CS 453/698: Software and Systems Security

Module: Hardware & Mobile Security

Lecture: Trusted Platform Modules (TPMs)

Adam Caulfield

University of Waterloo

Spring 2025

Reminders & Recap

Reminders:

- A3 is due on July 11
- Send your research project proposals to Meng and me!

Recap – last time we covered:

Software supply chain security

- What is it?
- Some models:
  - General software supply chain model
  - Open-source software supply chain model
- Attacks
- Safeguards
  - Classifications
  - Examples – reproducible builds, in-toto

Today

Start: Hardware and Mobile Security

In-toto: attestation or authentication?

Today

Start: Hardware and Mobile Security

In-toto: attestation or authentication?

- Attestation – why?

Today

Start: Hardware and Mobile Security

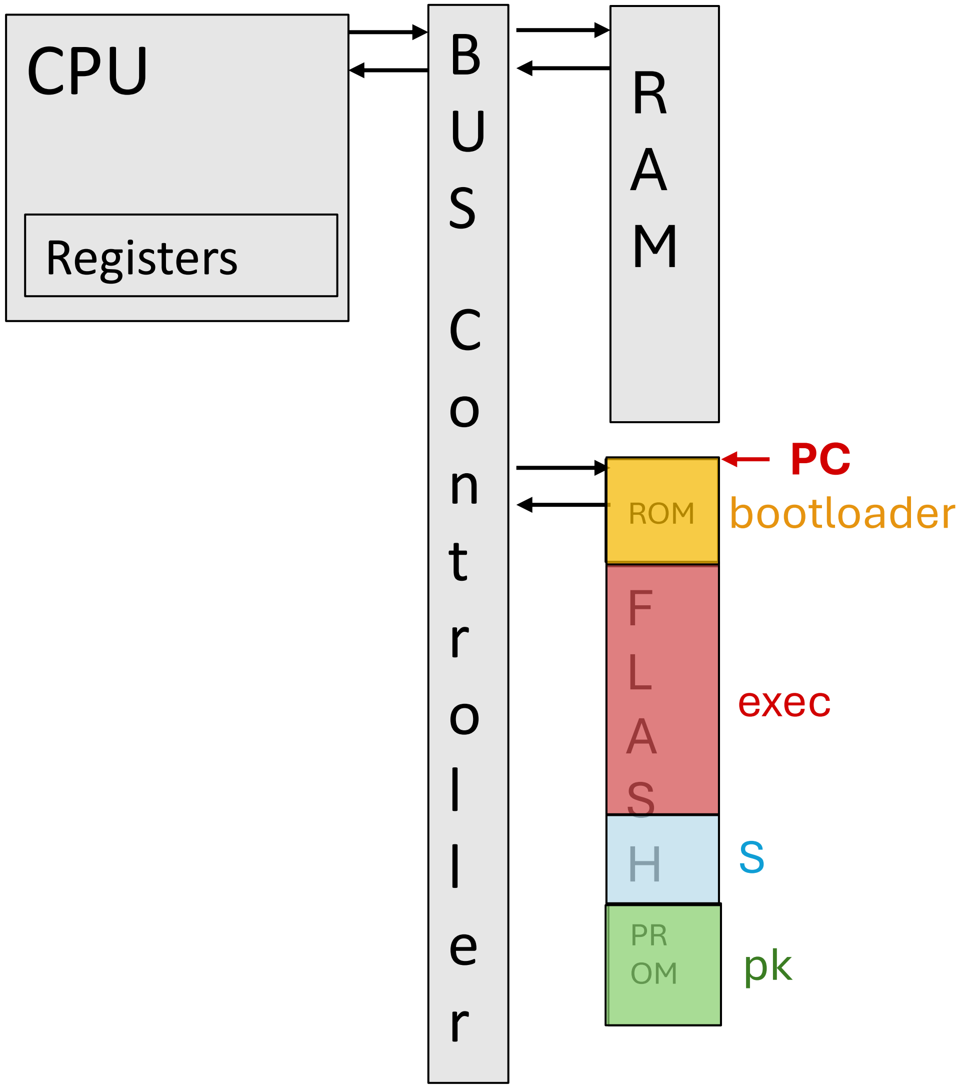
In-toto: attestation or authentication?

- Attestation – why?
- Security requirements – ***Attestation Root of Trust (RoT)***
  - Secure storage of secret/signing keys
  - Secure run-time environment
  - Required to prove to end-users

How to get there?

- Secure boot?
  - Performs measurement
  - Only starts running if passes a validity check on the measurement

Secure Boot

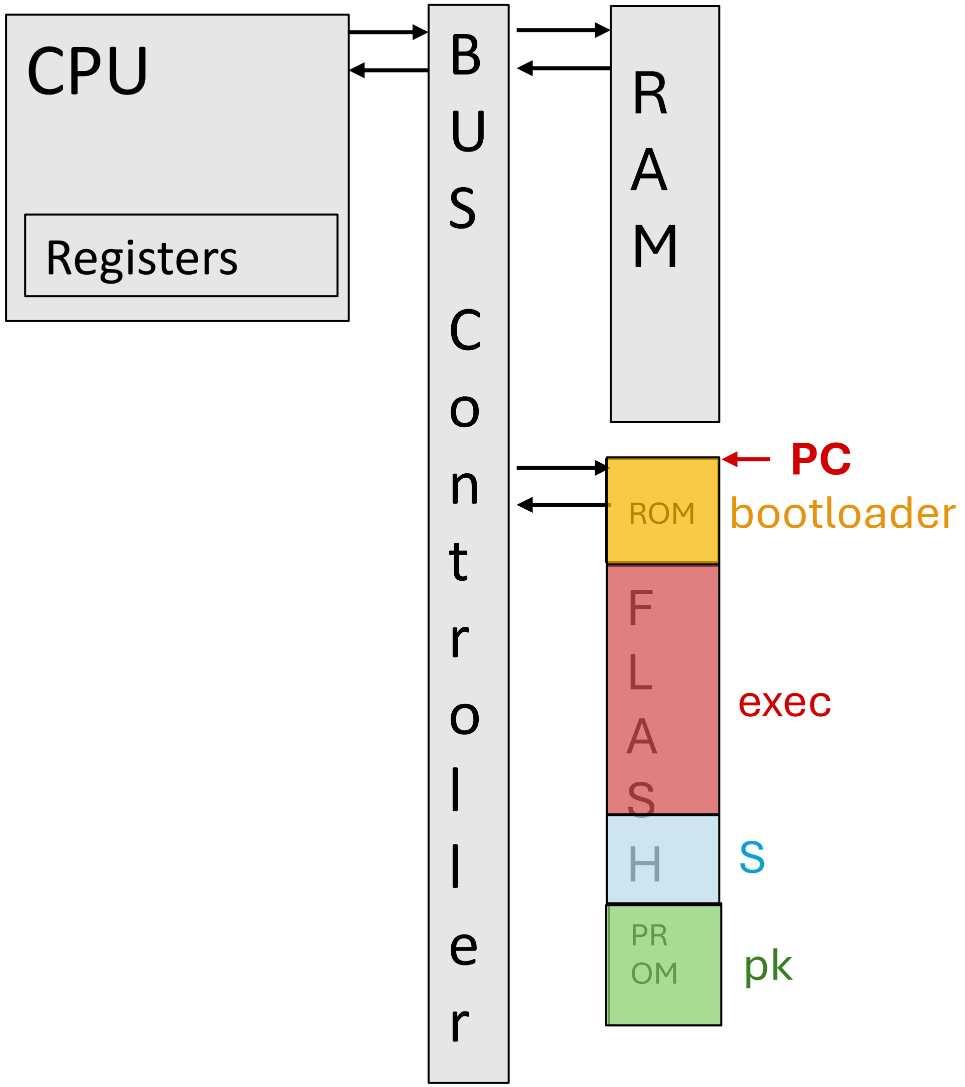


Recall simple secure boot:

Process:

- Device is installed with a **pk**
- Programmed with (**exec**, **S**)
- Boot: points PC to the **bootloader** code
- **Bootloader** code performs verification using **pk**
- If pass, begins executing **exec**

Secure Boot



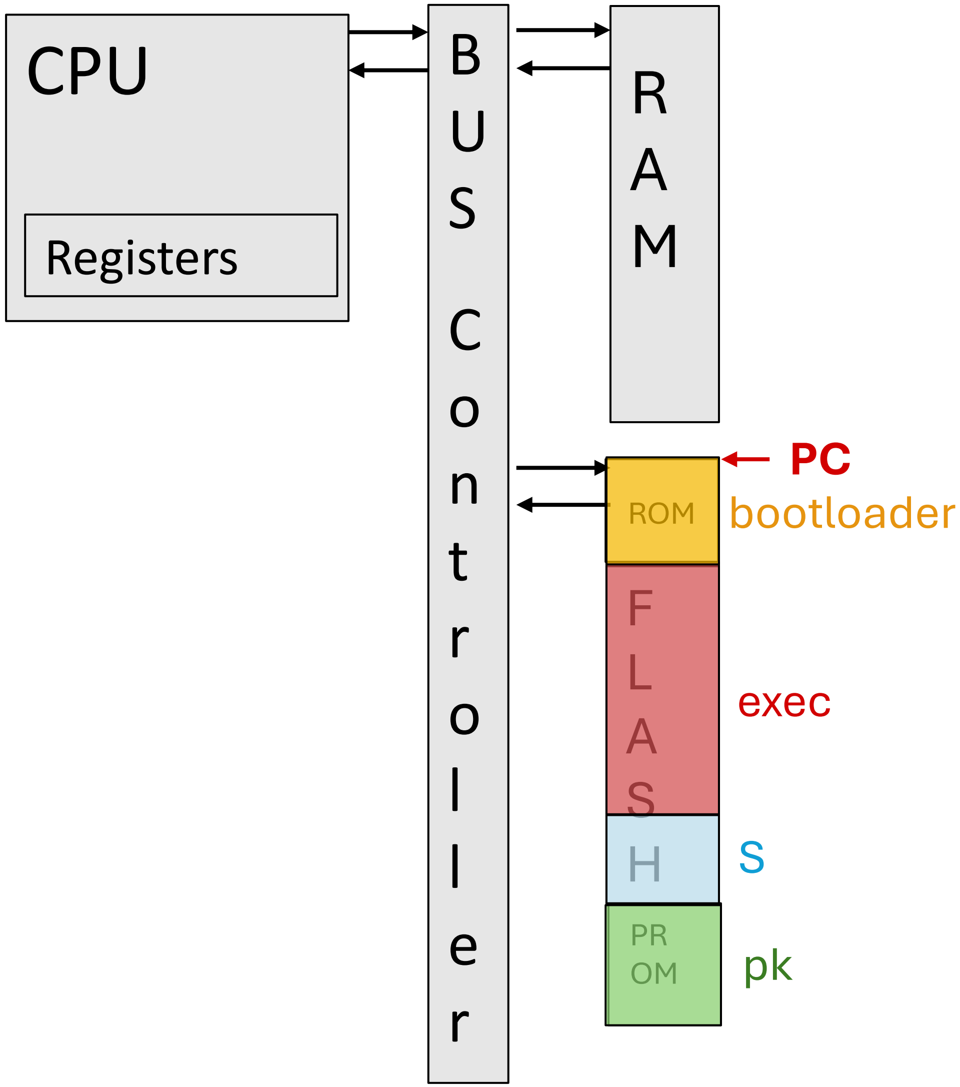
Recall simple secure boot:

Process:

- Device is installed with a **pk**
- Programmed with (**exec**, **S**)
- Boot: points PC to the **bootloader** code
- **Bootloader** code performs verification using **pk**
- If pass, begins executing **exec**

Can these components be used to prove exec is valid to someone else? (i.e., to get an attestation RoT?)

Secure Boot



Recall simple secure boot:

Process:

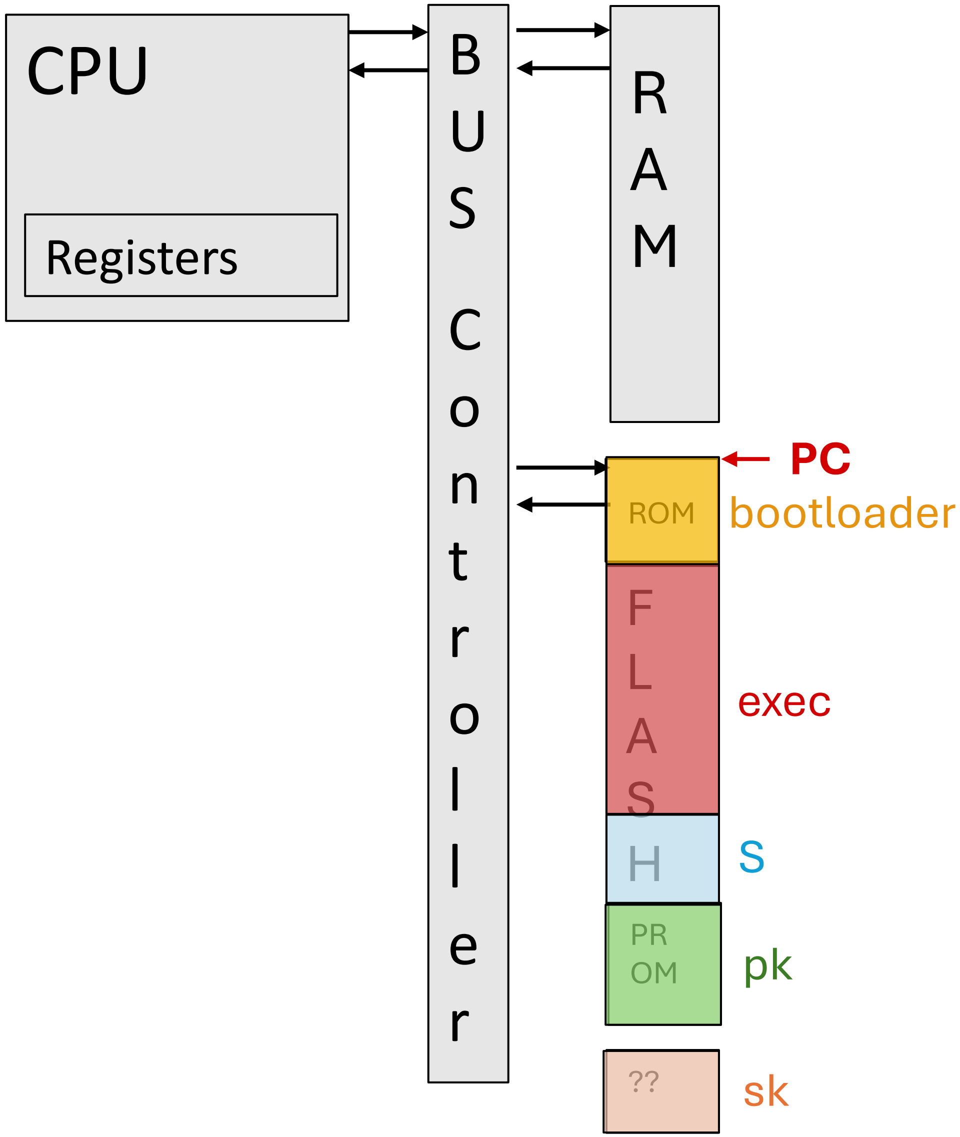
- Device is installed with a **pk**
- Programmed with (**exec**, **S**)
- Boot: points PC to the **bootloader** code
- **Bootloader** code performs verification using **pk**
- If pass, begins executing **exec**

Can these components be used to prove exec is valid to someone else? (i.e., to get an attestation RoT?)

No → requires the device to produce its own signature

- Requires secure storage of secret key

Secure Boot

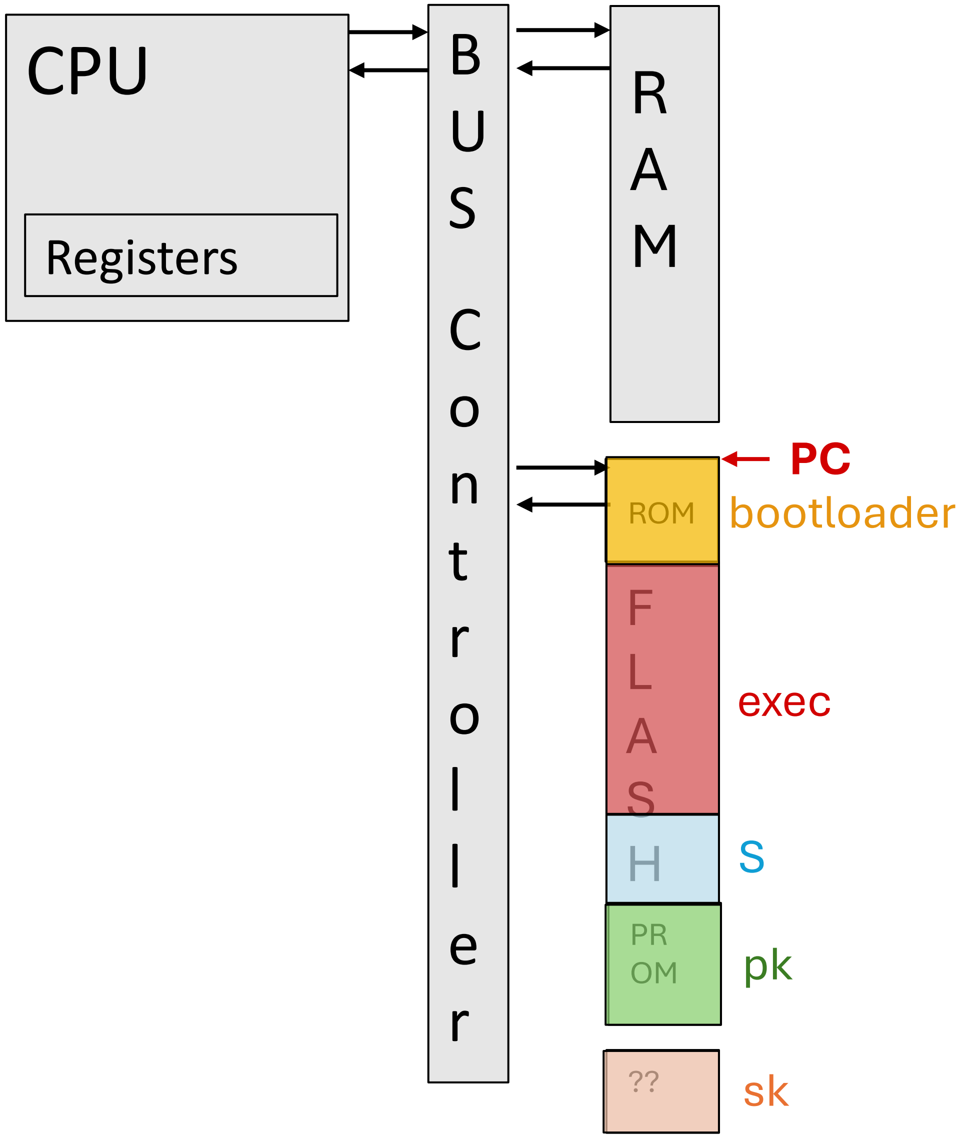


Recall simple secure boot:

So, what do we need?

- A secret key on our device
- Some way to securely store it
- Some way to securely use it

Secure Boot



Recall simple secure boot:

So, what do we need?

- A secret key on our device
- Some way to securely store it
- Some way to securely use it

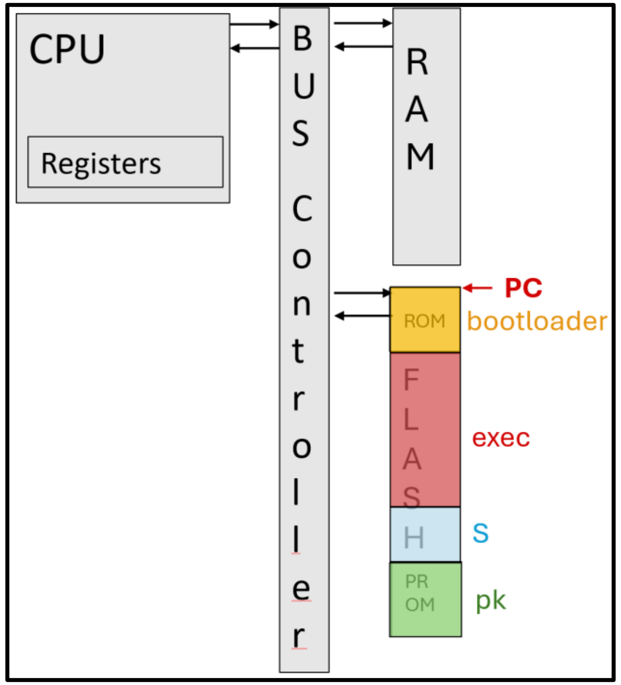
How?

Getting an Attestation RoT...

Option 1:

Keep modifying the secure boot architecture until it meets the reqs.

- Possible, but tricky (especially for higher-end devices)

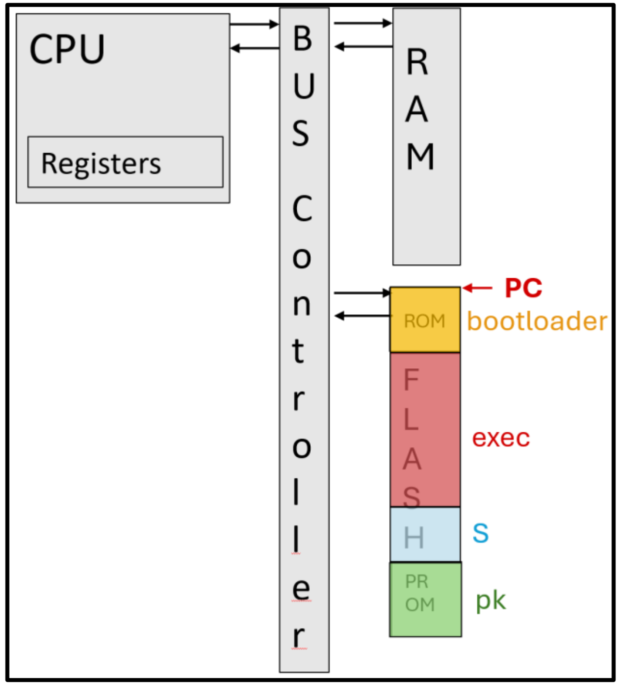


Getting an Attestation RoT...

Option 1:

Keep modifying the secure boot architecture until it meets the reqs.

- Possible, but tricky (especially for higher-end devices)



Option 2:

Use separate purpose-specific cryptographic co-processor to store/compute on secrets.

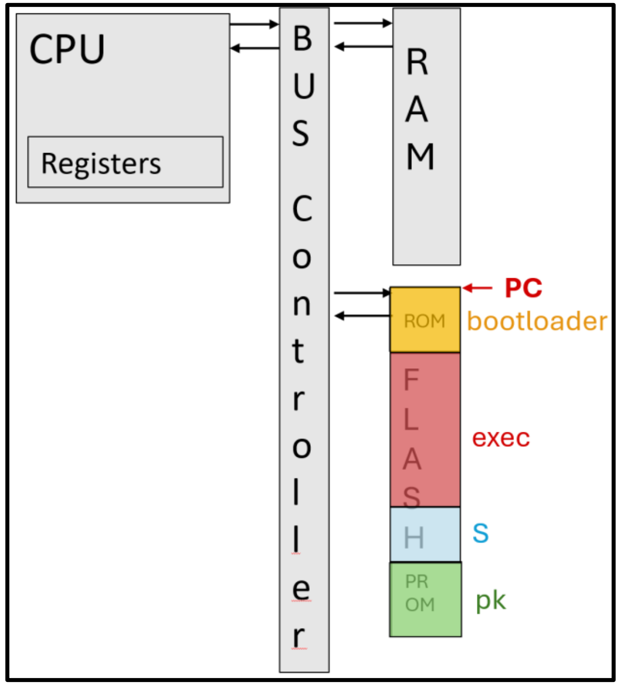
- Isolated & independent from the main system

Getting an Attestation RoT...

Option 1:

Keep modifying the secure boot architecture until it meets the reqs.

- Possible, but tricky (especially for higher-end devices)

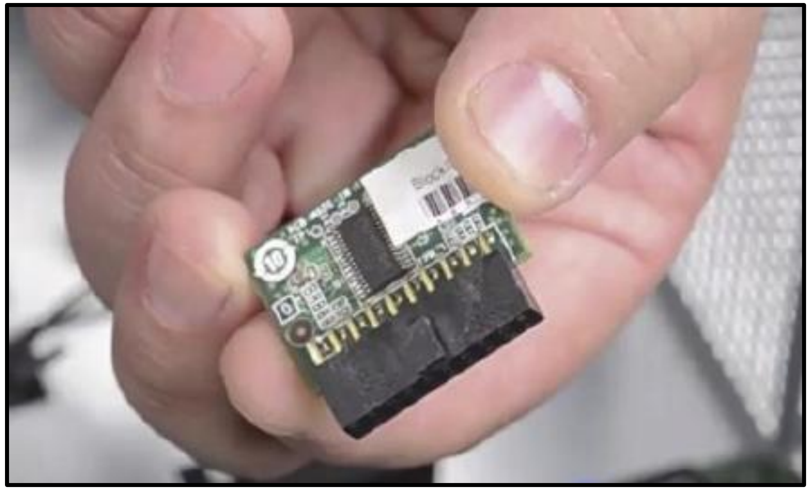


Option 2:

Use separate purpose-specific cryptographic co-processor to store/compute on secrets.

