

# TRUSTED COMPUTING

---

# Overview

- Why trusted computing ?
- Intuitive model of trusted computing
- Hardware versus software
- Root-of-trust concept
- Secure boot
- Trusted Platforms using hardware features
  - Description of TCG based trusted platforms; TPM working A & B
  - TPM use in UEFI
  - TPM use in Intel TXT
  - Intel TXT in Trusted Computing Pools (OpenStack)
  - ARM TrustZone
  - Intel SGX
- Software secured execution environment:
  - Java, STIP, .NET
  - OSes and Virtualization: Linux LSM, SELinux
- Special trusted computing devices

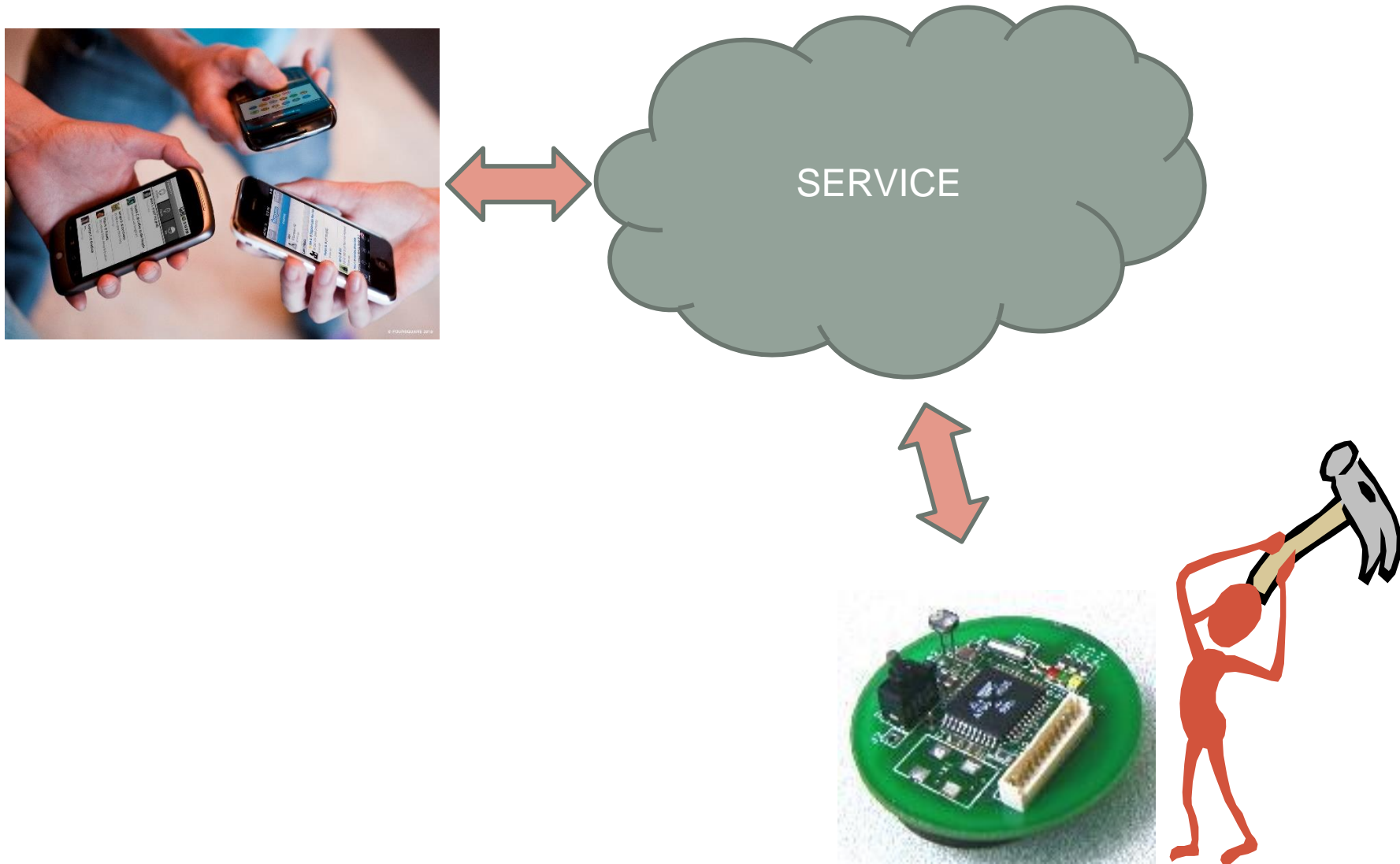
# TRUSTED COMPUTING

---

## PART 1

- Why trusted computing ?
- Intuitive model of trusted computing
- Hardware versus software
- Root-of-trust concept
- Secure boot

# Why trusted computing?



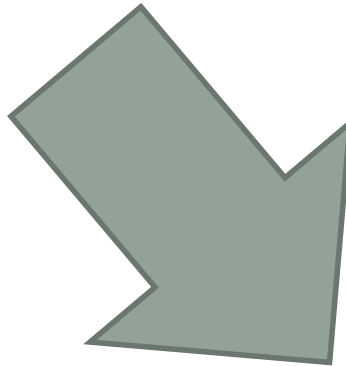
# Why trusted computing? new security challenges

- Computing devices are becoming **distributed**, **unsupervised**, and **physically exposed**
  - Computers on the Internet (with untrusted owners)
  - Cloud computing data centers
  - Embedded devices (cars, home appliances)
  - Mobile devices (cell phones, PDAs, laptops)
  - Smart sensor devices
- Attackers may **physically tamper** with devices
  - Invasive probing
  - Non-invasive measurement
  - Install malicious software

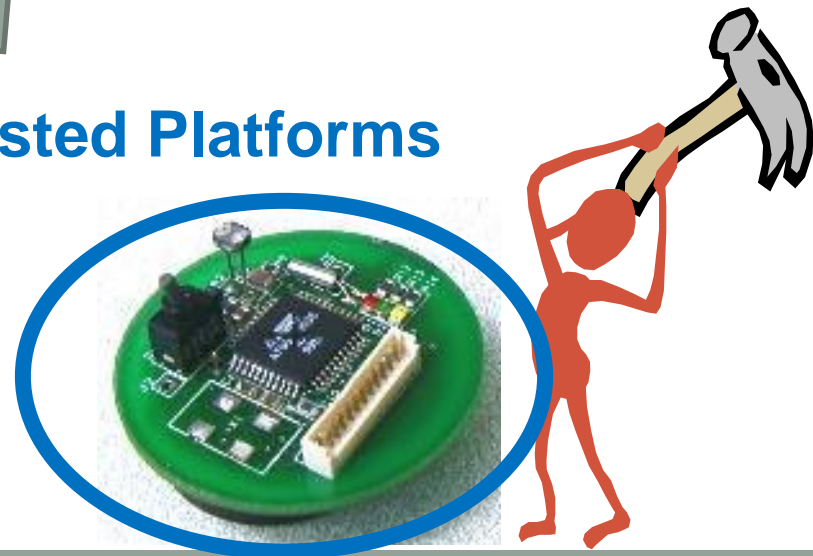


# Why trusted computing? new security challenges

- How to achieve that we can trust the devices?



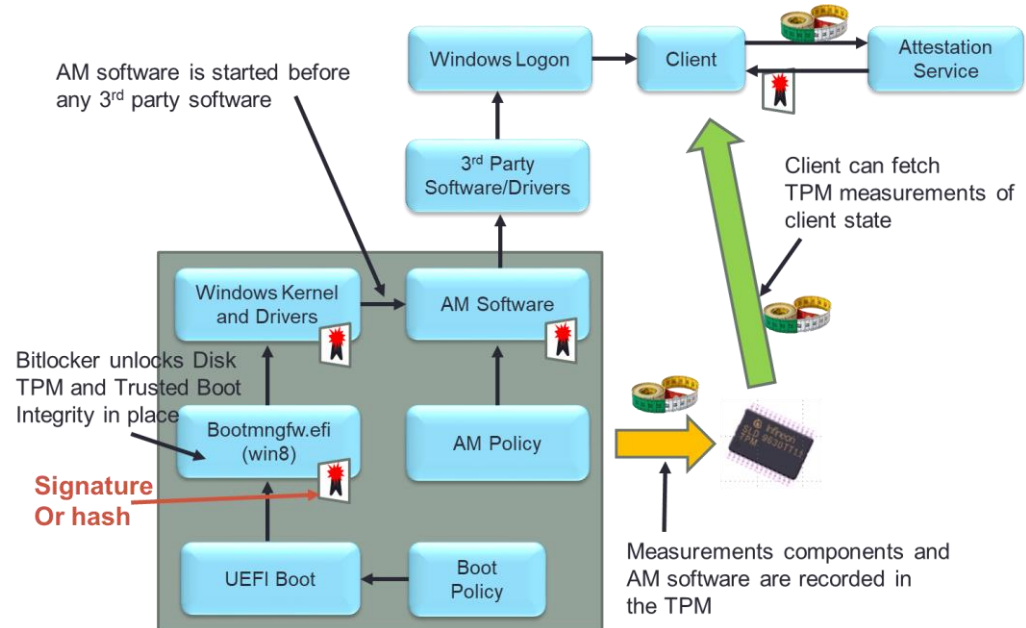
## Trusted Computing & Trusted Platforms



# Example: UEFI Trusted Boot Architecture

Many PC/laptop and Server system support secure/trusted boot as a way to secure that no unauthorized sw can start to run on the system.

