TPM - Trusted Platform Module

Martin Pirker <martin.pirker@iaik.tugraz.at>

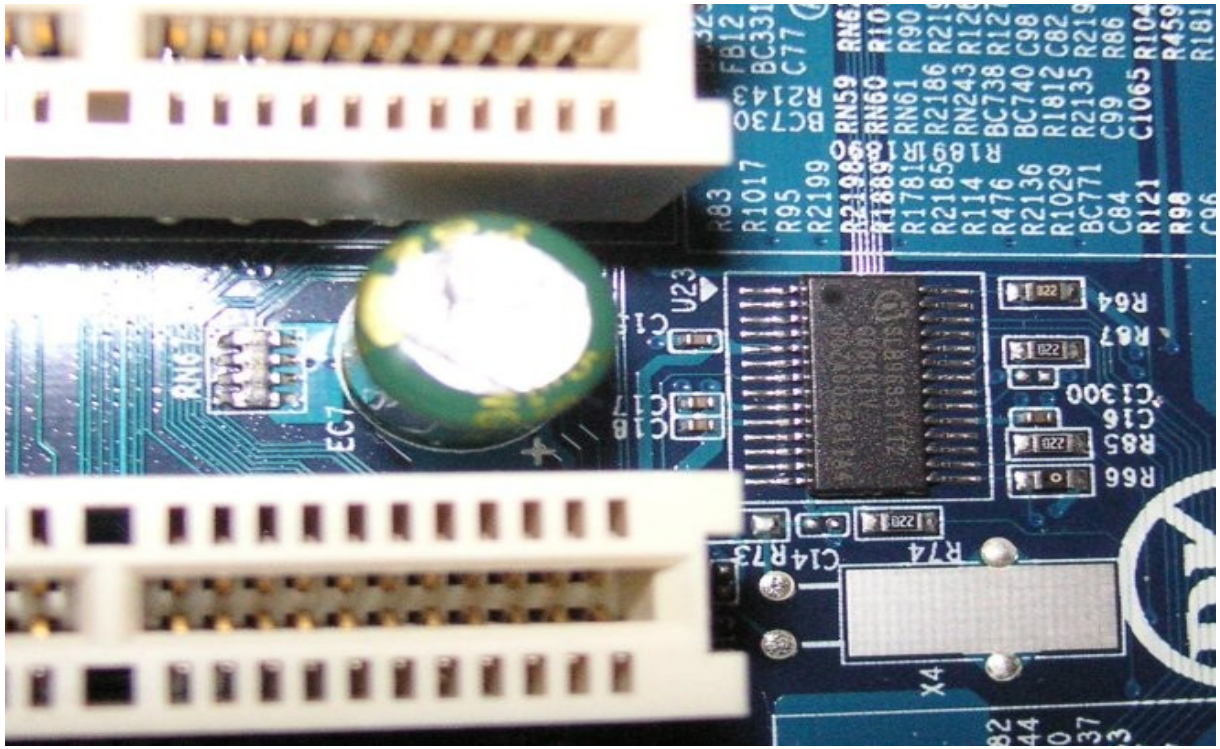
(orig.: Thomas Winkler <thomas.winkler@iaik.tugraz.at>)

Overview

- Motivation
- TC and TPM Design Goals
- TPM Overview and TPM Specification
- TPM Internals
- Taking and Clearing TPM Ownership
- TPM Keys, Key Creation and Key Hierarchy
- Roots of Trust (RTM, RTR, RTS)
- Attestation
- Low-level TPM Interaction
- Advanced TPM Concepts

Motivation

- hundred millions of PCs, laptops, ... shipped with a TPM
- many misconceptions and misunderstandings
- What have you heard about TPMs?
  - Do you have a machine with a TPM?
  - Are you using it?
  - What do you think it does?
  - What do you think it can be used for?



TC and TPM Design Goals

- A trusted system is one that “behaves in the expected manner for a particular purpose” (TCG).
  - How can that be achieved? Whom can you trust? The OS?
  - Would you trust your OS to be unmodified? If so - why?
  - If you don't know for sure that your OS really is in the state it claims to be - why trust any software running on top of it?
- How can one report the system state to a third party?
- Assumption: Much easier to manipulate software than hardware.
  - Idea of TC: implement trust anchor in hardware -> TPM
- TPMs are designed to be relatively low cost devices
  - limited resistance against sophisticated hardware attacks
  - not perfect security but better than pure software solution

Fundamental TC Functionality

- a mechanism to record (measure) what software is/was running
  - requires to monitor the boot process
  - needs an anchor to start the measurement from: a Root of Trust
  - nobody should be able to modify or forge these measurements
  - some shielded location for the measurements is required
- now you know that your platform is in a defined state
  - Why should someone else believe this claim?
  - A mechanism to securely report the measurements to a 3rd party is required.
  - The process of securely reporting the platform state is called “attestation”.
- secure storage
  - allow access to data only if system is in a known state

TPM Overview

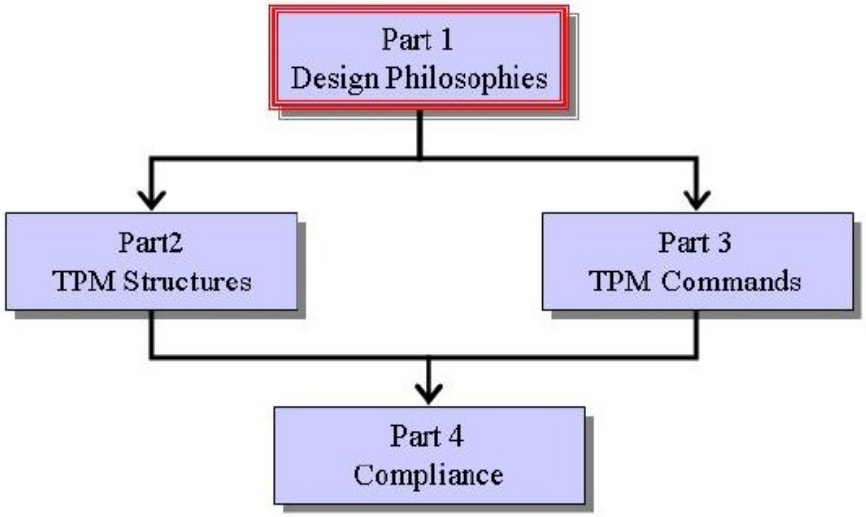
- ancestors of todays TPMs showed up in IBM ThinkPads (called Embedded Security Chip/Embedded Security Solution 1.0)
- TPMs have resemblance with SmartCard technologies
- standardization by TCG (formerly TCPA)
- open industry standard
- specs freely available from
  https://www.trustedcomputinggroup.org/specs/
- TPM specification 1.2 (versions 1.1b obsolete)
- TCG keeps working on new / improved versions
- TPM is a major building block to achieve the goals of a TC system

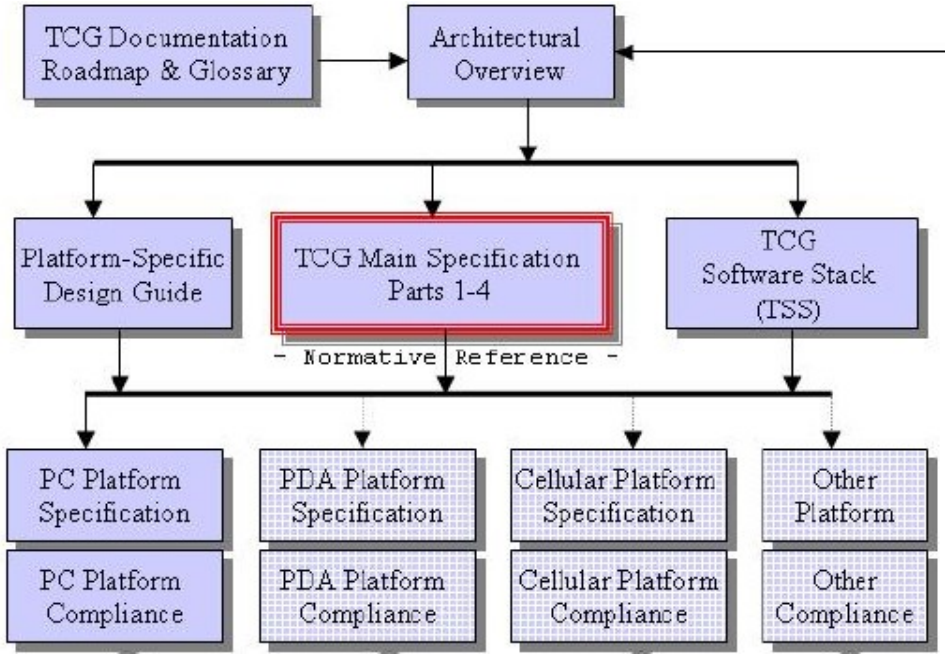
TPM Overview

- TPM spec is not targeted towards a specific device
- other specs such as the PC Client or Mobile Phone Spec "customize" the TPM spec for a specific device
- TPM Manufacturers (in no particular order)
  - Infineon
  - Intel
  - Atmel
  - ST Microelectronics
  - Winbond (bought business unit from National Semiconductors)
  - Broadcom
  - SinoSun (China)

TPM Specification

- TCG TPM spec for 1.2 consists of 4 parts (700+ pages total)
  - Part 1: Design Principles Part 2: Structures and Constants
    Part 3: Commands Part 4: Compliance
  - all structures and commands are specified in C like syntax
  - additional platform specific spec (e.g. PC)
  - spec says nothing about internal TPM design or speed requirements



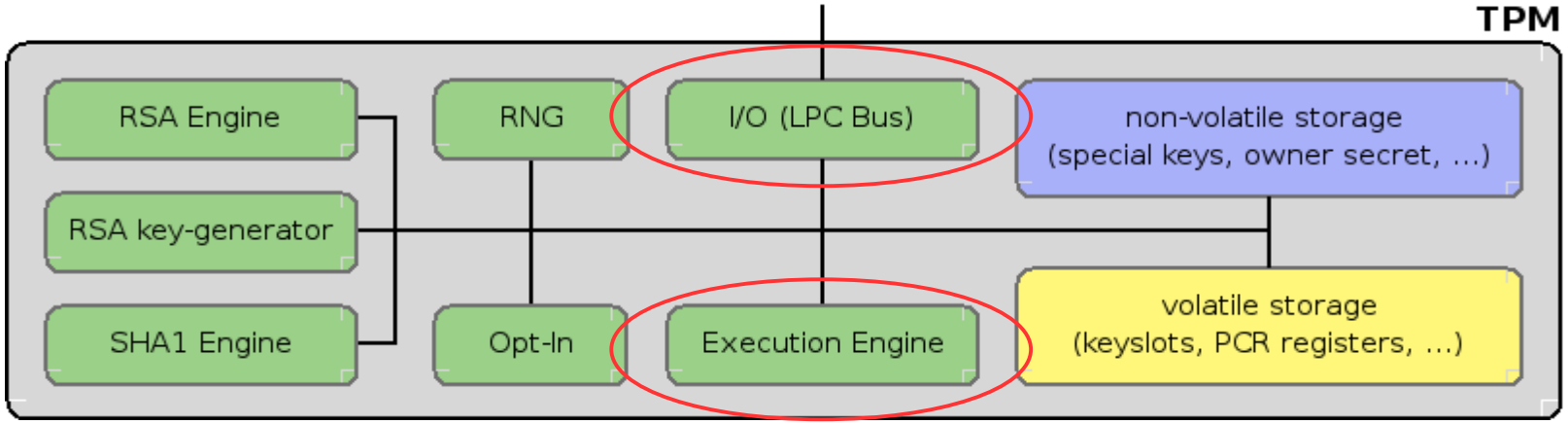


TPM Internals

- shielded location: an area where data is protected against interference from the outside and exposure
- protected capability: a function whose correct operation is necessary in order for the operation of the TCG subsystem to be trusted
- only protected capabilities operate on data in shielded locations
- both, shielded locations and protected capabilities are implemented in hardware and therefore resistant against software attacks
- TPM is a slave device that does not actively initiate communication
- TPM has no access to any system resources (LPC bus limitation)
- consequence: TPM can not alter execution flow of system (e.g. booting, execution of applications, ...)

