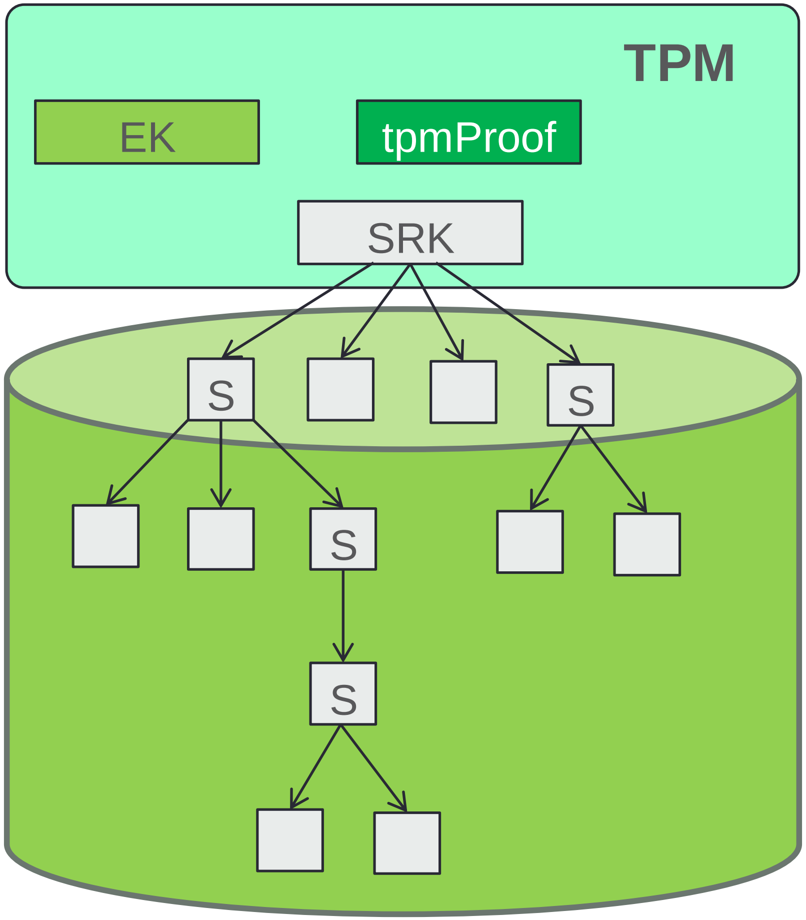
TRUSTED PLATFORMS

TPM continue'd

Lect 5

Key Hierarchy in TPM 1.2

- tpmProof can be used to tie keys to the TPM
- Each key has 160-bit authData
- EK is static throughout the lifetime of the TPM
  - Inserted by manufacturer
  - Comes with EK certificate stating it is a genuine EK
- **Intermediate keys** are storage keys
  - SRK is the root
  - Never leaves the TPM (fixed handle)
- **Leaves** are special purpose keys
  - Binding keys
  - Sealing keys
  - Attestation identity keys (AIK)
  - Signing keys



Stored outside TPM, e.g., hard-disk

TPM Protected storage - data

- The TPM
  - “Binding”: we can wrap data (wrap = encrypt using a key)
    - Thus **binding** (to a key)

  - “Sealing”: binds data to a certain value of the PCR and a key that is not migratable. 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

- Beside enabling and disabling (via BIOS)

- 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

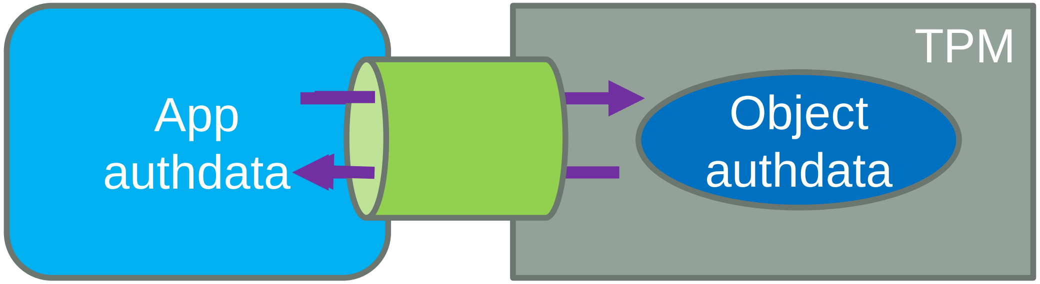
In order to protect the communication between the application and the TPM most commands support protection mechanisms.

- Use of the authdata of an object

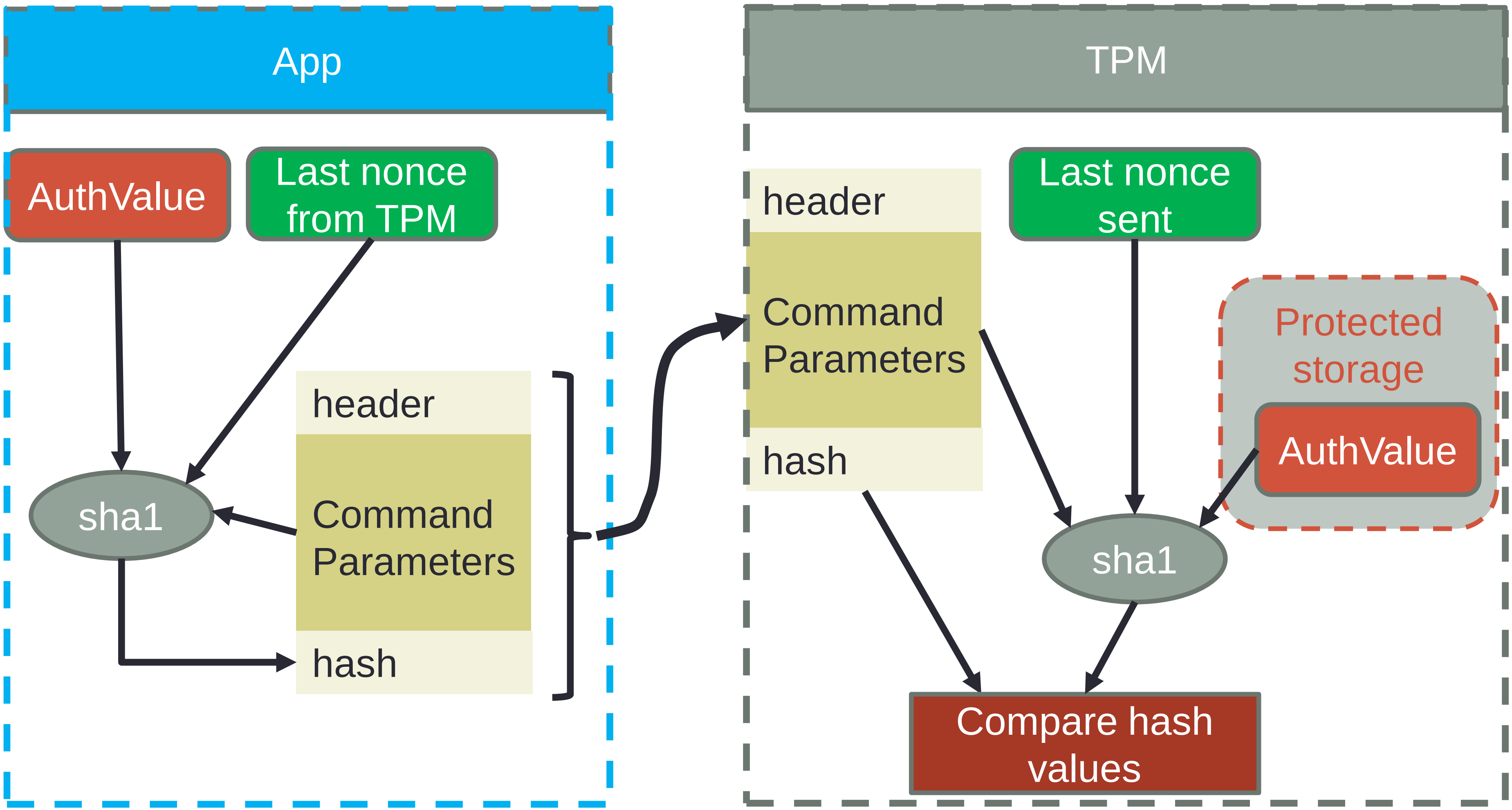
Since the interaction of an application with the TPM may involve several commands that consecutively have to be performed the TPM supports sessions

- 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



TPM command authentication



Binding

Binding is encrypting data using a the public key of a bind key. If the bind key is non-migratable the encrypted data is binded to the TPM where the secret portion of the bind key resides.

Binding is done outside the TPM (so there is no TPM_Bind command)

TPM_UnBind

- INPUT:
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: auth for using key with id `KeyHandle'
  - edata: encrypted data
  - edata_len
- OUTPUT:
  - data decrypted

Protected Storage: SEAL

Main Step: Encrypt data using RSA key on TPM

TPM_Seal

- INPUT:
  - KeyHandle: which TPM key to encrypt with
  - KeyAuth: auth for using key with id `KeyHandle'
  - PcrList: list with indices J and PCR[i] $i \in J$ to be embedded in output sealedData
  - data: at most 256 bytes (2048 bits)

    (typically used to encrypt symmetric key (e.g. AES))
- OUTPUT:
  - sealedData RSA encrypted data (and PcrList)

MAC protected list by TPM_Seal

- SEAL to set of PCRs J=(i1,…,in) using key

- TPM_Seal(J , data) $\rightarrow$

  (C,MAC(SK,((i1, PCR[i1]), (i2, PCR[i2]), ...))

Ex: C=RSA(SK,data) and SK is storage key

Protected Storage: UNSEAL

Main Step: Decrypt data using RSA key on TPM

TPM_UnSeal

- INPUT:
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: password for using key with id `KeyHandle'
  - sealedData: RSA decrypted data and PcrList
- OUTPUT: IF and Only IF

  $\forall$ i $\in$ J current PCR[i] = PCR in MAC protected list
  - data:

Example: Use case of a sealed key

- **Problem**: We want during machine start read out a secret that decrypts some site specific TLS client certificates (containing secret PKI keys) and put it into RAM but when the OS kicks-in the secret should not be recoverable anymore.
- **Solution:** Seal the secret to a PCR that "measures" the machine state during boot. When the boot comes to the correct point its TPM can do unseal and we load the RAM with our client cert. Then we erase the secret, verify the OS code, update the PCR and start the OS. Now we no longer can unseal the secret and is protected against wrong doings by the OS and the software that is running on the OS.

