Trusted Computing
→ Trusted Platform Module (TPM)

Prof. Dr.
Norbert Pohlmann

Institute for Internet Security - if(is)
University of Applied Sciences Gelsenkirchen
http://www.internet-sicherheit.de

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- Summary

Content

- **Aim and outcomes of this lecture**
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- Summary

Trusted Platform Module (TPM)
→ Aims and outcomes of this lecture

Aims

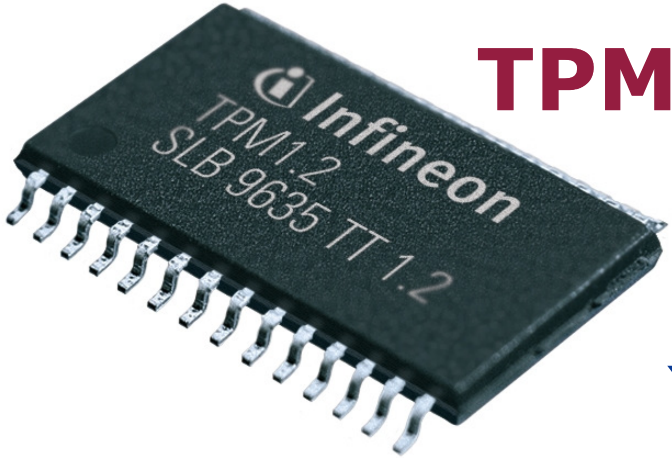
- To introduce the idea of the Trusted Platform Module (TPM)
- To explore the architecture and the functions of Trusted Platform Module (TPM)
- To analyze the functions and protocols of the Trusted Platform Module (TPM)
- To assess needs of the Trusted Platform Module (TPM)

At the end of this lecture you will be able to:

- Understand what is meant by the Trusted Platform Module (TPM).
- Know some of the functions of the Trusted Platform Module (TPM).
- Know what the protocols of the Trusted Platform Module (TPM) look like.
- Understand the capabilities and limitations of the Trusted Platform Module (TPM).

Content

- Aim and outcomes of this lecture
- **Overview of the idea of TPM**
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- Summary



TPM

The Safe

on our Motherboard!

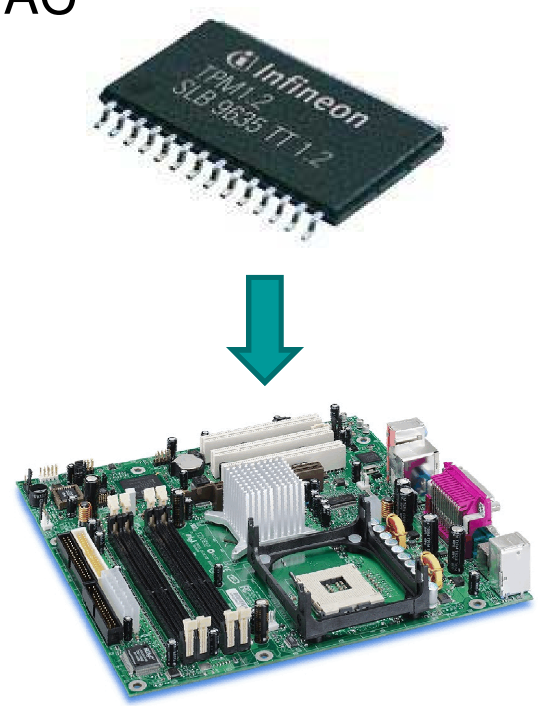
Trusted Platform Module (TPM)
→ Overview (2/4)

The Trusted Platform Module (TPM) is ...

- a **passive** security controller
- **bound to the mainboard** of a computing platform (e.g. PC, notebook, PDA, mobile phone, ...)
- but **physically separated** from the **main processor**
- capable to **withstand logical and physical attacks** to protect it's credentials
- proven and **certified by a third-party** Common Criteria evaluation
- **integrated in the booting process** as well as in the operating system

Trusted Platform Module (TPM)
→ Overview (3/4)

- Current implementation is a **security controller**
  - Hardware-based random number generation
  - Small set of cryptographic functions
    - Key generation, signing, encryption, hashing, MAC
- Offers **additional functionalities**
  - Secure storage (ideally tamper-resistant)
  - Platform integrity measurement and reporting
- **Embedded** into the platform’s motherboard
- Acts as a “**Root of Trust**”
  - TPM must be trusted by all parties
- Two versions of specification **available**
- Many vendors already ship their platforms with a TPM [TPMMatrix2006]



Trusted Platform Module (TPM)
→ Overview (4/4)

Common misconceptions

- The TPM does not measure, monitor or control anything
  - Software measurements are made by the "PC" and sent to the TPM
  - The TPM has no way of knowing what was measured
  - The TPM is unable to reset the PC or prevent access to memory

- The platform owner controls the TPM
  - The owner must opt-in using initialization and management functions
  - The owner can turn the TPM on and off
  - The owner and users control use of all keys

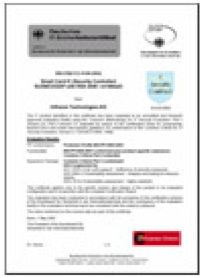
Security features of Infineon TPM
→ Overview (Example of one TPM)

Electro Magnetic Analysis (EMA)

Differential Fault Attack (DFA)

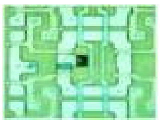
Alpha Particle Penetration

Timing Analysis



Global and Local Optical Attacks

Contrast Etching / Decoration



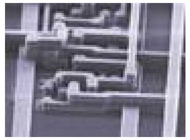
Countermeasures:

- ☺ Active Shields
- ☺ Security Memory Cells
- ☺ Hardware Encryption
- ☺ Hidden Layout Techniques
- ☺ Memory Scrambling
- ☺ Proprietary CPU Kernel
- ☺ Randomizing Features
- ☺ Test mode Locking Mechanism
- ☺ Sensors and Filters

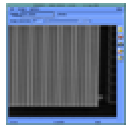
… more than 50 security features

Probing / Forcing

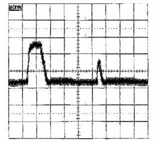
Reverse Engineering / Delayering



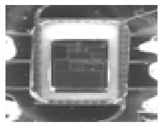
Electron Microscopy



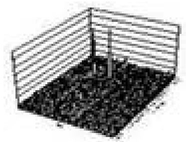
Spike / Glitch Penetration



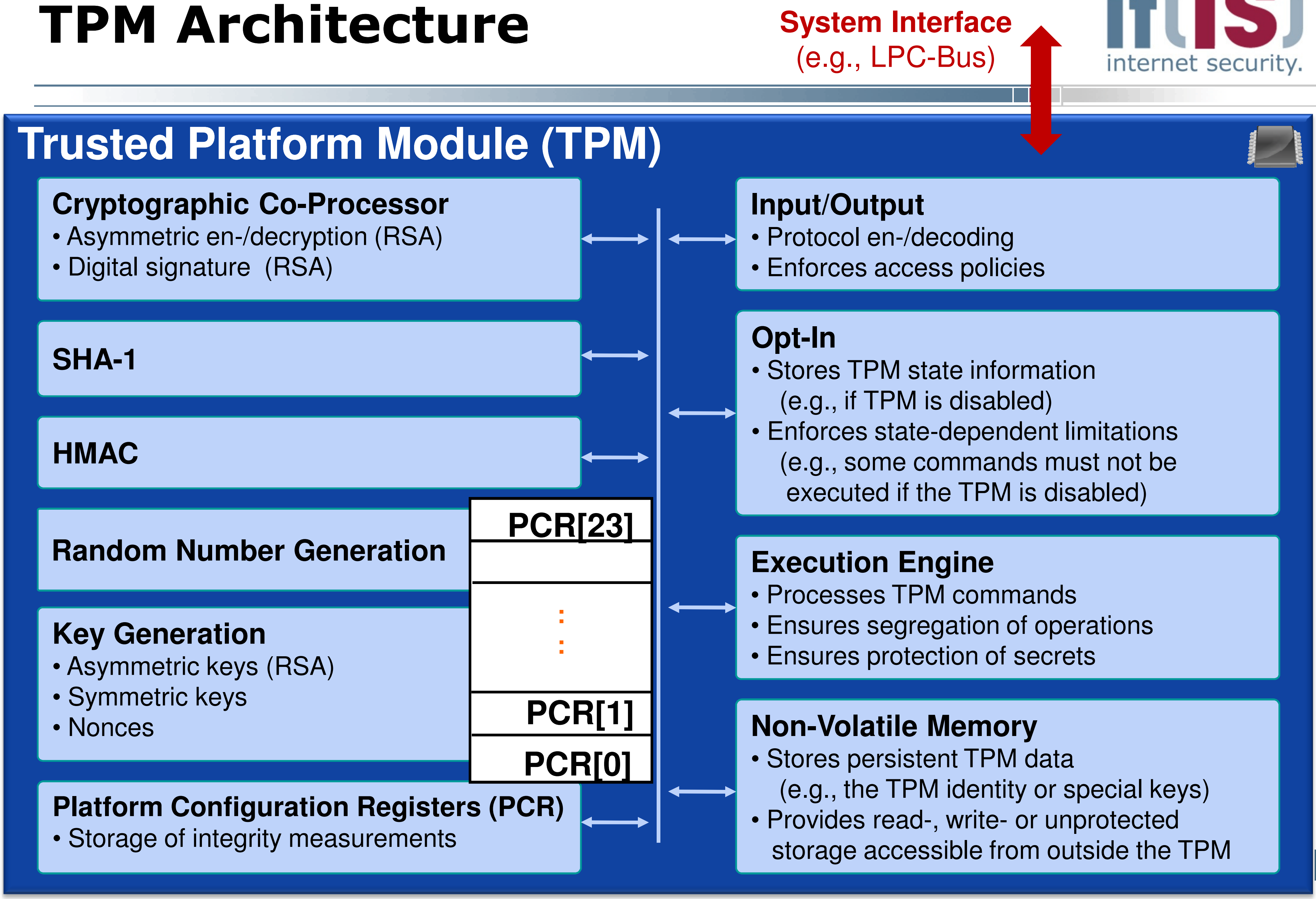
Atomic Force Microscopy (AFM)



Differential Power Analysis (DPA)



TPM Architecture



TPM Internal Functions
→ Features I

- **SHA-1 engine**
  - Computes the SHA-1 digest (digest) of arbitrary data (data)

$$\text{digest} \leftarrow \text{SHA-1(data)}$$

- **HMAC engine**
  - Computes the HMAC digest authDigest resulting from a secret secret and arbitrary data (data)

$$\text{authDigest} \leftarrow \text{HMAC(secret , data)}$$

  - Mainly used in TPM's authentication protocols
    - See OSAP/OIAP protocols (TPM authorization protocols)

- **Platform Configuration Registers (PCR)**
  - Copies the current values stored in the TPM's PCRs to state

$$\text{state} \leftarrow \text{getCurrentPCRs()}$$

  - e.g., used in the context of sealing to derive platform's current configuration

