Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Trusted Computing Group

→ Functionalities

Prof. Dr. (TU NN)
Norbert Pohlmann

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

if(is) internet security.

Content

Content

- **Aim and outcomes of this lecture**
- Authenticated Boot
- Binding and Sealing
- Integrity Reporting/Attestation
- Direct Anonymous Attestation (DAA)
- Summary

TCG Functionalities
→ Aims and outcomes of this lecture

Aims

- To introduce in the topic of the Trusted Computing functionalities.
- To explore the different Trusted Computing functionalities
- To analyze the function and interfaces of Trusted Computing
- To assess the concerns of Trusted Computing

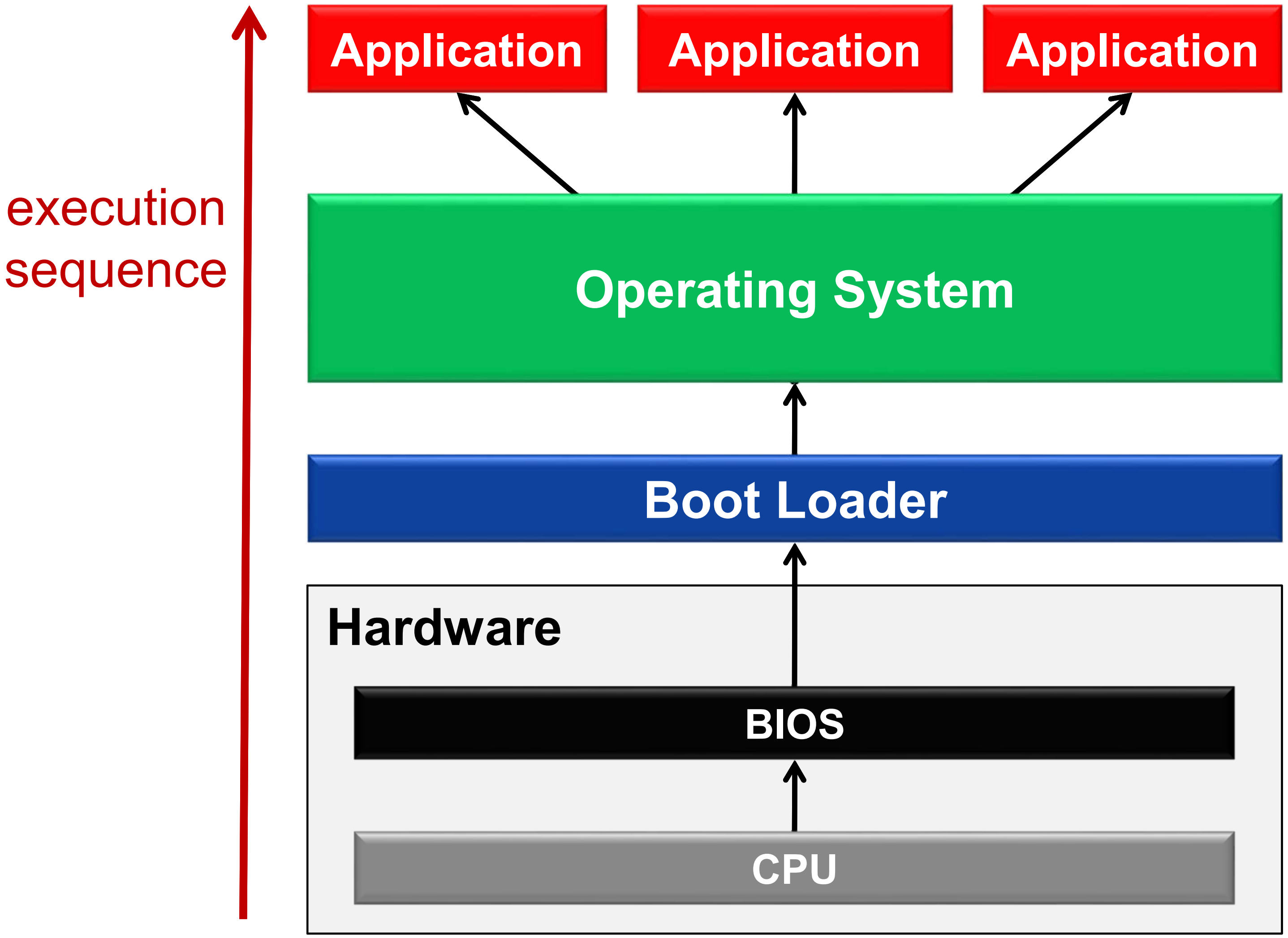
At the end of this lecture you will be able to:

- Understand what the Trusted Computing functionalities are.
- Know something about Trusted Computing mechanisms.
- Understand the reasoning behind the Trusted Computing functionalities.

Content

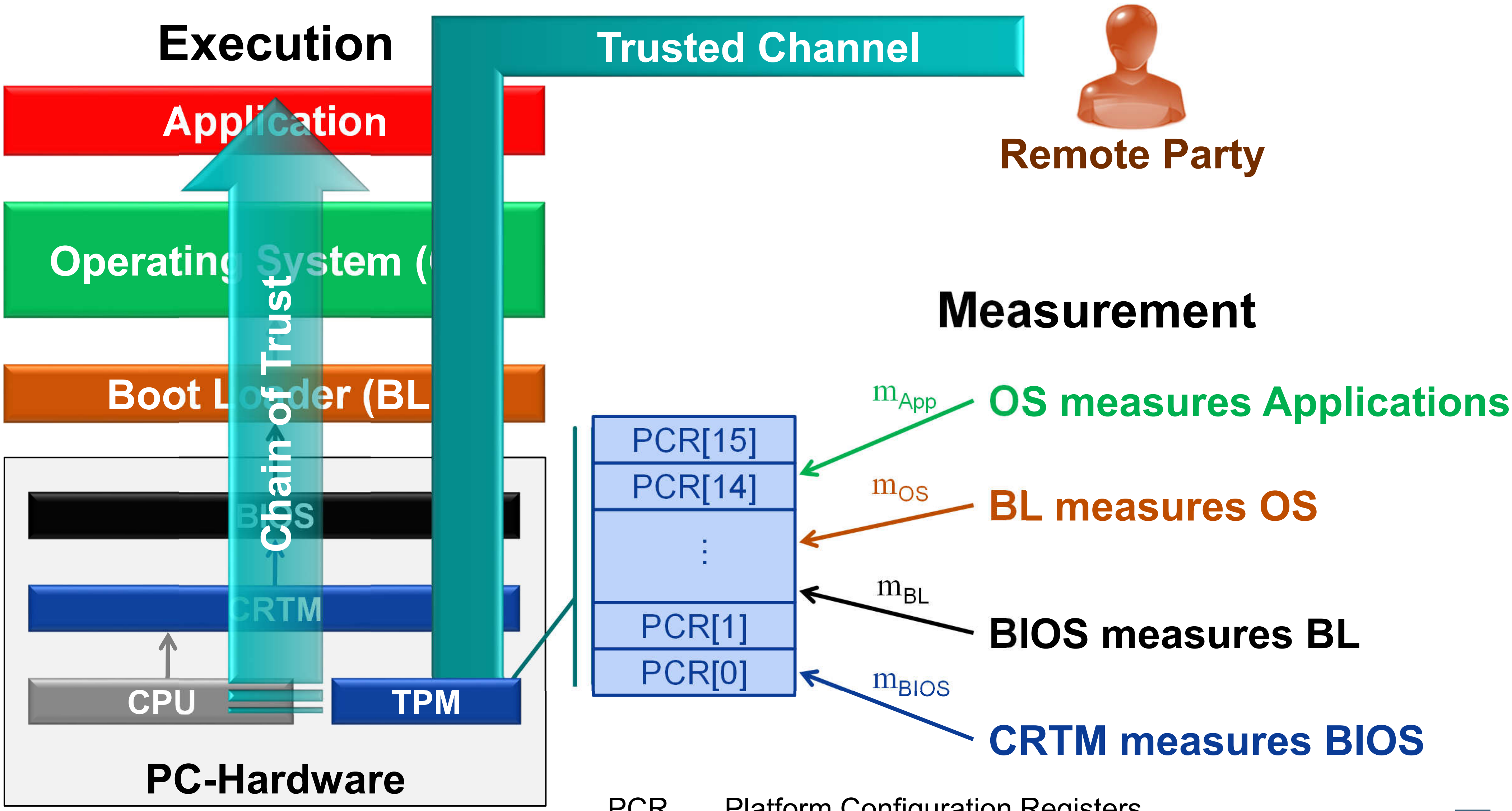
- Aim and outcomes of this lecture
- **Authenticated Boot**
- Binding and Sealing
- Integrity Reporting/Attestation
- Direct Anonymous Attestation (DAA)
- Summary

Authenticated Boot
→ Bootstrap Architecture in PC



boot loader provides a **link** between hardware and operating system

Authenticated Boot
→ Bootstrap and Integrity Measurement



Execution

Application

Operating System (OS)

Boot Loader (BL)

