DTU

Future Proofing the Connected World

Practical TPM Extensions in ICT Systems

July 22nd 2020
TheCamp

Agenda

- Introduction to Trusted Computing as a concept
- The TPM
- Platform Configuration Registers
- Key Storage
- Using TPMs to enable trust in networks
- Extended Authorization
- Practical Exercises

Trusted Computing

What is **trust?**

Trusted Computing

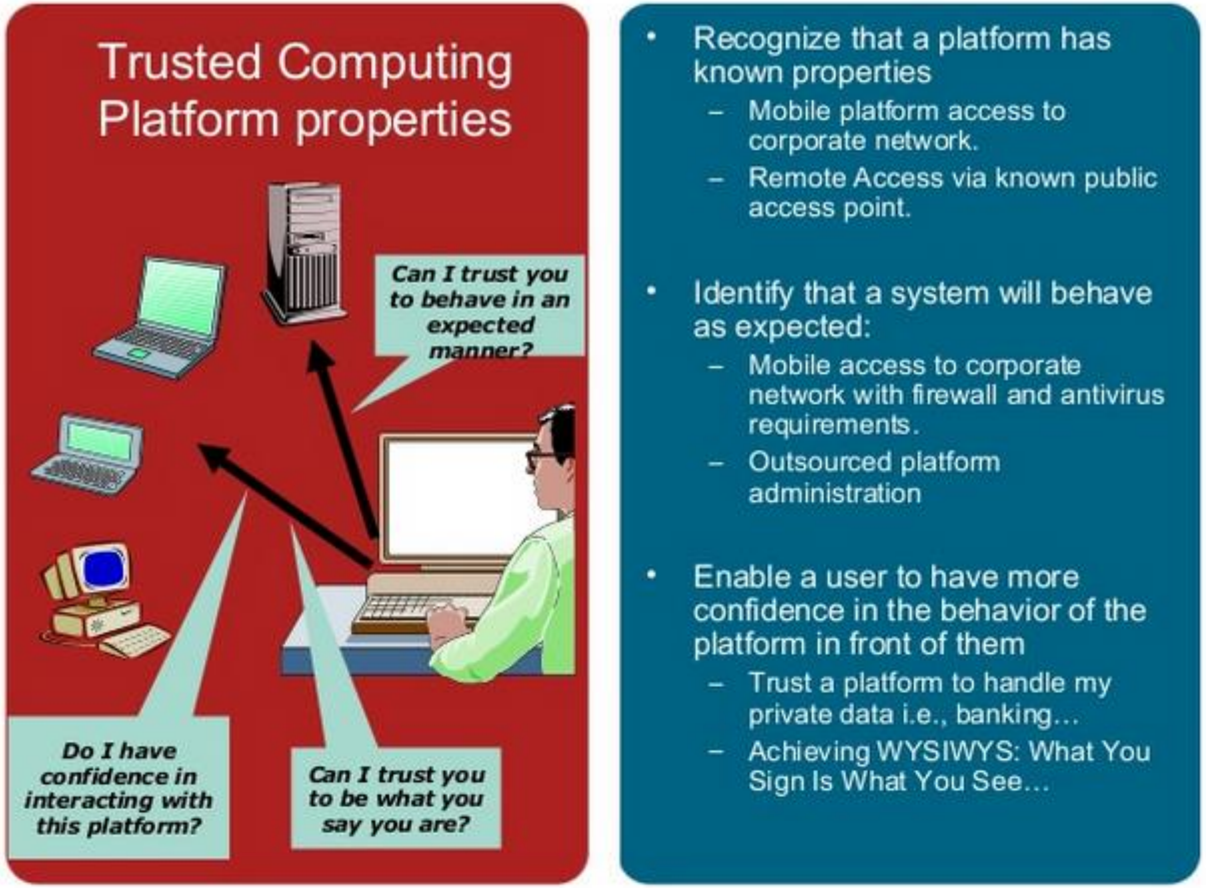
Why do we **trust?**

What defines **trustworthy**?

Trusted Computing

Why do we need **Trusted Computing?**

Trusted Computing



Trusted Computing Platform properties

Can I trust you to behave in an expected manner?
Do I have confidence in interacting with this platform?
Can I trust you to be what you say you are?

- Recognize that a platform has known properties
  - Mobile platform access to corporate network.
  - Remote Access via known public access point.
- Identify that a system will behave as expected:
  - Mobile access to corporate network with firewall and antivirus requirements.
  - Outsourced platform administration
- Enable a user to have more confidence in the behavior of the platform in front of them
  - Trust a platform to handle my private data i.e., banking…
  - Achieving WYSIWYS: What You Sign Is What You See…

Trusted Computing

Trusted

Platform

Module

~~Measurement~~
(very diffy, such hard, many complicate)