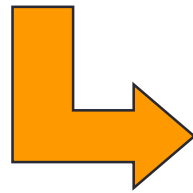
We explain how this is though to work!



# From trusted computing to trusted platform

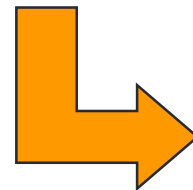
To achieve **Trusted Computing** one

1. Needs that the (application) software can be trusted



**SW security:** Not the subject of this lecture

2. Needs that the underlying system can be trusted



**Trusted Platforms**  
(or better Trustworthy Platforms)



# SW security: Coding practises

- Good coding practices are necessary to avoid security compromises due to programming errors.
- This will be an issue we return to later in this course


# Intuitive models cont'd

Trusted computing combines best properties of

- Open:
  - allow applications from many different sources to run on same platform
- Closed:
  - Only "known" software can execute
  - remote parties can determine what software is running and whether to expect the platform to be well behaved
    - Done through a procedure called attestation (we come back to this)

# What do we mean by trust in a system ?

- In words: we have an understanding of how the system should behave and have confidence that it does so.

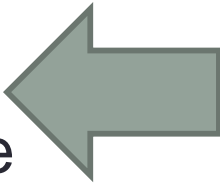
- "the system"
  - "should behave"
  - "confidence"
- 
- Is the system "trustworthy" ?

# Trusted vs Trustworthy

What to use ?

**Trusted:** A system can be trusted but is it trustworthy?

**Trustworthy:** The system can fulfill the requirements defined by a methodology. Is the methodology then trustworthy ( and we get a recursion) or we just trust the methodology.

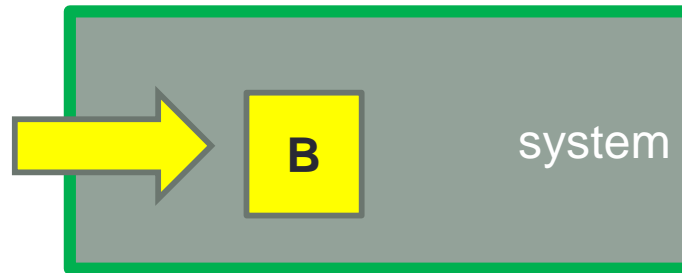


Recall: Using Common Criteria a system that is successfully evaluated at certain EALx level is trustworthy.

# A system is trusted because

- Its behaviour B is "known and immutable"

Behaviour we trust it to have e.g. WE put it in there

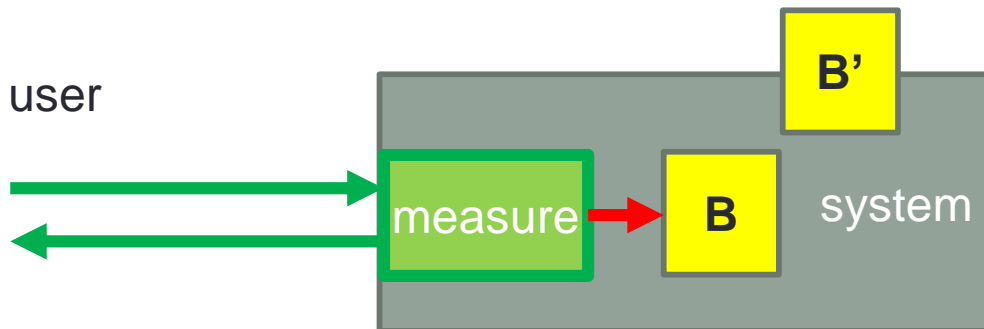


System defends itself

- Its behavior is "under control"

System user

checks



System measures its behaviour

Think of behaviour as the code/program

# Hardware vs Software

- Functionality in Hardware

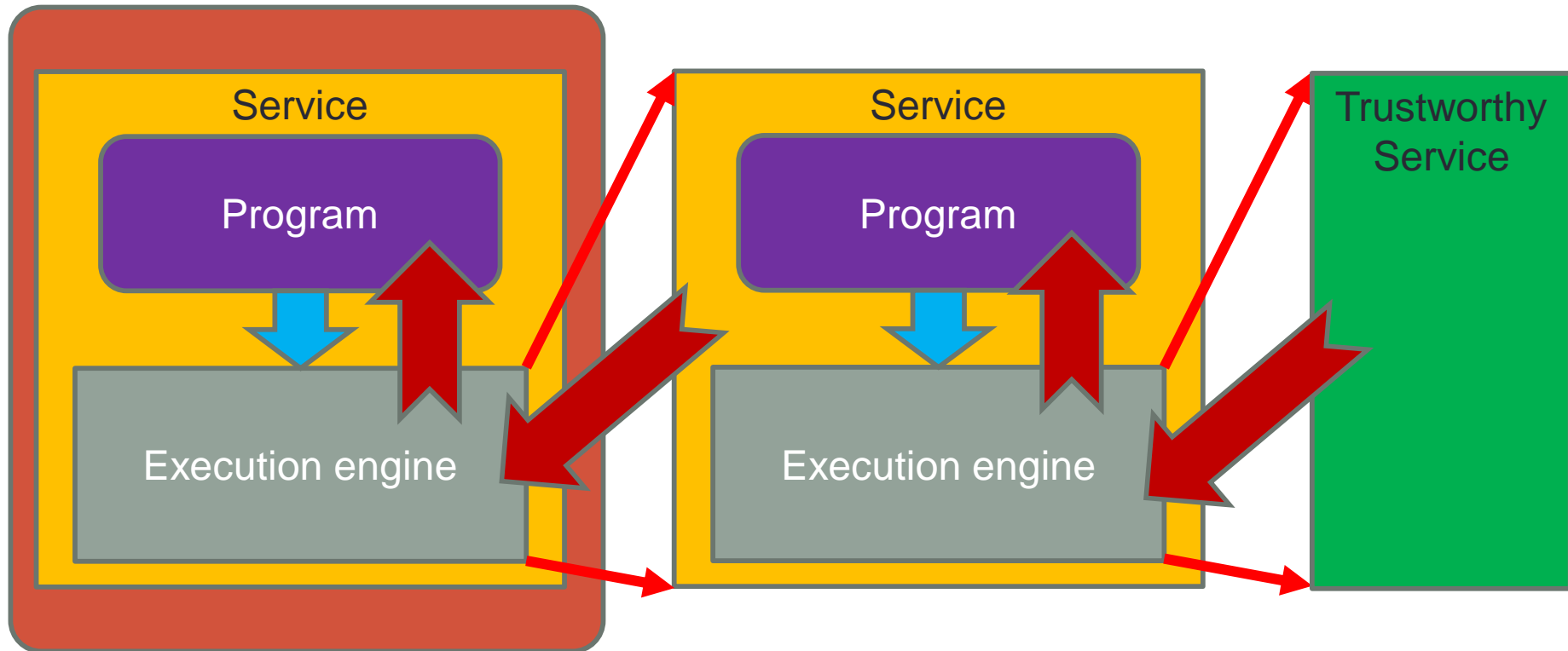
- hard to change
- high performance possible
- Better protection against
  - Extraction of data
  - Modification of data
- Hard to prove correctness

- Functionality in Software

- Easy to change
- Difficult to hold private keys
- Can formally verify program is correct (if it is not too complex)

# Trusted Platform – start of trust chain – root of trust

Root-of-trust (RoT)



**Recurssion must stop at a service we trust/have to trust,  
e.g. Intel HW.**

# Root of Trust

- So the RoT (Root of Trust) is that part of the system we consider trustworthy.
- Why can it be trustworthy?
  - We did a very careful analysis of the design and implementation
    - Recall Common Criteria (from Computer Security course)
  - We can verify that someone else (a third party) considers the part of the system trustworthy



# The different Roots of Trust

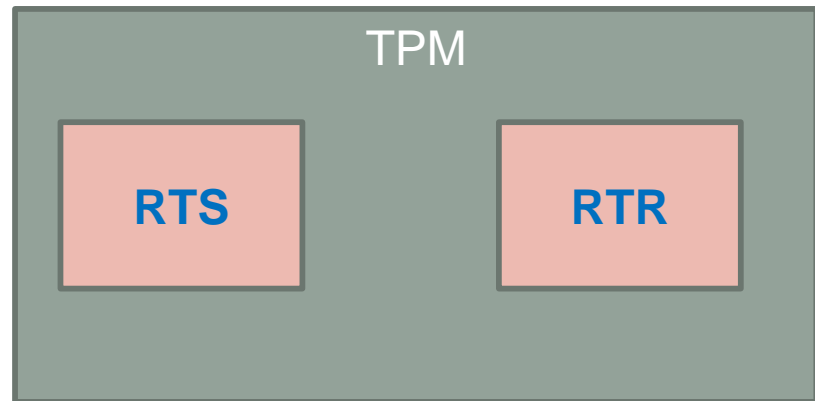
- It is useful to think a RoT to consist of different RoTs each with a special task
- The RTS (RoT for Storage):
  - A compute engine that protects use and access to data/keys
- The RTM (RoT for Measurement):
  - A computing engine capable of making reliable integrity measurements.
- The RTR (RoT for Reporting):
  - A computing engine capable of reliably reporting information held by the RTS

# Industry specifications/requirements on RoTs

- Trusted Computing Group (TCG)
  - GlobalPlatform
  - NIST
- 
- We will in the remainder look at the TCG (and NIST) approach.