BIOS

CRTM

CPU

TPM

PC-Hardware

Chain of Trust

Trusted Channel

Remote Party

Measurement

PCR[15]
PCR[14]
⋮
PCR[1]
PCR[0]

m_{App} **OS measures Applications**

m_{OS} **BL measures OS**

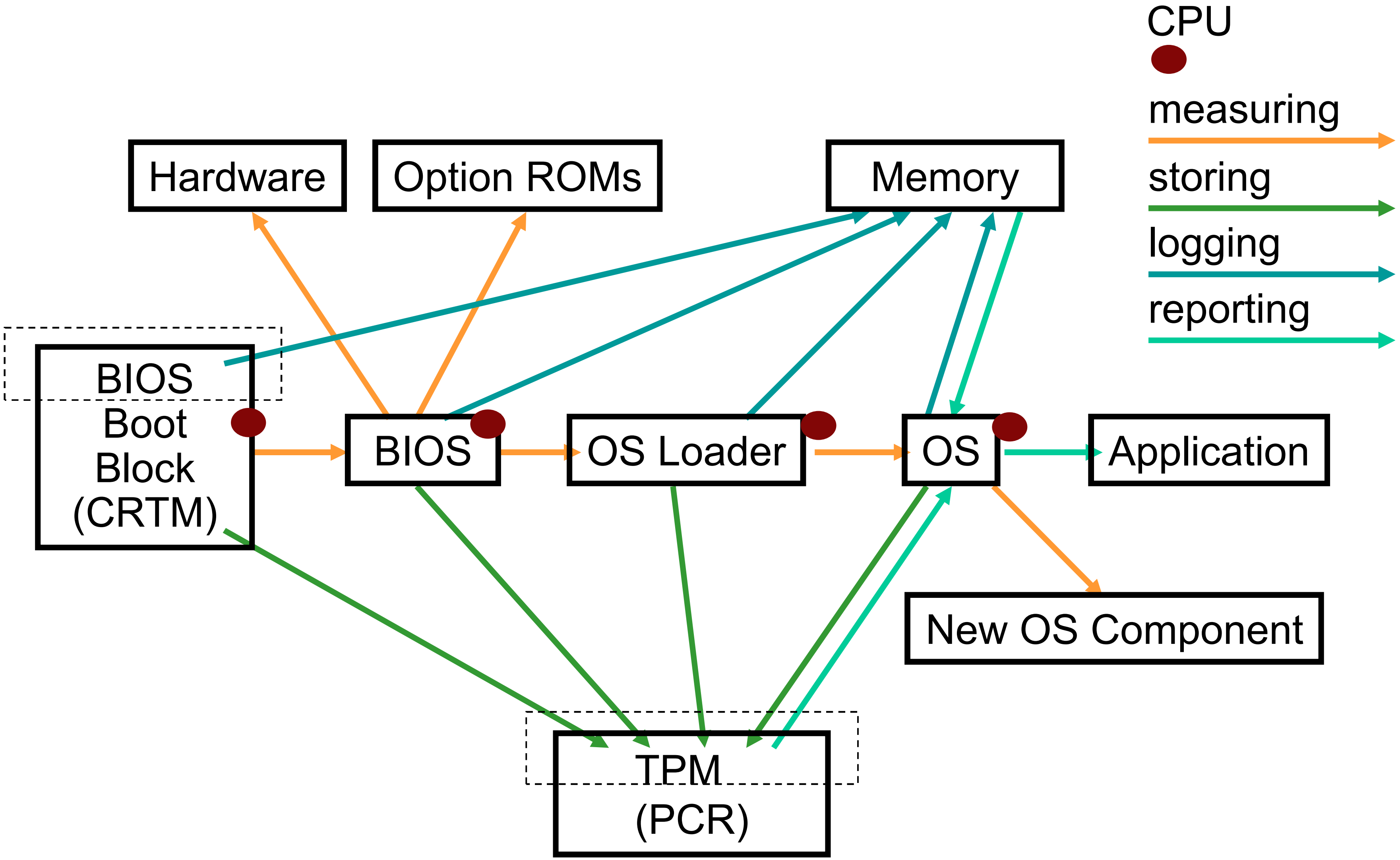
m_{BL} **BIOS measures BL**

m_{BIOS} **CRTM measures BIOS**

🟦 **Trusted components**

PCR	Platform Configuration Registers
CRTM	Core Root of Trust for Measurement
TPM	Trusted Platform Module

Authenticated Boot
→ Bootstrap and Integrity Measurement



CPU

●

measuring

storing

logging

reporting

Hardware

Option ROMs

Memory

BIOS Boot Block (CRTM)

BIOS

OS Loader

OS

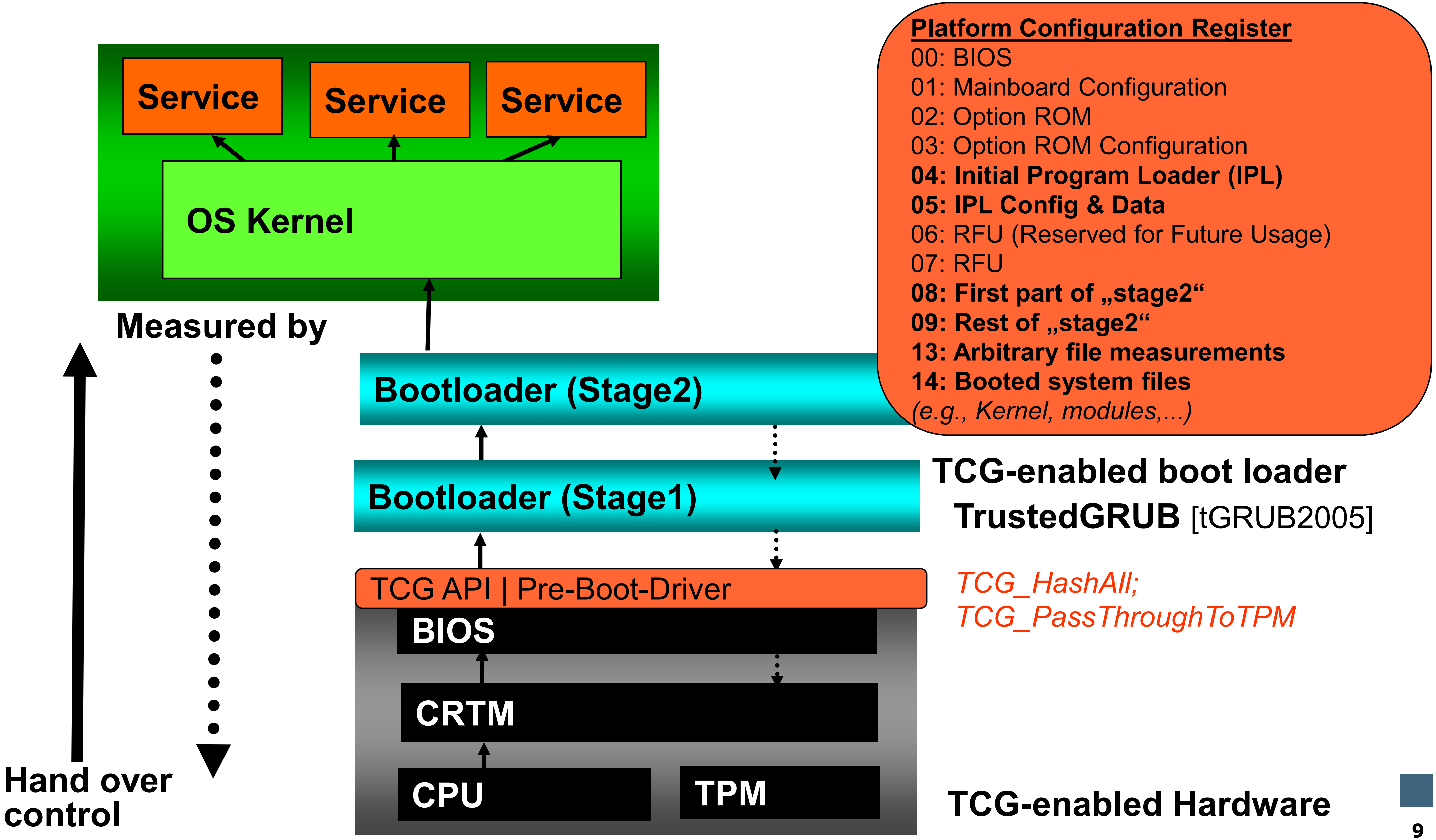
Application

New OS Component

TPM (PCR)

CRTM = Core Root of Trust Measurement,   PCR = Platform Configuration Register

Authenticated Boot
→ Bootstrap and Integrity Measurement



Platform Configuration Register
00: BIOS
01: Mainboard Configuration
02: Option ROM
03: Option ROM Configuration
04: Initial Program Loader (IPL)
05: IPL Config & Data
06: RFU (Reserved for Future Usage)
07: RFU
08: First part of „stage2“
09: Rest of „stage2“
13: Arbitrary file measurements
14: Booted system files
(e.g., Kernel, modules,...)

TCG-enabled boot loader
TrustedGRUB [tGRUB2005]

TCG_HashAll;
TCG_PassThroughToTPM

TCG-enabled Hardware

Content

- Aim and outcomes of this lecture
- Authenticated Boot
- **Binding and Sealing**
- Integrity Reporting/Attestation
- Direct Anonymous Attestation (DAA)
- Summary

