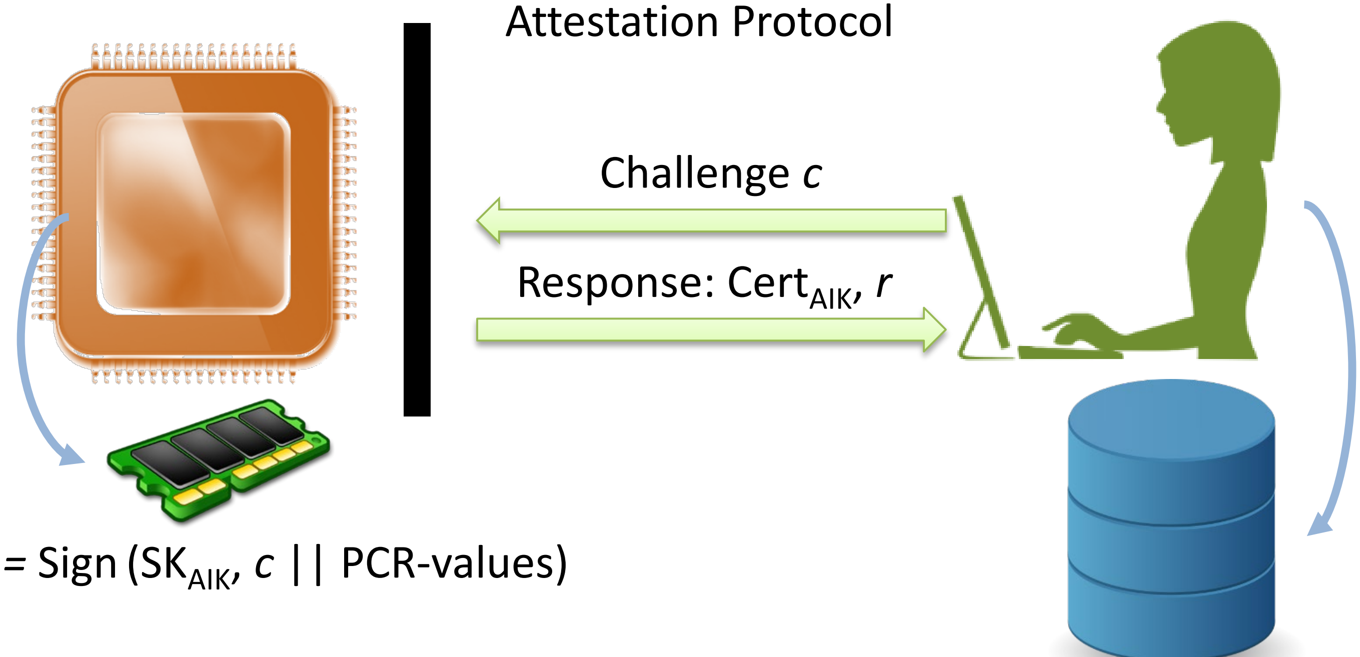
$H_1 = H(0|m1)$

$H_2 = H(H(0|m1)\ |m2)$

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

TPM Remote Attestation

Goal: Check whether the prover is in a trustworthy state



"TPM Quote" r = Sign (SK_{AIK}, c || PCR-values)

Attestation Identity Key (AIK) is a unique keypair whose private key (SK_{AIK}) is *TPM-protected*
$Cert_{AIK}$ certificate for PK_{AIK} issued by, e.g., manufacturer

Sealing

Goal: Bind secret data to a specific configuration

- E.g.,
  - create RSA keypair PK/SK when PCR_X is Y
  - bind private key: $Enc_{SRK}(SK, PCR_X=Y)$
    - SRK is known only to TPM (cf. "device key" K_D)
    - "**Storage Root Key**" (created on TPM "take ownership" process)
  - TPM will "unseal" key **iff** PCR_X value is Y
    - Y is the "reference value"

Isolated Execution with TPMs

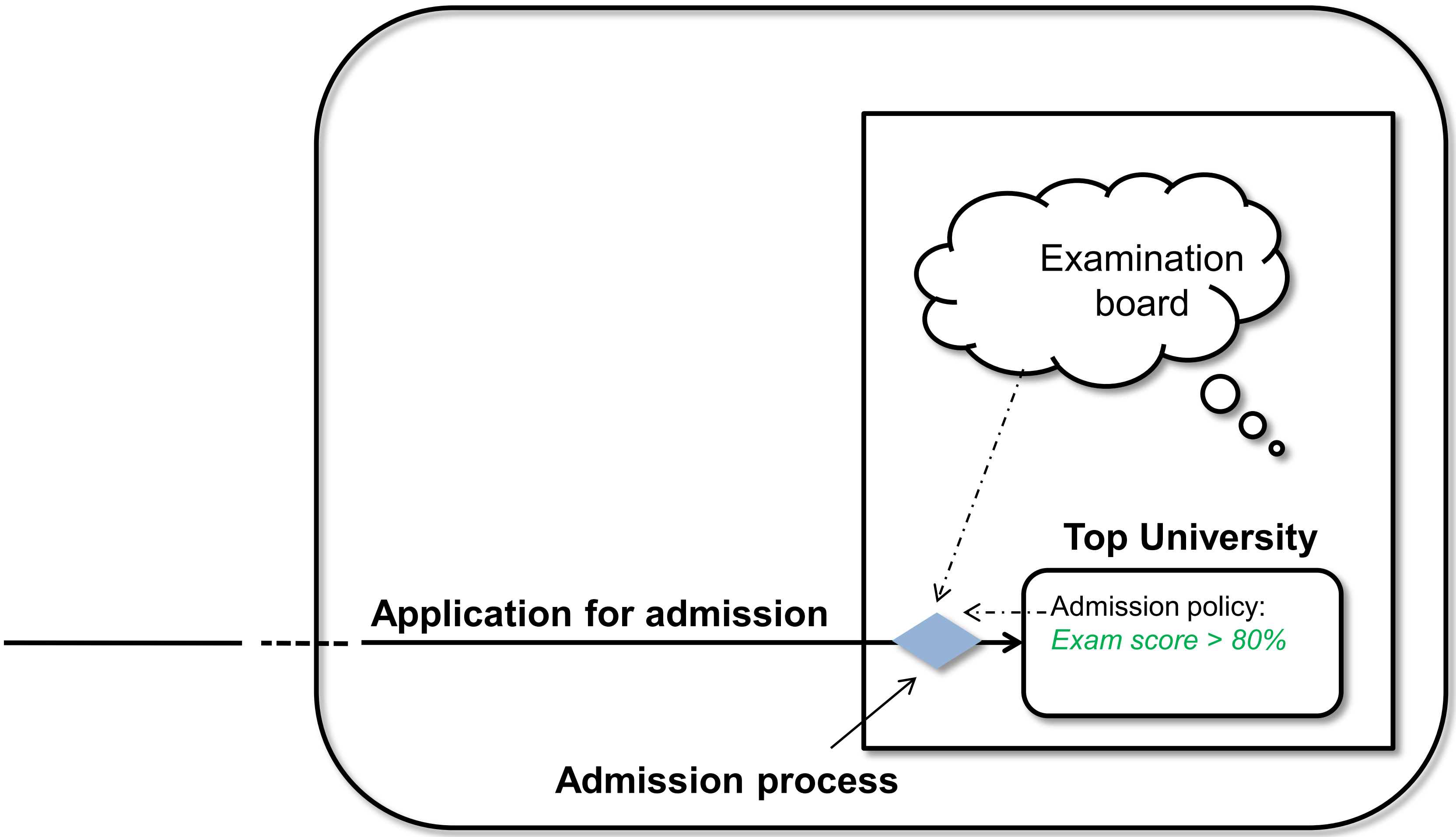
Dynamic RTM

- Dynamic PCRs (17-23) set to -1 on boot
- Special CPU instruction to
  - reset dynamic PCRs to 0
  - measure and extend a code block to PCR 17
  - launch that code
- "Late launch" of a hypervisor
- Can be used as a TEE for arbitrary code: Flicker by McCune et al: https://doi.org/10.1145/1352592.1352625

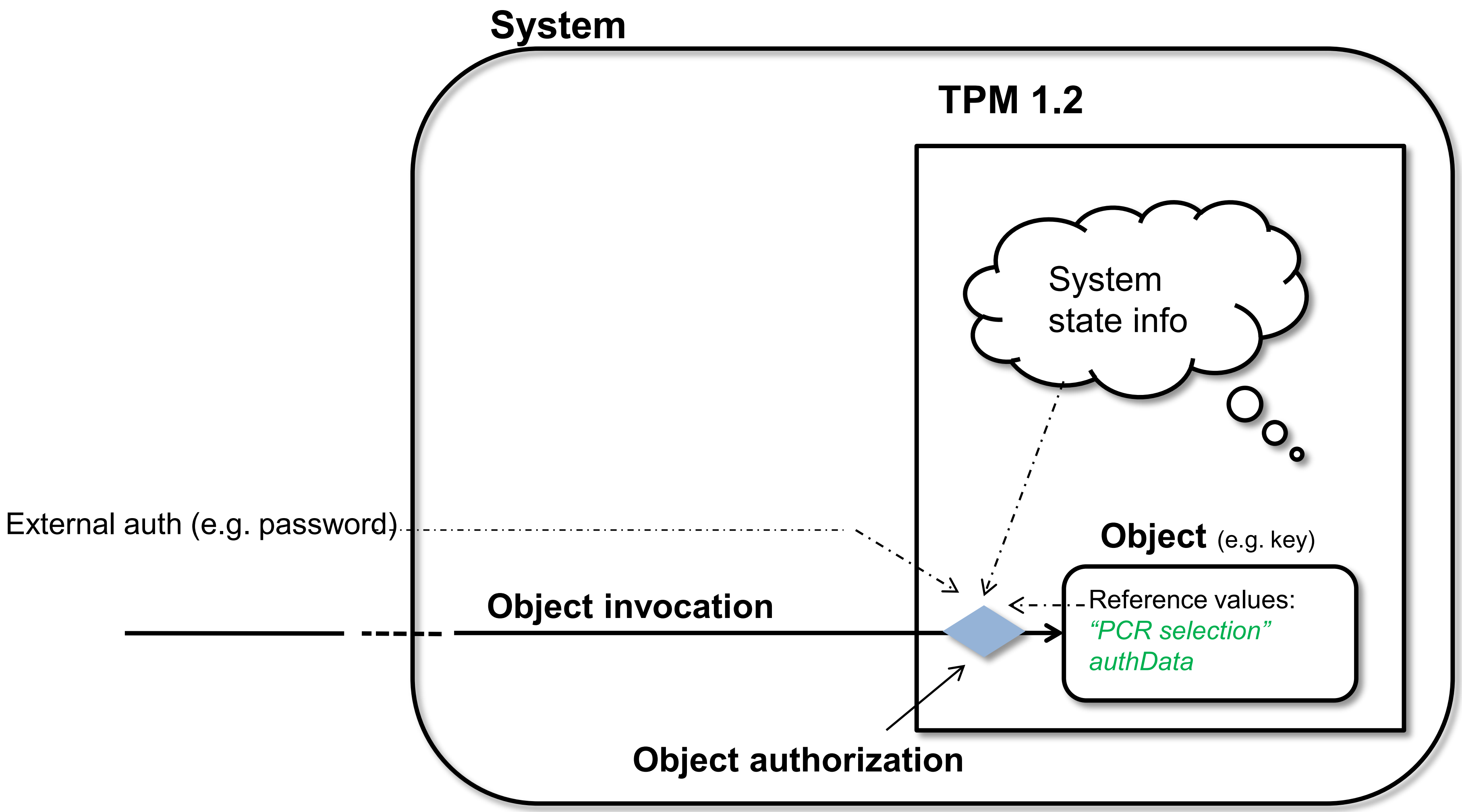
TPM authorization

- Authorization essential for access to sensitive TPM services/resources.
- TPMs have **awareness of system state** (cf., removable smartcards)