# TPM

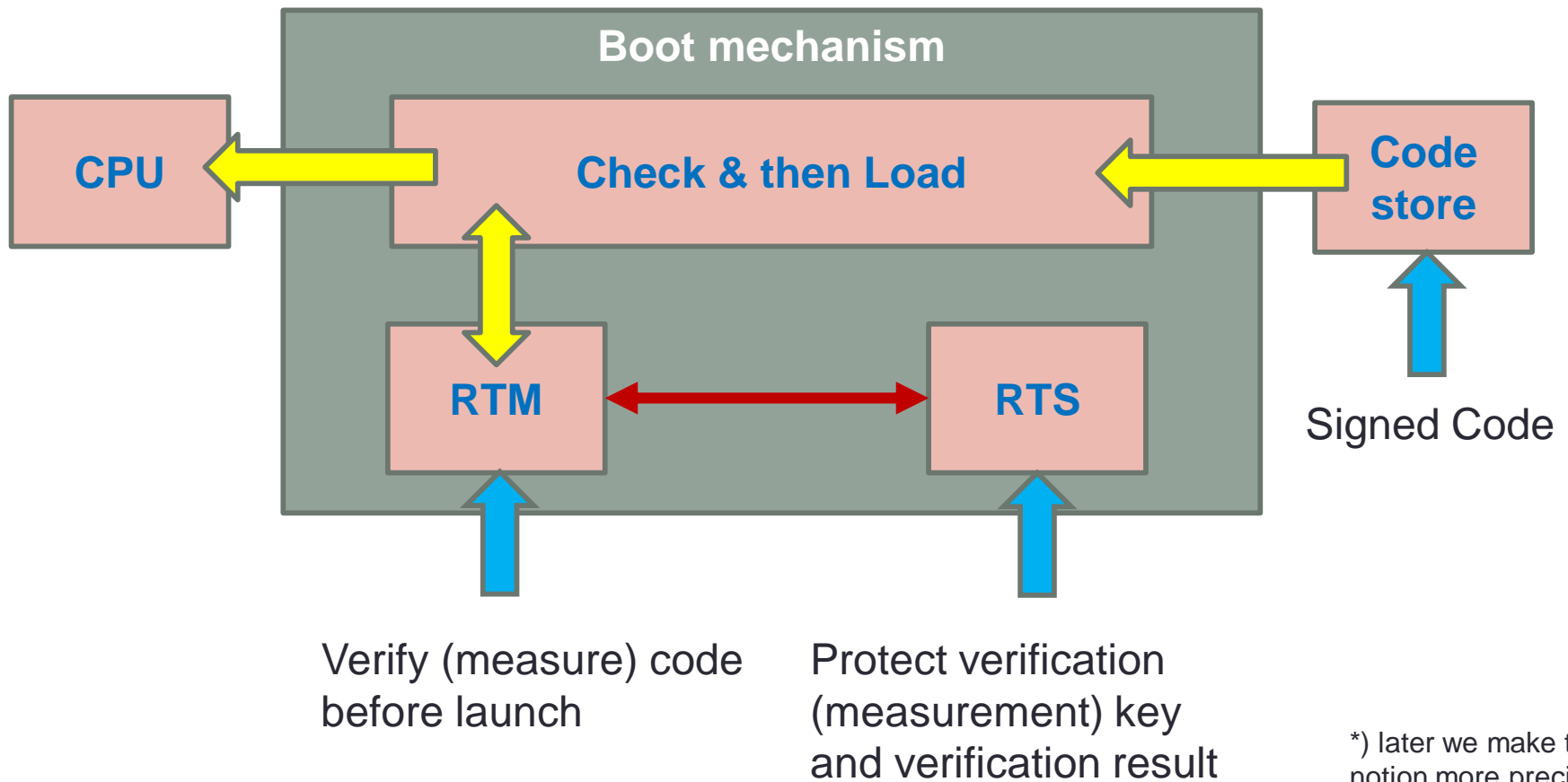
- The TCG has specified the Trusted Platform Module (TPM) as the secure device realizing an RTS and RTR



The TPM is a passive device and thus responds only on commands and thus must be driven by something (e.g. a process from the outside.)

# Use of RoT: Secure boot

- Using an RoT we can implement a secure<sup>\*)</sup> boot.



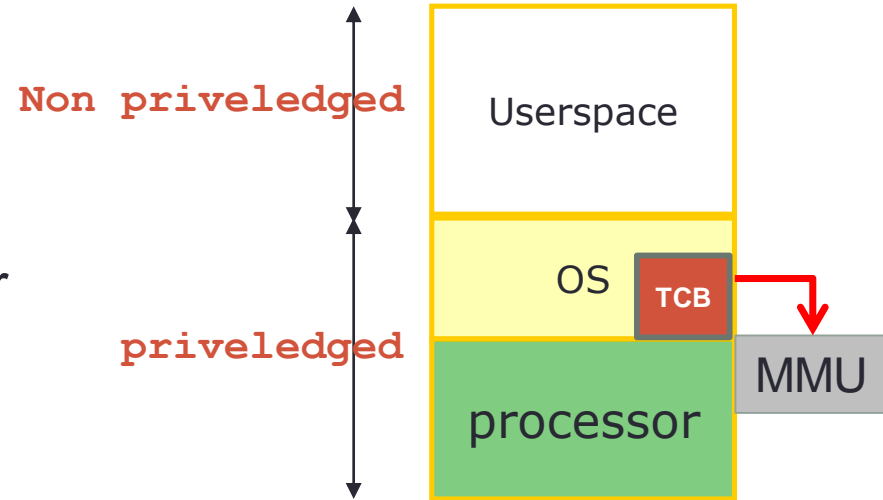
# RoT in Software

- Possible but we have limitations
  - owner of the device on which software runs should not be an attacker (he/she and the device "work together"/"have the same interests")
  - Does not work when the device is in the "enemy's territory"

We return to sw based trusted platforms later. We recall the notion of TCB. In an ordinary OS the TCB is the heart of the security in the platform

# Trusted Computing Base (TCB)

- In stead of having to trust the complete OS we organize the firmware & OS that our trust depends only on a (much) smaller part called the TCB
- The TCB consists of the mechanisms (from hardware, firmware, and software) that together are responsible for enforcing a computer security policy.
- Note that this indicates that the TCB has RTM capabilities.



# TRUSTED PLATFORMS

---

## PART 2

- Trusted Platforms using hardware features
  - Description of TCG based trusted platforms; TPM working A & B
  - TPM use in UEFI
  - TPM use in Intel TXT
  - Intel TXT in Trusted Computing Pools (OpenStack)
  - ARM TrustZone
  - Intel SGX



- Founded in 1999 by Compaq, HP, IBM, Intel and Microsoft
- Currently more than 200 members
- Issues industry specifications on trusted computing technologies for
  - PCs, tablets, servers
  - Storage devices
  - Virtualization

Note these specifications are not official standards.



# TPM WORKING

---

## Part A:

- TPMs place in a PC/server
- TPMs design and internal component

# TCG goal and impact on HW

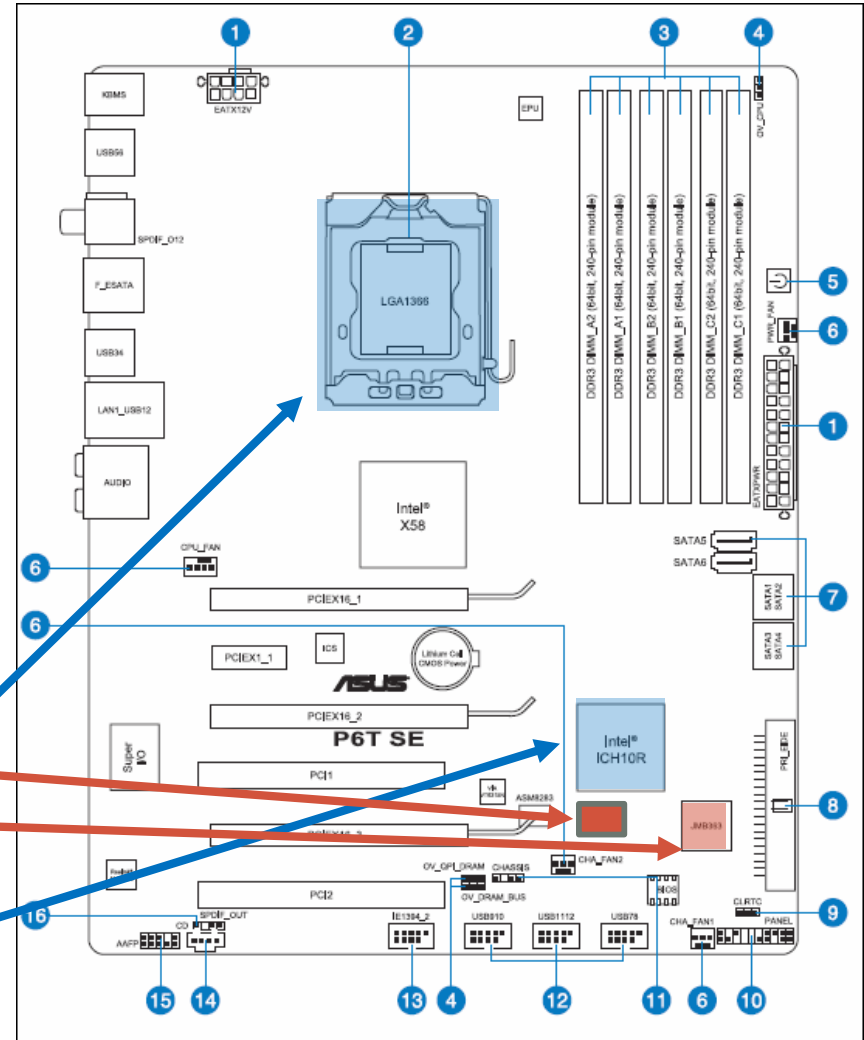
- On a high level TCG wants to foster technology that promotes and defines and promote hardware-based root of trust, **an RoT**.
- How does this affect the HW then??