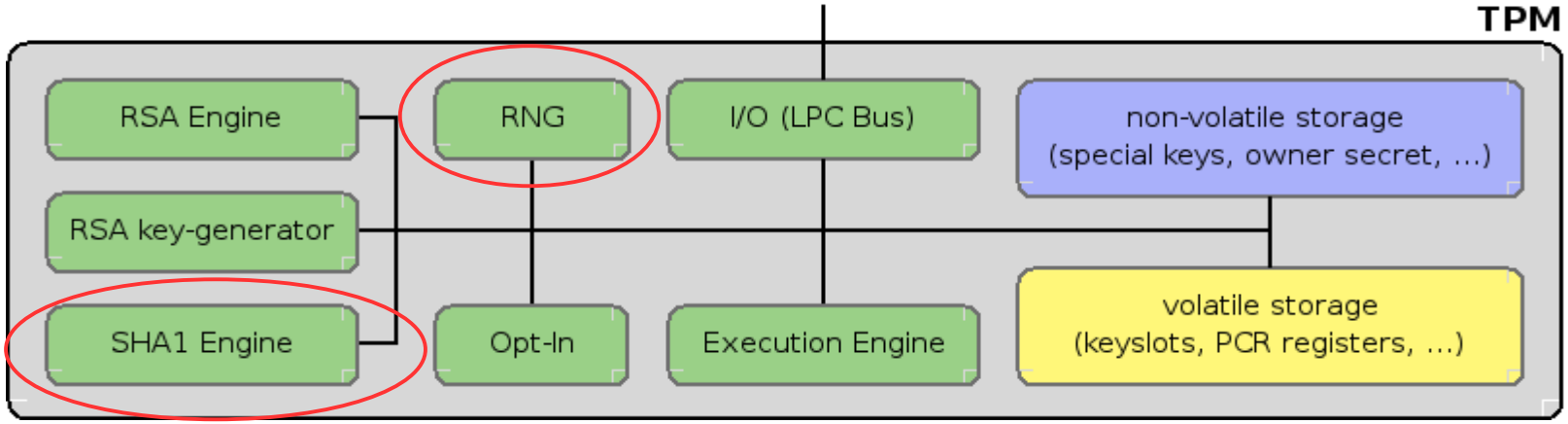
TPM Internals

- I/O
  - manages information flow over the communications bus
  - typically LPC - Low PinCount Bus, SM-Bus
- Execution Engine
  - command verification and parsing
  - execution of the appropriate command code
  - controls internal TPM execution flow
  - micro-controller



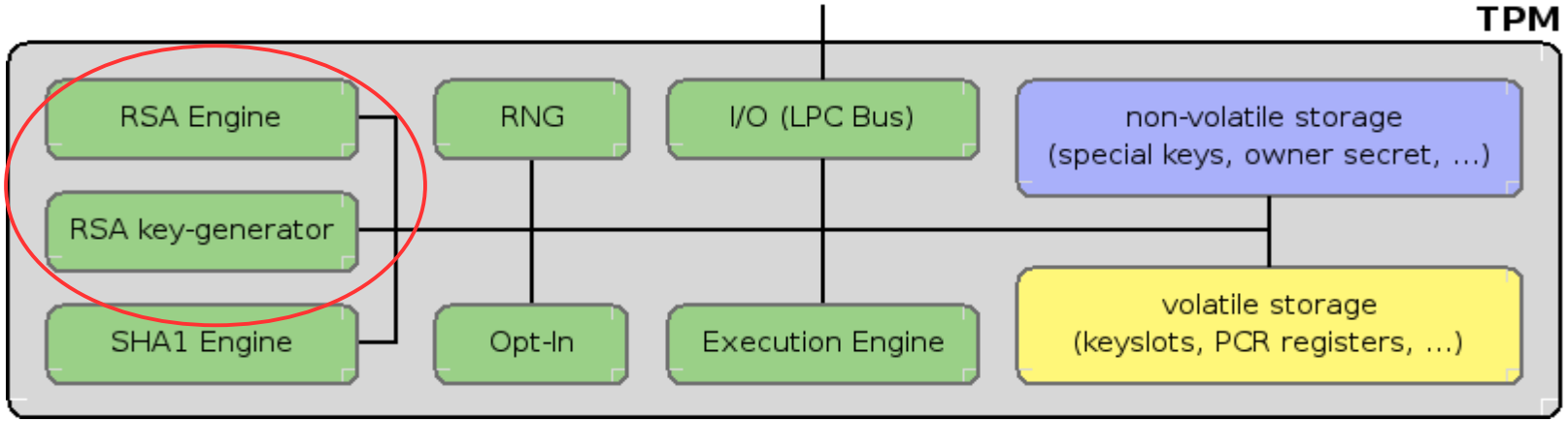
TPM Internals

- SHA-1 Engine (160 bits)
  - primarily used by the TPM as its trusted hash algorithm
  - exposed to the outside to be used in the boot process
  - TPM is not a crypto accelerator
  - future TPM revisions likely to add additional hash functions
- RNG
  - source of randomness in the TPM
  - used for nonces (Number Used Once), key generation, ...



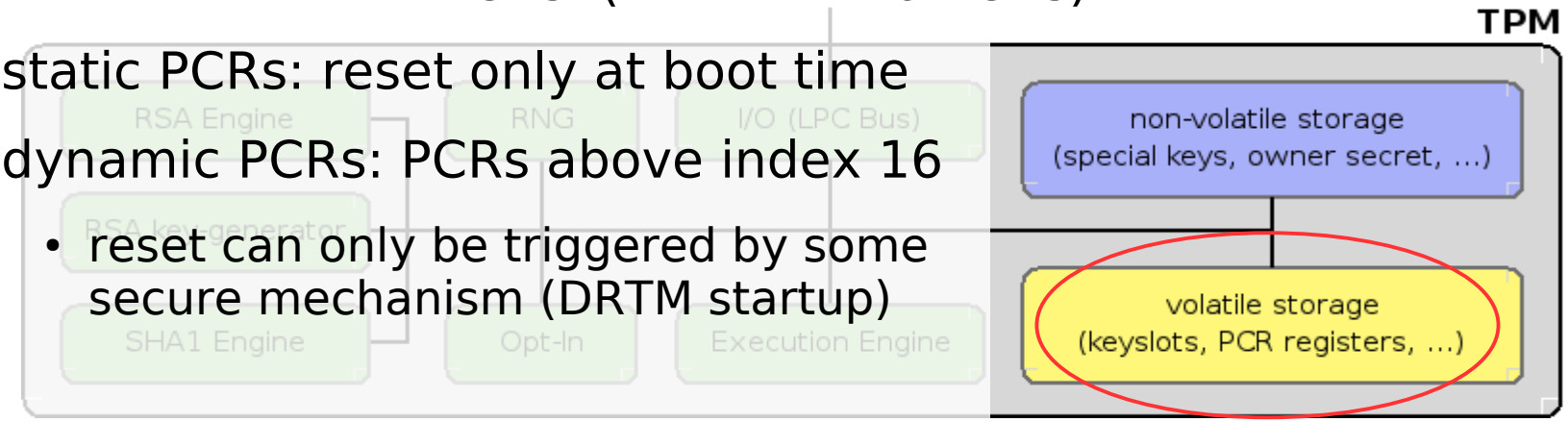
TPM Internals - RSA

- RSA Engine and Key Generator
  - asymmetric key generation (RSA; storage and AIK key size >= 2048)
  - must support 512, 1024, 2048 bit keys
  - use of 2048 recommended
  - RSA public exponent is fixed to 2^16 + 1
  - asymmetric encryption/decryption (RSA)
  - optionally may implement DSA, ECC (but no known implementation)
  - to use an RSA key it has to be loaded into the TPM



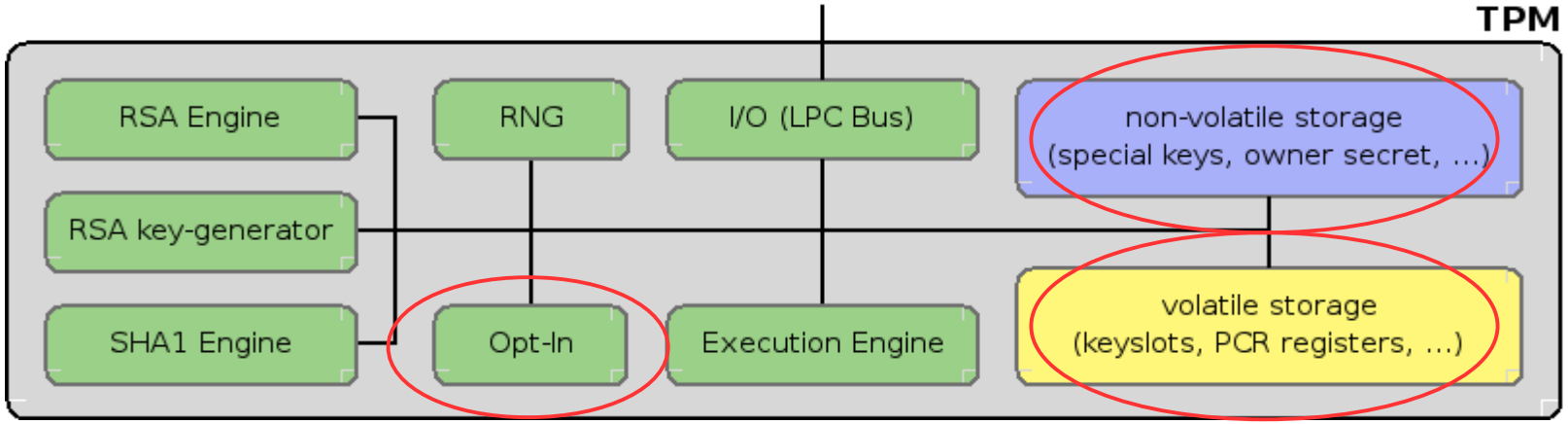
TPM Internals

- Platform Configuration Registers (PCRs)
  - PCR is a 160 bit storage location for integrity measurements
  - shielded location inside TPM
  - values are not set or overwritten but extended
    PCR[i] = SHA-1(PCR[i] || newMeasurement)
  - PCR extends are not commutative (i.e. measuring A then B does not result in the same PCR value as measuring B then A)
  - PCRs can keep track of unlimited number of measurements
    - TPM 1.2: 24 PCRs (TPM 1.1: 16 PCRs)
  - static PCRs: reset only at boot time
  - dynamic PCRs: PCRs above index 16
    - reset can only be triggered by some secure mechanism (DRTM startup)



TPM Internals

- Volatile Memory
  - key slots: to use a TPM key it has to be loaded into the TPM (remember: protected capabilities operate on shielded locations)
- Non-Volatile Memory
  - special keys (Endorsement Key (EK) and Storage Root Key (SRK), owner secret, ...
  - Endorsement Key Certificate (details follow in later lecture)
- Opt-In: Platform owner can decide whether or not he wants to make use of the TPM



TPM Internals

- Power Detection
  - TPM must be notified of all power changes
  - at some point in POST TPM gets notified that actions that require “physical presence” have to be disabled
- Symmetric Encryption for secrets
  - XOR with emphemeral session secret
- HMAC Engine
  - keyed hash function
  - proof of authentication data knowledge (secret = HMAC key)
  - proof that received command was not modified (integrity)