Locality on TPM

Locality is a concept that allows various trusted processes on the platform to communicate with the TPM such that the TPM is aware of which trusted process is sending commands.

This is implemented using dedicated ranges of LPC bus addresses and thus requires proper support in the chipset HW (e.g. Southbridge)

There are 6 Localities given numbers 0 – 4 and None.

The 6 Localities

Locality	Description
4	Trusted hardware component. This is used to establish the Dynamic RTM.
3	Auxiliary components. Use of this is optional
2	Dynamically Launched OS (Dynamic OS) runtime environment
1	An environment for use by the Dynamic OS.
0	The Static RTM, its chain of trust and its environment.
None	This locality is defined for using TPM 1.1 type I/O-mapped addressing. The TPM behaves as if Locality 0 is selected.

In simple setups

Localities and TPM objects

Objects going in and out the TPM are in general referred to as "blobs".

The TPM enforces also locality restrictions on TPM objects such as SEALED blobs restricted by PCR attributes (we explain this later in more detail what this means).

For example: if a blob is SEALED to PCR 19 (Locality 2), a code component executing at Locality 4 (extending PCR 17) cannot UNSEAL the blob. To UNSEAL the blob

it would require to extend PCR 19 which is not possible for a component executing at Location 4

For the moment think "sealed" is a kind
of locking to a value in the TPM.

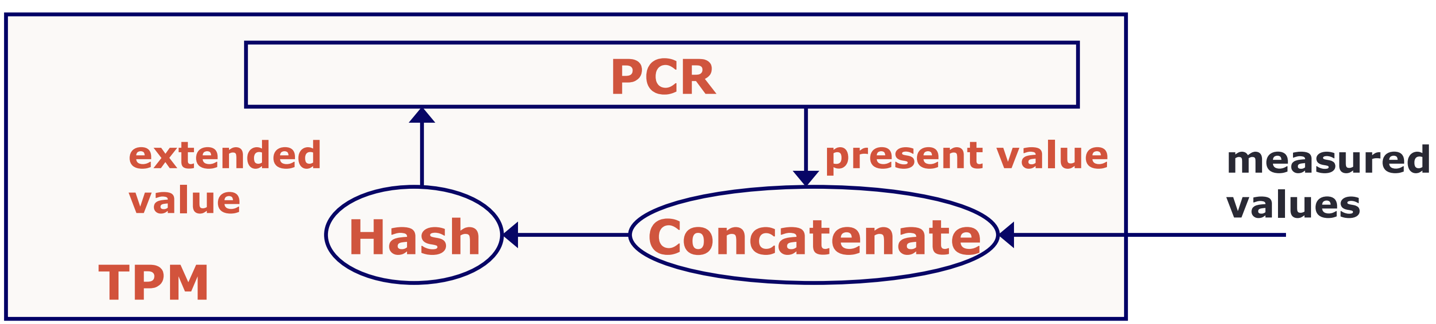
Secure bootstrap: PCRs and measuring

- Extending a PCR

  PCR_Extend(n,data): PCR[n]←SHA1(PCR[n] || data*)
- When booting
  1. Reset PCRs
  2. PCR_Extend(n,<Bios Code>)
  3. PCR_Extend(n,<MBR>)
  4. etc

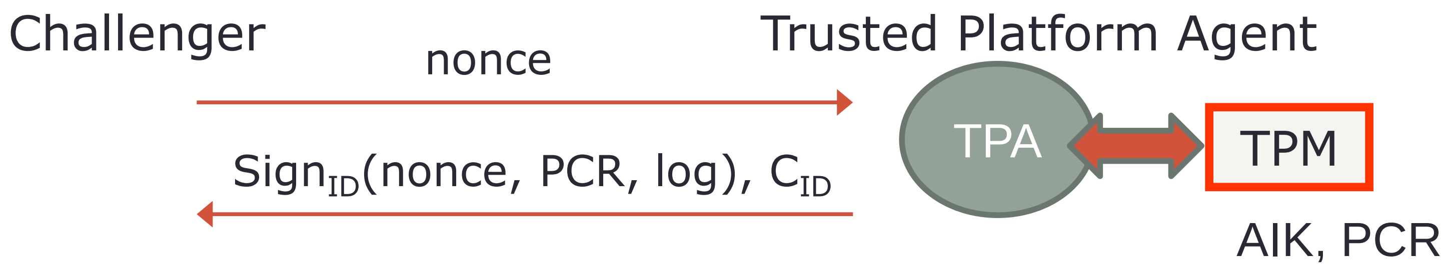
* data must be 20 bytes: Hash(data)

PCR contents reflects The SW that is loaded and running



TCG: (remote) Attestation

- Integrity reporting: report the value of the PCR
- Challenge-response protocol



- AIK keys used for signing are TPM Identities (pseudonyms)
  - Use different identity (AIK) for every challenger
  - CID is a certificate proving the AIK is trustworthy

TPM_Quote (prepare)

The TPM_Quote is the command that does the attestation

CreateIdentity

- A new Attestation Identity Key (AIK) is generated
- The AIK is linked to the TPM/EK by issuing a certificate that is has no link to EK (for privacy reasons)
- This certificate is issues by a special procedure in which the AIK certificate issuer uses the knowledge of genuine EKs. (we return to that shortly)

TPM_Quote

The TPM_Quote is the command that does the attestation

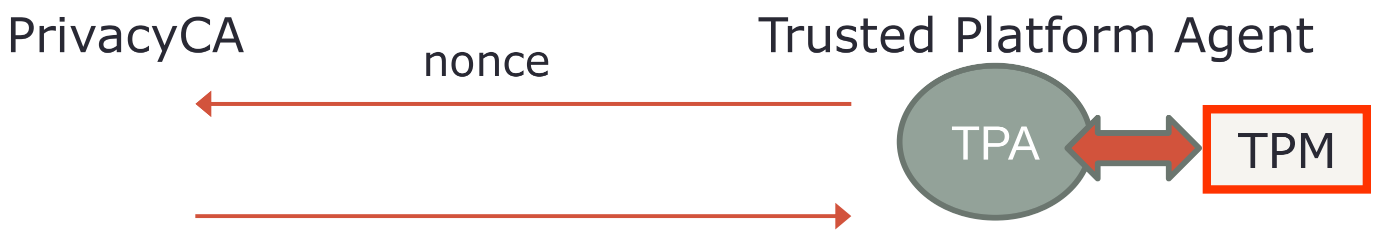
TPM_Quote

- INPUT:
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: password for using key with id `KeyHandle’
  - PCR list: the PCR to quote
  - ExternalData: 20 byte value
    - challenge to prevent replay attacks
    - Hash of a challenge and userdata to be included in the quote signature

- OUTPUT:
  - The signature of the quoted data.

Trusting the AIK – privacy CA

- How can we trust the AIK? Is it really an AIK inside a real TPM or are do we deal with an emulator.
- We cannot sign the (pub)AIK with EK´. There is no command for that

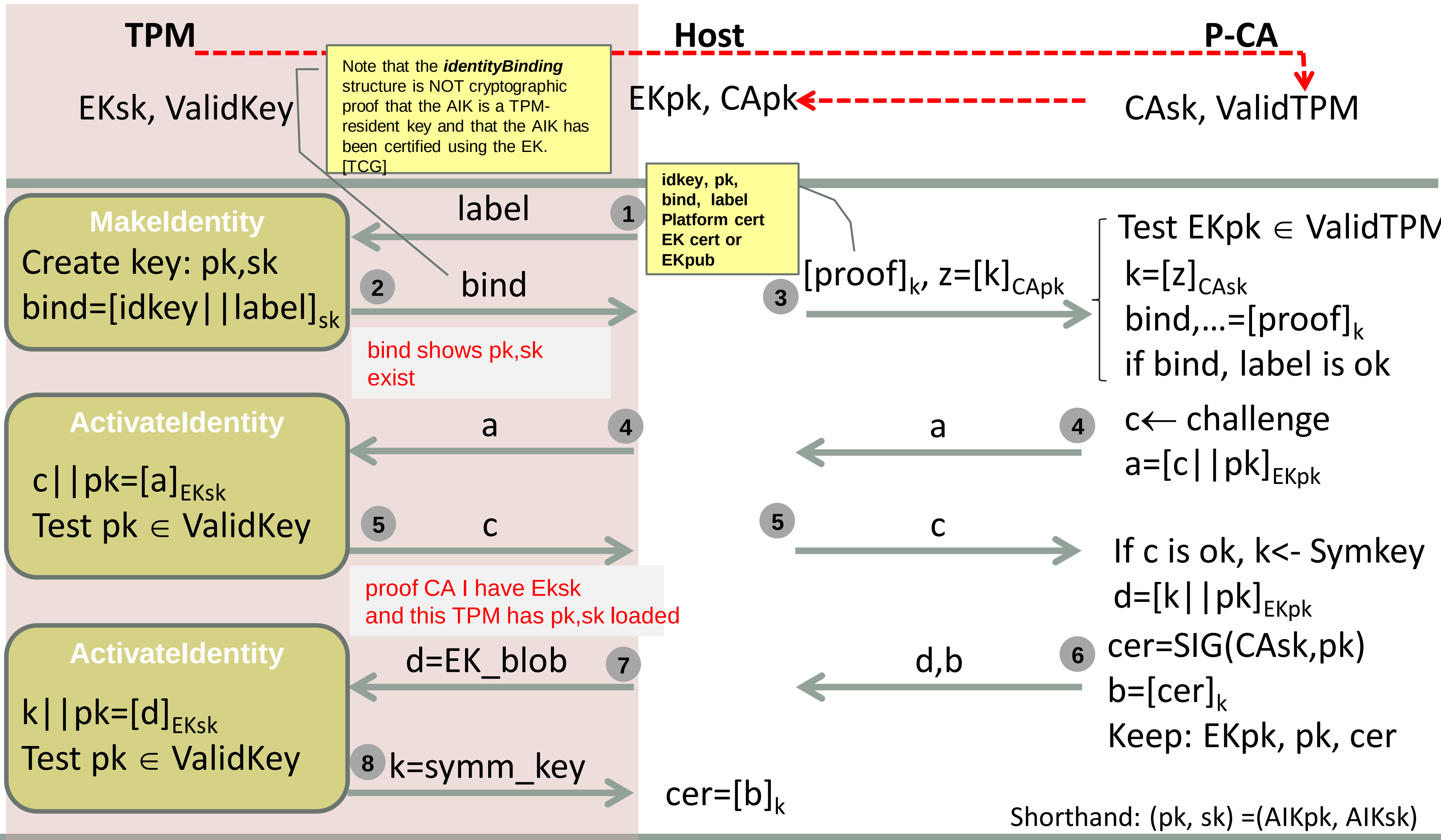


- We extract the pubAIK and then ask for a certificate

- The issuer encrypts the certificate with a key that is encrypted by pubEK so only the TPM with the right EK can recover the certificate. The TPA sends a proof that it had the correct TPM (the one with the correct EK)

