

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>



if(is)
internet security.

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**

- **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:

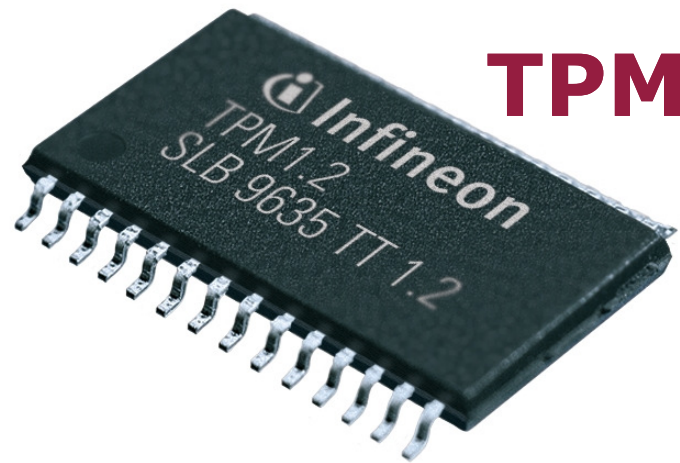
- Understand what is meant by the Trusted Platform Module (TPM).
- Know some of the functions of the Trusted Platform Module (TPM).
- Know what the protocols of the Trusted Platform Module (TPM) look like.
- Understand the capabilities and limitations of the Trusted Platform Module (TPM).

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

Trusted Platform Module (TPM)

→ Overview (1/4)



TPM

The Safe

on our Motherboard!



Trusted Platform Module (TPM)

→ Overview (2/4)

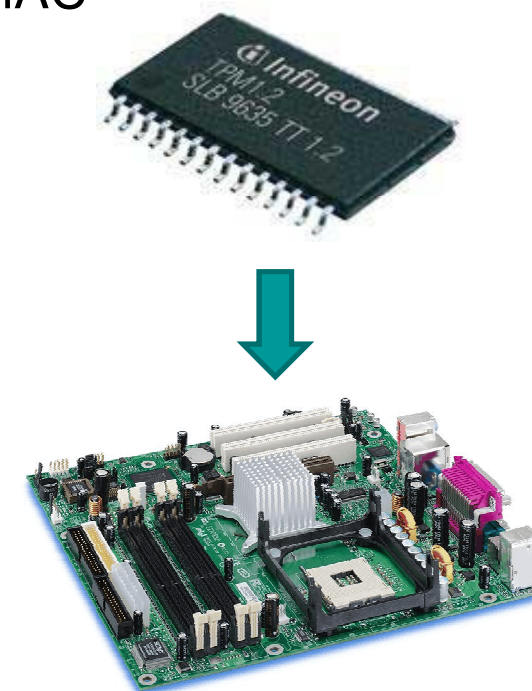
The Trusted Platform Module (TPM) is

- 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)

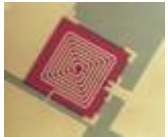
Common misconceptions

- 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

Electro Magnetic Analysis (EMA)



Differential Fault Attack (DFA)



Alpha Particle Penetration



Timing Analysis



Global And Local Optical Attacks



Contrast Etching / Decoration



Atomic Force Microscopy (AFM)



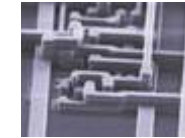
Differential Power Analysis (DPA)



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

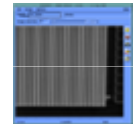
Reverse Engineering / Delayering



Probing / Forcing



Electron Microscopy



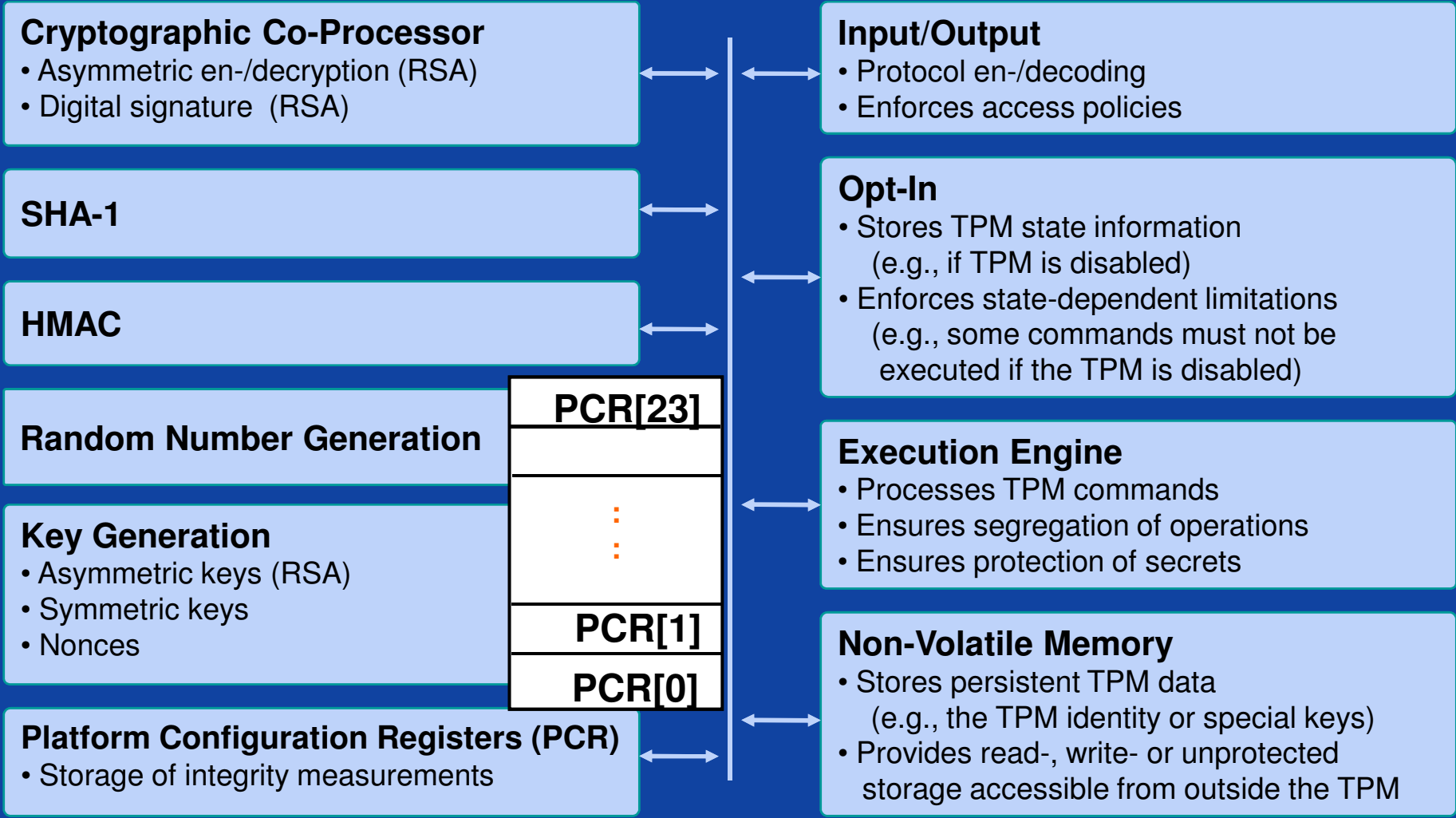
Spike / Glitch Penetration



TPM Architecture

System Interface
(e.g., LPC-Bus)