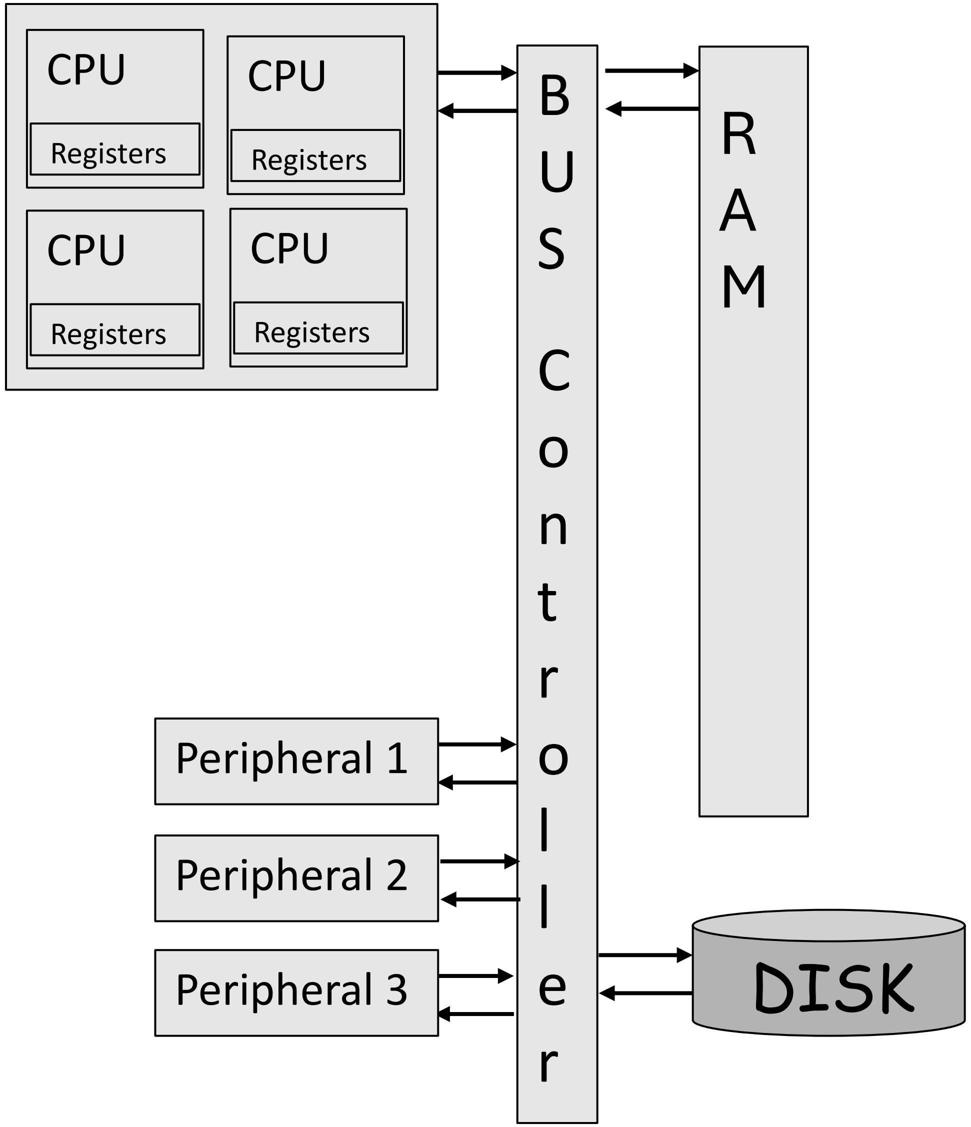
- Isolated & independent from the main system

Trusted Platform Module (TPM)



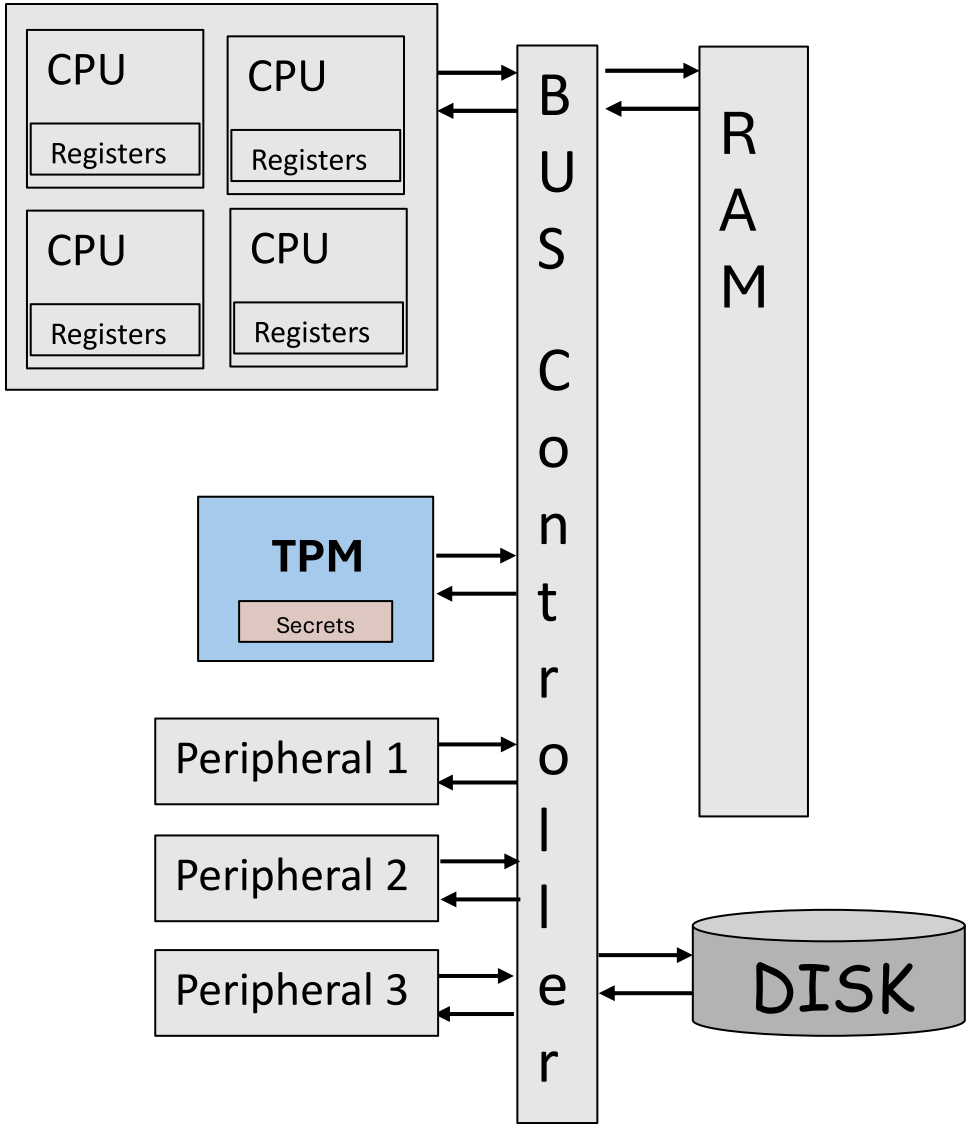
Completely isolated & independent from the main system & CPU

Computer model with TPM



Computer model with TPM

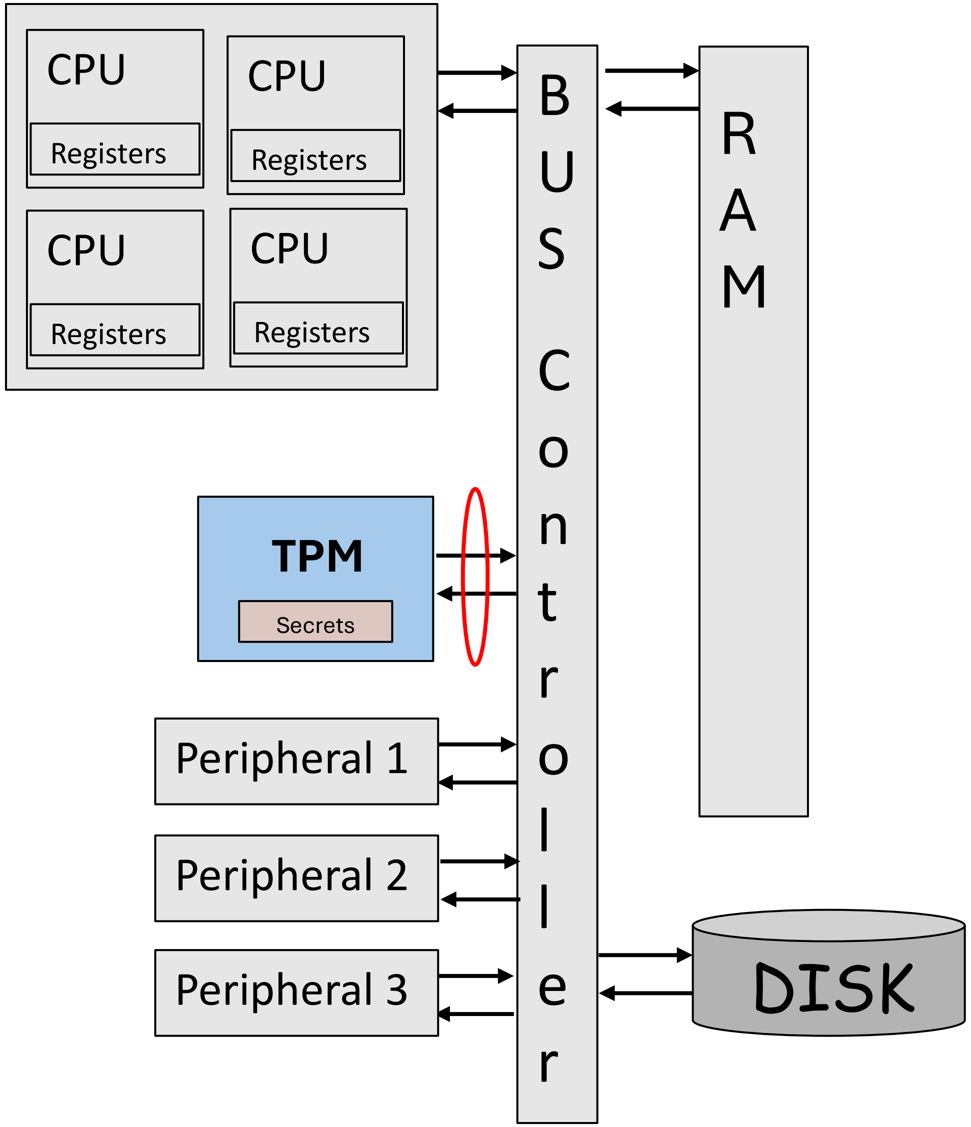
Separate module, with some stored **secret**



Computer model with TPM

Separate module, with some stored **secret**

<u>**Separate Interface**</u>

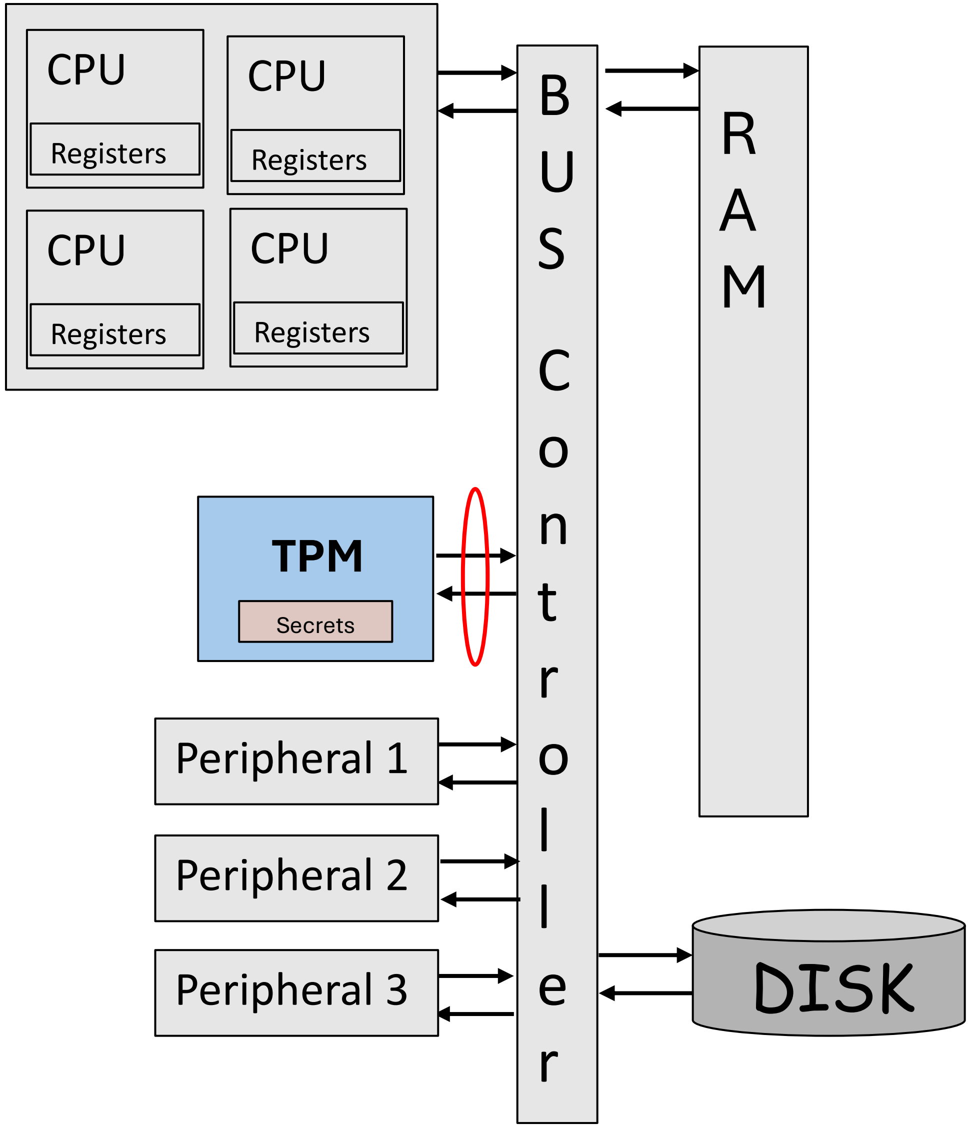


Computer model with TPM

Separate module, with some stored **secret**

<u>Separate Interface</u>

TPM is a ***<u>passive</u>*** device that only responds to ***<u>small</u>*** & ***<u>well-defined</u>*** requests issued by the main systems



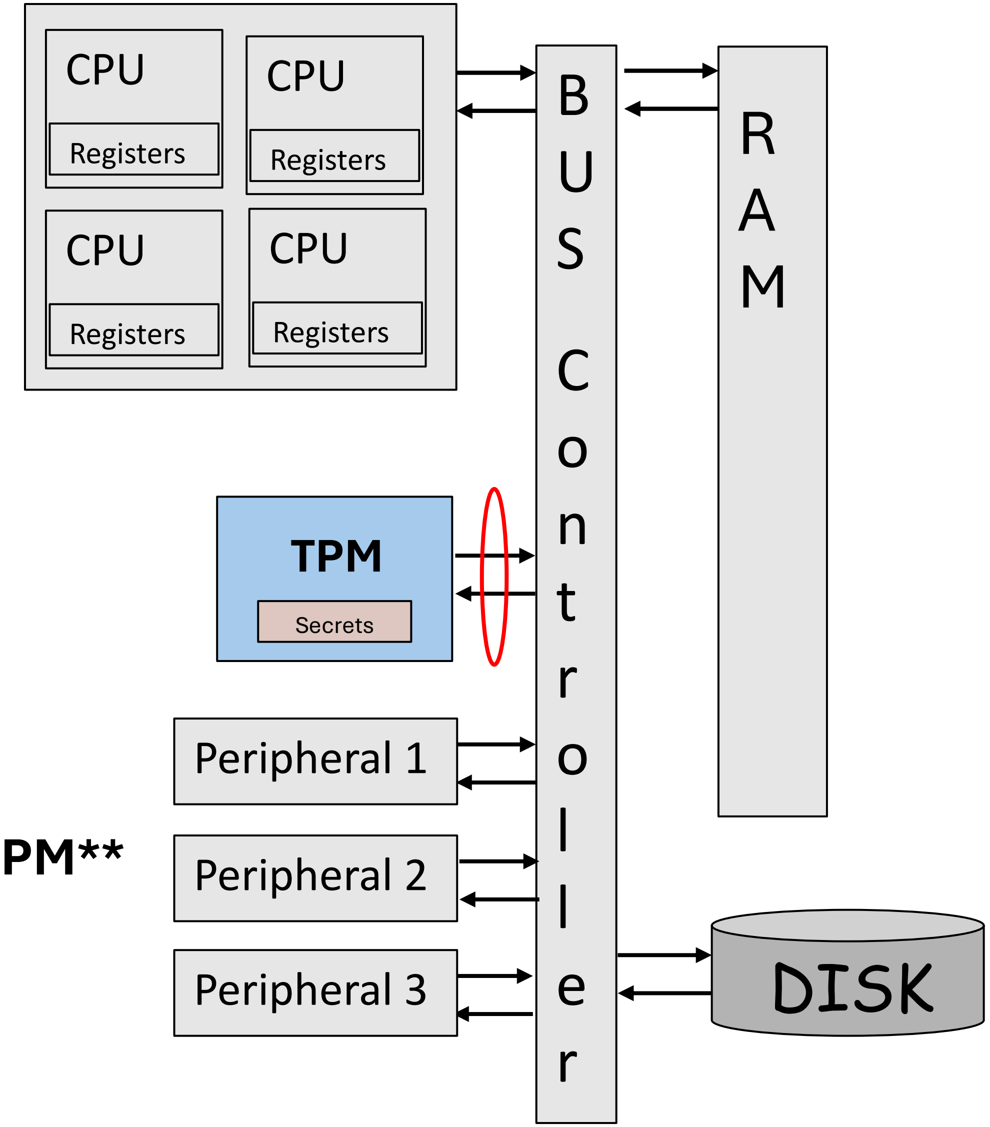
Computer model with TPM

Separate module, with some stored **secret**

<u>Separate Interface</u>

TPM is a ***<u>passive</u>*** device that only responds to ***<u>small</u>*** & ***<u>well-defined</u>*** requests issued by the main systems

****Secrets never leave the TPM****



Trusted Platform Module (TPM)

What is TPM?

- A cryptographic co-processor

- **NOT** a crypto accelerator

- **NOT** a general-purpose processor
  - Hard-coded & pre-defined functionality

Trusted Platform Module (TPM)

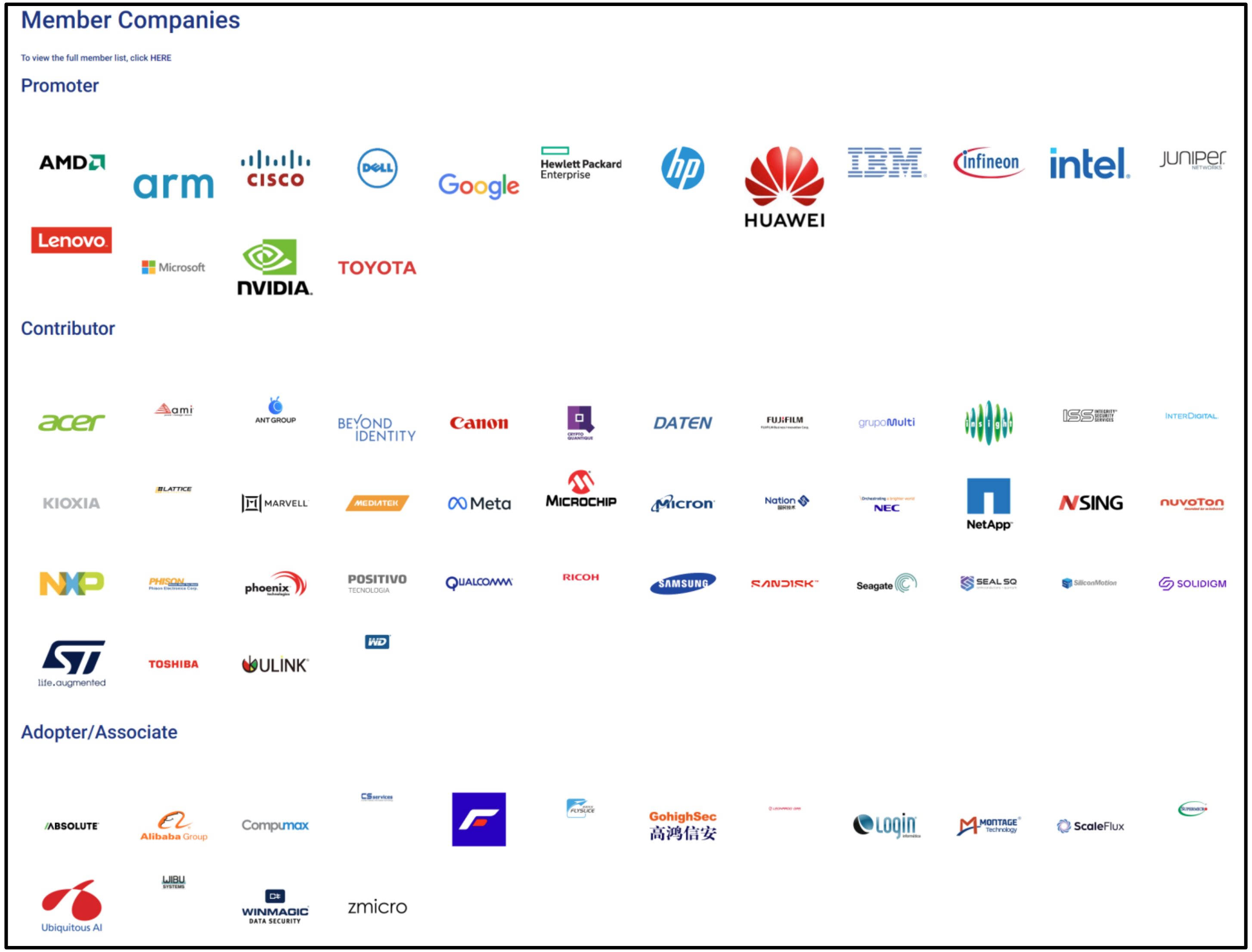
Trusted Computing Group (TCG)

- TPM was conceived by a computing industry consortium called the Trusted Computing Group

- A hardware anchor (RoT) on which secure systems could be built

- First version was standardized in 2009

Trusted Platform Module (TPM)



Member Companies

To view the full member list, click HERE

Promoter

Contributor

Adopter/Associate

From https://trustedcomputinggroup.org/membership/member-companies/

Trusted Platform Module (TPM)

Trusted Computing Group (TCG)

- TPM was conceived by a computing industry consortium called the Trusted Computing Group

- A hardware anchor (RoT) on which secure systems could be built

- First version was standardized in 2009

- TCG specifies a standard that TPM manufacturers should follow

Trusted Platform Module (TPM)

TCG TPM Releases

When TCD releases a new version of the TPM spec, it is divided into:

- Part 0: Introduction
- Part 1: Design Principles of TPM Architecture
- Part 2: Structures of the TPM
- Part 3: Commands (how to talk to the TPM)

Continuously revised to enhance its security and keep up with current needs

- TPM 1.2 (2005):
  - Hashing → SHA-1 (no longer considered secure)
  - Signing → RSA
- TPM 2.0 (2014)
  - Hashing → SHA-256
  - Signing → RSA or ECC

Trusted Platform Module (TPM)

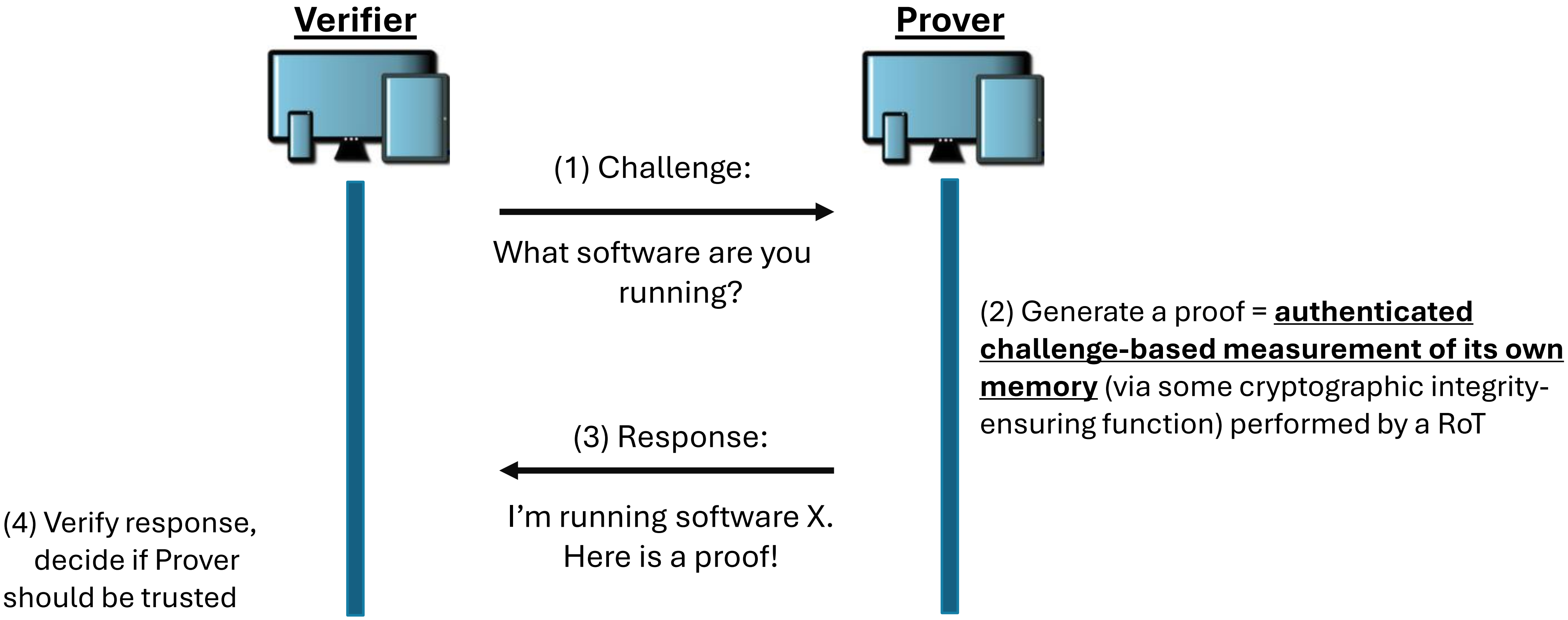
TPM Functions and uses:

- Hardware random number generation
- Secure generation of cryptographic keys (RSA, ECC)
- **Remote Attestation**
- **Binding:**
  - Encryption of data using a “TPM bind key”
- **Sealing:**
  - Similar to binding
  - Decryption only possible once certain TPM state has been reached

Plus anything else that one may come up with by combining these features!

Remote Attestation

Recall from previous lecture



Verifier

Prover

(1) Challenge:

What software are you running?

(2) Generate a proof = **authenticated challenge-based measurement of its own memory** (via some cryptographic integrity-ensuring function) performed by a RoT

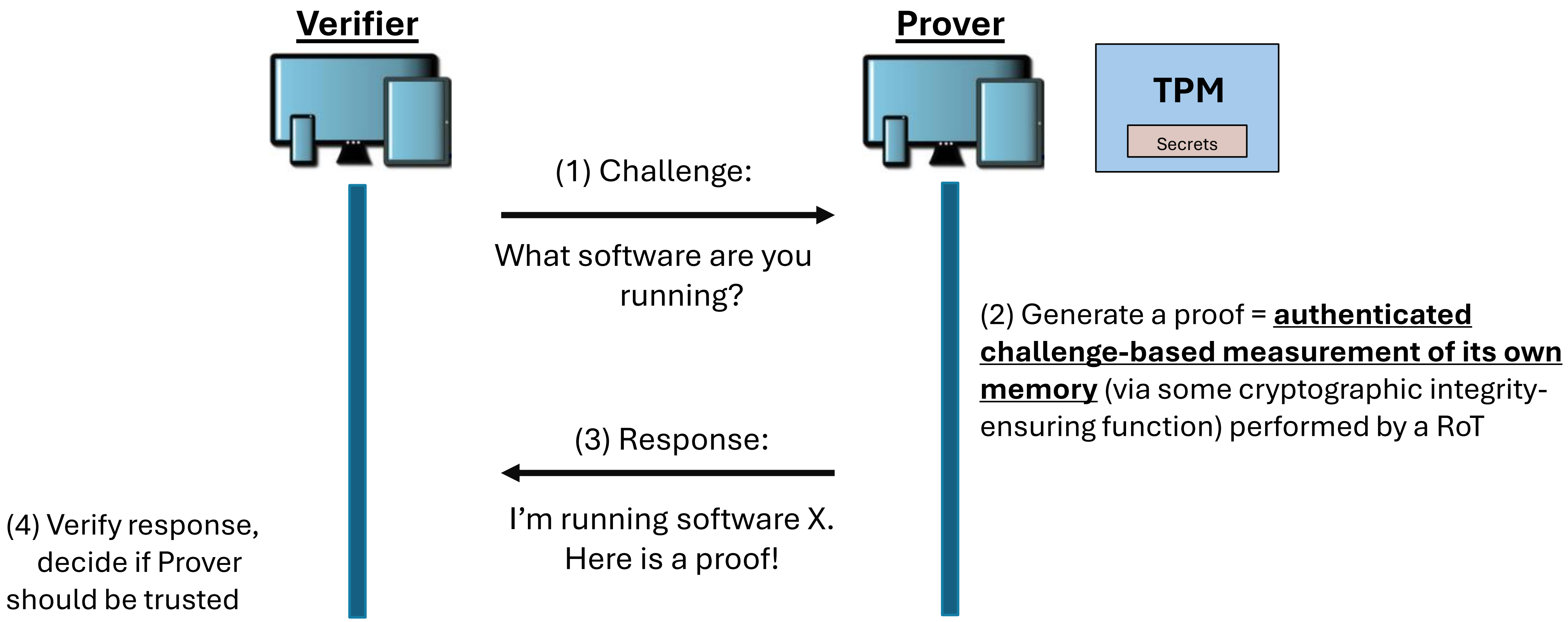
(3) Response:

I'm running software X. Here is a proof!

(4) Verify response, decide if Prover should be trusted

Remote Attestation

Now assume Prover has TPM…



Verifier

Prover

TPM

Secrets

(1) Challenge:

What software are you running?