Authorization example: university admissions

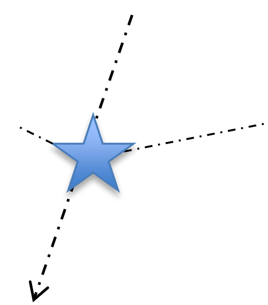


Authorization (policy) in TPM 1.2

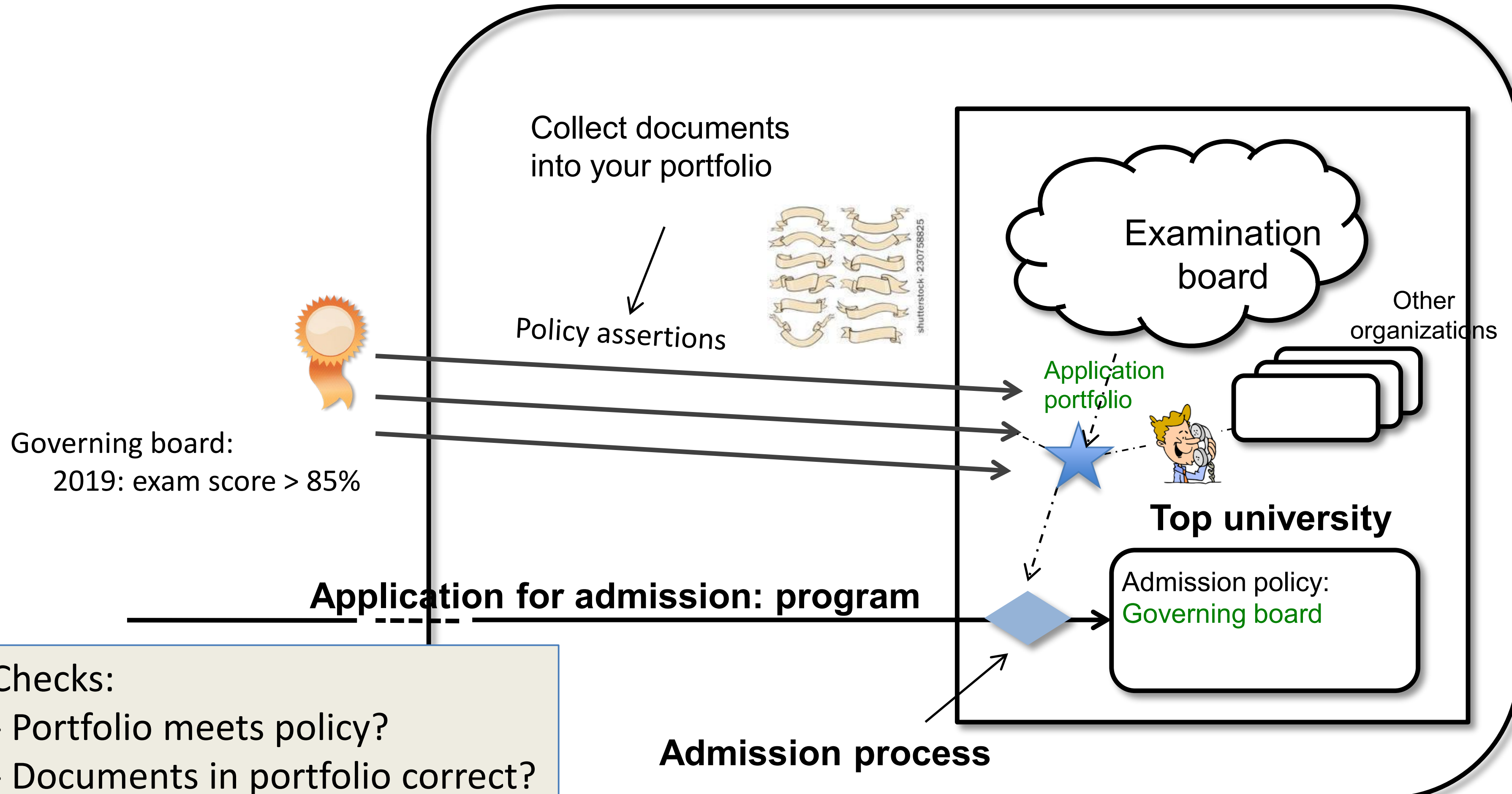


TPM 2.0

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



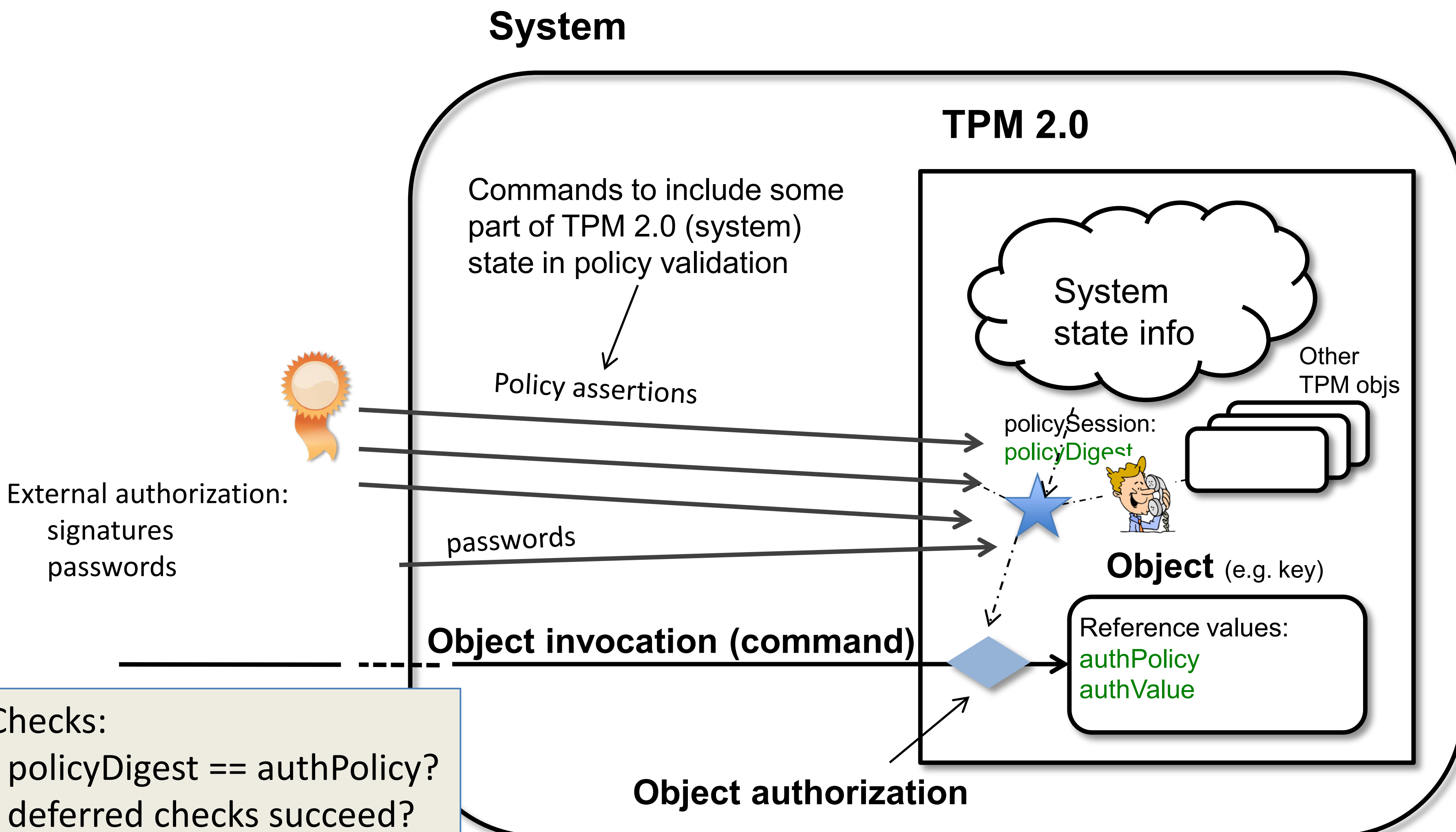
University admissions 2.0



Checks:

- Portfolio meets policy?
- Documents in portfolio correct?

Authorization (policy) in TPM 2.0

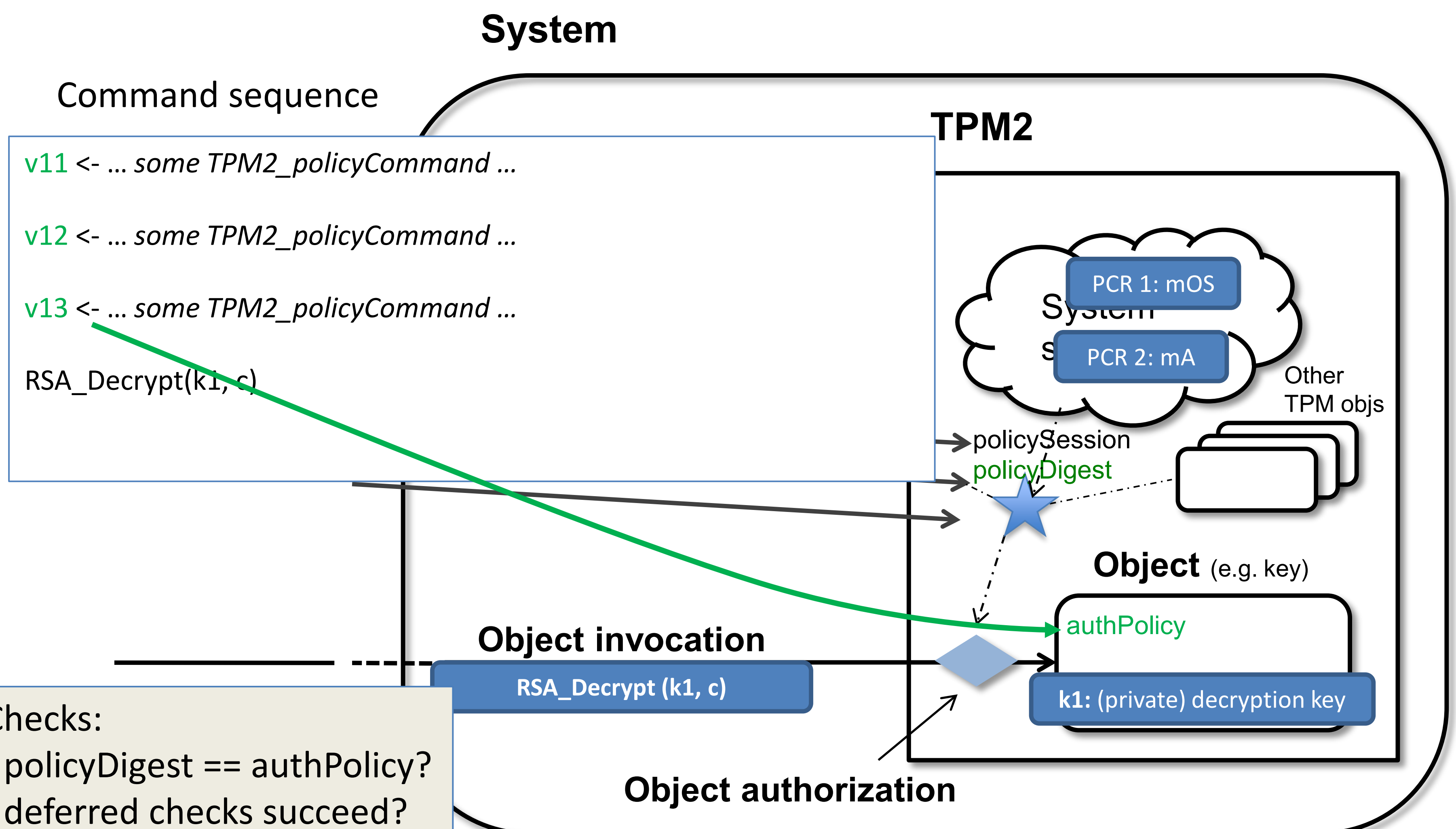


Checks:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == X?
  - PCR 1 Y == Z?

Authorization Policy Example

- Allow app A (and no other app) to use a TPM-protected RSA keypair **k1**
  - Only when a certain OS is in use
- Assume that
  - When right OS is used, **PCR 1 = mOS**
  - When app A in foreground, **PCR 2 = mA**

Enforcing the example policy



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

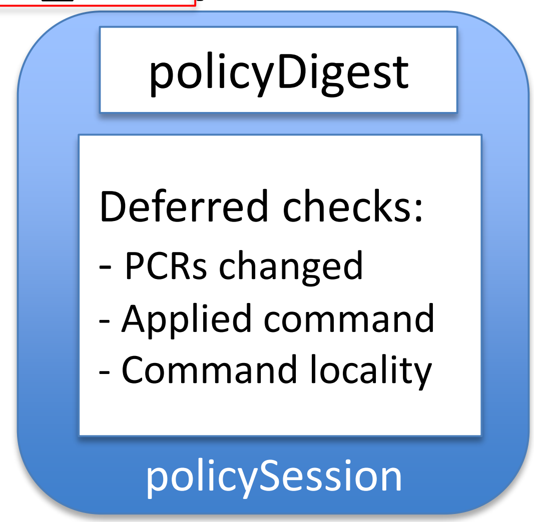
TPM2 Policy Session Contents

- accumulated session policy value: **policyDigest**

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

- Some policy commands **reset** value

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



- **deferred policy checks** at object access time.

TPM2 Policy Command Examples

- **TPM2_PolicyPCR:** PCR values

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

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

- **TPM2_PolicyNV:** 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:** Check command during "object invocation" :

  update *policyDigest* with *[command code]*

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

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

policySession->commandCode checked before object invocation!

Skip to example

Other policy commands

- **TPM2_PolicyOR:** Authorize one of several options:
  Input: *List* of digest values <D1, D2, D3, .. >

  IF *policyDigest* in *List* **THEN**
  newDigest := H(0 || TPM2_CC_PolicyOR || List)

- **TPM2_PolicyAuthorize:** Validate a signature on a policyDigest:
  Input: signature and pubic key
  IF signature validates **AND** signed text matches *policyDigest* **THEN**
  newDigest := H(0 || TPM2_CC_PolicyAuthorize|| **H(pub)**|| ..)

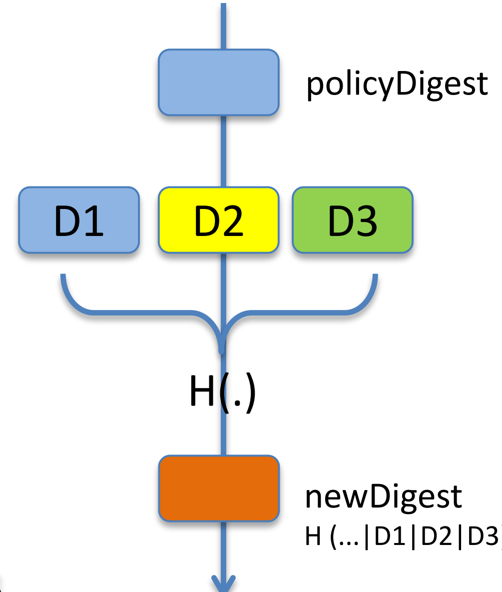
Policy disjunction

TPM2_PolicyOR: Authorize one of several options:

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

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

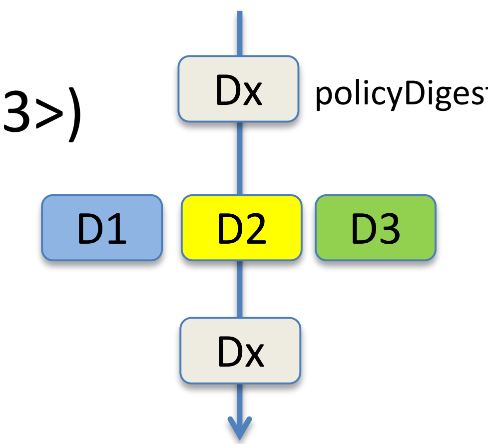
newDigest := H(0 || TPM2_CC_PolicyOR || List)



(Successful OR)