**At least:**

- We add TPM chip**
- We modify BIOS**

**Later also**

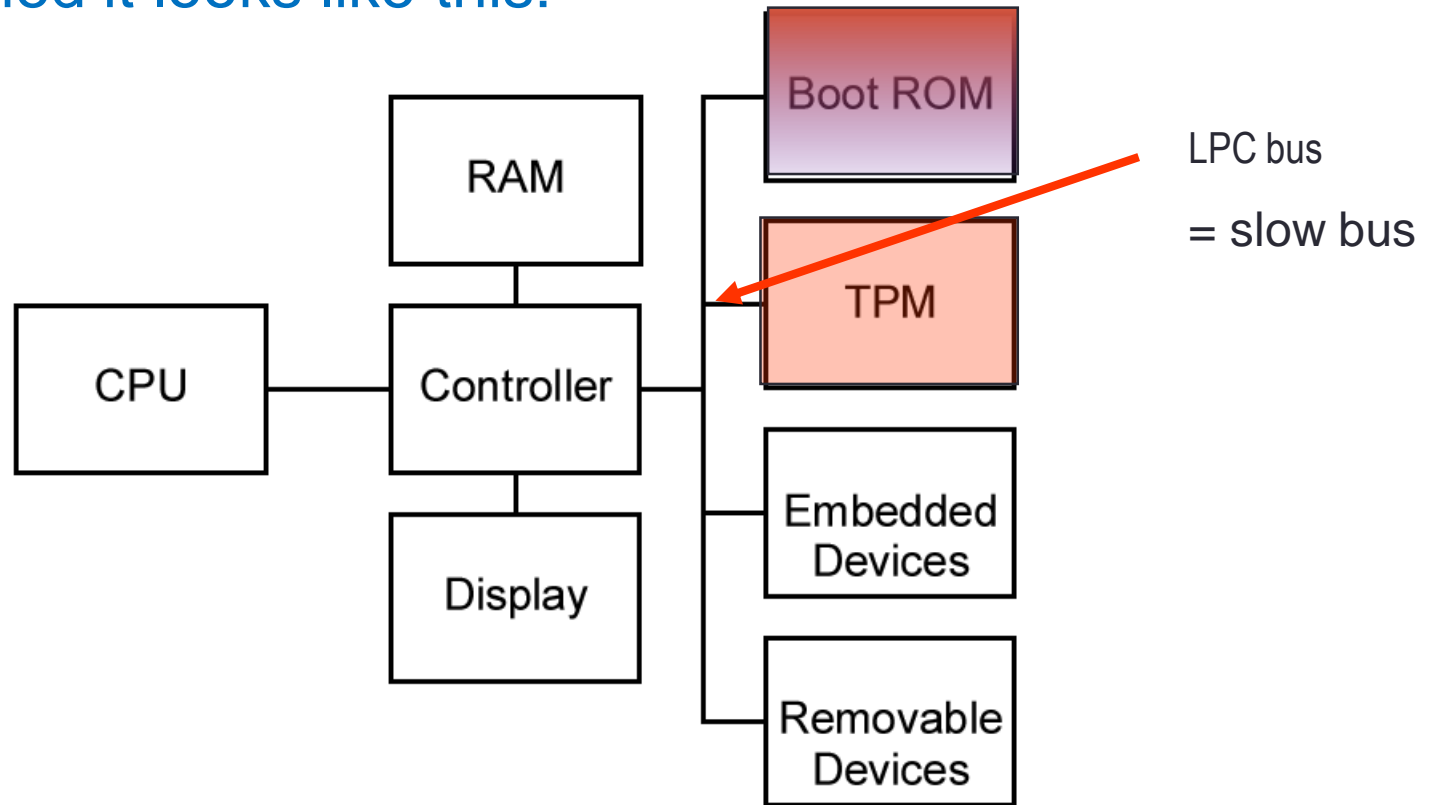
- Modify CPU**
- Southbridge**



TPM = Trusted Platform Module

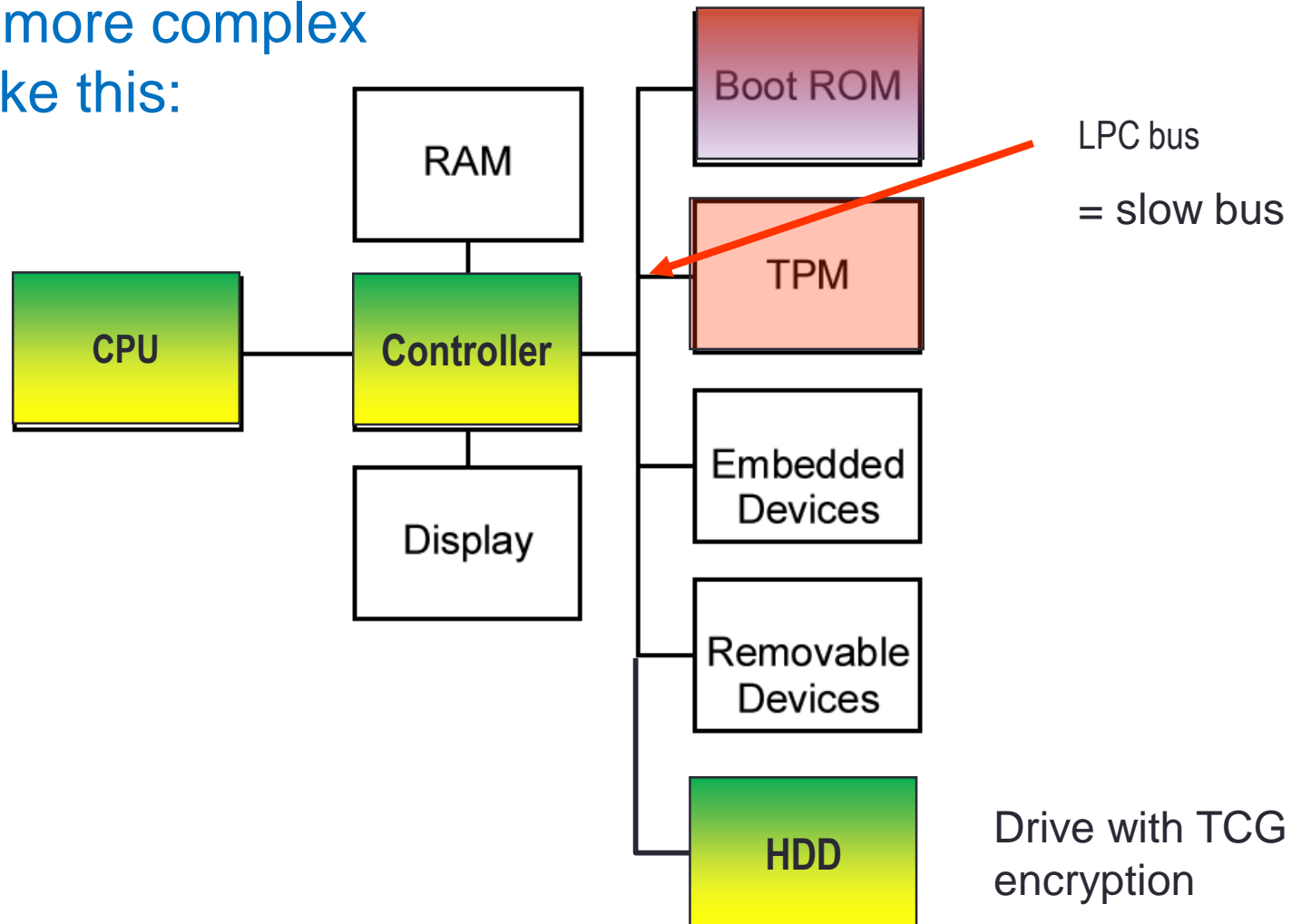
# TCG Architecture (typically PC or Server)

So simplified it looks like this:



# TCG Architecture (typically PC or Server)

And in a more complex system like this:



# Functions the ROT should provide

- ***Protected Capabilities***

Protected capabilities is a set of commands that grant the user issuing the command access to protected locations, memory (storage), registers, etc.