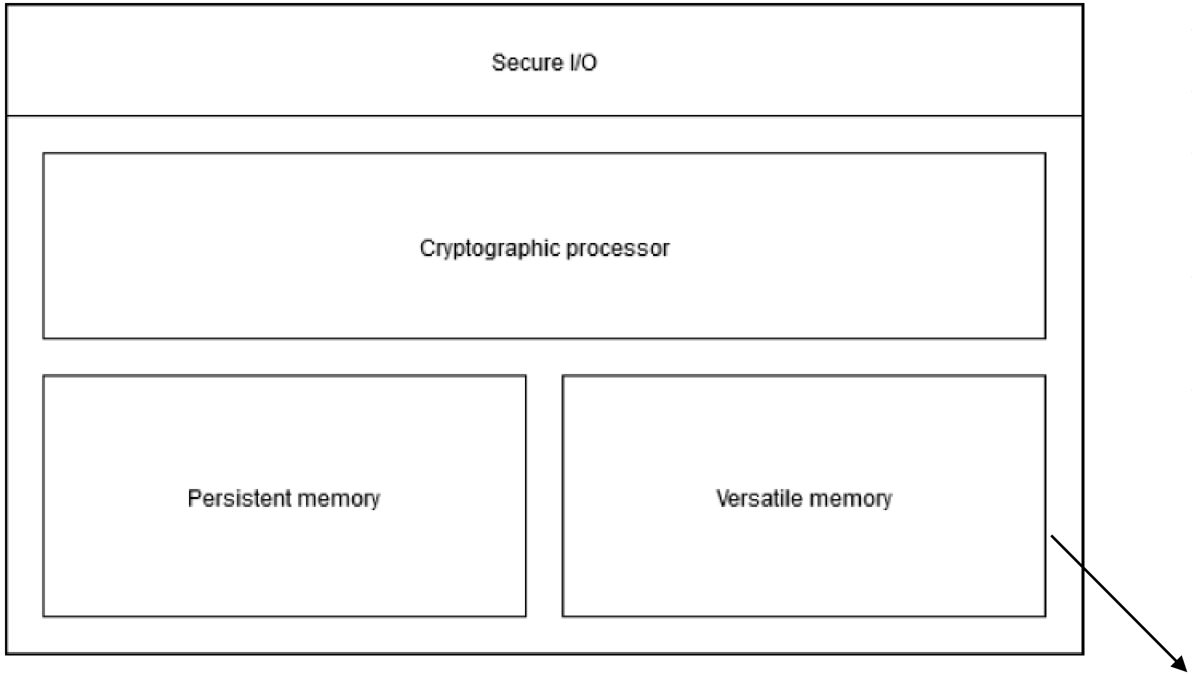
Storage

Reporting

Trusted Computing

- TPM is widely used already
  - Microsoft Bitlocker, Windows Hello, Measured Boot, HP Protect Tools
  - Secure boot
  - Intel’s Trusted Execution Technology (TXT)
  - Linux Unified Key Setup (LUKS) supports storing cryptographic keys in TPMs
  - IMA
- We will (hopefully) see it in many future applications
  - TPMs in automotive contexts
  - TPMs in industrial contexts
  - TPMs in general IoT (forces them to become even smaller!)
- And it’s currently being developed to be *quantum resistant!*

Anatomy of the TPM



- Slow processor
- ~4k non-volatile memory
- Volatile memory can only handle ~ 3 keys!

- It’s cheap for a reason… How do we manage multiple entities?
- Platform Configuration Registers
  - Extendable
  - Only clears during boot
  - Represent system state (e.g 0-7 is boot)

PCR 1
PCR 2
PCR 3
…
PCR 24

PCRs continued

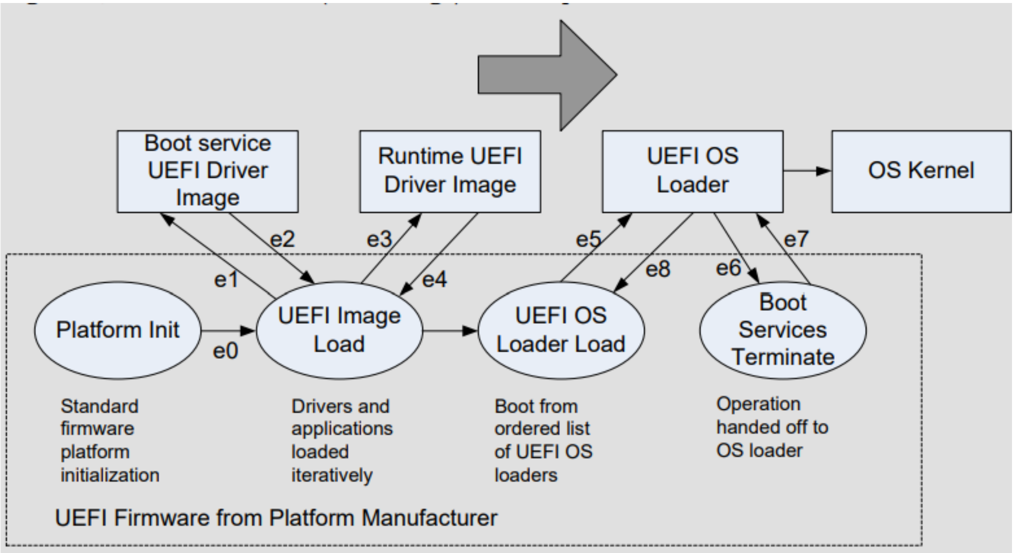

Boot service UEFI Driver Image
Runtime UEFI Driver Image
UEFI OS Loader
OS Kernel
e1
e2
e3
e4
e5
e8
e6
e7
Platform Init
e0
UEFI Image Load
UEFI OS Loader Load
Boot Services Terminate
Standard firmware platform initialization
Drivers and applications loaded iteratively
Boot from ordered list of UEFI OS loaders
Operation handed off to OS loader
UEFI Firmware from Platform Manufacturer