TPM Internal Functions
→ Features II

- **Random Number Generator**
  - Returns n random bytes

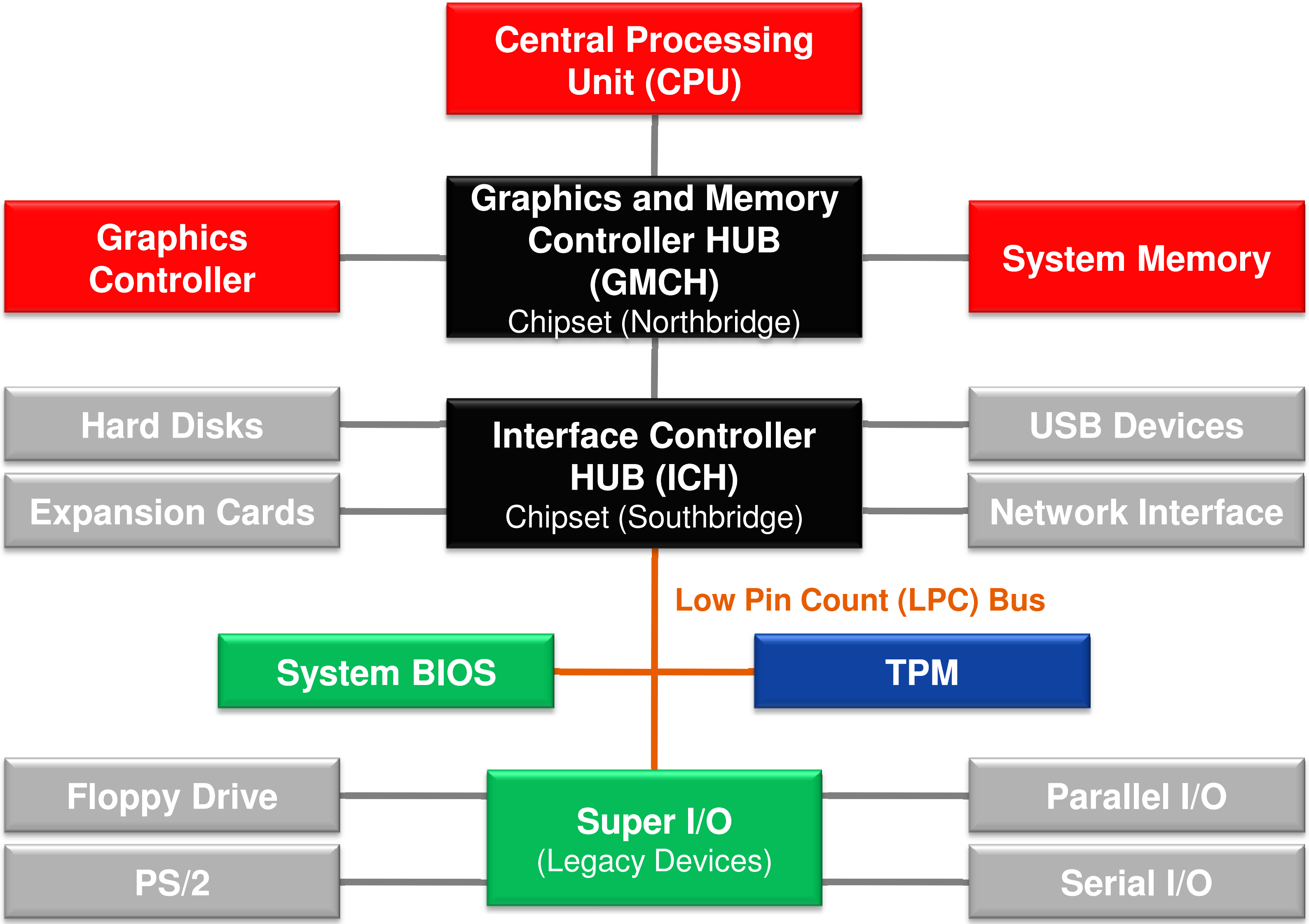
$$rand \leftarrow RNG(n)$$

  - Mainly used to derive 20 random bytes
    - e.g., to be used as nonce (anti-replay value)

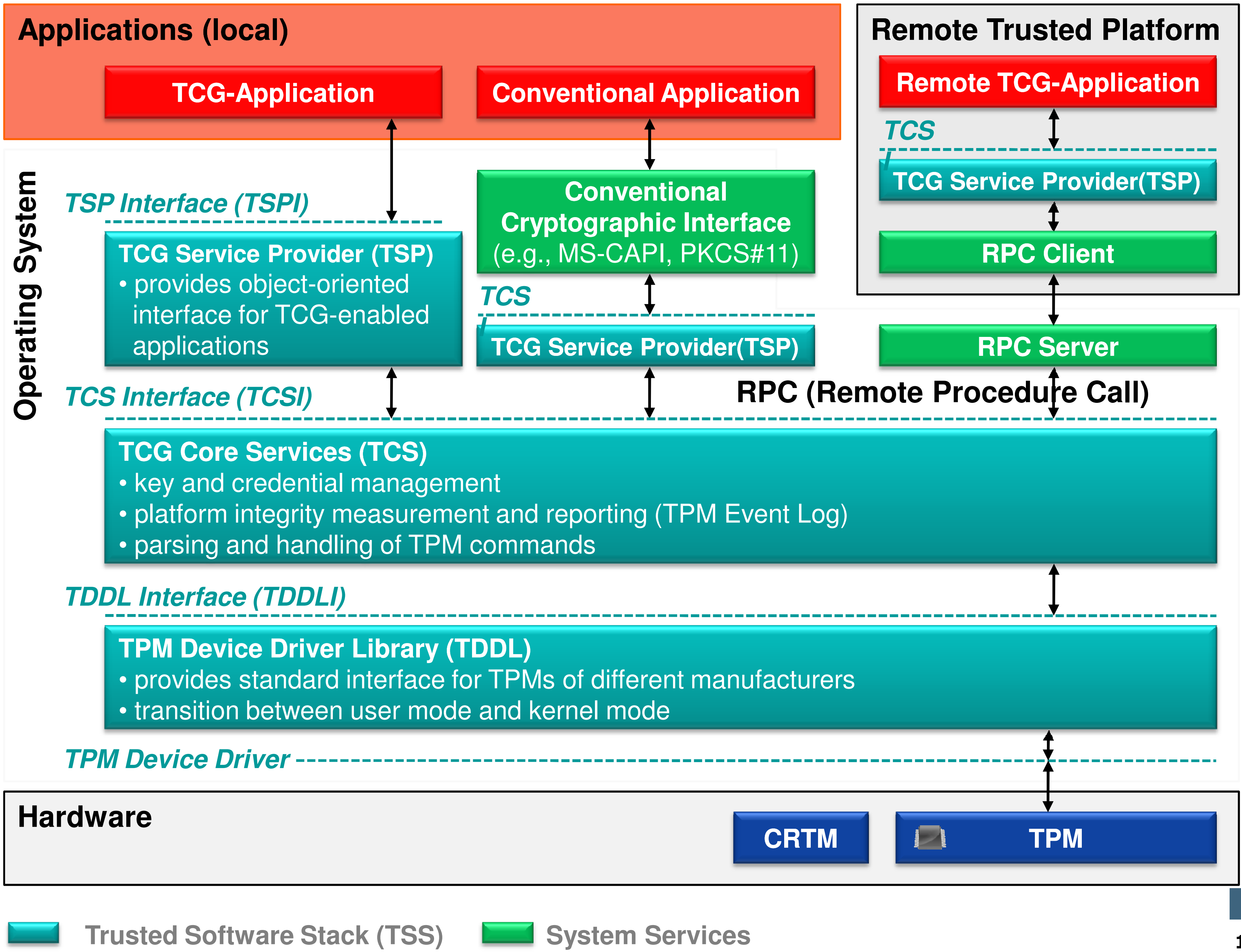
- **Key Generation Engine**
  - Generates a key pair (pk, sk) according to the parameters given in par (e.g., key size, key type, etc.)

$$(pk , sk) \leftarrow GenKey(par)$$

Trusted Platform Module (TPM)
→ TPM Integration into PC-Hardware



TPM Software Integration



Trusted Platform Module (TPM)
→ TPM Startup in a PC

1. **User powers on / resets platform**
   `TPM_Init()`
   - No software-executable command
   - Informs TPM about system-wide reset
   - Platform design must ensure that TPM receives `TPM_Init()` only if platform performs a complete reset

2. **BIOS starts TPM**
   `TPM_Startup(state)`
   - Executed by the system BIOS
   - state ϵ { clear , save , deactivated }
     - clear   volatile memory initialized with default values
     - save   volatile memory initialized with values previously saved to TPM's non-volatile memory
     - deactivated   deactivates the TPM

3. **BIOS instructs TPM to perform a full self-test**
   `TPM_ContinueSelfTest()`
   - Executed by the system BIOS
   - Instructs TPM to perform a full self-test

4. **TPM is ready to be used**

Trusted Platform Module (TPM)
→ Core Root of Trust for Measurement

- **Immutable portion** of the host platform's initialization code that executes upon a host platform reset
- **Trust** in all measurements is based on the **integrity of the "Core Root of Trust for Measurement" (CRTM)**
- Ideally the CRTM is **contained in the TPM**
- Implementation decisions may require it to be located in other firmware (e.g., BIOS boot block)

Two Possible CRTM Implementations

1. **CRTM is the BIOS Boot Block**
   - BIOS is composed of a BIOS Boot Block and a POST BIOS
   - Each of these are independent components
     - Each can be updated independent of the other
   - BIOS Boot Block is the CRTM while the POST BIOS is not, but is a measured component of the Chain of Trust

2. **CRTM is the entire BIOS**
   - BIOS is composed of a single atomic entity
   - Entire BIOS is updated, modified, or maintained as a single component

CRTM: Core Root of Trust for Measurement

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- **Terminology and Assumption**
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- Summary

Trusted Computing Group (TCG)
→ Terminology I

- **Shielded Location**
  - ***Place*** where ***sensitive data can*** **safely** ***be stored or operated***
    - e.g., memory locations **inside the TPM** or data objects **encrypted by the TPM** and stored on external storage (e.g., hard disk)
- **Protected Capabilities (Protected Functions)**
  - Set of commands with **exclusive permission** to access shielded locations
    - e.g., commands for cryptographic key management, sealing of data to a system state, etc.
- **Protected Entity**
  - Refers to a protected capability or sensitive data object stored in a shielded location

Trusted Computing Group (TCG) → Terminology II

- **Integrity Measurement**
  - Process of obtaining metrics of platform characteristics that affect the integrity (trustworthiness) of a platform and storing digests of those metrics to the TPM's **PCRs (Platform Configuration Registers)**
    - Platform characteristic = **digest of the software to be executed**

- **Platform Configuration Registers (PCR)**
  - **Shielded location** to **store integrity measurement values**
  - Can only be extended: $PCR_{i+1} \leftarrow$ SHA-1(PCR_i , value)
  - PCRs are reset only when the platform is rebooted

- **Integrity Logging**
  - Storing integrity metrics in a log for later use
  - e.g., storing additional information about what has been measured like software manufacturer name, software name, version, etc.