Trusted Platform Module (TPM)



TPM Internal Functions

→ Features I

- **SHA-1 engine**
 - Computes the SHA-1 digest (digest) of arbitrary data (data)
$$\text{digest} \leftarrow \text{SHA-1}(\text{data})$$
- **HMAC engine**
 - Computes the HMAC digest authDigest resulting from a secret secret and arbitrary data (data)
$$\text{authDigest} \leftarrow \text{HMAC}(\text{secret}, \text{data})$$
 - 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

$\text{rand} \leftarrow \text{RNG}(n)$

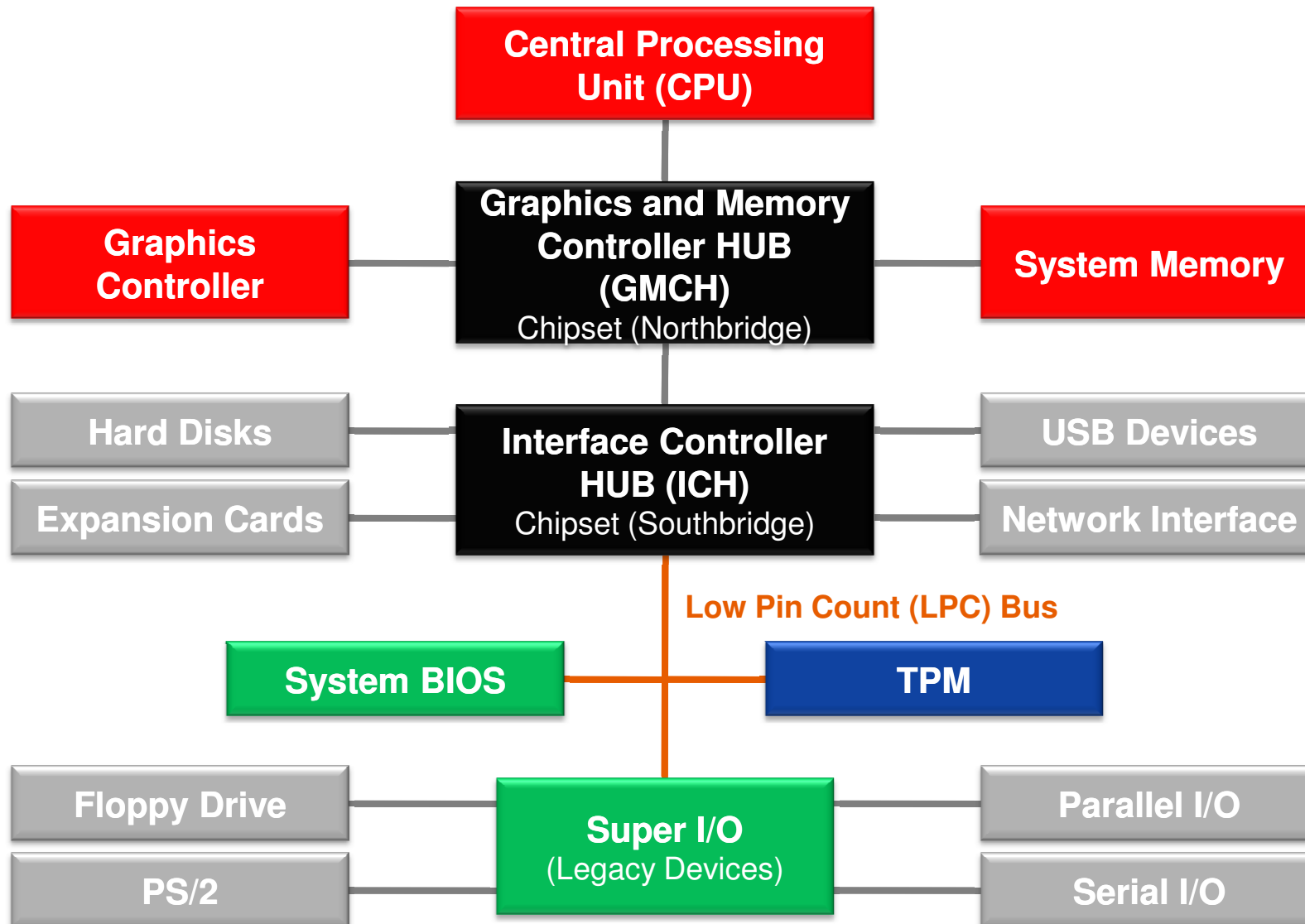
- **Mainly used to derive 20 random bytes**
 - e.g., to be used as nonce (anti-replay value)

- **Key Generation Engine**

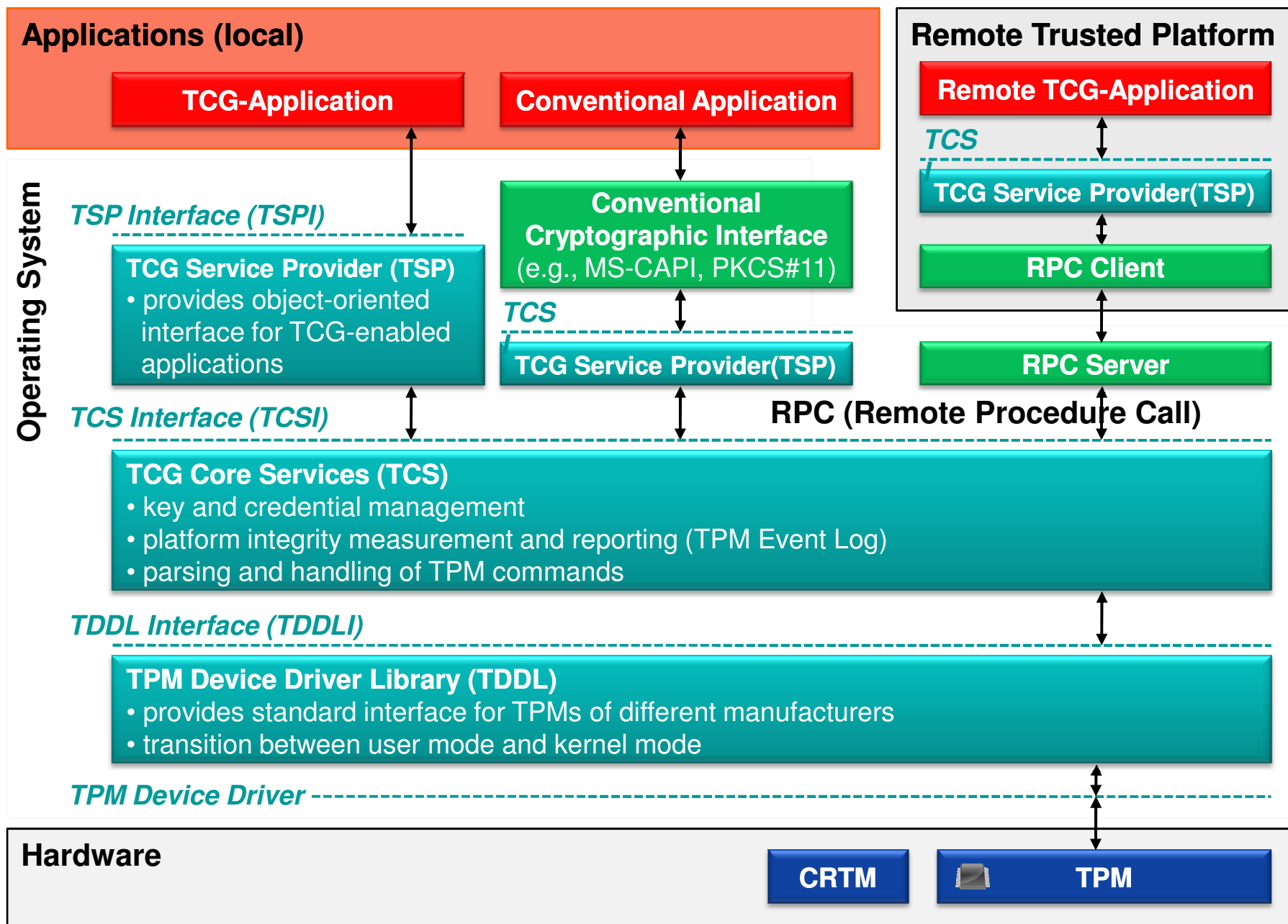
- Generates a key pair (pk, sk) according to the parameters given in par (e.g., key size, key type, etc.)