Taking Ownership of a TPM

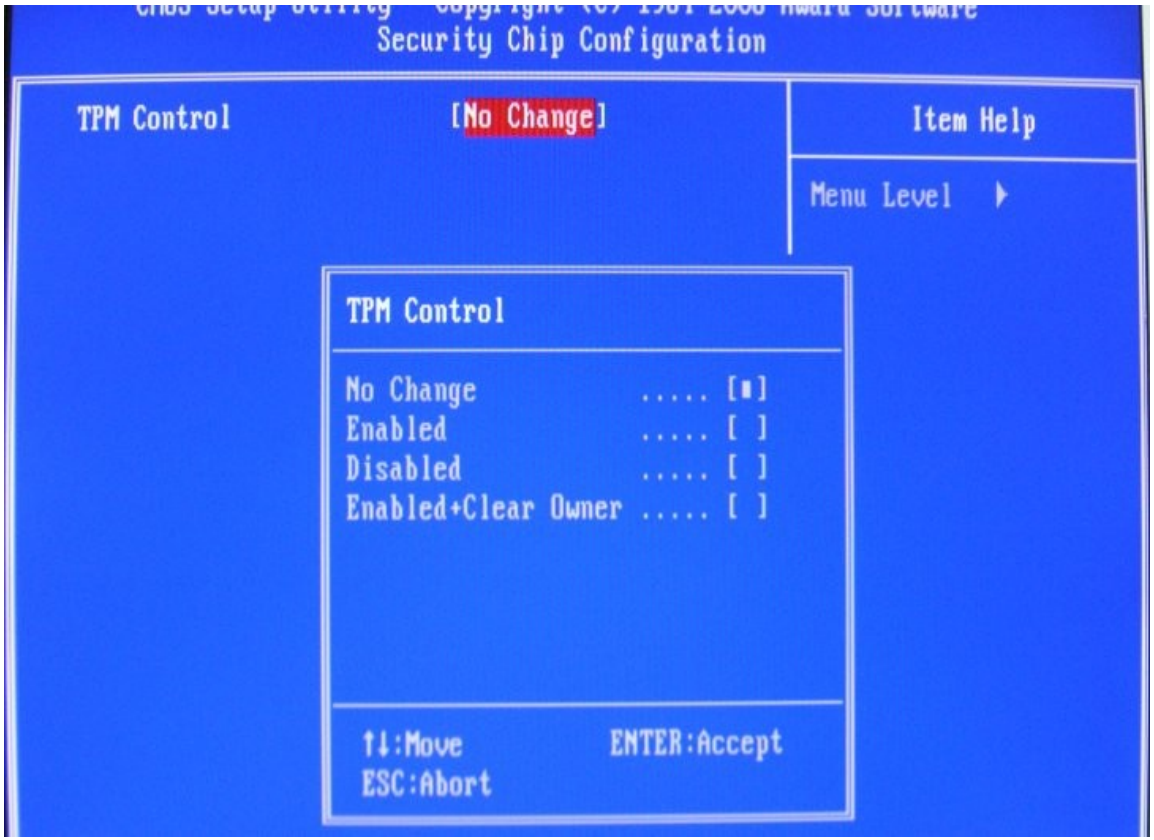
- TPM is shipped in "unowned" state
- to make proper use of TPM, platform owner has to execute "TakeOwnership" operation
- setting owner password - inserting a shared secret into the TPM (stored in shielded location; non volatile memory)
- certain TPM operations require owner authorization
- “physical presence” allows access to certain (otherwise owner protected) TPM functionality; does not reveal any TPM secrets
  - ForceClear allows to "clear" the TPM using physical presence
- Storage Root Key (SRK) is created as part of TakeOwnership
- (private) SRK is stored inside the TPM and never leaves it
- password required for SRK usage can be set

Clearing a TPM

- resetting the TPM to the factory defaults
- clearing requires owner secret or physical presence (ForceClear)
- there are no mechanisms to recover a lost TPM owner password
- tasks executed when clearing the TPM
  - invalidation of the SRK and thereby all data protected by the SRK (there simply is no way to decrypt the data blobs anymore)
  - invalidation of the TPM owner authorization value
  - reset of volatile and non-volatile memory to factory defaults
  - EK is NOT affected
  - PCR values are undefined after clear (reboot required)
- ForceClear is only available during boot (and disabled thereafter)
- OwnerClear can also be disabled (permanent; ForceClear required)

TPM BIOS control

- TCG compliant BIOS allows to ForceClear a TPM
- TCG compliant BIOS allows to change TPM status (disabled/enabled)



TPM Keys

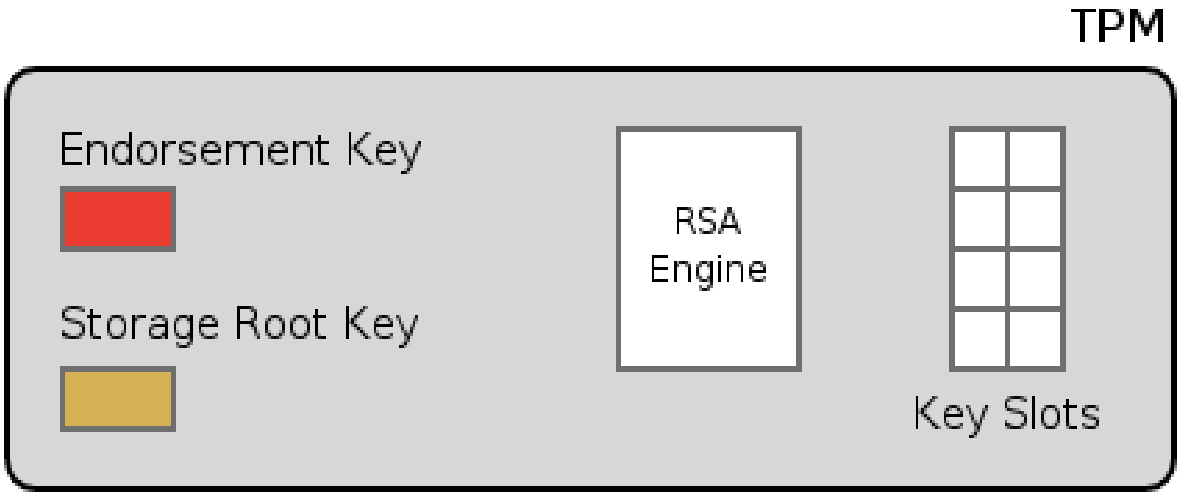
- Endorsement key (EK)
  - unique platform identity
  - details are discussed later in the context of AIKs and Attestation
- Storage Root Key (SRK)
  - 2048 bit RSA key
  - is top level element of TPM key hierarchy
- Storage Keys
  - RSA keys used to wrap (encrypt) other elements in the TPM key hierarchy
- Signature Keys
  - RSA keys used for signing operations
  - must be a leaf in the TPM key hierarchy

TPM Keys

- Binding Keys
  - key used for binding operations (TPM_Bind, TPM_Unbind)
- Legacy Keys
  - can be used for both, signing and binding
  - usage of this type of key is not recommended (use keys tagged for the specific purpose instead)
- Protection of Keys
  - usage secret: has to be provided for all operations that make use of a TPM key
  - migration secret: has to be provided when migrating a key between different platforms

Creating TPM Keys

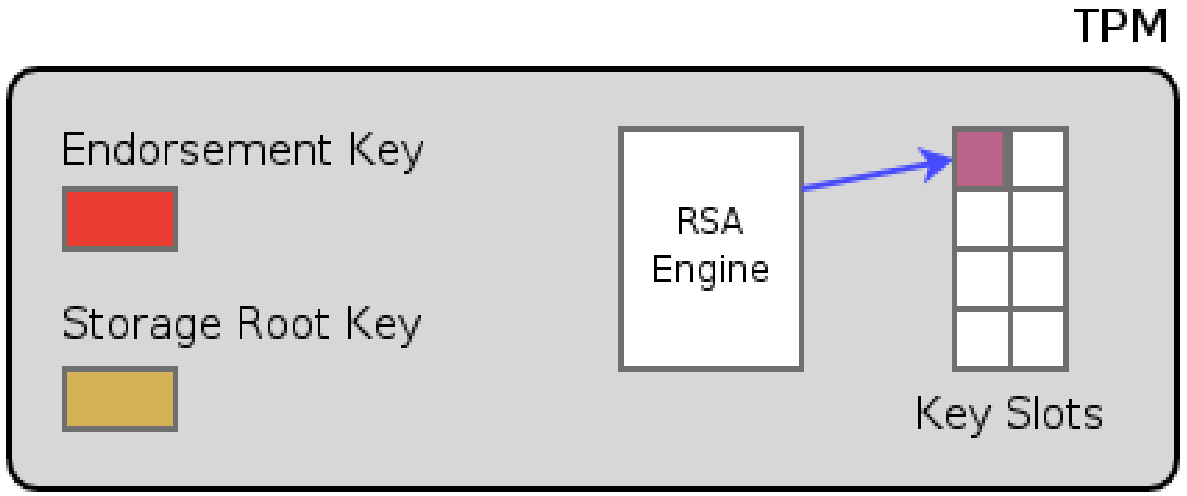
- Endorsement Key and Storage Root Key are the only keys permanently stored inside the TPM
- TPM keys are generated inside the TPM
- to use a TPM key, it has to be loaded into the TPM



- amount of TPM key slots is limited
- management of key slots is done in software (TSS)

Creating TPM Keys

- RSA Engine creates a new RSA key
- to create a key pair, a parent key has to be specified
- amount of key slots inside the TPM is limited -> a mechanism is required to (securely) swap out keys from the TPM

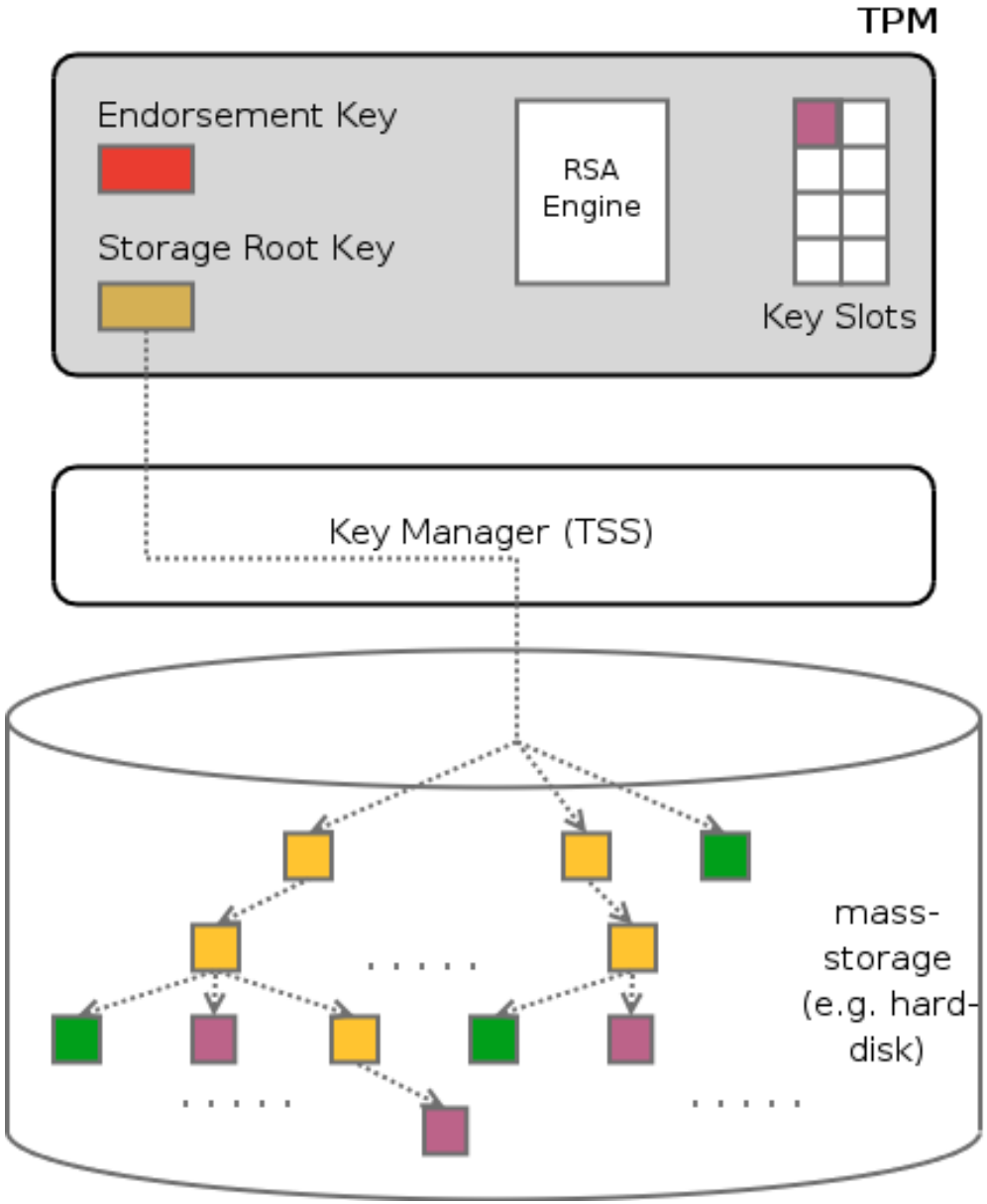


- if a key is exported from the TPM its private part never leaves the TPM in plain: it is encrypted with its public parent key