Trusted Computing Group (TCG)
→ Assumption and Trust Model I

- **Unforgeability of measurements**
  - Platform configuration **cannot be forged after measurements**
  - *However, today's OS can be modified*
- **Digest values express trustworthiness**
  - Verifier can determine initial configuration from **digests**
  - *However, TCBs of today's platforms are too complex*
- **Secure channels can be established**
  - Between HW components (TPM and CPU) since they might have certified authentication keys provided by a PKI
  - Between machines running on a platform (e.g., attestor and host), provided by operating system mechanisms (secure OS)

Trusted Computing Group (TCG)
→ Assumption and Trust Model II

- **Protection against software attacks only**
  - Unprotected communication link between TPM and CPU
  - See, e.g., [KuScPr2005]

- **Security issues of certain TPM aspects**
  - See, e.g., [GuRuScAtPl2007] for an automated verification

- **Integration of TPM functionality in chipset may potentially be problematic**
  - Engineering trade off between security and technical evaluation
  - TPM Construction Kit
  - Towards more security against hardware attacks

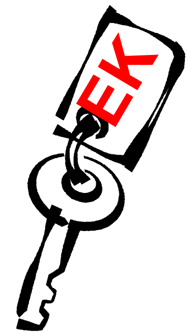
- **Currently**
  - TPMs have rudimentary protection mechanisms (TPM stems from smartcards)
  - Some manufacturers started third party certification
  - CRTM is not tamper-resistant

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- **Identities**
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- Summary

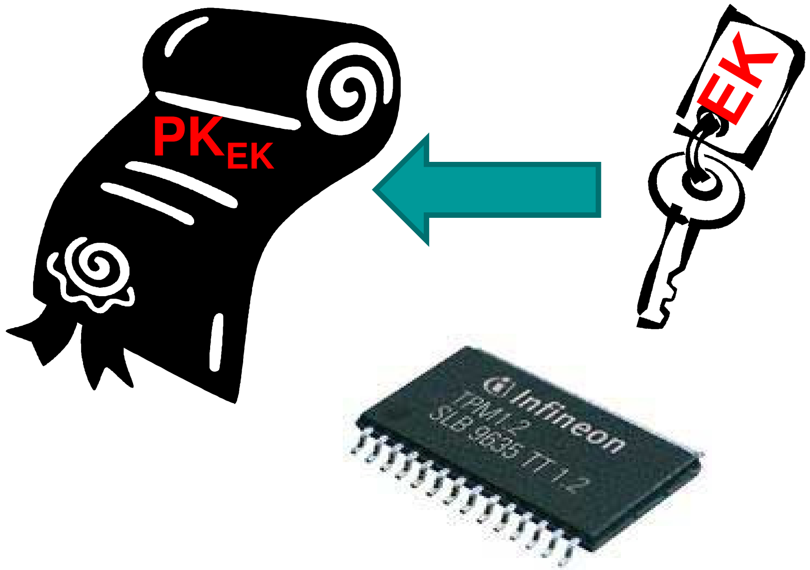
Identities
→ TPM Identity (Endorsement Key)

- **TPM identity represented as Endorsement Key (EK)**
- **Unique en-/decryption key pair**
  - Private key does not leave TPM
  - Public key is privacy-sensitive (since it identifies a TPM/platform)
- **Generated during manufacturing process of TPM**
  - Either **in TPM** or **externally** and then embedded into the TPM
- **Must be certified by EK-generating entity**
  - e.g., by the TPM manufacturer
- **Can be deleted (revoked) and re-generated by a TPM user**
  - Revocation must be enabled during creation of the EK
  - Deletion must be authorized by a secret defined during EK creation
  - EK-recreation invalidates Endorsement Credential (EC)
- **Readable from TPM via**
  - `TPM_ReadPubek` (command disabled after taking ownership)
  - `TPM_OwnerReadInternalPub` (requires owner authorization)



Identities
→ Endorsement Credential

- **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
- **Includes**
  - TPM manufacturer name
  - TPM model number
  - TPM version
  - Public EK (privacy sensitive)



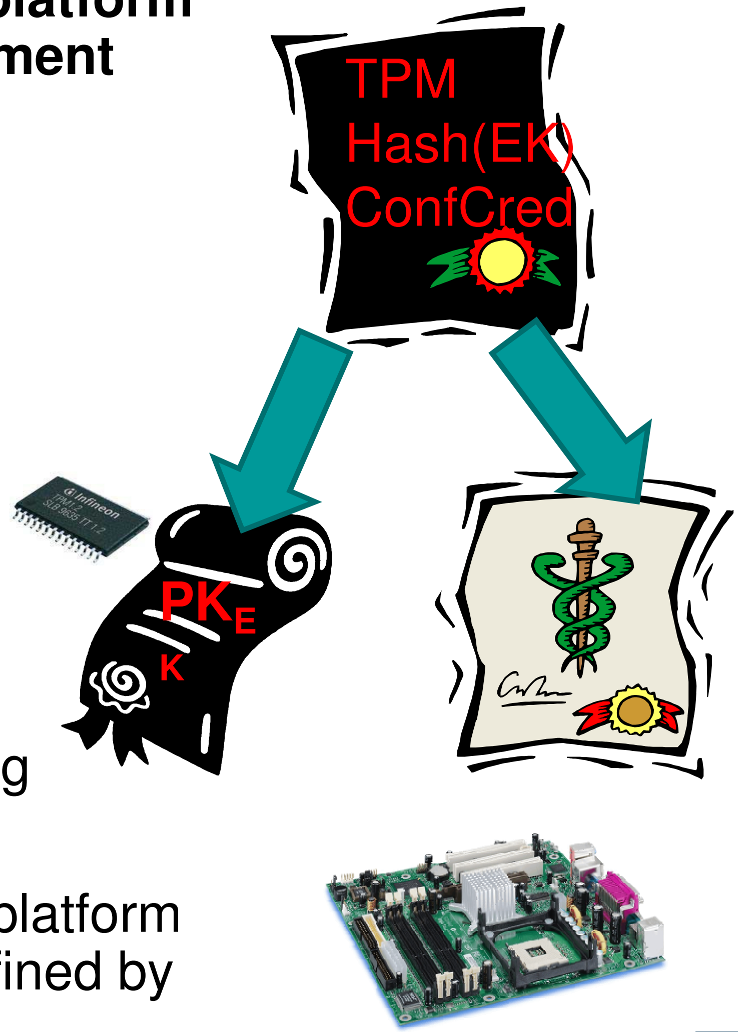
Identities
→ Platform Identity

- **Platform identity is equivalent to TPM identity (EK)**
  - EK is unique identifier for a TPM
  - A TPM must be bound to only one platform
    - Either physical binding (e.g., soldered to the platform's motherboard) or logical binding (e.g., by using cryptography)
    - Common implementation: TPM soldered to the platform's motherboard
  - Therefore an EK uniquely identifies a platform

- **Platform Credential asserts that a TPM has been correctly integrated into a platform**

Identities
→ Platform Credential

- **Digital certificate stating that an individual platform contains the TPM described in the Endorsement Credential (EC)**
- **Issued by the platform manufacturer**
  - e.g., system or motherboard manufacturer
- **Includes**
  - Platform manufacturer name
  - Platform model and version number
  - References to (digests of) the corresponding Endorsement and Conformance Credential
    - Conformance Credential asserts that a platform type fulfills the evaluation guidelines defined by the TCG



Content

TPM Keys and Keys´ Properties
→ Migratable and Non-Migratable Keys

internet security.

- **Migratable keys**
  - Can be migrated to other TPMs/platforms
  - Third parties have no assurance that such keys have been generated by a TPM
    - Third parties may not trust migratable keys

- **Non-migratable keys**
  - Cannot be migrated to other TPMs/platforms
  - Guaranteed to only reside in TPM-protected locations
  - TPM can generate certificate stating that a key is non-migratable

TPM Keys and Keys´ Properties
→ Certified Migratable Keys (CMK)

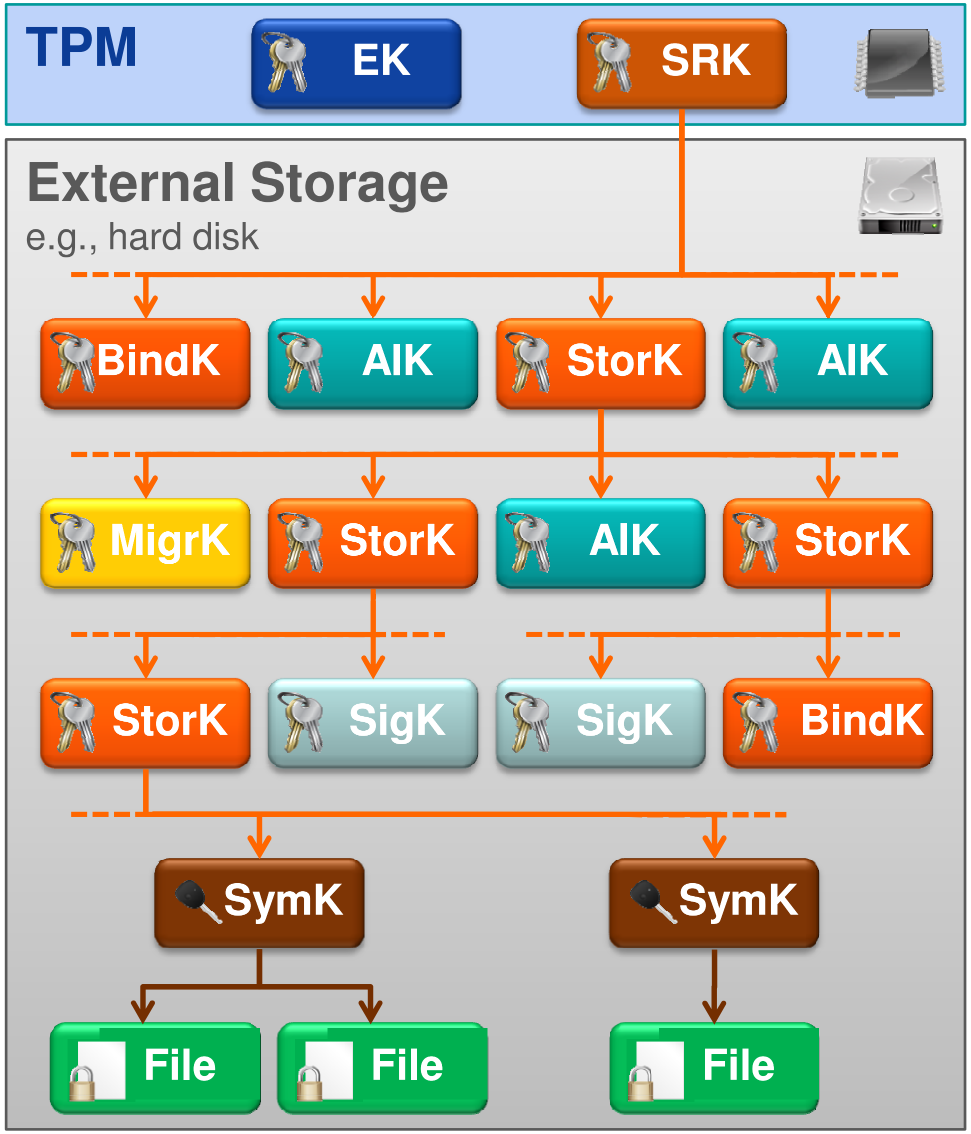
- **Introduced with TPM Specification 1.2**
- **Migration delegated to**
  - Migration-Selection Authority (MSA)
    - Controls migration of keys
  - Migration Authority (MA)
    - Performs the migration of keys
- **Migration of CMK to another TPM requires certificate of MA stating that the key is allowed to be transferred**
  - See Migration of TPM Keys

TPM Keys and Keys´ Properties
→ Secure Root Key (SRK)

- **TPM contains Root of Trust for Storage (RTS)**
  - Secure data storage implemented as a **hierarchy of keys**
  - Storage Root Key (SRK) is root of this key hierarchy
- **Storage Root Key (SRK) represents RTS**
  - RSA en-/decryption key pair
    - Must at least have 2048-bit key length
    - **Private SRK must not leave TPM**
  - Generated by TPM during process of installing TPM Owner
  - Deleted when the TPM Owner is deleted
    - This makes key hierarchy inaccessible and thus **destroys all data encrypted** with keys in that hierarchy!!!