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

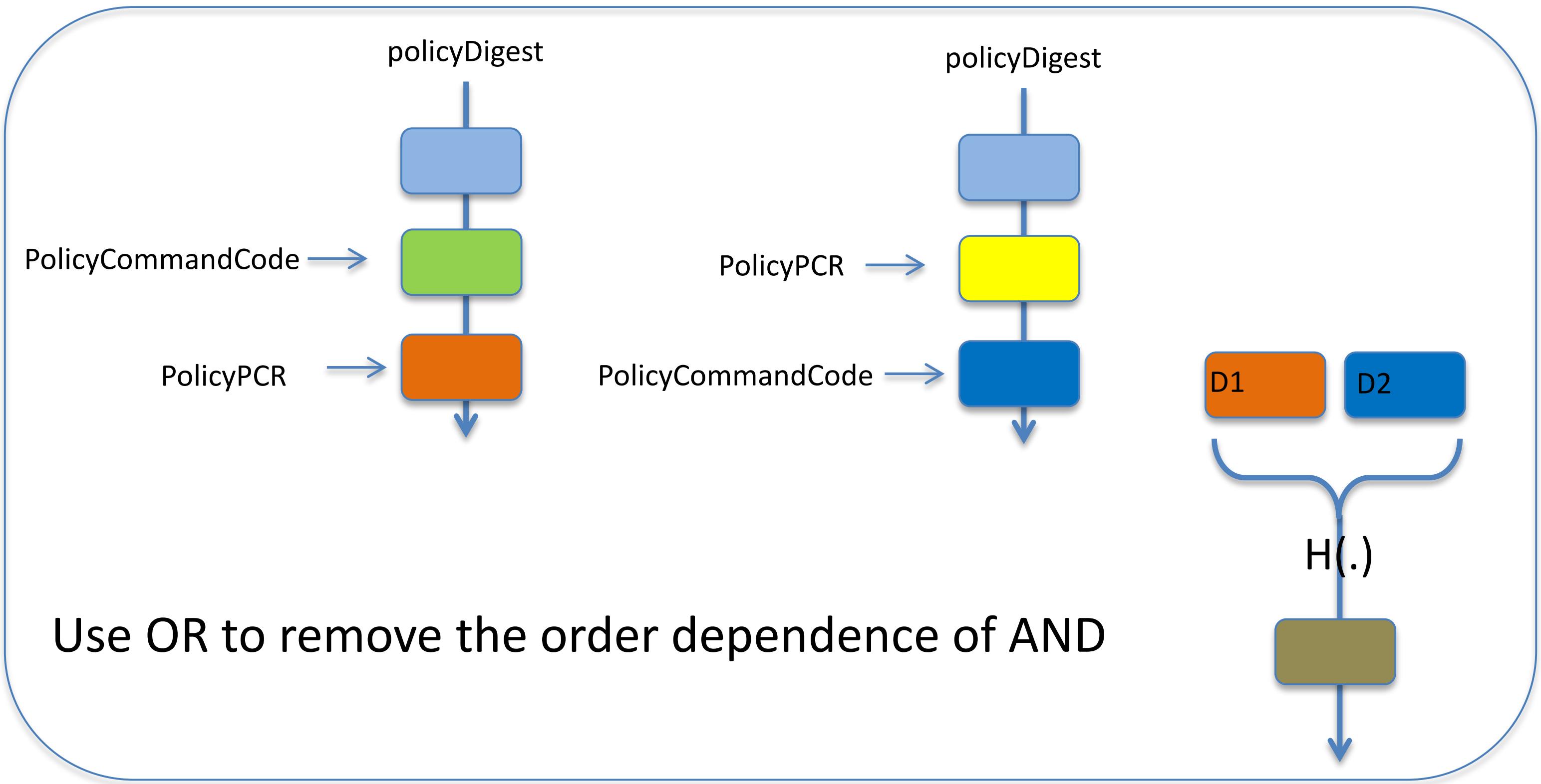
such that H(... |List) == H(... |List2)



(Failing OR)

Policy conjunction

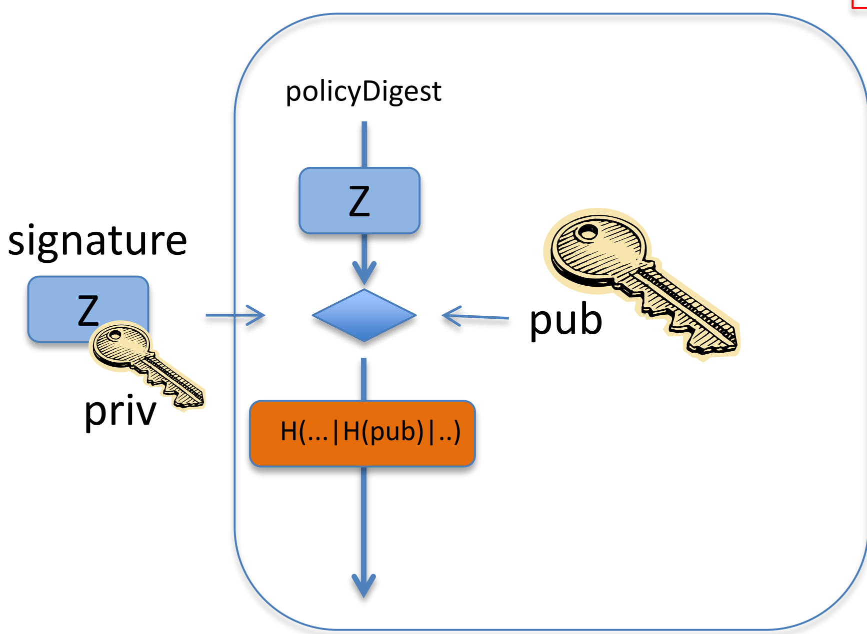
- No explicit AND command
- AND: consecutive auth. commands → order dependence

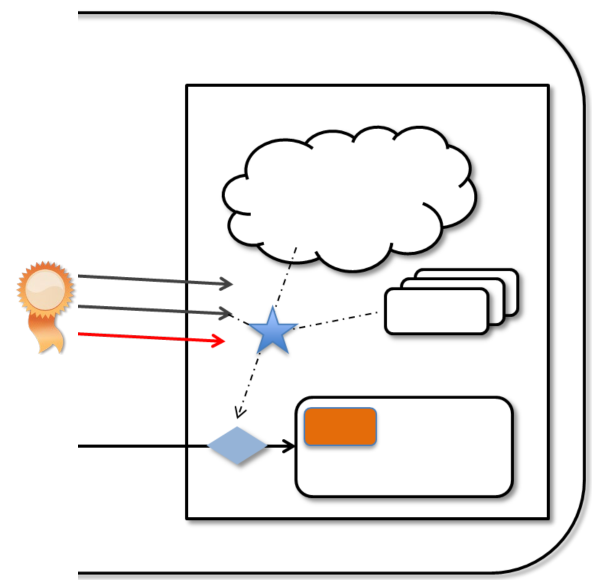


External Authorization

TPM2_PolicyAuthorize: Validate a signature on a policyDigest:

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





Let's try this out

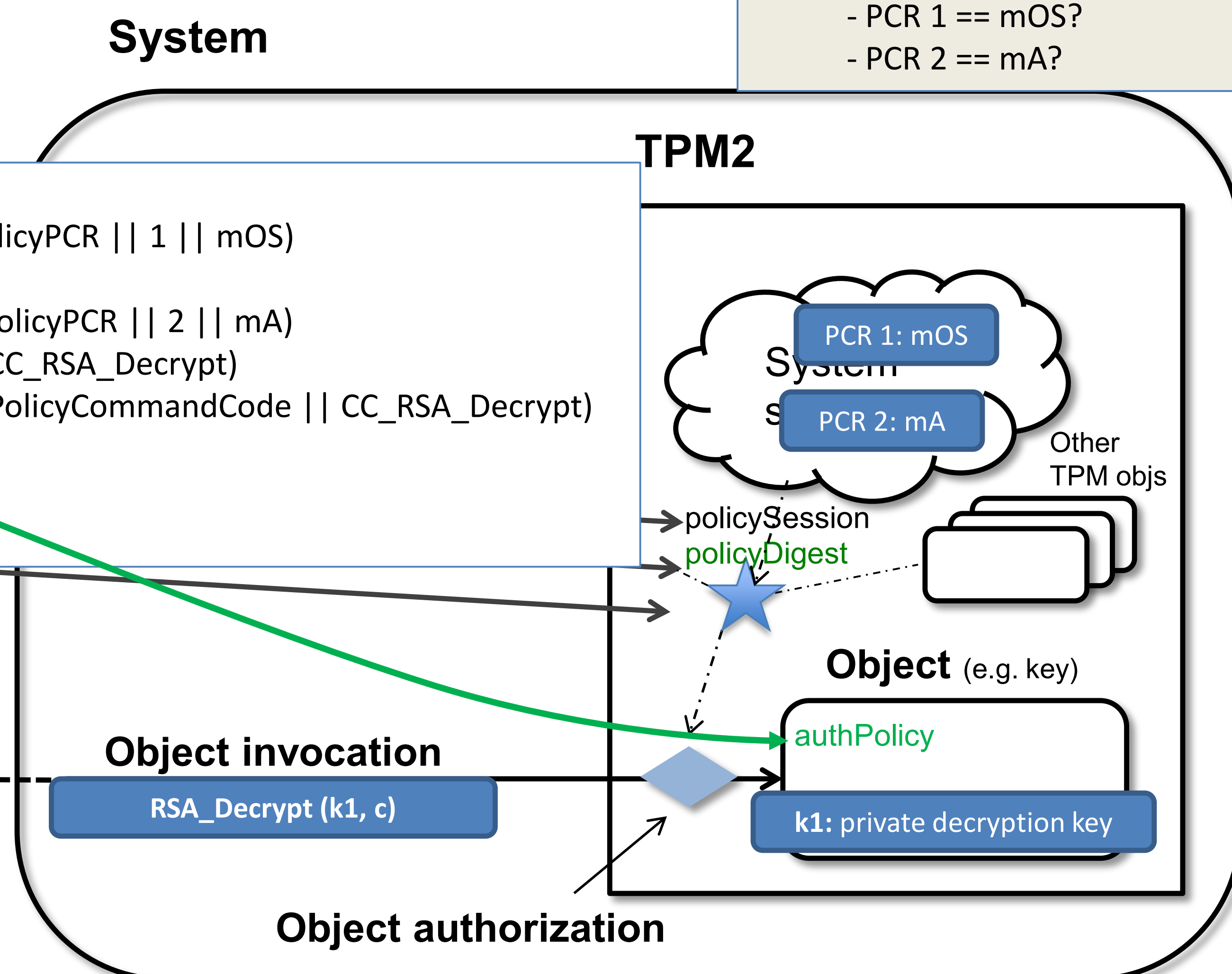
- Developer D
  - Has TPM2-protected keypair *k1* and Application A
  - Wants **only A** can use *k1* via
    - TPM2_RSA_Decrypt (key, ciphertext)
- Assume that
  - OS measured into PCR1 (if correct OS: PCR1 = mOS)
  - Foreground app into PCR2 (if A: PCR2 = mA)
- What should authPolicy of *k1* be?

Enforcing policy

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

Command sequence

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



NOTE: We drop “TPM2_” and “TPM_” prefixes for simplicity…

Exercise 3

(i) What if D wants to authorize

app A1 (PCR2=mA_1) ***or*** app A2 (PCR2=mA_2)

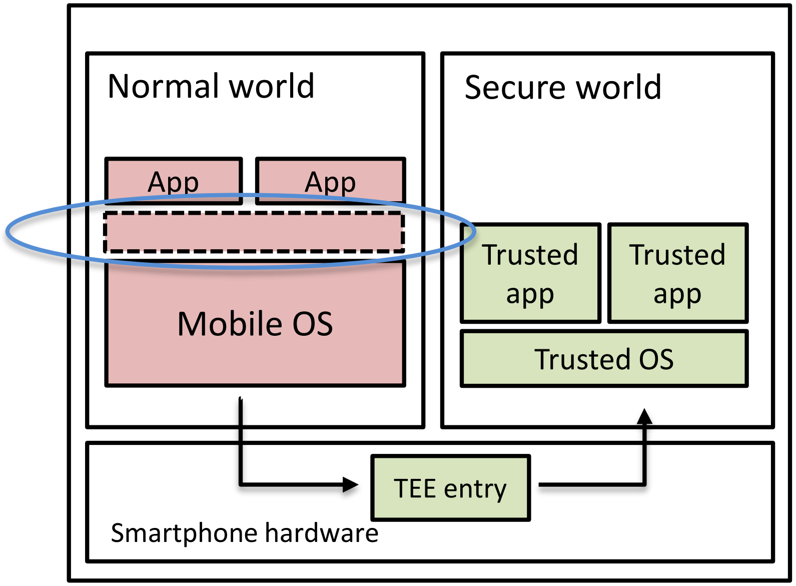
(ii) What if D wants to authorize many apps

Using TEEs

ANDROID KEYSTORE

Mobile TEE deployment

- TrustZone support available in majority of current smartphones
- Mainly used for manufacturer internal purposes
  - Digital rights management, Subsidy lock…
- *APIs for developers?*



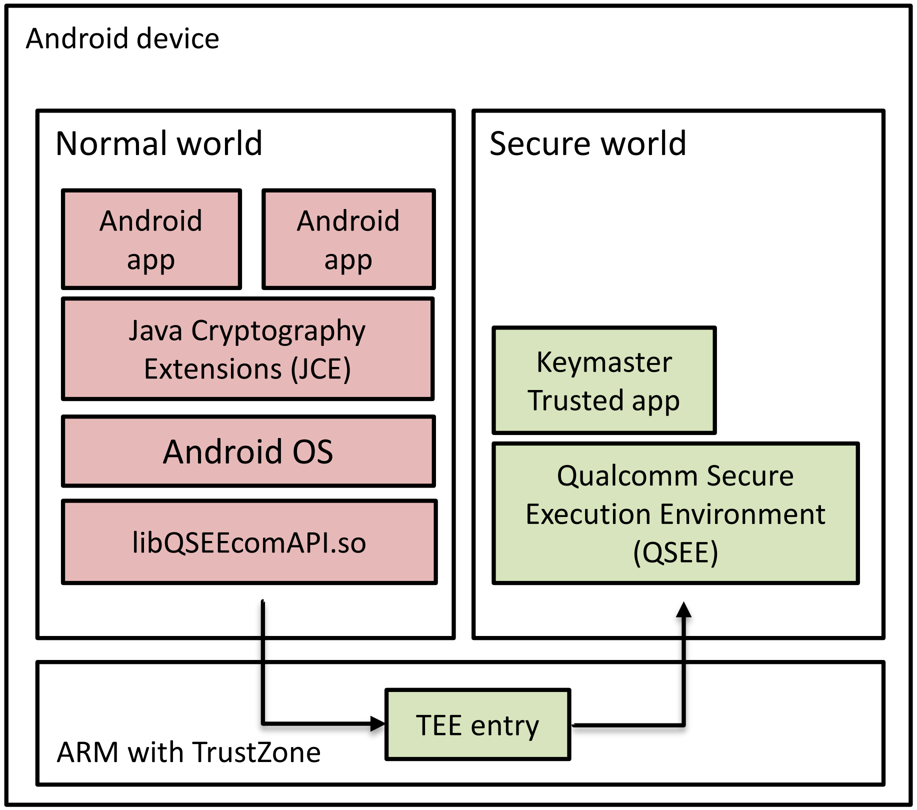
Android Key Store API

Android Key Store example

```
// create RSA key pair
Context ctx;
KeyPairGeneratorSpec spec = new
      KeyPairGeneratorSpec.Builder(“key1”,KeyProperties.PURPOSE_SIGN);
…
spec.build();

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

// make a signature
Signature sig = Signature.getInstance(“SHA256withRSA/PSS”):
sig.initSign(kp.getPrivate()):
```

Android, Hardware-backed Keystore, 2015-2018

Key Store implementation: example



Keymaster operations

- Public key algorithms
- Symmetric key algorithms (AES, HMAC) from v1.0
- Access control, key usage restrictions
- Key attestation (from v2.0), “ID attestation” (from v3.0)
- Android Protected Confirmation (Android 9, API level 28)

Persistent storage on Normal World

Elenkov. Credential storage enhancements in Android 4.3. 2013
Android, Hardware-backed Keystore, 2015-2018
Android, Protected Confirmation, 2018

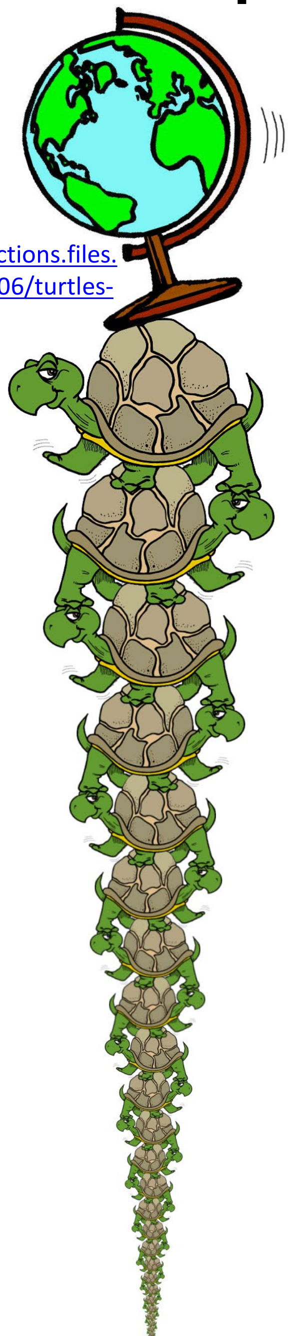
Android Key Store

- Available operations
  - Signatures
  - Encryption/decryption
  - Attestation, confirmation

- Developers cannot utilize programmability of mobile TEEs
  - Not possible to run arbitrary trusted applications

- Different API abstraction and architecture needed
  - Example: On-board Credentials
  - GlobalPlatform device working group specifications

What protects hardware platform security?