Binding vs. Sealing
→ Overview

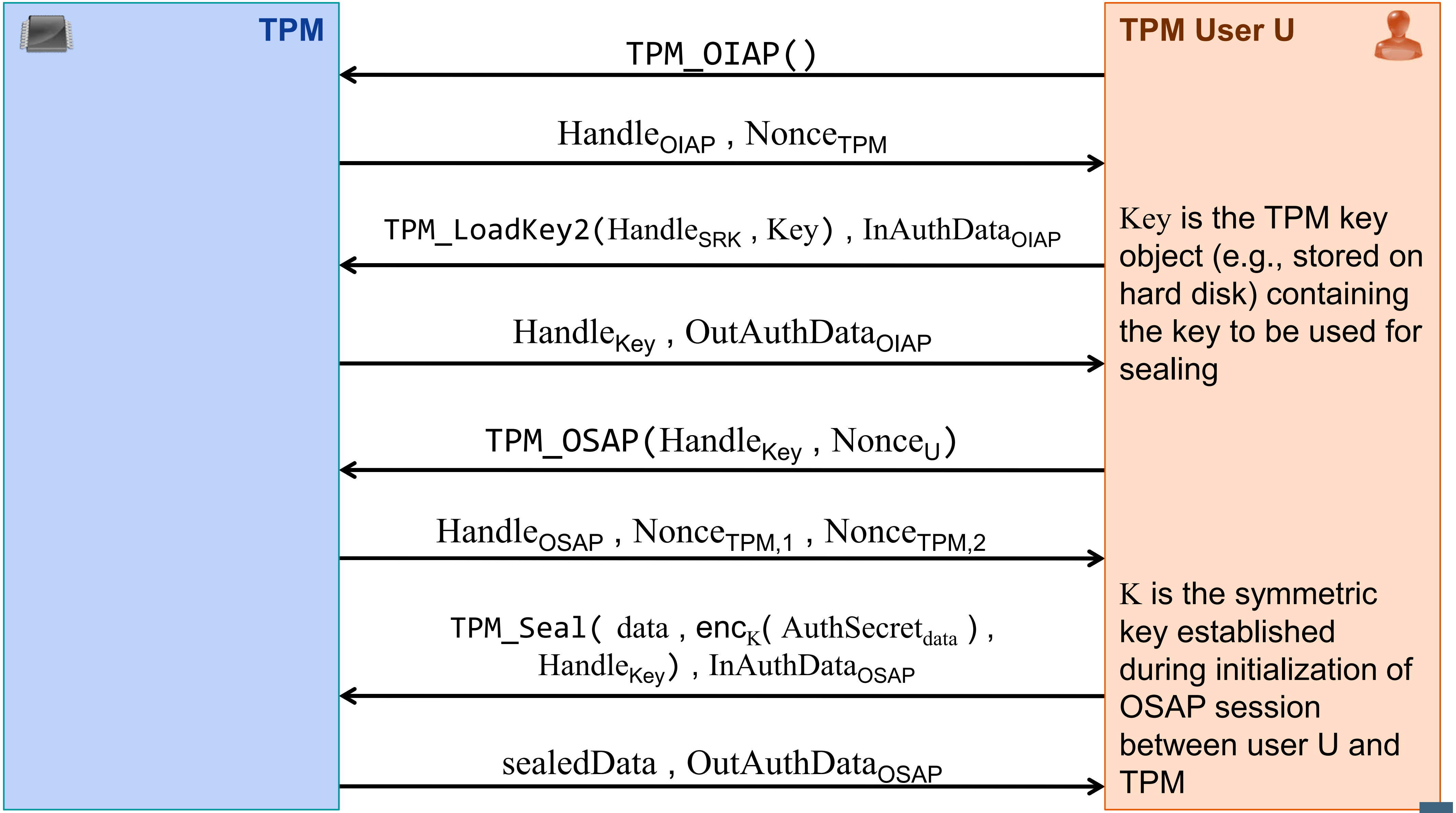
Binding

- Conventional asymmetric encryption
- May be used to bind data to a specific TPM/platform
  - Data encrypted with non-migratable key can only be recovered by TPM that knows corresponding secret key
- Usually no platform binding
  - Since binding can also be used with migratable keys
- **No interaction with TPM required**

Sealing (extension of binding)

- Always binds data to a specific TPM/platform
  - **Sealing can only be used with non-migratable storage keys**
- Configuration of encrypting platform can be verified
  - Ciphertext includes platform's **state** at the time of encryption
- May bind data to a specific platform configuration
  - Data can be decrypted only if platform is in a pre-defined (probably trusted) state

Protocol for Sealing



TPM

TPM User U

`TPM_OIAP()`

Handle$_{OIAP}$, Nonce$_{TPM}$

`TPM_LoadKey2(`Handle$_{SRK}$, Key`)` , InAuthData$_{OIAP}$

Handle$_{Key}$, OutAuthData$_{OIAP}$

`TPM_OSAP(`Handle$_{Key}$, Nonce$_{U}$`)`

Handle$_{OSAP}$, Nonce$_{TPM,1}$, Nonce$_{TPM,2}$

`TPM_Seal(` data , enc$_{K}$(AuthSecret$_{data}$) , Handle$_{Key}$`)` , InAuthData$_{OSAP}$

sealedData , OutAuthData$_{OSAP}$

Key is the TPM key object (e.g., stored on hard disk) containing the key to be used for sealing

K is the symmetric key established during initialization of OSAP session between user U and TPM

TPM-Interface for Sealing

$$(\text{sealedData}, \text{OutAuthData}_{\text{OSAP}}) \leftarrow \texttt{TPM\_Seal(}\ \text{data}, \text{enc}_K(\text{AuthSecret}_{\text{data}}), \text{pcr}, \text{Handle}_{\text{Key}}), \text{InAuthData}_{\text{OSAP}}$$

```
if OSAPVerify( $\text{InAuthData}_{\text{OSAP}}$ , $\text{Handle}_{\text{Key}}$ ) $\neq$ ok
or $\text{Handle}_{\text{Key}}$ is not a non-migratable storage key then
   return error;
else
   $\text{Key} \leftarrow \text{Handle}_{\text{Key}}$ ;
   $\text{AuthSecret'}_{\text{data}} \leftarrow \text{dec}_K(\text{enc}_K(\text{AuthSecret}_{\text{data}}))$;
   $\text{PCR}_{\text{seal}} \leftarrow$ getCurrentPCRs();
   $\text{PCR}_{\text{unseal}} \leftarrow \text{pcr}$;
   $\text{digest}_{\text{PCR}} \leftarrow \text{SHA-1}(\text{PCR}_{\text{seal}}, \text{PCR}_{\text{unseal}})$;
   $\text{ciphertext} \leftarrow \text{enc}_{\text{Key}}(\text{AuthSecret'}_{\text{data}}, \text{digest}_{\text{PCR}}, \text{data})$;
   $\text{sealedData} \leftarrow (\text{PCR}_{\text{seal}}, \text{PCR}_{\text{unseal}}, \text{ciphertext})$;
   compute $\text{OutAuthData}_{\text{OSAP}}$;
   return ( sealedData , $\text{OutAuthData}_{\text{OSAP}}$);
end if;
```

encryption of data cryptographically bound to PCR values

- that were present during encryption (PCR_{seal})
- that must be present for decryption ($\text{PCR}_{\text{unseal}}$)

Perquisites

- `TPM_OSAP()` must have been executed previously
- Key to be used to seal data must
  - have been previously loaded into the TPM
  - be accessible via $\text{Handle}_{\text{Key}}$

Notes

- K is the shared OSAP session key
- $\text{AuthSecret}_{\text{data}}$ is a secret chosen by the caller of the command and is required for later authorization of data to be unsealed
- pcr represents PCR values that must exist inside TPM's PCRs during unsealing operation to allow decryption of data

TPM-Interface for Unsealing

$$(\text{unsealedData} , \text{OutAuthData}_{OIAP,Key} , \text{OutAuthData}_{OIAP,Key}) \leftarrow \texttt{TPM\_UnSeal(}\text{Handle}_{Key} , \text{sealedData}\texttt{)} , \text{InAuthData}_{OSAP,Key} , \text{InAuthData}_{OIAP,Data}$$

```
if OSAPVerify( $InAuthData_{OSAP,Key}$ , $Handle_{Key}$ ) $\neq$ ok then
   return error;
else if $Handle_{Key}$ is not a non-migratable storage key then
   return error;
else
   Key $\leftarrow$ $Handle_{Key}$ ;
   ( $AuthSecret_{data}$ , $digest_{PCR}$ , data ) $\leftarrow$ $dec_{Key}$( ciphertext );
   if Verify( $digest_{PCR}$ , $PCR_{seal}$ , $PCR_{unseal}$ ) $\neq$ ok
   or  getCurrentPCRs() $\neq$ $PCR_{unseal}$ then
      return error;
   else if OIAPVerifiy( $InAuthData_{OIAP,Data}$ , $AuthSecret_{data}$ ) $\neq$ ok then
      return error;
   else
      unsealedData $\leftarrow$ $enc_{K}$( data , $PCR_{seal}$ );
      return unsealedData;
   end if;
end if;
```

sealedData = (PCR_{seal} , PCR_{unseal} , ciphertext)

Perquisites

- Requires authorization for
  - using the unsealing key
  - releasing unsealed data
- Sealing key must
  - have been previously loaded into the TPM
  - be accessible via $Handle_{Key}$

Notes

- K_{OSAP} is the symmetric key generated during OSAP initialization shared between the TPM and the caller of the command

TPM-Interface for Unsealing

$(\mathrm{unsealedData}\ ,\ \mathrm{OutAuthData}_{\mathrm{OIAP,Key}}\ ,\ \mathrm{OutAuthData}_{\mathrm{OIAP,Key}}\ ) \leftarrow$ `TPM_UnSeal`$(\mathrm{Handle}_{\mathrm{Key}}\ ,\ \mathrm{sealedData})\ ,$
$\mathrm{InAuthData}_{\mathrm{OSAP,Key}}\ ,\ \mathrm{InAuthData}_{\mathrm{OIAP,Data}}$

```
if OSAP_Verify( $\mathrm{InAuthData}_{\mathrm{OSAP,Key}}$ , $\mathrm{Handle}_{\mathrm{Key}}$ ) $\neq$ ok then
  return error;
else if $\mathrm{Handle}_{\mathrm{Key}}$ is not a non-migratable storage key then
  return error;
else
  Key $\leftarrow$ $\mathrm{Handle}_{\mathrm{Key}}$ ;
  ( $\mathrm{AuthSecret}_{\mathrm{data}}$ , $\mathrm{digest}_{\mathrm{PCR}}$ , data ) $\leftarrow$ $\mathrm{dec}_{\mathrm{Key}}$( ciphertext );
  if Verify( $\mathrm{digest}_{\mathrm{PCR}}$ , $\mathrm{PCR}_{\mathrm{seal}}$ , $\mathrm{PCR}_{\mathrm{unseal}}$ ) $\neq$ ok
  or  getCurrentPCRs() $\neq$ $\mathrm{PCR}_{\mathrm{unseal}}$ then
    return error;
  else if OIAPVerifiy( $\mathrm{InAuthData}_{\mathrm{OIAP,Data}}$ , $\mathrm{AuthSecret}_{\mathrm{data}}$ ) $\neq$ ok then
    return error;
  else
    unsealedData $\leftarrow$ $\mathrm{enc}_{\mathrm{K}}$( data , $\mathrm{PCR}_{\mathrm{seal}}$ );
    return unsealedData;
  end if;
end if;
```

verification of authorized use of the key to be used to unseal sealedData

decryption of sealedData

integrity check of PCR information stored with sealed data

verify that current PCR values match $\mathrm{PCR}_{\mathrm{unseal}}$, which are the PCR values the data is bound to

verify that the caller of the unsealing command is authorized to release the unsealed data

use OSAP key K to encrypt data and PCR values $\mathrm{PCR}_{\mathrm{seal}}$ that were present during sealing operation