A → B means A encrypts B
A is called **parent key** of B

TPM Key Hierarchy



- Depth of hierarchy and number of TPM-protected keys only limited by size of external storage
- **Storage keys (StoreK)** protect all other key types
  - **Attestation ID keys (AIK)**
  - **Signing keys (SigK)**
  - **Binding keys (BindK)**
  - **Migration Keys (MigrK)**
  - **Symmetric keys (SymK)**
- Transitive protection
  - SRK indirectly protects arbitrary data (e.g., **files)**

TPM Keys and Keys´ Properties
→ TPM Key Object – Important Fields

internet security.

TPM Key Object

General Information

- Key Type
- Algorithm
- Authorization Secret

Specific Information

- Key Length
- Key Data

Key Properties

- Migration
- PCR Values

Key Type: e.g., signing key, binding key, storage key, …

Algorithm: e.g., RSA, DSA, HMAC, AES, …

Authorization Secret: Authorization secret required to use the key

Key Data: **Public and private key, asymmetric key. Secret key data is encrypted with the corresponding parent key.**

Migration: **Information about the migratability of the key:**

- **migratable**
- **certified mitgratable**
- **non-migratable**

PCR Values: **A key can be sealed to specific PCR values. This means that such a key can only be used (successfully) when the platform is in a specific (trusted) state.**

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- **TPM Key Types**
- Some More TPM Details
- Summary

TPM Key Types
→ Overview

- **TPM provides 9 different types of keys**
  - 3 special TPM key types
    - Endorsement Key, Storage Root Key, Attestation Identity Keys
  - 6 general key types
    - Storage, signing, binding, migration, legacy and “authchange” keys
  - Most important key types explained in following slides …

- **Each key may have additional properties, the most important ones are**
  - Migratable, non-migratable, certified migratable
    - e.g., whether the key is allowed to be migrated to another TPM
  - Whether the key is allowed only to be used when the platform is in a specific (potentially secure) configuration

Legacy Keys (not recommended)

TPM Key Types
→ Attestation Identity Keys (AIK)

internet security.

- **Purpose**
  - Used to attest to current platform configuration
    - e.g., authentically report the current hard- and software environment to a remote party (see attestation)
  - **Alias for TPM/platform identity (Endorsement Key)**
  - Use of AIKs should prevent tracking of TPMs/platforms
    - e.g., the transactions of a platform can be traced if the EK is used in various protocol runs with different colluding service providers

- **Properties**
  - AIKs are non-migratable signing keys (e.g., 2048-bit RSA)
  - **Generated by the TPM Owner**
  - TPM/platform may have multiple AIKs
    - e.g., one for online-banking, one for e-mail, etc.

TPM Key Types
→ Certification of AIKs

- **AIK requires certification by Trusted Third Party (Privacy CA in TCG Terminology) certifying that an AIK comes from a TPM**

- **Unlinkability achieved by DAA (Direct Anonymous Attestation) protocols**
  - No Privacy CA needed
  - Zero-knowledge proof of knowledge of possession of a valid certificate

TPM Key Types
→ Storage Keys

- **Purpose: Protection of keys outside the TPM**
  - e.g., a storage key can be used to encrypt other keys, which can be stored on a hard disk
  - **Storage Root Key (SRK) is a special storage key**
  - Strong protection of arbitrary TPM-external data (sealing)
    - e.g., encryption of secrets, which can only be recovered if the platform has a defined hard-/software environment (see **sealing**)
- **Properties**
  - Typically 2048-bit RSA en-/decryption key pair
  - Generally allowed to be migrated to other TPMs
    - Are not allowed to be non-migratable if one of their parent keys is migratable
    - **Must be non-migratable if used for sealing**

TPM Key Types
→ Binding Keys

- **Purpose**
  - Protection of arbitrary data outside the TPM
    - Binding is equivalent to traditional asymmetric encryption
- **Description**
  - Asymmetric en-/decryption key pair
    - Typically RSA 2048-bit
    - Other asymmetric encryption schemes may be supported by the TPM
  - Migratable to other TPMs/platforms
    - Are not allowed to be non-migratable if one of their parent keys is migratable
- **Can only be used with binding-commands**

TPM Key Types
→ Signing Keys

- **Purpose**
  - Message authentication of arbitrary TPM-external data
    - e.g., to ensure integrity of arbitrary files stored on the platform or protocol messages sent by the platform and their origin
  - Authentic report of TPM-internal information
    - e.g., for auditing TPM commands or reporting TPM capabilities

- **Description**
  - Typically 2048-bit RSA signing/verification key pair
    - Other signing algorithms may be supported by the TPM
  - Signing keys may be migrated to other TPMs/platforms
    - Are not allowed to be non-migratable if one of their parent keys is migratable

TPM Key Types
→ Migration Keys

- **Purpose**
  - Enable TPM to act as migration authority
  - Used to encrypt migratable keys for secure transport from one TPM to another

- **Description**
  - 2048-bit RSA en-/decryption key pair
  - Are allowed to be migrated to another TPM

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- **Some More TPM Details**
- Summary

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- **Some More TPM Details**
  - **Creating TPM Identity**
- Summary

Creating TPM Identity
→ Creating a Non-Revocable EK

$(pk_{EK} , digest_{EK}) \leftarrow$ TPM_CreateEndorsementKeyPair(Nonce , par_{EK})

```
if EK exists or then
   return error;
else
   if par_EK describes a storage key providing security at least
   equivalent to RSA-2048 then
      ( sk_EK , pk_EK ) ← GenKey( par_EK );
      digest_EK ← SHA-1( pk_EK , Nonce );
      return ( pk_EK , digest_EK );
   else
      return error;
   end if;
end if;
```

Input

- Nonce is an anti-replay value chosen by the caller of the command (e.g., a software for creating the EK)
- par_{EK} are parameters for the key generation algorithm (e.g., key size, key type, etc.) chosen by the caller of the command

Note

- EK typically is a RSA key

Creating TPM Identity
→ Creating a Revocable EK

(pk_{EK}, $digest_{EK}$, A_{Rev}) ← TPM CreateRevocableEK(Nonce, par_{EK}, $par_{A_{Rev}}$, A'_{Rev})

```
if EK exists then
   return error;
else
   if par_EK provides security at least equivalent to RSA-2048 then;
      ( sk_EK , pk_EK ) ← GenKey( par_EK );
      if par_A_Rev = TRUE then
         A_Rev ← RNG( 20 );
      else
         A_Rev ← A'_Rev;
      end if;
      digest_EK ← SHA-1( pk_EK , Nonce );
      return ( pk_EK , digest_EK , A_Rev );
   else
      return error;
   end if;
end if;
```

Perquisites
- Command is executed in a secure environment (e.g., during manufacturing)

Input
- A'_{Rev} is authorization secret chosen by the caller of the command that must be presented to TPM in order to revoke the EK later

Note
- This is an optional command

Creating TPM Identity
→ Revoking a revocable EK

( ) ← TPM_RevokeTrust(A_{Rev})

```
if EK is non-revocable then
   return error;
else
   if A'Rev = ARev and physical presence is asserted then
      TPM_OwnerClear(...);
      invalidate all TPM-internal EK-related data;
      invalidate the EK;
   else
      return error;
   end if;
end if;
```

Perquisites

- Existing EK is revocable
- Authorization data required to revoke EK is A_{rev} , which has been defined during creation of the EK

Note

- The TPM recognizes physical presence, e.g., via a pin at the TPM wired to a button at the platform
- This is an optional command
- `TPM_OwnerClear()` resets all owner-specific data to default values (see TPM Owner)

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- **Some More TPM Details**
  - **TPM Owner**
- Summary

TPM Owner
→ Overview

- **Entity owning a TPM-enabled platform**
  - e.g., platform owning person or IT-department
- **TPM Owner must initialize TPM to use its full functionality (“take ownership” of the TPM)**
  - Owner sets owner authorization secret
  - Owner creates the **Storage Root Key (SRK)** (see TPM keys)
- **Owner authorization**
  - Proof of knowledge of the owner credentials to the TPM
    - e.g., via a challenge and response protocol or physical presence
  - Permits the TPM to use several protected capabilities
    - e.g., migration of cryptographic keys or deletion of TPM Owner

TPM Owner
→ Methods of Proving Ownership to a TPM

- **User proves knowledge of TPM owner authorization secret to the TPM**
  - e.g., OIAP or OSAP (see TPM authorization protocols)
- **Assertion of physical presence**
  - Proof of physical access to the TPM/platform
    - e.g., by using a hardware switch or changing a BIOS setting
  - Interface for asserting physical presence specified by the PC Client Specification
  - Only a few commands can be authorized via physical presence
    - e.g., deletion of TPM Owner, activation/deactivation of the TPM, enabling/disabling the TPM

TPM Owner
→ Protocol for Creating a TPM Owner

if(is) internet security.