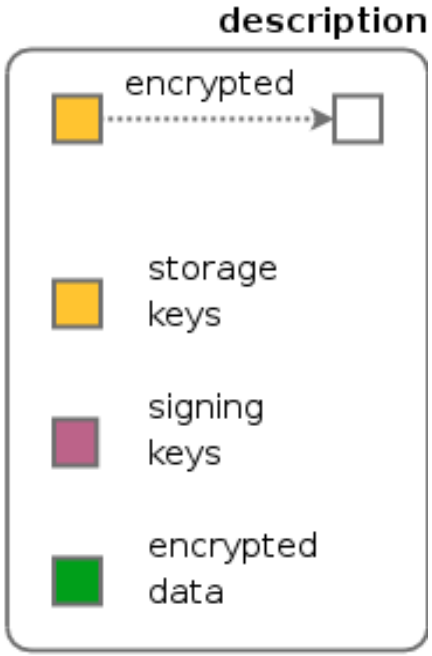
TPM Key Hierarchy

- When moving out keys from a TPM a key hierarchy is established.
- Whenever a key is exported from the TPM, its private part is encrypted using the public key of the parent.
  In TCG terminology the child key is wrapped using the parent key.
- Since the parents private key (required to load/decrypt the child key) never leaves the TPM in plain, the private key of a TPM can never be decrypted/used outside of the TPM.
- The private SRK, sitting at the top level of the key hierarchy, is never exported from the TPM.
- Storage keys form the nodes of the key hierarchy while signing keys always are leaves.

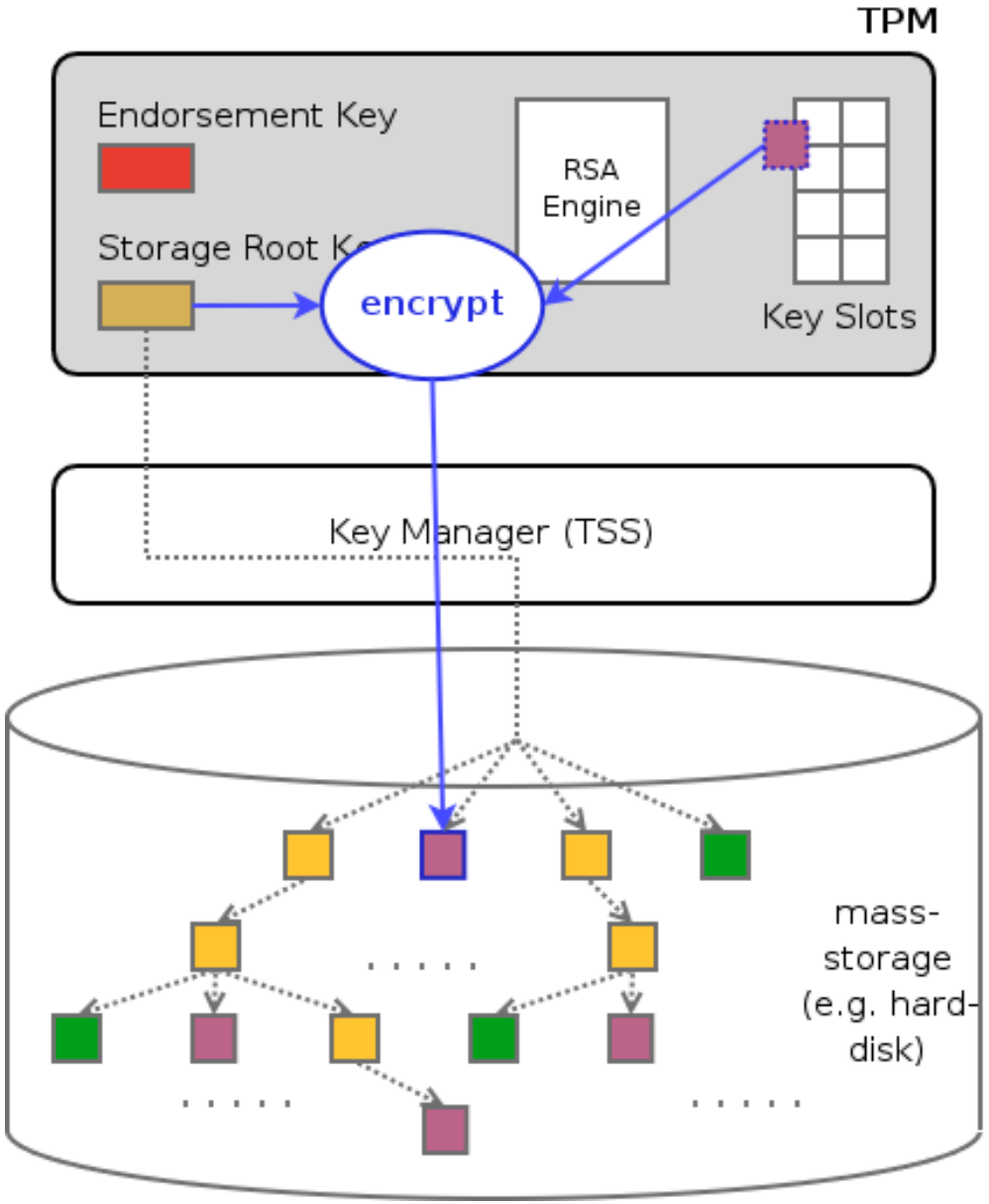
TPM Key Hierarchy



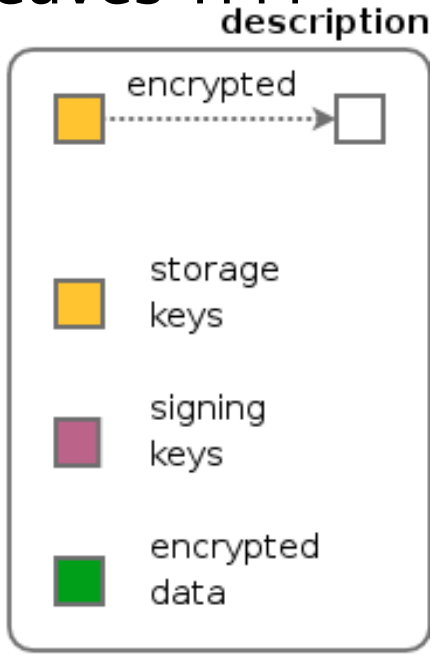
- key hierarchy with SRK as root
- private SRK never leaves the TPM



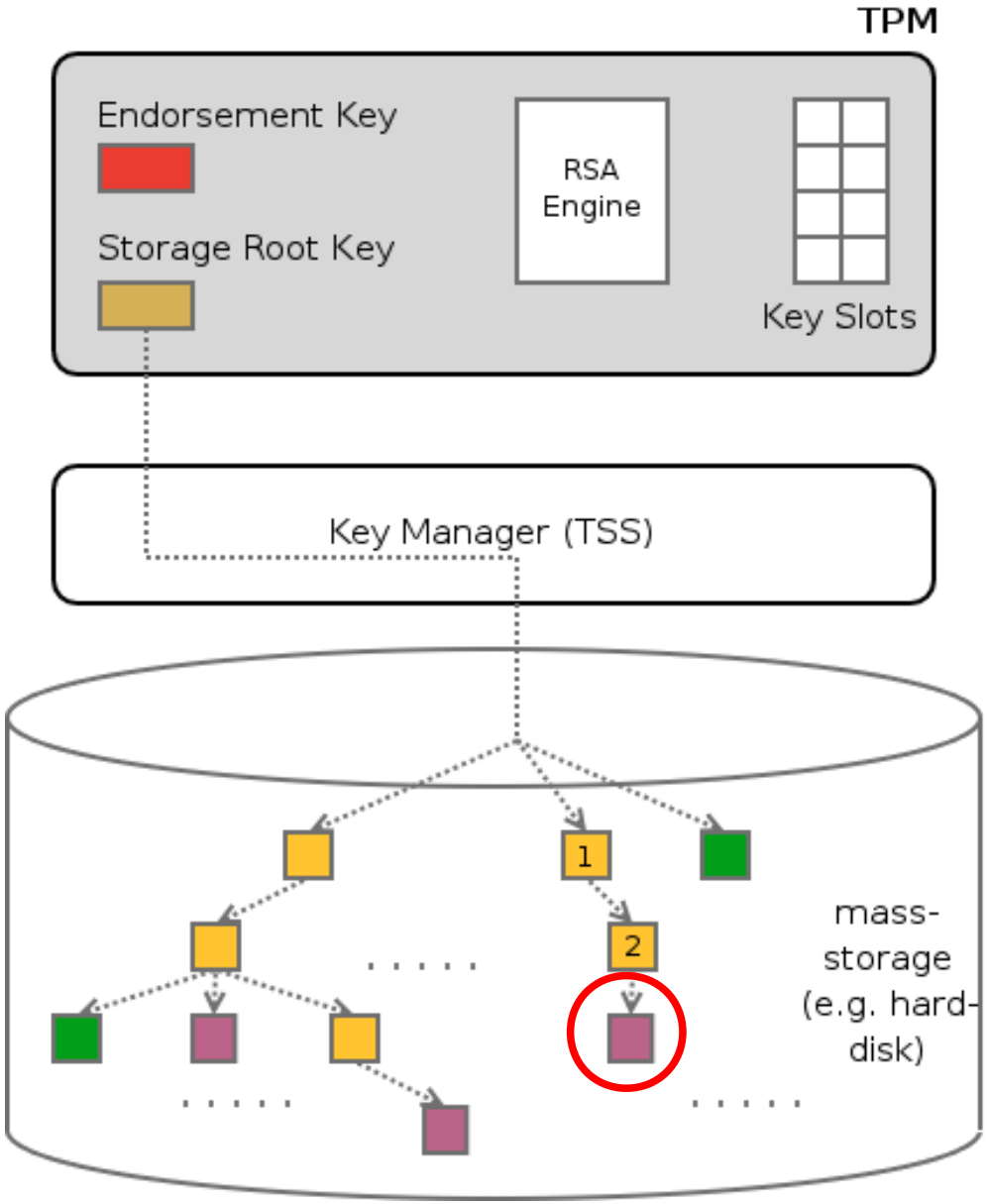
Unloading TPM Keys



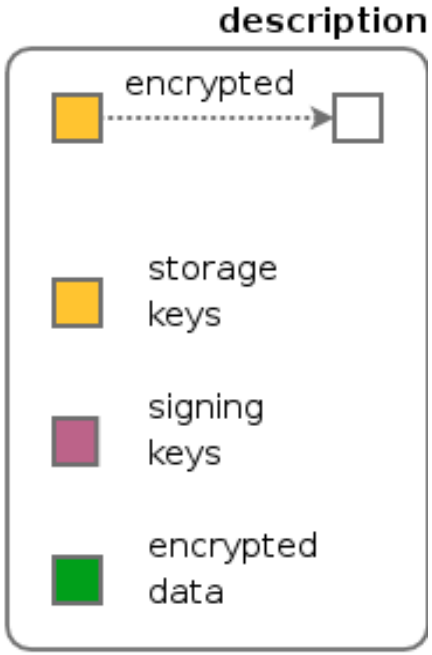
- exporting key blob from TPM
- private part is encrypted with public parent key before key blob leaves TPM