TCG Privacy-CA (sketch)

Privacy-Preserving AIK Certificate Enrollment

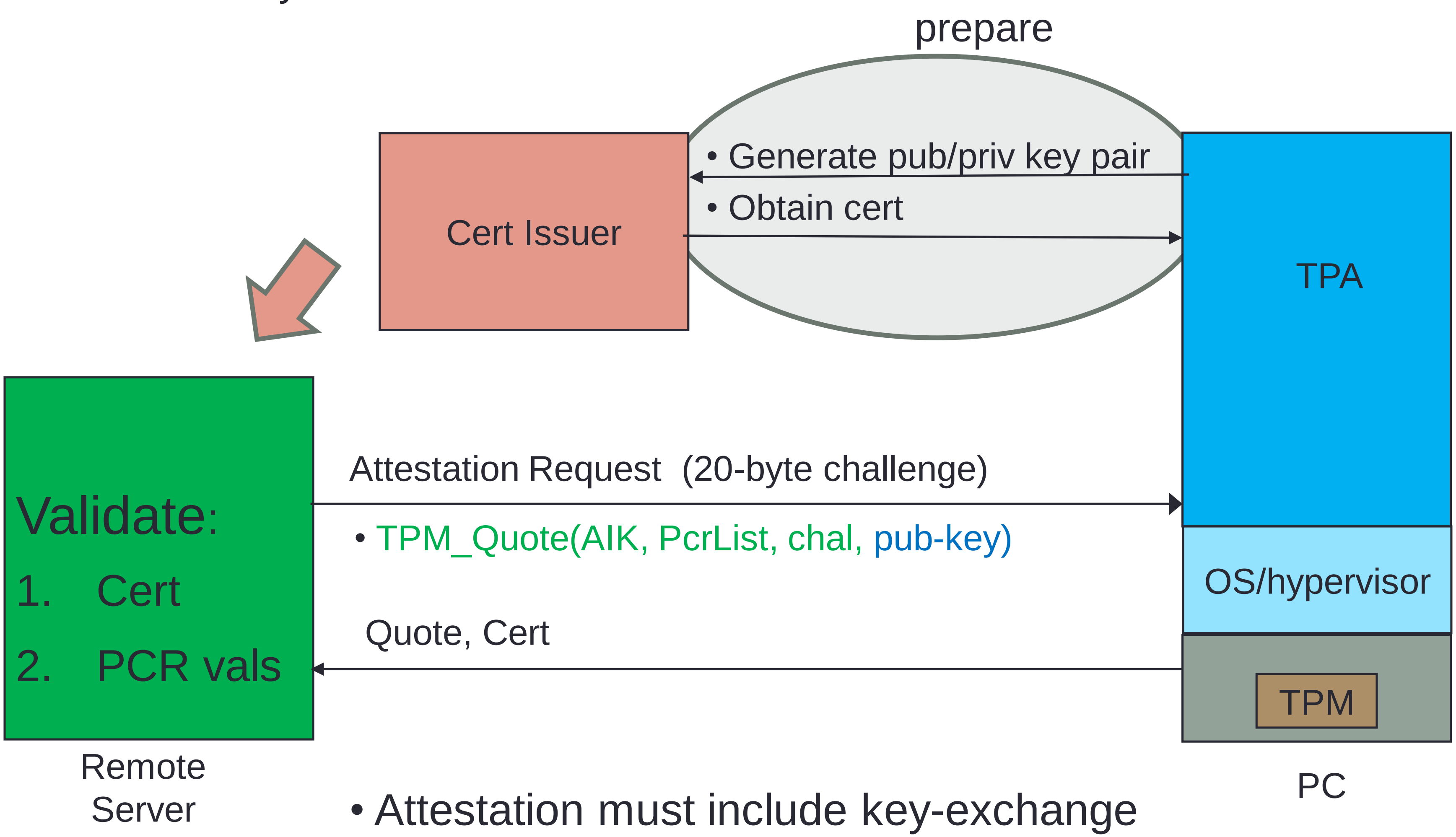


Shorthand: (pk, sk) =(AIKpk, AIKsk)

= Out of band

Attestation: 1

In an actual system



- Attestation must include key-exchange
- App must be isolated from rest of system

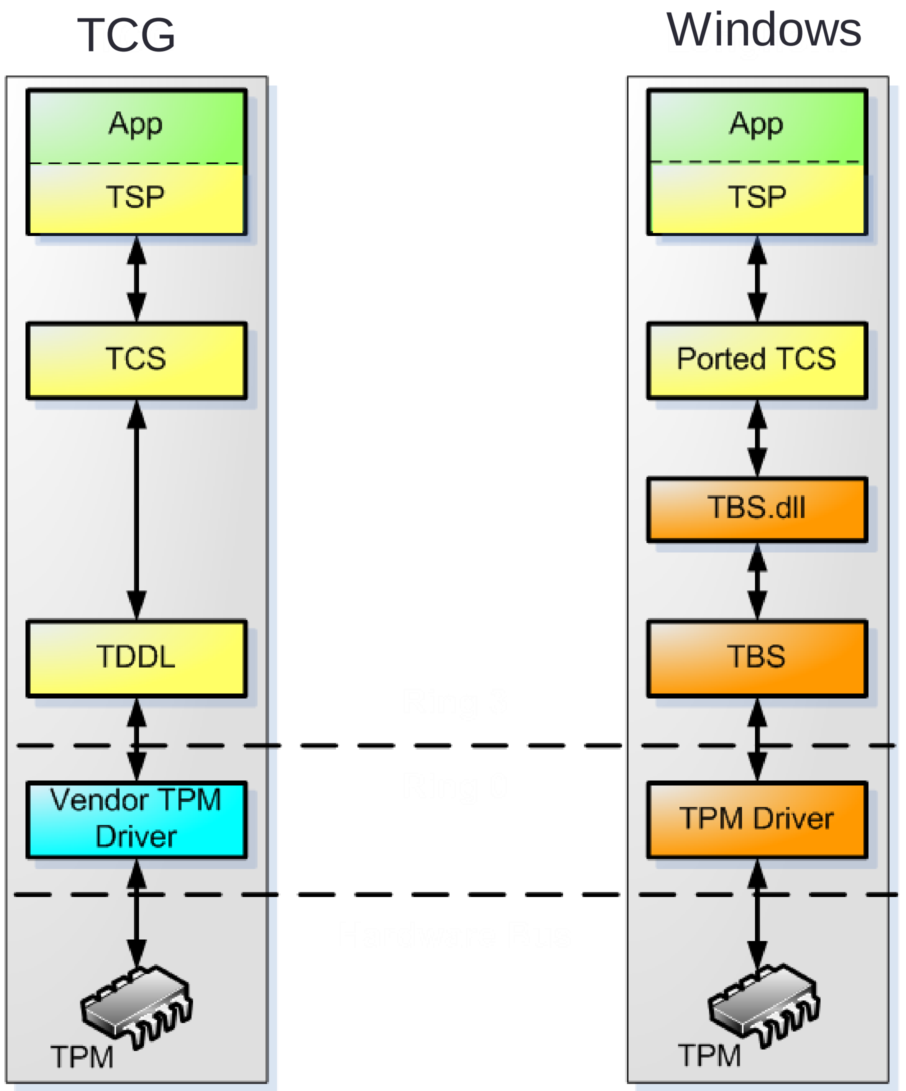
AIKs: privacyCA and concerns

- Initially a privacyCA was architectured through which AIKs could be bind to a specific TPM.
- Yet there were concerns that the binding compromises anonymity and therefore TCG has implemented a more advanced attestation method based on zero-knowledge techniques: Direct Anonymous Attestation (DAA).
- DAA is a cryptographic protocol which enables the remote authentication of a trusted platform yet preserving the user's privacy. DAA is a very complicated protocol.

Zero-knowledge technique: idea is to have an interactive protocol in which a verifier can
Be convinced that the other party has a secret without any knowledge of the secret
being revealed.

TCG Stack vs. TPM Services Stack

- TPM applications use the TCG Service Provider (TSP) interfaces
- The TCG Core Services component (TCS) is ported to communicate with the TBS instead of the TCG Device Driver Layer (TDDL)
- TPM applications are more agile and better protected when using TBS



TPM in actual use

- TPM support in Windows 7, 8 and 10
  - Use case
    - Bitlocker (file encryption)
    - Secure boot (UEFI boot)

- TPM and Intel TXT
  - OpenStack trust pools
  - (Intel Cloud Integrity Technology)