Figure 5 UEFI Platform Boot Process

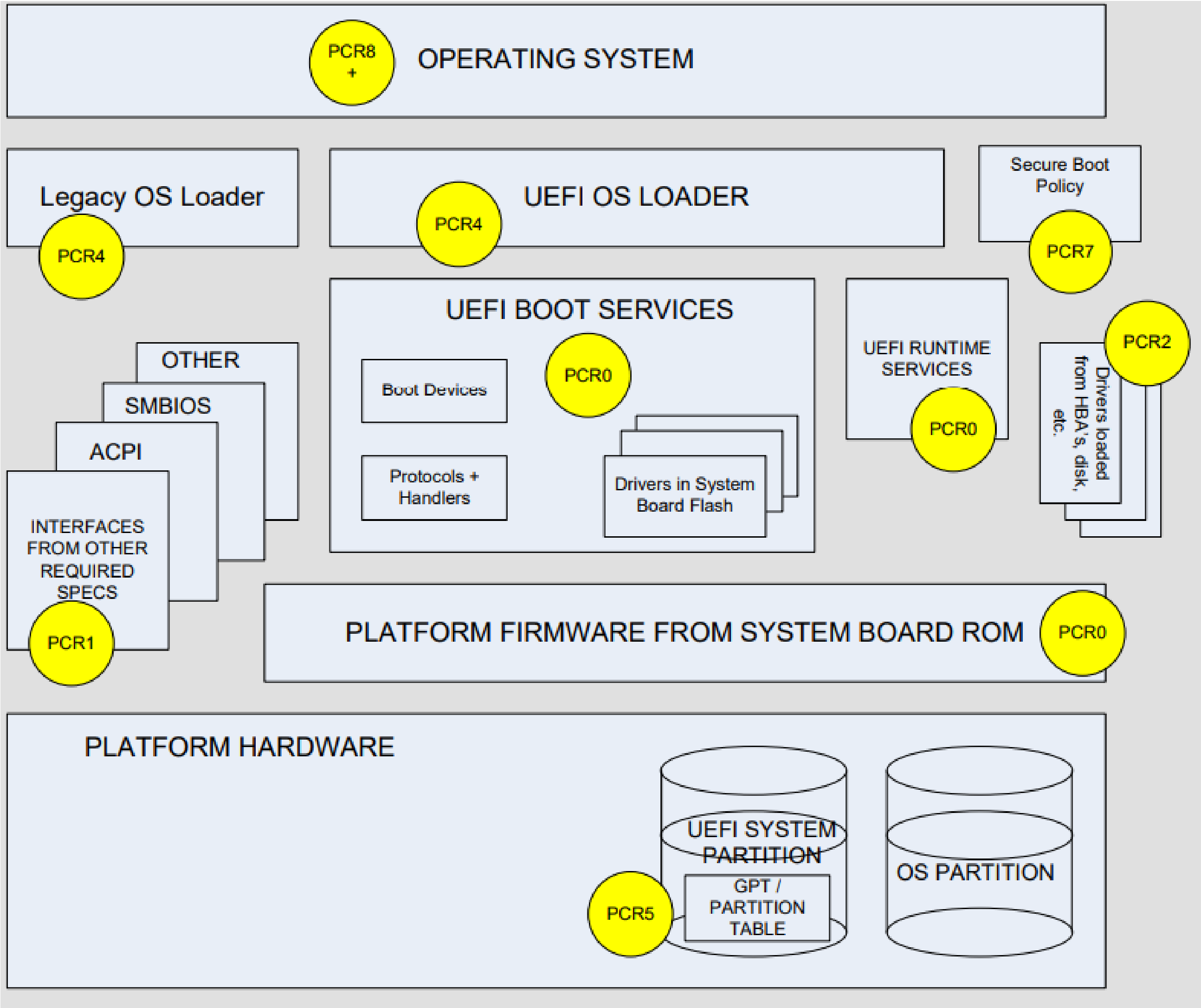


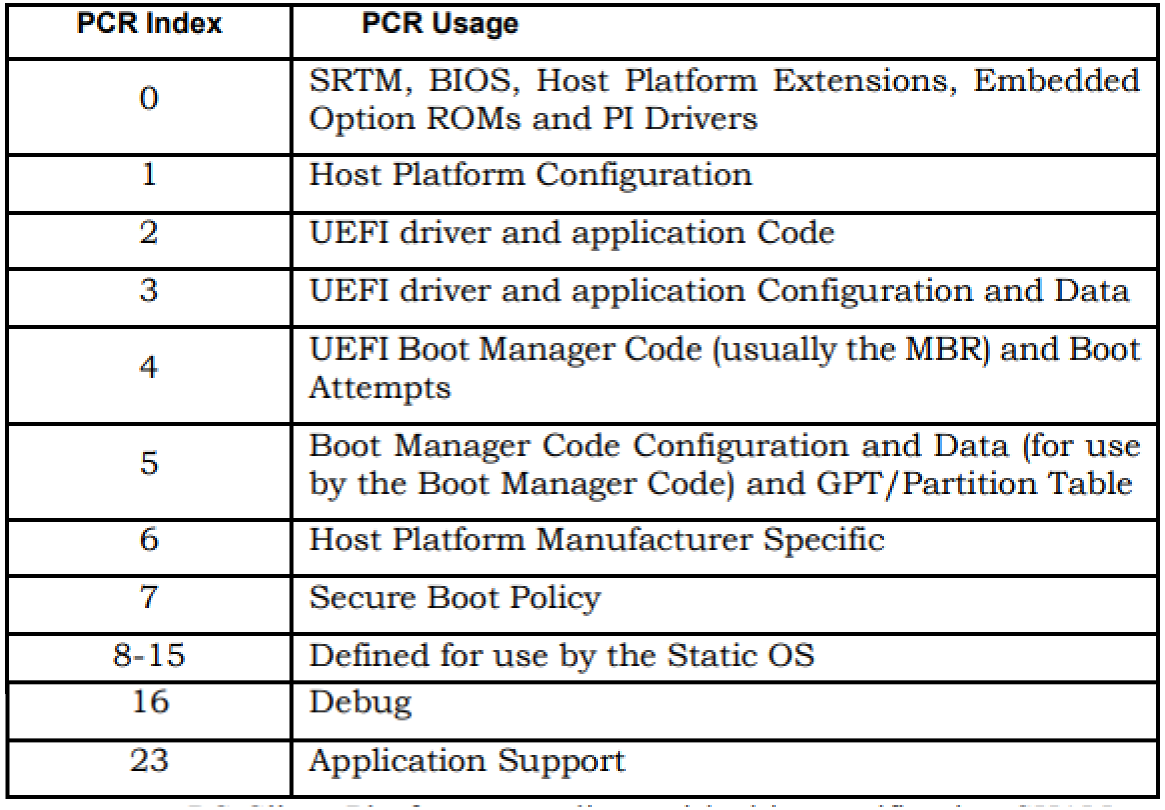
Figure 6 PCR Mapping of UEFI Components

E0: RoT-M: Measure UEFI Firmware to PCR 0

…

E5: Measure PE/COFF image (OS loader) to PCR 4 (0,2)

PCRs continued

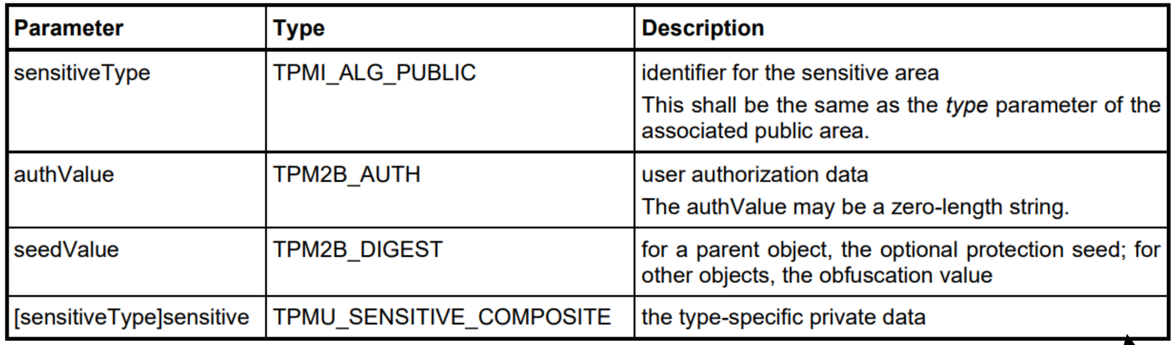


PCR Index	PCR Usage
0	SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers
1	Host Platform Configuration
2	UEFI driver and application Code
3	UEFI driver and application Configuration and Data