(2) Generate a proof = **authenticated challenge-based measurement of its own memory** (via some cryptographic integrity-ensuring function) performed by a RoT

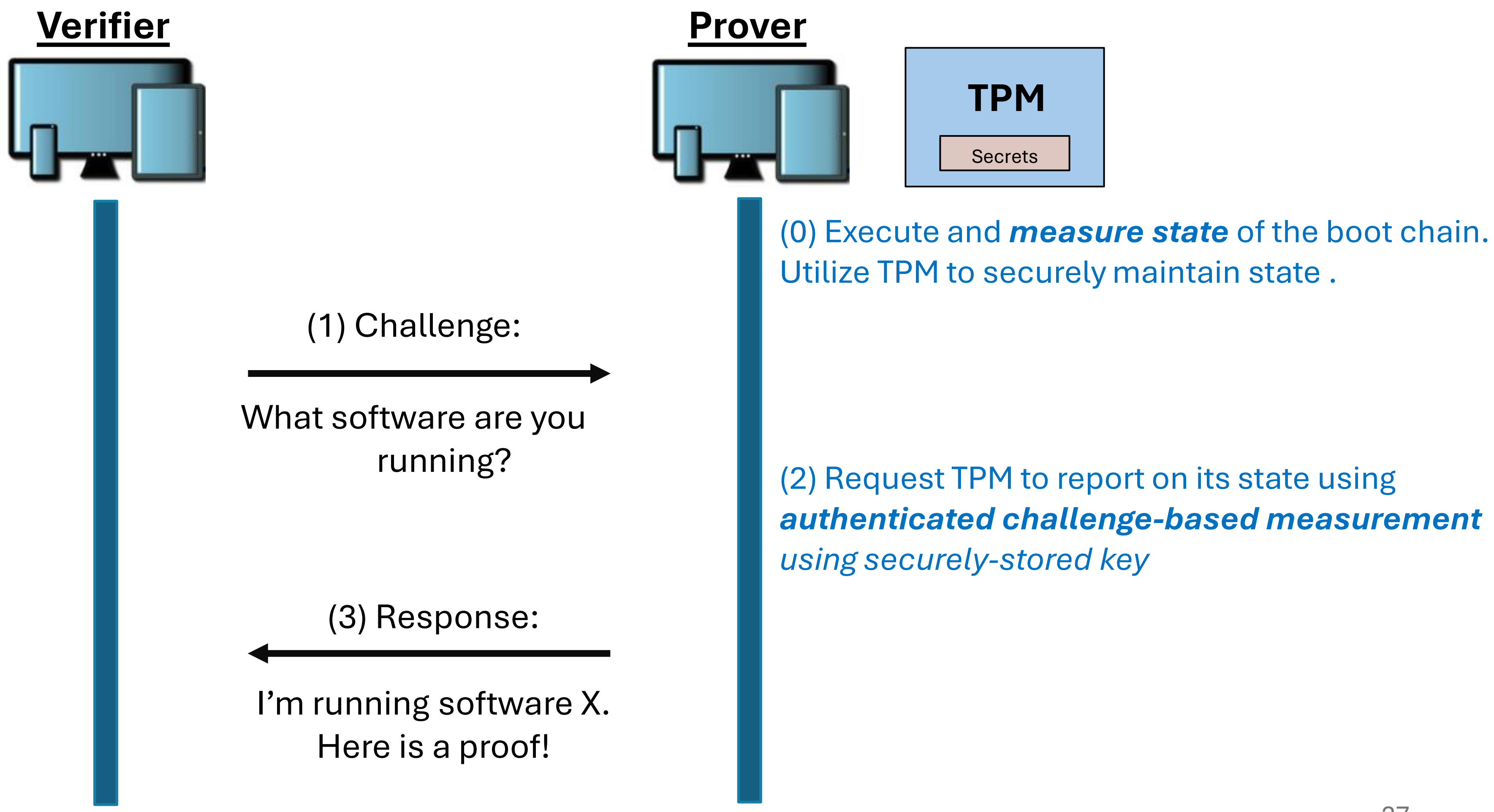
(3) Response:

I’m running software X. Here is a proof!

(4) Verify response, decide if Prover should be trusted

Remote Attestation

Now assume Prover has TPM… slightly modified



Verifier

Prover

TPM

Secrets

(0) Execute and ***measure state*** of the boot chain. Utilize TPM to securely maintain state .

(1) Challenge:

What software are you running?

(2) Request TPM to report on its state using ***authenticated challenge-based measurement*** *using securely-stored key*

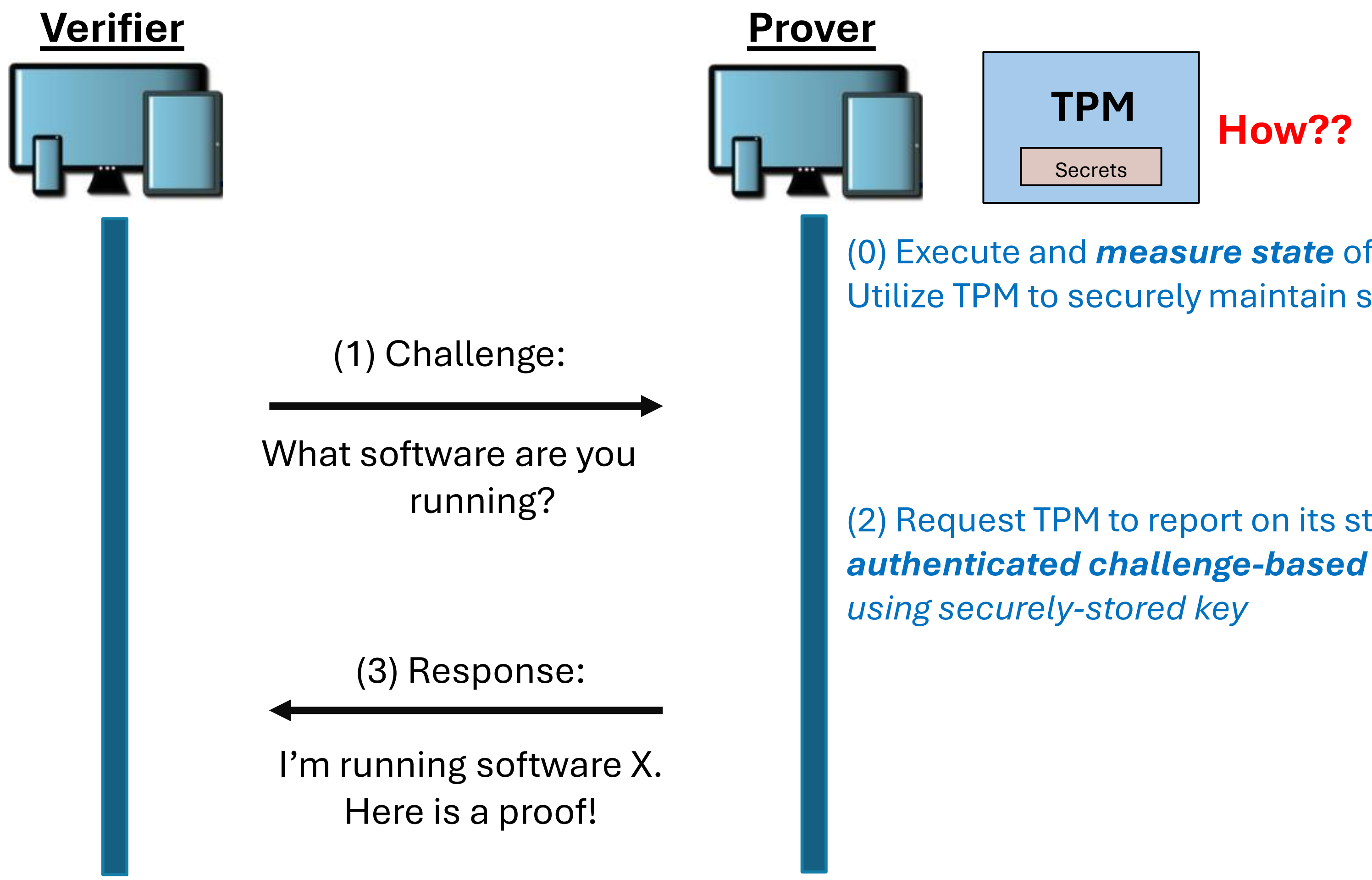
(3) Response:

I'm running software X. Here is a proof!

(4) Verify response, decide if Prover should be trusted

Remote Attestation

Now assume Prover has TPM… slightly modified



Verifier

Prover

TPM

Secrets

How??

(0) Execute and ***measure state*** of the boot chain. Utilize TPM to securely maintain state .

(1) Challenge:

What software are you running?

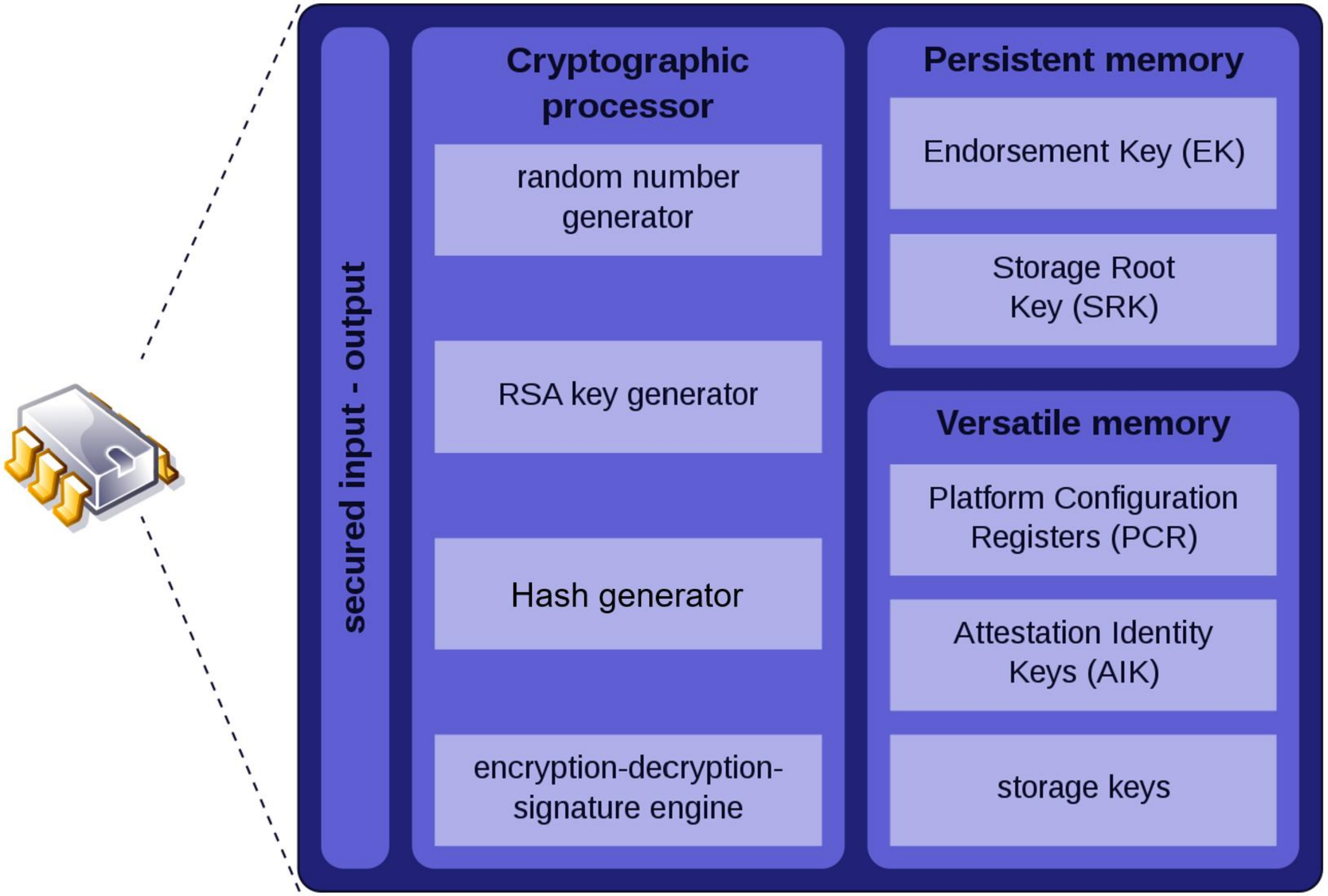
(2) Request TPM to report on its state using ***authenticated challenge-based measurement*** *using securely-stored key*

(3) Response:

I'm running software X. Here is a proof!

(4) Verify response, decide if Prover should be trusted

TPM Architecture



secured input - output

Cryptographic processor

random number generator

RSA key generator

Hash generator

encryption-decryption-signature engine

Persistent memory

Endorsement Key (EK)

Storage Root Key (SRK)

Versatile memory

Platform Configuration Registers (PCR)

Attestation Identity Keys (AIK)

storage keys

TPM Architecture

TPM Provides:

- A **Root of Trust for Storage**
  - Secure TPM encryption key

- A **Root of Trust for Reporting**
  - Secure TPM signing key (used to establish TPM's identity)

- **TPM State**
  - Limited internal storage
  - Loading & storing keys
  - **Platform Configuration Registers (PCR)**

TPM Architecture

TPM Provides:

- A **Root of Trust for Storage**
  - Secure TPM encryption key
- A **Root of Trust for Reporting**
  - Secure TPM signing key (used to establish TPM's identity)
- **TPM State**
  - Limited internal storage
  - Loading & storing keys
  - **Platform Configuration Registers (PCR)**

TPM Architecture

Root of Trust for Storage:

- Core question:
  - How are the secrets actually kept secret?
- TCG: Can we store them all locally (i.e., internal to the TPM)?
  - It depends… how many secrets do we need to keep secret?
  - TCG: “hmm more than three?” → need ability to store arbitrary number of secrets

TPM Architecture

Root of Trust for Storage:

- Core question:
  - How are the secrets actually kept secret?

- TCG: Can we store them all locally (i.e., internal to the TPM)?
  - It depends… how many secrets do we need to keep secret?
  - TCG: “hmm more than three?” → need ability to store arbitrary number of secrets

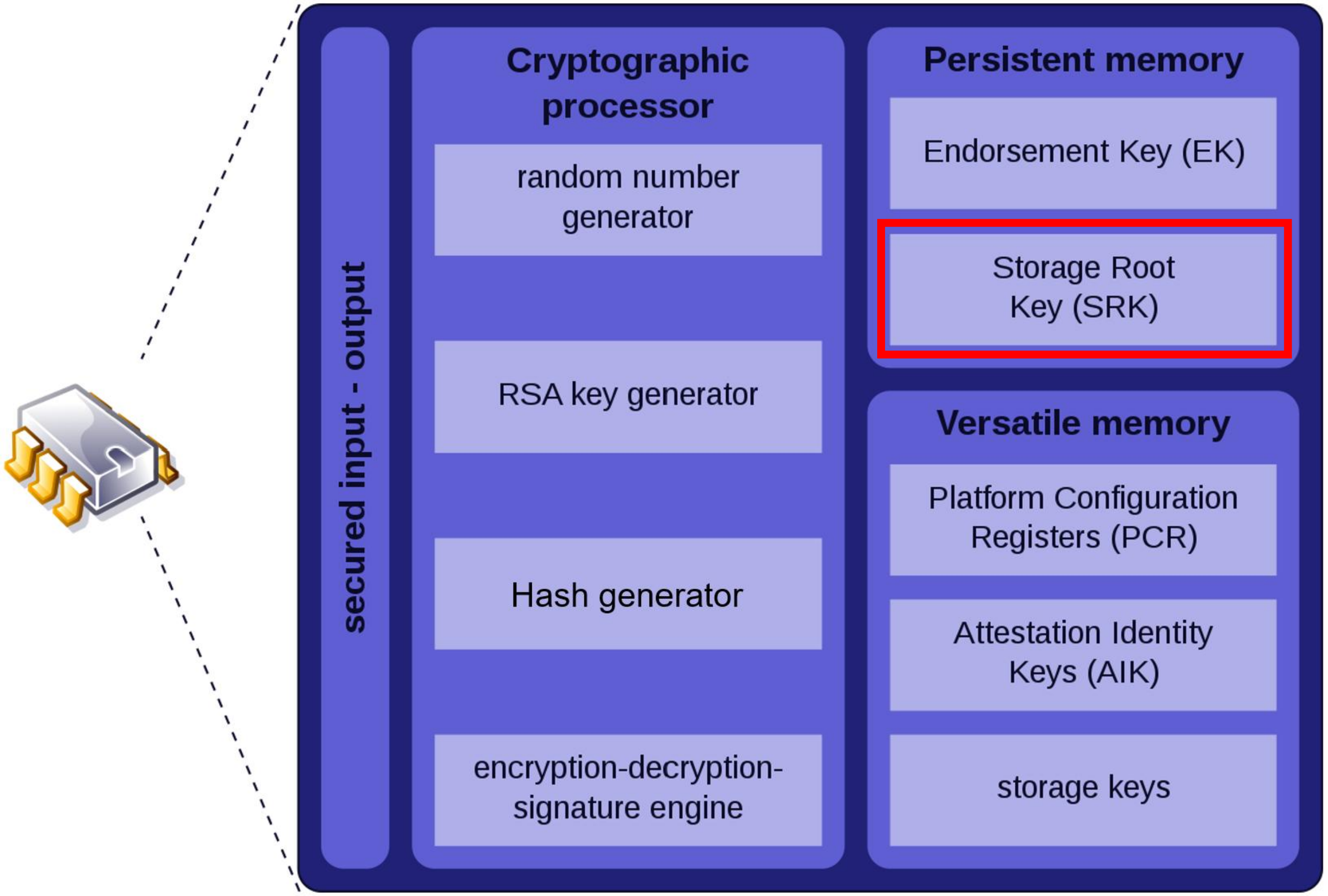
- **TPM as a Root of Trust for Storage**
  - Does not store all secrets directly
  - Stores ***one main secret*** used to protect other secrets in the system
  - Other secrets then can be stored outside the TPM (e.g., Disk)
  - Secrets stored outside are encrypted under the TPM main secret

The **“root secret”** helps ensure the **confidentiality** of other secrets in external storage

Hence, **Root of Trust**

TPM Architecture

Root of Trust for Storage



TPM Architecture

Storage Root Key (SRK)

- Burned inside TPM persistent memory by manufacturer
- Never leaves the TPM
- Provides confidentiality of externally stored keys

Other new keys are generated by the TPM

TPM Architecture

Storage Root Key (SRK)

- Burned inside TPM persistent memory by manufacturer
- Never leaves the TPM
- Provides confidentiality of externally stored keys

Other new keys are generated by the TPM

- E.g., RSA keys: (PK, SK) pairs
- Stored outside the TPM
- How? → encrypt the private half (SK)

  $[\ \mathrm{Enc}_{SRK}(SK_1), PK_1\ ] \rightarrow \mathrm{blob}_1$
  $[\ \mathrm{Enc}_{SRK}(SK_2), PK_2\ ] \rightarrow \mathrm{blob}_2$
  …
  $[\ \mathrm{Enc}_{SRK}(SK_N), PK_N\ ] \rightarrow \mathrm{blob}_N$

TPM Architecture

Storage Root Key (SRK)

- Burned inside TPM persistent memory by manufacturer
- Never leaves the TPM
- Provides confidentiality of externally stored keys

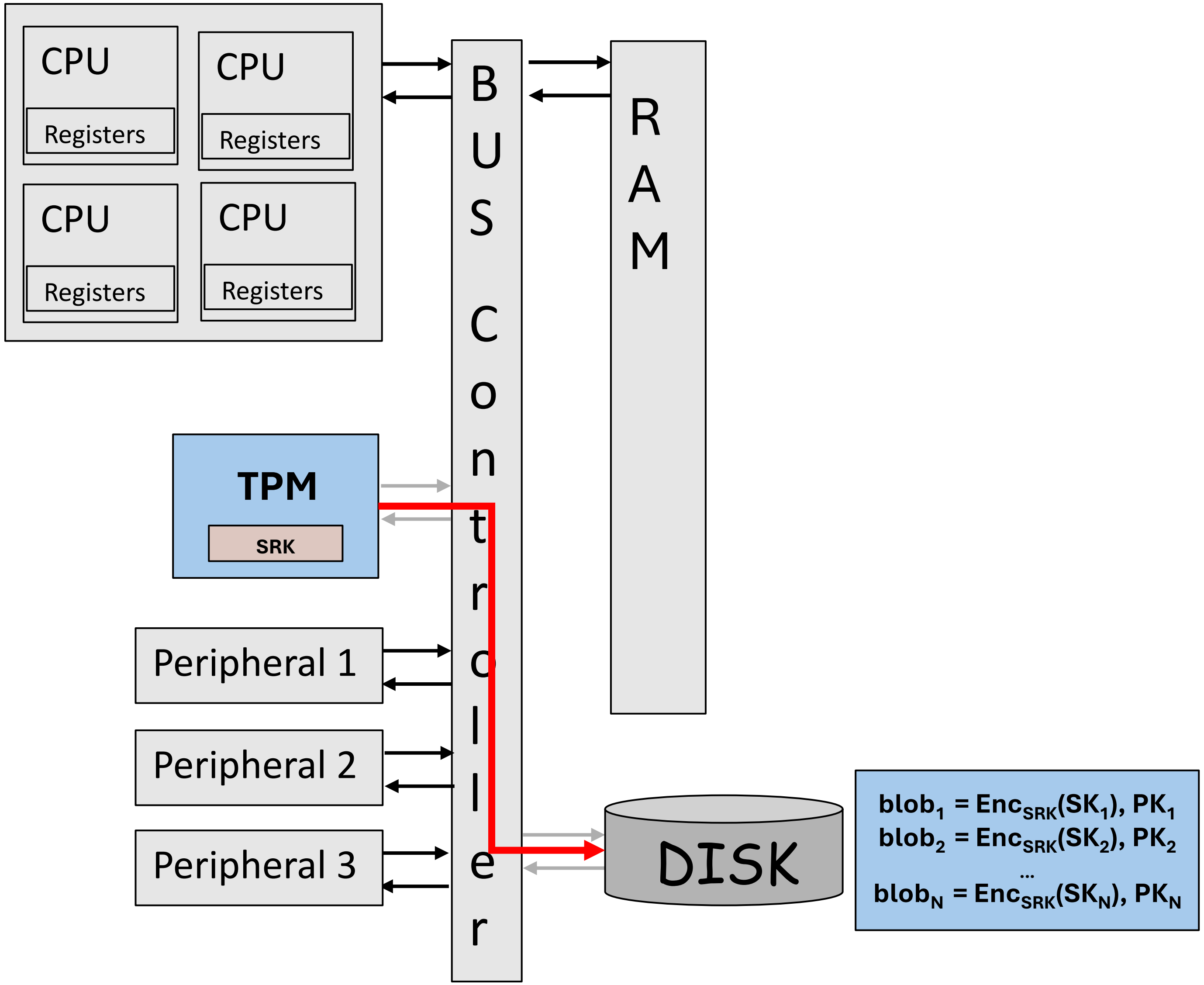
Other new keys are generated by the TPM

- E.g., RSA keys: (PK, SK) pairs
- Stored outside the TPM
- How? → encrypt the private half (SK)

  [$Enc_{SRK}(SK_1)$, PK_1] → $blob_1$
  [$Enc_{SRK}(SK_2)$, PK_2] → $blob_2$
  …
  [$Enc_{SRK}(SK_N)$, PK_N] → $blob_N$

Blobs can be stored anywhere
e.g., disk, another machine, cloud

TPM Architecture



$\text{blob}_1 = \text{Enc}_{\text{SRK}}(\text{SK}_1), \text{PK}_1$

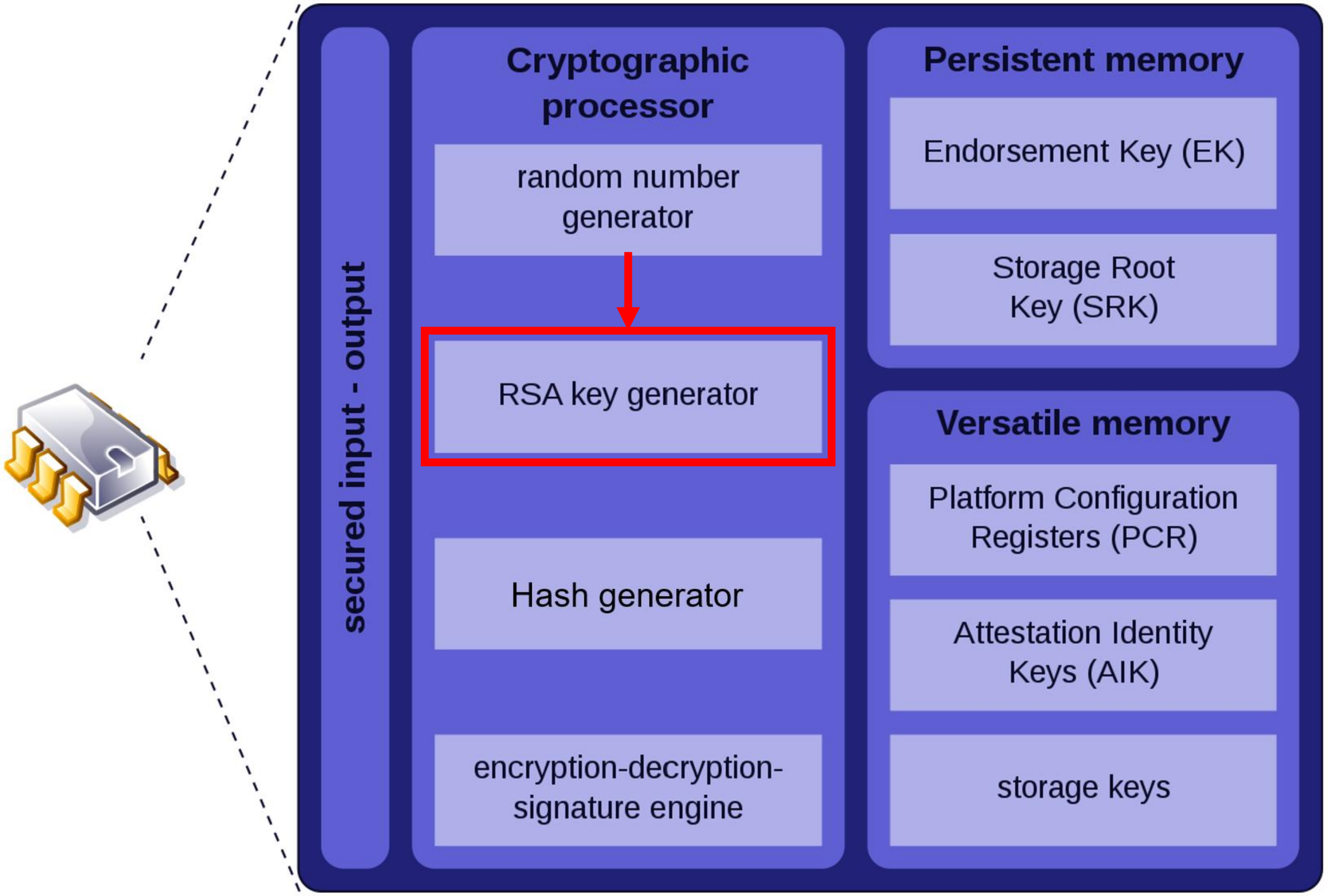
$\text{blob}_2 = \text{Enc}_{\text{SRK}}(\text{SK}_2), \text{PK}_2$

…

$\text{blob}_N = \text{Enc}_{\text{SRK}}(\text{SK}_N), \text{PK}_N$

TPM Architecture

Key Generation



TPM Architecture

Key Generation

Two basic key generation operations:

- **TPM_CreateWrapKey:**
  - (1) Creates a key pair (2) ties it to a system state
  - General purpose
- **TPM_MakeIdentity**
  - Creates an “Attestation Identity” key pair
  - Used for signing