Content

- Aim and outcomes of this lecture
- Authenticated Boot
- Binding and Sealing
- **Integrity Reporting / Attestation**
- Direct Anonymous Attestation (DAA)
- Summary

Integrity Reporting / Attestation
→ Overview

- **Authentic report of a platform's state to a (remote) verifier**
  - A local or remote verifier (challenger) is interested in platform configuration (e.g., hard- and software environment)
  - Verifier is able to decide whether it trusts the attested configuration
    - e.g., an online-banking client checks whether the bank's server is in a known secure configuration (e.g., has not been tampered with)

- **TPM and CRTM act as Root of Trust for Reporting (RTR)**
  - TPM can generate authentic reports of current integrity measurement values (current PCR content)

Integrity Reporting / Attestation
→ Requirements on Attestation

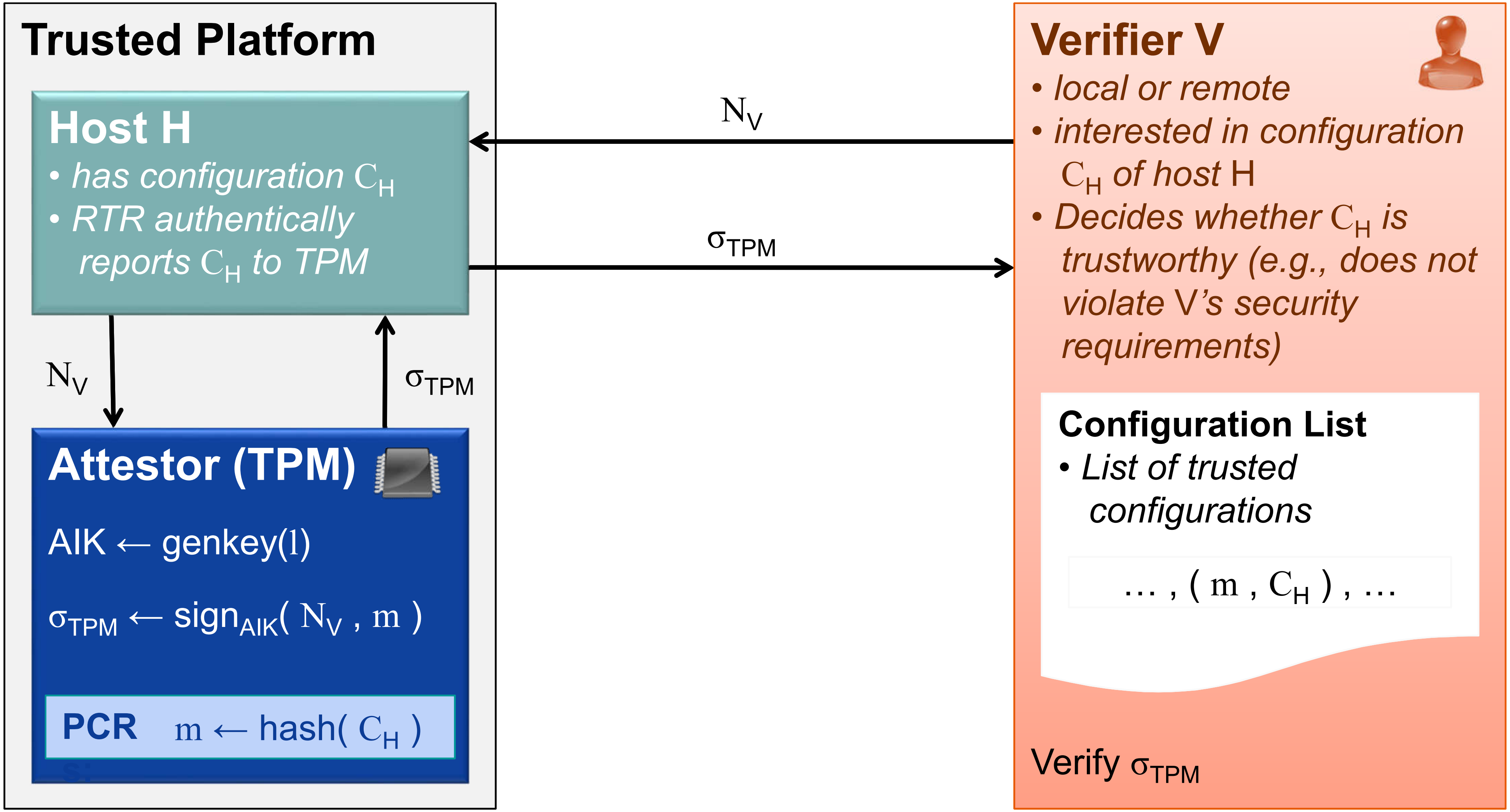
- **Attest to all states of entities (machines) capable of affecting the behavior of the entity being attested**
  - e.g., hard- and software environment of the attesting platform including history of all executed program code

- **Attestation vector** (platform’s state report)
  - Integrity, confidentiality, freshness

- **Authenticity of attestor**

- **Privacy**
  - Minimal/zero information disclosure on system configuration and platform identity

Integrity Reporting / Attestation
→ Simplified TCG Attestation Concept



Trusted Platform

Host H
- *has configuration C_H*
- *RTR authentically reports C_H to TPM*

Attestor (TPM)

AIK ← genkey(l)

σ_{TPM} ← $\text{sign}_{AIK}(N_V , m)$

PCR m ← hash(C_H)

Verifier V
- *local or remote*
- *interested in configuration C_H of host H*
- *Decides whether C_H is trustworthy (e.g., does not violate V's security requirements)*

Configuration List
- *List of trusted configurations*

… , (m , C_H) , …

Verify σ_{TPM}

N_V Nonce (anti-replay value) chosen by the verifier

C_H current configuration of host H

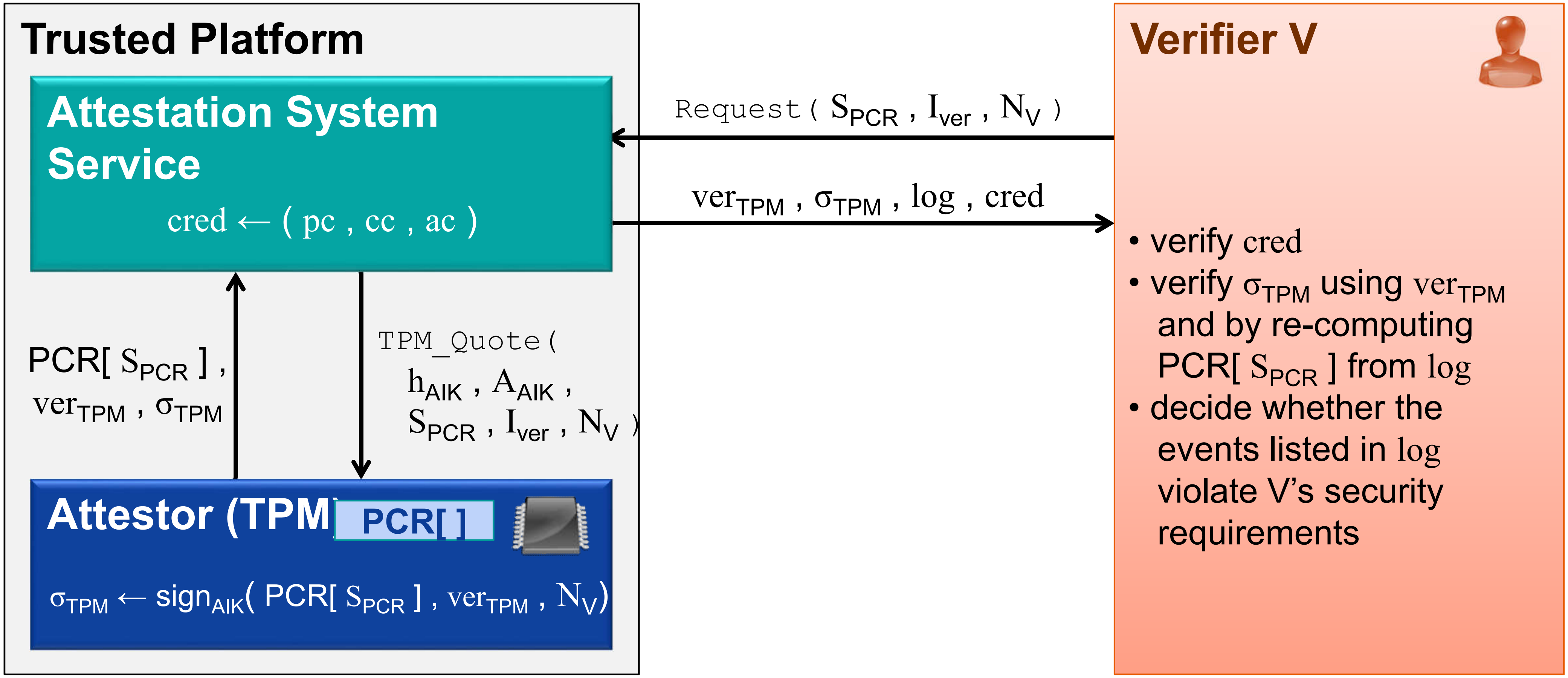
Integrity Reporting / Attestation
→ Related TPM-Interface

- **Reporting of PCR values signed by the TPM**
  - Command: `TPM_Quote2` and `TPM_Quote` (deprecated)
  - May be called by an attestation system service that handles attestation requests

- **Input to `TPM_Quote2` / `TPM_Quote`**
  - AIK to be used to sign current PCR values
  - Nonce (anti-replay value)
  - Selection of PCRs to be reported
  - Indicator whether the TPM version and revision should be added to the signed report of PCR values
  - Authorization data for using the AIK