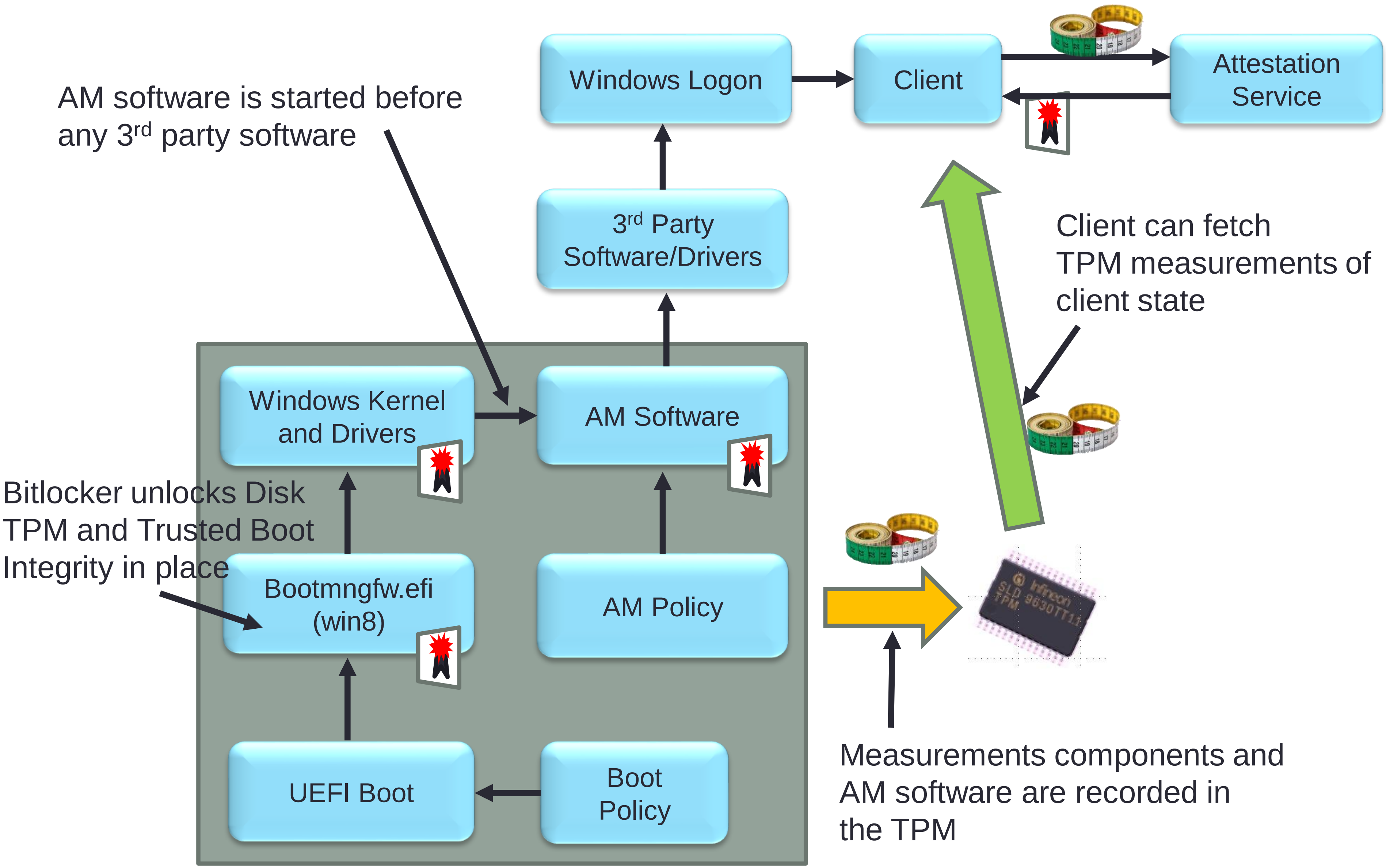
Secure boot

Trusted boot is generic term that covers different ways to improve the security of a systems boot.

Warning: Terminology in different publications is not 100% aligned

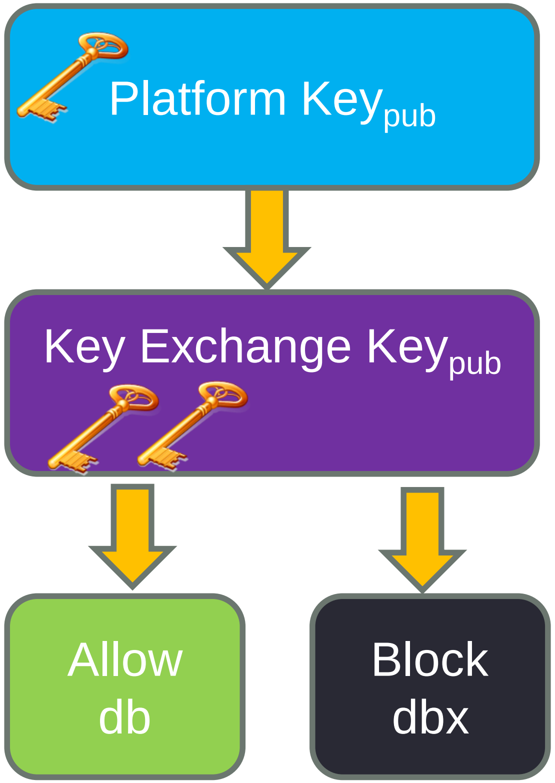
- **Measured boot:** the boot components are measured and measurements are recorded but boot continues irrespective if code of components has correct characteristic (hash or signature) or not
- **Secure or Verified boot**: the boot components are verified using a defined procedure (e.g. hash or signature) and boot halts if checks will fail.

UEFi Trusted Boot Archtecture



UEFi Secure Boot Keys

- Platform Key (PK)
  - Only one
  - Allows modification of KEKs
- Key Exchange Key(KEK)
  - Can be multiple
  - Allows modification of db and dbx
- Authorized Database(db)
  - CA, key, or image hash to allow
- Forbidden Database(dbx)
  - CA, key, or image hash to block



Keys required for Secure Boot

Microsoft's setup

Key/db name	variable	Owner	Info
PKpub	PK	OEM	Must be RSA 2048 or stronger
Microsoft KEK CA	KEK	Microsoft	Allows updates to db and dbx
Microsoft Windows Production CA	db	Microsoft	This CA in the allowed signature database (db) allows Windows 8 to boot
Forbidden Signature database	dbx	Microsoft	List of bad/compromised keys, CAs images from Microsoft

+ Required for secure firmware updates

Key/db name	Owner	Info
Secure firmware update key	OEM	Should differ from PK, Must be RSA 2048 or stronger

UEFi secure boot and TPM

- Observe that actually the TPM is not needed for secure boot if one skips the requirement to support attestation.

(one basically has no secrets to protect then).

http://technet.microsoft.com/en-us/library/hh824987.aspx

◢ Manufacturing Requirements

Secure Boot requires a computer that meets the UEFI Specifications Version 2.3.1, Errata C or higher.

Secure Boot is supported for UEFI Class 2 and Class 3 computers. For UEFI Class 2 computers, when Secure Boot is enabled, the compatibility support module (CSM) must be disabled so that the computer can only boot authorized, UEFI-based operating systems.

Secure Boot does not require a Trusted Platform Module (TPM).

To enable kernel-mode debugging, enable TESTSIGNING, or to disable NX, you must disable Secure Boot. For more info, see Windows 8 Secure Boot Key Creation and Management Guidance.

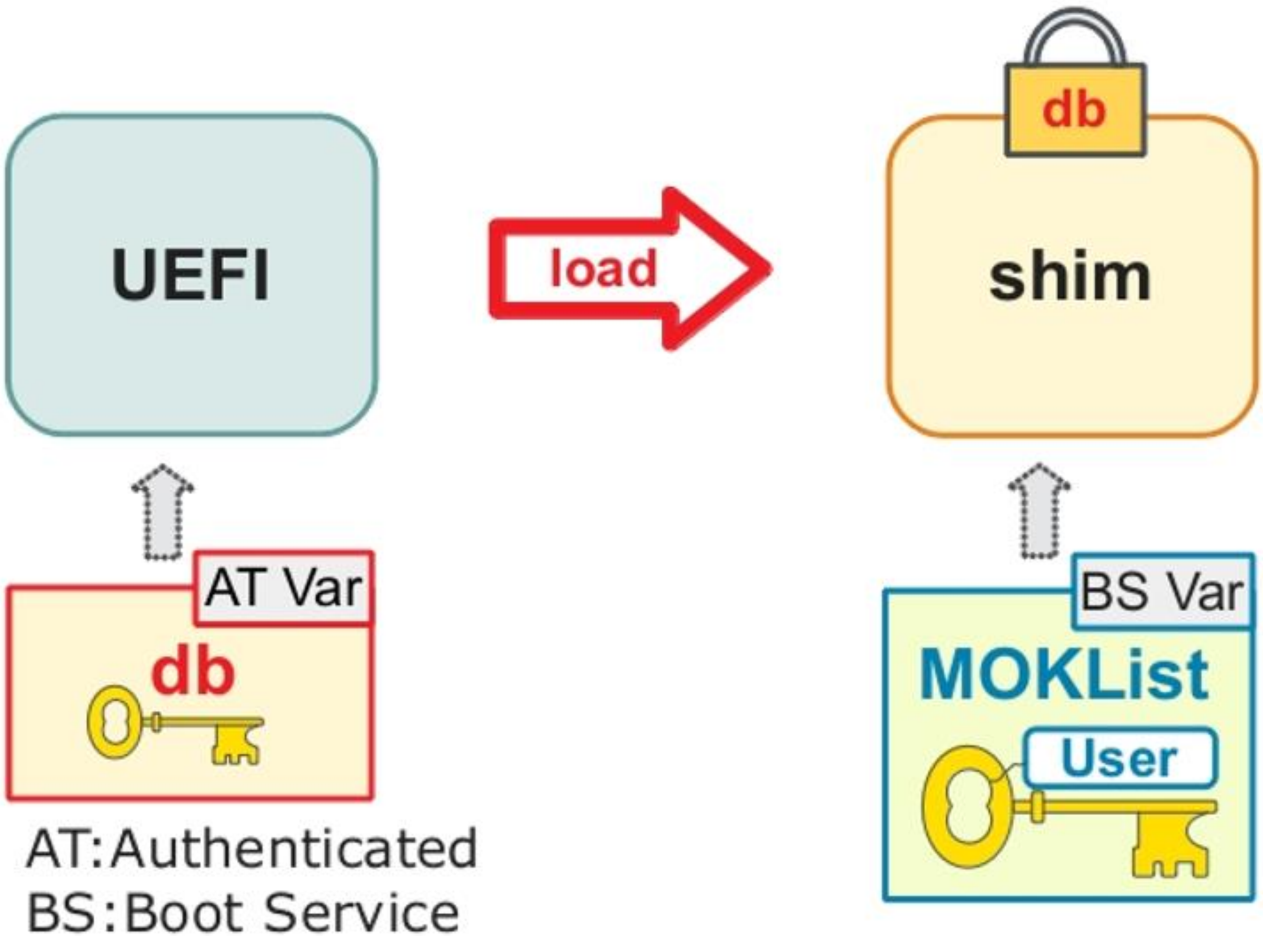
OEM's role

- Thus the OEM generates and own the PK secret key.
  - Basically the OEM can decide what can be loaded/booted is defined in the boot policy
- However, Microsoft, can demand the OEM boot policy to comply with Microsoft requirements if the OEM want to run Microsoft software (say Windows 8 or 10)

Criticism

THE CIRCUIT
Software Freedom Law Center
Blog
Microsoft confirms UEFI fears, locks down ARM devices
By Aaron Williamson | January 12, 2012
PCWorld
LINUX LINE Linux and open-source news and advice
Windows 8 Secure Boot: The Controversy Continues
Jan 18, 2012 10:29 AM
By Katherine Noyes, PCWorld
TECHSPOT
Microsoft under fire in Europe over Windows 8 UEFI Secure Boot
By Jose Vilches
Microsoft Windows 8 UEFI Secure Boot complaint: The case for and against
By Liam Tung | March 28, 2013
fsfe
Free Software Foundation Europe is a non-profit organisation dedicated to promoting Free Software
"Secure Boot": Who will control your next computer?