TPM Architecture

Key Generation

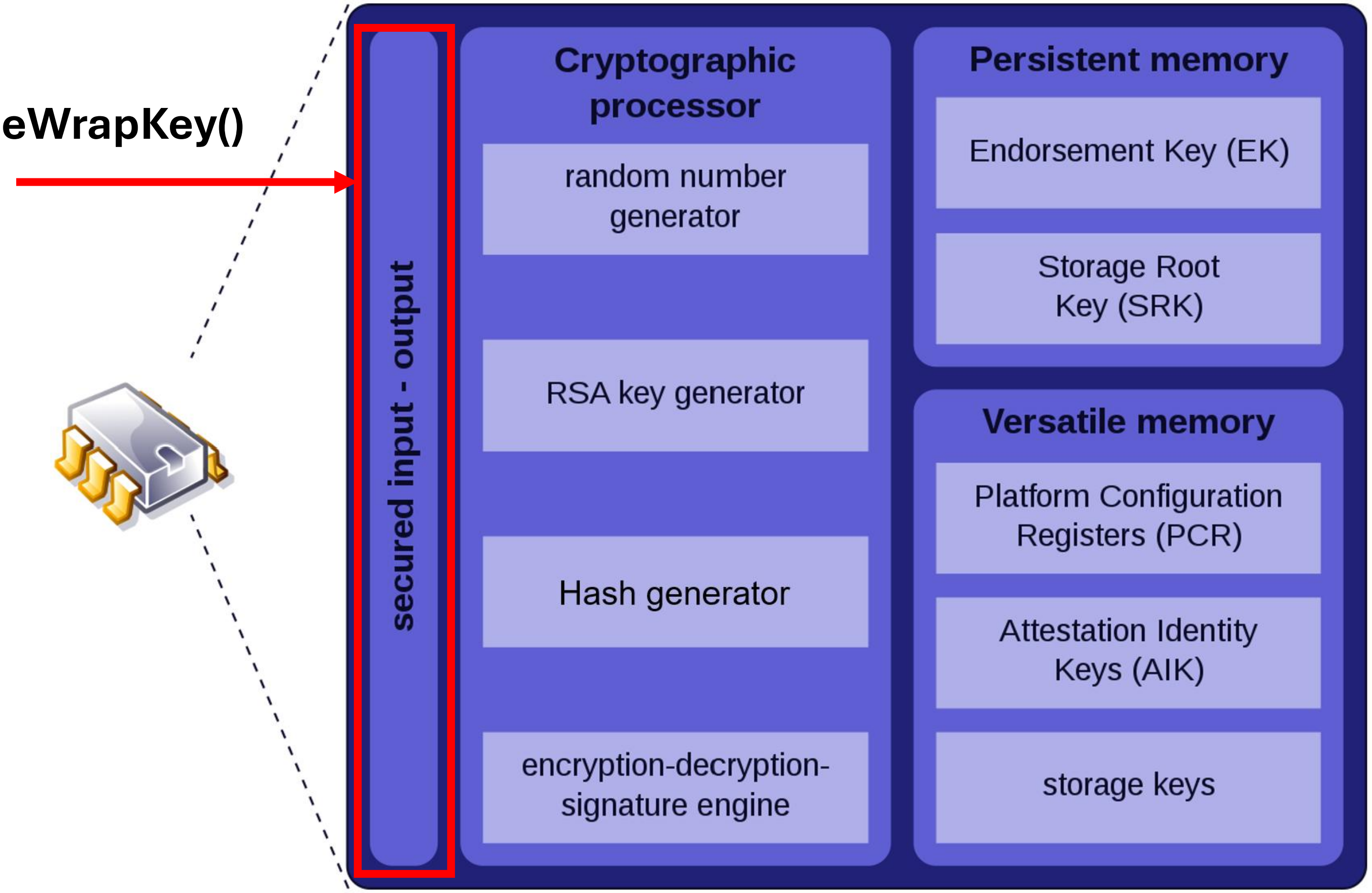
Two basic key generation operations:

- **TPM_CreateWrapKey:**
  - (1) Creates a key pair (2) ties it to a system state
  - General purpose
- **TPM_MakeIdentity**
  - Creates an “Attestation Identity” key pair
  - Used for signing

TPM Architecture

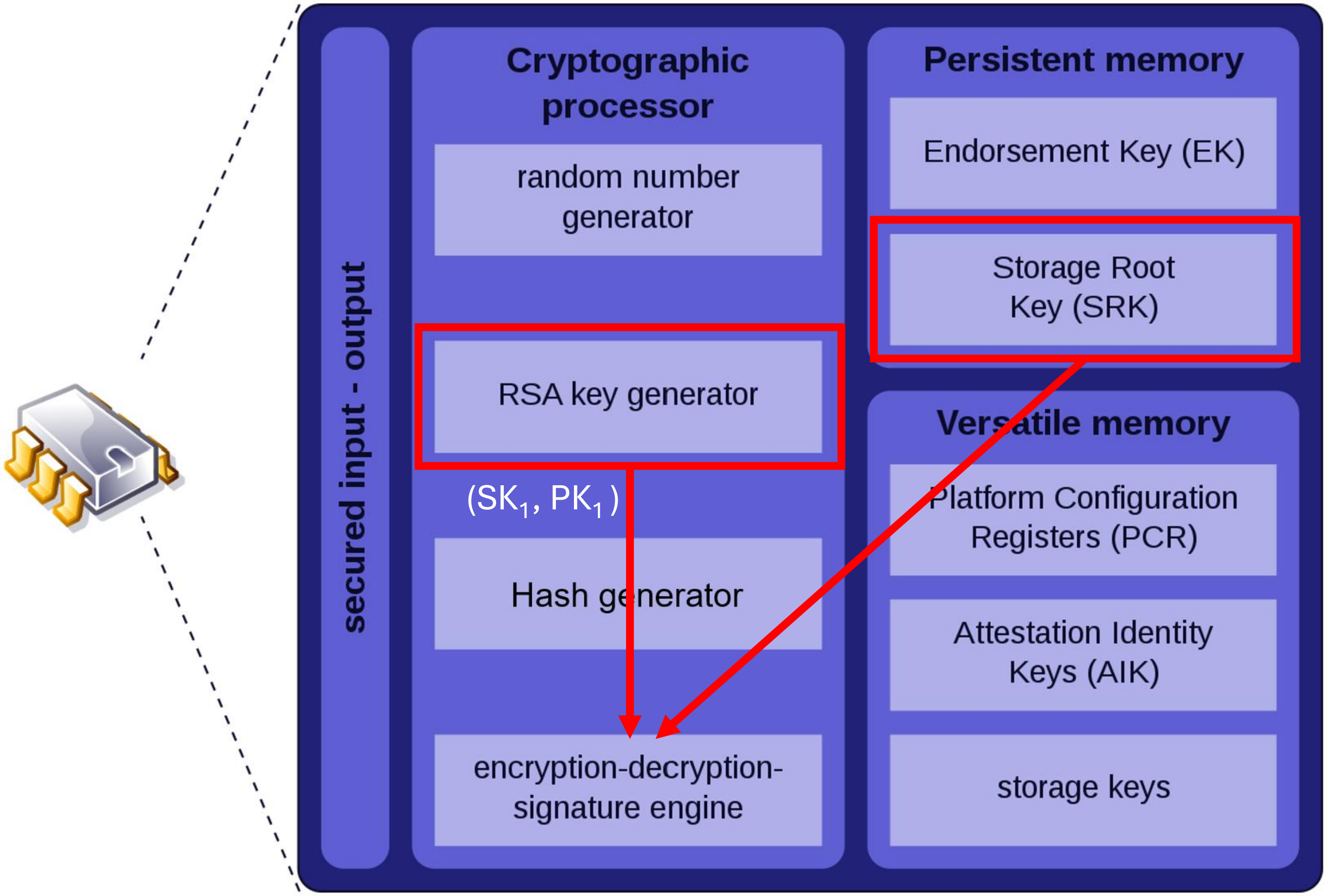
Key Generation

TPM_CreateWrapKey()



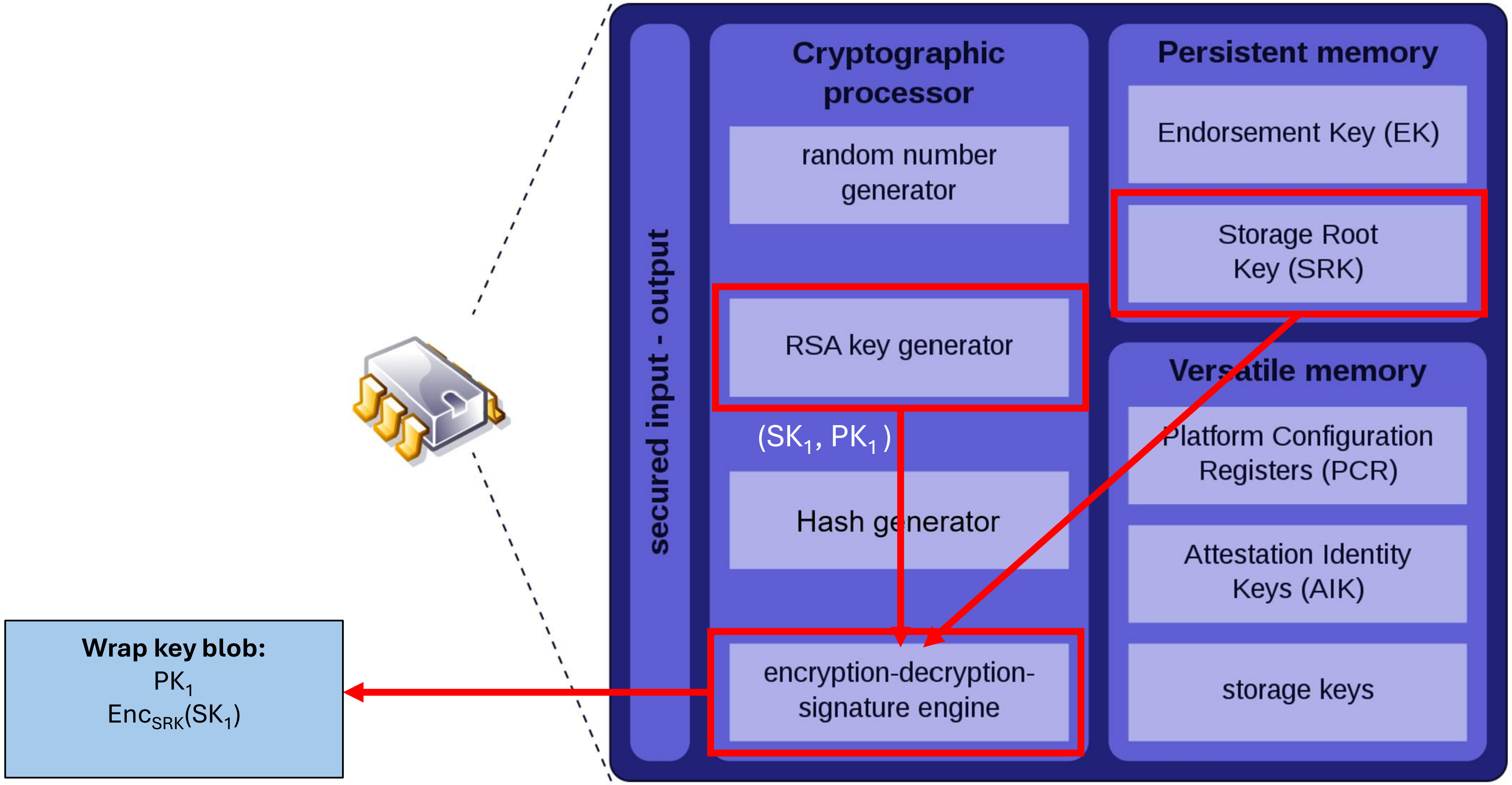
TPM Architecture

Key Generation



TPM Architecture

Key Generation



secured input - output

Cryptographic processor

random number generator

RSA key generator

(SK_1, PK_1)

Hash generator

encryption-decryption-signature engine

Persistent memory

Endorsement Key (EK)

Storage Root Key (SRK)

Versatile memory

Platform Configuration Registers (PCR)

Attestation Identity Keys (AIK)

storage keys

Wrap key blob:

PK_1

$Enc_{SRK}(SK_1)$

TPM Architecture

Key Generation

Two basic key generation operations:

- **TPM_CreateWrapKey:**
  - (1) Creates a key pair (2) ties it to a system state
  - General purpose
- **TPM_MakeIdentity**
  - Creates an “Attestation Identity” key pair
  - Used for signing

Wrap key generation:

Optional authorization parameters

- Require a password to use a key
- Require a system state to use the key
- More coming up…

TPM Architecture

Key Generation

Two basic key generation operations:

- **TPM_CreateWrapKey:**
  - (1) Creates a key pair (2) ties it to a system state
  - General purpose

- **TPM_MakeIdentity**
  - Creates an "Attestation Identity" key pair
  - Used for signing

Wrap key generation:

Optional authorization parameters

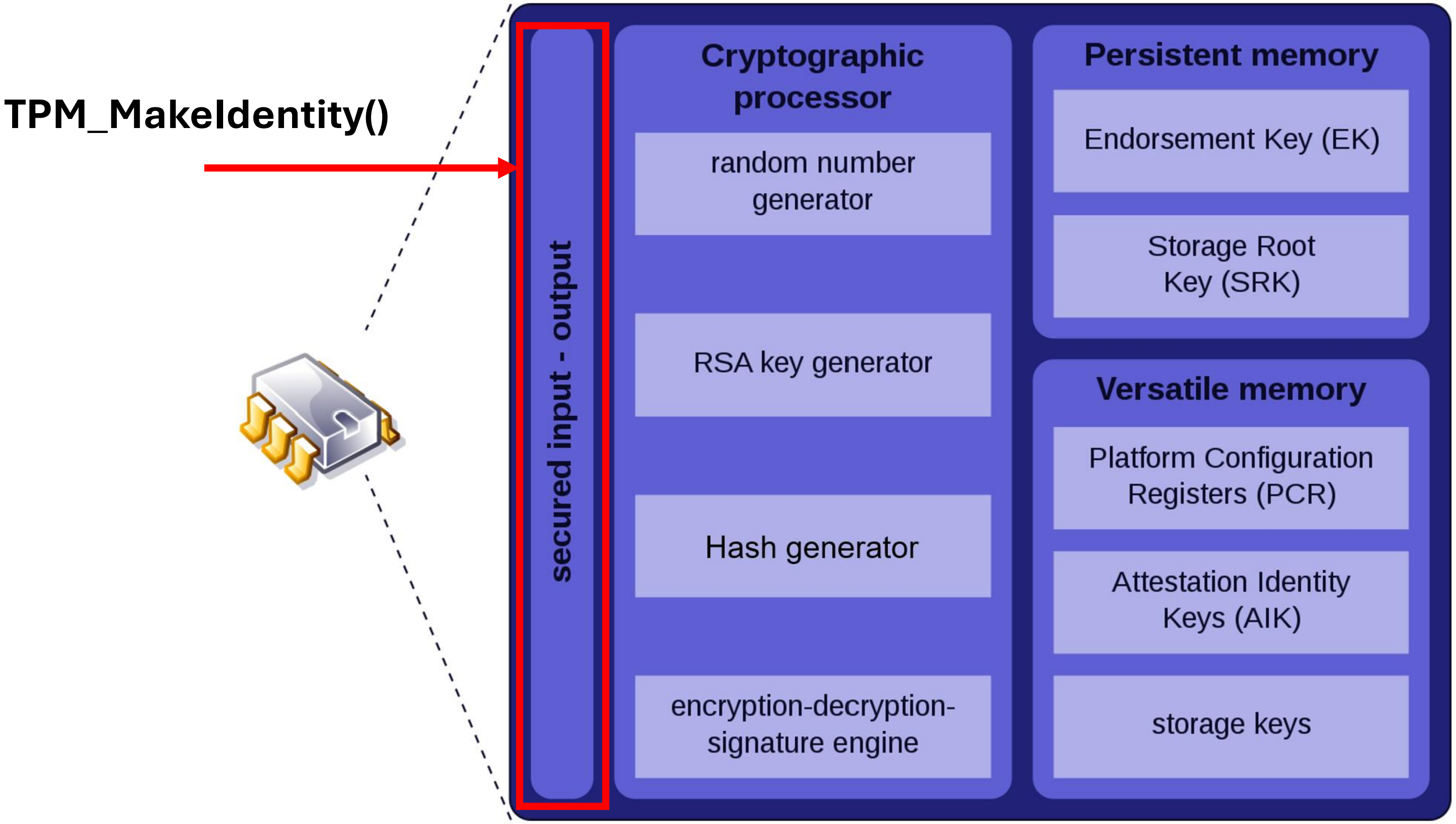
- Require a password to use a key
- Require a system state to use the key
- More coming up…

Similar to wrap keys, but…

- Used for identity
- All new attestation identity key (AIK) pairs are signed
- Signed with the TPM's Endorsement Key
- ***Certification:*** proves PK was issued by the TPM → hence, identity

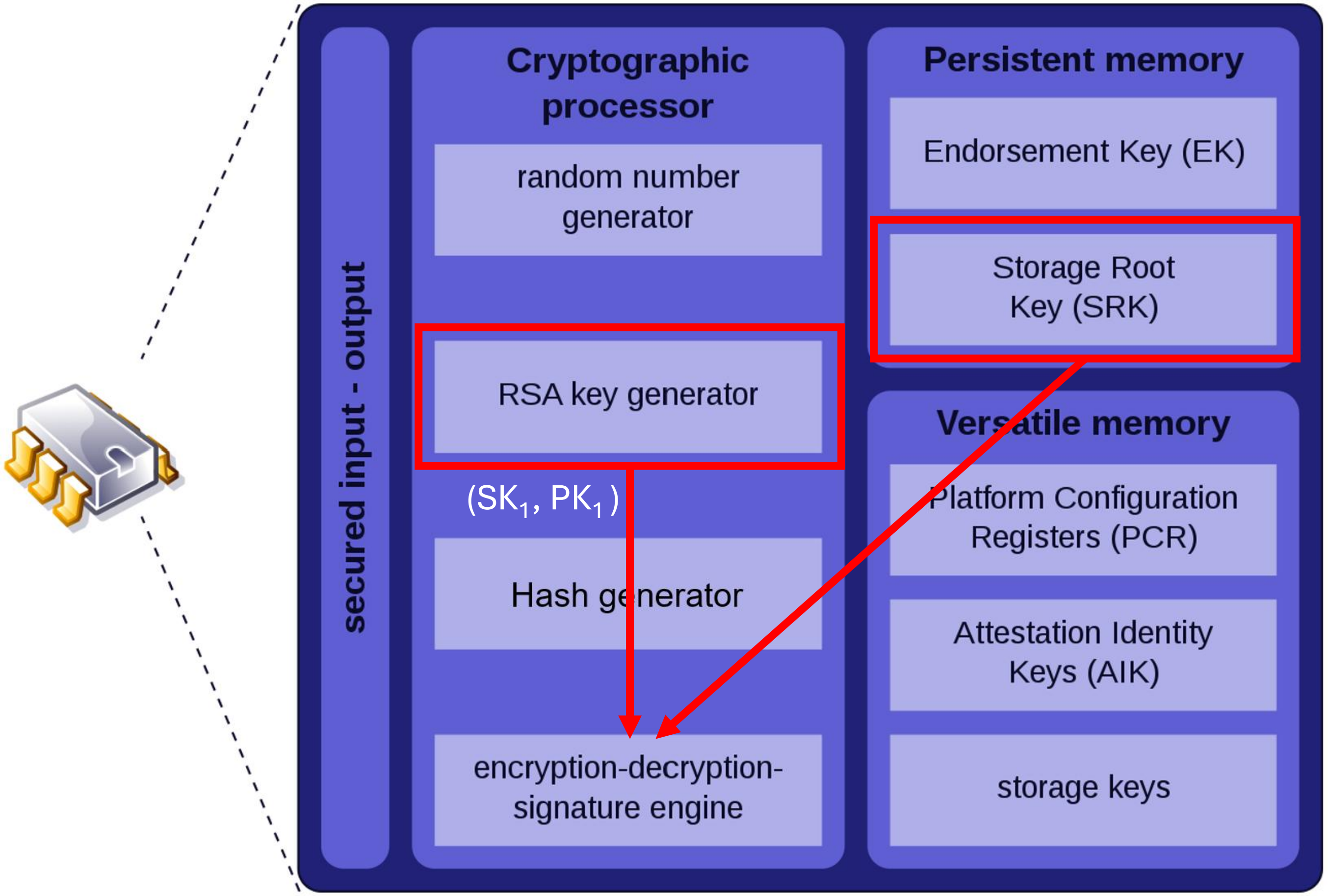
TPM Architecture

Key Generation



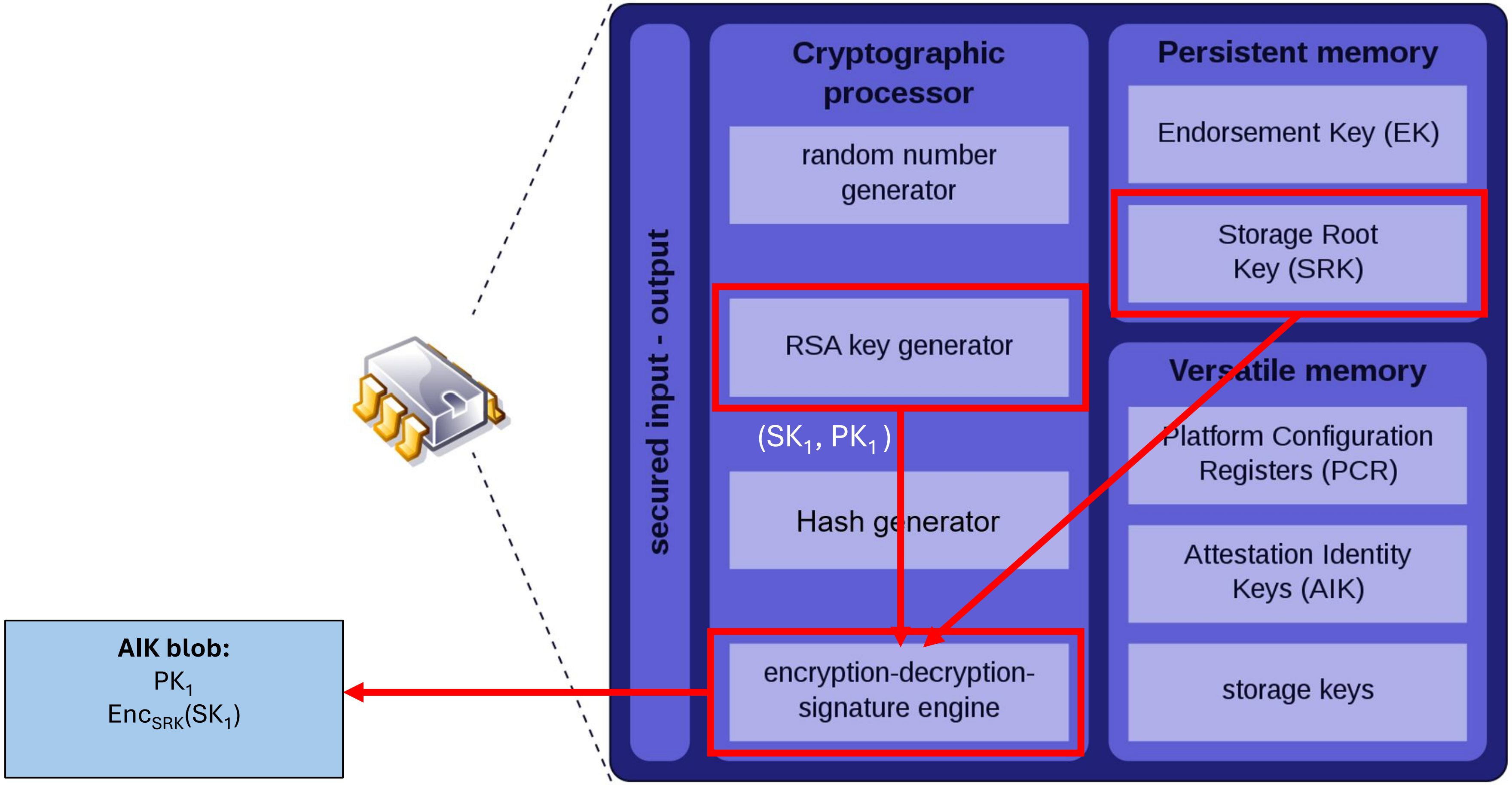
TPM Architecture

Key Generation



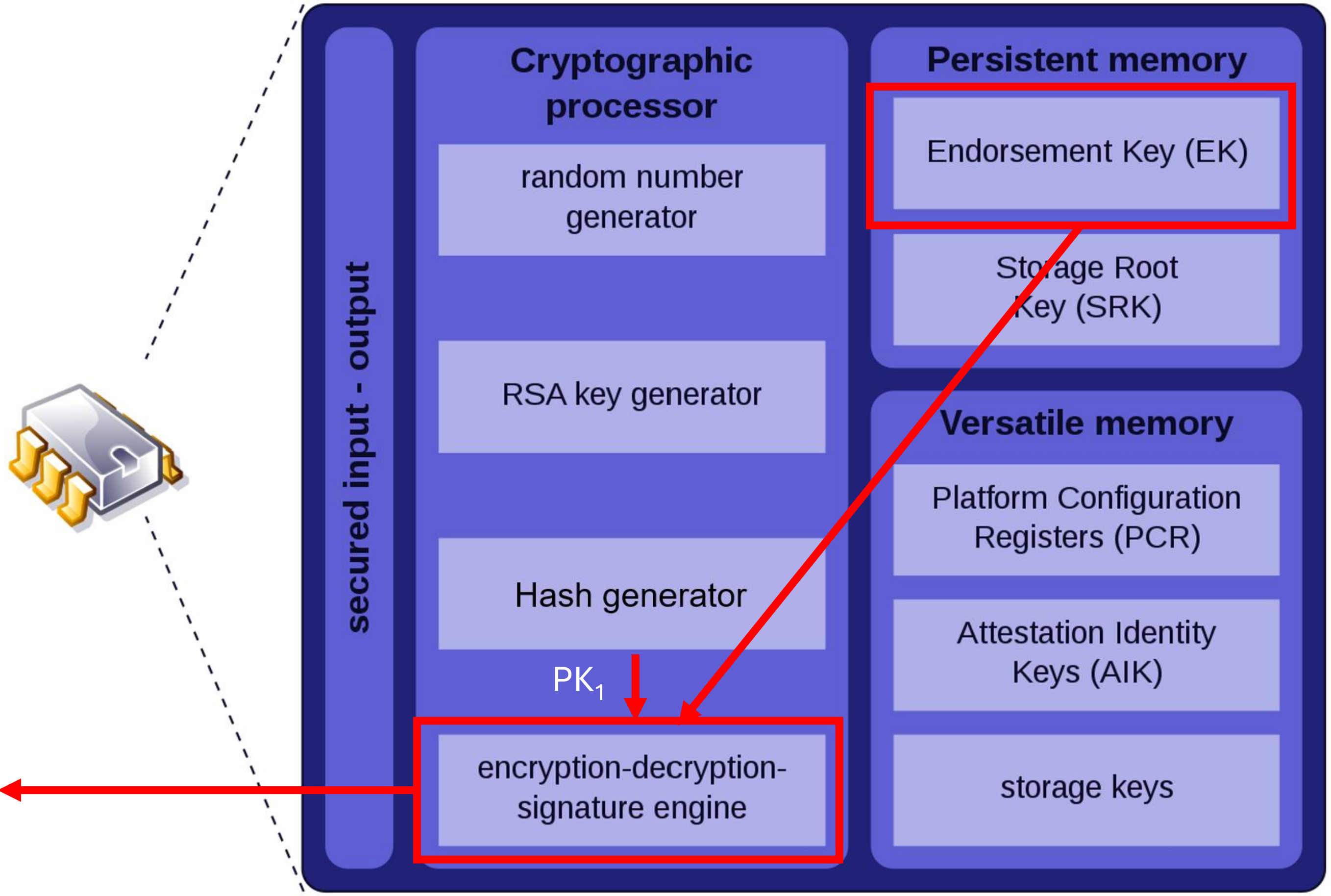
TPM Architecture

Key Generation



TPM Architecture

Key Generation



AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{EK}(PK_1)$

TPM Architecture

Important takeaways…

Storage Root Key (SRK):

1. SRK is securely stored in the TPM permanently
2. Never leaves the TPM
3. Used to encrypt the private half of any new key pair

→ Only the same TPM that generates a key pair can later decrypt it (Secrecy)

TPM Architecture

Important takeaways…

Storage Root Key (SRK):

1. SRK is securely stored in the TPM permanently
2. Never leaves the TPM
3. Used to encrypt the private half of any new key pair

→ Only the same TPM that generates a key pair can later decrypt it (Secrecy)

Endorsement Key (EK):

1. EK is securely stored in the TPM permanently
2. Never leaves the TPM
3. Used to sign the public half of new AIK pairs

→ Anyone can verify that a key pair was generated by a particular TPM (Authentication)

TPM Architecture

TPM Provides:

- A **Root of Trust for Storage**
  - Secure TPM encryption key
- A **Root of Trust for Reporting**
  - Secure TPM signing key (used to establish TPM's identity)
- **TPM State**
  - Limited internal storage
  - Loading & storing keys
  - **Platform Configuration Registers (PCR)**

TPM Architecture

Root of Trust for Reporting

AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{EK}(PK_1)$

- Core question:
  - Is this system in a good state?