4	UEFI Boot Manager Code (usually the MBR) and Boot Attempts
5	Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table
6	Host Platform Manufacturer Specific
7	Secure Boot Policy
8-15	Defined for use by the Static OS
16	Debug
23	Application Support

Focus of today

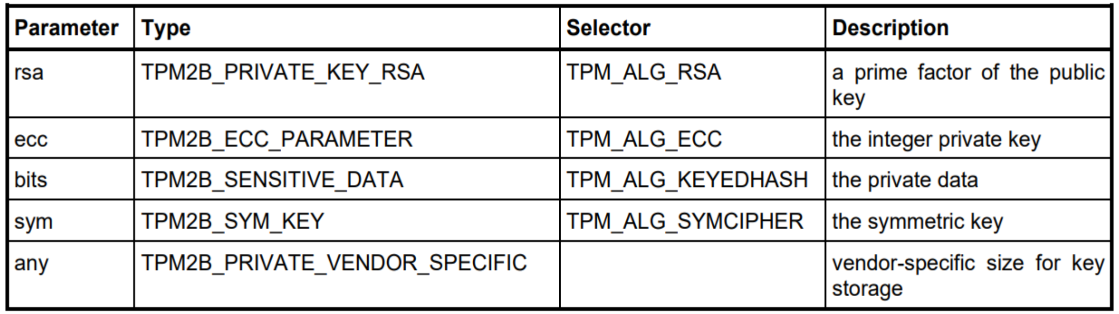
- We will investigate how we can use secure cryptographic keys
  - How do we generate them?
  - How do we secure them?
    - E.g. by use of PCRs
  - Most important: What can we do with them?
  - **AND WE ARE GONNA DO EXERCISES :D**