TPM

TPM Owner O

Initialization of authorization protocol

`TPM_OIAP()` ←

$Handle_{OIAP}$, $Nonce_{TPM}$ →

`TPM_TakeOwnership(` $enc_{EK}(A'_{Owner})$, $enc_{EK}(A'_{SRK})$, par_{SRK}), $InAuthData_{OIAP}$ ←

$OutAuthData_{OIAP}$ →

TPM: **verifies** $InAuthData_{OIAP}$

TPM Owner: **computes** $InAuthData_{OIAP}$

TPM Owner: **verifies** $OutAuthData_{OIAP}$

Here, OIAP is only used to authenticate the TPM's response to the TPM Owner

- e.g., on successful verification of $OutAuthData_{OIAP}$ the TPM Owner can be assured that the TPM has created a TPM Owner and set the correct authorization secrets A'_{Owner} and A'_{SRK}
- See OIAP protocol (OIAP = Object Independent Authorization Protocol)

TPM Owner
→ TPM Interface for Taking Ownership

$(pk_{SRK} , OutAuthData_{OIAP}) \leftarrow$ `TPM_TakeOwnership(` $enc_{EK}(A'_{Owner}), enc_{EK}(A'_{SRK}), par_{SRK}),$
$InAuthData_{OIAP}$

if owner exists or EK is invalid
or $InAuthData_{OIAP}$ does not refer to an active OIAP session then
  return error;
else
  if par_{SRK} describes 2048-bit non-migratable RSA encryption key then
    $A_{Owner} \leftarrow dec_{EK}(enc_{EK}(A'_{Owner}));$
    store A_{Owner} as owner authorization data in non-volatile memory;
    $A_{SRK} \leftarrow dec_{EK}(enc_{EK}(A'_{SRK}));$
    $(sk_{SRK} , pk_{SRK}) \leftarrow GenKey(par_{SRK});$
    $SRK \leftarrow ((sk_{SRK} , pk_{SRK}) , A_{SRK});$
    store SRK in non-volatile memory;
    initialize all owner-related TPM-internal variables;
    compute $OutAuthData_{OIAP}$;
    return $(pk_{SRK} , OutAuthData_{OIAP});$
  else
    return error;
  end if;
end if;

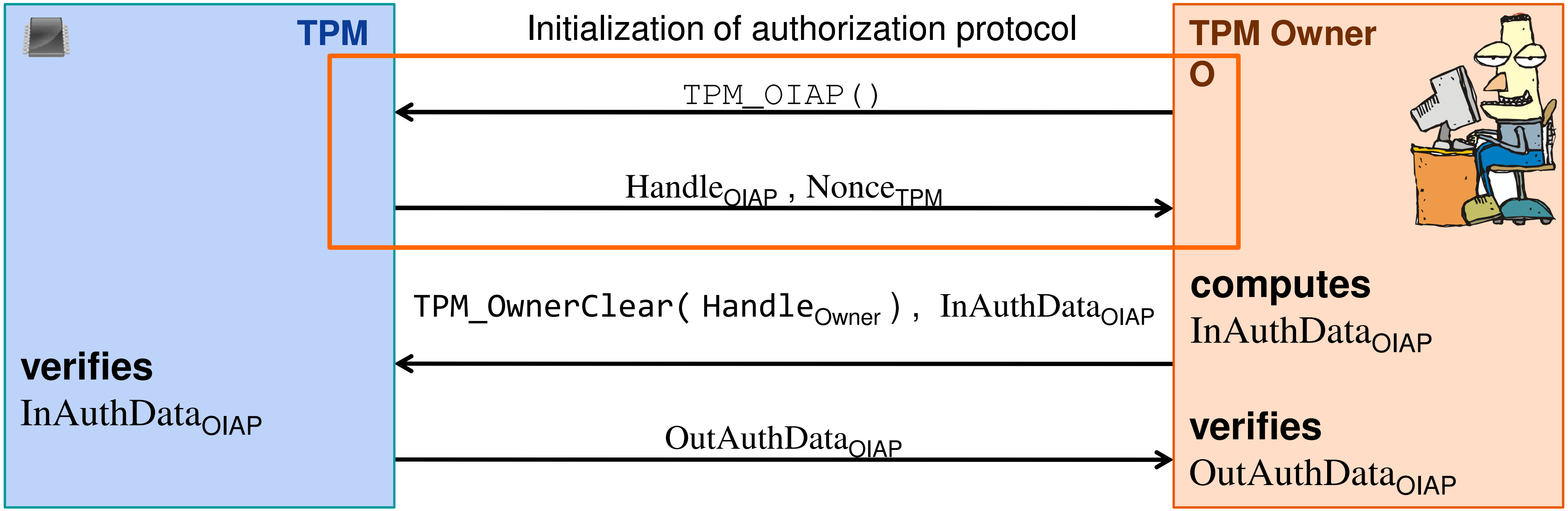
- SRK is used to protect shielded locations moved off the TPM to, e.g., a hard disk (see TPM keys)

Perquisites
- TPM Owner obtained authentic pk_{EK}, e.g., from Endorsement Credential

Input
- A'_{Owner} and A'_{SRK} are authorization secrets (e.g., digests of passphrases) chosen by the TPM Owner

Notes
- $InAuthData_{OIAP}$ is used to prove knowledge of the owner authorization secret to the TPM
- $OutAuthData_{OIAP}$ provides authenticity of the TPM's output to TPM Owner
- See OIAP protocol

TPM Owner
→ Protocol for Deleting a TPM Owner



OIAP session is used to authenticate

- **the TPM Owner to the TPM**

  e.g., on successful verification of InAuthData$_{OIAP}$ the TPM can be assured that the command has been called by the TPM Owner

- **the TPM's response to the TPM Owner**

  e.g., on successful verification of OutAuthData$_{OIAP}$ the TPM user can be assured that the TPM has actually deleted the TPM Owner and all associated data

TPM Owner
→ TPM Interface for Deleting Owner

$OutAuthData_{OIAP} \leftarrow$ TPM_OwnerClear($Handle_{Owner}$) , $InAuthData_{OIAP}$

```
if OIAPVerify( Handle_Owner , InAuthData_OIAP ) ≠ ok
or deletion of owner has been disabled then
   return error;
else
   compute OutAuthData_OIAP;
   unload all currently loaded keys;
   delete A_Owner;
   delete SRK;
   set all owner-related internal variables to their defaults;
   terminate all currently open sessions;
   return OutAuthData_OIAP;
end if;
```

Notes

- $Handle_{Owner}$ informs the TPM that the TPM Owner should be authorized
- InAuthDataOIAP refers to parameters of a previously opened OIAP authorization session used to prove knowledge of the owner authorization secret to the TPM
- $OutAuthData_{OIAP}$ refers to the parameters of a previously opened OIAP session providing authenticity of the TPM's output (e.g., proof that the TPM actually deleted the TPM Owner)
- OIAP_Verify() verifies if user knows owner authorization secret
- See OIAP authorization protocol

TPM Owner
→ Deleting Owner via Physical Presence

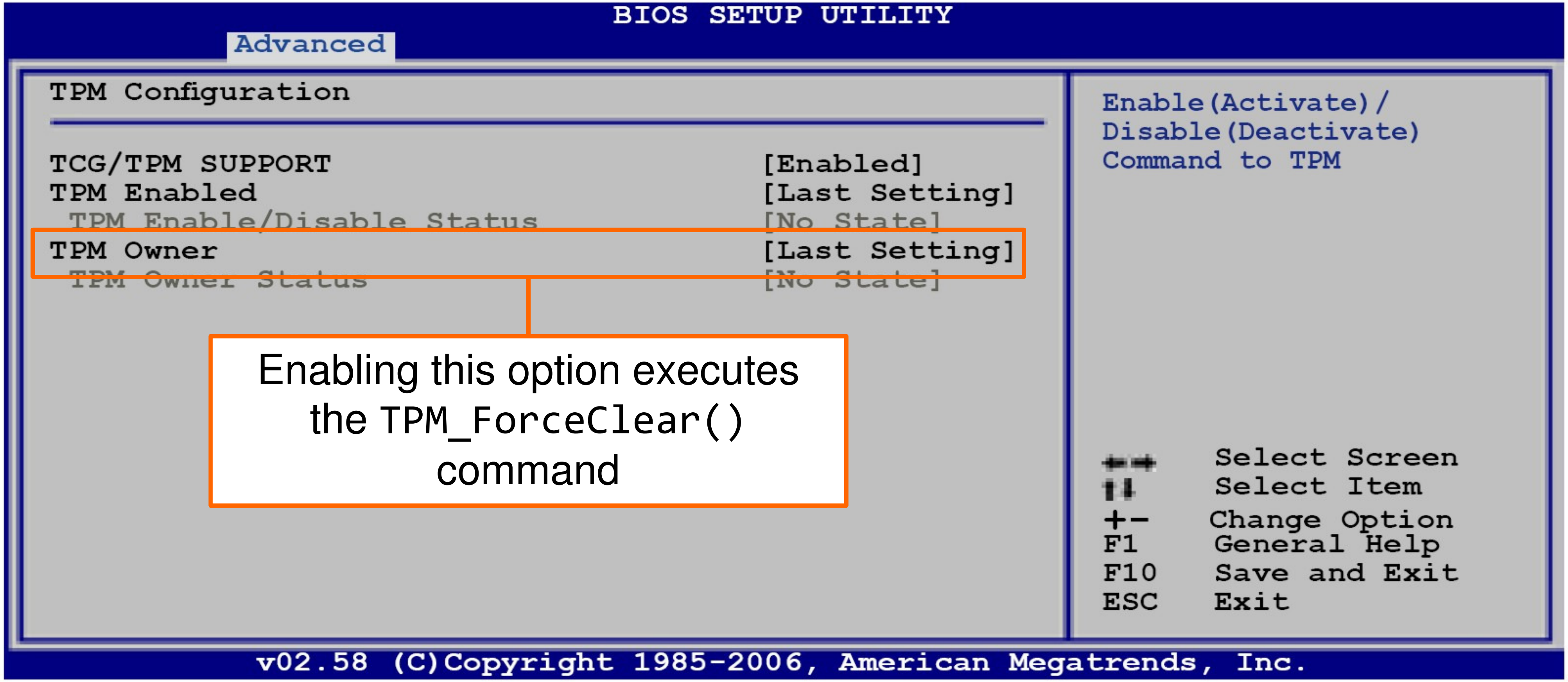
( ) ← TPM_ForceClear()

```
if physical presence is not asserted
   return error;
else
   unload all currently loaded keys;
   delete A_Owner;
   delete SRK;
   set all owner-related internal variables to their defaults;
   terminate all currently open sessions;
end if;
```

Note

- This command is authorized by asserting physical presence (e.g., via a pin at the TPM wired to a button at the platform)

TPM Owner
→ Asserting Physical Presence via BIOS



Enabling this option executes the `TPM_ForceClear()` command

A remote adversary cannot access the BIOS.
A local adversary with access to the BIOS is able to disable the TPM and even to delete the TPM Owner without the need to know any secret!

Content

Authentication to the TPM
→ Accessing Protected Entities

- **Typically requires authorization**
  - User must prove knowledge of an authorization secret
    - e.g., authorization secret = digest of a passphrase
- **Authorization secrets are set by TPM users and stored inside shielded locations**
  - e.g., during the process of creating a key, a user sets a passphrase required for authorizing later use of the key.
  - TPM stores the passphrase together with the key in a shielded location.