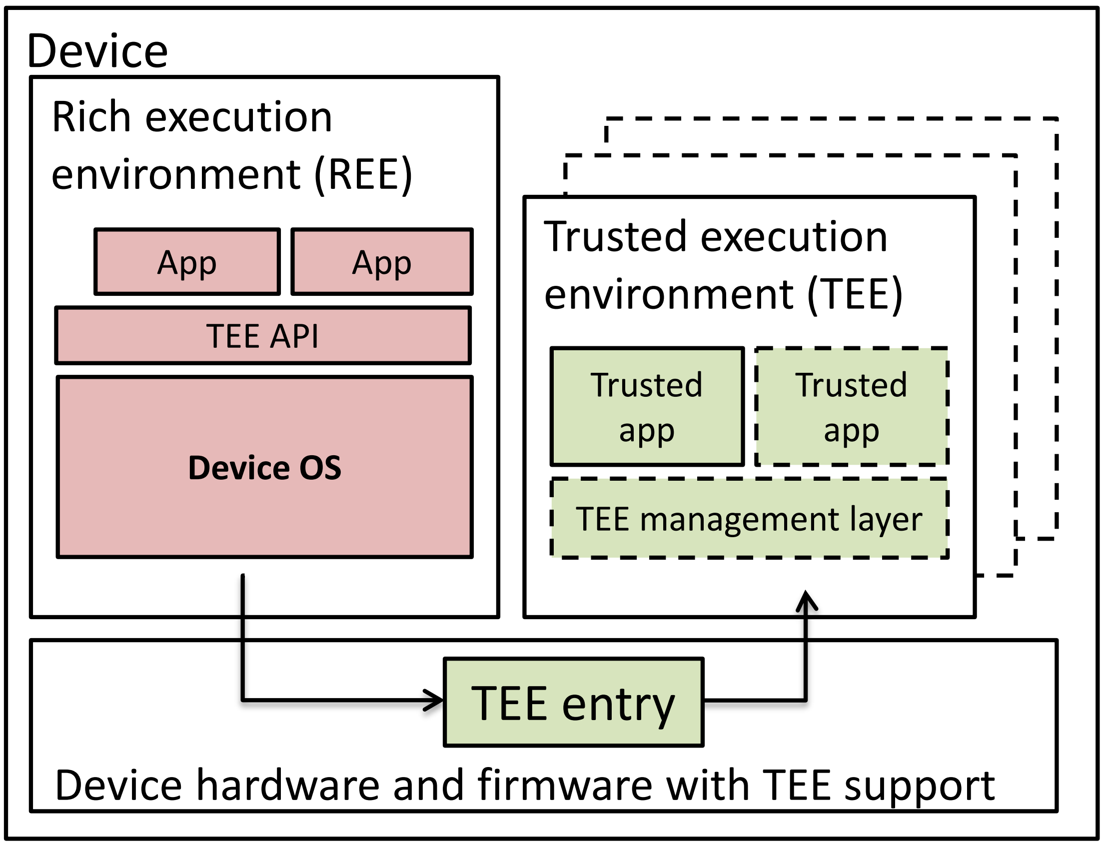


http://transversalinflections.files.wordpress.com/2011/06/turtles-all-the-way-down.png

A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever," said the old lady. "**But it's tortoises all the way down**!“

- Stephen Hawking, in *A Brief History of Time*

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

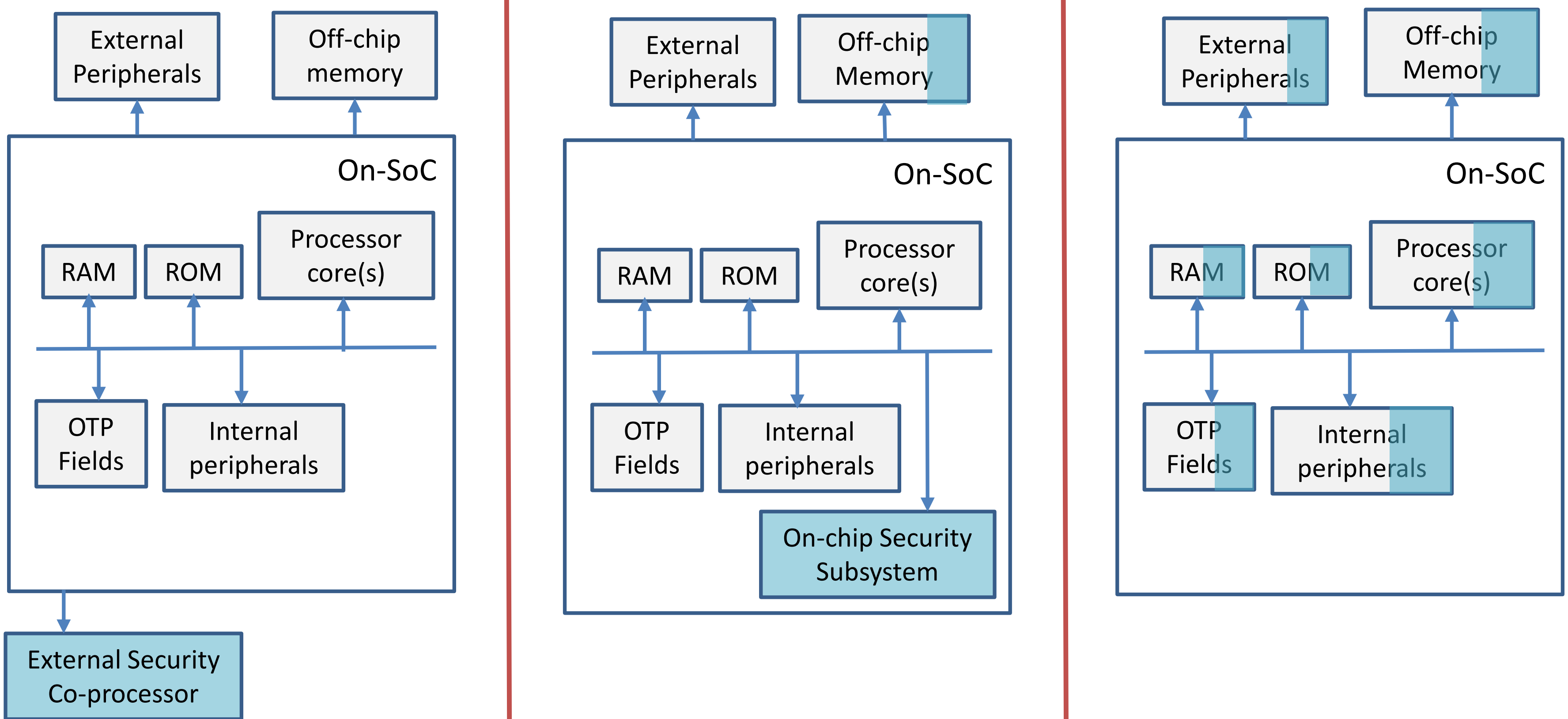
- Intel SGX
- TPM (and "Late Launch")
- Hypervisor

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

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

TEE hardware realization alternatives

TEE component



External Secure Element (TPM, smart card)

Embedded Secure Element (smart card)

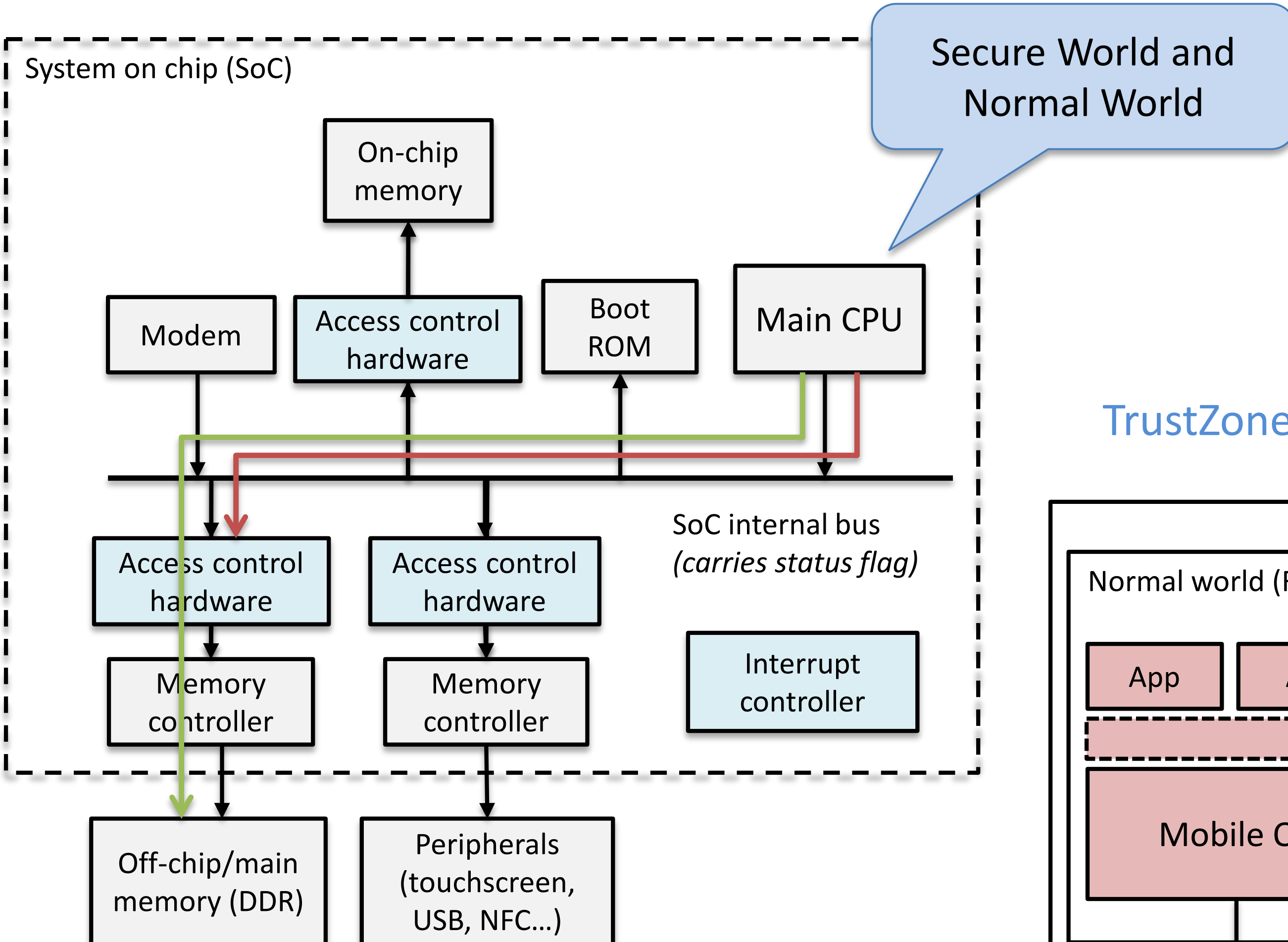
Processor Secure Environment (TrustZone, M-Shield)