Cryptographic Keys



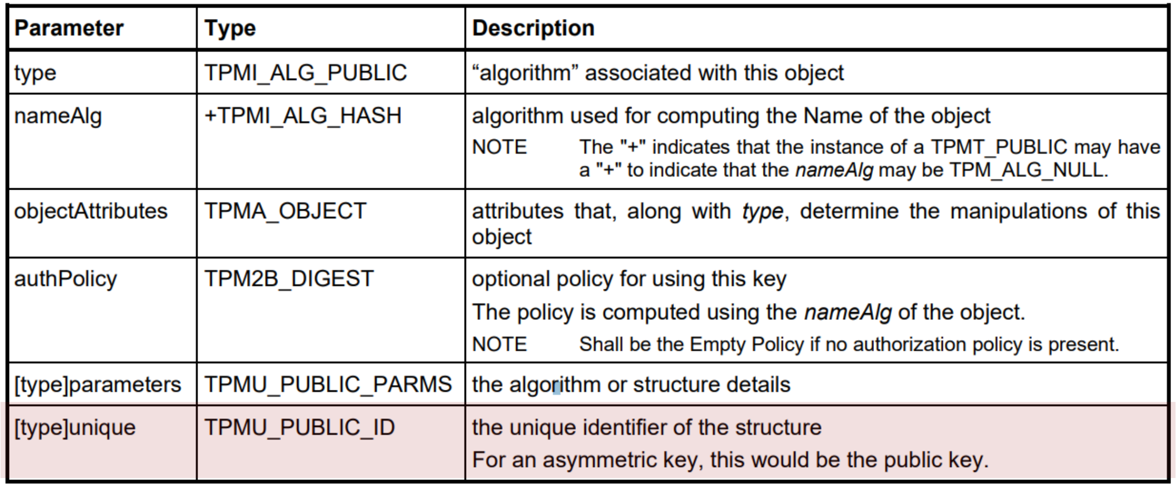
Parameter	Type	Description
sensitiveType	TPMI_ALG_PUBLIC	identifier for the sensitive area This shall be the same as the *type* parameter of the associated public area.
authValue	TPM2B_AUTH	user authorization data The authValue may be a zero-length string.
seedValue	TPM2B_DIGEST	for a parent object, the optional protection seed; for other objects, the obfuscation value
[sensitiveType]sensitive	TPMU_SENSITIVE_COMPOSITE	the type-specific private data

Table 205 (Part 2)



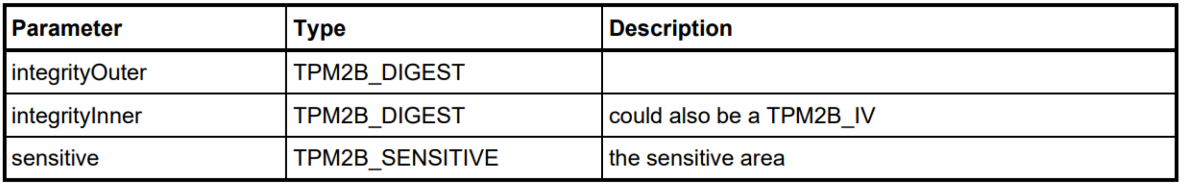
Parameter	Type	Selector	Description
rsa	TPM2B_PRIVATE_KEY_RSA	TPM_ALG_RSA	a prime factor of the public key
ecc	TPM2B_ECC_PARAMETER	TPM_ALG_ECC	the integer private key
bits	TPM2B_SENSITIVE_DATA	TPM_ALG_KEYEDHASH	the private data
sym	TPM2B_SYM_KEY	TPM_ALG_SYMCIPHER	the symmetric key
any	TPM2B_PRIVATE_VENDOR_SPECIFIC		vendor-specific size for key storage

Table 204 (Part 2)



Parameter	Type	Description
type	TPMI_ALG_PUBLIC	"algorithm" associated with this object
nameAlg	+TPMI_ALG_HASH	algorithm used for computing the Name of the object NOTE The "+" indicates that the instance of a TPMT_PUBLIC may have a "+" to indicate that the *nameAlg* may be TPM_ALG_NULL.
objectAttributes	TPMA_OBJECT	attributes that, along with *type*, determine the manipulations of this object
authPolicy	TPM2B_DIGEST	optional policy for using this key The policy is computed using the *nameAlg* of the object. NOTE Shall be the Empty Policy if no authorization policy is present.
[type]parameters	TPMU_PUBLIC_PARMS	the algorithm or structure details
[type]unique	TPMU_PUBLIC_ID	the unique identifier of the structure For an asymmetric key, this would be the public key.

Table 200 (Part 2) Public part

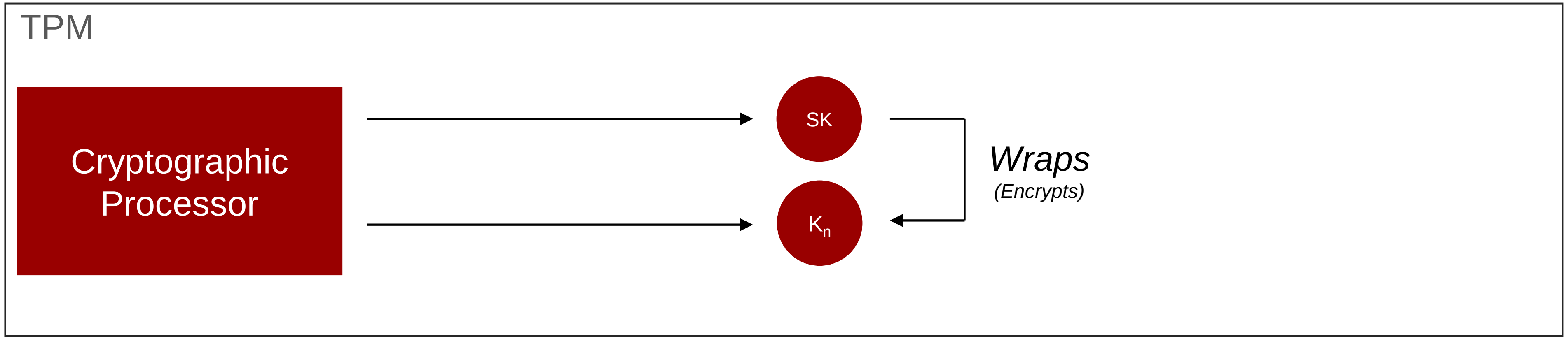


Parameter	Type	Description
integrityOuter	TPM2B_DIGEST	
integrityInner	TPM2B_DIGEST	could also be a TPM2B_IV
sensitive	TPM2B_SENSITIVE	the sensitive area

Table 207 (Part 2): Private (encrypted) part

Cryptographic Keys

How do we create and use them with such limited space?