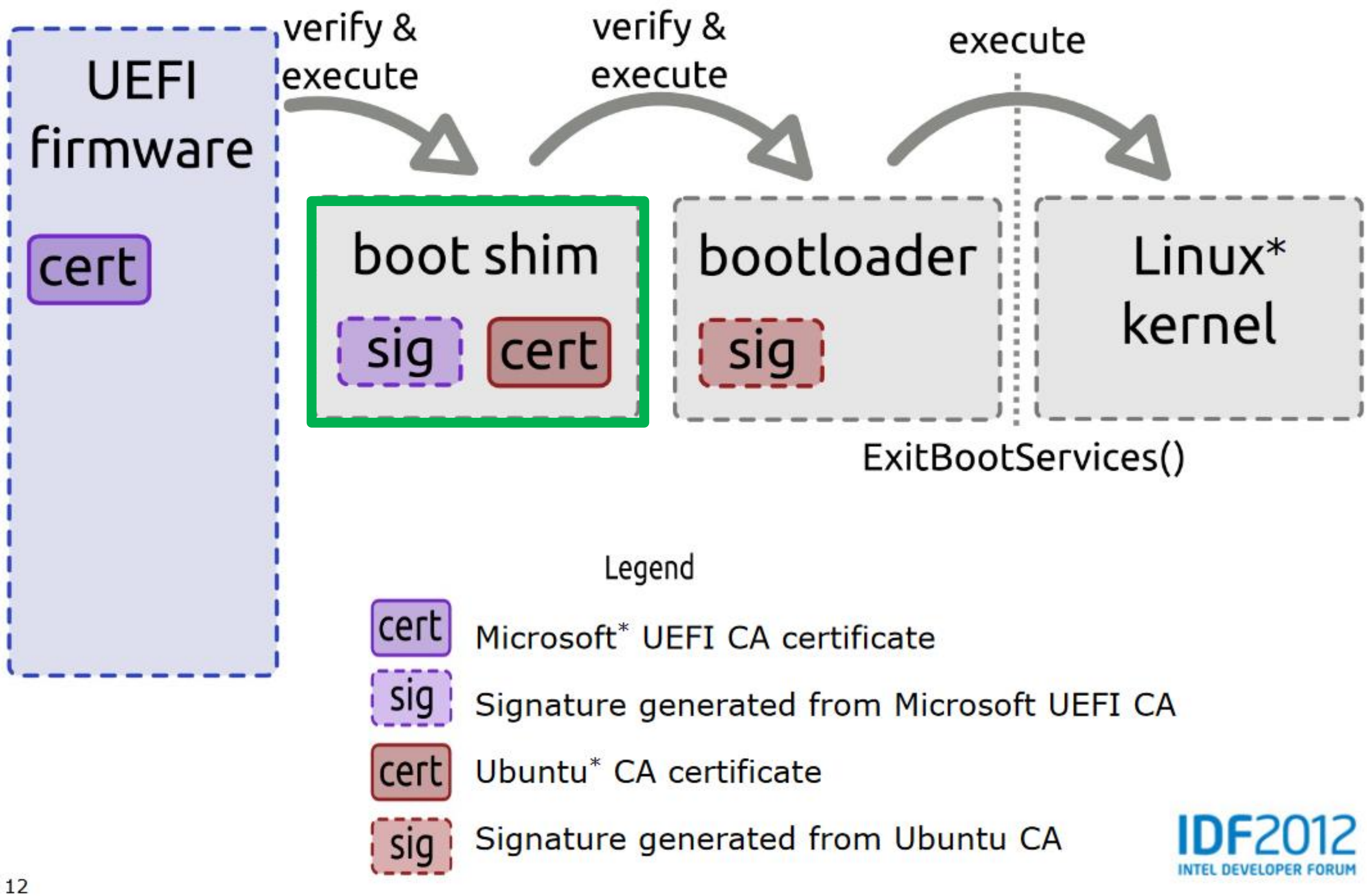
Linux secure boot - SHIMs



MOClist in UEFI NVRAM

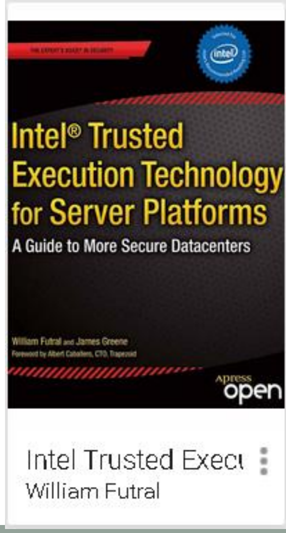
MOK = Machine Owner Key

Ubuntu – shim approach



Intel TXT

- Intel has implemented the TCG technology in something which is called Intel Trusted eXecution Technology (TXT)
  - It affects beside the CPU other chipsets and required a TPM with at least 24 PCRs.
  - The use of localities is essential
  - And it implements a **D-RTM**. That is we can move to a trustworthy state during runtime even if we were not running in a trusted/verified state before



- The purpose of TXT is getting a trusted execution platform. (and not really creating a Cryptographic Service Provider)

Intel TXT components

- **The CPU**
  - New instruction: GETSEC
    - with leave functions SENTER and ENTERACCS
    - Halt execution of cores and calls SINT ACM,
- **Other HW in the PC/server**
  - ACM modules (firmware)

  Special signed sw modules by HW manufacturer that execute at highest security level and execute in special separate secure memory

  BIOS ACM: code that measures BIOS +init

  SINT ACM: code that is part of the DRTM for the secure init/launch

Intel TXT for system with hypervisor

1 Provisioning:
Known good values for BIOS
And hypervisor are provisioned into the TPM

Policies and configuration files determine behavior

2 At power up, Intel TXT initiates measured launch of BIOS and report in PCRs. Do the result match ?

3
If match:
Indicate hypervisor in trusted state

If mismatch:
A policy action is enforced
Platform indicates un trusted state

4 Intel TXT initiated measure launch of hypervisor. Do the results match ?

5
If match:
Indicate hypervisor in trusted state

If mismatch:
A policy action is enforced
Platform indicates un trusted state

Software measured and verified

Platform trustworthiness is reported

Intel TXT measurements – at boot

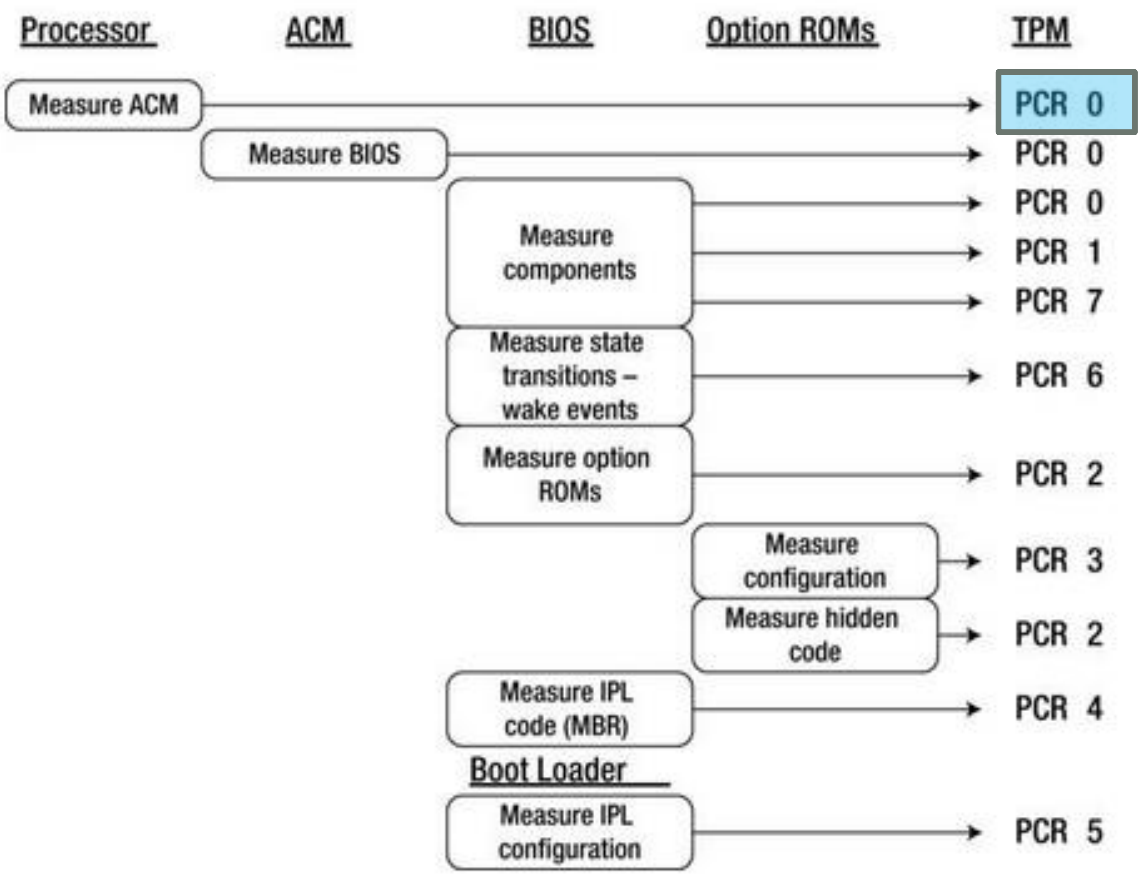


Figure 2-4. Typical static measurement sequence

Reset Attack – and protection

- A reset attack is where an attacker causes a platform reset before the OS/hypervisor can do an orderly shutdown.
  - As a result there can be secrets such as encryption keys, passwords, and personal information left unprotected in memory.

- To mitigate such an attack Intel TXT maintains a “Secrets” flag, which the OS/hypervisor sets <u>after</u> it does a measured launch, but <u>before</u> placing any secrets in memory.
- Normally, the OS/hypervisor clears the Secrets flag when it does a graceful shutdown (after removing all secrets and sensitive information from memory).
- If the platform resets before the Secrets flag is cleared, one relies during boot/init on **ACM** functionality to act on a remaining Secrets flag by disabling memory controllers and clear the memory only after proper checking.