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

TEE instances

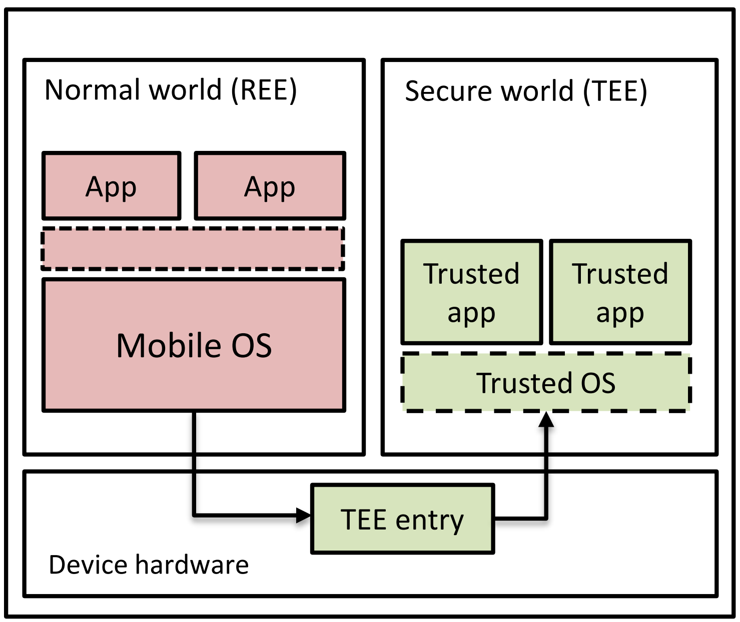
ARM TRUSTZONE

ARM TrustZone architecture



TrustZone hardware architecture

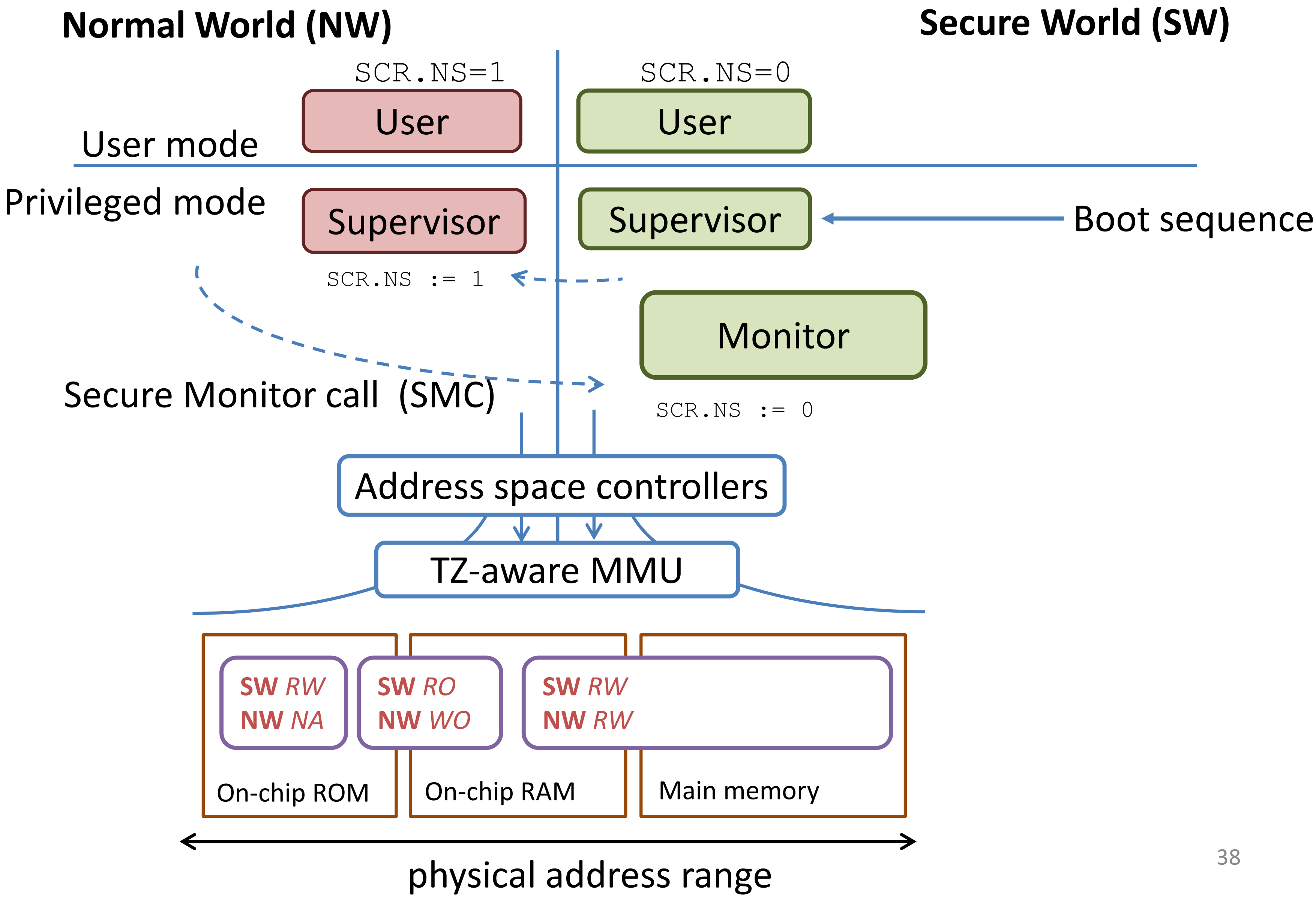
TrustZone system architecture



ARM Ltd., "ARM Security Technology - Building a Secure System using TrustZone Technology", Whitepaper 2009

Legend:
MMU: memory management unit

TrustZone overview



TrustZone example (1/2)

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

Boot sequence → Secure World Supervisor

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

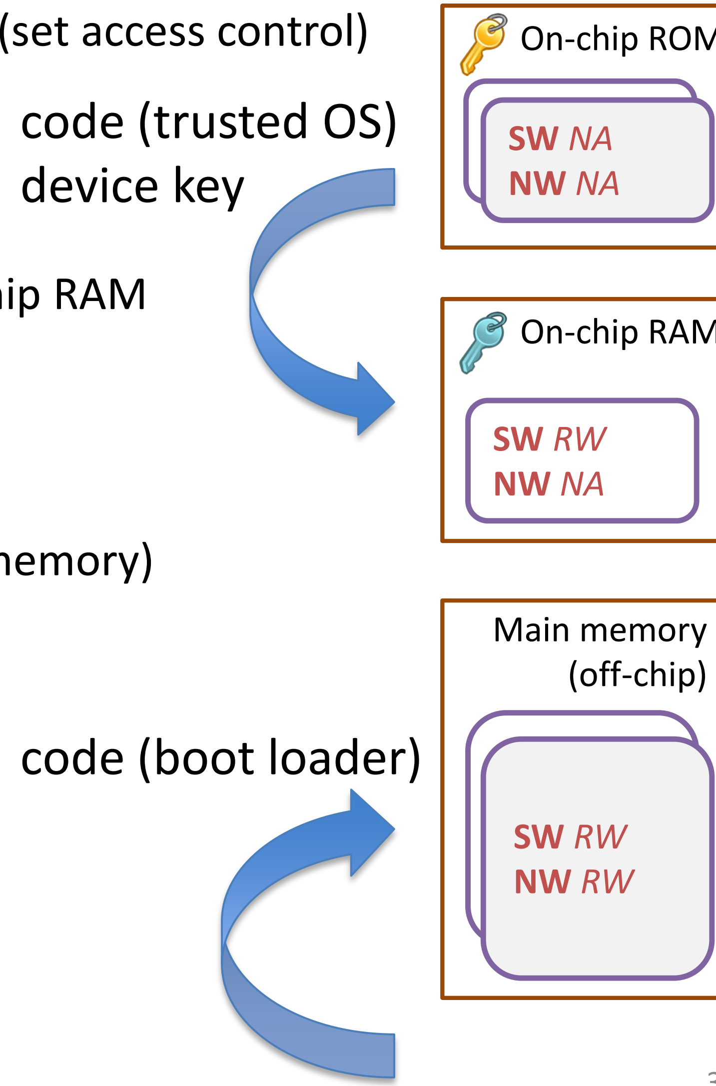
Secure World Supervisor

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

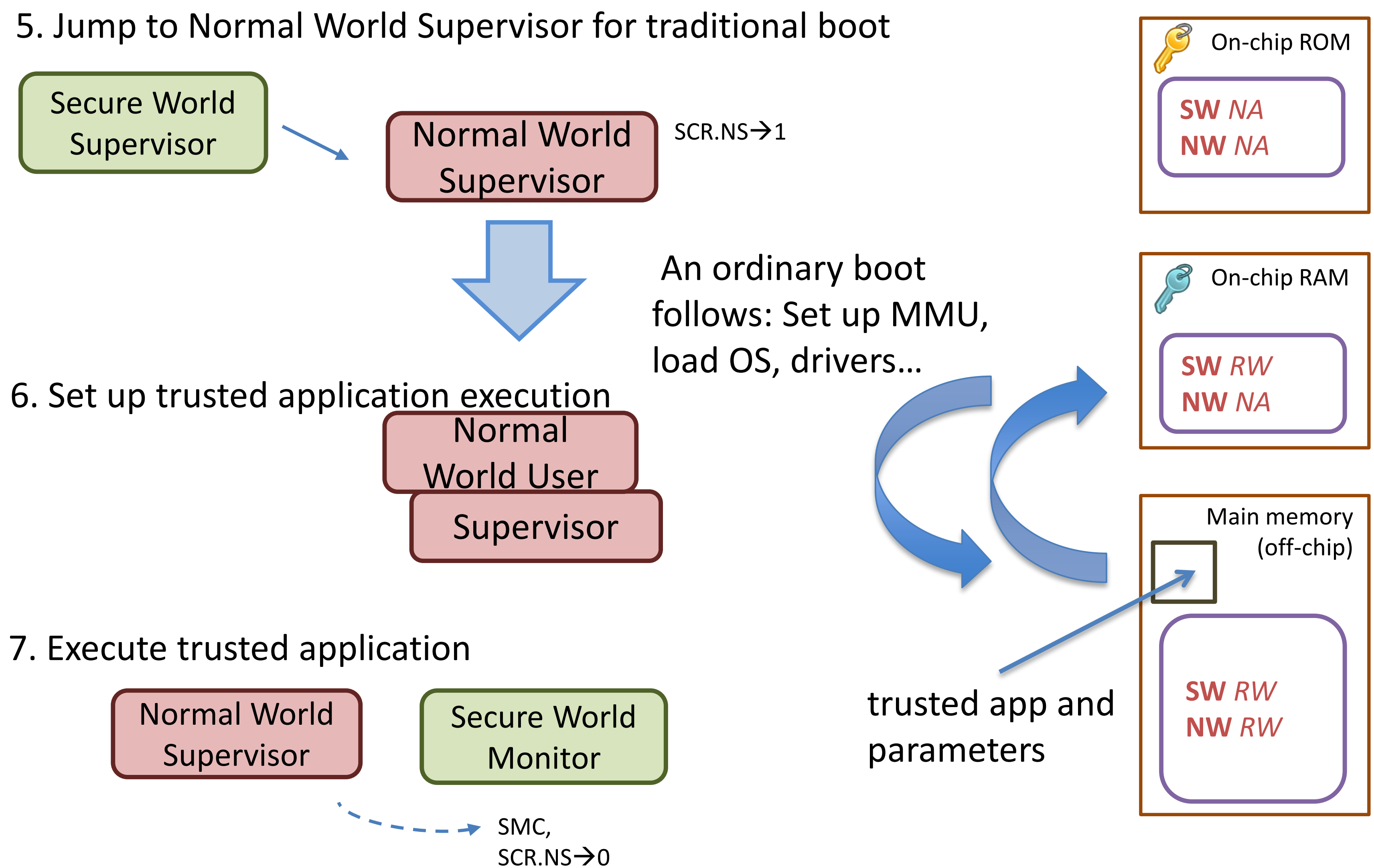
Secure World Supervisor

4. Prepare for Normal World boot

Secure World Supervisor



TrustZone example (2/2)



Skip to GP

TZ-enabled CPUs

- TZ: set of ARM processor extensions
- Combined with other building blocks needed for TEEs
  - Trust root to verify code (e.g., hash of manufacturer's code signing key)
  - Device-secret initialized during chip manufacture
  - Monotonic counter or writable secure memory

Skip to GP

Secure state entry/exit in TrustZone

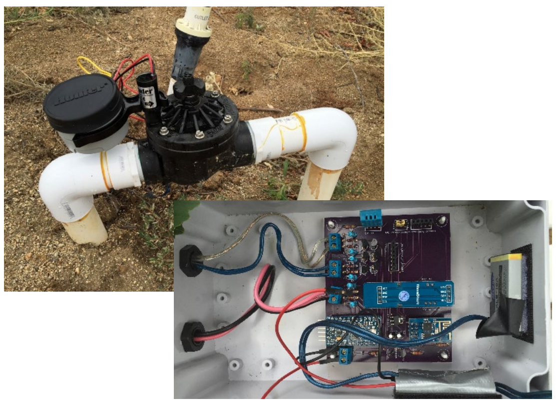
What happens during entry/exit?

- Store/restore all shared registers
  - Kernel: switching between processor modes
  - Secure monitor: switching between worlds
- Validate/(un)marshal parameters
  - TEE driver
- Reconfigure MMU
  - Secure monitor

Register banking: copies of registers

- Special purpose registers (SP, LR, SPSR)
  - Banked between modes, but not worlds
  - except at **highest privilege mode**
- Ordinary registers are not banked

Internet of resource constrained things



Solar-powered soil-moisture sensor for agricultural irrigation
https://hackaday.io/project/6444-vinduino-a-wine-growers-water-saving-project

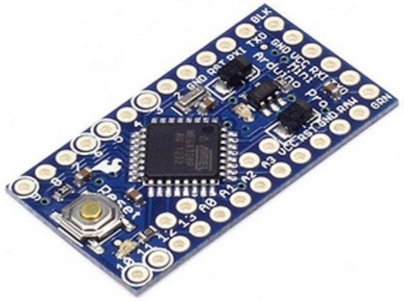
Wireless-enabled wearable activity tracker
https://en.wikipedia.org/wiki/Fitbit *(MorePix)*

Remote-controlled consumer smart lighting platform
http://www.ikea.com/se/sv/catalog/categories/departments/lighting/36812/

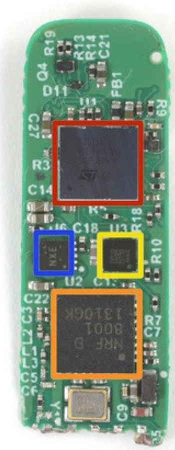
Wireless vehicle-presence sensor with 7 to 10 years of battery life
http://embedded-computing.com/articles/sensor-enabled-nodes-support-the-iot-for-smart-buildings-and-smart-transport/

Workhorses for small IoT devices

ATmega328
Up to **16 MHz** Clock Speed
Up to **2 KB** SRAM
Up to **32KB** Flash
Up to **1 KB** EEPROM
Wifi + **Long range RF** (external)

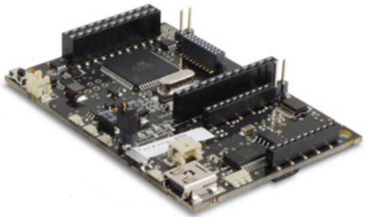


https://hackaday.io/project/6444-vinduino-a-wine-growers-water-saving-project
https://store.arduino.cc/arduino-pro-mini



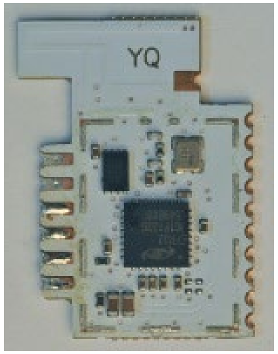
ARM Cortex-M3
Up to **32 MHz** Clock Speed
Up to **16 kB** RAM
Up to **4kB** EEPROM
Up to **128 kB** Flash
Bluetooth LE

https://www.ifixit.com/Teardown/Fitbit+Flex+Teardown/16050
http://www.st.com/en/microcontrollers/stm32l151c6.html



ATmega1281
14.74 MHz Clock Speed
8 kB SRAM
4 kB EEPROM
128 kB Flash
ZigBee (external)

http://www.libelium.com/v11-files/documentation/waspmote/smart-parking-sensor-board_eng.pdf
http://www.libelium.com/products/waspmote/hardware/



ARM Cortex-M4 + Floating Point Unit
Up to **40 MHz** Clock Speed
Up to **256 kB** RAM
Up to **1024 kB** Flash
ZigBee and **Thread** Radio (6LoWPAN)
Hardware Crypto Accelerator w/
AES-256/128, ECC, SHA-1, SHA-2

https://www.heise.de/make/artikel/Das-steckt-in-Ikea-Tradfri-3597295.html
https://www.silabs.com/products/wireless/mesh-networking/efr32mg-mighty-gecko-zigbee-thread-soc

Characteristics of a *resource constrained* IoT system

- Monolithic firmware written in embedded C/C++
  - interrupt-driven, reacts to external events
  - simple real-time scheduling O/S (or no O/S at all!)