Authentication to the TPM
→ TPM Authorization Protocols (AP)

- **Authentication of commands and their parameters**
  - Provide assurance that the command, its parameters and the corresponding response of the TPM have not been modified during their transmission to or from the TPM
- **TPM basically supports two authorization protocols**
  - OSAP (Object Specific Authorization Protocol)
  - OIAP (Object Independent Authorization Protocol)
- **TPM must support at least two parallel authorization sessions**
  - Some TPM commands require two authorizations
    - e.g., command for unsealing data (see sealing)

Authentication to the TPM
→ Basic Functionality of TPM's APs

AuthSecret is transmitted to the TPM during entity creation

TPM

knows AuthSecret *for protected entity E*

- Generate nonce $Nonce_{TPM}$
- Initialize authorization session S referenced by session $Handle_S$ (session identifier)

AuthSecret has been chosen by the TPM user during entity creation (e.g., as a hash of a passphrase)

User U

knows AuthSecret *for protected Entity E* (*referenced by* $Handle_E$)

InitAuthProt() ←

$Handle_S$, $Nonce_{TPM}$ →

- Generate $Nonce_U$
- Compute $AuthData_U$ (authenticating the identifier `Command` for the command to be executed and its input Input)

Command(Input , $Handle_E$) , $Handle_S$, $Nonce_U$, $AuthData_U$ ←

- Verifies $AuthData_U$ and aborts protocol on error
- Execute command

Output ← Command(Input, $Handle_E$)

- Compute $AuthData_{TPM}$ (authenticating the output Output of command `Command()`)

if o.k., TPM can be assured that call is
- fresh (no replay)
- authentic (has not been modified)
- performed by an authorized user

Output , $AuthData_{TPM}$ →

- Verifies $AuthData_{TPM}$ and aborts protocol on error

if o.k., user can be assured that the response
- is fresh (no replay)
- is authentic (has not been modified)
- has been sent by the TPM

$AuthData_U$ ← HMAC(AuthSecret , SHA-1(Command , Input) , $Nonce_{TPM}$, $Nonce_U$)

$AuthData_{TPM}$ ← HMAC(AuthSecret , SH-A-1(Command , Output) , $Nonce_U$, …)

Authentication to the TPM
→ OIAP vs. OSAP

OIAP

Object Independent Authorization Protocol

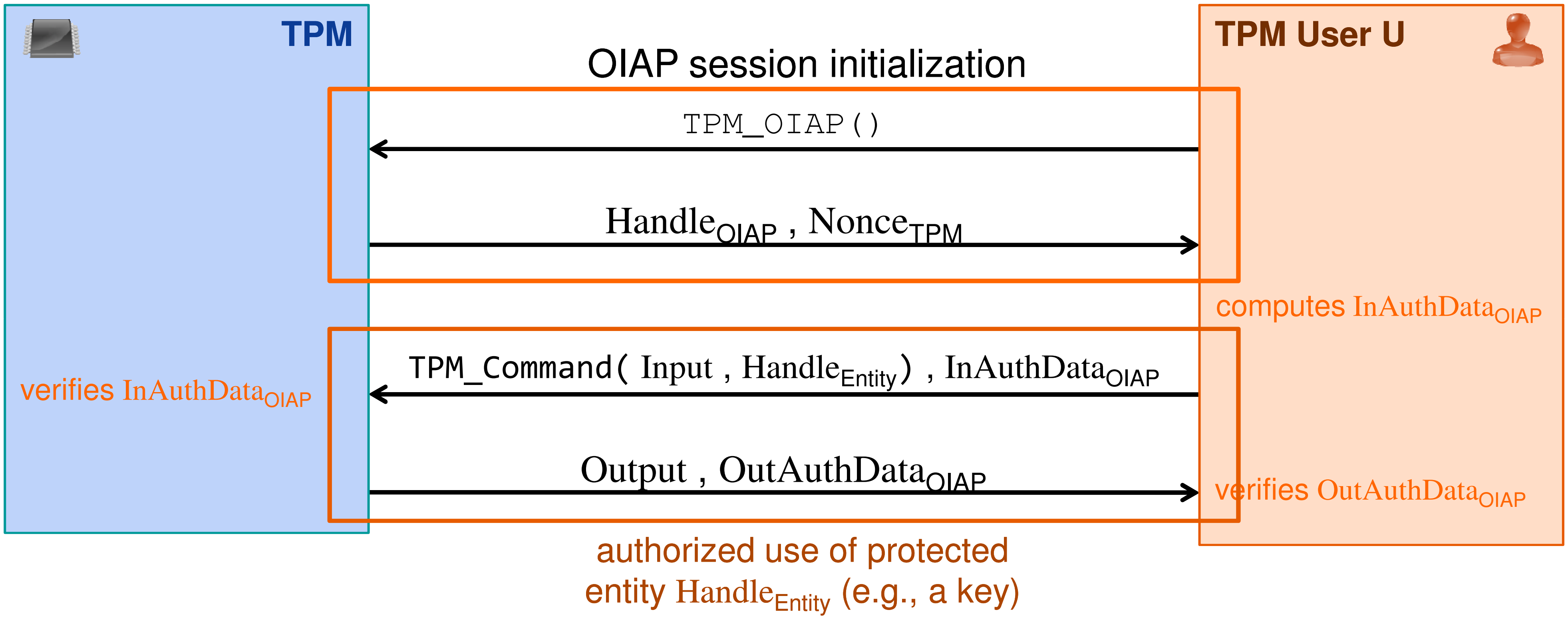
- **Properties**
  - Can authorize use of multiple different protected entities with multiple commands
  - Only one setup necessary for many different entities to be authorized
  - **No session key establishment**
- **Mainly used for**
  - Authorization of using protected entities without the need for a shared session secret/key

OSAP

Object Specific Authorization Protocol

- **Properties**
  - Can authorize use of a single protected entity with multiple commands
  - One setup required for each entity to be authorized
  - Establishes an ephemeral shared session secret, which can be used as a cryptographic key
- **Mainly used for**
  - Setting or changing authorization data for protected entities

Authentication to the TPM
→ OIAP Protocol Session



Nonce is chosen by user U

$$InAuthDigest_{OIAP} = HMAC(AuthSecret_{Entity} , \text{SHA-1}(\texttt{TPM\_Command} , Input) , Nonce_{TPM} , Nonce)$$

$$InAuthData_{OIAP} = (Handle_{OIAP} , Nonce , InAuthDigest_{OIAP})$$

$$OutAuthDigest_{OIAP} \leftarrow HMAC(AuthSecret_{Entity} , \text{SHA-1}(\texttt{TPM\_Command} , Ouput) , Nonce_{TPM,2} , Nonce)$$

$$OutAuthData_{OIAP} \leftarrow (Nonce_{TPM,2} , OutAuthDigest_{OIAP})$$

Authentication to the TPM
→ Initialization of OIAP Session

($Handle_{OIAP}$, $Nonce_{TPM}$) ← TPM_OIAP()

```
if maximum number of authorization sessions has been reached then
   return error;
else
   create Handle_OIAP;
   Nonce_TPM ← RNG( 20 );
   store ( Handle_OIAP , Nonce_TPM ) in volatile memory;
   return ( Handle_OIAP , Nonce_TPM );
end if;
```

Notes

- $Handle_{OIAP}$ is an identifier for the new OIAP session
- TPM must ensure that no other active auth. session is referenced by $Handle_{OIAP}$
- S_{OIAP} represents the data associated with an OIAP session

Verification of an OIAP Session

$InAuthDigest_{OIAP}$ = HMAC($AuthSecret_{Entity}$, SHA-1(`TPM_Command` , Input) , $Nonce_{TPM}$, Nonce)

$InAuthData_{OIAP}$ = ($Handle_{OIAP}$, Nonce , $InAuthDigest_{OIAP}$)

(Output , $OutAuthData_{OIAP}$) ← TPM_Command(Input , $Handle_{Entity}$) , $InAuthData_{OIAP}$

```
if OIAPVerify( InAuthData_OIAP , Handle_Entity ) ≠ ok then
   return error;
else
   Output ← TPM_Command(Input , Handle_Entity);
   Nonce_TPM,2 ← RNG( 20 );
   OutAuthDigest_OIAP ← HMAC( AuthSecret_Entity ,
         SHA-1( TPM_Command , Ouput ) , Nonce_TPM,2 , Nonce );
   OutAuthData_OIAP ← ( Nonce_TPM,2 , OutAuthDigest_OIAP );
   return ( Output , OutAuthData_OIAP ) ;
end if;
```

ind ← OIAPVerify($InAuthData_{OIAP}$, $Handle_{Entity}$)

```
if Handle_OIAP does not refer to an open OIAP session then
   return error;
else
   obtain AuthSecret_Entity from entity referred to by Handle_Entity;
   return Verify( InAuthDigest_OIAP , AuthSecret_Entity );
end if;
```

Perquisites

- `TPM_OIAP()` must have been executed before
- The protected entity (e.g., key) to be authorized must have been previously loaded into the TPM. The command that loaded the entity returns an identifier $Handle_{Entity}$ for that entity

Notes

- `TPM_Command()` may be any command that requires authorization via OIAP
- Verify() re-computes $InAuthDigest_{OIAP}$ using $AuthSecret_{Entity}$ stored with the entity to be authorized and compares it to $InAuthDigest_{OIAP}$

Authentication to the TPM
→ Verification of an OIAP Session

(Output , $OutAuthData_{OIAP}$) ← `TPM_Command`(Input , $Handle_{Entity}$) , $InAuthData_{OIAP}$

if OIAPVerify($InAuthData_{OIAP}$, $Handle_{Entity}$) ≠ ok then
  return error;
else
  Output ← `TPM_Command`(Input , $Handle_{Entity}$);
  $Nonce_{TPM,2}$ ← RNG(20);
  $OutAuthDigest_{OIAP}$ ← HMAC($AuthSecret_{Entity}$,
    SHA-1(`TPM_Command` , Ouput) , $Nonce_{TPM,2}$, Nonce);
  $OutAuthData_{OIAP}$ ← ($Nonce_{TPM,2}$, $OutAuthDigest_{OIAP}$);
  return (Output , $OutAuthData_{OIAP}$) ;
end if;