- Answer requires:
  - Looking at the system state → a ***Root of Trust for Measurement***
  - Proving authenticity of the state → a ***Root of Trust for Reporting***

- What is TPM?

A Root of Trust for Reporting (RTR)

NOT A Root of Trust for Measurement (RTM)

- Recall: TPM is *passive* → responds to requests, does not proactively check anything

TPM Architecture

Root of Trust for Reporting

AIK blob:

PK_1

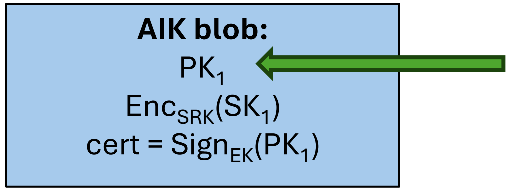
$Enc_{SRK}(SK_1)$

cert = $Sign_{EK}(PK_1)$

How to know that a report/signature was issued by a trusted TPM?

TPM Architecture

Root of Trust for Reporting



AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{EK}(PK_1)$

How to know that a report/signature was issued by a trusted TPM?

It must come with a signature that can be verified using an AIK public key

TPM Architecture

Root of Trust for Reporting

AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{EK}(PK_1)$

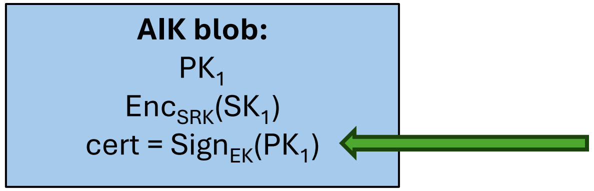
How to know that a report/signature was issued by a trusted TPM?

It must come with a signature that can be verified using an AIK public key

How to know that this public key was indeed generated by a trusted TPM?

TPM Architecture

Root of Trust for Reporting



AIK blob:

PK_1

$Enc_{SRK}(SK_1)$

cert = $Sign_{EK}(PK_1)$

How to know that a report/signature was issued by a trusted TPM?

It must come with a signature that can be verified using an AIK public key

How to know that this public key was indeed generated by a trusted TPM?

*Verify the **cert:** a proof signed using the TPM's endorsement key (EK)*

TPM Architecture

Root of Trust for Reporting

AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
$cert = Sign_{EK}(PK_1)$

How to know that a report/signature was issued by a trusted TPM?

It must come with a signature that can be verified using an AIK public key

How to know that this public key was indeed generated by a trusted TPM?

*Verify the **cert:** a proof signed using the TPM's endorsement key (EK)*

How to know which public key should be used to verify the **cert**?

TPM Architecture

Root of Trust for Reporting

AIK blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{EK}(PK_1)$

How to know that a report/signature was issued by a trusted TPM?

It must come with a signature that can be verified using an AIK public key

How to know that this public key was indeed generated by a trusted TPM?

*Verify the **cert:** a proof signed using the TPM's endorsement key (EK)*

How to know which public key should be used to verify the **cert**?

Verify the TPM endorsement key certificate

TPM Architecture

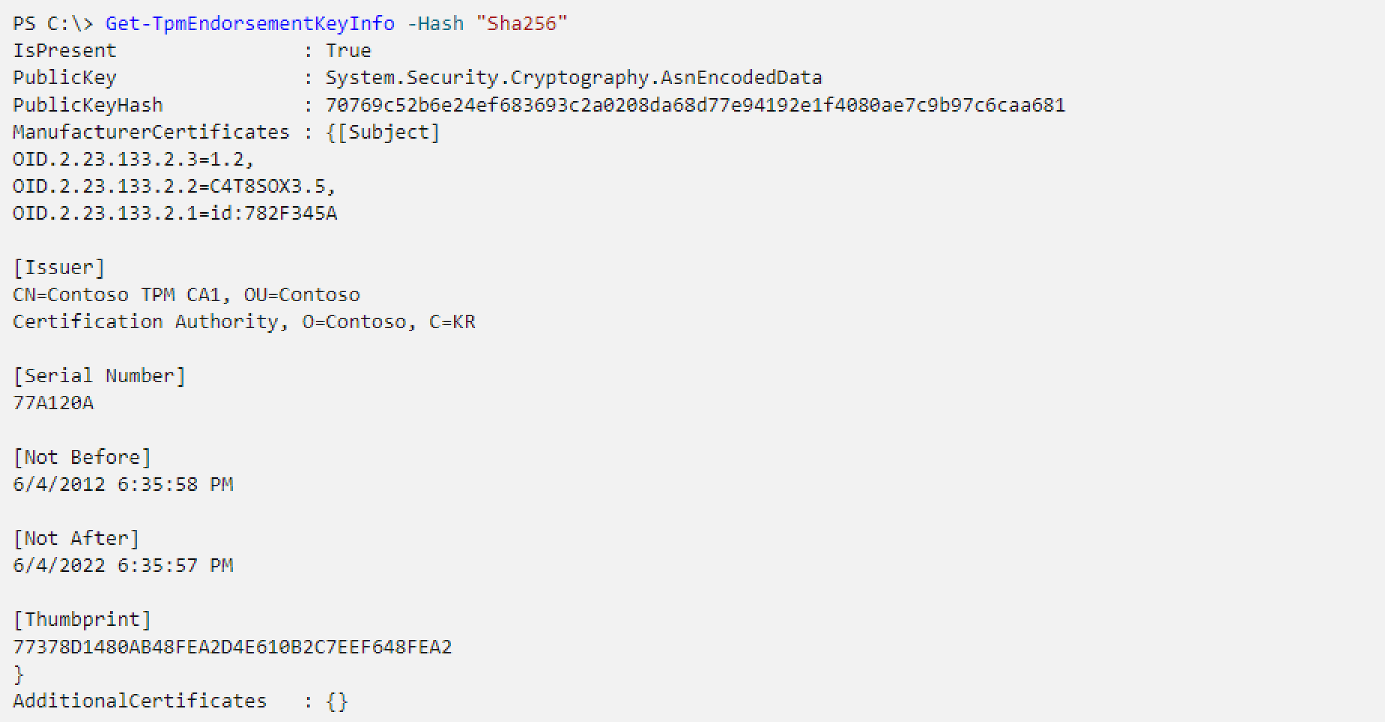
EK Certificate

Modern TPMs store their own certificate metadata and public key for convenience

Public part of the EK can be retrieved with a command

Private part of the EK can never be retrieved

Available certificate metadata can also be retrieved



```
PS C:\> Get-TpmEndorsementKeyInfo -Hash "Sha256"
IsPresent                 : True
PublicKey                 : System.Security.Cryptography.AsnEncodedData
PublicKeyHash             : 70769c52b6e24ef683693c2a0208da68d77e94192e1f4080ae7c9b97c6caa681
ManufacturerCertificates : {[Subject]
OID.2.23.133.2.3=1.2,
OID.2.23.133.2.2=C4T8SOX3.5,
OID.2.23.133.2.1=id:782F345A

[Issuer]
CN=Contoso TPM CA1, OU=Contoso
Certification Authority, O=Contoso, C=KR

[Serial Number]
77A120A

[Not Before]
6/4/2012 6:35:58 PM

[Not After]
6/4/2022 6:35:57 PM

[Thumbprint]
77378D1480AB48FEA2D4E610B2C7EEF648FEA2
}
AdditionalCertificates   : {}
```

TPM Architecture

TPM Provides:

- A **Root of Trust for Storage**
  - Secure TPM encryption key
- A **Root of Trust for Reporting**
  - Secure TPM signing key (used to establish TPM's identity)
- **TPM State**
  - Limited internal storage
  - Loading & storing keys
  - **Platform Configuration Registers (PCR)**

TPM Architecture

How to use generate keys?

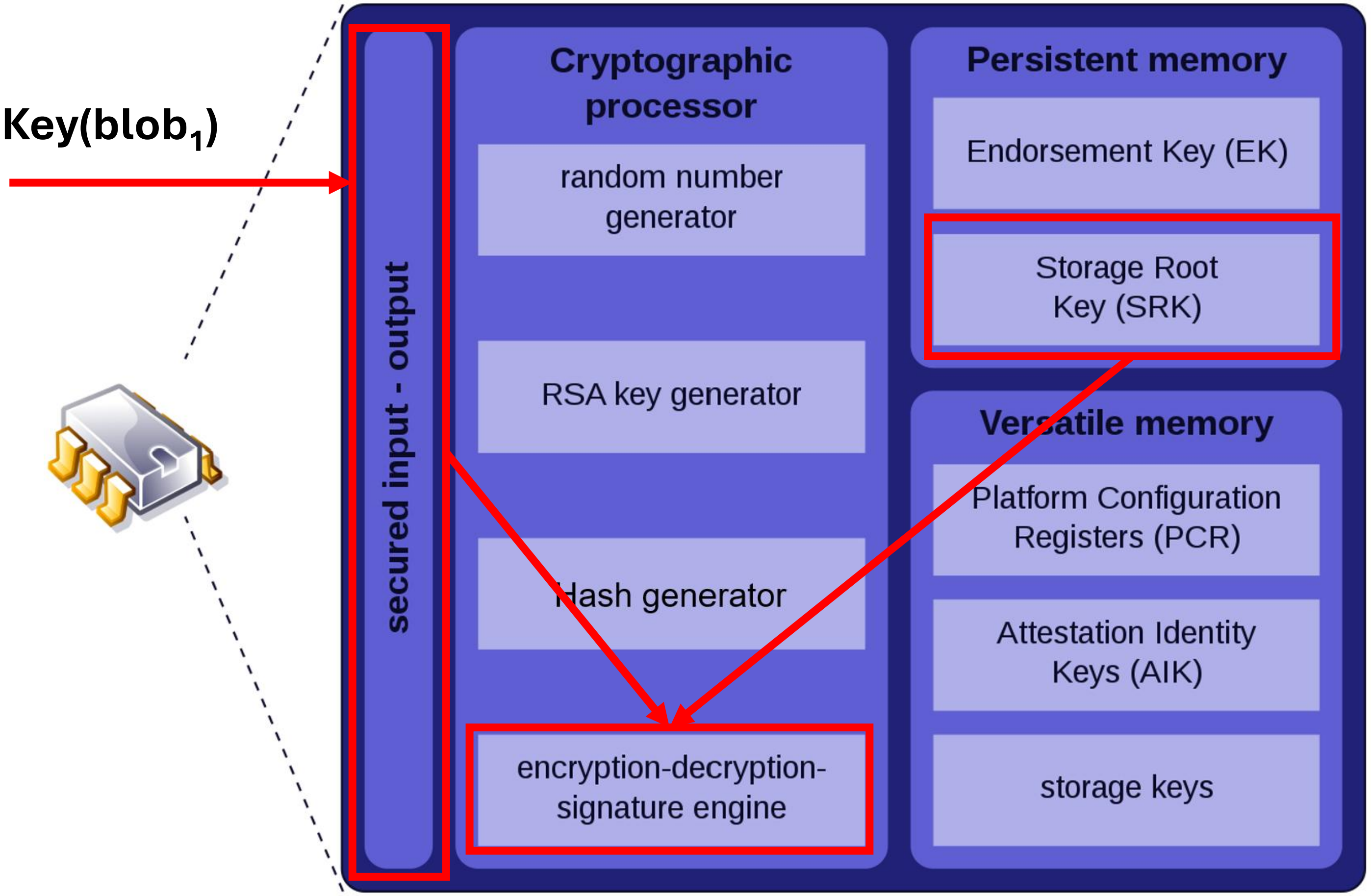
TPM_LoadKey

- Input – a key blob
- Loads a key blob into the TPM
- Internally → decrypts the private half using the parent key (e.g., the SRK)
- Stores the decrypted private half in TPM's versatile memory
- Returns a *key handle* → identifier for the loaded key

TPM Architecture

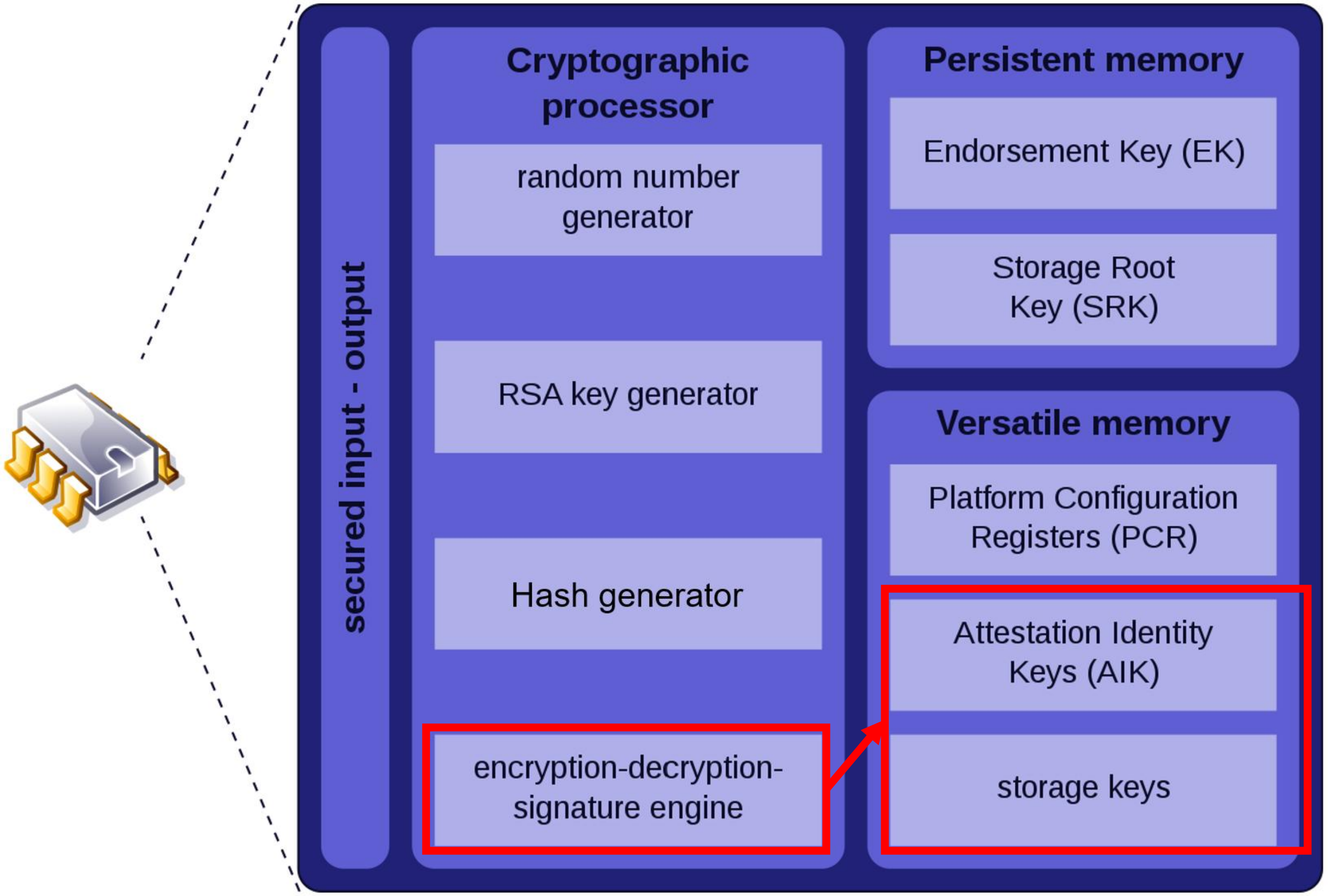
Key Generation

TPM_LoadKey(blob$_1$)



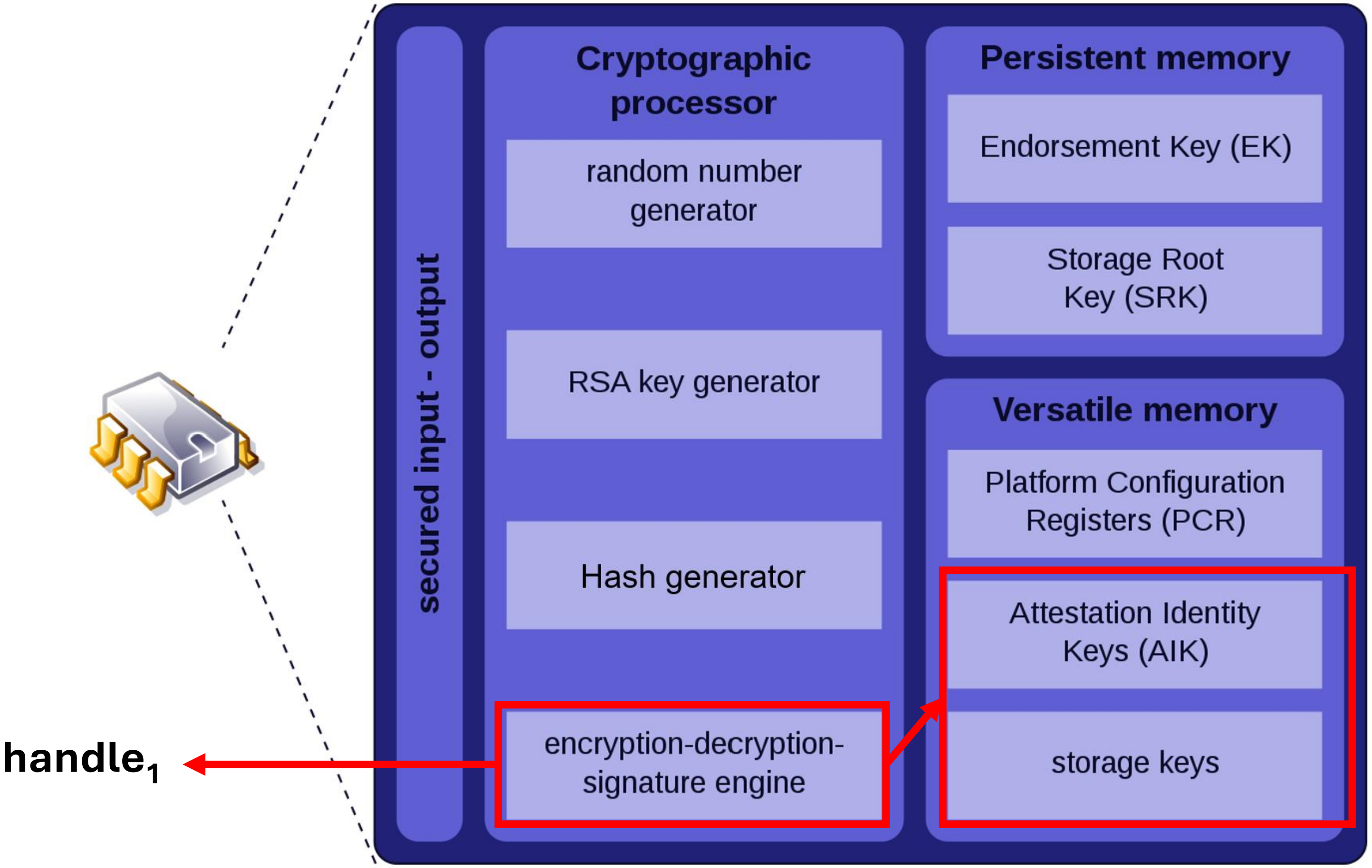
TPM Architecture

Key Generation



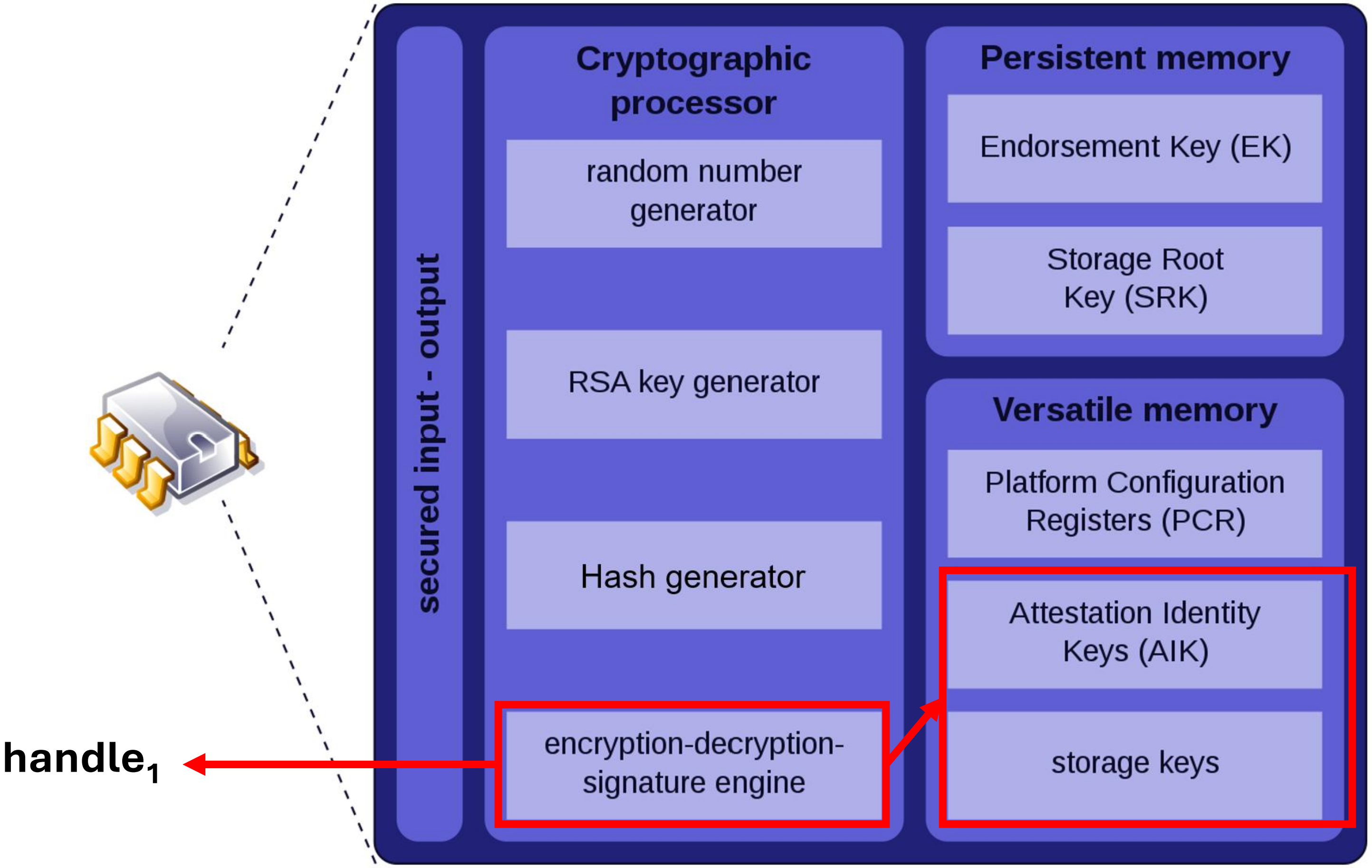
TPM Architecture

Key Generation



TPM Architecture

Key Generation



Once key is loaded, the handle can be given as input to other TPM commands
<u>Handle allows external operations without ever directly seeing keys</u>

TPM Architecture

Using loaded keys

- Once a key is loaded, TPM can perform typical cryptographic operations like a black-box

- One very important feature makes these operations special…

TPM Architecture

Using loaded keys

- Once a key is loaded, TPM can perform typical cryptographic operations like a black-box

- One very important feature makes these operations special…

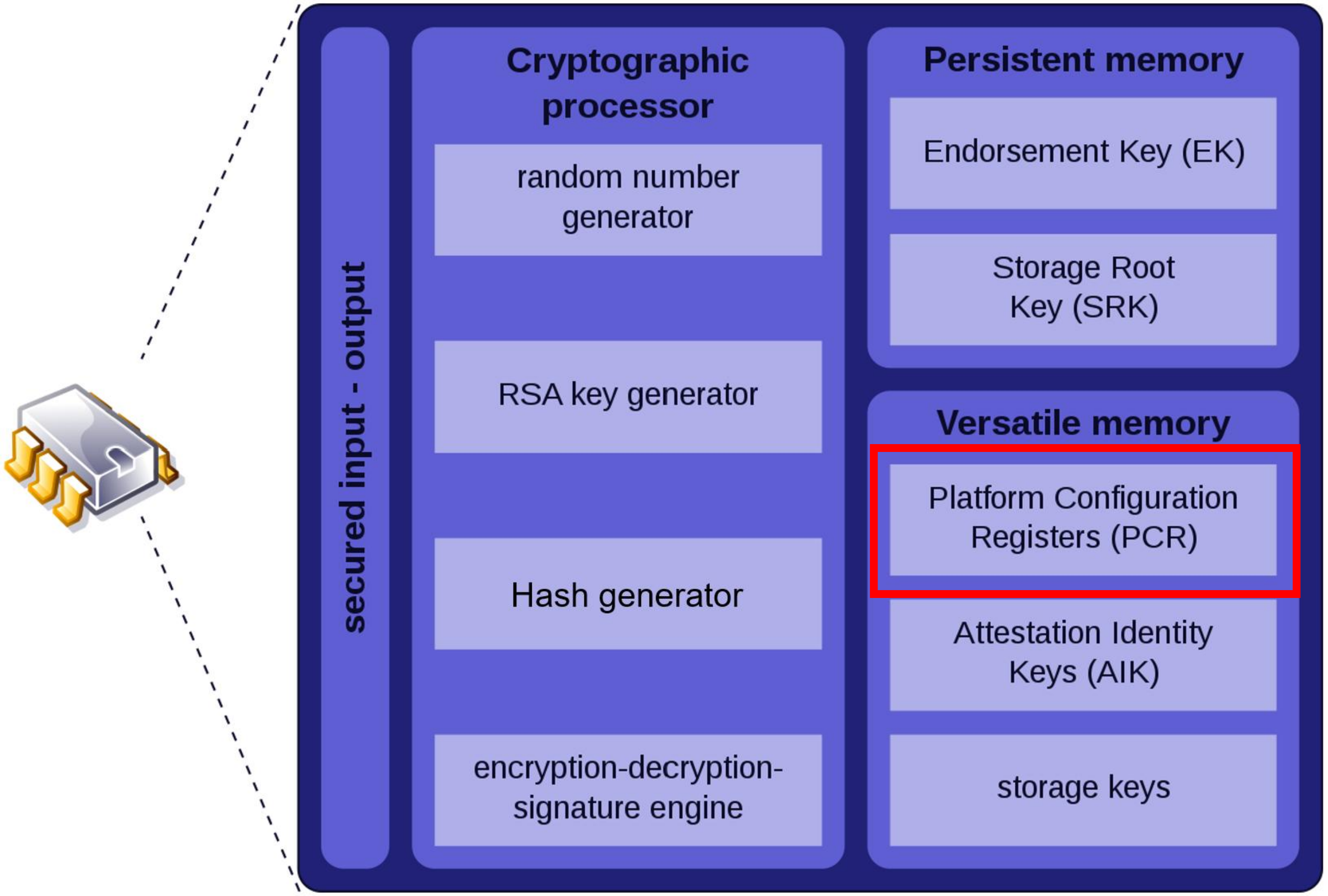
- **There usage can be conditioned to the current system state**

- How to record state?