Integrity Reporting / Attestation

→ More Details about TCG Attestation



Trusted Platform

Attestation System Service

cred ← (pc , cc , ac)

$\mathrm{PCR}[\, S_{PCR} \,]$, ver_{TPM} , σ_{TPM}

`TPM_Quote(` h_{AIK} , A_{AIK} , S_{PCR} , I_{ver} , N_V)

Attestor (TPM) **PCR[ ]**

$\sigma_{TPM} \leftarrow \mathrm{sign}_{AIK}(\, \mathrm{PCR}[\, S_{PCR} \,] , ver_{TPM} , N_V)$

`Request(` S_{PCR} , I_{ver} , N_V)

ver_{TPM} , σ_{TPM} , log , cred

Verifier V

- verify cred
- verify σ_{TPM} using ver_{TPM} and by re-computing $\mathrm{PCR}[\, S_{PCR} \,]$ from log
- decide whether the events listed in log violate V's security requirements

S_{PCR}	selection of PCR values V is interested in
I_{ver}	indicator whether V is interested in TPM version information
N_V	Nonce (anti-replay value) chosen by the verifier
h_{AIK}	pointer (handle) to the AIK to be used
A_{AIK}	authorization secret required to use AIK

ver_{TPM}	TPM version information
pc	platform credential
cc	Conformance Credential
ac	Attestation Credential (e.g., from Privacy CA)
log	TPM Event Log

Integrity Reporting / Attestation

→ Attestation using Privacy CA

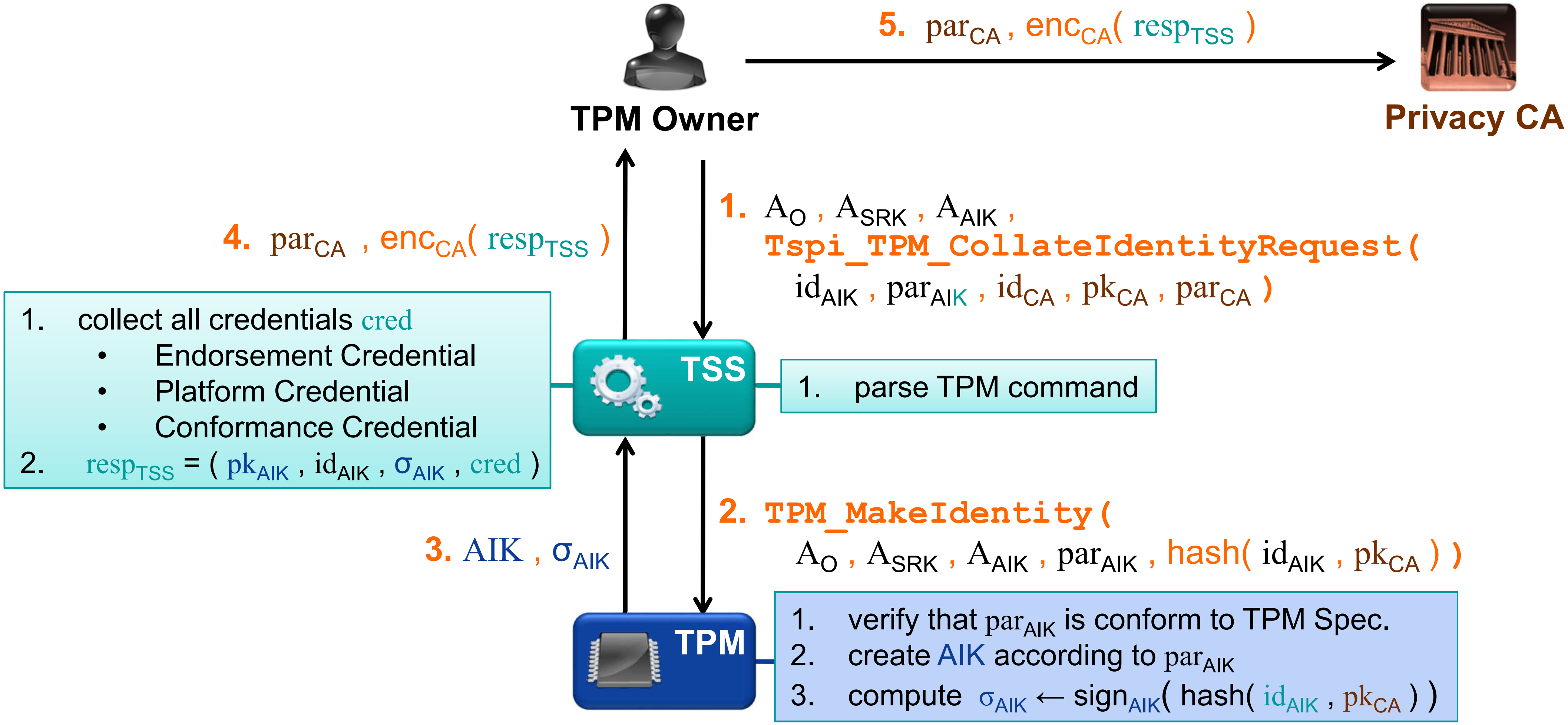
TPM Owner

- Prove to third parties that it’s platform is in a trustable state
    - e.g., by reporting platform integrity measurements signed with a certified key
- Colluding third parties should not be able to track platform’s transactions
    - e.g., by signing every integrity measurement report with a (ideally) different AIK for each transaction

Privacy CA

- Trusted Third Party
- Attests that an AIK belongs to a valid TPM (Attestation Credential)
    - Protocol for certification of an AIK requires disclosure of public EK to Privacy CA
- Must be trusted not to reveal any information that might enable correlation of AIKs with the corresponding platform identity (EK)

Integrity Reporting / Attestation
→ AIK Creation with Privacy CA I



5. par_{CA} , $\text{enc}_{CA}(\ \text{resp}_{TSS}\ )$

1. A_O , A_{SRK} , A_{AIK} ,
`Tspi_TPM_CollateIdentityRequest(`
id_{AIK} , par_{AIK} , id_{CA} , pk_{CA} , par_{CA} **`)`**

4. par_{CA} , $\text{enc}_{CA}(\ \text{resp}_{TSS}\ )$

1. collect all credentials cred
   - Endorsement Credential
   - Platform Credential
   - Conformance Credential
2. $\text{resp}_{TSS} = (\ \text{pk}_{AIK}\ ,\ \text{id}_{AIK}\ ,\ \sigma_{AIK}\ ,\ \text{cred}\ )$

1. parse TPM command

2. **`TPM_MakeIdentity(`**
A_O , A_{SRK} , A_{AIK} , par_{AIK} , $\text{hash}(\ \text{id}_{AIK}\ ,\ \text{pk}_{CA}\ )$ **`)`**

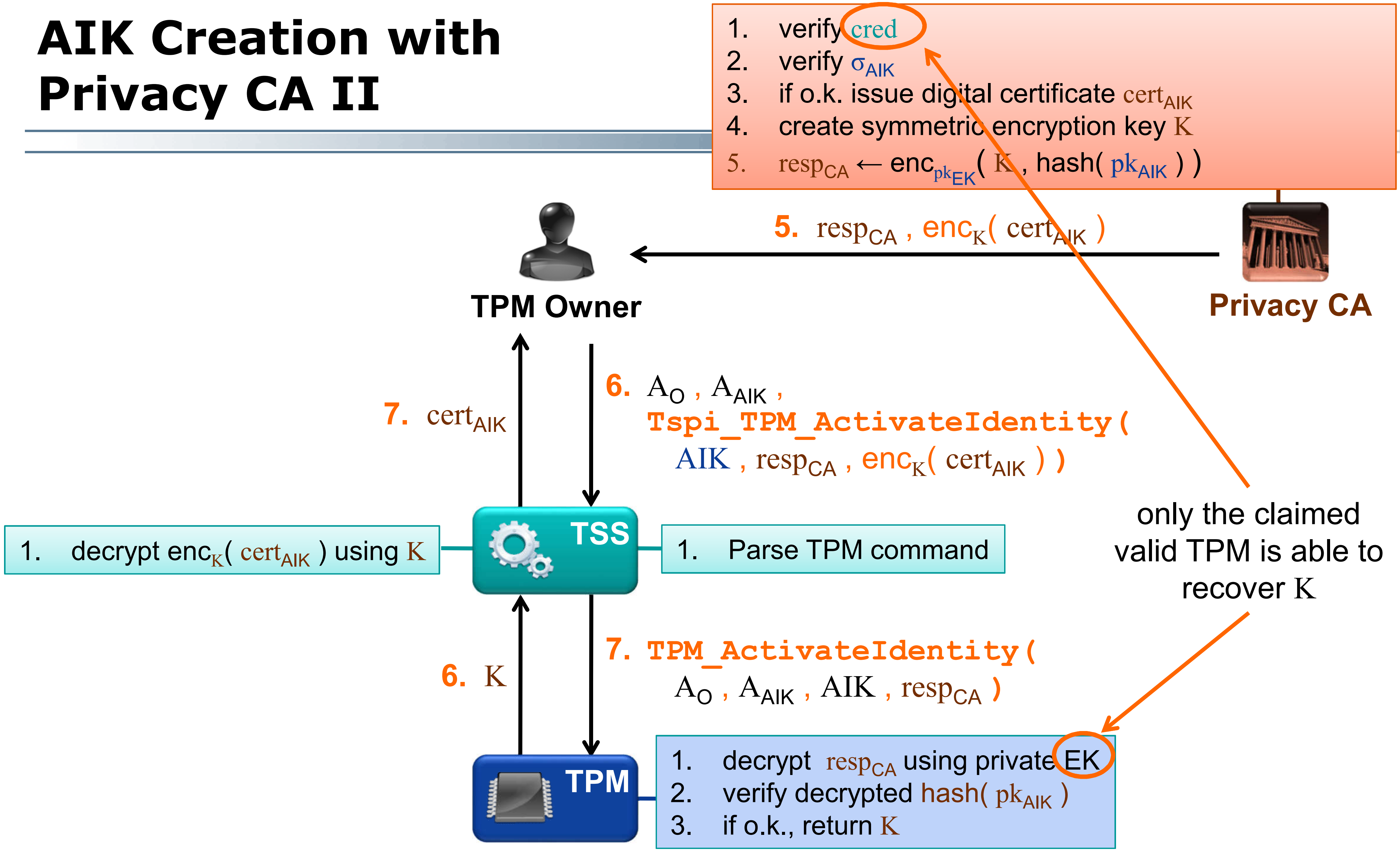
3. AIK , σ_{AIK}

1. verify that par_{AIK} is conform to TPM Spec.
2. create AIK according to par_{AIK}
3. compute $\sigma_{AIK} \leftarrow \text{sign}_{AIK}(\ \text{hash}(\ \text{id}_{AIK}\ ,\ \text{pk}_{CA}\ )\ )$