authorized use of Entity

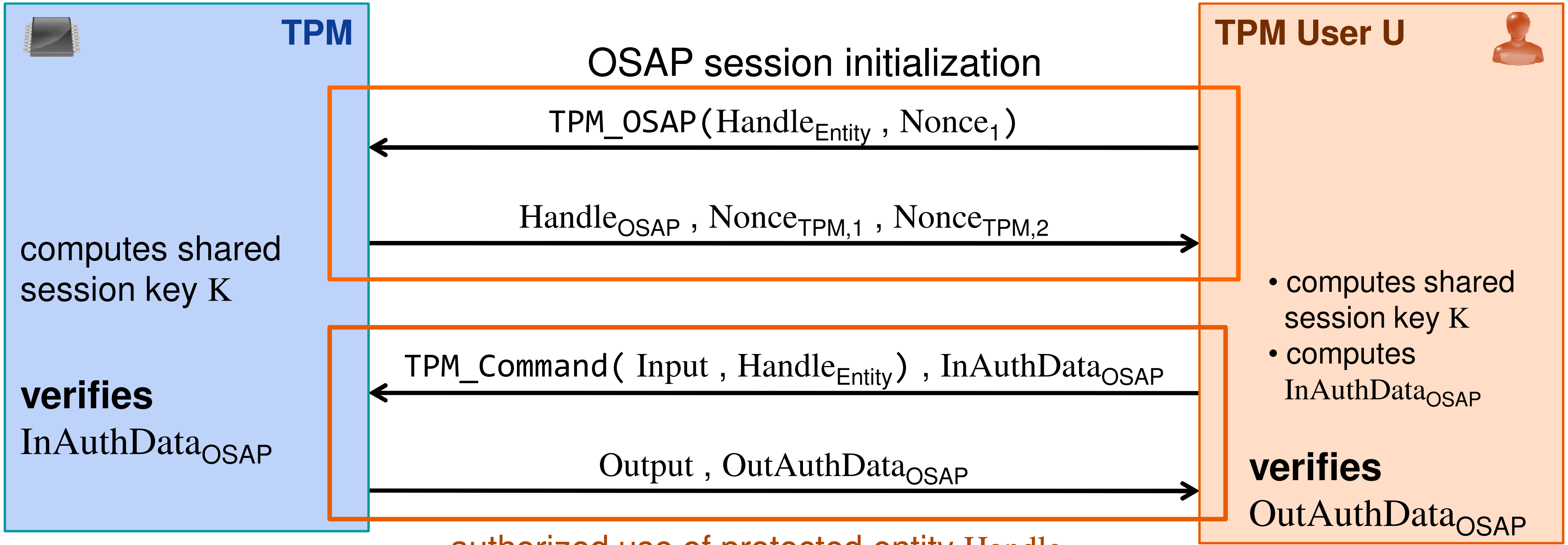
authenticator for TPM's response

verification of authorization

ind ← OIAPVerify($InAuthData_{OIAP}$, $Handle_{Entity}$)

if $Handle_{OIAP}$ does not refer to an open OIAP session then
  return error;
else
  obtain $AuthSecret_{Entity}$ from entity referred to by $Handle_{Entity}$;
  return Verify($InAuthDigest_{OIAP}$, $AuthSecret_{Entity}$);
end if;

Authentication to the TPM
→ OASP Protocol Session



authorized use of protected entity Handle_{Entity} (e.g., key) and shared session secret K

Nonce is chosen by user U

$$K \leftarrow \text{HMAC}(\,\mathrm{AuthSecret}_{Entity}\,,\ \mathrm{Nonce}_{TPM,2}\,,\ \mathrm{Nonce}_1\,)$$

$$\mathrm{InAuthDigest}_{OSAP} = \text{HMAC}(\,K\,,\ \text{SHA-1}(\,\texttt{TPM\_Command}\,,\ \mathrm{Input}\,)\,,\ \mathrm{Nonce}_{TPM,1}\,,\ \mathrm{Nonce}_2\,)$$

$$\mathrm{InAuthData}_{OSAP} = (\,\mathrm{Handle}_{OSAP}\,,\ \mathrm{Nonce}_2\,,\ \mathrm{InAuthDigest}_{OSAP}\,)$$

$$\mathrm{OutAuthDigest}_{OSAP} \leftarrow \text{HMAC}(\,K\,,\ \text{SHA-1}(\,\texttt{TPM\_Command}\,,\ \mathrm{Ouput}\,)\,,\ \mathrm{Nonce}_{TPM,3}\,,\ \mathrm{Nonce}_2\,)$$

$$\mathrm{OutAuthData}_{OSAP} \leftarrow (\,\mathrm{Nonce}_{TPM,3}\,,\ \mathrm{OutAuthDigest}_{OSAP}\,)$$

Authentication to the TPM
→ Initialization of OSAP Session

$(\text{Handle}_{OSAP} , \text{Nonce}_{TPM,1} , \text{Nonce}_{TPM,2}) \leftarrow \text{TPM\_OSAP}(\text{Handle}_{Entity} , \text{Nonce}_1)$

```
if maximum number of authorization sessions has been reached then
   return error;
else
   create Handle_OSAP;
   Nonce_TPM,1 ← RNG();
   Nonce_TPM,2 ← RNG();
   K ← HMAC( AuthSecret_Entity , Nonce_TPM,2 , Nonce_1 );
   store ( Handle_OSAP , Handle_Entity , K , Nonce_TPM,1 , Nonce_TPM,2 ) in
      volatile memory;
   return ( Handle_OSAP , Nonce_TPM,1 , Nonce_TPM,2 );
end if;
```

Prequisites

- The protected entity (e.g., key) to be authorized must have been previously loaded into the TPM. The command that loaded the entity returns an identifier Handle_{Entity} for that entity

Notes

- Handle_{OSAP} is identifier for the new OSAP session
- TPM must ensure that no other active auth. session is referenced by Handle_{OSAP}

Authentication to the TPM
→ Initialization of OSAP Session

(Handle_{OSAP} , $\text{Nonce}_{TPM,1}$, $\text{Nonce}_{TPM,2}$) ← `TPM_OSAP`(Handle_{Entity} , Nonce_1)

if maximum number of authorization sessions has been reached then
  return error;
else
  create Handle_{OSAP};
  $\text{Nonce}_{TPM,1} \leftarrow$ RNG();
  $\text{Nonce}_{TPM,2} \leftarrow$ RNG();
  K ← HMAC($\text{AuthSecret}_{Entity}$, $\text{Nonce}_{TPM,2}$, Nonce_1);
  store (Handle_{OSAP} , Handle_{Entity} , K , $\text{Nonce}_{TPM,1}$, $\text{Nonce}_{TPM,2}$) in
    volatile memory;
  return (Handle_{OSAP} , $\text{Nonce}_{TPM,1}$, $\text{Nonce}_{TPM,2}$);
end if;

creation of shared session secret

Notes

- Handle_{OSAP} is identifier for the new OSAP session
- TPM must ensure that no other active auth. session is referenced by Handle_{OSAP}

Verification of an OSAP Session

$K \leftarrow$ HMAC($AuthSecret_{Entity}$, $Nonce_{TPM,2}$, $Nonce_1$)

$InAuthData_{OSAP}$ = ($Handle_{OSAP}$, $Nonce_2$, $InAuthDigest_{OSAP}$)

$InAuthDigest_{OSAP}$ = HMAC(K , SHA-1(`TPM_Command` , Input) , $Nonce_{TPM,1}$, $Nonce_2$)

(Output , $OutAuthData_{OSAP}$) ← TPM_Command(Input , $Handle_{Entity}$) , $InAuthData_{OSAP}$

```
if OSAPVerify( InAuthData_OSAP , Handle_Entity ) ≠ ok then
   return error;
else
   Output ← TPM_Command(Input , Handle_Entity , K);
   Nonce_TPM,3 ← RNG( 20 );
   OutAuthDigest_OSAP ← HMAC( K ,
         SHA-1( TPM_Command , Ouput ) , Nonce_TPM,3 , Nonce_2 );
   OutAuthData_OSAP ← ( Nonce_TPM,3 , OutAuthDigest_OSAP );
   return ( Output , OutAuthData_OSAP ) ;
end if;
```

ind ← OSAPVerify($InAuthData_{OSAP}$, $Handle_{Entity}$)

```
if Handle_OSAP does not refer to an open OSAP session then
   return error;
else
   obtain AuthSecret_Entity from entity referred to by Handle_Entity;
   return Verify( InAuthDigest_OSAP , AuthSecret_Entity );
end if;
```

Perquisites

- `TPM_OSAP()` must have been executed before
- Protected entity (e.g., key) to be authorized must have been previously loaded into the TPM
- $Handle_{Entity}$ refers to entity to be authorized

Notes

- `TPM_Command()` may be any command supporting authorization via OSAP
- Verify() re-computes $InAuthDigest_{OSAP}$ using $AuthSecret_{Entity}$ stored with the entity to be authorized and compares it to $InAuthDigest_{OSAP}$

Authentication to the TPM
→ Verification of an OSAP Session

$(\text{Output} , \text{OutAuthData}_{OSAP}) \leftarrow$ `TPM_Command`$(\text{Input} , \text{Handle}_{Entity}) , \text{InAuthData}_{OSAP}$

```
if OSAPVerify( InAuthData_OSAP , Handle_Entity ) ≠ ok then
   return error;
else
   Output ← TPM_Command(Input , Handle_Entity , K);
   Nonce_TPM,3 ← RNG( 20 );
   OutAuthDigest_OSAP ← HMAC( K ,
         SHA-1( TPM_Command , Ouput ) , Nonce_TPM,3 , Nonce_2 );
   OutAuthData_OSAP ← ( Nonce_TPM,3 , OutAuthDigest_OSAP );
   return ( Output , OutAuthData_OSAP ) ;
end if;
```

authorized use of Entity and session secret K

authenticator for TPM's response

verification of authorization

$\text{ind} \leftarrow \text{OSAPVerify}(\text{InAuthData}_{OSAP} , \text{Handle}_{Entity})$

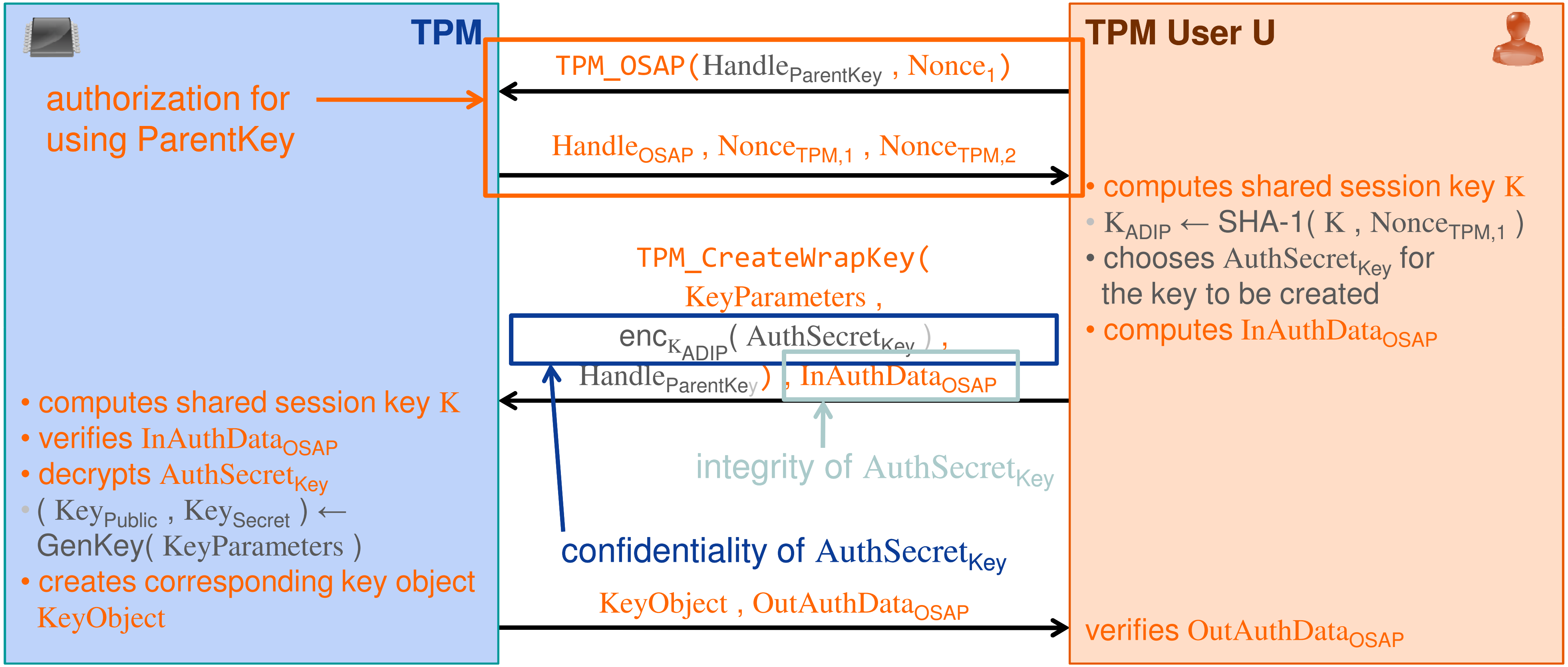
```
if Handle_OSAP does not refer to an open OSAP session then
   return error;
else
   obtain AuthSecret_Entity from entity referred to by Handle_Entity;
   return Verify( InAuthDigest_OSAP , AuthSecret_Entity );
end if;
```