Platform Configuration Registers

TPM Architecture

Key Generation



TPM Architecture

Platform Configuration Registers (PCRs)

Implement an append-only secure state chain

- PCR Size: size of TPM Hash Algorithm

- Modern TPMs have 24 PCRs → old ones have 16
  - Labeled: PCR-0, PCR-1, ..., PCR-23

- Typically used to store system states (though other uses are possible)

TPM Architecture

Platform Configuration Registers (PCRs)

Key Features:

- Always reset to a default value at boot (e.g., zero)
- Can never be freely overwritten
- Highly-constrained & well-defined behavior:

Only modifiable using ***extend*** operation:

Extend(PCR-id, <input>)

TPM Architecture

Platform Configuration Registers (PCRs)

extend(PCR-id, <input>)

- PCR-id = H(PCR-id || <input>)
- With TPM’s hash function H

TPM Architecture

Platform Configuration Registers (PCRs)

extend(PCR-id, <input>)

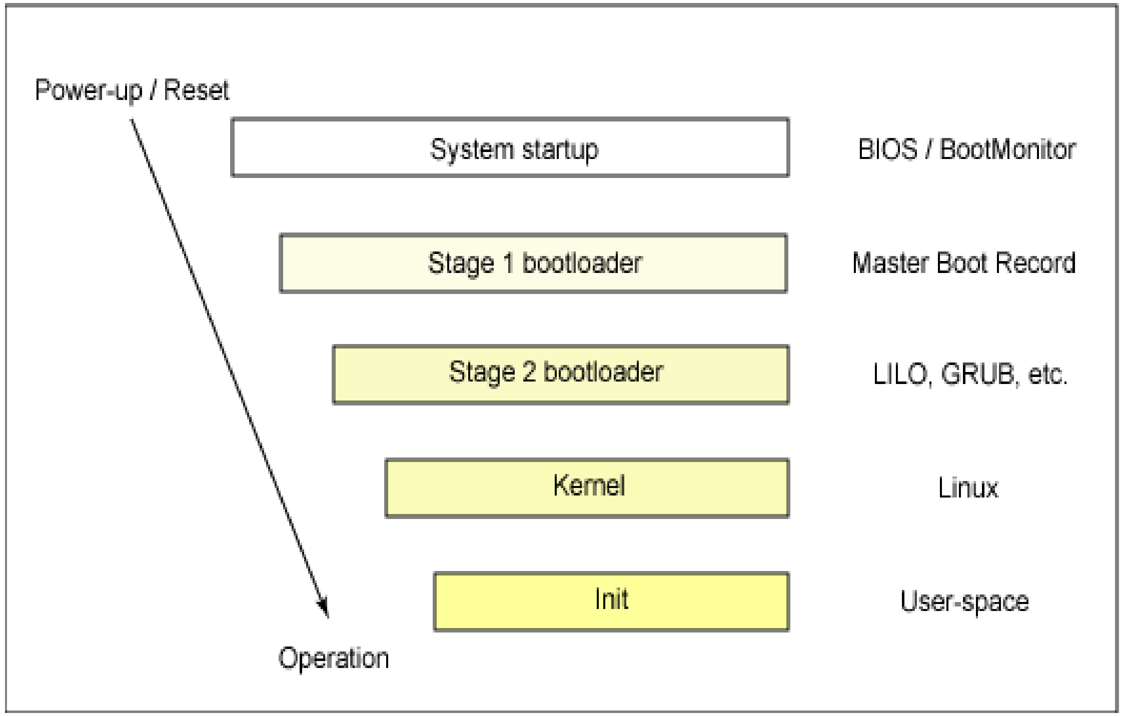
- PCR-id = H(PCR-id || <input>)
- With TPM’s hash function H

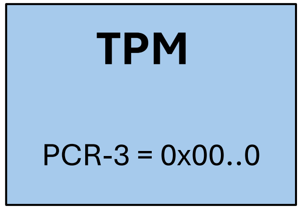
Example:

Boot (power on):	PCR-0 = 0x00…0
extend(PCR-0,”adam”)	PCR-0 = H(0x00…0 | | “adam”) = **0xF3…7**
extend(PCR-0,”cs453”)	PCR-0 = H(0xF3…7 | | “cs453”) = **0xAE…2**
extend(PCR-0,”TPM”)	PCR-0 = H(0xAE…2 | | “TPM”) = **0xD4…C**

TPM Architecture

Measuring Boot State into PCRs

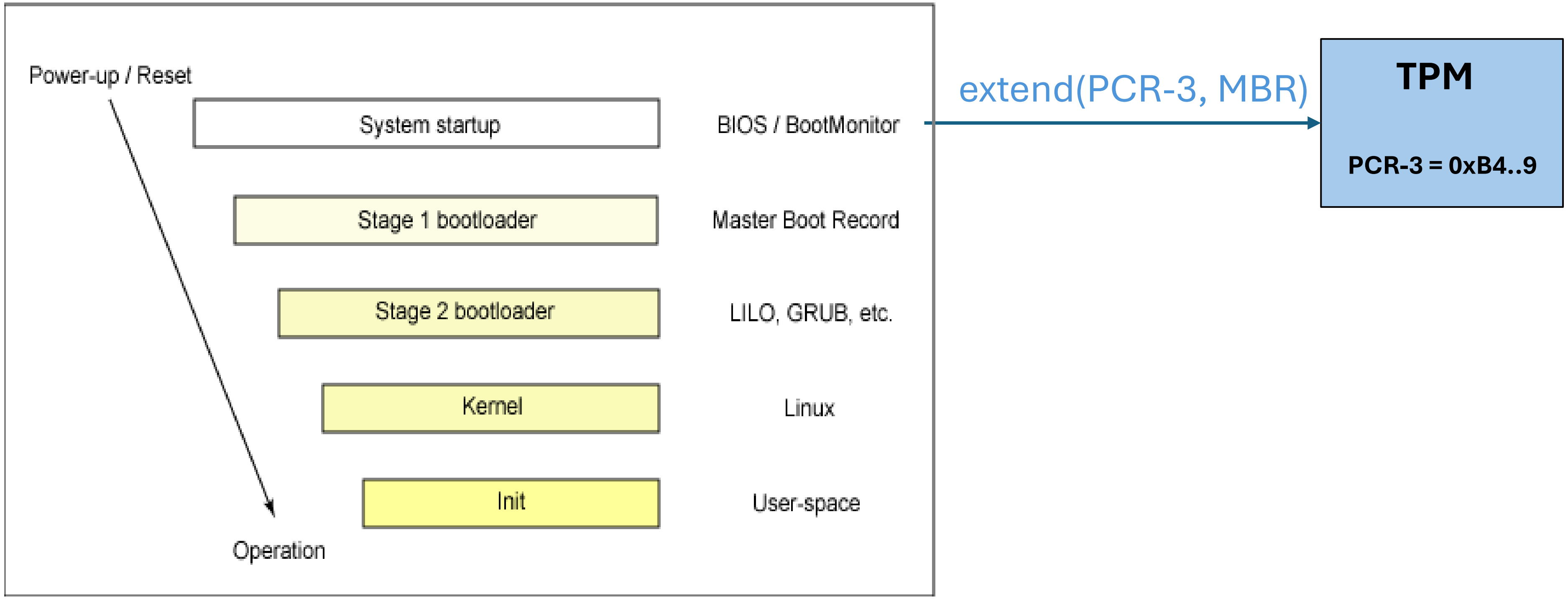




TPM Architecture

Measuring Boot State into PCRs

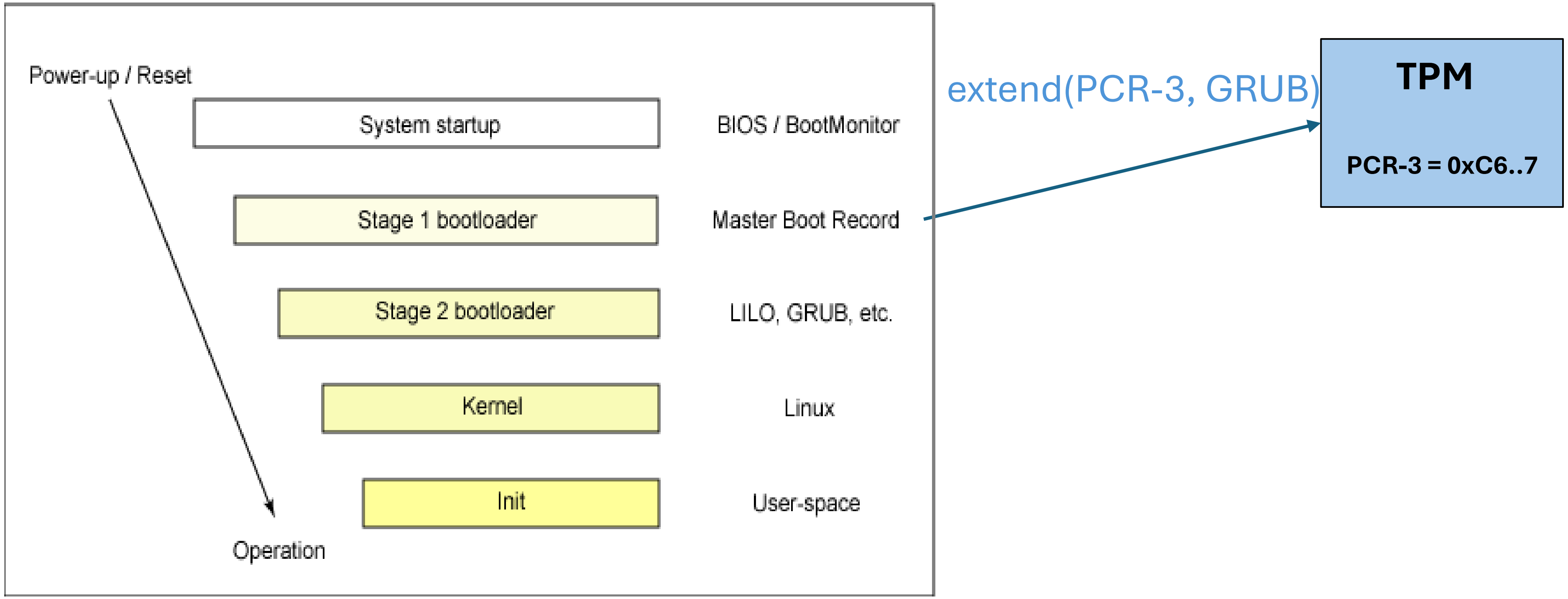
Before loading next module, extend it into PCR



TPM Architecture

Measuring Boot State into PCRs

Before loading next module, extend it into PCR



Power-up / Reset

System startup — BIOS / BootMonitor

Stage 1 bootloader — Master Boot Record

Stage 2 bootloader — LILO, GRUB, etc.

Kernel — Linux

Init — User-space

Operation

extend(PCR-3, GRUB)

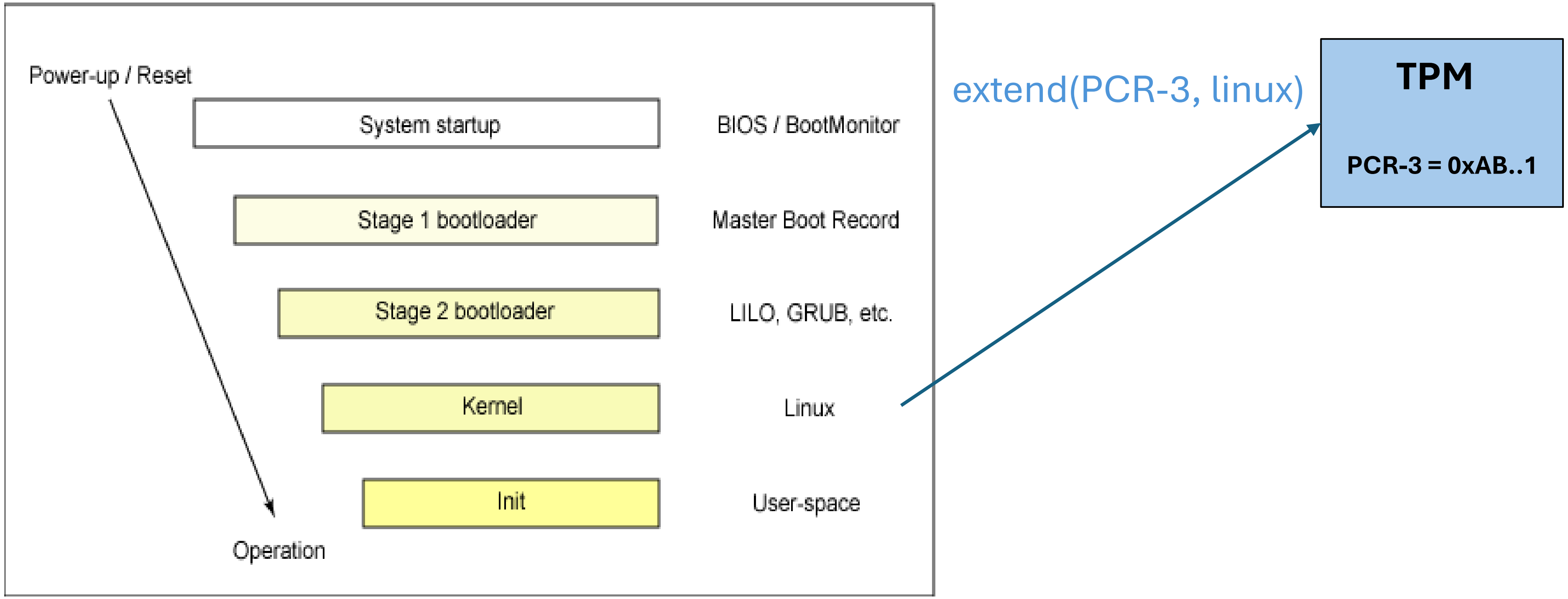
TPM

PCR-3 = 0xC6..7

TPM Architecture

Measuring Boot State into PCRs

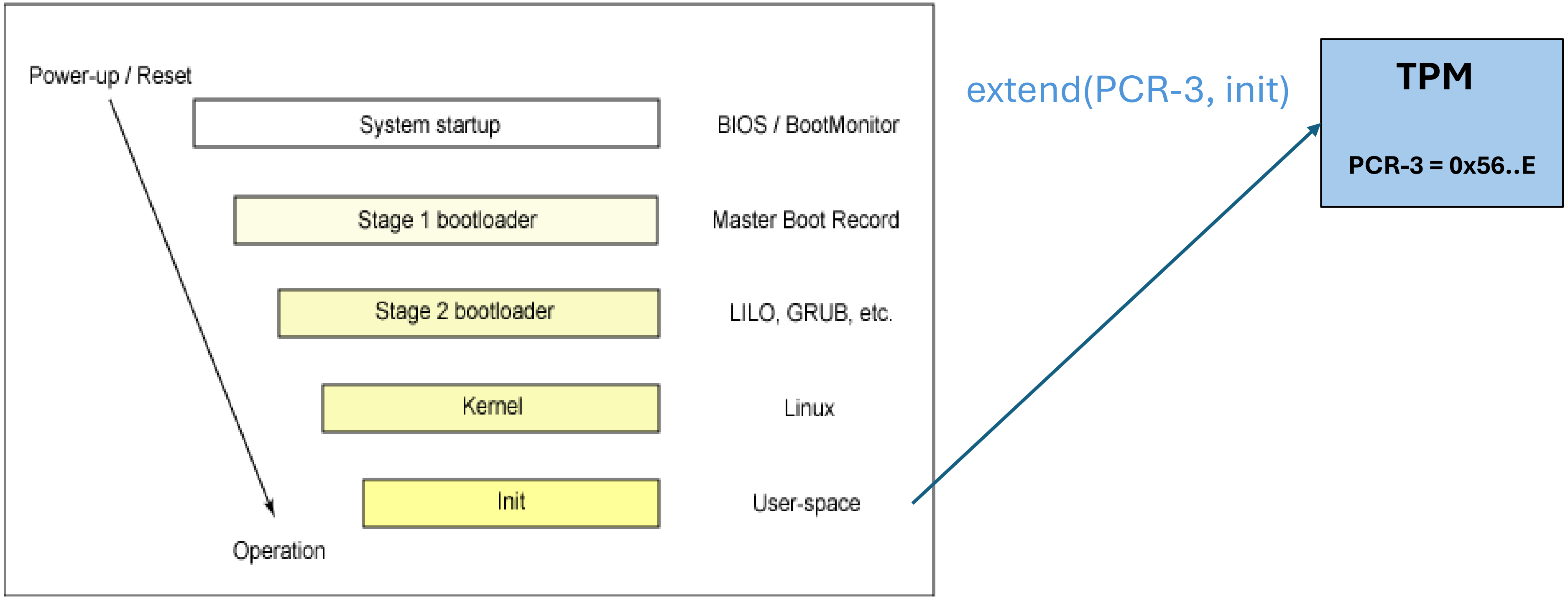
Before loading next module, extend it into PCR



TPM Architecture

Measuring Boot State into PCRs

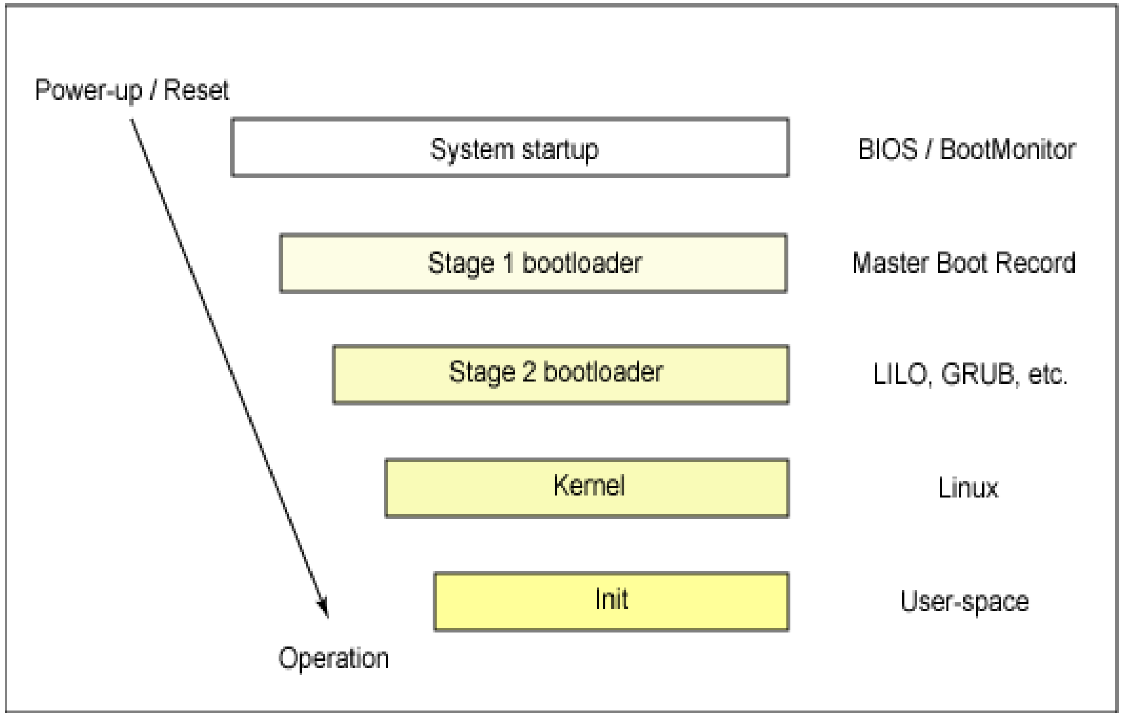
Before loading next module, extend it into PCR



TPM Architecture

Measuring Boot State into PCRs

Before loading next module, extend it into PCR



extend(PCR-3, “done”)

TPM

PCR-3 = 0x8D..C

If anything different is loaded, the final PCR-3 value will be different than expected (0x8D..C)

TPM-based Remote Attestation:

How can this be used for Remote Attestation?

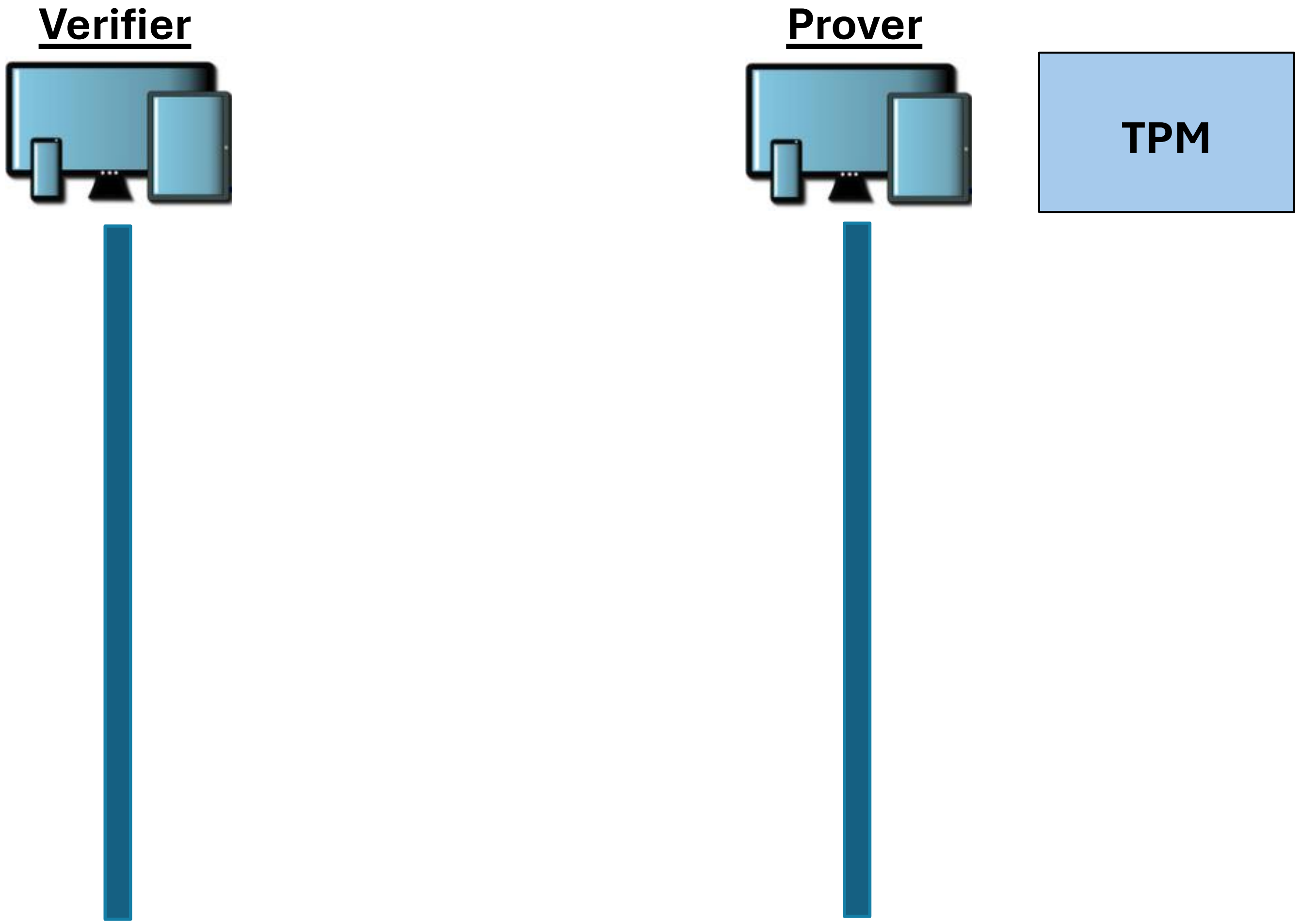
1. Provide a “quote” of challenge || PCR-of-interest

2. Signing the challenge with PCR-bound key

TPM-based RA (v1)

TPM Quote: quote(nonce, PCRs (selection), AIK_handle)

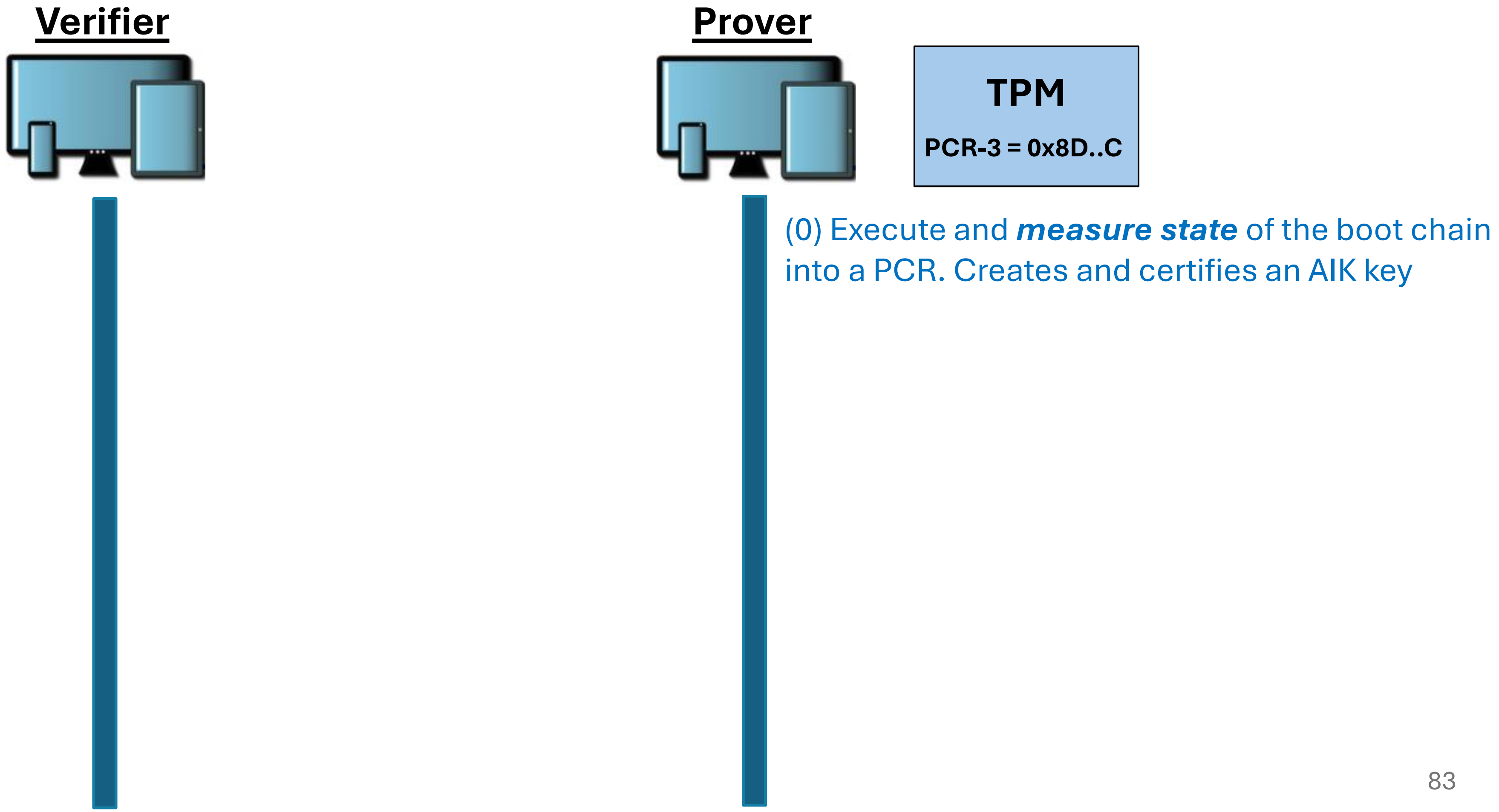
- TPM uses AIK to sign selected PCRs and a nonce → returns a signature
- Nonce externally provided input (e.g., RA challenge)