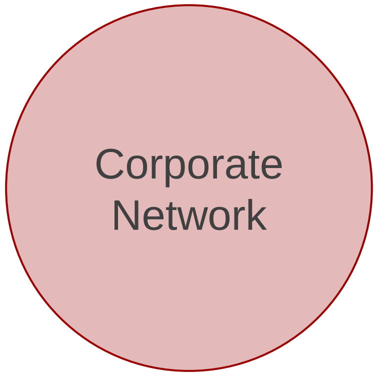


1) Create a Storage Key (SK, Primary Key) (**never leaves the TPM)**
2) Create a Cryptographic Key (encryption key, signing key, etc)
3) Wrap (encrypt) the newly created key by the Storage Key
4) Evict wrapped (encrypted) key to host

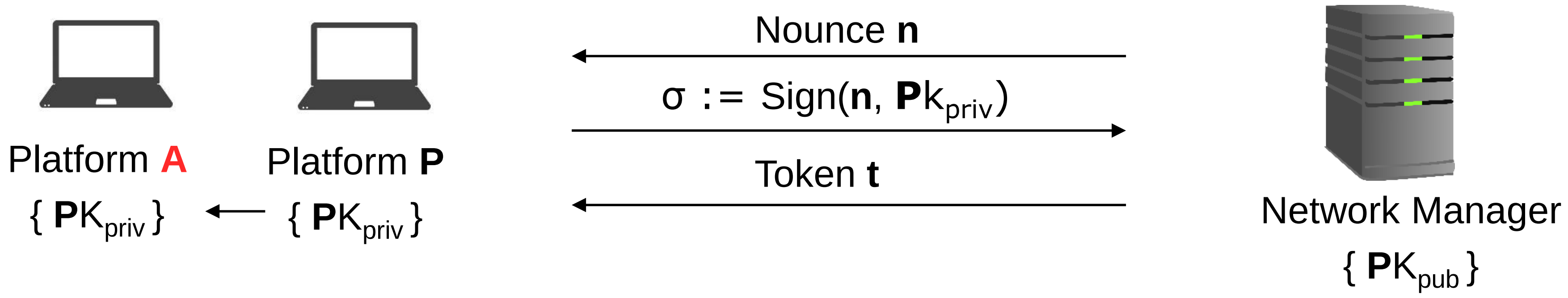
Achieving Trust in Networks

…The naïve approach

Goal: Only issued platforms may log on to the network.