- Execute-in-place from persistent storage (NOR flash)
  - reduces total RAM requirements
  - flat memory space (no virtual memory)
  - access control by Memory Protection Unit (MPU)

- Limited processing power, storage and memory
  - Restricted lifetime in battery operated devices

45

TrustZone-M (ARMv8-M)

```
    bl   <Secure_Function>
```

REE

In non-secure memory

```
Secure_Function:
    sg
    b    <Secure_Function_entry>
```

Secure gateway veneer in secure non-secure callable memory (NSC)

TEE

```
Secure_Function_entry:
    push     {r7, lr}
    ...
    pop  {r0}
    mov  lr, r0
    mov  r0, lr
    mov  r1, lr
    mov  r2, lr
    mov  r3, lr
    bxns lr
```

In secure memory

Memory regions labeled (secure, non-secure, NSC) during device initialization

ARM Ltd., "TrustZone technology for the ARMv8-M architecture", Whitepaper 2016

Nyman et al "CFI CaRE", RAID 2017

Secure state entry/exist in TZ-M

```
bl   <Secure_Function>
```

Branch to entry point in secure gateway veneer

In non-secure memory

```
Secure_Function:
    sg
    b    <Secure_Function_entry>
```

Secure Gate & branch to target in secure state

Secure gateway veneer in secure non-secure callable memory (NSC)

```
Secure_Function_entry:
    push      {r7, lr}
    ...
    pop  {r0}
    mov  lr, r0
    mov  r0, lr
    mov  r1, lr
    mov  r2, lr
    mov  r3, lr
    bxns lr
```

Store callee saved registers

Sanitize secure state registers

Return to non-secure state

In secure memory

Memory regions labeled (secure, non-secure, NSC) during device initialization
Automatic transition to secure state on entering NSC, limited to SG instruction

ARM Ltd., “TrustZone technology for the ARMv8-M architecture”, Whitepaper 2016
Nyman et al “CFI CaRE”, RAID 2017

TrustZone-A vs. TrustZone-M

- Secure state transition via SMC
- Single entry point (Monitor)
- Kernel & monitor save/restore registers
- Monitor reconfigures MMU on entry/exit
- Context switch costs thousands of instructions

- Automatic transition on entering NSC
- Multiple entry points (SG veneers)
- Secure functions save/restore registers
- No MMU in embedded devices
- Context switch costs a few instructions

TEE specifications:
https://www.globalplatform.org/specificationsdevice.asp

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

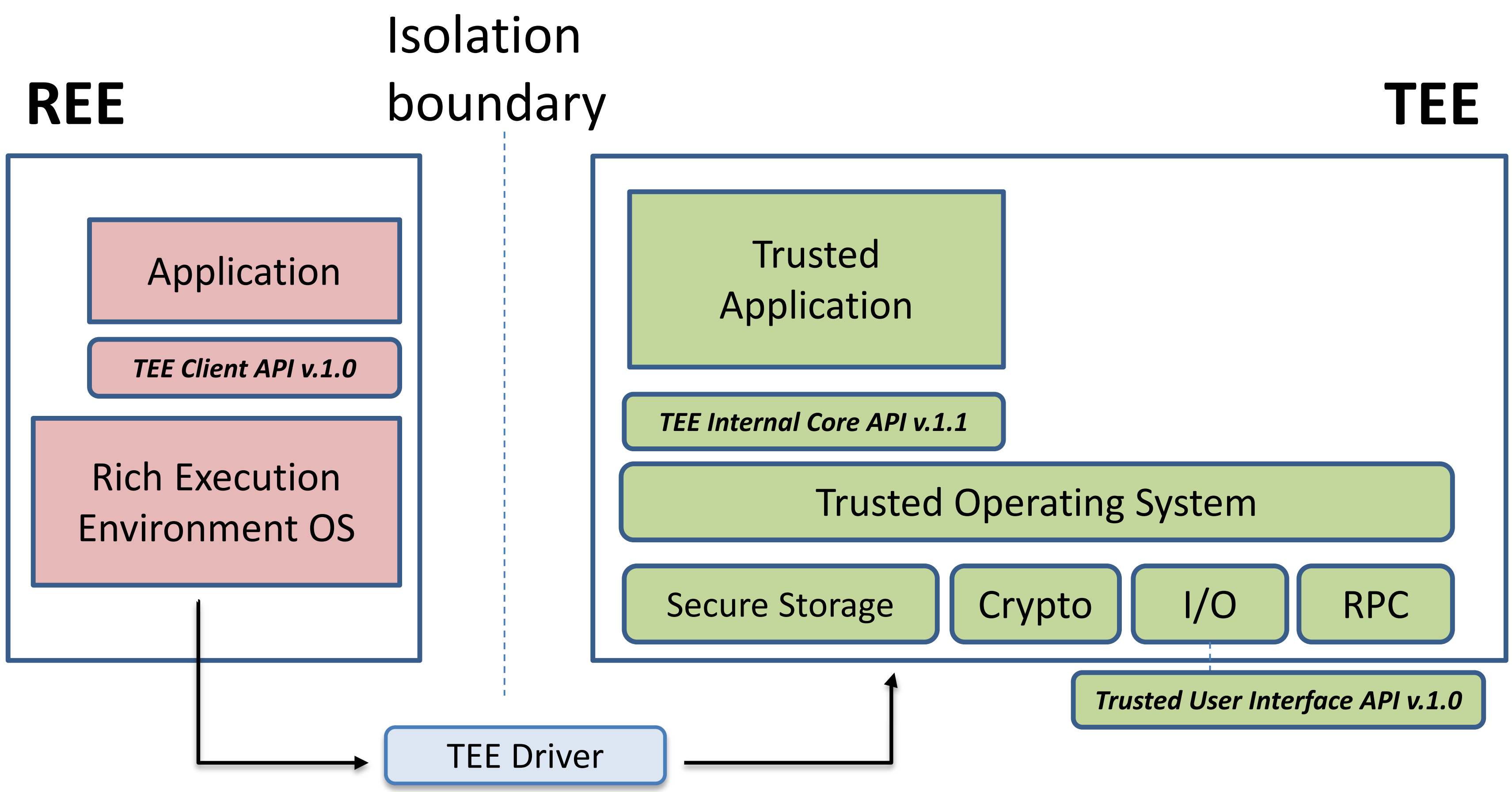
Recently GP has released standards for mobile TEEs

- Architecture and interfaces

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

- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal Core API Specification v1.1
- Trusted User Interface API v 1.0

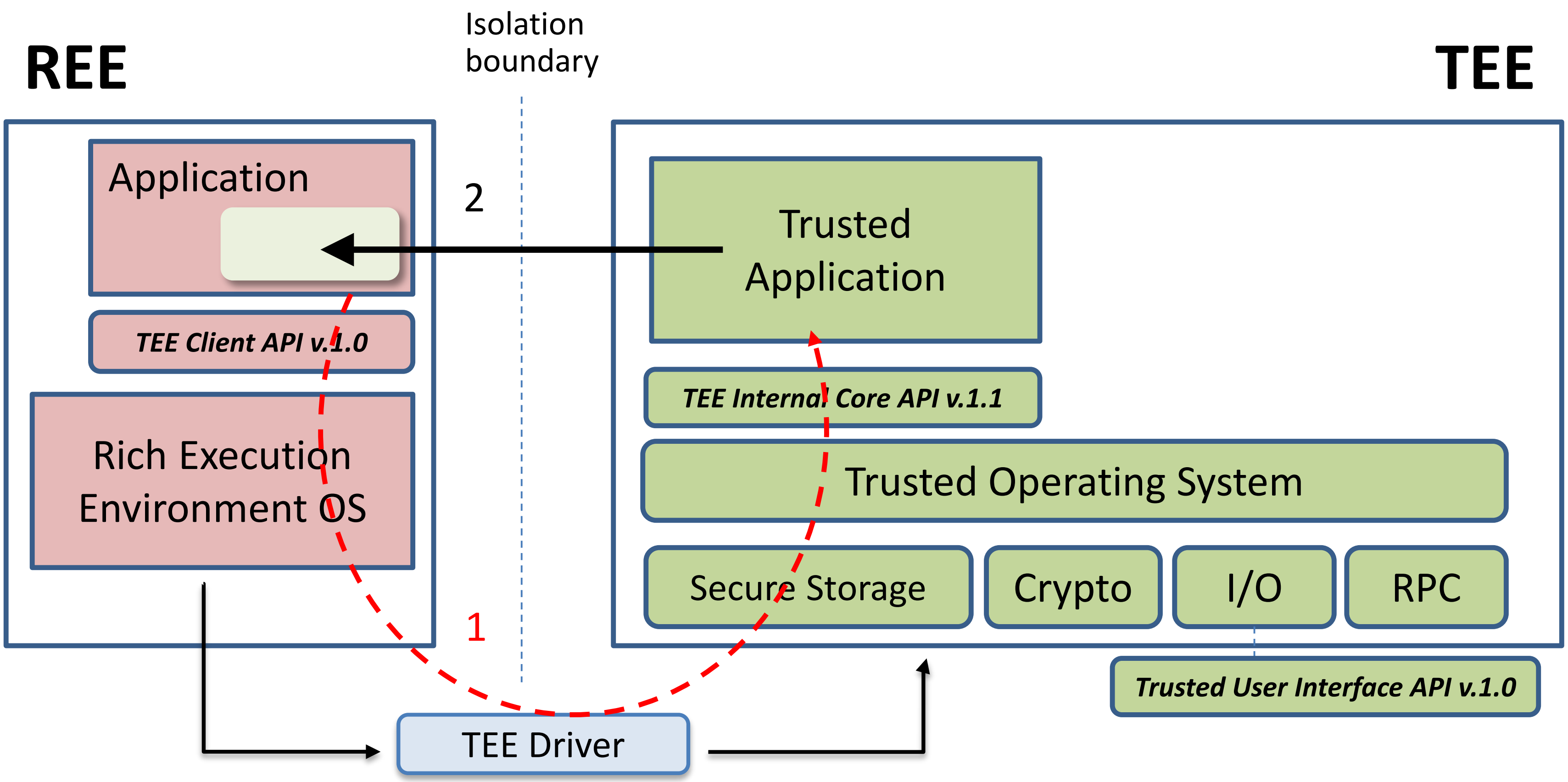
GP TEE System Architecture



Interaction with Trusted Application

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

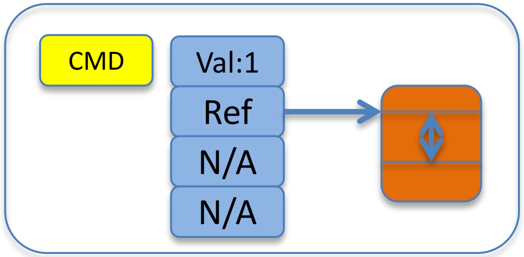
- Example: Efficient in place encryption



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 Core 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 Core 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

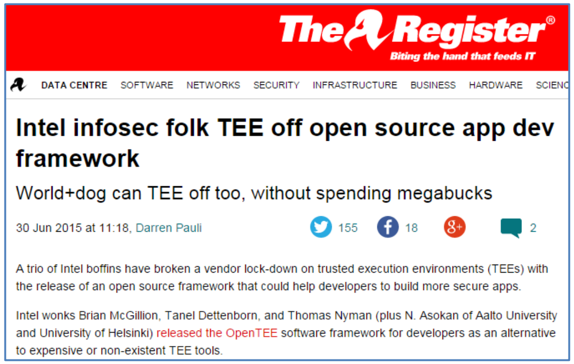
```
TEE_OpenTASession(TEE_UUID* destination, …,  paramTypes, params[4], &session);
TEE_InvokeTACommand(session, …, commandId, paramTypes, params[4]);
```

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

Legend:
OEM: Original Equipment Manufacturer

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
  - Original intent: virtual TEE platform for TA developers
    - Implements GP interfaces: TA development w/ standard Linux tooling
  - Port for Android (requested by an OEM)
  - https://github.com/Open-TEE


The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY INFRASTRUCTURE BUSINESS HARDWARE SCIENC

Intel infosec folk TEE off open source app dev framework

World+dog can TEE off too, without spending megabucks

30 Jun 2015 at 11:18, Darren Pauli 155 18 2

A trio of Intel boffins have broken a vendor lock-down on trusted execution environments (TEEs) with the release of an open source framework that could help developers to build more secure apps.