Trusted Computing Pools -idea

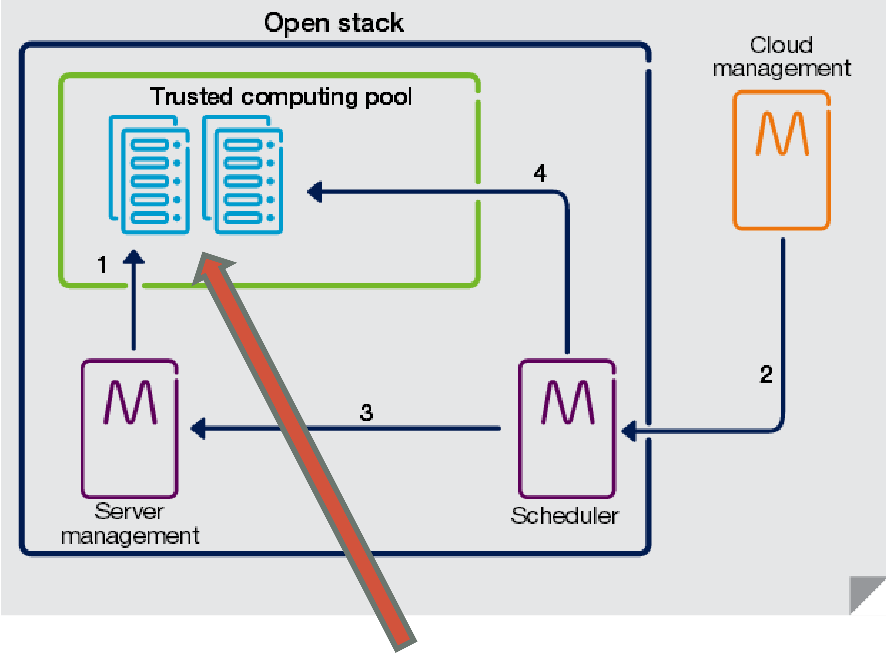
1. Server checks via remote attestation the compute resources and marks those that ok as trusted

2. Cloud manager initiated Virtual Machine (VM)

3. OpenStack scheduler checks server for trusted compute resource

4. VM is launched on trusted resource

FIGURE 4 Trusted computing attestation process



Intel TXT

See
[Trusted computing for infrastructures](#)
[Trusted Computing Pools](#)

TCG/TPM Issues 1

- Often seen as DRM enabler (no longer true)
- Privacy issues
  - TPM has unique embedded key
  - Get pseudonyms in anonymous way (e.g. anonymous credentials)
- User loses control over own computer
  - Secure boot: refuse to boot own compiled open source kernel
  - Disable reverse engineering: need for hardware hack

TCG/TPM Issues 2

- Current TPM 1.2 cannot be remotely managed.
  - IT-departments do not like this. Hard to use
- Integration of TPM in a product is not the same for different computer vendors (even true for PC/laptops)
- Why trust a US standard ?

Chinese TPM

- Chinese authorities: import control on equipment which uses crypto. Permission is needed.

- There TPM v2.0 is available in a Chinese version that has support for Chinese algorithms

TPM V2.0 - highlights

- Physically it is similar to the TPM1.2 and even some of the new TPM1.2 can be upgraded to a V2.0 by a firmware upgrade.
- TPM2.0 is using an entirely redesigned concept of key hierarchies, uses more symmetric crypto and has APIs that make it easier to use the TPM as a Crypto Service Provider (i.e. general engine for crypto operations).
- Concepts in TPM1.2 like CMKs are replaced by a much more powerful and flexible policy framework that controls behavior for key duplication (duplication is now what was migration)
- TPM2.0 is designed so that crypto algorithms can be easily replaced. E.g. a “Chinese” TPM2.0 variant.

Four types of Authorizations

- **Physical presence**
  - Some command requires physical presence
  - Up to manufacturer to define how this is accomplished, e.g., bios setting
- **Password authorization**
  - Plaintext password (authValue)
  - When path is trusted, password is well known or caller is very resource constrained
  - Session is not needed (no nonces used), handle is fixed
- **HMAC authorizations**
  - Session very similar to those in TPM 1.2, but with some extra functionality
- **Policy**
  - Some specific properties need to be met in order to access an object

Authorization session, Policy

- Policy session is called ”enhanced authorization”
  - Add conditions under which an entity can be used, e.g.,
    - Certain PCR states
    - Time limit
    - ...

Policy assertions

- An entity has an authPolicy (in addition to authValue)
  - Combination of one or more assertions
- Can use combination of AND and OR, e.g.,

A & B | C & D

- A policyDigest is updated by commands corresponding to the authPolicy

$$\mathit{policyDigest}_{new} = \mathbf{H}\,(\mathit{policyDigest}_{old} \,||\, \text{"value"})$$

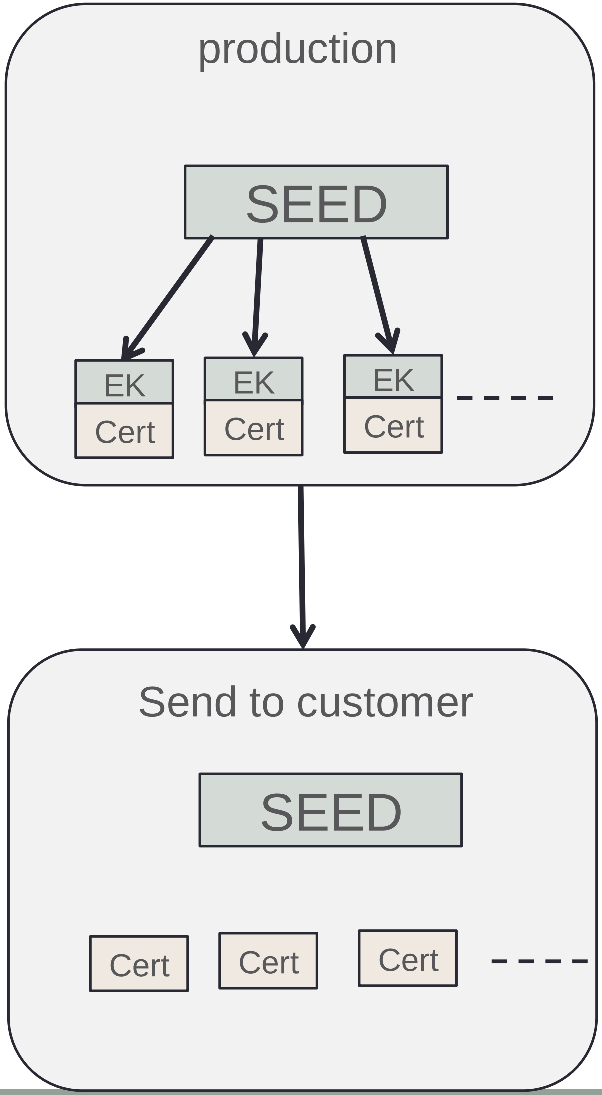
Policy session

- **TPM2_PolicyAuthValue()**
  - Require that command is authenticated with MAC using AuthValue as key
- **TPM2_PolicyPassword()**
  - Require that command is authenticated with password
- **TPM2_PolicyPCR()**
  - Extend policyDigest with hash of selected PCRs
- **TPM2_PolicyCpHash()**
  - Command parameters must have a given hash
  - PolicyDigest is extended with cpHash and cpHash is stored in policy session
- **TPM2_PolicySigned()**
  - Command parameters are signed. If signature is OK the policyDigest is extended
- Create policy hash before object is created and add resulting policyDigest to object
  - Force policy authorization needed (object attribute)

Mutually exclusive

Keys

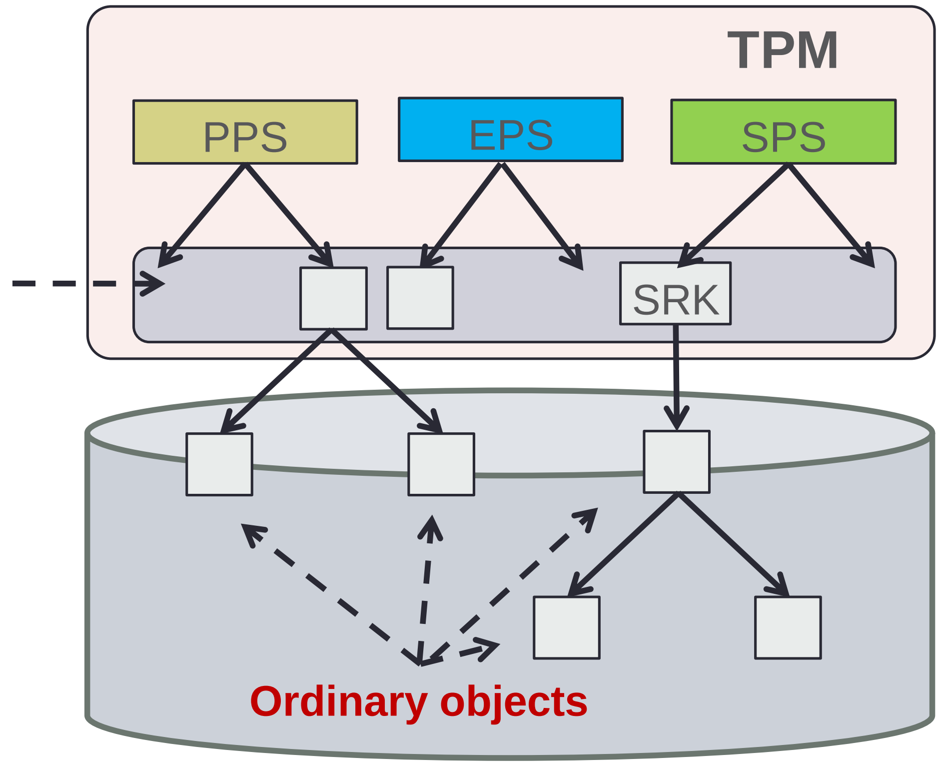
- TPM 2.0 structures has support for many algorithms
  - Symmetric (AES with 128/192/256 bit key, SM4 128 bit key)
  - Asymmetric (RSA 1024/2048 with different padding, ECDSA, SM2, ECDH, ...)
- What should EK and SRK be in TPM 2.0?
  - EK created by manufacturer (at least certificate so EK must be known to them)
  - EK used by TPM owner or platform owner
  - Generating all keys and store in TPM is unfeasible
- Problem solved using seeds that generate keys
  - Manufacturer can generate all possible keys and certificates and then throw away the keys
  - Owner can regenerate the key(s) that are needed