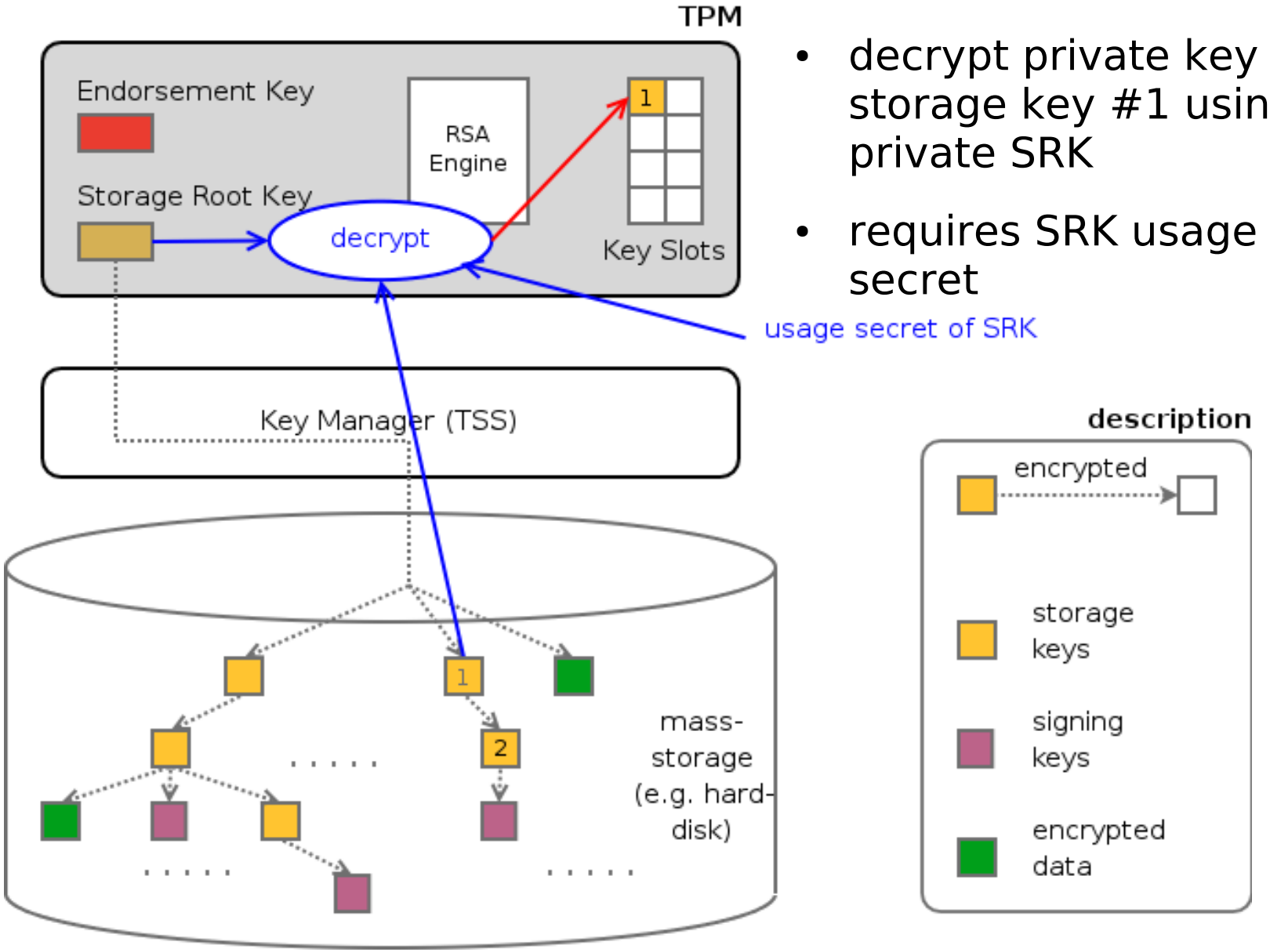
Loading TPM Keys



- task: load signing key into TPM to use it for signing operation
- establish entire key chain up to SRK

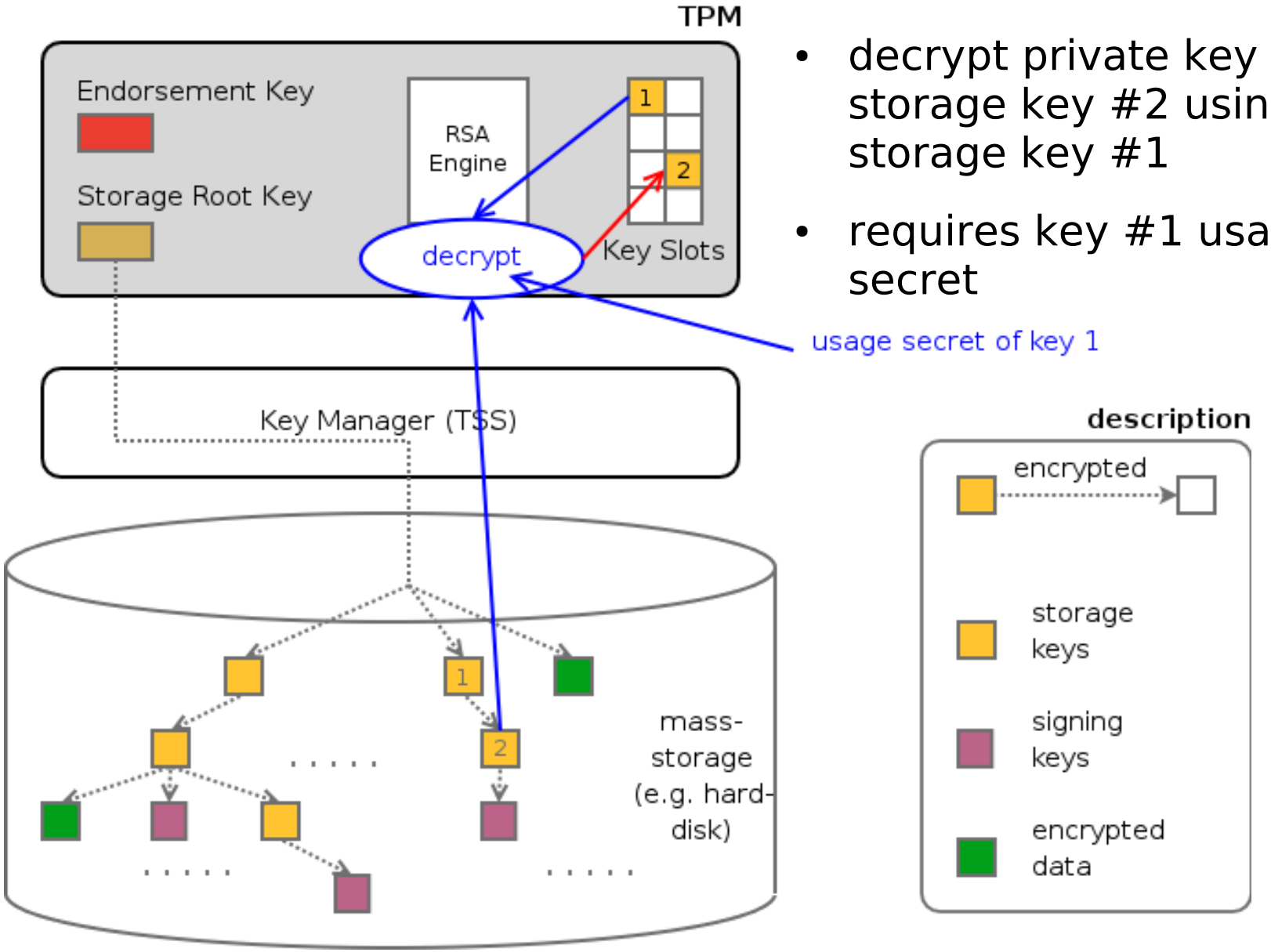


Loading TPM Keys



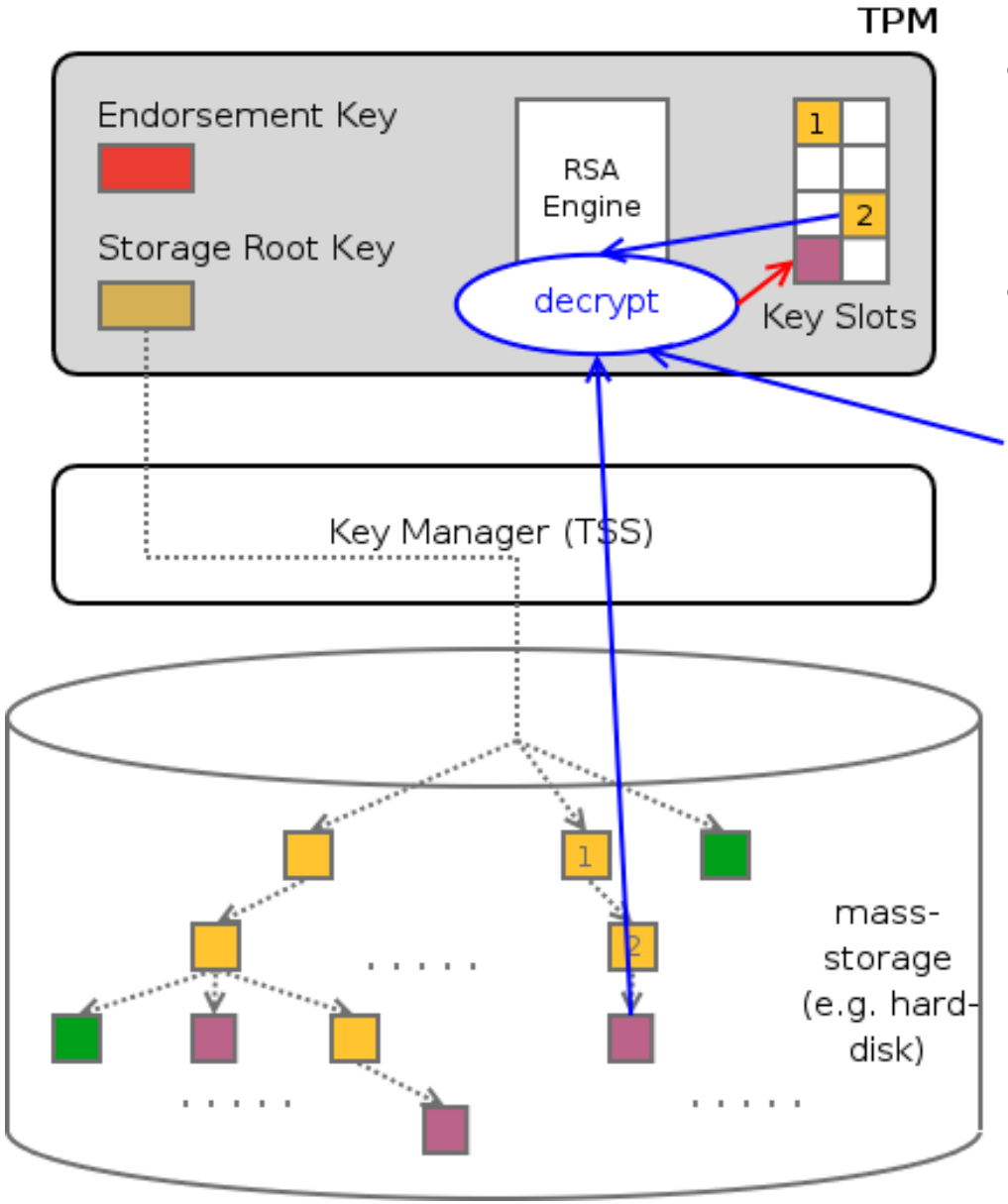
- decrypt private key of storage key #1 using the private SRK
- requires SRK usage secret

Loading TPM Keys

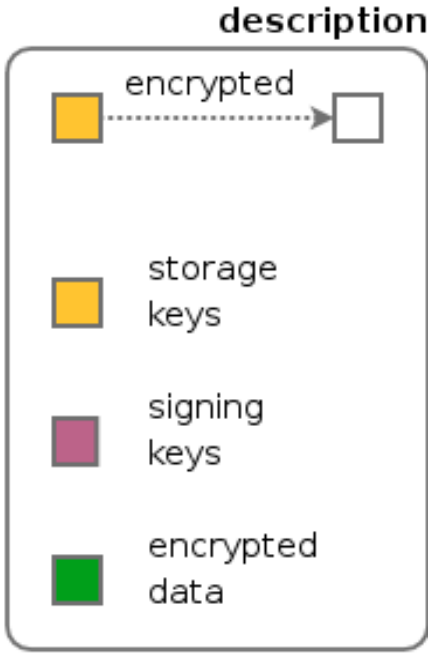


- decrypt private key of storage key #2 using the storage key #1
- requires key #1 usage secret