Corporate Network



Nounce **n**

$\sigma := \text{Sign}(\mathbf{n}, \mathbf{P}\text{k}_{priv})$

Token **t**

Platform **A**

$\{ \mathbf{P}\text{K}_{priv} \}$

Platform **P**

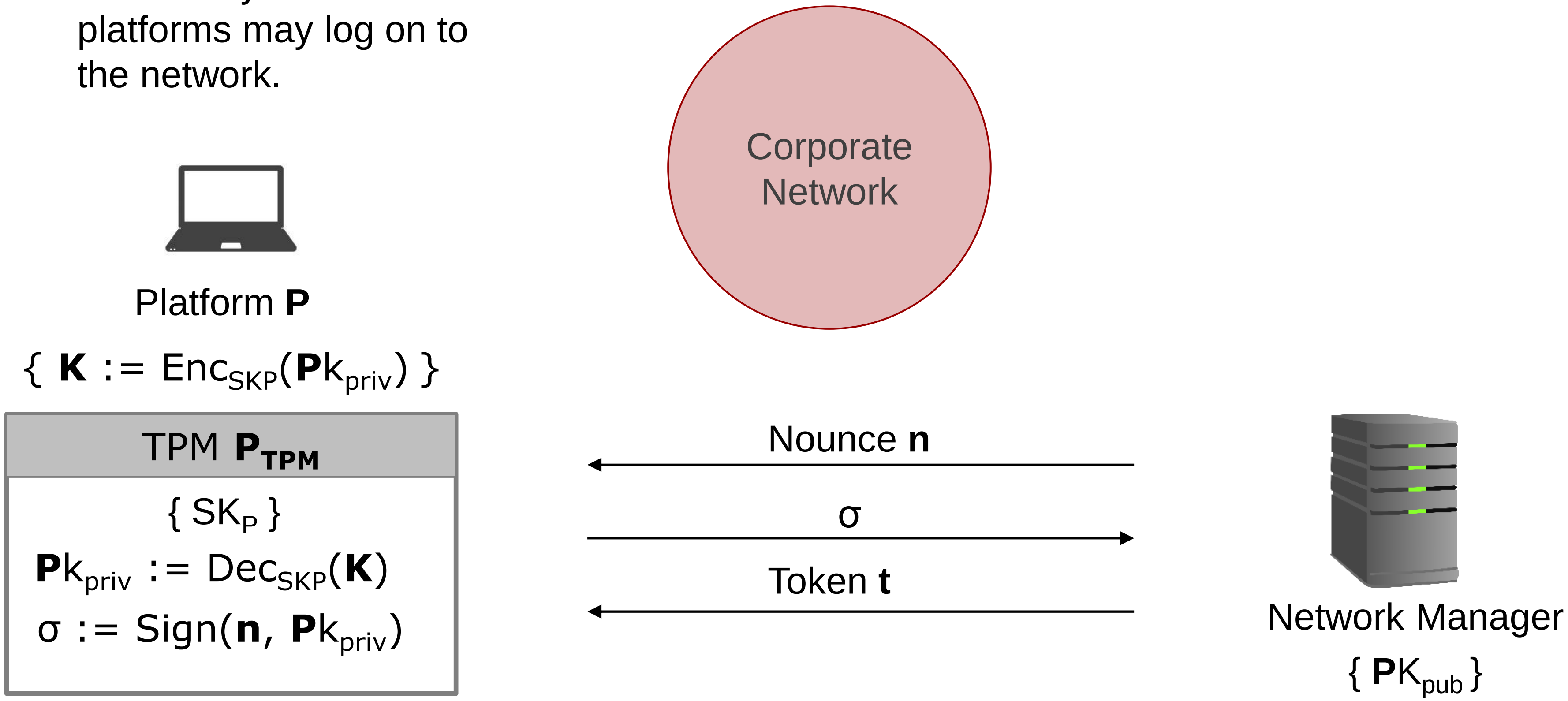
$\{ \mathbf{P}\text{K}_{priv} \}$

Network Manager

$\{ \mathbf{P}\text{K}_{pub} \}$

Achieving Trust in Networks

Goal: Only issued platforms may log on to the network.



Corporate Network

Platform **P**

$\{\ \mathbf{K} := \mathrm{Enc}_{SKP}(\mathbf{P}k_{priv})\ \}$

TPM $\mathbf{P}_{\mathbf{TPM}}$

$\{\ SK_P\ \}$

$\mathbf{P}k_{priv} := \mathrm{Dec}_{SKP}(\mathbf{K})$

$\sigma := \mathrm{Sign}(\mathbf{n}, \mathbf{P}k_{priv})$

Nounce **n**

σ

Token **t**

Network Manager

$\{\ \mathbf{P}K_{pub}\ \}$

Achieving Trust in Networks

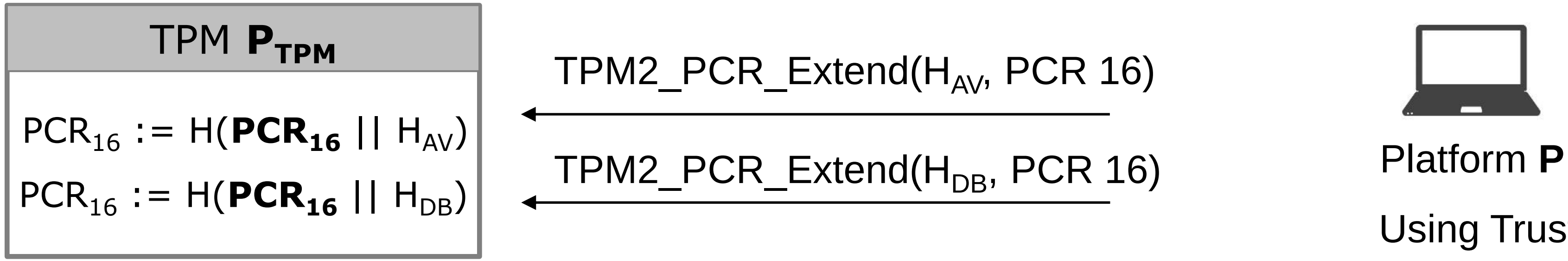
Goal: Only issued platforms with antivirus software **s** and virus database version **d** may log on to the network.

- Company policies is far away from *guaranteeing* that software is installed
  - Hurr durr, we automatically deploy the newest software all the time
  - …When you *are* logged on.
- But we can’t just start sending huge files to a verifier every time we want to log on.
- We could send a hash of the files during the authentication?
  - But we don’t have trust in the platform – an adversary (or user) could just send the correct values...
- **Any ideas?** (Hint: UEFI)

We trust the PCRs to reflect the *actual* configuration – what if we could bind the key to specific PCR contents?

Enhanced Authorization

After all other measurements are done…



TPM $\mathbf{P_{TPM}}$

$PCR_{16} := H(\mathbf{PCR_{16}} \,||\, H_{AV})$

$PCR_{16} := H(\mathbf{PCR_{16}} \,||\, H_{DB})$

TPM2_PCR_Extend(H_{AV}, PCR 16)

TPM2_PCR_Extend(H_{DB}, PCR 16)

Platform **P**

Using Trusted Measurement Agent:
$\mathbf{H_{AV}}$:= H(Antivirus.exe)
$\mathbf{H_{DB}}$:= H(Database.db)

Assuming PCR 16 isn't used for anything else, it will not purely represent the antivirus software *and* the database update, let's say the correct value would be **0xCC**.

Now let's say the key has a policy auth value of this….

Extended Authorization (EA)