Key Hierarchy

- Three hierarchies defined in this way
- Three primary seeds
  - Platform primary seed: PPS
  - Endorsement primary seed: EPS
  - Storage primary seed: SPS

Primary objects



Stored outside TPM, e.g., hard-disk

- Seeds are used to derive symmetric keys
- Children are protected
  - Encryption using symmetric algorithms
    - Key derived from seedValue stored in parent's sensitive area
  - Integrity protection
    - HMAC

MATERIAL HERE AFTER IS NOT PART OF THE MANDATORY READING

Comparing keys – Key types

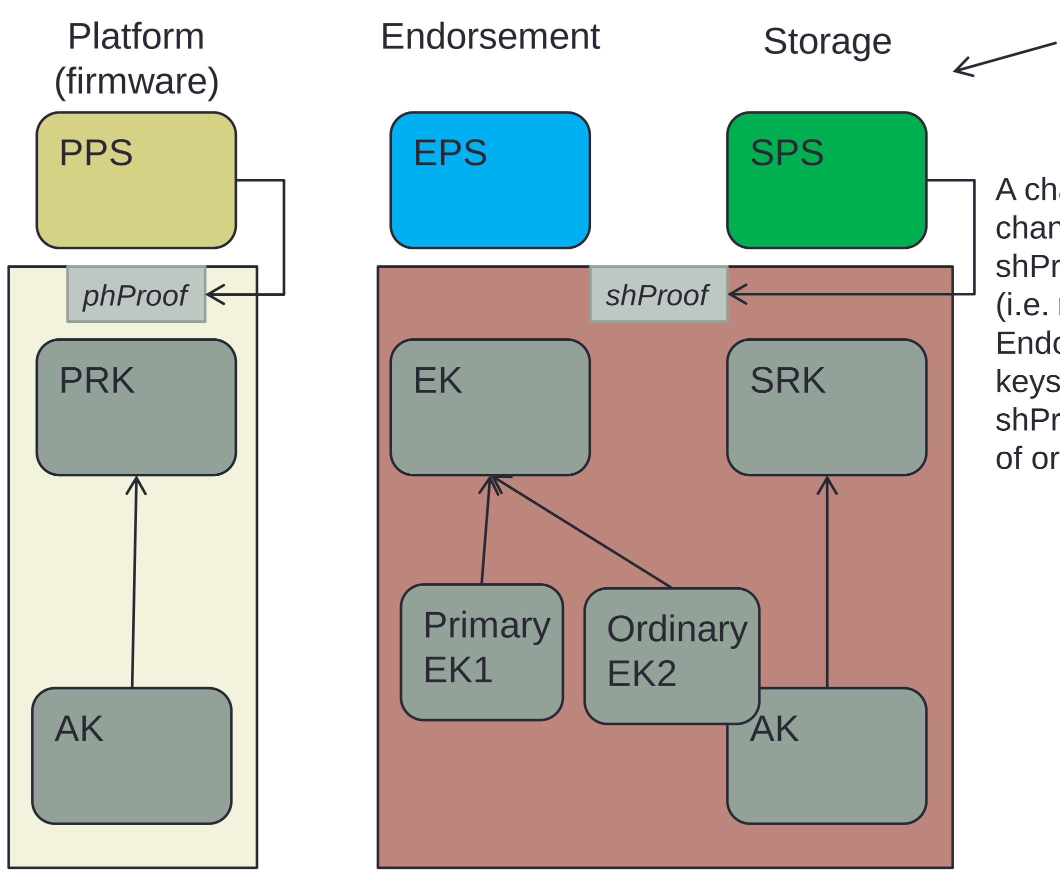
- Three attributes
  - Sign – key can be used to sign data (digital signature and HMAC)
  - Decrypt – can be used for decryption (symmetric and asymmetric)
  - Restricted – restrictions apply
    - Can only sign TPM generated data
    - Only decrypts certain structures (storage key)

sign	decrypt	restricted	comment
0	0	0	Data object, not a key
0	0	1	Not allowed
0	1	0	Can be used for encryption/decryption
0	1	1	Storage key
1	0	0	Can sign any data
1	0	1	Sign TPM generated hashes, internal data*, external data**
1	1	0	Can both sign and decrypt (but can not be storage key)
1	1	1	Currently not supported

* Will start with magic number
** If it does not start with magic number

KEY Hierarchies

hierarchy proofs



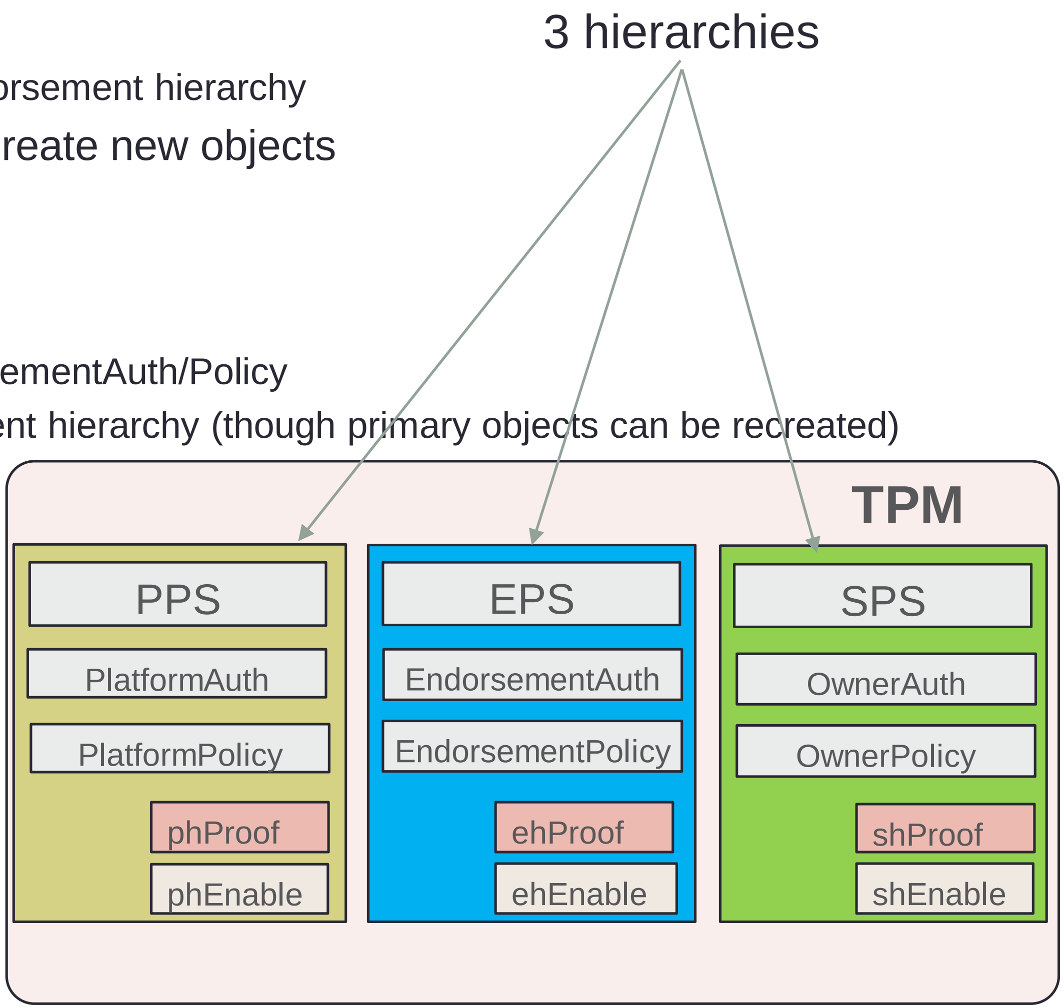
A change of SPS indicates a change of owner and changes shProof and invalidates ordinary (i.e. not primary) keys in the Endorsement hierarchy and all keys In the Storage hierarchy. shProof is include in the protection of ordinary objects

AKs created firmware/platform

AKs created by applications
(for the moment we do not consider these)

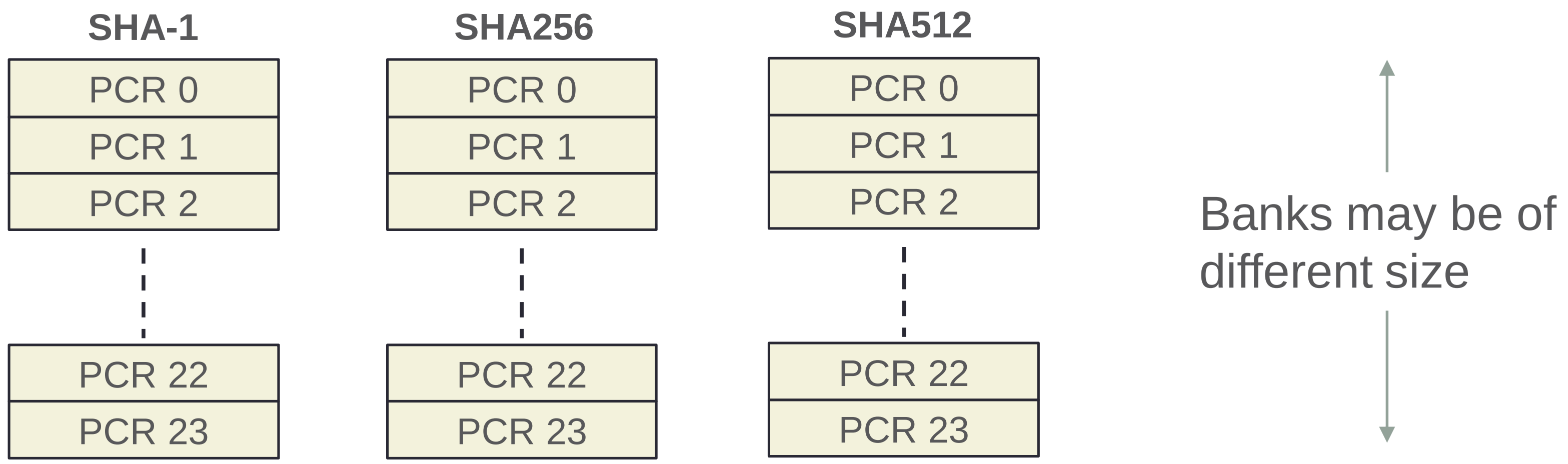
Ownership commands

- Ordinary objects bound to hierarchy using **proofs**
  - Proofs change when seed changes
  - shProof also used for objects in endorsement hierarchy
- Auth and policy values needed to create new objects
- **TPM2_Clear()** removes SPS
  - Need platformAuth/Policy
  - creates new shProof and ehProof
  - Clears ownerAuth/Policy and endorsementAuth/Policy
  - Also invalidates objects in Endorement hierarchy (though primary objects can be recreated)
- **TPM2_ChangeEPS()** creates new EPS
  - Need platformAuth/Policy
  - All EK certificates are invalidated
- **TPM2_ChangePPS()** creates new PPS and invalidates previous hierarchy
  - Need PlatformAuth/Policy