TPM-based RA (v1)

TPM Quote: quote(nonce, PCRs (selection), AIK_handle)

- TPM uses AIK to sign selected PCRs and a nonce → returns a signature
- Nonce externally provided input (e.g., RA challenge)



Verifier

Prover

TPM

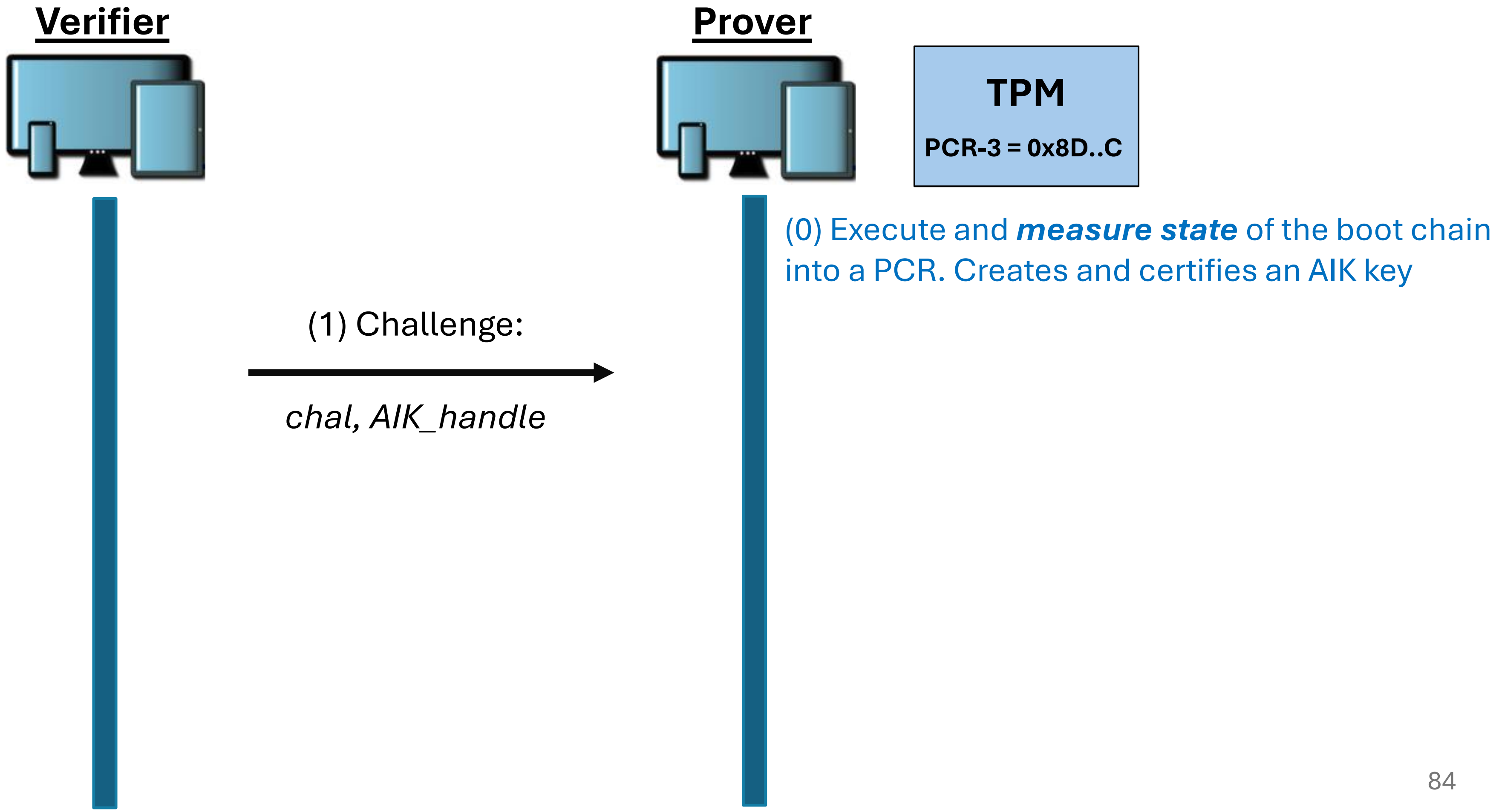
PCR-3 = 0x8D..C

(0) Execute and ***measure state*** of the boot chain into a PCR. Creates and certifies an AIK key

TPM-based RA (v1)

TPM Quote: quote(nonce, PCRs (selection), AIK_handle)

- TPM uses AIK to sign selected PCRs and a nonce → returns a signature
- Nonce externally provided input (e.g., RA challenge)



(0) Execute and ***measure state*** of the boot chain into a PCR. Creates and certifies an AIK key

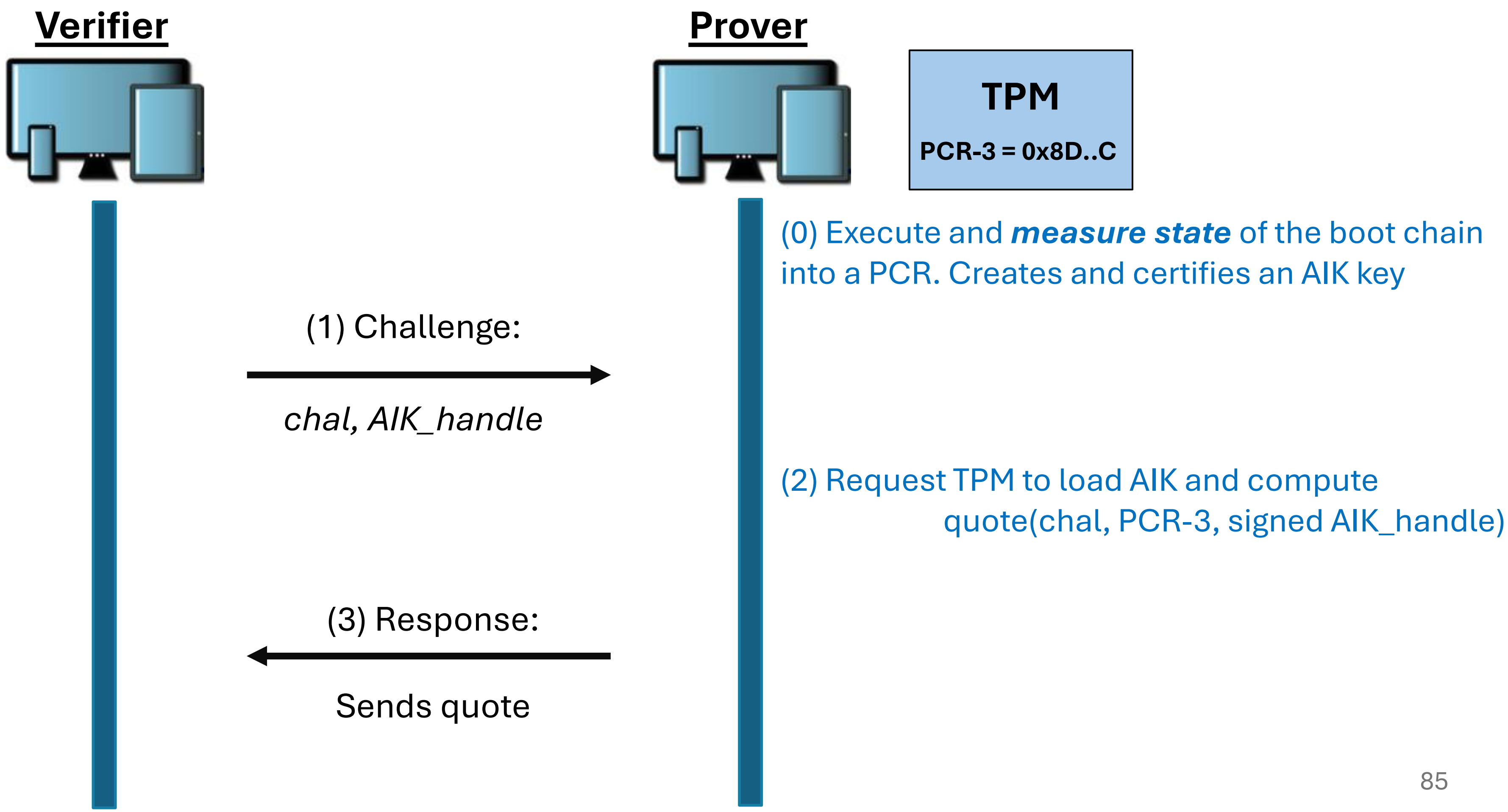
(1) Challenge:

chal, AIK_handle

TPM-based RA (v1)

TPM Quote: quote(nonce, PCRs (selection), AIK_handle)

- TPM uses AIK to sign selected PCRs and a nonce → returns a signature
- Nonce externally provided input (e.g., RA challenge)



Verifier

Prover

TPM

PCR-3 = 0x8D..C

(0) Execute and ***measure state*** of the boot chain into a PCR. Creates and certifies an AIK key

(1) Challenge:

chal, AIK_handle

(2) Request TPM to load AIK and compute quote(chal, PCR-3, signed AIK_handle)

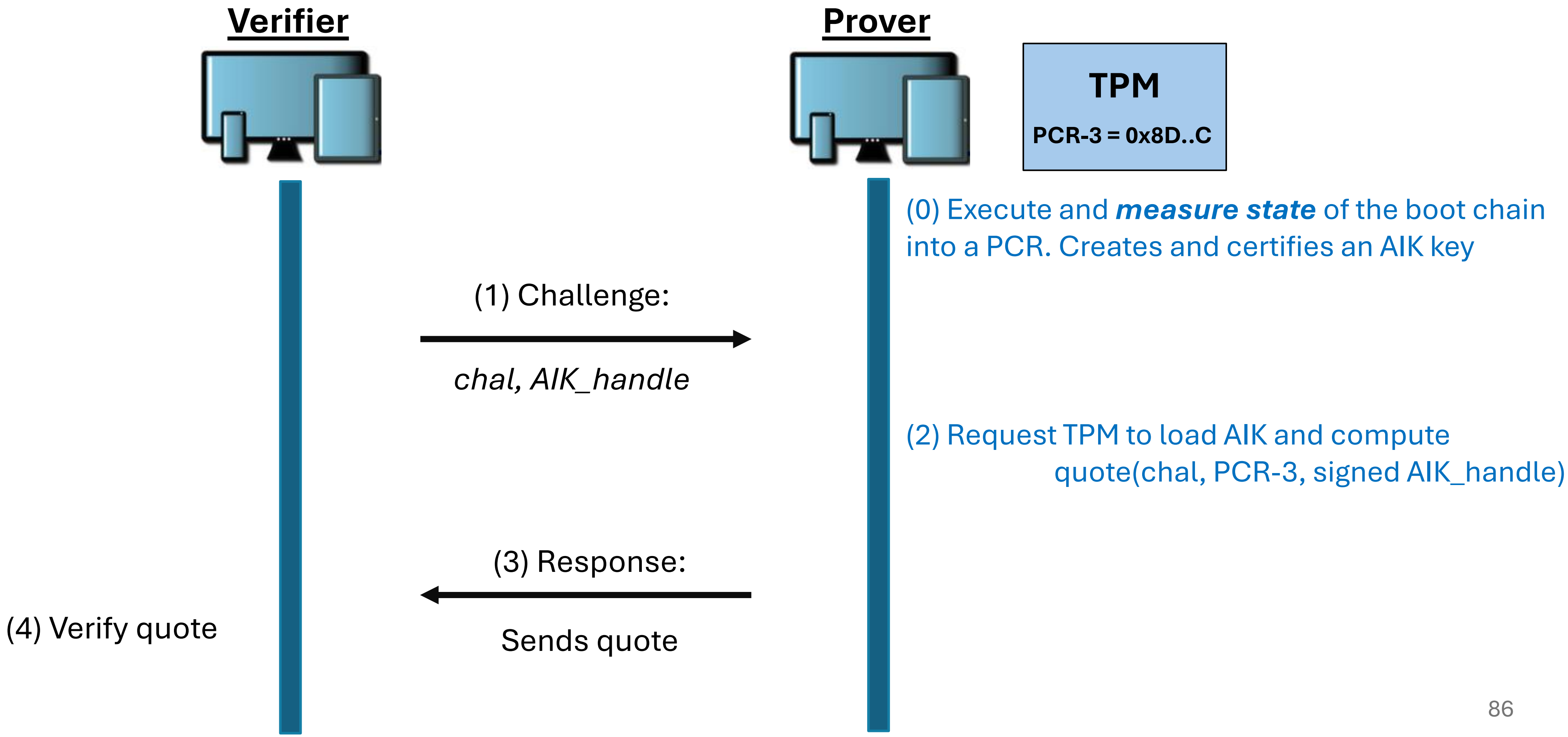
(3) Response:

Sends quote

TPM-based RA (v1)

<u>TPM Quote:</u> quote(nonce, PCRs (selection), AIK_handle)

- TPM uses AIK to sign selected PCRs and a nonce → returns a signature
- Nonce externally provided input (e.g., RA challenge)



Verifier

Prover

TPM

PCR-3 = 0x8D..C

(0) Execute and ***measure state*** of the boot chain into a PCR. Creates and certifies an AIK key

(1) Challenge:

chal, AIK_handle

(2) Request TPM to load AIK and compute quote(chal, PCR-3, signed AIK_handle)

(3) Response:

Sends quote

(4) Verify quote

TPM-based RA (v1)

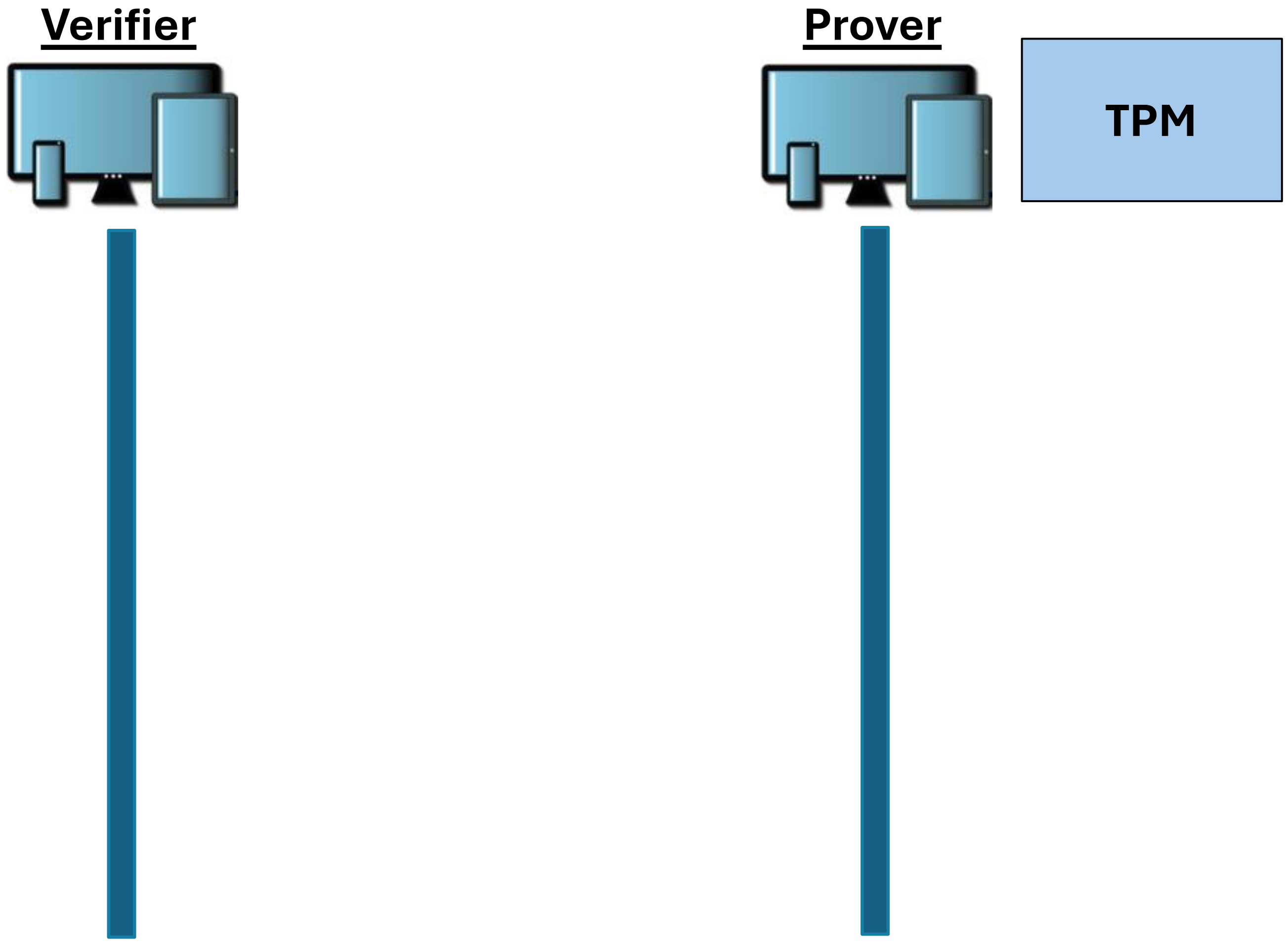
Verification chain

1. Check if the reported PCR(s) value(s) match the expected system state
2. Check the signature on the reported PCRs using the signed AIK public key
3. Check if AIK was signed by EK (using the public EK)
4. Check if public-EK is certified by the TPM manufacturer

TPM-based RA (v2)

Without quote: “seal based attestation”

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

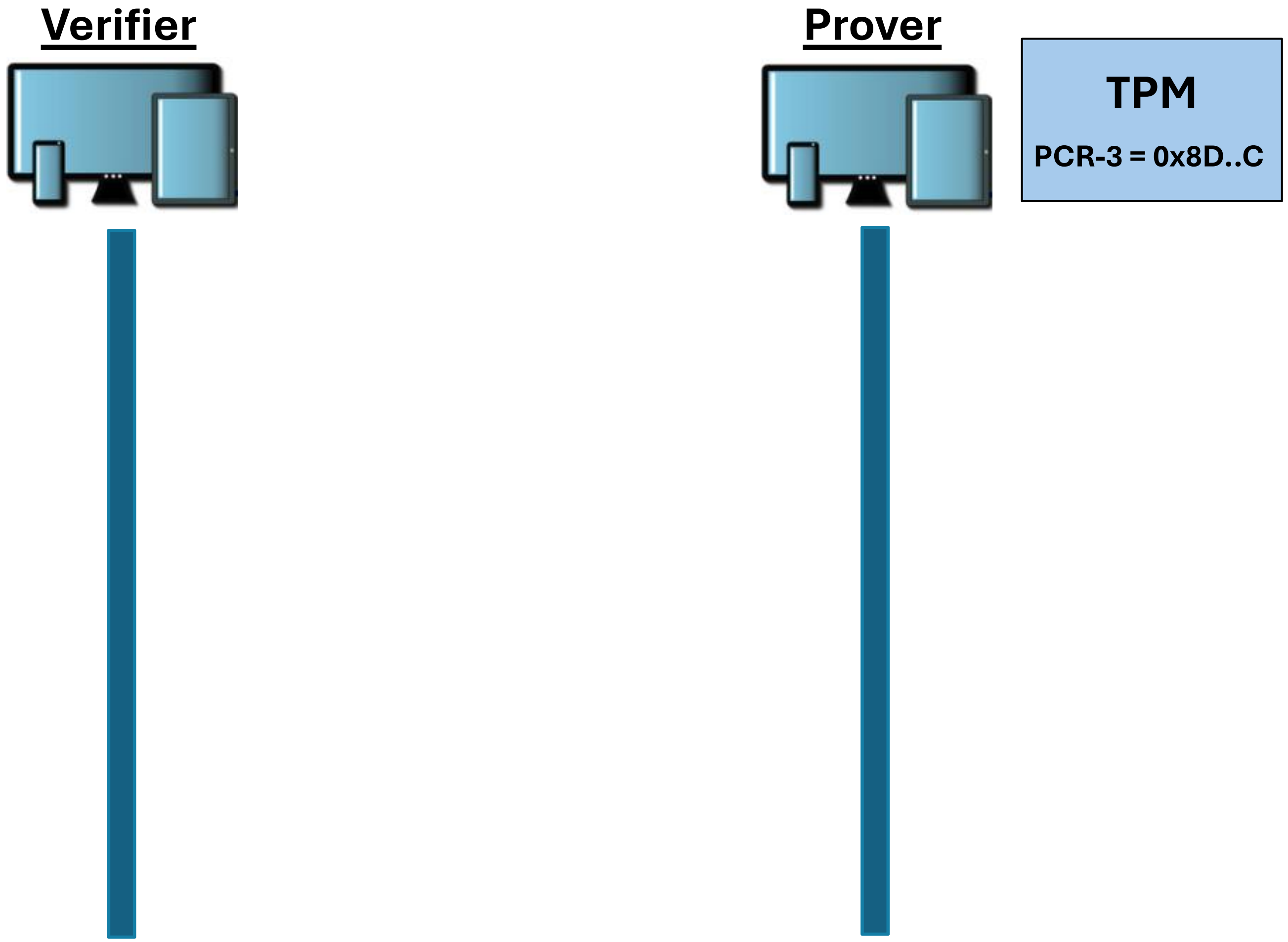
Prover

TPM

TPM-based RA (v2)

Without quote: “seal based attestation”

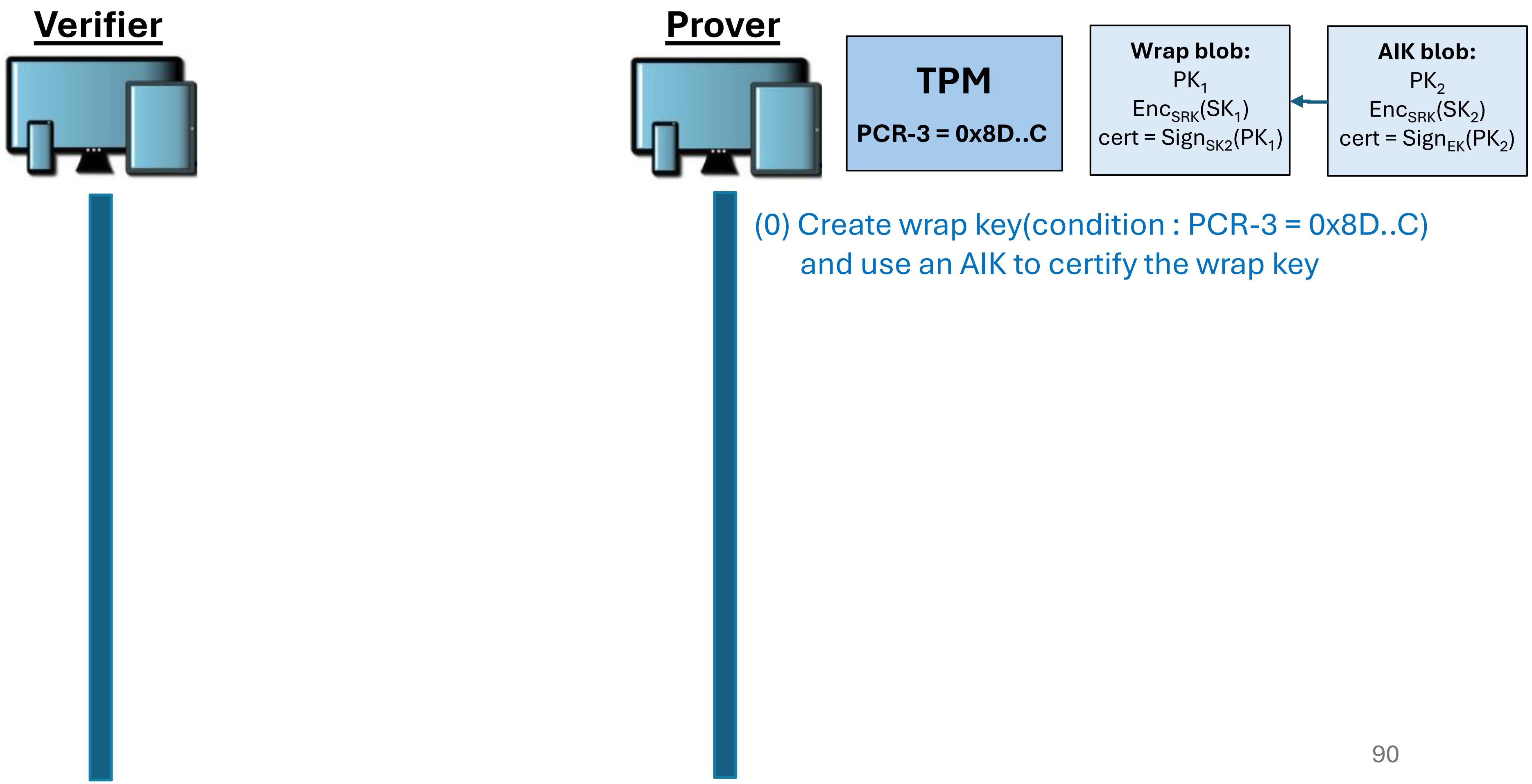
- Use wrap key → recall, use can be conditioned on a PCR state.