- All keys can be protected by a governing policy

  PolicyPCR: Binds the use of an entity to certain PCR values

  PolicySigned: Binds the use of the key to a signature from another key

  PolicyCommand: Binds the use of the key to certain commands

  And much more

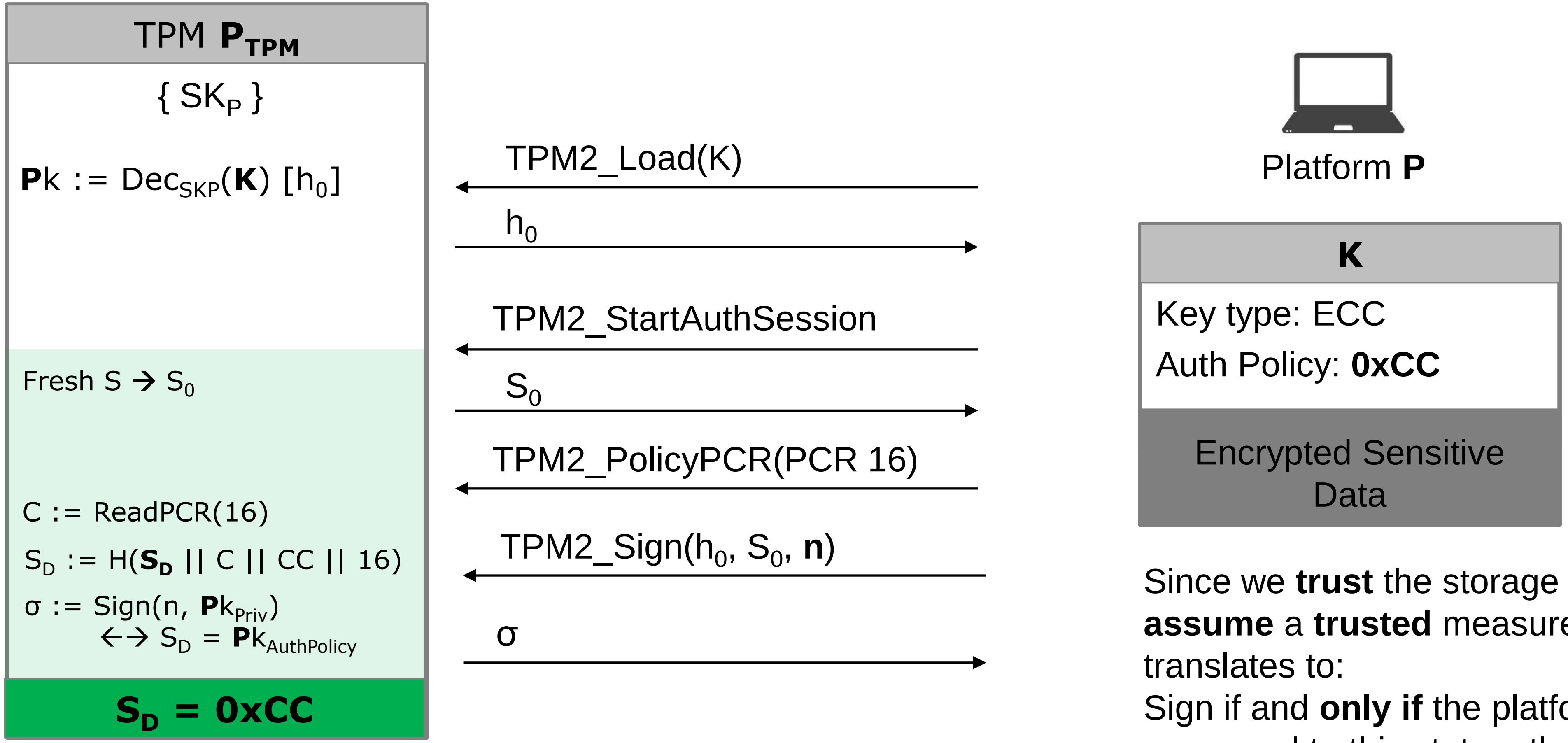
  Can be AND’ed and OR’ed together → MFA!

  Don’t even need no TPM to do → Simple Hash function!

- Works with **sessions**
  - TRIAL sessions are used to build digests (but can be done without)
  - POLICY sessions are used to satisfy policies
  - Each policy command extends the session digest
  - In TRIAL sessions the digest can be retrieved by TPM2_GetPolicyDigest
  - In POLICY sessions the digest is compared to a keys policy auth value before execution

Extended Authorization

Signing the nounce from the Network Manager



TPM $\mathbf{P_{TPM}}$

{ SK_P }

$\mathbf{P}k := Dec_{SKP}(\mathbf{K})\ [h_0]$

Fresh S → S_0

C := ReadPCR(16)

$S_D := H(\mathbf{S_D} \,||\, C \,||\, CC \,||\, 16)$

$\sigma := Sign(n, \mathbf{P}k_{Priv})$
$\leftrightarrow S_D = \mathbf{P}k_{AuthPolicy}$

$\mathbf{S_D = 0xCC}$

TPM2_Load(K)

h_0

TPM2_StartAuthSession

S_0

TPM2_PolicyPCR(PCR 16)

TPM2_Sign(h_0, S_0, **n**)

σ

Platform **P**

K

Key type: ECC

Auth Policy: **0xCC**

Encrypted Sensitive Data

Since we **trust** the storage (PCR) of the TPM and **assume** a **trusted** measurement agent, this translates to:
Sign if and **only if** the platform has been measured to this state, otherwise do nothing.
→ Implicit attests to state of the platform!

When to use a TPM?

- Does it make sense?
- Your CPU is a way better cryptoprocessor
  - If it’s not sensitive operations, it might not be worth the effort.
- Do we have strict timing requirements?
  - TPMs are slow
  - On the other hand, if you don’t have strict timing requirements: why not?
- Define your trust, is it worth the hard work?
  - If you already have trust in the user and platform, then why enhance it? (An adversary is also a user!)

Task of the day

Let’s implement the usecase we talked about: **Trusted Network Management**

If you haven’t done it, you’ll need to install IBM Software TPM and IBM TSS.

Exercises and code can be found here:
https://github.com/benlarsendk/TheCampTPM/

Good luck and remember: **have fun!**

(We’ll do a 15min break, and then we’ll take a quick look at the code before you start yourself)

DTU