$(pk, sk) \leftarrow \text{GenKey}(par)$

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



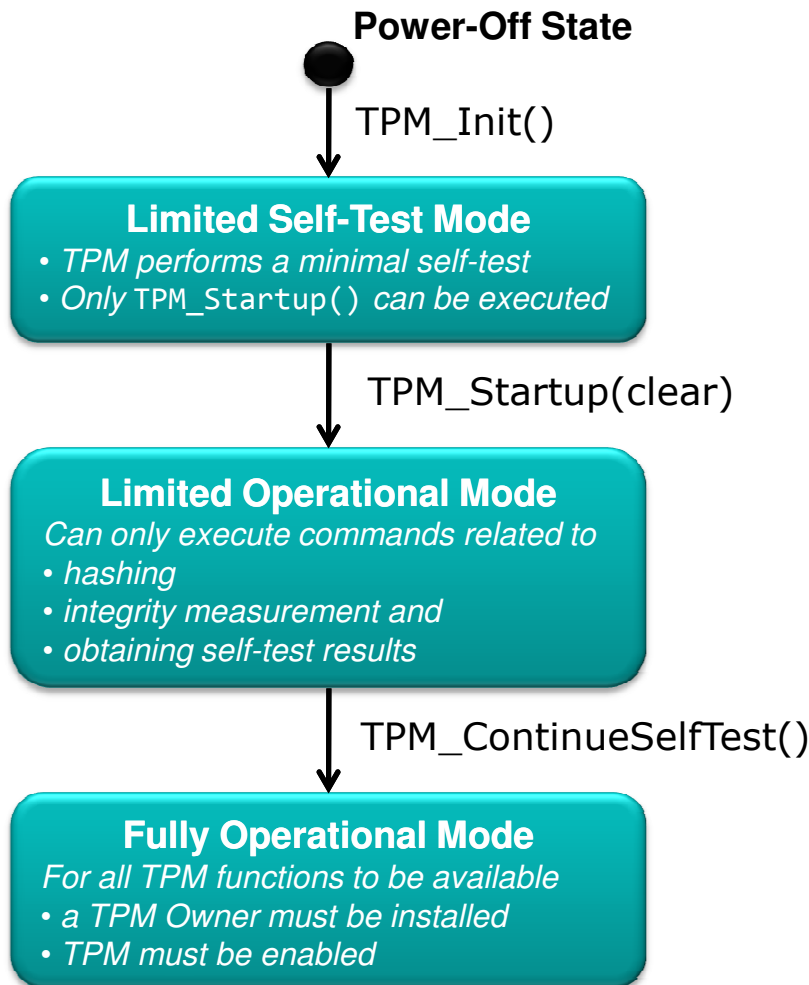
TPM Software Integration



Trusted Software Stack (TSS)
 System Services

Trusted Platform Module (TPM)

→ TPM Startup in a PC



TPM state

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 ∈ { 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

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 characteristic = digest of the software to be executed
- **Platform Configuration Registers (PCR)**
 - Shielded location to store integrity measurement values
 - Can only be extended: $PCR_{i+1} \leftarrow \text{SHA-1}(PCR_i, \text{value})$
 - 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., attester 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., [GuRuScAtPI2007] 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)

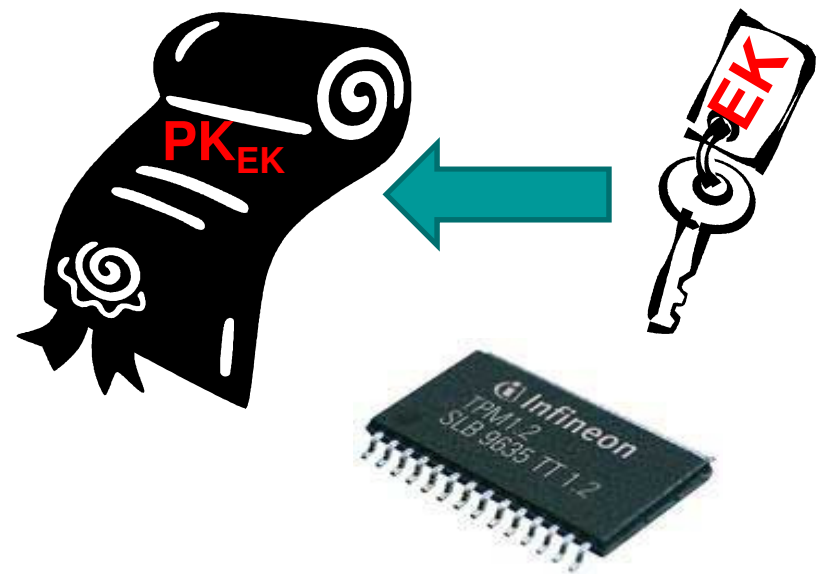
- **TPM identity represented as Endorsement Key (EK)**
- **Unique en-/decryption key pair**
 - Private key does not leave TPM
 - Public key is privacy-sensitive (since it identifies a TPM/platform)
- **Generated during manufacturing process of TPM**
 - Either in TPM or externally and then embedded into 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