Authentication to the TPM
→ Insertion and Change of Auth Secrets

- **Authorization Data Insertion Protocol (ADIP)**
  - Used to set authorization secret for protected entities
  - Extension of OSAP to protect the authorization secret
    - Confidentiality: Encryption with key derived from OSAP session
    - Integrity: HMAC of OSAP session ($InAuthData_{OSAP}$)
    - Authorization for using the corresponding parent key: OSAP

- **Authorization Data Change Protocol (ADCP)**
  - Used to change authorization secrets for protected entities
  - Defines how to use ADIP and OIAP/OSAP to protect new authorization secret and to authorize change
    - Confidentiality & integrity: ADIP
    - Authorization for access to the new protected entity: OSAP
    - Authorization for changing authorization secret: OIAP or OSAP

Authentication to the TPM
→ ADIP Example: Creation of a new Key



$$\text{KeyObject} = (\ \text{KeyParameters}\ ,\ \text{Key}_{Public}\ ,\ \text{enc}_{ParentKey}(\ \text{AuthSecret}_{Key}\ ,\ \text{Key}_{Secret}\ )\ )$$

ADIP extensions

Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- **Some More TPM Details**
  - **Migration of TPM Keys**
- Summary

Migration of TPM Keys
→ Overview of Maintenance

- **Transfers all TPM-protected data to another TPM**
  - Necessary when exchanging a (defective) subsystem that contains a TPM without loosing non-migratable data
- **Different from backup/migration**
  - Maintenance can also migrate data that cannot be migrated using the TPM's migration functionality
  - **Requires intervention of the subsystem's manufacturer**
- **Vendor-specific feature**
  - Maintenance commands are not exactly specified by TCG
- **Optional feature, but if implemented**
  - All specified maintenance capabilities are mandatory
  - No other maintenance capabilities must be implemented

Migration of TPM Keys
→ Specified Security Requirements

internet security.

- **Confidentiality and cloning: Data to be migrated must not be**
  - accessible by more than one TPM at a time nor
  - exposed to third parties including the manufacturer
- **Policy conformance: Maintenance must require**
  - Source and target platforms are from the same manufacturer and model
  - Active participation of the TPM Owner
- **Migration of non-migratable data requires cooperation of**
  - owner of the non-migratable data
    - e.g., to authorize moving his sensitive data to another platform
  - manufacturer of the subsystem
    - e.g., must revoke old Endorsement Credential and guarantee destruction of old TPM (which still contains the migrated data)

Migration of TPM Keys
→ Interface to Perform Maintenance I

- **TPM_CreateMaintenanceArchive**
  - Creates maintenance archive encrypted with
    - Symmetric key derived from TPM Owner's authorization secret or the TPM's random number generator (TPM Owner decides)
    - Subsystem manufacturer's public maintenance key
  - Requires authorization by the TPM Owner
- **TPM_LoadMaintenanceArchive**
  - Loads and restores a maintenance archive
    - All current TPM-protected data will be overwritten with the data from the maintenance archive
  - Must be authorized by the TPM Owner

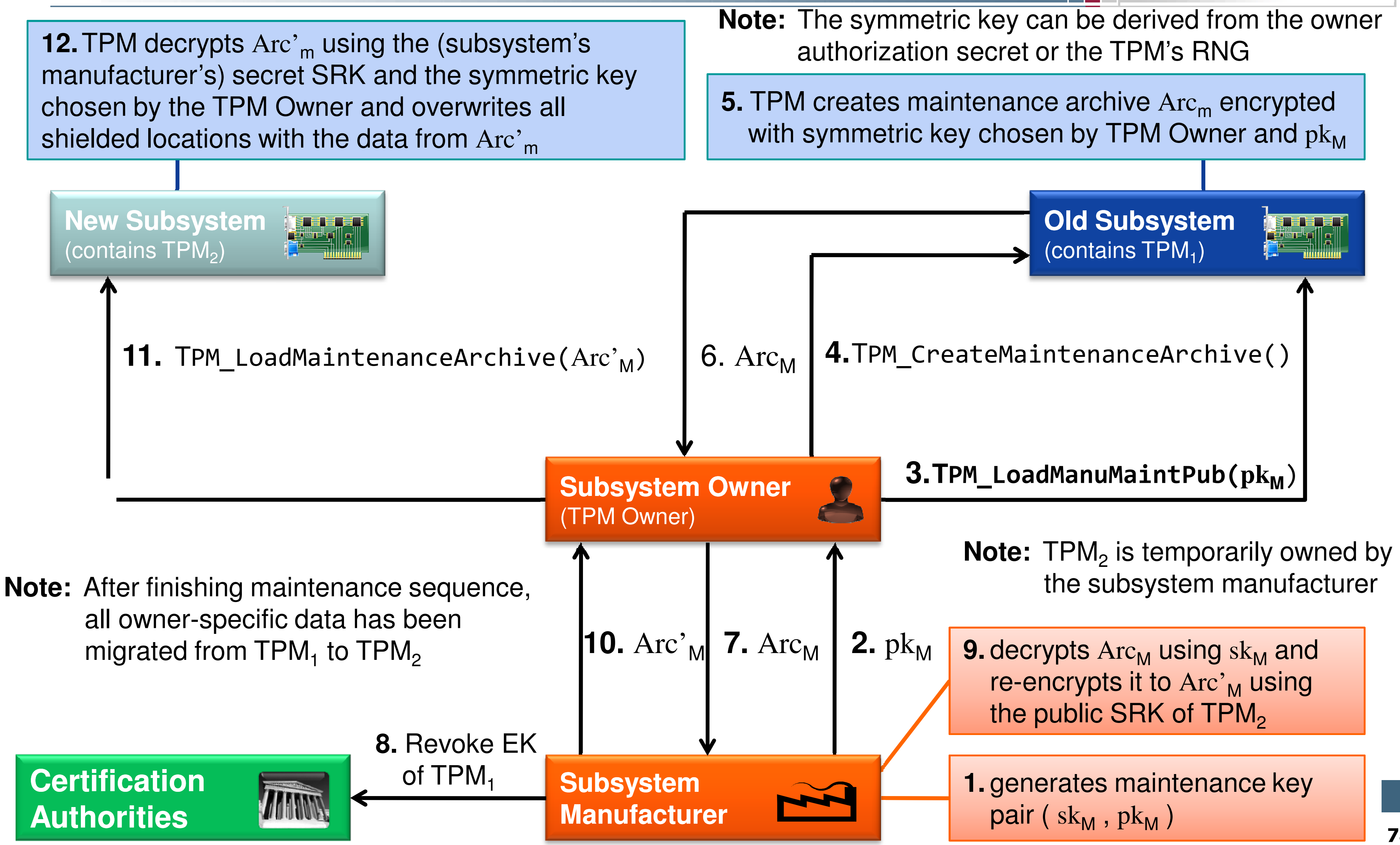
Migration of TPM Keys
→ Interface to Configure Maintenance II

- **TPM_KillMaintenanceFeature**
  - Disables all maintenance commands until a new TPM Owner is set
  - Must be authorized by the current TPM Owner

- **TPM_LoadManuMaintPub**
  - Installs a manufacturer's public maintenance key into TPM
  - Usually done by the subsystem manufacturer before delivery

- **TPM_ReadManuMaintPub**
  - Reads manufacturer's public maintenance key from TPM

Typical Maintenance Sequence



Content

- Aim and outcomes of this lecture
- Overview of the idea of TPM
- Terminology and Assumption
- Identities
- TPM Keys and Keys´ Properties
- TPM Key Types
- Some More TPM Details
- **Summary**

Trusted Platform Module (TPM)
→ Summary

- The TPM is the **anchor** for Trusted Computing

- The TPM is a **passive security controller** with
  - cryptographic functions
  - a secure storage and
  - with **Platform Configuration Registers (PCR)**
  - **…**

- Has a **complex key hierarchy** and different types of keys with additional properties

- Offers a lot of intelligent functions (protocols) with help together with additional components (e.g. TCB) to **measure and prove the integrity** of IT systems

Trusted Computing
→ Trusted Platform Module (TPM)

Thank you for your attention!
Questions?

Prof. Dr.
Norbert Pohlmann

Institute for Internet Security - if(is)
University of Applied Sciences Gelsenkirchen
http://www.internet-sicherheit.de

Trusted Platform Module (TPM)
→ Literature

- [1] **Prof-. Dr.-Ing. Ahmad Reza Sadeghi**
  http://www.trust.rub.de/home/
- [2] N. Pohlmann, A.-R. Sadeghi, C. Stüble: "European Multilateral Secure Computing Base", DuD Datenschutz und Datensicherheit – Recht und Sicherheit in Informationsverarbeitung und Kommunikation, Vieweg Verlag, 09/2004
- [3] N. Pohlmann, H. Reimer: „Trusted Computing – eine Einführung", in "Trusted Computing - Ein Weg zu neuen IT-Sicherheitsarchitekturen", Hrsg.: N. Pohlmann, H. Reimer; Vieweg-Verlag, Wiesbaden 2008
- [4] M. Linnemann, N. Pohlmann: "An Airbag for the Operating System – A Pipedream?", ENISA Quarterly Vol. 3, No. 3, July-Sept 2007

Links:

Institute for Internet Security:
http://www.internet-sicherheit.de/forschung/aktuelle-projekte/trusted-computing/