- ***Attestation***

Attestation is the process of verifying the accuracy of information and the characteristics of the TPM chip current state.

- ***Integrity (Measurement and Reporting )***

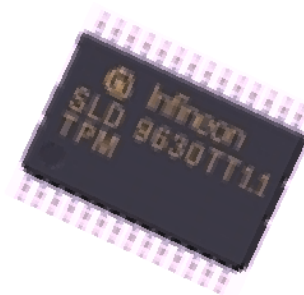
Integrity measurement is the process of obtaining metrics of the platform characteristics and storing the information digest in a protected locations (registers) in the TPM chip. Integrity reporting is to attest the integrity measurements that are recorded in these locations.

# Commercial TPM example - Infineon

## Infineon Technologies Platform Module Solution Provides the Following Features

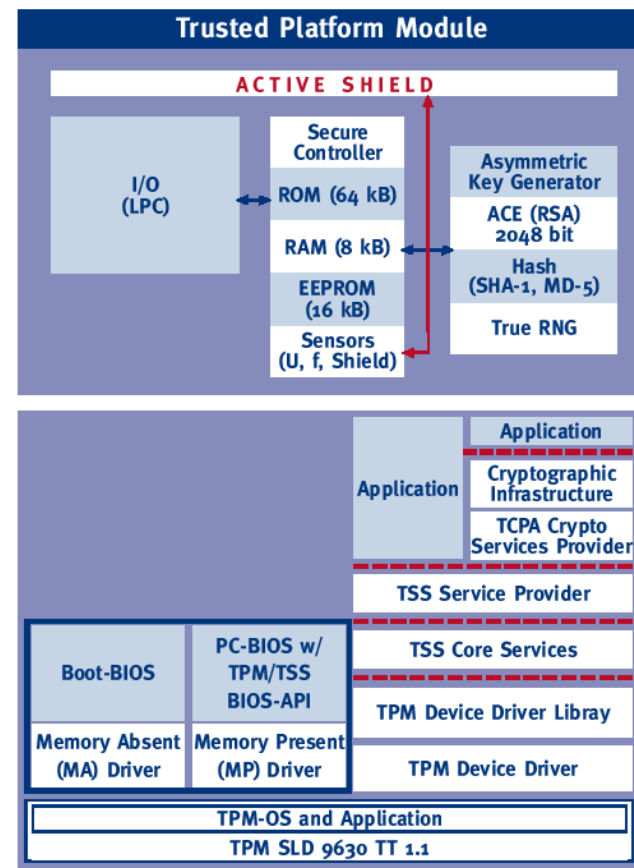
### Infineon Technologies TPM Hardware Overview:

- 64 kBytes of ROM & 8 kBytes of RAM
- 16 kBytes of EEPROM with 500 write-erase cycles
- 48 kBytes of EEPROM for firmware secure updates
- RSA hardware accelerator for signature calculation and verification as well as 2048 bit key generation when using CRT
- World-leading security protection against SPA and DPA
- Low Pin Count (LPC) bus optimized
- Low power consumption



### Software Architecture Overview:

- Embedded Secure Operating System
- Embedded Secure Application Support
- TCGA Software Stack (TSS) compliant to current and released specifications
- TCGA PC BIOS support available with design guide
- TPM cryptographic service providers for MS-CAPI 2.0 and PKCS#11



# TPM design

- The original thinking was that the TPM is a device that the owner has access too. This is not a really good assumption for servers. Why?
- It is the owner that decides to use (enable) the TPM or not and to use it. This called *opt-in* and requires the user to *take ownership* of the TPM.

Note: So there must be a GUI to do this!

- TPM should only have that part of the RoT that needs additional (HW) protection that the normal PC HW cannot provide.
- TPM impact on HW should be kept minimal and possibly work with ordinary chipset (CPU + controller combination)

# Physical Presence (PP)

- Certain operations on the TPM should only be allowed by an operator that has physical access to the device with the TPM. That is, a remote SW process cannot run such an operation. These operations are said to be allowed under **Physical Presence (PP)**.
- Examples of commands that should be implemented to function only under PP regime
  - TakeOwnership
  - ForceClear

Note that PP is a property that the system implementation must have as the TPM is never able to assert physical presence of an operator.



# TPM States

- Three operational substates  
{Enabled, Active, Owned}

E	A	O	Description
✓	✓	✓	S1: Fully Operational State
	✓	✓	S2: Disabled, Ownership is and can be set
✓		✓	S3: Enabled, Inactive but Ownership is set
		✓	S4: Disabled, Ownership cannot be set
✓	✓		S5: Enabled, Local or remote ownership set possible
	✓		S6: Disabled, Ownership can be set
✓			S7: Enabled, Inactive and no Ownership set
			S8: All functions are off

S1 Enabled – Active - Owned
S2 Disabled – Active - Owned
S3 Enabled – Inactive - Owned
S4 Disabled – Inactive - Owned
S5 Enabled – Active - Unowned
S6 Disabled – Active - Unowned
S7 Enabled – Inactive - Unowned
S8 Disabled – Inactive - Unowned

↑ BIOS normally has functions to enable or disable the TPM

# TPM Monotonic Counters

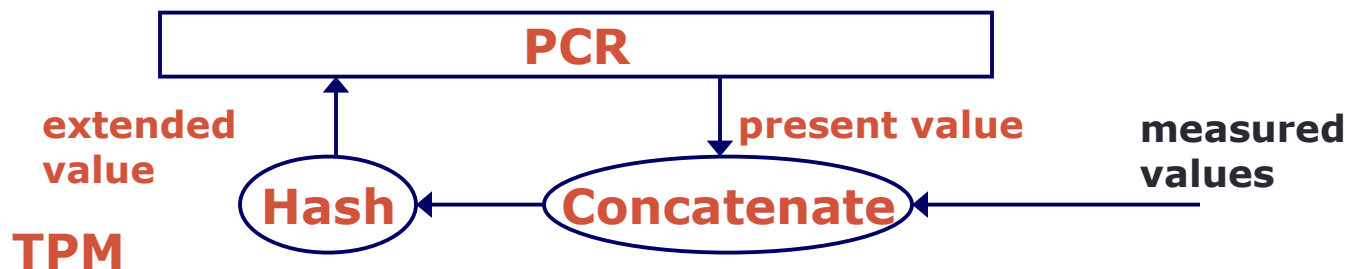
TPM has monotonic counters (at least 4)

- Increment rate: Every 5 secs for at least 7 years (so at least 26 bit counter needed).
- Note: For telco grade equipment that has to last 20 years 26 bits is not enough!
- Can be used to implement anti-roll back protection
  - (old SW version in system can be blocked from loading after new SW has been loaded once)

# TPM PCR<sub>s</sub>

- A PCR is a register that contains a SHA1 hash and is used to accumulate “measurements”
  - Accumulation of new data in the PCR is called extending a PCR
  - PCRs can be read from the outside
  - Are reset to zero at power up
    - Some PCR can be setup to be resettable when in use, be warned!
  - At least 16. In Intel TXT use of TPM there are at least 24 PCRs