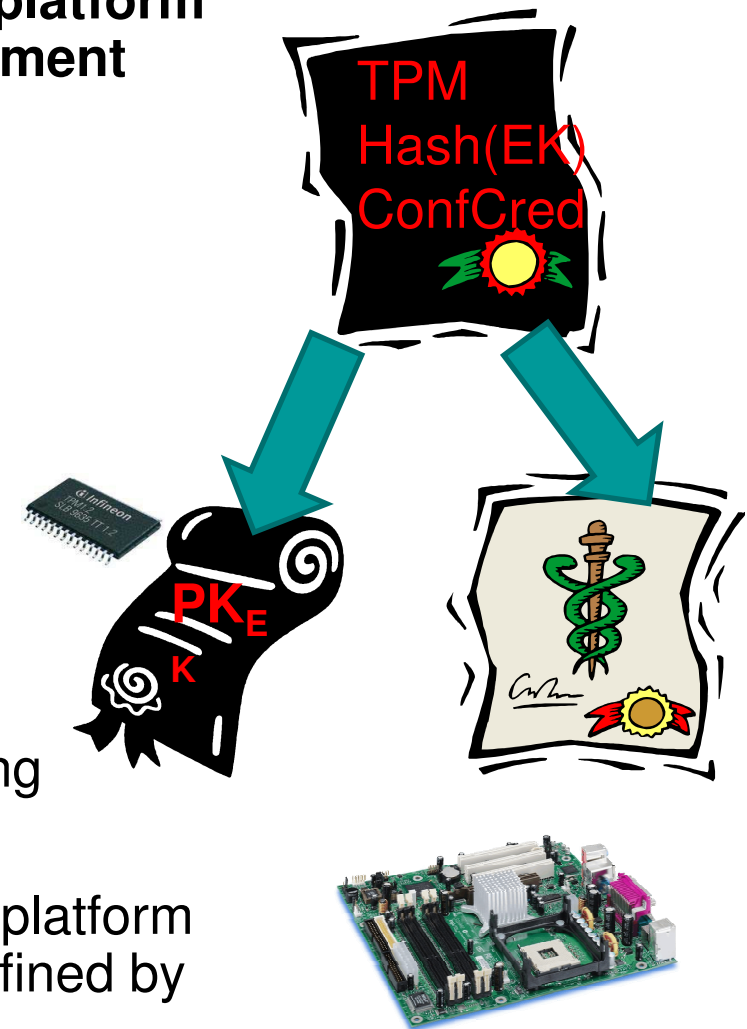
→ Platform Identity

- **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

- 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 Keys and Keys' Properties