Loading TPM Keys



- decrypt private key of signing key using the storage key #2
- requires key #2 usage secret



Roots of Trust

- Root of Trust: “a hardware or software mechanism that one implicitly trusts”
- Root of Trust for Measurement (RTM)
  - uses PCRs to record the state of a system
  - static entity like the PC BIOS or dynamic entity (e.g. LT)
- Root of Trust for Reporting (RTR)
  - entity trusted to report information accurately and correctly
  - uses PCRs and RSA signatures to report the platform state to external parties in an unforgeable way
- Root of Trust for Storage (RTS)
  - entity trusted to store information without interference leakage
  - uses PCRs and RSA encryption to protect data and ensure that data can only be accessed if platform is in a known state

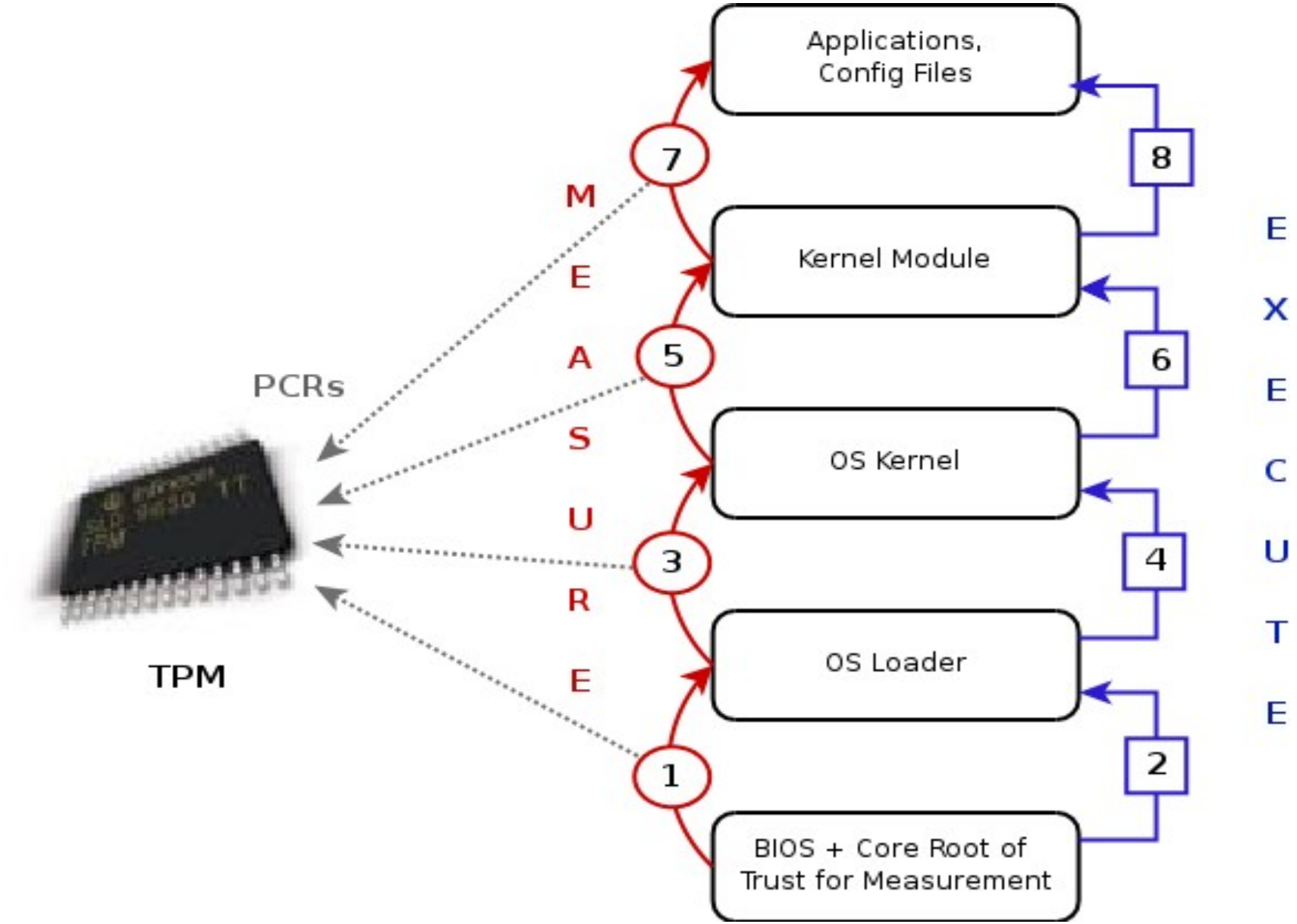
Root of Trust for Measurement

- goal: measure system state into PCRs
- using PCRs a communication party can be convinced that the system is in some known state
- system users are NOT prevented from running any software they want
- but the execution is logged and can not be denied
- problem: some anchor is needed to start the measurements from -> Root of Trust for Measurement
- static RTM: part of the BIOS
- dynamic RTM: provided by technologies such as LaGrande (Intel) or Pacifica (AMD)

Root of Trust for Measurement

- From the RTM the trust is extended to other system components. This concept is called transitive trust.
- involved steps:
  - measure (compute the hash value of) the next entity: e.g. the BIOS measures the OS loader
  - the measurement is extended into one of the TPMs PCRs
  - control is passes to the measured entity
- This process is continued for all components of a system up to user level applications.
- PC client specifications defines which PCRs are used for what
- note 1: measurements change with system updates and patches
- note 2: number of measurements can get very large (already for average systems)

Chain of Trust



PCR Event Log

- Together with PCR extensions also PCR event log entries can be made.
- A log entry contains the PCR number, the value that was extended into the PCR and a log message (giving details what was measured).
- The event log does not need to be protected by the TPM and therefore is managed on external mass storage (managed by TSS).
- The event log can be used to validate the individual steps that lead to the current PCR value.
  - calculate the extends in software starting at the beginning of the log
  - compare the result to the PCR value in the TPM
  - if the values match the verifier has assurance that the log was not tampered with

Measurement Log

- Example: Integrity Measurement Architecture (IMA) on Linux boot

	Measurement Value (fingerprint == SHA1)	Measurement Hook	File Name	
#000	9797EDF8D0EED36B1CF92547816051C8AF4E45EE	ima-init	boot-aggregate	Aggregate
#001	F7A0BF5A67CE98BC06316F77CA1F404A2D447534	mmap-file	init	Executable
#002	38C5D31E5DAD3F1B012FDD35B4E011E783CE6FD8	mmap-file	ld-2.3.2.so	Library
#003	42F7960321992201671388B8AAFC9E37F6936B226	mmap-file	libc-2.3.2.so	Library
#004	A4DC5EDF06698646CD76916F16E95C37E55DC12B	mmap-file	bash	Executable
#005	F4F6CB0ACC2F1BEE13D60330011DF926D24E5688	mmap-file	libtermcap.so.2.0.8	Library
#006	AE1BC1746AFD2AC1ECD1D9EEEAEBD125A6A9EB8D	mmap-file	libdl-2.3.2.so	Library
#007	CFBC7EC3302145AB78A307C0D41DBB9A4251377B	mmap-file	libnss_files-2.3.2.so	Library
#008	805572455CF5BF50A7EE42E3CC6B0EDA65AF17A4	mmap-file	initlog	Executable
#009	C95CBC5625719649103E0D1C3595967474842F7B	mmap-file	hostname	Executable
#010	0CAA342424F420FF29B7FB2FCF278F973600681B	mmap-file	mount	Executable
#011	5E45D898530F31BADEF5E247EBCF4AB57A795366	bash-source	functions	Bash Source
#012	A253AF3AB981711A13AE45D6B46462386E628076	mmap-file	consoletype	Executable
#013	2E37B839BC4EC1B6BE1BDF5BACD1E7B56567D8D9	bash-source	i18n	Bash Source
#014	C9D1B3E2CD0995E16AE6DD98B388FD873324740D	bash-source	init	Bash Source
#015	590F75EE97E0FC560F07FCB07A8646FADEC88C2A	mmap-file	uname	Executable
#016	5E851EFA4601B3AFCA9EAE75ED53688606630BFA	mmap-file	grep	Executable
#017	32798F58C4F1B4CD017B09BCAAF2A22D345E7E4F	mmap-file	sed	Executable
#018	CE516DE1DF0CD230F4A1D34EFC89491CAF3D50E4	mmap-file	libpcre.so.0.0.1	Library
#019	22EAF1B6009B23150367F465694AC63314866558	bash-script	setsysfont	Bash Comma
#020	8B15F3556E892176B03D775E590F8ADF9DA727C5	bash-script	unicode_start	Bash Comma
#021	A4C5F9D457DA16E47768423A68F135259F7180D7	mmap-file	kbd_mode	Executable
#022	497ED7F80C33AF25307DFC80970571C51006CE6A	mmap-file	dumpkeys	Executable
#023	04A0599405EBD306CEF2447679C8F4B5159A55C7	mmap-file	loadkeys	Executable
#024	AE327AD27D02BF2DE96557A1B4053D02129B1394	mmap-file	setfont	Executable
#025	7334B75FDF47213FF94708D2862978D0FF36D682	mmap-file	gzip	Executable
#026	93D65AB85CF5EE1ACD9E6BE5057D622D80AB5E10	mmap-file	dmesg	Executable
#027	B6E90C3A25B69C3B1D3B643DB7D9504FBC36C1D1	mmap-file	minilogd	Executable

Root of Trust for Reporting

- RTR is a mechanism to securely report that state of a platform to a third party. The idea is to digitally sign the PCR values inside the TPM and send the signature to the requester.
- Endorsement Key (EK) forms the Root of Trust for Reporting (RTR)
  - 2048 bit RSA key contained inside the TPM
  - private part never leaves the TPM (only exists in shielded location)
  - EK is unique for every TPM and therefore uniquely identifies a TPM
  - typically generated by TPM manufacturer in the fab, either inside the TPM or generated off-chip and then injected (private EK is exposed)
  - The EK is backed by an EK certificate typically issued by the TPM manufacturer. The EK certificate guarantees that the key actually is an EK and is protected by a genuine TPM.
  - EK can not be changed or removed

Attestation Identity Keys

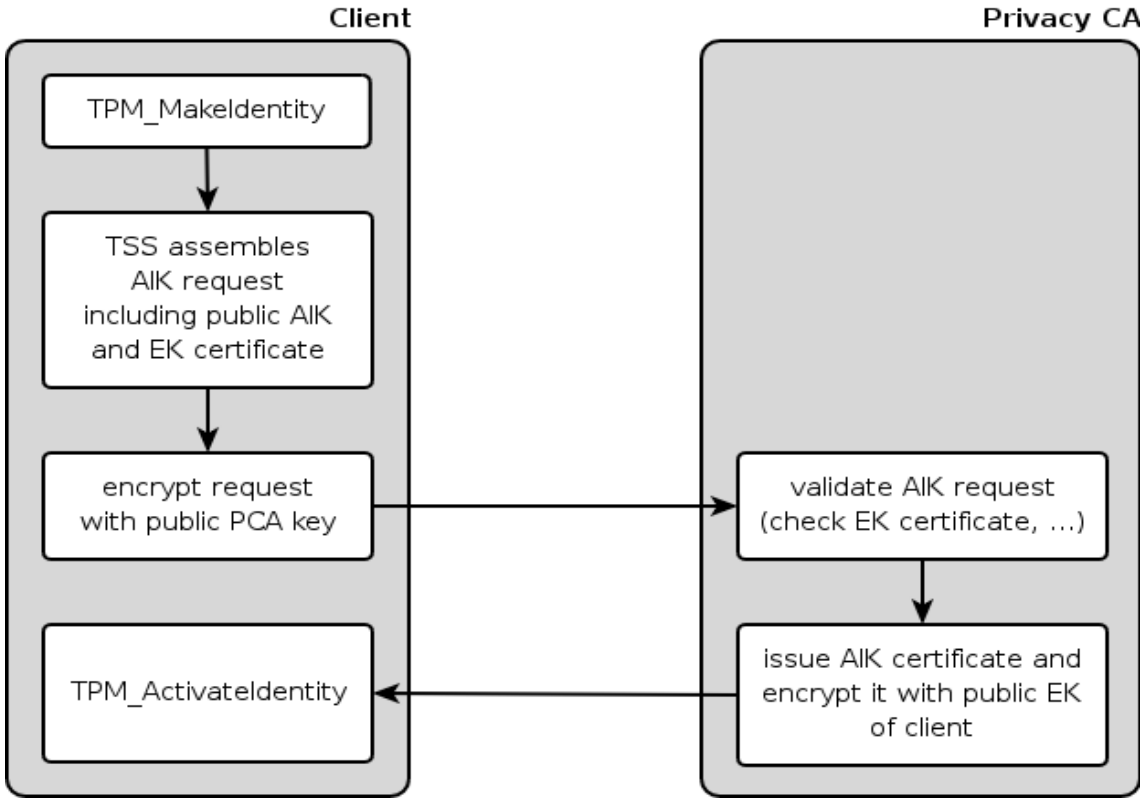
- Uniqueness of the EK would be a privacy problem if the EK were used directly in digital signature operations. All signatures done with the EK could be tracked back to one single machine.
- Therefore signing the PCRs using the EK and thereby proving that the PCR values are protected by a TPM is not an option.
- Attestation Identity Keys (AIKs) have been introduced as alias keys for the EK. The AIKs are designed to provide privacy to users.
- Attestation Identity Keys (AIKs)
  - 2048 bit RSA keys
  - provides a mechanism to ensure that one is communicating with a TPM, but not which TPM
  - signature key that signs only information generated inside the TPM
  - number of AIKs is not limited