Intel wonks Brian McGillion, Tanel Dettenborn, and Thomas Nyman (plus N. Asokan of Aalto University and University of Helsinki) released the OpenTEE software framework for developers as an alternative to expensive or non-existent TEE tools.

http://www.theregister.co.uk/2015/06/30/opentee_an_open_virtual_trusted_execution_environment/

TEE standards and specifications

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

TEE environment

TrustZone Security Foundation by ARM

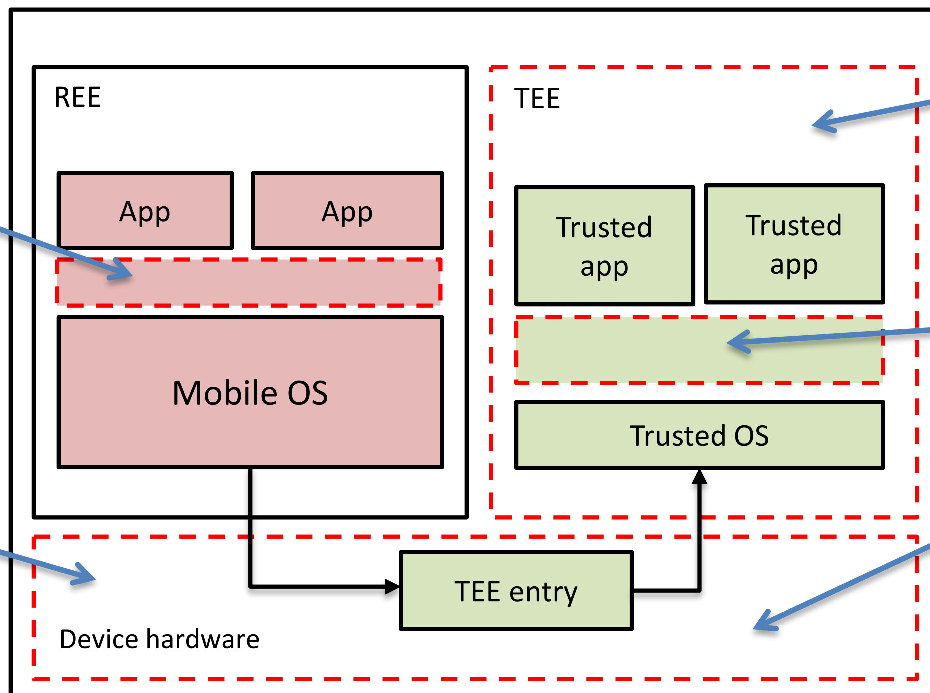
Intel SGX

TRUSTED COMPUTING GROUP

REE app API

TRUSTED COMPUTING GROUP

GLOBALPLATFORM



TEE app API

GLOBALPLATFORM

Secure Boot

TRUSTED COMPUTING GROUP

NIST

Hardware trust roots

Standards summary

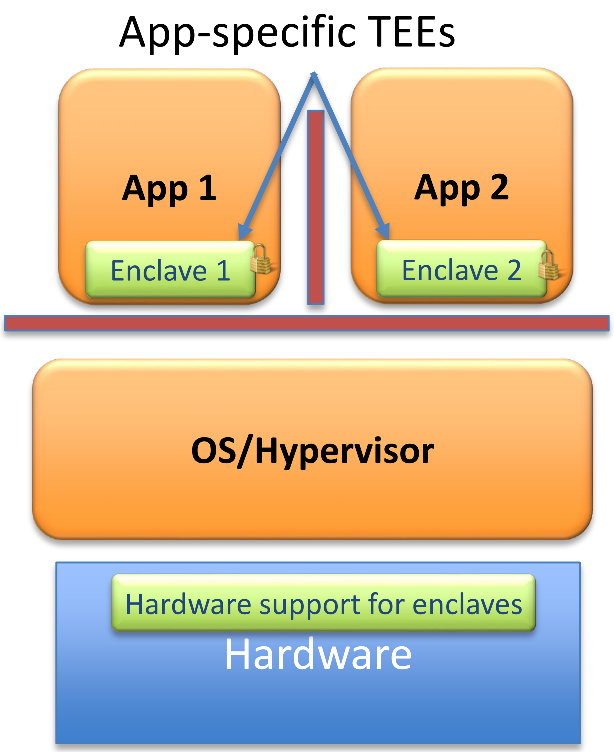
- Global Platform Mobile TEE specifications
  - Sufficient foundation to build trusted apps for mobile devices
- TPM 2.0 library specification
  - TEE interface for various devices (also Mobile Architecture)
  - Extended Authorization model is (too?) powerful and expressive
  - Short tutorial on TPM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

TEE instances

INTEL SOFTWARE GUARD EXTENSIONS (SGX)

Skip to end

Intel Software Guard Extensions

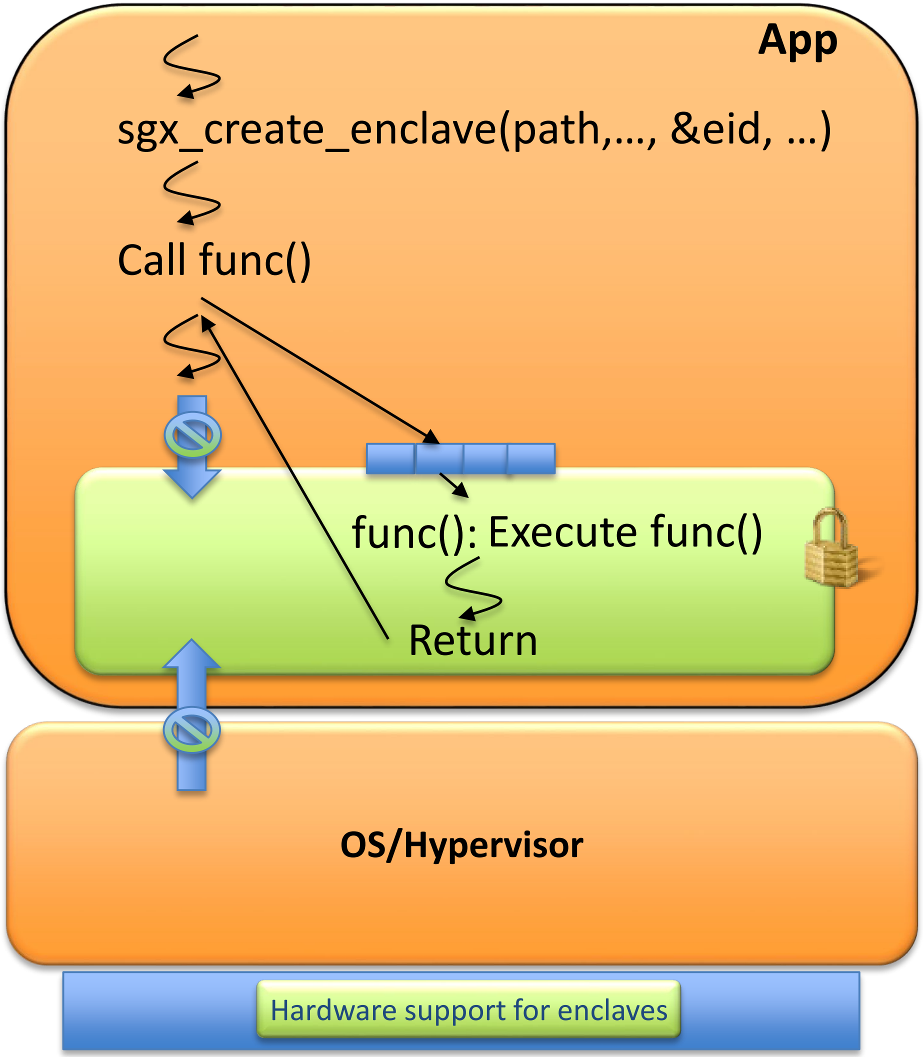


- HW-supported TEE functionality in ring-3
- Enclave code/data encrypted by HW
- Supports attestation and sealing

Intel Software Guard Extensions :
"Theory of Operations": https://software.intel.com/en-us/sgx/resource-library
Academic papers: https://software.intel.com/en-us/sgx/academic-research

Skip to SGX attestation

How does SGX work?



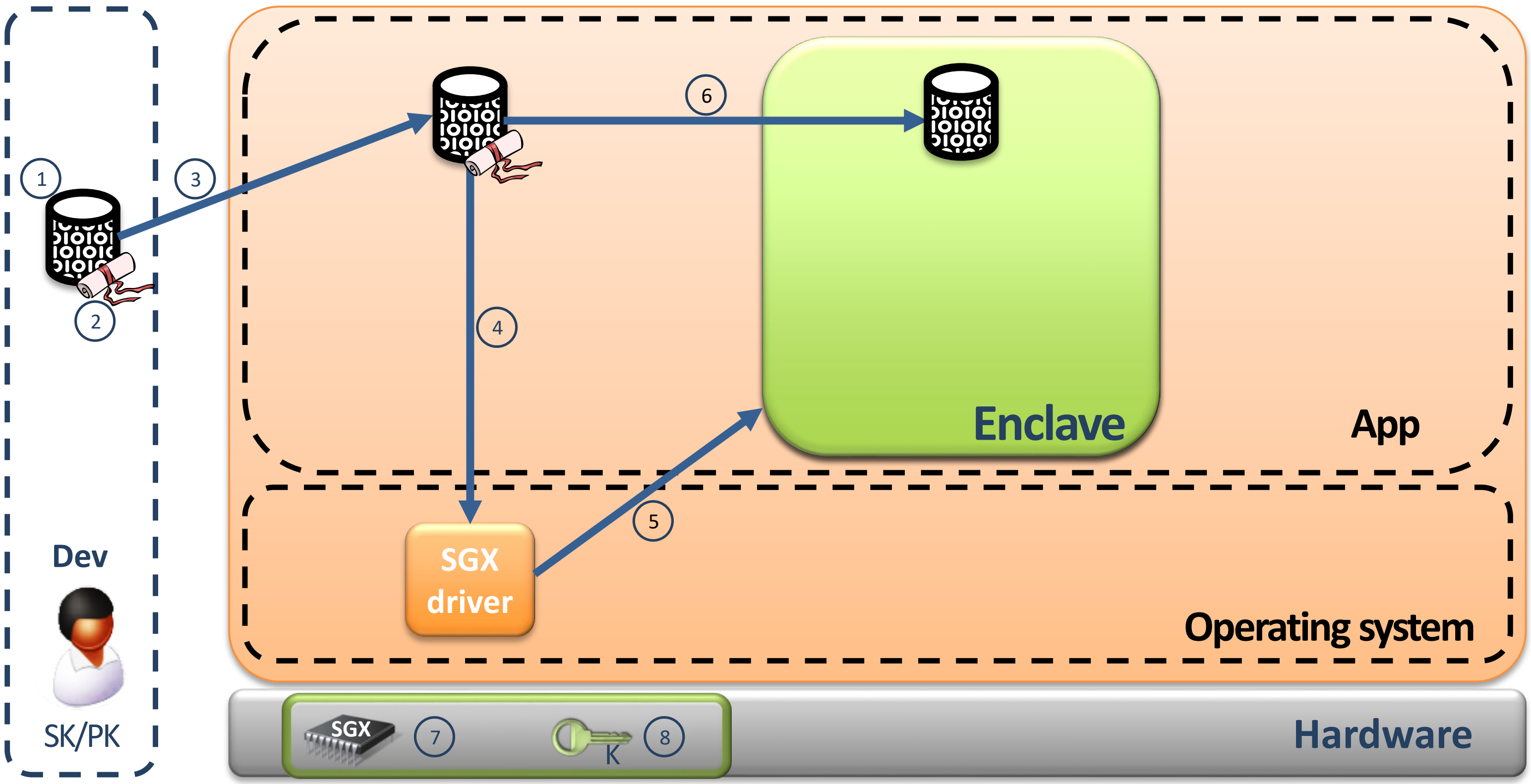
ECREATE, EADD, EEXTEND, EINIT

EENTER

EEXIT

Skip to SGX attestation

SGX – Create Enclave

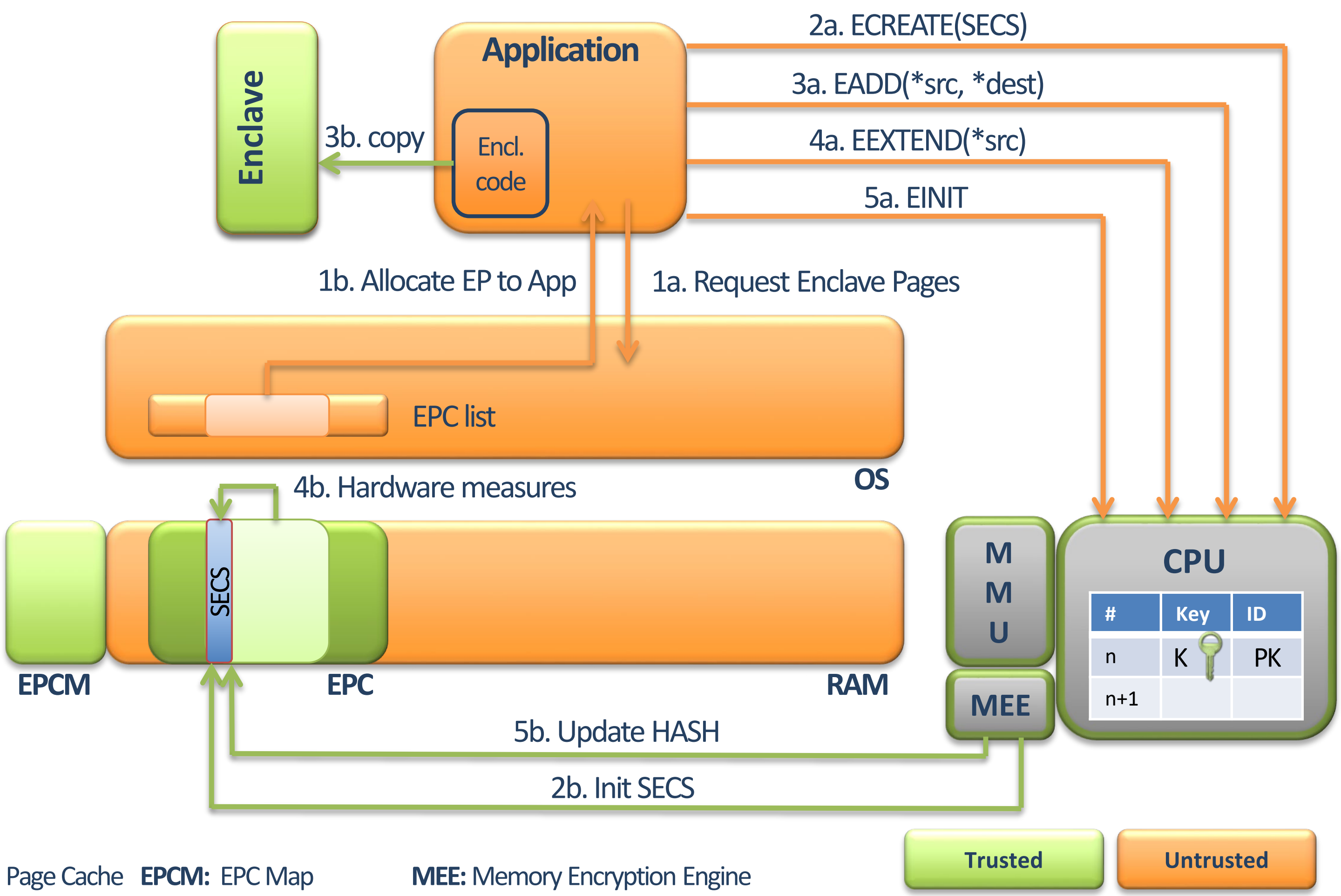


1. Create App 2. Create app certificate (includes HASH(App) and Dev PK) 3. Upload App
4. Create enclave 5. Allocate enclave pages 6. Load & measure enclave 7. Validate certificate and enclave integrity
8. Generate enclave **K** key 9. Protect enclave