→ Migratable and Non-Migratable Keys

- **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

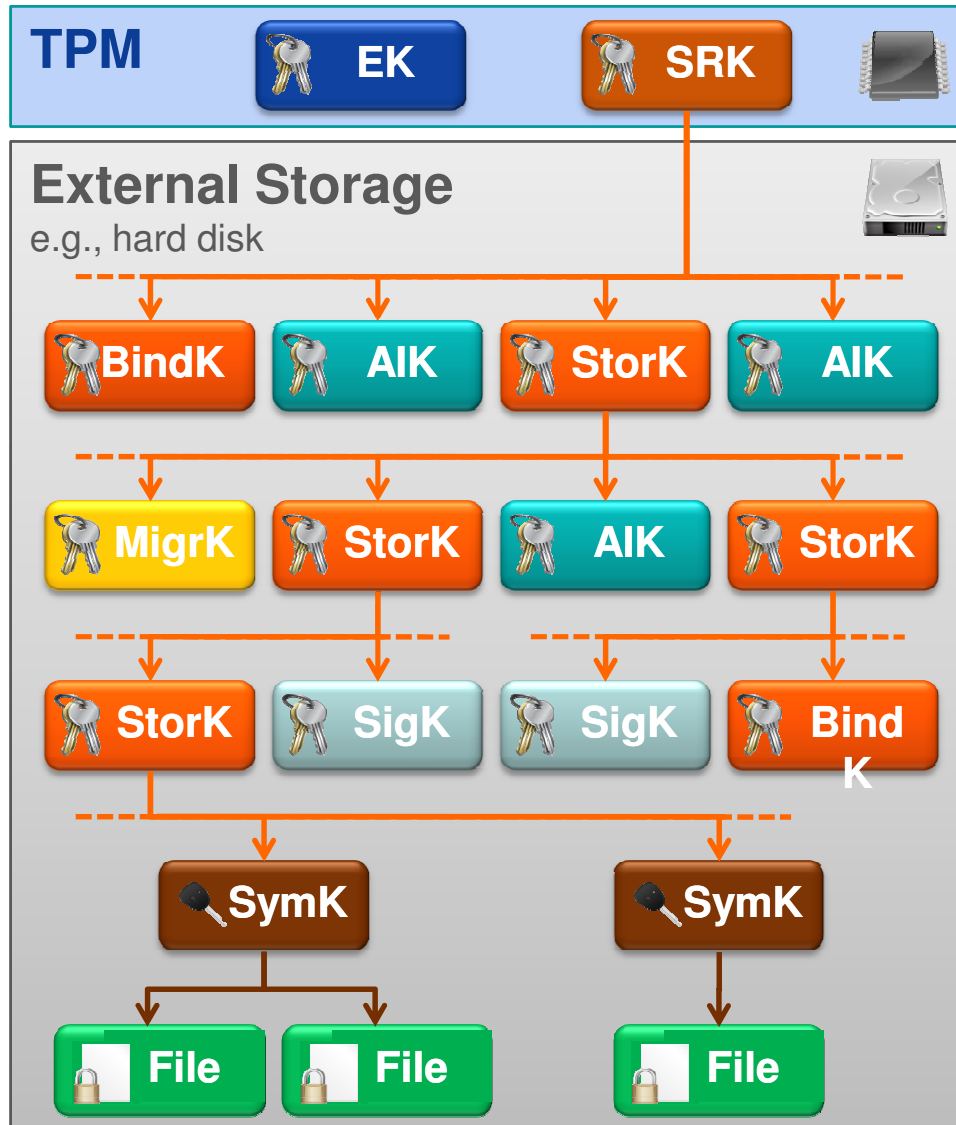
TPM Keys and Keys' Properties

→ Secure Root Storage (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

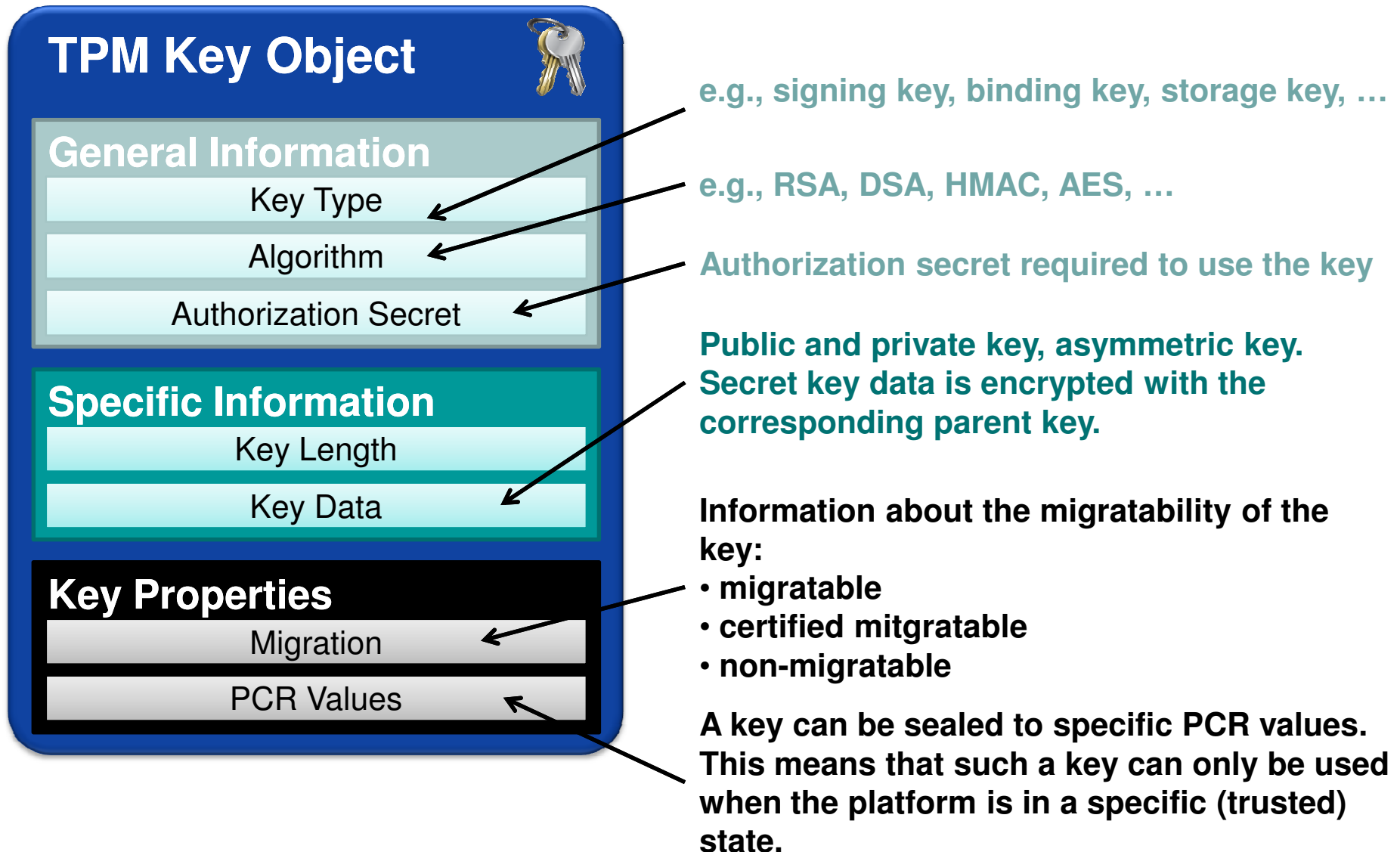
TPM Key Hierarchy



- 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



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

TPM Key Types

→ Attestation Identity Keys (AIK)

■ 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

```
( pkEK , digestEK ) ← TPM_CreateEndorsementKeyPair(Nonce , parEK)
```

```
if EK exists or then  
  return error;  
else  
  if parEK describes a storage key providing security at least  
  equivalent to RSA-2048 then  
    ( skEK , pkEK ) ← GenKey( parEK );  
    digestEK ← SHA-1( pkEK , Nonce );  
    return ( pkEK , digestEK );  
  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

```
( pkEK, digestEK, ARev ) ← TPM CreateRevocableEK(Nonce, parEK, parARev', A'Rev )
```

```
if EK exists then  
    return error;  
else  
    if parEK provides security at least equivalent to RSA-2048 then;  
        ( skEK, pkEK ) ← GenKey( parEK );  
        if parARev' = TRUE then  
            ARev ← RNG( 20 );  
        else  
            ARev ← A'Rev;  
        end if;  
        digestEK ← SHA-1( pkEK, Nonce );  
        return ( pkEK, digestEK, ARev );  
    else  
        return error;  
    end if;  
end if;
```

Prerequisites

- 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 theEK later

Note

- This is an optional command

Creating TPM Identity

→ Revoking a revocable EK

```
( ) ← TPM_RevokeTrust(ARev)
```

```
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;
```

Prerequisites

- 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

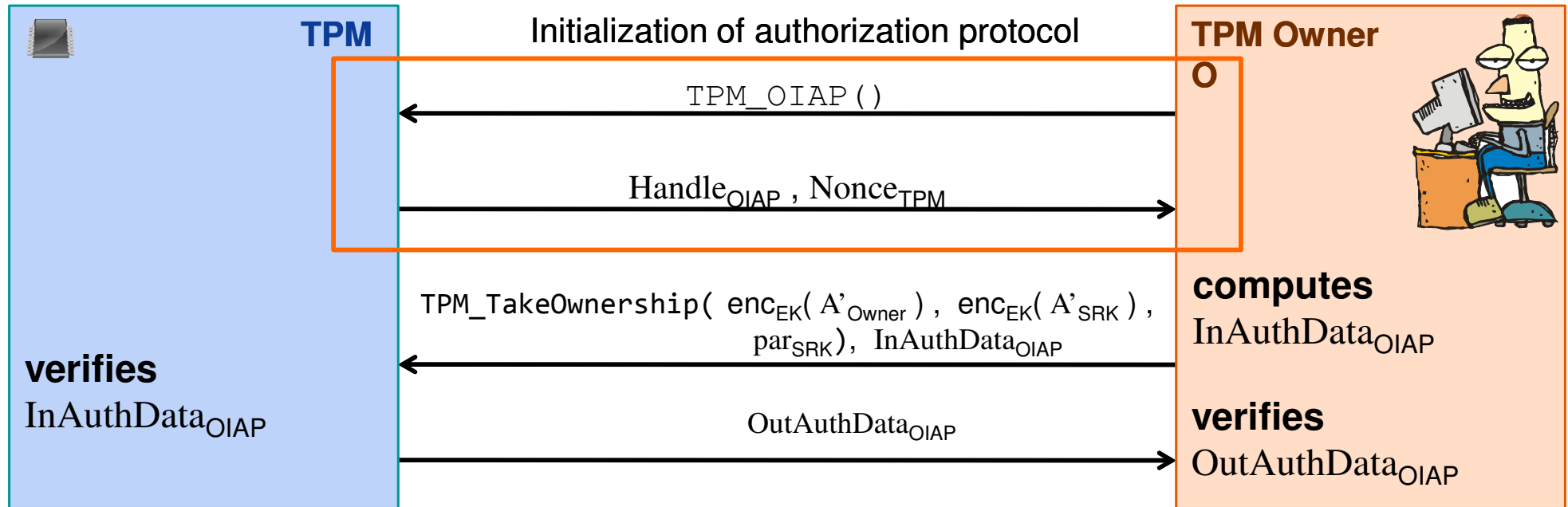
TPM Owner

→ 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



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

TPM Owner

→ TPM Interface for Taking Ownership

```
( pkSRK , OutAuthDataOIAP ) ← TPM_TakeOwnership( encEK( A'Owner ), encEK( A'SRK ), parSRK ),  
InAuthDataOIAP
```

```
if owner exists or EK is invalid  
or InAuthDataOIAP does not refer to an active OIAP session then  
    return error;  
else  
    if parSRK describes 2048-bit non-migratable RSA encryption key then  
        AOwner ← decEK( encEK( A'Owner ) );  
        store AOwner as owner authorization data in non-volatile memory;  
        ASRK ← decEK( encEK( A'SRK ) );  
        ( skSRK , pkSRK ) ← GenKey( parSRK );  
        SRK ← ( ( skSRK , pkSRK ) , ASRK );  
        store SRK in non-volatile memory;  
        initialize all owner-related TPM-internal variables;  
        compute OutAuthDataOIAP;  
        return ( pkSRK , OutAuthDataOIAP );  
    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