Obtaining AIKs

- when signing data with an AIK, the verifier wants assurance that the key is a TPM protected key
- AIKs are backed by a certificate that vouches for the fact that the AIK is such a TPM protected key
- Privacy CA: trusted third party that issues certificates for AIKs
- basic AIK cycle
  - creates AIK key inside the TPM
  - AIK request data plus certificates are sent to PrivacyCA
  - PrivacyCA examines the supplied data
  - if PrivacyCA is convinced that the AIK is a TPM key it issues a certificate stating that fact

Basic AIK cycle

- client creates a new AIK key pair inside the TPM (TPM_MakeIdentity)
- TSS assembles AIK request
  - TPM EK certificate
  - public AIK
  - ...
- AIK request is encrypted with public key of PCA

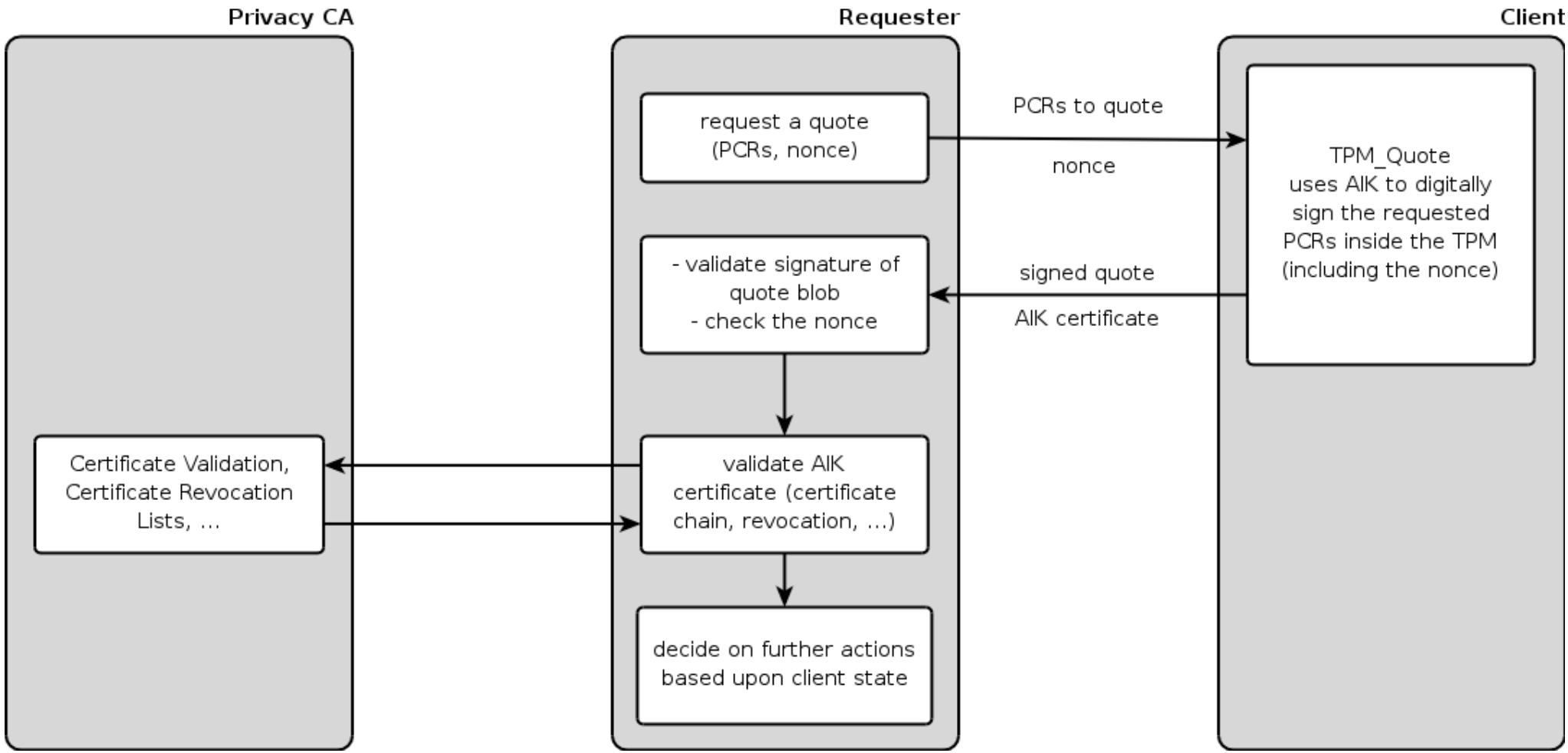


- PrivacyCA validates the AIK request; checks EK certificate, ... If PCA is convinced that the AIK is a proper TPM key, it issues an AIK certificate. PCA response is encrypted with public EK -> only the requester can read it.
- client activates AIK and stores AIK certificate

Attestation

- goal: prove to an outside entity that the system is in a specific state
- involved steps
  - requester specifies the set of PCRs he is interested in and supplies a nonce
  - client digitally signs the requested PCRs and the nonce inside the TPM using an AIK
  - signed data plus AIK certificate are sent to requester
  - requester checks the digital signature (using AIK certificate) and the nonce (to ensure freshness)
  - requester contacts PrivacyCA to determine the state of the AIK certificate
  - depending on the outcome (PCRs as expected by the requester, signature key is a TPM key, ...) requester decides if he wants to communicate with the client or not

Attestation



Certifying Keys

- certifying keys = TPM makes statement that “this key is held in a TPM-shielded location, and it will never be revealed” (when using an AIK to certify the key!)
  - certifying essentially means to sign the hash of the public key and related key parameters
- AIKs can certify only non-migratable keys (exception CMKs)
  - challenger must trust the TPM manufacturer
  - to validate a certified key the AIK credential can be used (verifier has to trust the policy of the PrivacyCA)
  - for keys certified with an AIK, the user can ask a CA to issue a certificate with a special extension
- normal singing keys and legacy keys can certify migratable and non-migratable keys
  - usefulness of this type of certification highly depends on the trust in the signing/legacy key

Root of Trust for Storage

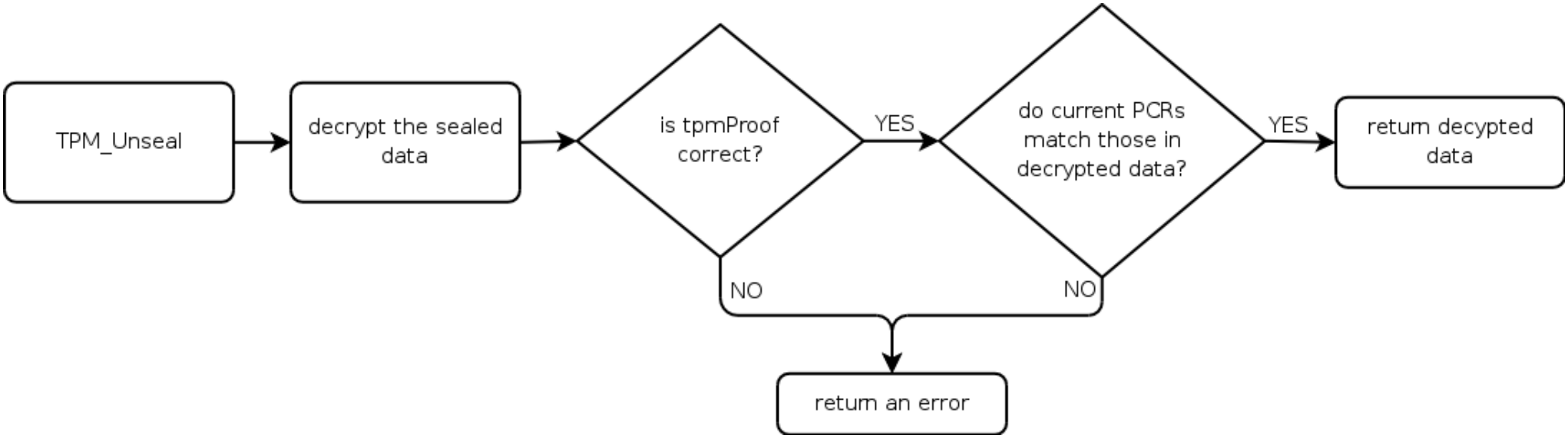
- use of TPM keys for encrypting data and keys
- two flavors:
  - without using PCRs: bind/unbind
  - with using PCRs: seal/unseal
- binding
  - happens outside of the TPM
  - encrypt data with the public part of a TPM key
  - only the TPM the key pair belongs to can decrypt the data
    remember: private key can only be used inside the TPM
  - binding to a specific TPM: use a non-migratable binding key
- unbinding
  - decryption of bound data inside the TPM using the private key

Root of Trust for Storage

- sealing
  - a way to combine measurements (PCR content) and external data
  - encrypt externally provided data with reference to a specific PCR state
  - only the TPM that sealed the data can do the unseal (ensured by including a nonce that only is known to this specific TPM)
  - PCR values specified do not have to be the platforms current PCR values but can be some other (future) PCR values
  - using a storage key (or legacy key)
- unsealing
  - load key that was used for sealing into TPM
  - decrypt sealed blob inside TPM

Root of Trust for Storage

- unsealing (cont.)
  - TPM checks the tpmProof included in the internal data: if the nonce does not match the one of the TPM it returns an error
  - if the specified PCR values do not match the platforms current PCR values an error is returned



PCRs revisited

- Summary of PCR usage scenarios
  - attestation of platform state (TPM_Quote)
  - protecting data (TPM_Seal/TPM_Unseal)
  - specify set of PCRs upon key creation: key is only usable if these PCRs are present
- Collection of measurements is done outside of the TPM by the platform (chain of trust starting at the RTM).
- problematic issues:
  - chain must not be broken
  - what to measure (binaries, configuration files, scripts, ...)
  - how to handle system updates?
  - pool of measurement values can become very large
- field of ongoing research (e.g. property based attestation)

Low-Level TPM Interaction

- TPM specification defines a set of basic and complex types
  - definitions are in C style
  - primitive types: BYTE, BOOL, UINT16, UINT32
  - complex types are defined as C structures (typically prefixed with TPM_ (in spec 1.2) or TCPA_ (in spec 1.1))
  - all type definitions, structures, constants and error codes are located in part 2 of the specification
- TPM has a byte stream oriented interface, meaning that all commands are serialized into byte arrays which are sent to the TPM. The TPM response again is a byte stream that has to be parsed.
- The typical (and only) system entity directly accessing the TPM is the TCG Software Stack (TSS).