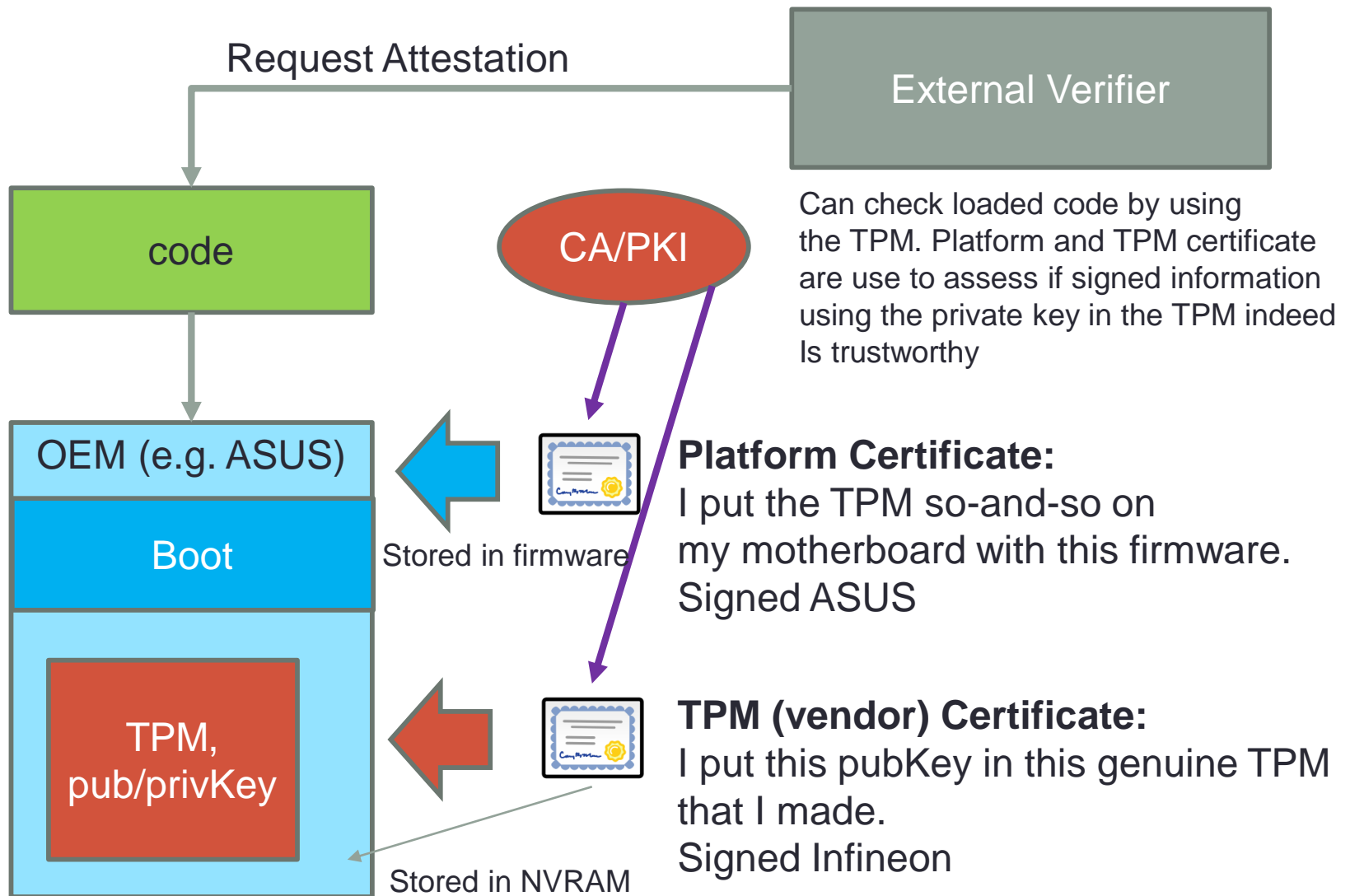
We return to this later



# Main logical components in a PC with a TPM

- **The chain of trust**
  - TPM certificate
  - Platform certificate
- **The RoT**
  - Root of Trust for Measurement (RTM)
    - The Core Root of Trust for Measurement (CRTM)
      - Static and Dynamic RTM
    - The Trusted Building Block (TBB)
  - Root of Trust for Storage (RTS)
  - Root of Trust for Reporting (RTR)
- **The TCG TPM SW Stack (TSS)**
  - make RoT functions available for applications

# Trust Chain



# TPM Certificate

(Endorsement credential)

Is actually called the Endorsement credential.

It is Digital certificate stating that EK has been properly created and embedded into a TPM Issued by the entity who generated the EK e.g., the TPM manufacturer

## It Includes

- TPM manufacturer name
- TPM model number
- TPM version
- Public EK (Note this maybe privacy sensitive data)

# The CRTM: S-RTM and D-RTM

- The Core RTM (CRTM) is the *a priori trusted code* that is referred by the platform credential.
- In the **Static RTM** Model, this MUST be the very first piece of code executed on power on or upon reset of the server or complete physical hardware environment.
  - Note: at startup the CRTM will check for physical presence of the TPM
  - REMEMBER: TPM is not the root-of-trust but trust starts with the CRTM
- In the **Dynamic RTM** model the hardware is designed to support that while running a trusted execution thread can be started:
  - Intel call their implementation (Intel) Trusted eXecution Technology
  - AMD: DRTM instruction, SKINIT

# Building RoTs

## Trusted Building Blocks (TBBs)

Trusted Building Blocks (TBB) are the parts of the RoTs that do not have shielded locations or protected capabilities. Normally these include just the instructions for the RTM and TPM initialization functions (enable, reset, etc.).

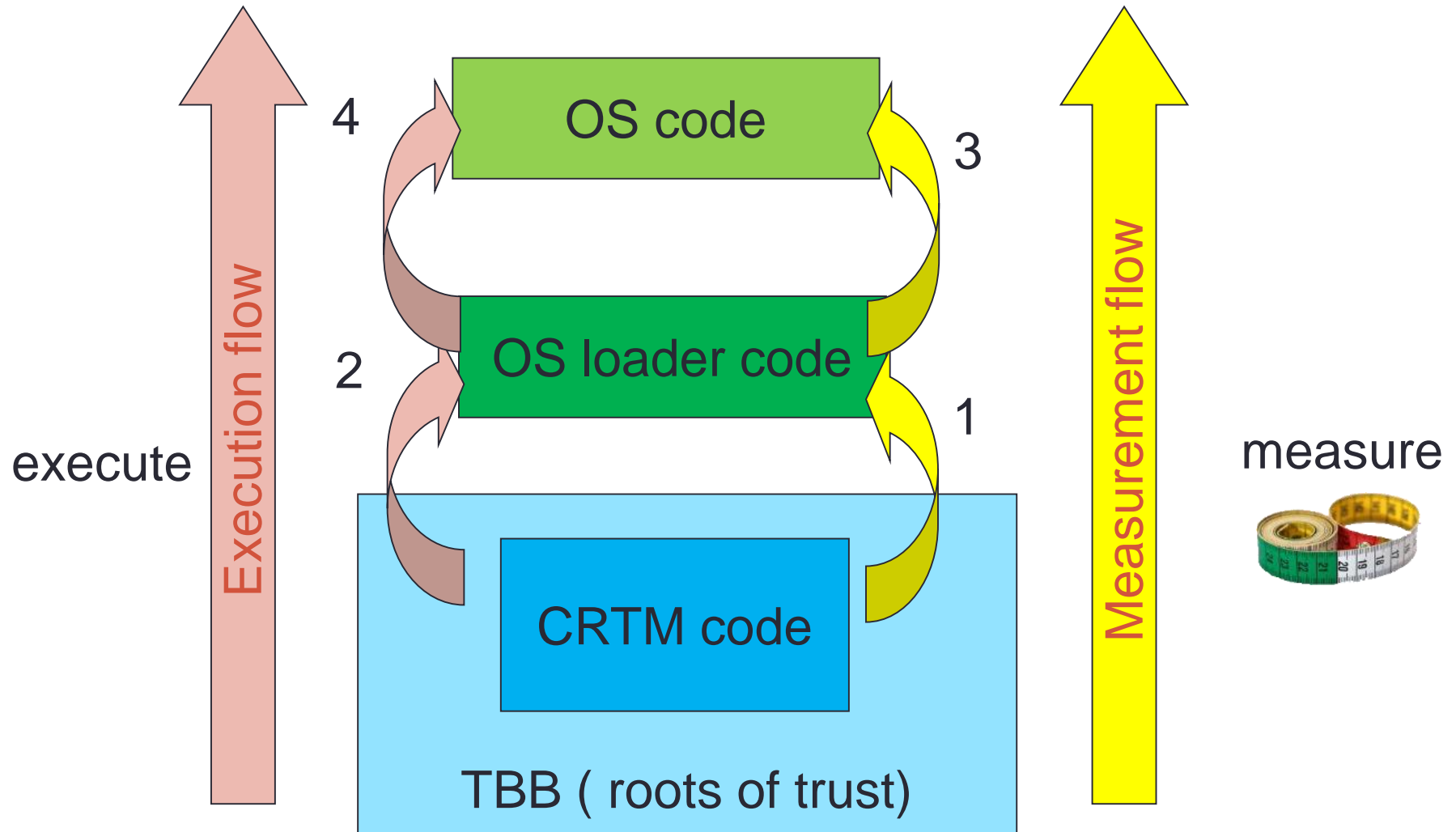
- One example of a TBB is the combination of the CRTM, connection of the CRTM storage to a motherboard, the connection of the TPM to a motherboard, and mechanisms for determining Physical Presence.

The TBB is trusted through the platform certificate which implies that the TBB will behave so the protection by trusted computing is not compromised

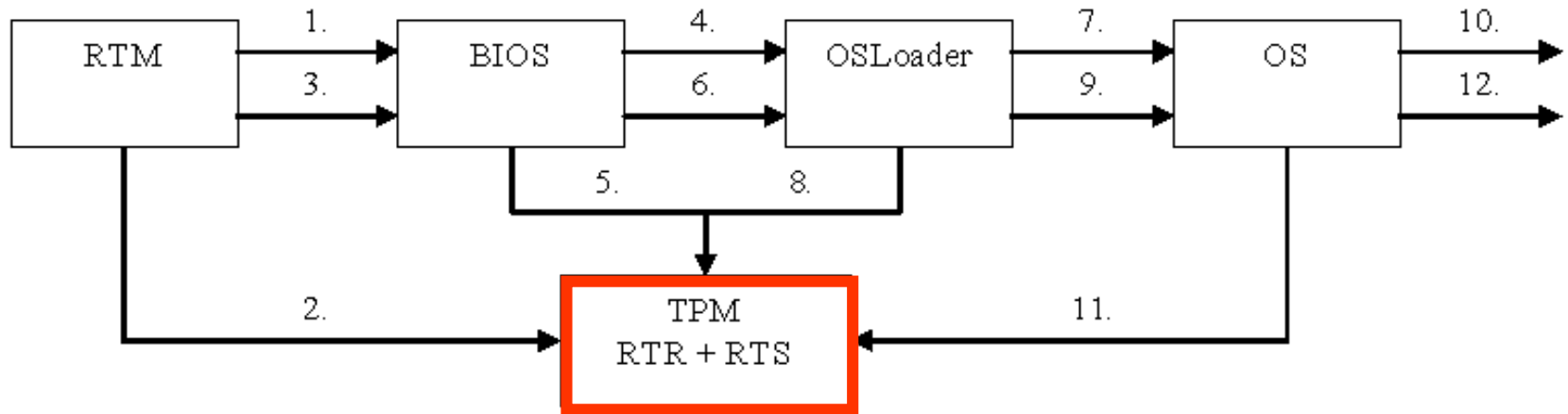
Trust in the connection of the CRTM to the TPM is transitive, and relies upon trust in the CRTM connection to the system/platform and in the TPM connection to the system/platform. Here the TPM and Platform certificates are essential!