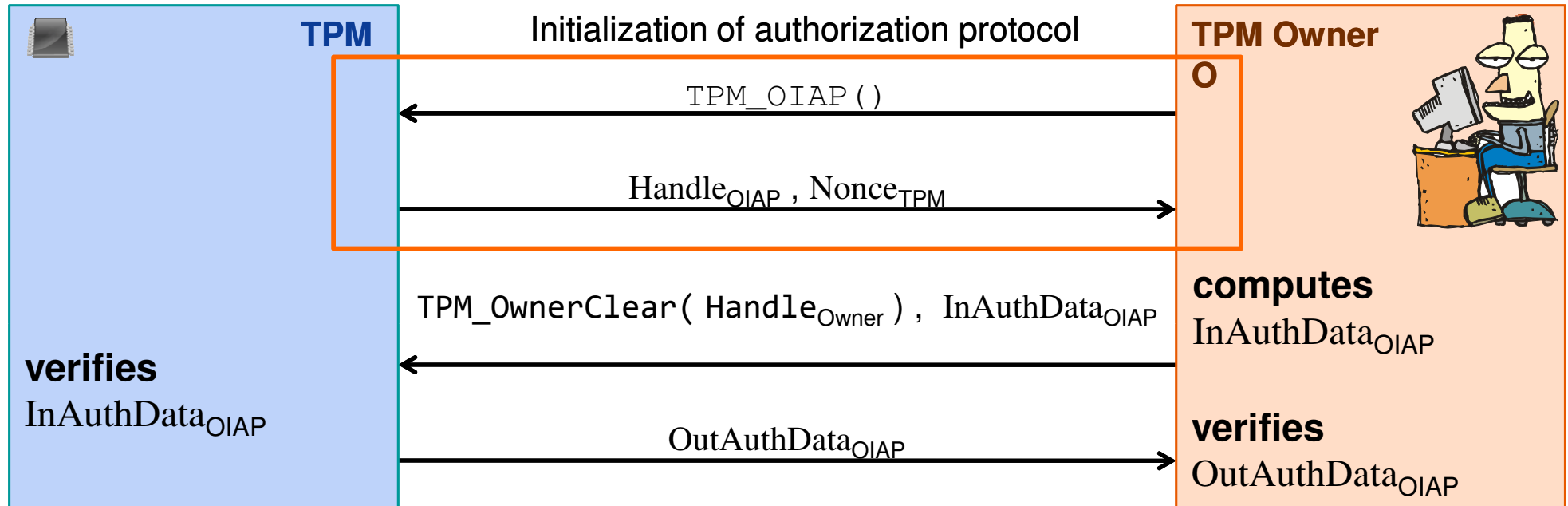
- A'_{Owner} and A'_{SRK} are authorization secrets (e.g., digests of passphrases) chosen by the TPM Owner

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 `InAuthDataOIAP` 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 `OutAuthDataOIAP` 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

```
OutAuthDataOIAP ← TPM_OwnerClear(HandleOwner) , InAuthDataOIAP
```

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

Notes

- Handle_{Owner} informs the TPM that the TPM Owner should be authorized
- InAuthData_{OIAP} 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 AOwner;  
    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

BIOS SETUP UTILITY

Advanced

TPM Configuration

TCG/TPM SUPPORT	[Enabled]
TPM Enabled	[Last Setting]
TPM Enable/Disable Status	[No State]
TPM Owner	[Last Setting]
TPM Owner Status	[No State]

Enable (Activate) /
Disable (Deactivate)
Command to TPM

←→ Select Screen
↑↓ Select Item
+- Change Option
F1 General Help
F10 Save and Exit
ESC Exit

v02.58 (C) Copyright 1985-2006, American Megatrends, Inc.

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

- 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**
 - **Authentication to the TPM**
- Summary

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

Output \leftarrow Command(Input, $Handle_E$)

- Compute $AuthData_{TPM}$ (authenticating the output)

Output of command Command()

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

InitAuthProt()

$Handle_S, Nonce_{TPM}$

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

if o.k., TPM can be assured that call is

- fresh (no replay)
- authentic (has not been modified)
- performed by an authorized user

Output, $AuthData_{TPM}$

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

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

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

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

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

$$AuthData_{TPM} \leftarrow HMAC(AuthSecret, SH-A-1(Command, Output), Nonce_U, \dots)$$

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

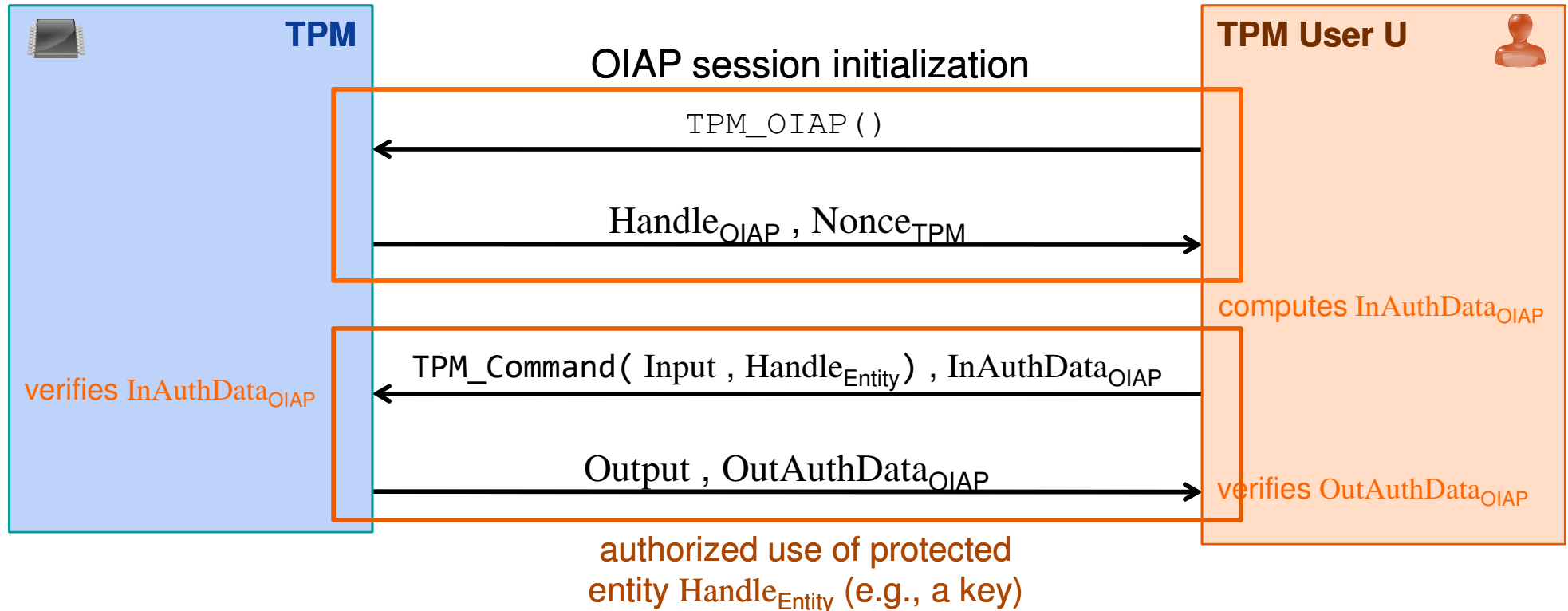
- Can authorize use of a single protected entity with multiple commands
- One setup required for each entity to be authorized
- Establishes an ephemeral shared session secret, which can be used as a cryptographic key

■ Mainly used for

- Setting or changing authorization data for protected entities

Authentication to the TPM

→ OIAP Protocol Session



Nonce is chosen by user U

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

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

$$OutAuthDigest_{OIAP} \leftarrow HMAC(AuthSecret_{Entity}, SHA-1(TPM_Command, Output), Nonce_{TPM,2}, Nonce)$$