TPM Command/Object Authorization

- many commands require the knowledge of authorization data
- typically not commands themselves require authorization but the objects the commands operate on (e.g. keys, encrypted data)
- some commands require owner authorization
- 20 bytes shared secret (SHA-1 hash of the password) is required to use protected objects
- secret is stored together with TPM controlled objects in shielded locations
- owner and SRK secret are stored inside the TPM
  note: TPM owner is no “super user” (e.g. has not access to all keys without knowing their secret)
- if the owner of an object (e.g. key) can provide the correct secret (authData) it also can execute all TPM operations applicable to that object

TPM Authorization Protocols

- TPM supports different authorization protocols depending on the specific purpose
- OIAP: Object Independent Authorization Protocol
  - supports authorization for arbitrary entities
  - session lives indefinitely until it is explicitly terminated
- OSAP: Object Specific Authorization Protocol
  - supports an authentication session of a single entity
  - enables confidential transmission of new authorization information
- ADIP: Authorization Data Insertion Protocol
  - used to insert new authorization information during the creation of a new object
- additional protocols to change authorization secrets

Rolling Nonce Paradigm

- on session establishment the TPM creates a nonce called “nonceEven” which is returned to the client
- the client creates a new nonce called “nonceOdd”
- the client includes the “nonceEven” received from the TPM and its own “nonceOdd” in the next request sent to the TPM
- the TPM validates the “nonceEven” contained in the request, creates a new “nonceEven” and includes this new “nonceEven” together with the “nonceOdd” from the client in its response
- client validates the “nonceOdd” received in the response, generates a new “nonceOdd”... and so on
- Summary: each new input message contains a new nonceOdd and each response contains a new nonceEven
- rolling nonces are never used on their own but as part of an authorization protocol

Authorization Data Insertion

- TPM entities such as keys have an associated usage secret (and if migratable, a migration secret)
- when creating a new key, the key secrets must not be transmitted to the TPM in plain -> they are XOR encrypted
- establish an OSAP session using the parent of the new entity

XOR_KEY = SHA1(sharedOSAPsecret || *nonceEven)*
encAuth = XOR(newEntity.secret, XOR_KEY)

- *note: newEntity.secret is the SHA1 hash of the secret provided by the user (i.e. the TPM stores all secrets as SHA1 hashes)*
- The encrypted authorization data (encAuth) is sent to the TPM as part of the TPM command that creates the new entity (e.g. key)
- The TPM computes the XOR_KEY the same way and then retrieves the secret of the new entity, which is stored together with this entity.

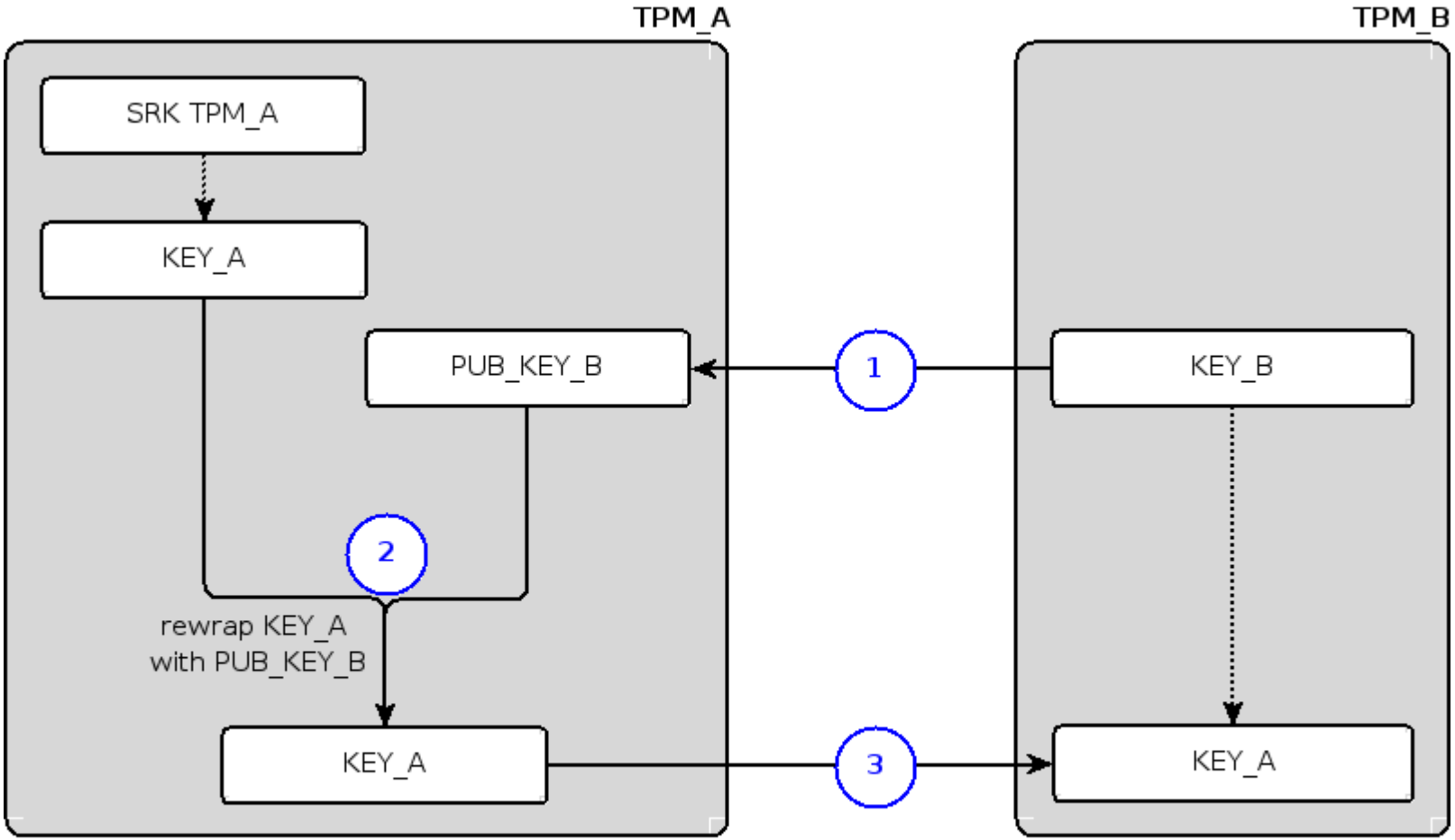
Advanced TPM concepts

- Overview
  - Key Migration and Certified Migratable Keys
  - Monotonic Counters and Timestamping
  - Non-Volatile Storage
  - Delegation
  - Transport Sessions
  - Maintenance
  - Localities (covered in later lecture)
  - Direct Anonymous Attestation (covered in later lecture)
- Many of those concepts have been introduced with TPM specification 1.2.

TPM Key Migration

- basic migration scheme (TPM 1.1)
  - migrate RSA keypair KEY_A from TPM_A to TPM_B
  - parent of KEY_A is the SRK (i.e. private part of KEY_A is encrypted with the public SRK of TPM_A)
  - storage key KEY_B of TPM_B to become new parent of KEY_A
  - prepare KEY_B by calling TPM_AuthorizeMigrationKey on TPM_B
  - transport the resulting public key structure to TPM_A
  - call TPM_CreateMigrationBlob using the public part of KEY_B on TPM_A
  - TPM_A internally re-wraps KEY_A: decryption of the private part of KEY_A using the private SRK and then encrypting using the public part of KEY_B (requires knowledge of migration secret of KEY_A)
  - re-wrapped KEY_A can now be loaded into TPM_B (unwrapping it with its new parent: KEY_B)

Basic TPM Key Migration at a Glance



- **1** – transfer public part of KEY_B to TPM_A
- **2** – re-wrap private part of KEY_A using public part of KEY_B
- **3** – KEY_A can now be loaded into TPM_B (KEY_B, its parent, is a key of TPM_B)

TPM Key Migration

- keys that can not be migrated: EK, SRK, AIKs, Non-Migratable Keys
- migratable keys have a migration secret
- advanced key migration (TPM 1.2)
  - Certified Migration Keys (CMKs)
  - requires external 3rd parties:
    - Migration Selection Authority (MSA)
    - Migration Authority (MA)
  - so far no known implementation of this infrastructure
- TPM Maintenance: vendor specific migration
  - optional TPM feature; vendor specific; requires owner authorization
  - allows to export all TPM data (migratory and non-migratory data)
  - never implemented by any manufacturer due to security issues

Monotonic Counters and Timestamps

- Monotonic Counter
  - allows for 7 years of increments every 5 seconds
  - useful e.g. against replay attacks
- Time Stamping
  - TPM offers time stamping mechanism
  - time stamps are not an universal time clock (UTC) value but the number of timer ticks counted by the TPM
  - TPM has a tick session (started when TPM is powered on)
    - tick value, increment rate, session nonce (new at every power cycle)
  - TPM_TickStampBlob takes a digest to be signed with a TPM signing key; current tick value and session nonce are included in signature
- mechanism outside the TPM is required to associate the tick value with an UTC value

Non-Volatile Storage

- TPM contains protected non-volatile storage
- TPM Owner can define storage locations and their appropriate access protection (e.g. requiring authorization for read/write, coupling access to the platform state, ...).
- stored entities addressed using indices
- potential usage scenarios:
  - secure storage for certificates such as EK certificate
  - storage for early bootup stages where other forms of persistent storage are not yet available

Transport Sessions

- a mechanism to protect information sent to and returned from the TPM
- data transmitted to/from the TPM is encrypted
- sequence of commands: groups a set of commands and provides a digital signature on all of the commands
  - these transport logs provide evidence that a series of operations actually took place inside the TPM
  - may also include timing information of actual execution
- exclusive transport sessions: any command outside of the session causes the session to terminate

Further Reading

- Trusted Computing Group:
  http://www.trustedcomputinggroup.org/

  Architecture Overview
  https://www.trustedcomputinggroup.org/specs/IWG/TCG_1_0_Architecture_Overview.pdf

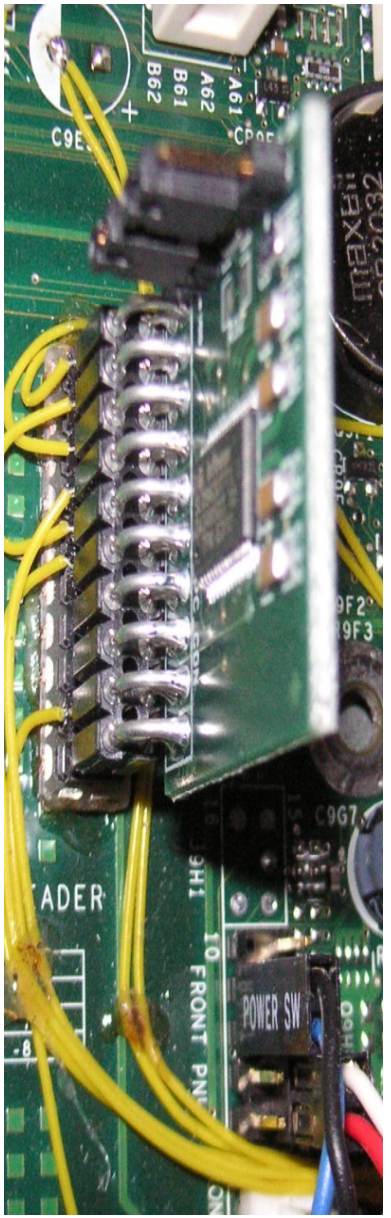
  TPM Specification
  https://www.trustedcomputinggroup.org/specs/TPM

- Trusted Computing Systeme
  Thomas Müller; Springer

- The Intel Safer Computing Initiative
  Building Blocks for Trusted Computing
  David Grawrock; intel press

Questions?

Thank you very much for your attention!



Open_TC EC Contract No: IST-027635

- The Open_TC project is co-financed by the EC.
  contract no: IST-027635

- If you need further information, please visit our website www.opentc.net or contact the coordinator:

  Technikon Forschungs- und Planungsgesellschaft mbH
  Richard-Wagner-Strasse 7, 9500 Villach, AUSTRIA
  Tel. +43 4242 23355 0
  Fax. +43 4242 23355 77
  Email coordination@opentc.net