# Putting RoT into use for secure boot



# BOOT Process with TPM



1. The *Root of Trust for Measurement* (RTM) measures the BIOS metric.
2. Measured value is reported to the *Trusted Platform Module* (TPM).
3. BIOS is executed.
4. *Operating System* (OS) Loader metric is measured.
5. Measured value is reported.

6. OSLoader is executed.
7. OS metric is measured.
8. Measured value is reported.
9. OS is executed.
10. An application is measured.
11. Measured value is reported.
12. Application is executed

# But wait a minute!

- But if we only measure the code we launch this doesn't say that we prevent bad code from running!
- Indeed !
- Hence in the TBB there must be logic that decides what will happen.
- Common logic (behavior) is
  - Authenticated Boot
  - Verified Boot
  - Or hybrid thereof

# Secure bootstrap: secure vs authenticated boot

Two methods of booting

- **Secure Boot:** boot can be halted when check fails
- **Authenticated (or Measured) Boot:** just reporting

For checking we can use

- Integrity metric (e.g. hashes, signatures)
- Platform Configuration Register (PCR)

# TPM WORKING

---

## Part B:

- The TPM functions themselves
- Software stack for TPM
- TPM in actual use
  - e.g secure boot, in cloud systems

# TPM working B- overview

We will look at

- The TPM functions themselves
  - TPM key types
  - TPM Key hierarchy
  - TPM commands
  - TPM command sessions
  - Binding and Sealing
  - Key Migration
- TPM secure storage
- Software stack for TPM
- TPM in actual use

# TPM Versions

- Most TPM in use are TPM version 1.2
- There was a TPM version 1.0 but we can today forget about that version.
- TPM Version 2.0 is being introduced, differences between these version we return to later. There are many differences but on a high level there are a lot communalities.

# Main keys – always remain in TPM

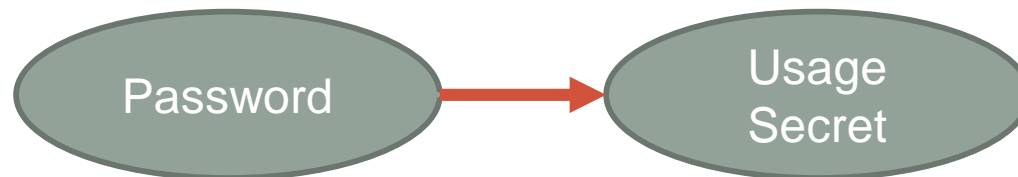
1. **Endorsement Key (EK)** (2048-bit RSA)
  - Created at manufacturing time. Cannot be changed.
  - Used for “attestation” (described later)
2. **Storage Root Key (SRK)** (2048-bit RSA)
  - Used for implementing encrypted storage
  - Created after running  
**TPM\_TakeOwnership( OwnerPassword, ... )**
  - Can be cleared later with **TPM\_ForceClear** from BIOS
3. **OwnerPwd(password)** (160 bits) and persistent **flags**

Private **EK**, **SRK**, and **OwnerPwd** never leave the TPM



# Passwords and Secrets

- When taking ownership an owner(ship) secret is set that is needed later for certain TPM commands
- Each key except EK has a usage secret which must be presented when certain operations with the key is to be performed. (one could regard the owner secret as the usage secret of EK).
- To each secret is connected a password from which it could be derived.



# Authdata

In fact each TPM object or resource (e.g., a key) is associated with an authdata value a 160-bit shared secret between a user process and the TPM.

The user processes issuing a command that uses a TPM object or resource must provide proof of knowledge of the associated authdata.

- HMAC of command arguments, keyed with a value based on the authdata

# TakeOwnership

- The TakeOwnership results in
  - a (re)computation of the SRK private and public key
  - The usage secret for SRK is set
  - The owner secret is set
  - A new **tpmProof** value is set which is a random value kept secret inside the tpm
  - Future reading of pubEK will require knowledge of owner secret

The tpmProof is used in several TPM functions to give a binding which is a) unique for the given TPM and b) unique for the current active TPM. We will example of this later.

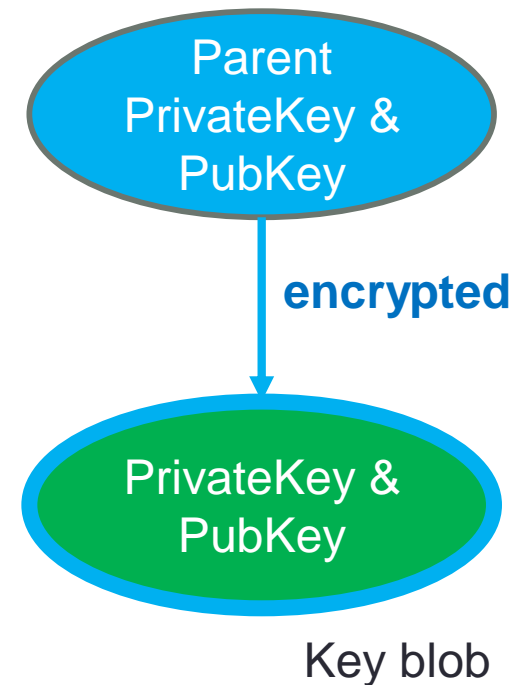
# TPM Key types

- Endorsement Key (EK)
- Storage Root Key (SRK)
- Attestation Identity Key<sup>s</sup> (AIK)
  - sign data from the TPM. A TPM can have many identities!
- Storage: encrypt data, including other keys, (SRK is a special storage key)
- Signing: key only for signing
- Binding: decrypt data (usually from remote platforms)
- Certified Migration Key (CMK), is of one above type but tagged as migratable
- Legacy: signing or encryption (compatible with TPM v1)

# TPM Protected key storage

- So TPM holds only two permanent keys
  - The EK (sort of TPM private key)
  - The SRK
- Internal storage is smartcard alike
  - Persistent, secure storage of keys
  - Not 100 % hardware tamper resistant

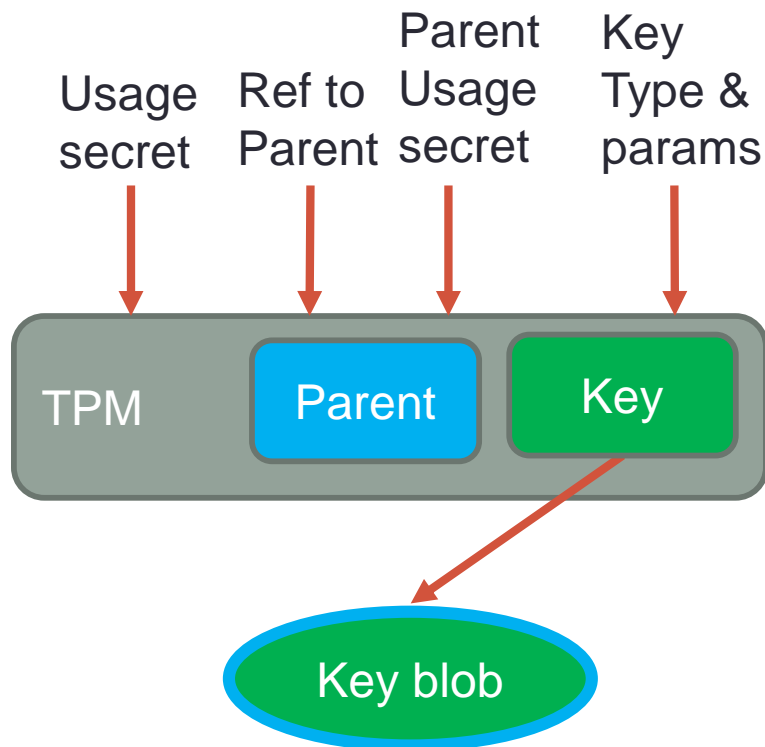
Other keys are stored encrypted outside using the public key of storage parent key and loaded internally as needed during processing. The encrypted object is called a *key blob*