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

Authentication to the TPM

→ Initialization of OIAP Session

```
( HandleOIAP , NonceTPM ) ← TPM_OIAP()
```

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

$$\text{InAuthDigest}_{\text{OIAP}} = \text{HMAC}(\text{AuthSecret}_{\text{Entity}}, \text{SHA-1}(\text{TPM_Command}, \text{Input}), \text{Nonce}_{\text{TPM}}, \text{Nonce})$$
$$\text{InAuthData}_{\text{OIAP}} = (\text{Handle}_{\text{OIAP}}, \text{Nonce}, \text{InAuthDigest}_{\text{OIAP}})$$
$$(\text{Output}, \text{OutAuthData}_{\text{OIAP}}) \leftarrow \text{TPM_Command}(\text{Input}, \text{Handle}_{\text{Entity}}), \text{InAuthData}_{\text{OIAP}}$$

```
if OIAPVerify( InAuthDataOIAP, HandleEntity ) ≠ ok then
    return error;
else
    Output ← TPM_Command( Input, HandleEntity );
    NonceTPM,2 ← RNG( 20 );
    OutAuthDigestOIAP ← HMAC( AuthSecretEntity,
        SHA-1( TPM_Command, Output ), NonceTPM,2, Nonce );
    OutAuthDataOIAP ← ( NonceTPM,2, OutAuthDigestOIAP );
    return ( Output, OutAuthDataOIAP );
end if;
```

$$\text{ind} \leftarrow \text{OIAPVerify}(\text{InAuthData}_{\text{OIAP}}, \text{Handle}_{\text{Entity}})$$

```
if HandleOIAP does not refer to an open OIAP session then
    return error;
else
    obtain AuthSecretEntity from entity referred to by HandleEntity;
    return Verify( InAuthDigestOIAP, AuthSecretEntity );
end if;
```

Prerequisites

- 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 , OutAuthDataOIAP ) ← TPM_Command( Input , HandleEntity ) , InAuthDataOIAP
```

```
if OIAPVerify( InAuthDataOIAP , HandleEntity ) ≠ ok then  
    return error;  
else  
    Output ← TPM_Command( Input , HandleEntity );  
    NonceTPM,2 ← RNG( 20 );  
    OutAuthDigestOIAP ← HMAC( AuthSecretEntity ,  
        SHA-1( TPM_Command , Output ) , NonceTPM,2 , Nonce );  
    OutAuthDataOIAP ← ( NonceTPM,2 , OutAuthDigestOIAP );  
    return ( Output , OutAuthDataOIAP );  
end if;
```

authorized use of Entity

authenticator for TPM's response

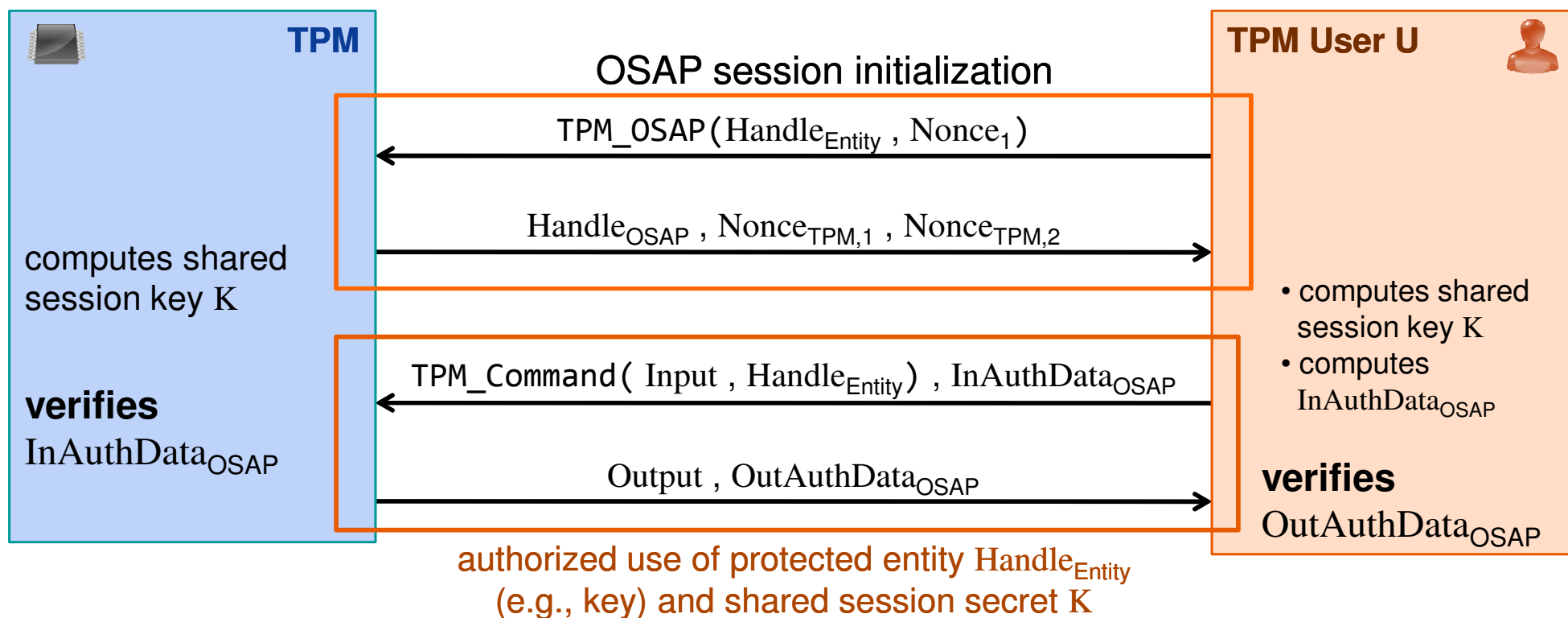
verification of authorization

```
ind ← OIAPVerify( InAuthDataOIAP , HandleEntity )
```

```
if HandleOIAP does not refer to an open OIAP session then  
    return error;  
else  
    obtain AuthSecretEntity from entity referred to by HandleEntity;  
    return Verify( InAuthDigestOIAP , AuthSecretEntity );  
end if;
```

Authentication to the TPM

→ OASP Protocol Session



Nonce is chosen by user U

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

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

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

$$\text{OutAuthDigest}_{\text{OSAP}} \leftarrow \text{HMAC}(K, \text{SHA-1}(\text{TPM_Command}, \text{Output}), \text{Nonce}_{\text{TPM},3}, \text{Nonce}_2)$$

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

Authentication to the TPM

→ Initialization of OSAP Session

```
( HandleOSAP , NonceTPM,1 , NonceTPM,2 ) ← TPM_OSAP(HandleEntity , Nonce1)
```

```
if maximum number of authorization sessions has been reached then  
    return error;  
else  
    create HandleOSAP;  
    NonceTPM,1 ← RNG();  
    NonceTPM,2 ← RNG();  
    K ← HMAC( AuthSecretEntity , NonceTPM,2 , Nonce1 );  
    store ( HandleOSAP , HandleEntity , K , NonceTPM,1 , NonceTPM,2 ) in  
        volatile memory;  
    return ( HandleOSAP , NonceTPM,1 , NonceTPM,2 );  
end if;
```

Prerequisites

- 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 $\text{Handle}_{\text{Entity}}$ for that entity

Notes

- $\text{Handle}_{\text{OSAP}}$ is identifier for the new OSAP session
- TPM must ensure that no other active auth. session is referenced by $\text{Handle}_{\text{OSAP}}$

Authentication to the TPM

→ Initialization of OSAP Session