TPM-based RA (v2)

Without quote: "seal based attestation"

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

Prover

TPM

PCR-3 = 0x8D..C

Wrap blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{SK2}(PK_1)$

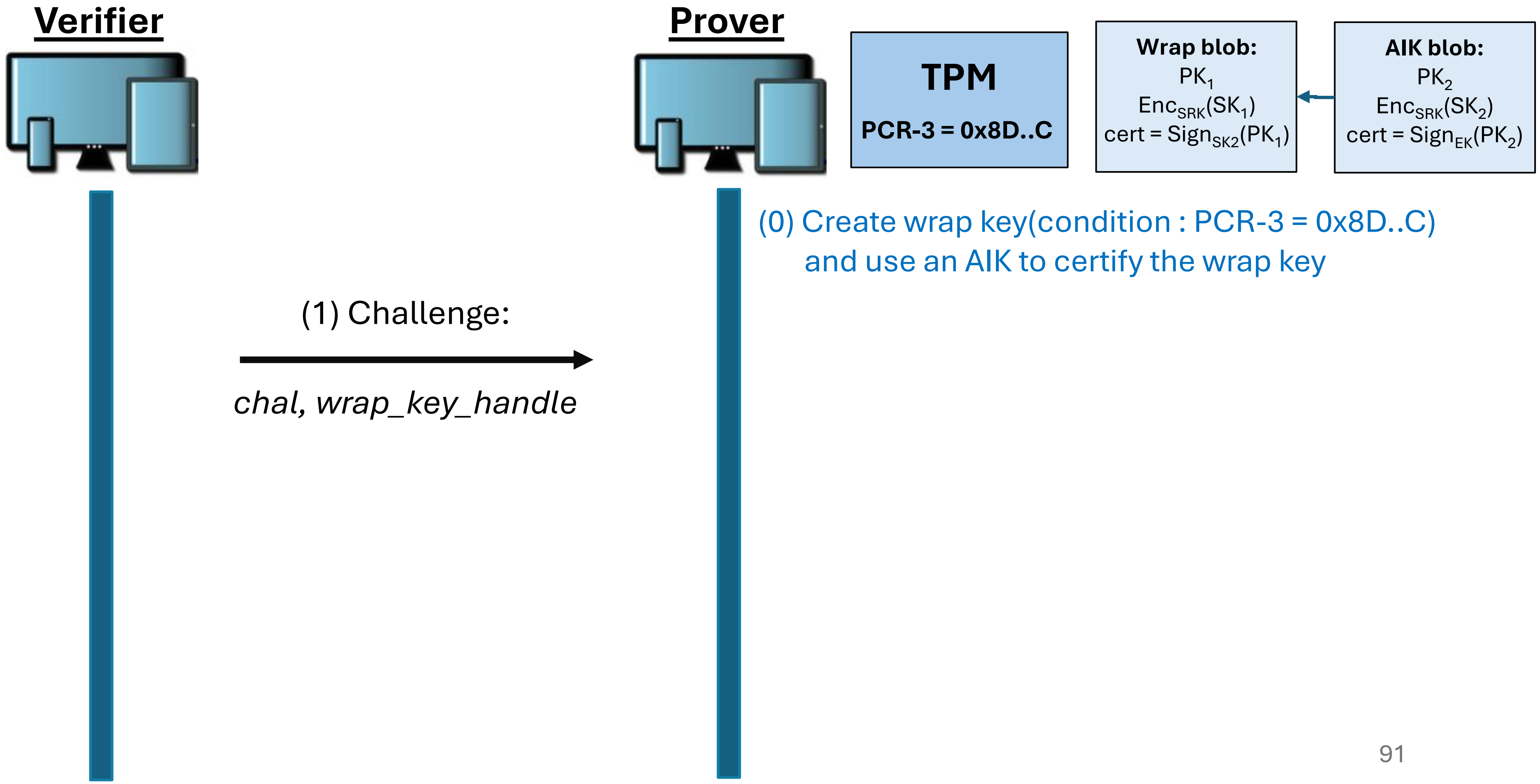
AIK blob:
PK_2
$Enc_{SRK}(SK_2)$
cert = $Sign_{EK}(PK_2)$

(0) Create wrap key(condition : PCR-3 = 0x8D..C)
and use an AIK to certify the wrap key

TPM-based RA (v2)

Without quote: “seal based attestation”

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

Prover

TPM

PCR-3 = 0x8D..C

Wrap blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{SK2}(PK_1)$

AIK blob:
PK_2
$Enc_{SRK}(SK_2)$
cert = $Sign_{EK}(PK_2)$

(0) Create wrap key(condition : PCR-3 = 0x8D..C) and use an AIK to certify the wrap key

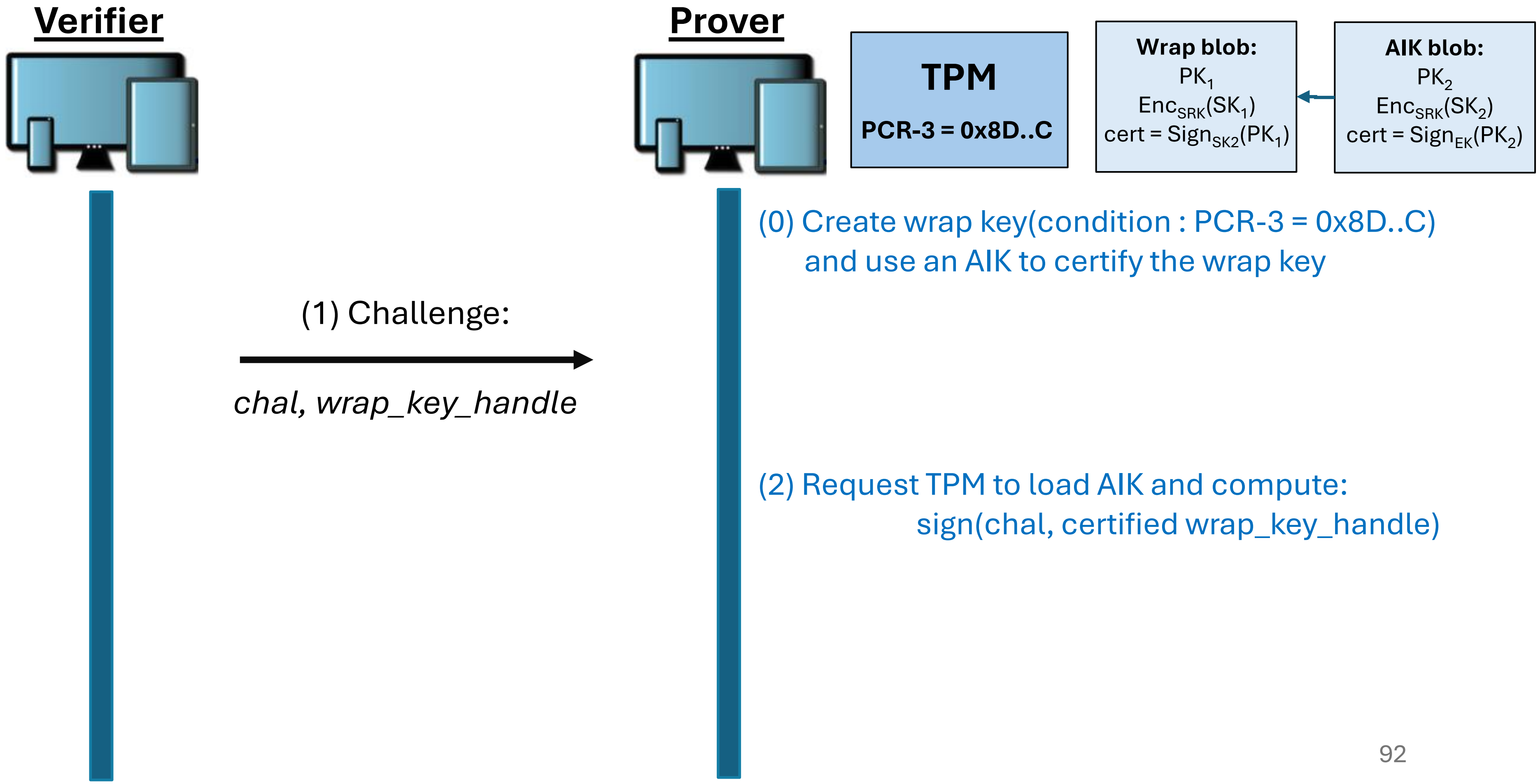
(1) Challenge:

chal, wrap_key_handle

TPM-based RA (v2)

Without quote: "seal based attestation"

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

Prover

TPM

PCR-3 = 0x8D..C

Wrap blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{SK2}(PK_1)$

AIK blob:
PK_2
$Enc_{SRK}(SK_2)$
cert = $Sign_{EK}(PK_2)$

(0) Create wrap key(condition : PCR-3 = 0x8D..C)
and use an AIK to certify the wrap key

(1) Challenge:

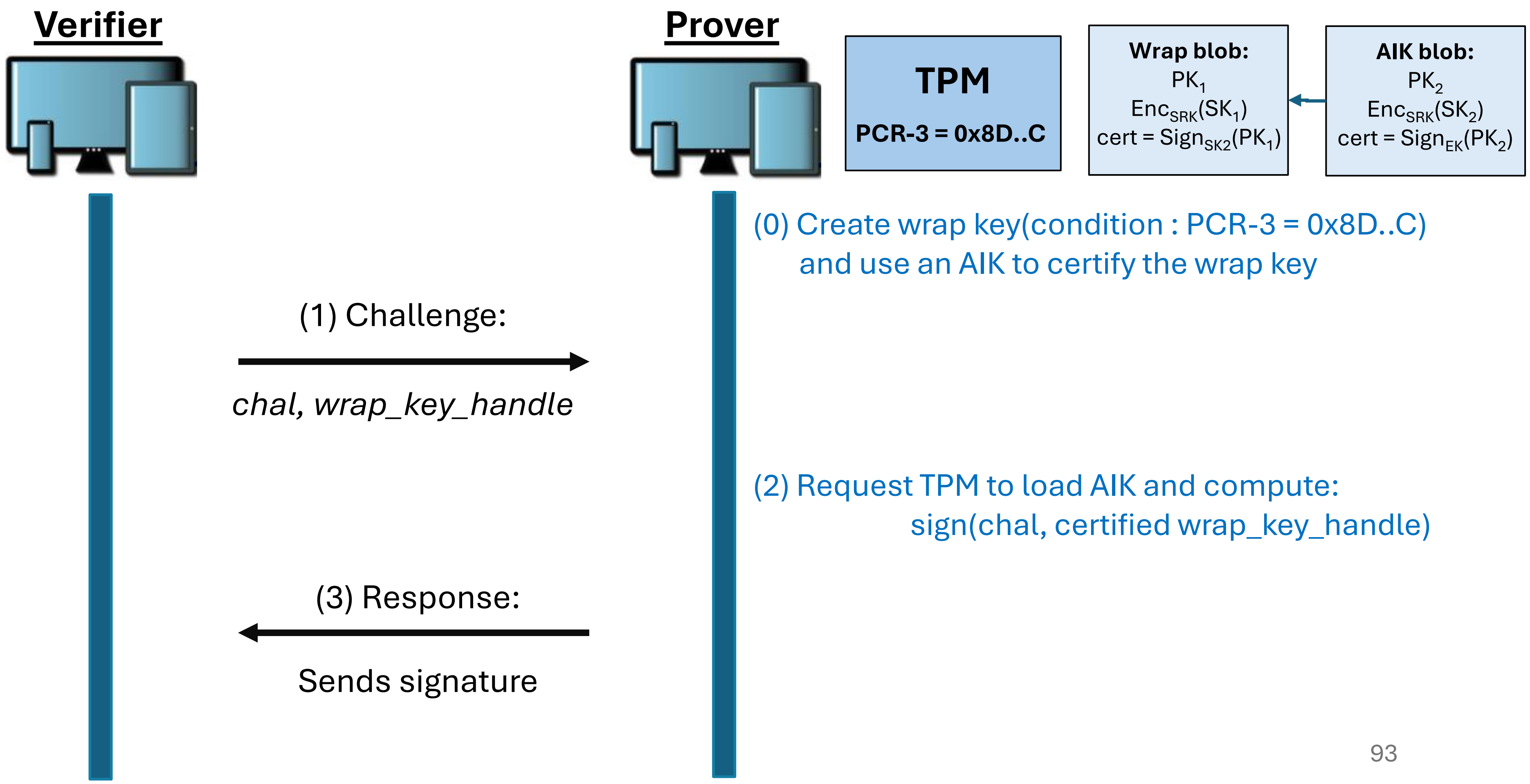
chal, wrap_key_handle

(2) Request TPM to load AIK and compute:
sign(chal, certified wrap_key_handle)

TPM-based RA (v2)

Without quote: “seal based attestation”

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

Prover

TPM

PCR-3 = 0x8D..C

Wrap blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{SK2}(PK_1)$

AIK blob:
PK_2
$Enc_{SRK}(SK_2)$
cert = $Sign_{EK}(PK_2)$

(0) Create wrap key(condition : PCR-3 = 0x8D..C) and use an AIK to certify the wrap key

(1) Challenge:

chal, wrap_key_handle

(2) Request TPM to load AIK and compute: sign(chal, certified wrap_key_handle)

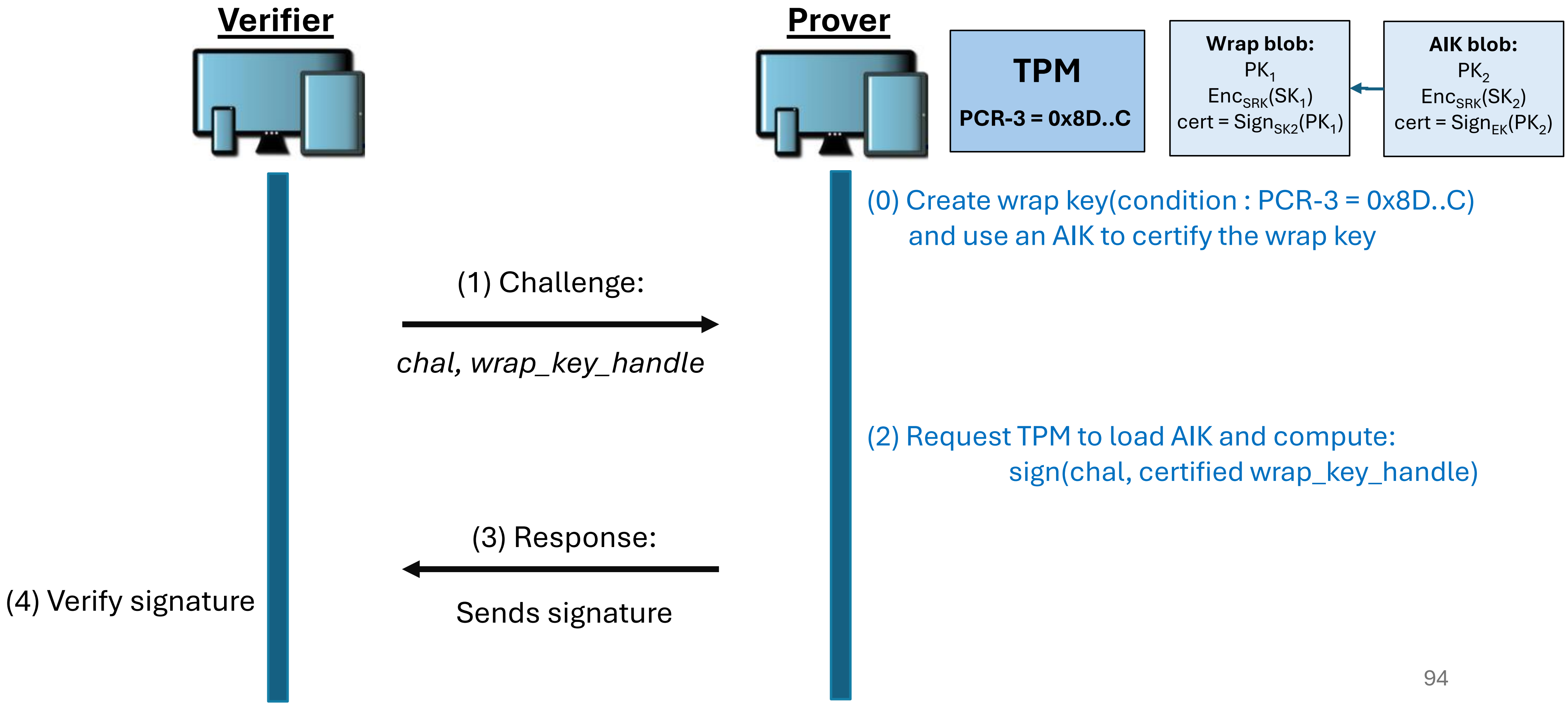
(3) Response:

Sends signature

TPM-based RA (v2)

Without quote: "seal based attestation"

- Use wrap key → recall, use can be conditioned on a PCR state.



Verifier

Prover

TPM

PCR-3 = 0x8D..C

Wrap blob:
PK_1
$Enc_{SRK}(SK_1)$
cert = $Sign_{SK2}(PK_1)$

AIK blob:
PK_2
$Enc_{SRK}(SK_2)$
cert = $Sign_{EK}(PK_2)$

(0) Create wrap key(condition : PCR-3 = 0x8D..C)
and use an AIK to certify the wrap key

(1) Challenge:

chal, wrap_key_handle

(2) Request TPM to load AIK and compute:
sign(chal, certified wrap_key_handle)

(3) Response:

Sends signature

(4) Verify signature

TPM-based RA (v2)

Verification chain

1. Check if the reported PCR(s) value(s) match the expected system state
2. Check the signature on the reported PCRs using the certified wrap key
3. Check if wrap key was signed by known AIK (using the public AIK)
4. Check if AIK was signed by EK (using the public EK)
5. Check if public-EK is certified by the TPM manufacturer

TPM operations

Similar conditional operation:

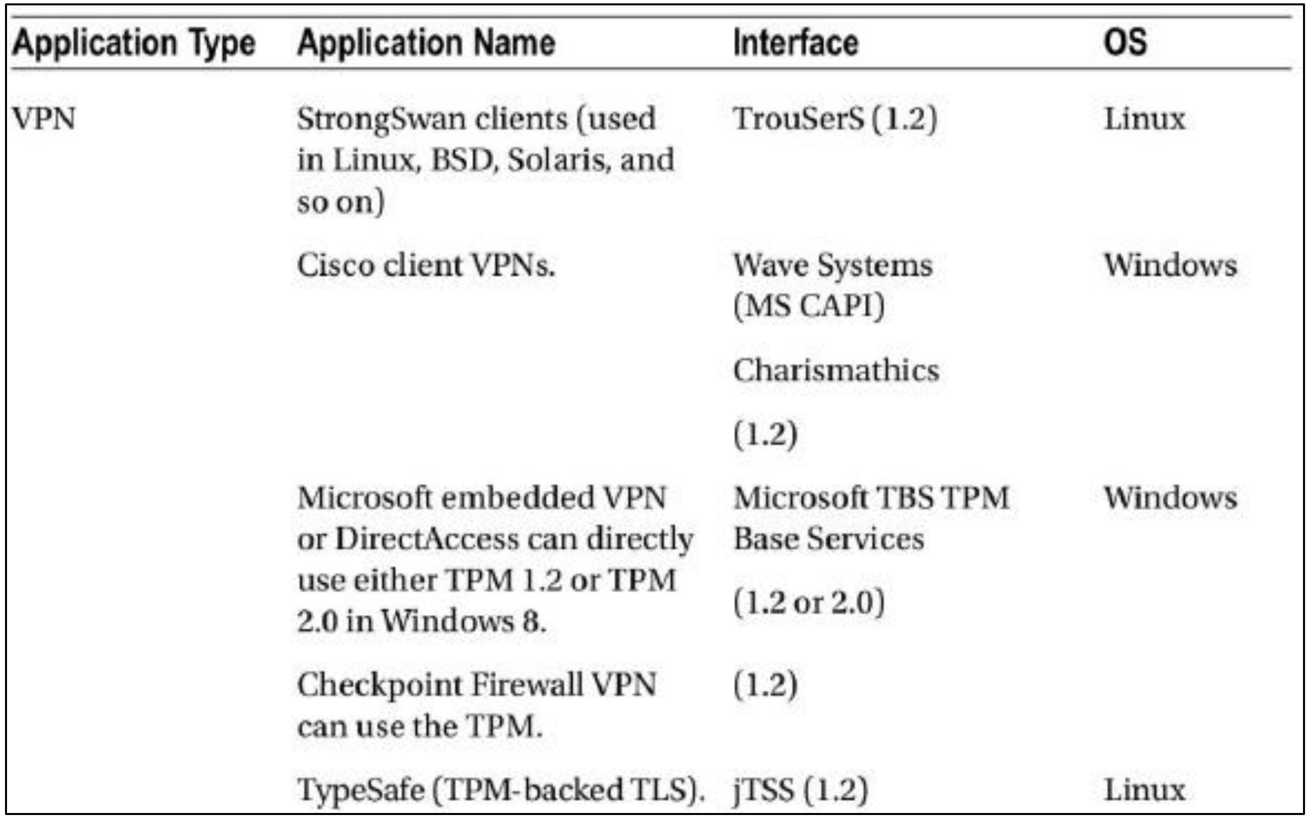
TPM_Seal

- Encrypts data, conditions decryption on PCR state

TPM Applications

Many applications can benefit from TPM

- Can be used to implement secure boot (though not required)
- Other applications:

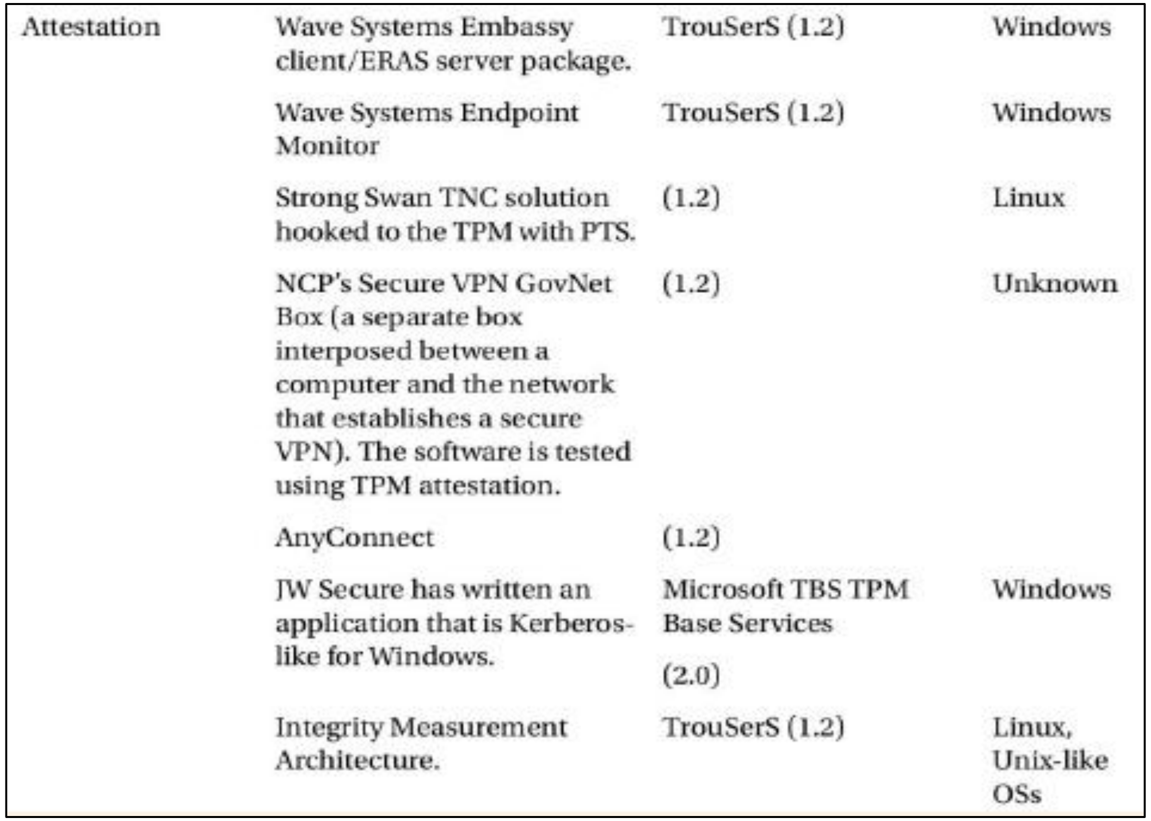


Application Type	Application Name	Interface	OS
VPN	StrongSwan clients (used in Linux, BSD, Solaris, and so on)	TrouSerS (1.2)	Linux
	Cisco client VPNs.	Wave Systems (MS CAPI) Charismathics (1.2)	Windows
	Microsoft embedded VPN or DirectAccess can directly use either TPM 1.2 or TPM 2.0 in Windows 8.	Microsoft TBS TPM Base Services (1.2 or 2.0)	Windows
	Checkpoint Firewall VPN can use the TPM.	(1.2)	
	TypeSafe (TPM-backed TLS).	jTSS (1.2)	Linux

TPM Applications

Many applications can benefit from TPM

- Can be used to implement secure boot (though not required)
- Other applications:

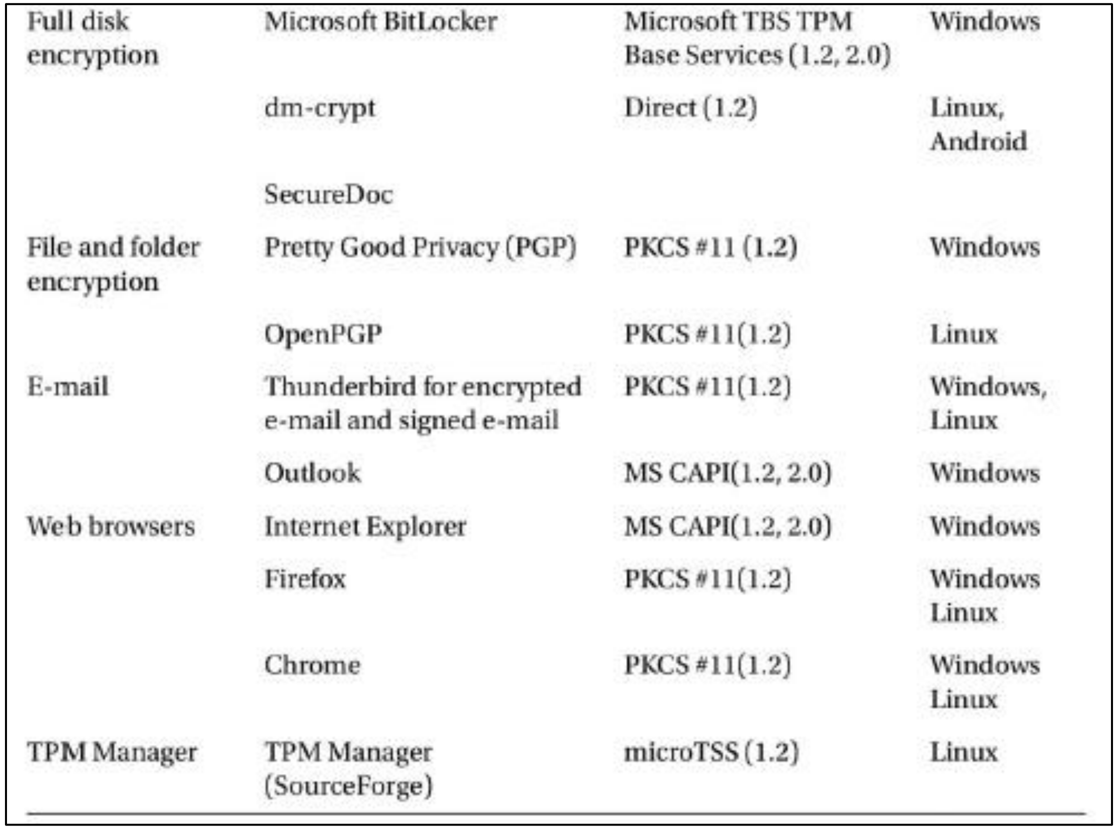


Attestation	Wave Systems Embassy client/ERAS server package.	TrouSerS (1.2)	Windows
	Wave Systems Endpoint Monitor	TrouSerS (1.2)	Windows
	Strong Swan TNC solution hooked to the TPM with PTS.	(1.2)	Linux
	NCP's Secure VPN GovNet Box (a separate box interposed between a computer and the network that establishes a secure VPN). The software is tested using TPM attestation.	(1.2)	Unknown
	AnyConnect	(1.2)	
	JW Secure has written an application that is Kerberos-like for Windows.	Microsoft TBS TPM Base Services (2.0)	Windows
	Integrity Measurement Architecture.	TrouSerS (1.2)	Linux, Unix-like OSs

TPM Applications

Many applications can benefit from TPM

- Can be used to implement secure boot (though not required)
- Other applications:



Full disk encryption	Microsoft BitLocker	Microsoft TBS TPM Base Services (1.2, 2.0)	Windows
	dm-crypt	Direct (1.2)	Linux, Android
	SecureDoc		
File and folder encryption	Pretty Good Privacy (PGP)	PKCS #11 (1.2)	Windows
	OpenPGP	PKCS #11(1.2)	Linux
E-mail	Thunderbird for encrypted e-mail and signed e-mail	PKCS #11(1.2)	Windows, Linux
	Outlook	MS CAPI(1.2, 2.0)	Windows
Web browsers	Internet Explorer	MS CAPI(1.2, 2.0)	Windows
	Firefox	PKCS #11(1.2)	Windows Linux
	Chrome	PKCS #11(1.2)	Windows Linux
TPM Manager	TPM Manager (SourceForge)	microTSS (1.2)	Linux

Concluding thoughts

Nice characteristics of TPM:

- Logically separated from CPU and main system
- Provides core building block cryptographic operations
- Provides state-aware operations

Limitations:

- Not programmable
- Do not provide a run-time environment: protects data, but not the host itself
- Passive: no availability guarantees if the host is compromised

That’s all for today!

Coming up….

- Alternative designs that can address limitations of TPM
- Trusted Execution Environments
  - User-space TEE in Servers → Intel SGX
  - System-wide TEE in Mobile → ARM TrustZone

Reminders:

- A3 is due on July 11
- Research project proposals

Resources:

- TPM specifications: 1.2, 2.0
- More TPM details (Microsoft)
- Simulating TPM