A_O	authorization secret required to create a new AIK
A_{SRK}	authorization data required to use the SRK
A_{AIK}	authorization data required for using the new AIK
id_{AIK}	identity label (e.g., name) for new AIK
AIK	key object storing the (public and private) AIK

par_{AIK}	parameters for the new AIK (e.g., key size and type)
id_{CA}	identity label (e.g., name) of the Privacy CA
par_{CA}	parameters for encrypted communication with A
pk_{CA}	public verification key of the Privacy CA

AIK Creation with Privacy CA II



1. verify cred
2. verify σ_{AIK}
3. if o.k. issue digital certificate $cert_{AIK}$
4. create symmetric encryption key K
5. $resp_{CA} \leftarrow enc_{pk_{EK}}(K , hash(pk_{AIK}))$

5. $resp_{CA}$, $enc_{K}(cert_{AIK})$

6. A_O , A_{AIK} , `Tspi_TPM_ActivateIdentity(` AIK , $resp_{CA}$, $enc_{K}(cert_{AIK})$ `)`

7. $cert_{AIK}$

1. decrypt $enc_{K}(cert_{AIK})$ using K

1. Parse TPM command

only the claimed valid TPM is able to recover K

6. K

7. `TPM_ActivateIdentity(` A_O , A_{AIK} , AIK , $resp_{CA}$ `)`

1. decrypt $resp_{CA}$ using private EK
2. verify decrypted hash(pk_{AIK})
3. if o.k., return K

A_O	authorization secret required to create a new AIK
A_{SRK}	authorization data required to use the SRK
A_{AIK}	authorization data required for using the new AIK
id_{AIK}	identity label (e.g., name) for new AIK
AIK	key object storing the (public and private) AIK

par_{AIK}	parameters for the new AIK (e.g., key size and type)
id_{CA}	identity label (e.g., name) of the Privacy CA
par_{CA}	parameters for encrypted communication with CA
pk_{CA}	public verification key of the Privacy CA

Integrity Reporting / Attestation

→ Attestation Identity Credential

Field Name	Description	Status
Credential Type Label	Type of the certificate	MUST
Public AIK	Public AIK value	MUST
TPM Model	Manufacturer-specific identifier	MUST
Platform Model	Manufacturer-specific identifier	MUST
Issuer	Identifies the issuer of the certificate	MUST
TPM Specification	Identifies the specification this TPM conforms to	MUST
Platform Specification	Identifies the specification this platform conforms to	MUST
Signature Value	Signature of the issuer over the other fields	MUST
Identity Label	String associated with the AIK by the issuer	MUST
TPM Assertions	Security Assertions about the TPM	MAY
Platform Assertions	Security Assertions about the platform	MAY
Validity Period	Time period when credential is valid	MAY
Policy Reference	Credential Policy Reference	MAY
Revocation Locator	Identifies source of revocation status information	MAY

Integrity Reporting / Attestation
→ Problems of Attestation with Privacy CA

- **No anonymity**
  - Collusion of Privacy CAs and verifiers enables tracking of platforms
- **Availability**
  - Certification of AIKs requires interaction with Privacy CA
  - A TPM may have a large number of AIKs
    - Worst case: one for each connection
  - Privacy CA may encounter heavy load serving certification requests of a huge number of TPMs
- **Solution:** Direct Anonymous Attestation (DAA) [BrCaCh2004,Brik2007]

Content

- Aim and outcomes of this lecture
- Authenticated Boot
- Binding and Sealing
- Integrity Reporting/Attestation
- **Direct Anonymous Attestation (DAA)**
- Summary

Overview

- **Entities**
  - DAA issuer: a DAA certificate issuer (e.g., a manufacturer of TCG platforms)
  - DAA signer: a trusted platform module (TPM) with help from a host platform
  - DAA verifier: an external partner (e.g., a service provider)

- **Primitives**
  - System and issuer setup
  - Join protocol
  - Signing algorithm
  - Verifying algorithm
  - Solution of restricted link
  - Solution of revocation

Camenisch-Lysyanskaya (CL) Signatures

Key Generation:

choose primes p , q

$n \leftarrow p \cdot q$

$R_1 , \ldots , R_k , S , Z \in_R QR_n$

$pk \leftarrow (n , R_1 , \ldots , R_k , S , Z)$

$sk \leftarrow (p , q)$

Signing: $(A , e , v) \leftarrow \text{Sign}(sk , m_1 , \ldots , m_k)$

choose prime $e > 2^l$

choose integer $v \approx n$

$A \leftarrow [Z \cdot (R_1^{m_1} \cdot \ldots \cdot R_k^{m_k} \cdot S^s)^{-1}]^{1/e} \bmod n$

can be computed efficiently only if p and q are known (Strong RSA Assumption)

Verification: $ind \leftarrow \text{Verify}(pk , (A , e , v) , (m_1 , \ldots , m_k))$

```
if  $m_1 , \ldots , m_k \in \{ 0 , 1 \}^l$  and  $e > 2^l$ prime  and  $Z = A^e \cdot R_1^{m1} \cdot \ldots \cdot R_k^{mk} \cdot S^v \bmod n$  then
    $ind \leftarrow$ valid;
else
    $ind \leftarrow$ invalid;
endif;
```

Randomization of CL-Signatures

- CL-signature (A , e , v) can be transformed into another valid CL-Signature (A' , e , v') on the same message
- This can be used to randomize signatures
  - e.g., to prevent tracking of signatures

Randomization: (A' , e , v') ← Randomize(pk , (A , e , v))

choose random $v^* \approx n$

$A' \leftarrow A \cdot S^{v^*}$

$v' \leftarrow v - e \cdot v^*$

Proof of Knowledge of CL-Signatures

Prover (P)

- has $cert_I(sk) = (A , e , v)$ on sk which should not be revealed to V
- wants to prove: “I have a valid CL-signature over a message m under my certified secret key sk”

choose random integers

$r_b , r_e , r_{sk} , r_v , n_P$

blind $cert_I(sk)$

$A' \leftarrow A \cdot S^{r_b}$

commit to A'

$T \leftarrow A'^{r_e} \cdot R^{r_{sk}} \cdot S^{r_v}$

$c \leftarrow \text{Hash}(pk , T , n_V , n_P , m)$

compute

$s_e \leftarrow r_e + c \cdot e$

$s_{sk} \leftarrow r_{sk} + c \cdot sk$

$s_v \leftarrow r_v + c \cdot v$

$\sigma \leftarrow (T , c , s_e , s_{sk} , s_v , n_P)$

$\xleftarrow{n_V}$

$\xrightarrow{\sigma , m}$

Verifier (V)