```
( HandleOSAP , NonceTPM,1 , NonceTPM,2 ) ← TPM_OSAP( HandleEntity , Nonce1 )
```

```
if maximum number of authorization sessions has been reached then  
    return error;  
else  
    create HandleOSAP;  
    NonceTPM,1 ← RNG();  
    NonceTPM,2 ← RNG();  
    K ← HMAC( AuthSecretEntity , NonceTPM,2 , Nonce1 );  
    store ( HandleOSAP , HandleEntity , K , NonceTPM,1 , NonceTPM,2 ) in  
        volatile memory;  
    return ( HandleOSAP , NonceTPM,1 , NonceTPM,2 );  
end if;
```

Notes

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

creation of shared session secret

Verification of an OSAP Session

$$K \leftarrow \text{HMAC}(\text{AuthSecret}_{\text{Entity}}, \text{Nonce}_{\text{TPM},2}, \text{Nonce}_1)$$
$$\text{InAuthData}_{\text{OSAP}} = (\text{Handle}_{\text{OSAP}}, \text{Nonce}_2, \text{InAuthDigest}_{\text{OSAP}})$$
$$\text{InAuthDigest}_{\text{OSAP}} = \text{HMAC}(K, \text{SHA-1}(\text{TPM_Command}, \text{Input}), \text{Nonce}_{\text{TPM},1}, \text{Nonce}_2)$$
$$(\text{Output}, \text{OutAuthData}_{\text{OSAP}}) \leftarrow \text{TPM_Command}(\text{Input}, \text{Handle}_{\text{Entity}}, \text{InAuthData}_{\text{OSAP}})$$

```
if OSAPVerify( InAuthDataOSAP, HandleEntity ) ≠ ok then
    return error;
else
    Output ← TPM_Command( Input, HandleEntity, K );
    NonceTPM,3 ← RNG( 20 );
    OutAuthDigestOSAP ← HMAC( K,
        SHA-1( TPM_Command, Output ), NonceTPM,3, Nonce2 );
    OutAuthDataOSAP ← ( NonceTPM,3, OutAuthDigestOSAP );
    return ( Output, OutAuthDataOSAP );
end if;
```

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

```
if HandleOSAP does not refer to an open OSAP session then
    return error;
else
    obtain AuthSecretEntity from entity referred to by HandleEntity;
    return Verify( InAuthDigestOSAP, AuthSecretEntity );
end if;
```

Prerequisites

- 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

```
( Output , OutAuthDataOSAP ) ← TPM_Command( Input , HandleEntity ) , InAuthDataOSAP
```

```
if OSAPVerify( InAuthDataOSAP , HandleEntity ) ≠ ok then  
    return error;  
else  
    Output ← TPM_Command( Input , HandleEntity , K );  
    NonceTPM,3 ← RNG( 20 );  
    OutAuthDigestOSAP ← HMAC( K ,  
        SHA-1( TPM_Command , Output ) , NonceTPM,3 , Nonce2 );  
    OutAuthDataOSAP ← ( NonceTPM,3 , OutAuthDigestOSAP );  
    return ( Output , OutAuthDataOSAP );  
end if;
```

authorized use of Entity
and session secret K

authenticator for TPM's
response

verification of
authorization

```
ind ← OSAPVerify( InAuthDataOSAP , HandleEntity )
```

```
if HandleOSAP does not refer to an open OSAP session then  
    return error;  
else  
    obtain AuthSecretEntity from entity referred to by HandleEntity;  
    return Verify( InAuthDigestOSAP , AuthSecretEntity );  
end if;
```

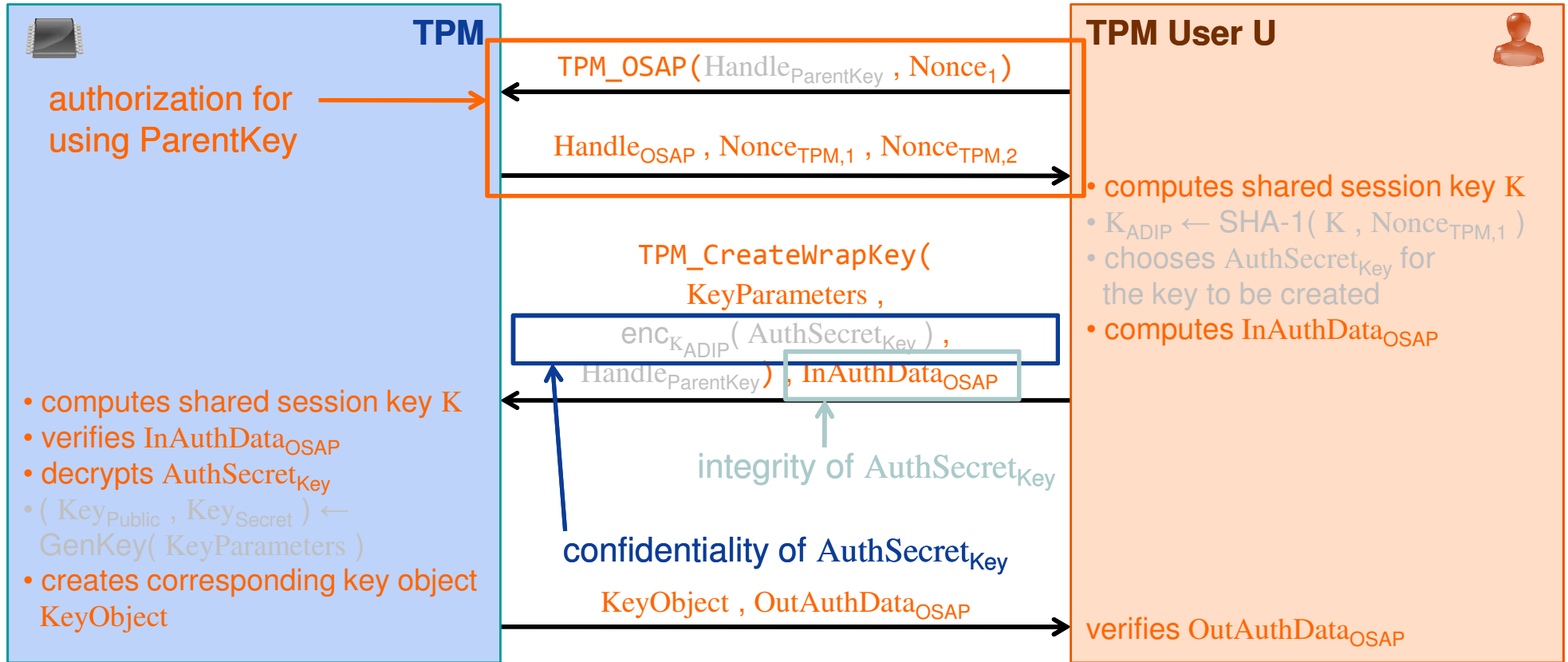
Authentication to the TPM

→ 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 ($\text{InAuthData}_{\text{OSAP}}$)
 - 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



$$KeyObject = (KeyParameters, Key_{Public}, enc_{ParentKey}(AuthSecret_{Key}, Key_{Secret}))$$

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 losing 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

- **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

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>



if(is)
internet security.

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/>