PCR register banks

- Support for 3 hash algorithms SHA-1, SHA256, and SHA512. Therefor there is one register bank for each supported algorithm



- Authorization required to change PCRs
  - PCRs can be grouped so that all in one group has same authorization value

PCR commands

Changing content of PCRs

- TPM2_PCR_Extend
  - **Input**: A set of hashes with corresponding algorithm, index to extend
    - Authentication with secret of PCR (EmptyAuth, AuthValue, AuthPolicy)
  - **Output**: -

At least 1024 bytes must be supported

- TPM2_PCR_Event
  - **Input**: Data to hash and extend with, index to extend
    - Authentication with secret of PCR (EmptyAuth, AuthValue, AuthPolicy)
  - **Output**: Computed hashes

- TPM2_PCR_Reset
  - **Input**: PCR index to reset
    - Authentication with secret of PCR (EmptyAuth, AuthValue, AuthPolicy)
  - **Output**: -

PCR commands

Get information from PCRs

- TPM2_PCR_Read (no authorization)
  - **Input**: set of PCR indexes and hash algorithms
  - **Output**: set of PCR values

- TPM2_Quote
  - **Input**: Key to sign with, nonce from server, signing scheme (if not stored with key), PCR set to qoute
    - Authentication with secret of signing key
  - **Output**: Signed data, Signature

Signing data

- Possible to both sign and verify signature
  - Both symmetric and asymmetric supported
  - Must be signing key – can be restricted

- **TPM2_Sign()** – Make a digital signature on a hash
  - Only asymmetric supported
  - Use HMAC for symmetric

- **TPM2_VerifySignature()**
  - Verifies digital signature using public key
  - Verifies HMAC using symmetric key
    - Restricted to checking HMAC on a hash

Encryption/Decryption

- The TPM can be used as a crypto service provider (CSP)
  - Asymmetric encryption/decryption
    - RSA
    - Elliptic curve Diffie-Hellman
  - Symmetric decryption/encryption
    - Choose from different block cipher modes (ECB, CBC, CTR, OFB, CFB
    - Choose IV
    - No specific algorithms are supported, but structures name some (AES SM4)
  - Hashing and HMAC

- This provides an equivalence to Binding in TPM 1.2

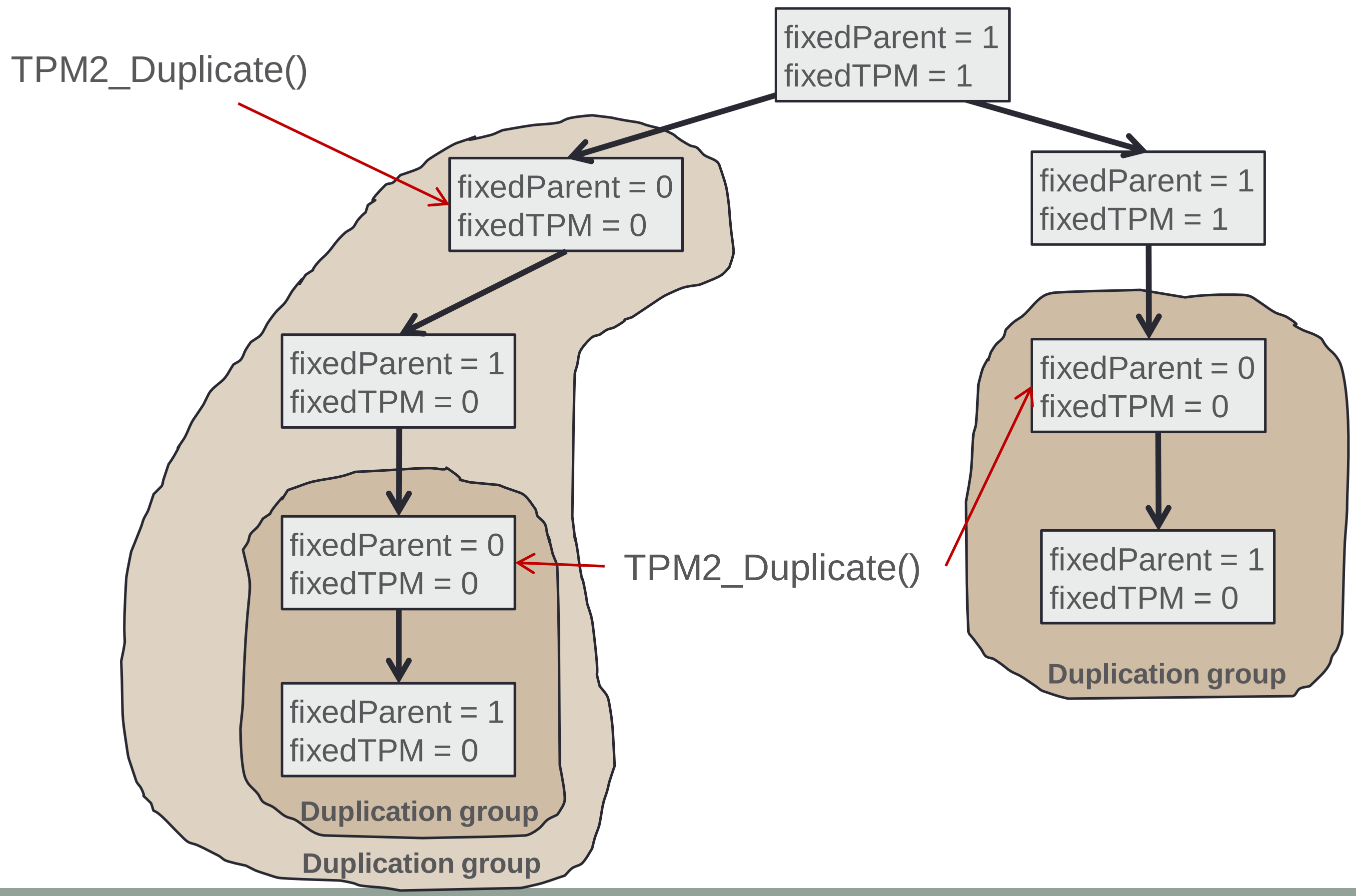
Sealing

- There is no sealing command – policies are used instead
  - Provides same functionality
  - Additionally provides much more functionality
- Seal in TPM 1.2 is same as policy

TPM2_PolicyPCR() & TPM2_PolicyAuthValue()

Protection group

- Root in a protection group does not have fixedParent SET



Duplication

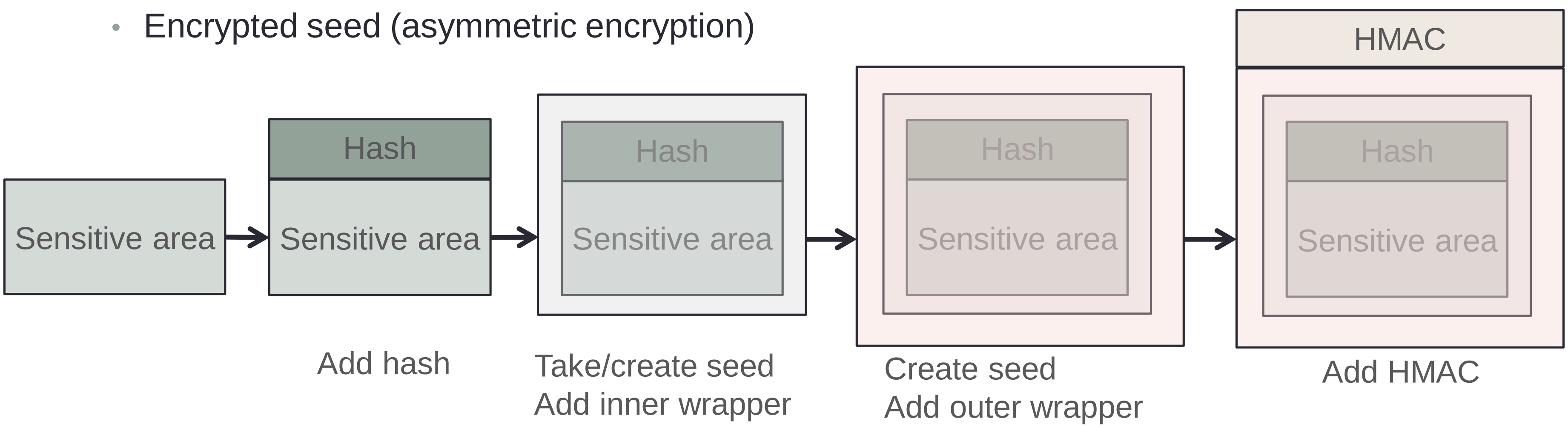
- Policy authorization is *required*
- Object creator sets duplication policy
  - Set name of destination, require extra authorization secret, etc

- Key is symmetrically encrypted
  - Inner wrapping (compare with r1 in TPM 1.2)
  - Outer wrapping (compare to encrypting with destination public key in TPM 1.2)
  - Seed to symmetric outer encryption key is asymmetrically encrypted with destination public key

- Both inner and outer wrapper are optional

Duplicate command

- **Input:**
  - Object to duplicate
  - New parent public key,
  - Inner wrapping key (can also be chosen by TPM)
  - Algorithm for inner wrapper
- **Output:**
  - Inner wrapping key if chosen by TPM
  - Encrypted blob (symmetric encryption)
  - Encrypted seed (asymmetric encryption)



Duplication in TPM 2.0

- TPM2_Import must be used to convert the key on the new TPM
  - Similar to convertMigrationBlob
- When creating key fixedParent and fixedTPM is used to determine duplication properties.

fixedParent	fixedTPM	
0	0	This combination represents a duplication root.
0	1	This combination is not allowed.
1	0	This combination indicates an object that is permanently in the protection group of its parent. It cannot be operated on by TPM2_Duplicate(). Not allowed for primary objects.
1	1	This combination indicates an object that was created on a specific TPM and no duplicate of the object is possible.