- knows $pk = (n , R , S , Z)$

choose random nonce

n_V

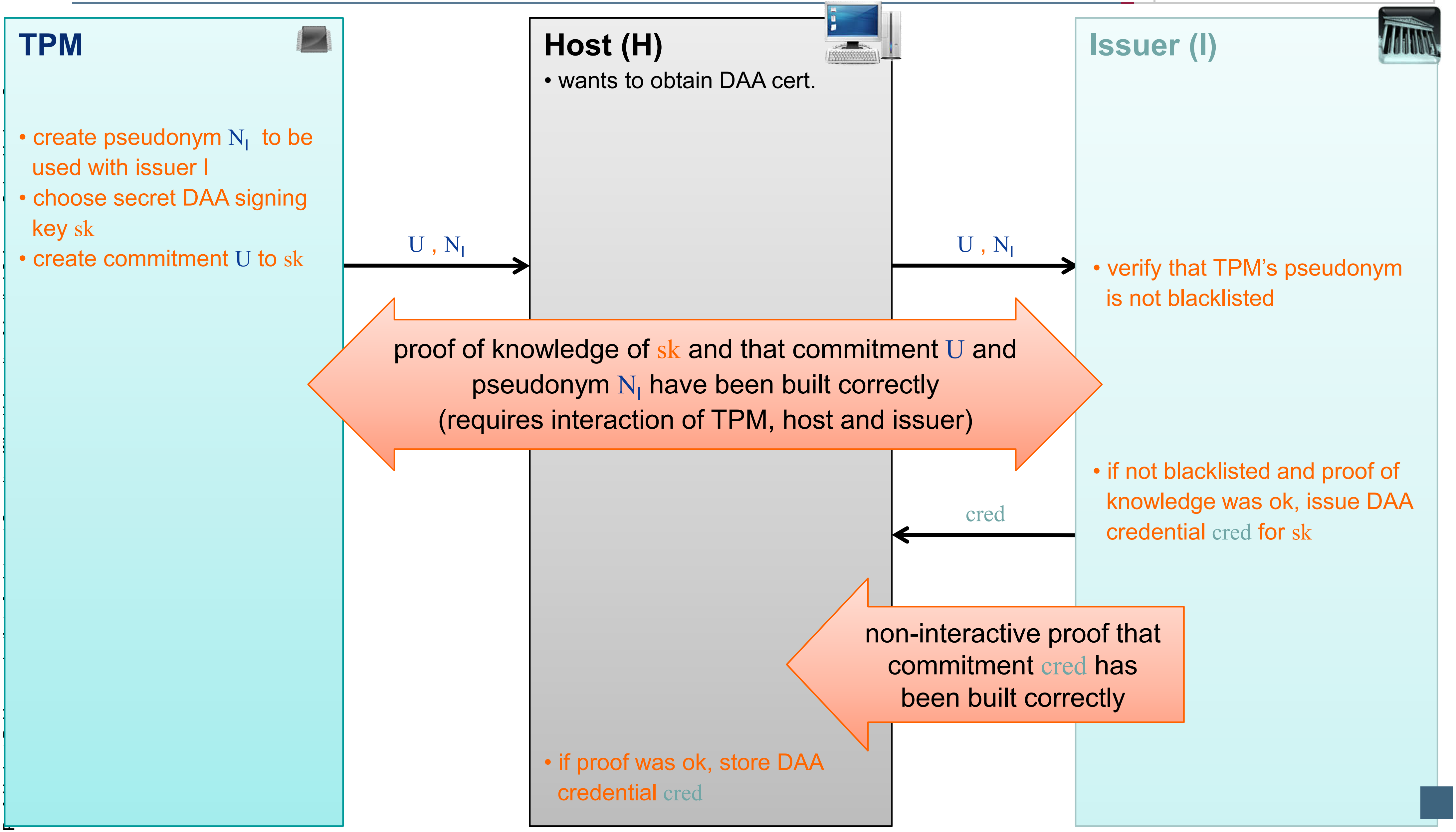
compute

$T^* \leftarrow Z^{-c} \cdot A'^{s_e} \cdot R^{s_{sk}} \cdot S^{s_v}$

verify that

$c = \text{Hash}(pk , T^* , n_V , n_P , m)$

DAA Join Protocol – Overview



TPM

- create pseudonym N_I to be used with issuer I
- choose secret DAA signing key sk
- create commitment U to sk

Host (H)

- wants to obtain DAA cert.

Issuer (I)

- verify that TPM's pseudonym is not blacklisted

proof of knowledge of sk and that commitment U and pseudonym N_I have been built correctly (requires interaction of TPM, host and issuer)

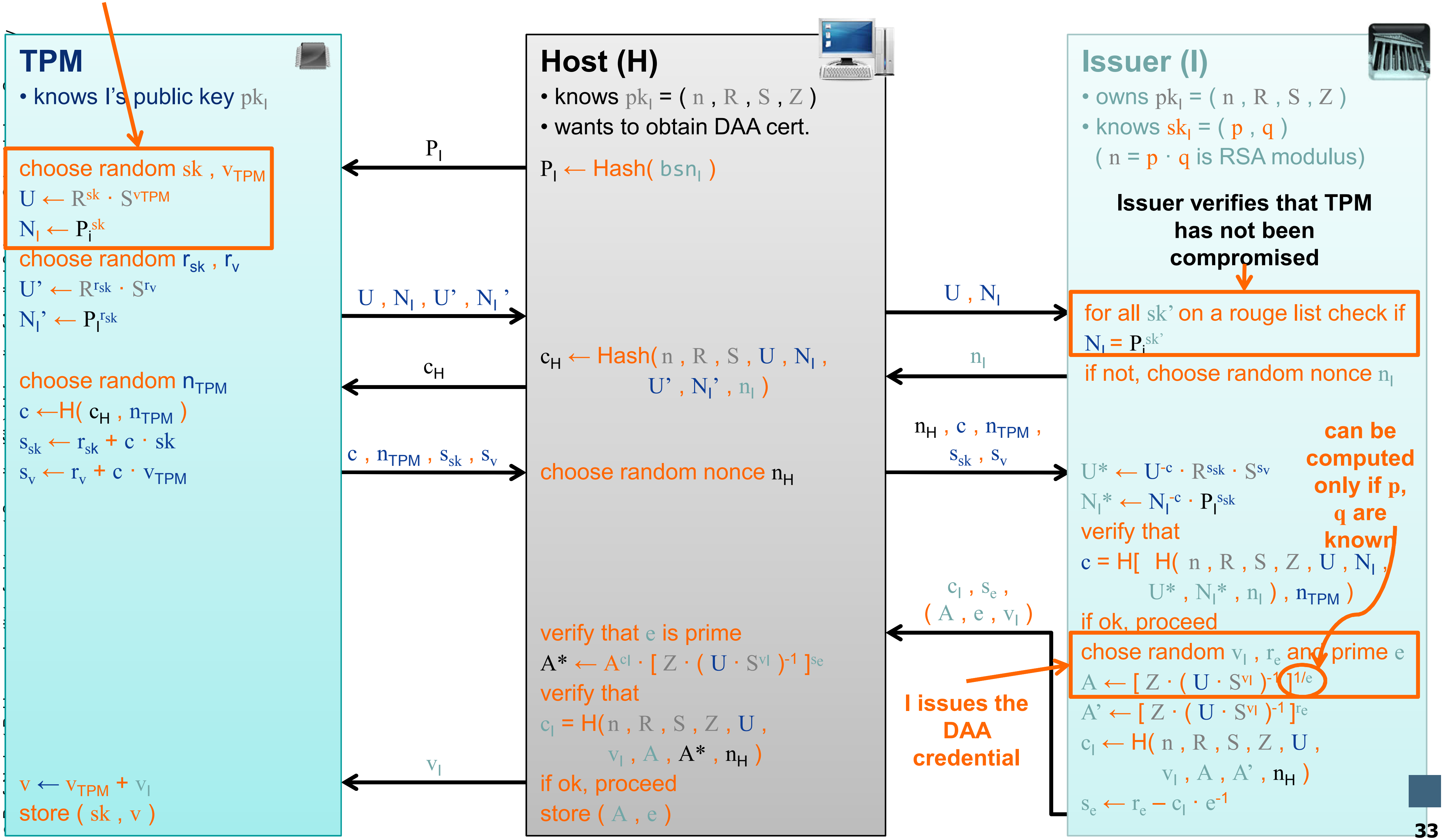
- if not blacklisted and proof of knowledge was ok, issue DAA credential $cred$ for sk

non-interactive proof that commitment $cred$ has been built correctly

- if proof was ok, store DAA credential $cred$

DAA Join Protocol (Simplified)

TPM chooses secret DAA signing key sk and commits to it



TPM

- knows I's public key pk_I

choose random sk , v_{TPM}
$U \leftarrow R^{sk} \cdot S^{v_{TPM}}$
$N_I \leftarrow P_i^{sk}$
choose random r_{sk} , r_v
$U' \leftarrow R^{r_{sk}} \cdot S^{r_v}$
$N_I' \leftarrow P_I^{r_{sk}}$

choose random n_{TPM}
$c \leftarrow H(c_H , n_{TPM})$
$s_{sk} \leftarrow r_{sk} + c \cdot sk$
$s_v \leftarrow r_v + c \cdot v_{TPM}$

$v \leftarrow v_{TPM} + v_I$
store (sk , v)

P_I

U , N_I , U' , N_I'

c_H

c , n_{TPM} , s_{sk} , s_v

v_I

Host (H)

- knows pk_I = (n , R , S , Z)
- wants to obtain DAA cert.

$P_I \leftarrow$ Hash(bsn_I)

$c_H \leftarrow$ Hash(n , R , S , U , N_I , U' , N_I' , n_I)

choose random nonce n_H

verify that e is prime
$A^* \leftarrow A^{c_I} \cdot [Z \cdot (U \cdot S^{v_I})^{-1}]^{s_e}$
verify that
c_I = H(n , R , S , Z , U , v_I , A , A^* , n_H)
if ok, proceed
store (A , e)

U , N_I

n_I

n_H , c , n_{TPM} , s_{sk} , s_v

c_I , s_e , (A , e , v_I)

I issues the DAA credential

Issuer (I)

- owns pk_I = (n , R , S , Z)
- knows sk_I = (p , q)
  ($n = p \cdot q$ is RSA modulus)

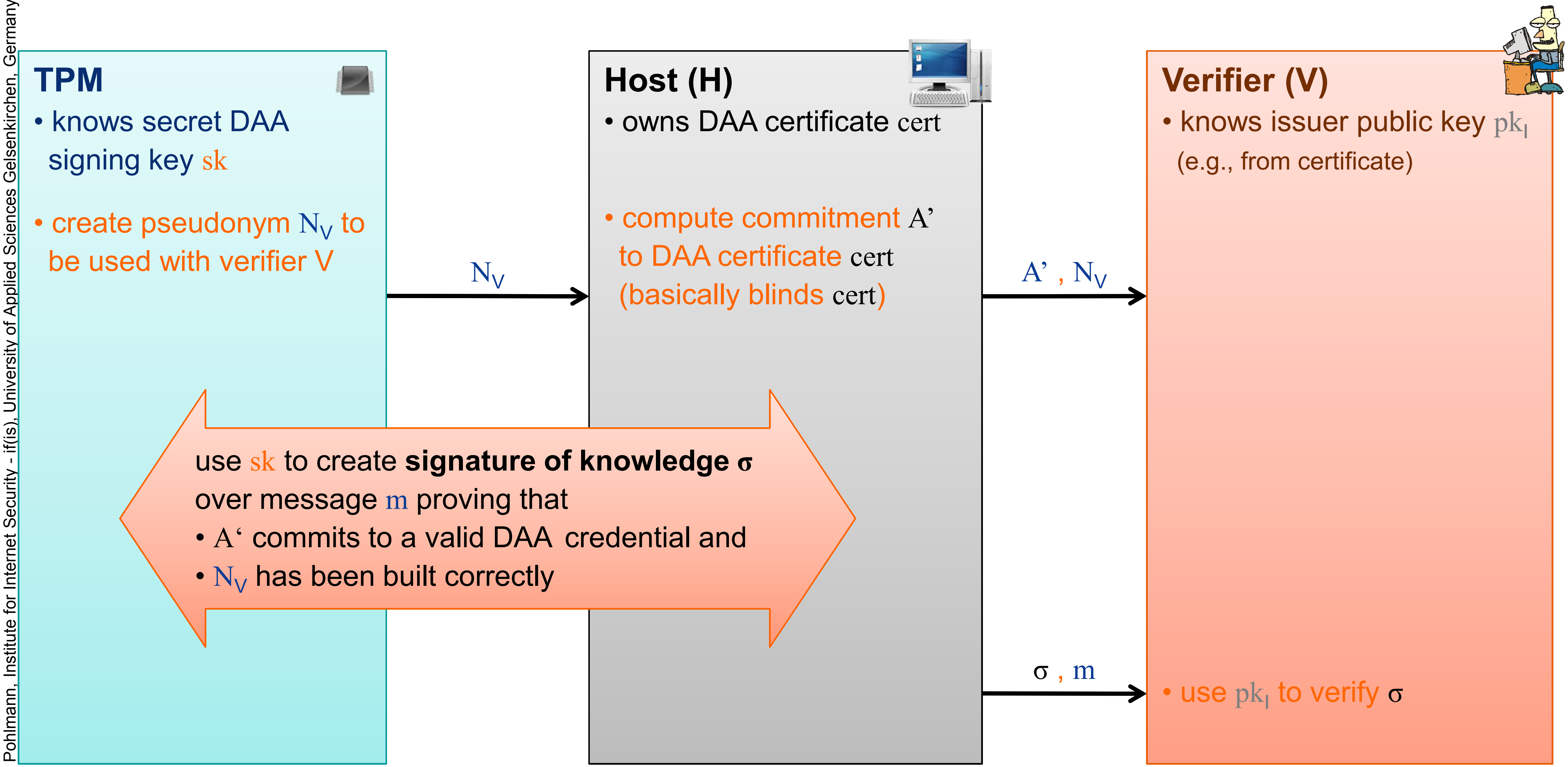
Issuer verifies that TPM has not been compromised