Trusted Untrusted

Skip to SGX attestation

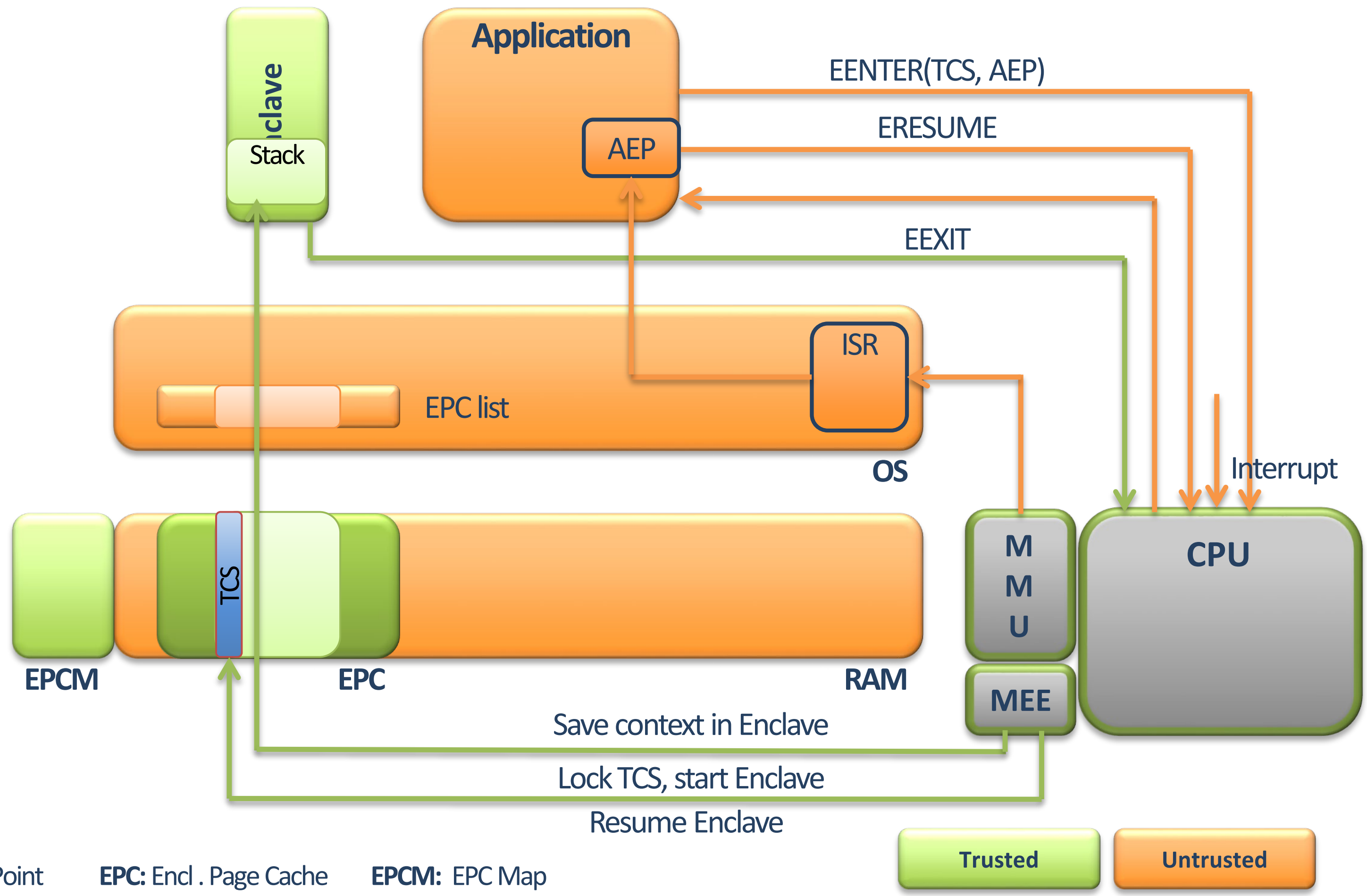
Enclave Creation – Details



EPC: Enclave Page Cache **EPCM:** EPC Map
MMU: Memory Management Unit
MEE: Memory Encryption Engine
SECS: SGX Enclave Control Structure

Skip to SGX attestation

Enclave Entry and Exit – Details



AEP: Async Exit Point
ISR: Int. Service Routine
EPC: Encl . Page Cache
MEE: Mem. Enc. Engine
EPCM: EPC Map
TCS: Thread Control Structure

Attestation in SGX

- Local Attestation: one enclave verifies another on the same device

- Remote Attestation: a remote party verifies an enclave

Enclave Identity

Identity of an enclave:

- Enclave’s **initial state**
- **sealing identity**

Initial State

- *Enclave measurement* representing:
  - Contents of enclave pages (initial code/data)
  - Relative position of enclave’s pages
- Determined during enclave creation:
  - Log activities during enclave creation
  - Digest of log contents in *MRENCLAVE*
  - Only CPU can modify the MRENCLAVE

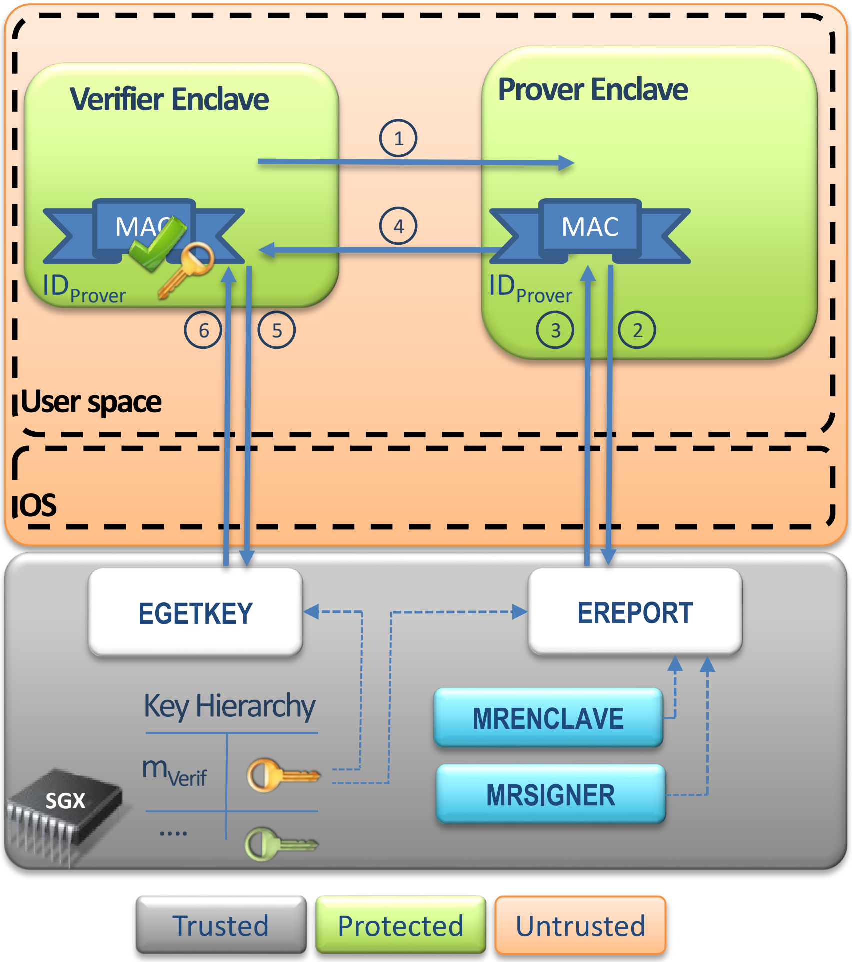
Sealing Identity

- *Sealing authority (SA)* signs enclaves prior to distribution:
  - Signature on trusted (expected) value of initial state
  - Signature and SA's public key sent to devices that need to run the enclave
- During enclave creation on device:
  - signed measurement
    - verified using SA's public key
    - compared with local measurement
    - If matched, sealing identity (hash of the SA's public key) stored in the MRSIGNER register

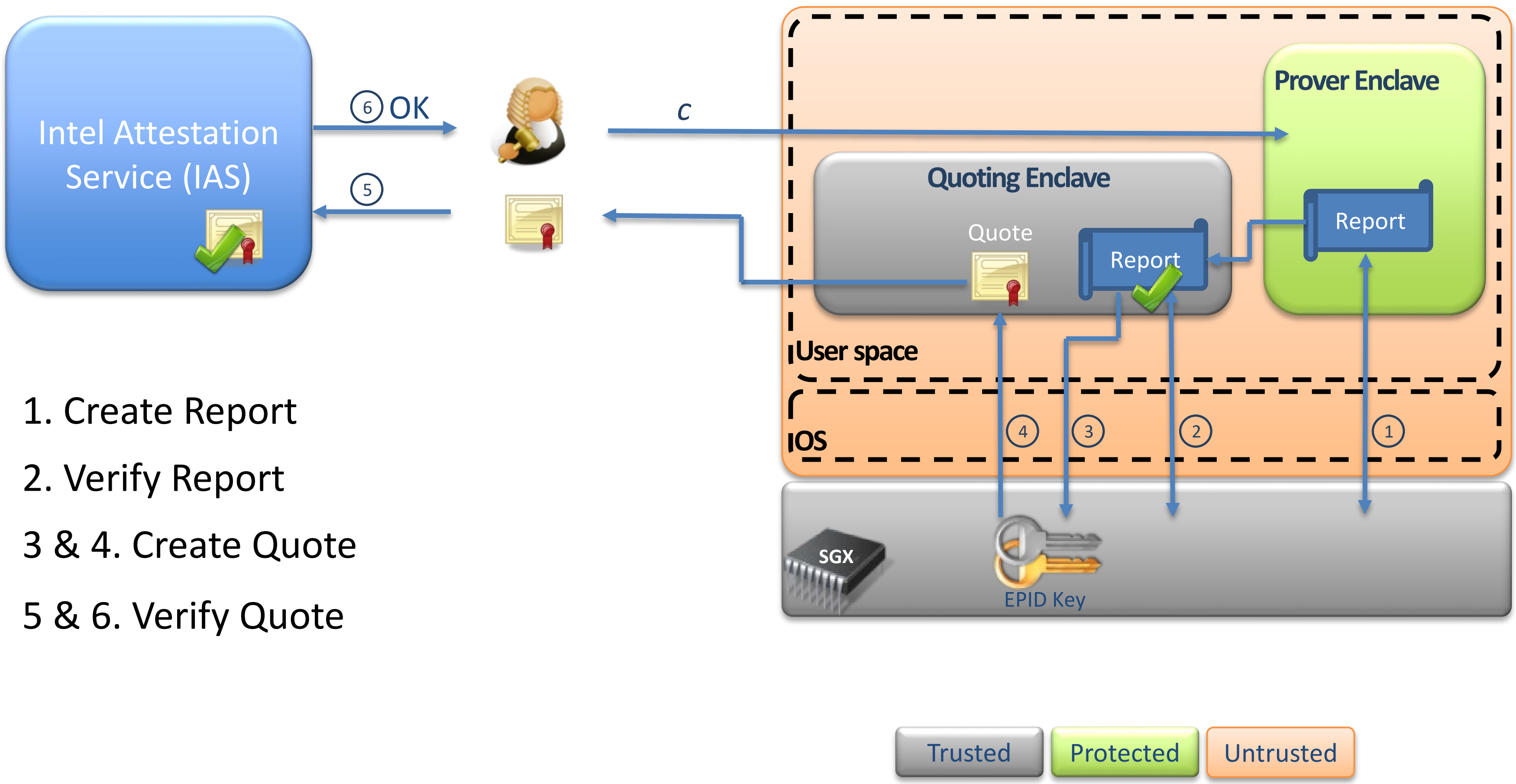
Local Attestation

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

   Report := ID_{Prover}, $MAC(ID_{Prover})_{RepKey_{Verifier}}$
4. Report transferred to verifier
5. Verifier calls *EGETKEY* (for reports)
6. Verifier's *report key* is returned
7. MAC included in Report verified using received *report key*



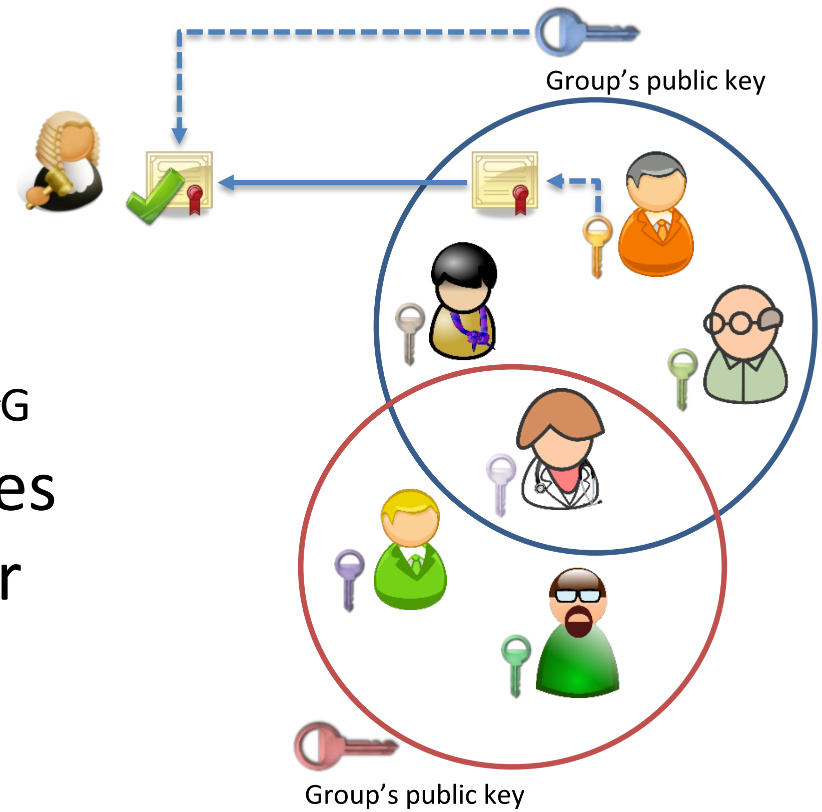
Remote Attestation



1. Create Report
2. Verify Report
3 & 4. Create Quote
5 & 6. Verify Quote

Intel Enhanced Privacy ID (EPID)

- Group signature scheme
- Each signer
  - owns a secret key
  - belongs to a group
- Group has a public key PK_G
- Use PK_G to verify signatures generated by any member



Sealing

- Store persistent data securely
- Enclaves get sealing keys via EGETKEY
- Two modes:
  - Sealing to Enclave-Identity
    - key derived from contents of MRENCLAVE
  - Sealing to Sealing-Identity
    - key derived from contents MRSIGNER

Did you learn:

- What are example instances of hardware platform security?
  - Fixed function TEEs: Trusted Platform Module (TPM)
  - Programmable TEEs:
    - ARM TrustZone
    - Intel Software Guard Extensions (SGX)
  - Standardized interfaces for using TEEs

Contributors: Jan-Erik Ekberg, Kari Kostiainen, Thomas Nyman, N. Asokan, Ferdinand Brasser, Ahmad-Reza Sadeghi

Plan for the course

- Lecture 1: Platform security basics
- Lecture 2: Case study – Android OS Platform Security
- Lecture 3: Mobile platform security
- Lecture 4: Hardware security enablers
- Lecture 5: Usability of platform security
- Lecture 6: Summary and outlook
- Lecture 7: SE Android policies
- Lecture 8: Machine learning and security
- Lecture 8: IoT Security