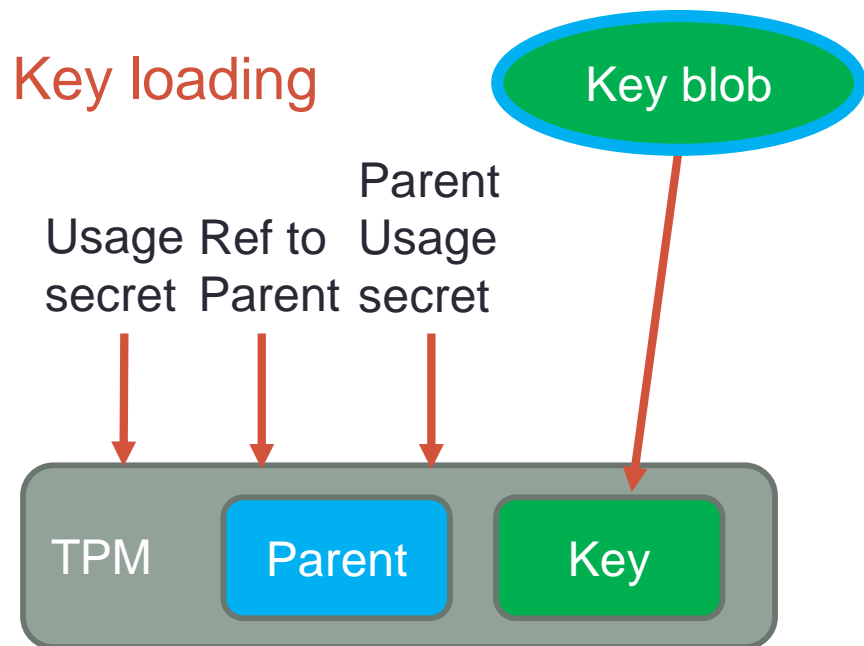
# Key generation & loading

- When a key (except EK, SRK) is generated we get a key blob

## Key generation

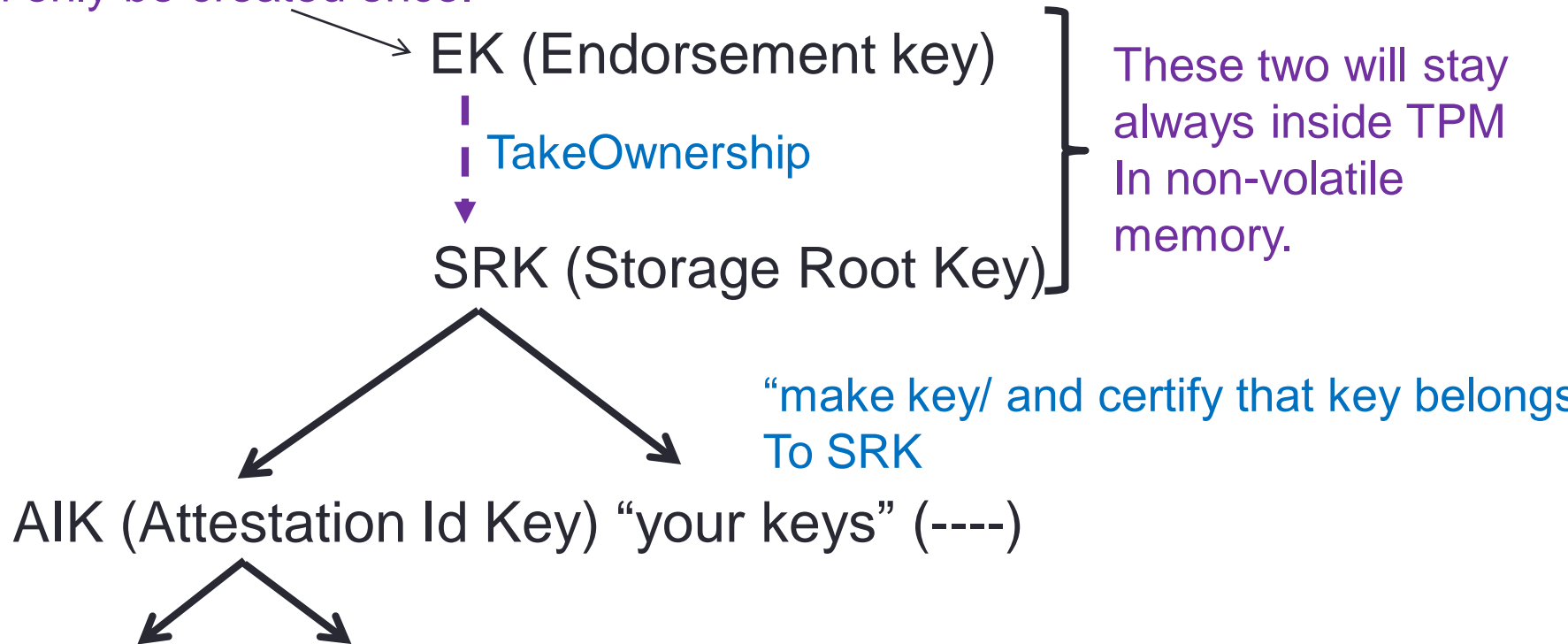


## Key loading



# Key Hierarchies

Can only be created once.



Question: How to prove that SRK inside TPM ?

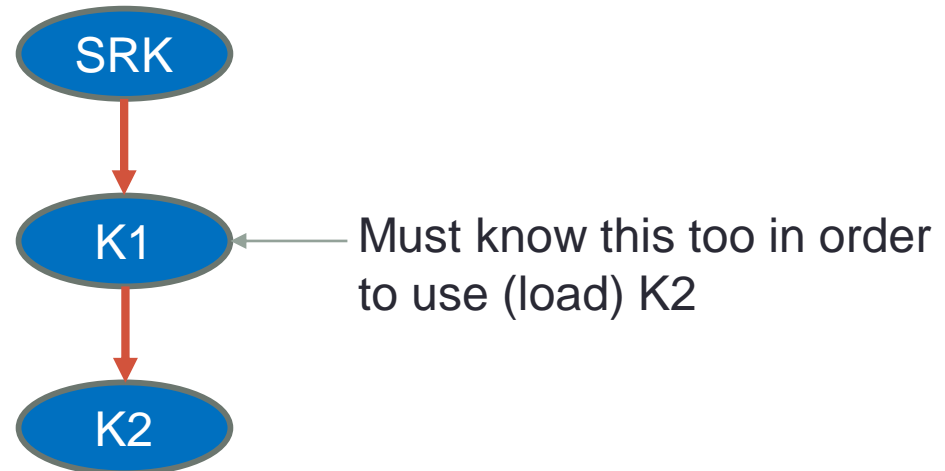
# Endorsement key

- Is a very special key since it stays the same during the TPM lifetime. This gives privacy concern due to linkability to the user when the EK is used.
- To reduce the risk of EK being used in an improper way even the use of EK is limited
  - Basically it allows only EK for encryption. So signing type of operations are not allowed.
  - We come back to the use of EK later



# Two Warnings

- All keys “below” SRK are lost for ever if a new SRK is generated by a re-takeOwnership command.
- To be able to load a key into the TPM by importing its key blob one must have all the parent keys between SRK and the key



# TPM Protected storage - data

- The TPM
  - can wrap data (wrap = encrypt using a key)
  - “Sealing”: binds data to a certain value of the PCR. Then the TPM can only decrypt (unseal) if the PCR value(s) is the same as when encryption happened (seal)
- Management: migration, backup

# TPM management

- Enabling and disabling (via BIOS) – recall the TPM state
- Generating keys
- Moving (migrating) keys between TPMs

# TPM commands

- Interaction with the TPM occurs through commands send to the TPM and responses the TPM return as a result.
- Example: on the right the TakeOwnership command and response data.
- The TPM\_COMMAND\_CODE type field contains the command number of the TakeOwnership command

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The ownership protocol in use.
5	4	3S	4	UINT32	encOwnerAuthSize	The size of the encOwnerAuth field
6	↔	4S	↔	BYTE[]	encOwnerAuth	The owner AuthData encrypted with PUBEK
7	4	5S	4	UINT32	encSrKAuthSize	The size of the encSrKAuth field
8	↔	6S	↔	BYTE[]	encSrKAuth	The SRK AuthData encrypted with PUBEK
9	↔	7S	↔	TPM_KEY	srkParams	Structure containing all parameters of new SRK. pubKey keyLength & encSize are both 0. This structure MAY be TPM_KEY12.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for this command
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	ownerAuth	Authorization session digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	↔	3S	↔	TPM_KEY	srkPub	Structure containing all parameters of new SRK. srkPub encData is set to 0. This structure MAY be TPM_KEY12.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: the new ownerAuth value

# TPM sessions

In order to protect the communication between the application and the TPM most commands support protection mechanisms.

- **Use of the authdata of an object**

Since the interaction of an application with the TPM may involve several commands that consecutively have to be performed the TPM supports sessions

- TPM1.2 supports three types of session
  - **OIAP**: Object Independent Authorization Protocol which creates a session that can manipulate any object, but works only for certain command
  - **OSAP**: Object Specific Authorization Protocol which creates a session that manipulates a specific object specified when the session is set up.
  - **DSAP**: Delegate-specific Authorization Protocol. Similarly to OSAP sessions, DSAP sessions are restricted to a single object.

TPM2.0 will do this differently