for all sk' on a rouge list check if $N_I = P_i^{sk'}$
if not, choose random nonce n_I

$U^* \leftarrow U^{-c} \cdot R^{s_{sk}} \cdot S^{s_v}$
$N_I^* \leftarrow N_I^{-c} \cdot P_I^{s_{sk}}$
verify that
c = H[H(n , R , S , Z , U , N_I , U^* , N_I^* , n_I) , n_{TPM})
if ok, proceed
chose random v_I , r_e and prime e
$A \leftarrow [Z \cdot (U \cdot S^{v_I})^{-1}]^{1/e}$
$A' \leftarrow [Z \cdot (U \cdot S^{v_I})^{-1}]^{r_e}$
$c_I \leftarrow$ H(n , R , S , Z , U , v_I , A , A' , n_H)
$s_e \leftarrow r_e - c_I \cdot e^{-1}$

can be computed only if p, q are known

DAA Sign Protocol – Overview



TPM

- knows secret DAA signing key sk
- create pseudonym N_V to be used with verifier V

Host (H)

- owns DAA certificate $cert$
- compute commitment A' to DAA certificate $cert$ (basically blinds $cert$)

Verifier (V)

- knows issuer public key pk_I (e.g., from certificate)
- use pk_I to verify σ

use sk to create **signature of knowledge σ** over message m proving that

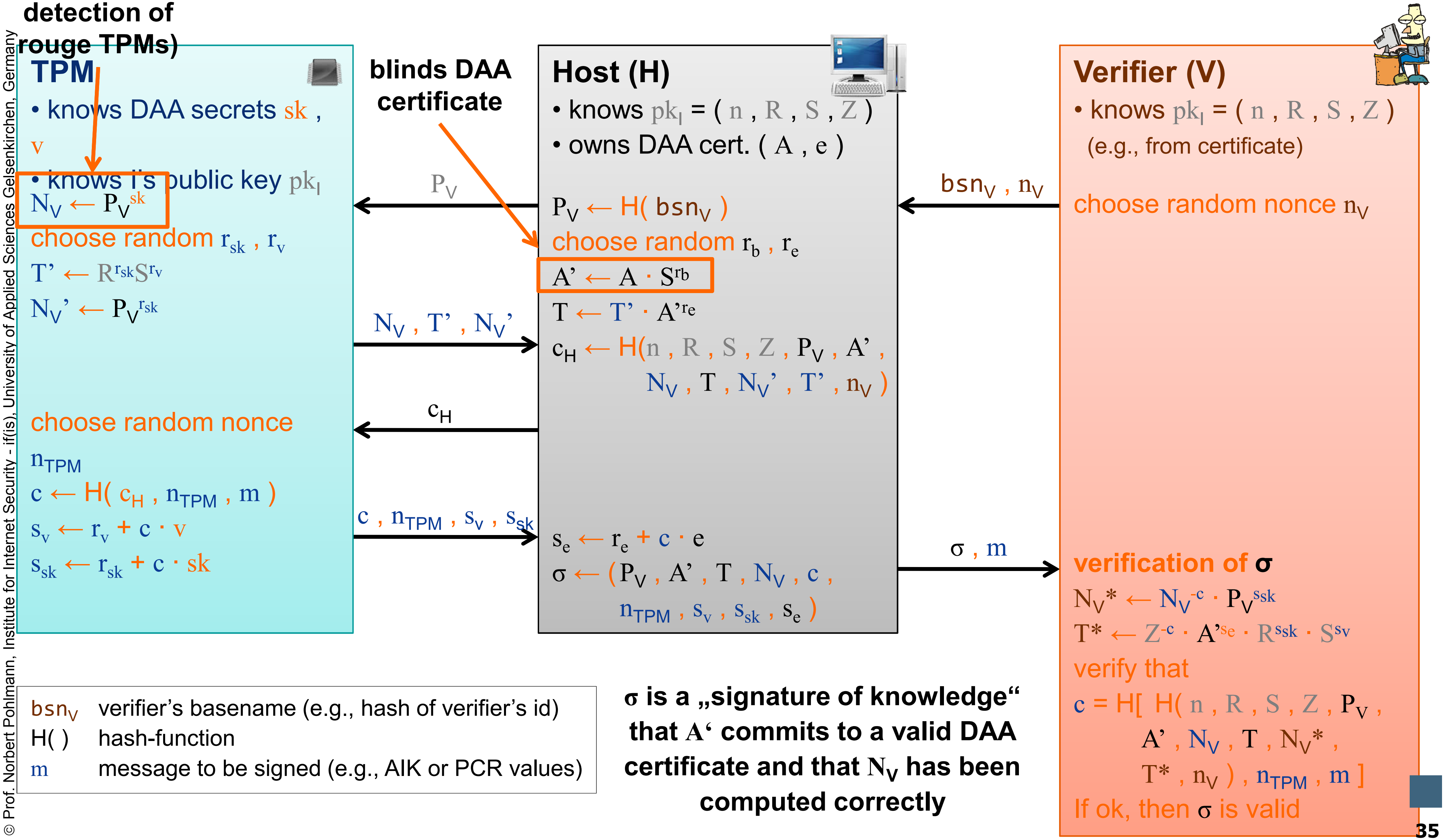
- A' commits to a valid DAA credential and
- N_V has been built correctly

message m may be an AIK or some PCR values

DAA Sign Protocol (Simplified)

TPM's pseudonym (enables detection of rouge TPMs)

blinds DAA certificate



TPM

- knows DAA secrets sk , v
- knows I's public key pk_I

$N_V \leftarrow P_V^{sk}$

choose random r_{sk} , r_v

$T' \leftarrow R^{r_{sk}} S^{r_v}$

$N_V' \leftarrow P_V^{r_{sk}}$

choose random nonce n_{TPM}

$c \leftarrow H(c_H , n_{TPM} , m)$

$s_v \leftarrow r_v + c \cdot v$

$s_{sk} \leftarrow r_{sk} + c \cdot sk$

Host (H)

- knows $pk_I = (n , R , S , Z)$
- owns DAA cert. (A , e)

$P_V \leftarrow H(bsn_V)$

choose random r_b , r_e

$A' \leftarrow A \cdot S^{r_b}$

$T \leftarrow T' \cdot A'^{r_e}$

$c_H \leftarrow H(n , R , S , Z , P_V , A' , N_V , T , N_V' , T' , n_V)$

$s_e \leftarrow r_e + c \cdot e$

$\sigma \leftarrow (P_V , A' , T , N_V , c , n_{TPM} , s_v , s_{sk} , s_e)$

Verifier (V)

- knows $pk_I = (n , R , S , Z)$ (e.g., from certificate)

choose random nonce n_V

verification of σ

$N_V^* \leftarrow N_V^{-c} \cdot P_V^{s_{sk}}$

$T^* \leftarrow Z^{-c} \cdot A'^{s_e} \cdot R^{s_{sk}} \cdot S^{s_v}$

verify that

$c = H[H(n , R , S , Z , P_V , A' , N_V , T , N_V^* , T^* , n_V) , n_{TPM} , m]$

If ok, then σ is valid

bsn_V	verifier's basename (e.g., hash of verifier's id)
H( )	hash-function
m	message to be signed (e.g., AIK or PCR values)

σ is a „signature of knowledge" that A' commits to a valid DAA certificate and that N_V has been computed correctly

Content

- Aim and outcomes of this lecture
- Authenticated Boot
- Binding and Sealing
- Integrity Reporting/Attestation
- Direct Anonymous Attestation (DAA)
- **Summary**

TCG Functionalities
→ Summary

- The Trusted Computing functionalities helps to make the integrity level of IT system higher.
- **Binding**
  - May be used to bind data to a specific TPM/platform
    - Data encrypted with non-migratable key can only be recovered by TPM that knows corresponding secret key
  - **Usually no platform binding**
    - Since binding can also be used with migratable keys
- **Sealing** (extension of binding)
  - Always binds data to a specific TPM/platform
    - **Sealing can only be used with non-migratable storage keys**
  - Configuration of encrypting platform can be verified
    - Ciphertext includes platform's state at the time of encryption

Trusted Computing Group

→ Functionalities

Thank you for your attention!
Questions?

Prof. Dr. (TU NN)
Norbert Pohlmann

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

internet security.

TCG Functionalities
→ 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/