TPM Transport Security

Project Kirkland:
Defeating Active Interposers with DICE

EMPOWERING OPEN.

OCP
GLOBAL
SUMMIT

OCTOBER 18-20, 2022
SAN JOSE, CA

TPM Transport Security

Defeating Active Interposers with DICE

SECURITY

Ahmad Abdullateef, Principal Software Architect, Microsoft

Jeff Andersen, Staff Software Engineer, Google

Jordan Hand, Software Engineer, Google

OPEN PLATINUM™

A.K.A. Project Kirkland

Snapshot of Kirkland Demo
Clear Link between TPM and CPU
SPI Bus Analyzer
OS Command Line
SPI bus analyzer and OS command line show the same value for PCR bank Sha256:0 as both of them are in clear text
SPI bus analyzer and OS command line show the different values for PCR bank Sha256:0; As SPI bus analyzer data is encrypted, while OS command line is in clear text.
Encrypted Link between TPM and CPU with Project Kirkland
SPI Bus Analyzer
OS Command Line
intel
OCP GLOBAL SUMMIT
OCTOBER 18-20, 2022 SAN JOSE, CA
EMPOWERING OPE

Agenda

TPM provides powerful attestation primitives

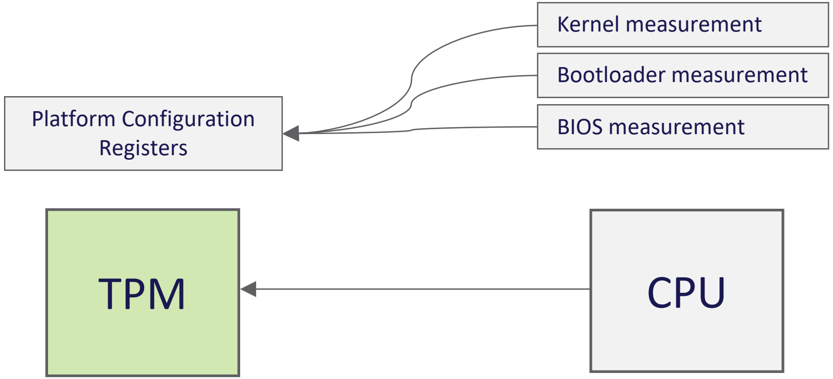
These primitives can be badly abused by active interposers

We need a datacenter-friendly solution

SPDM and DICE to the rescue

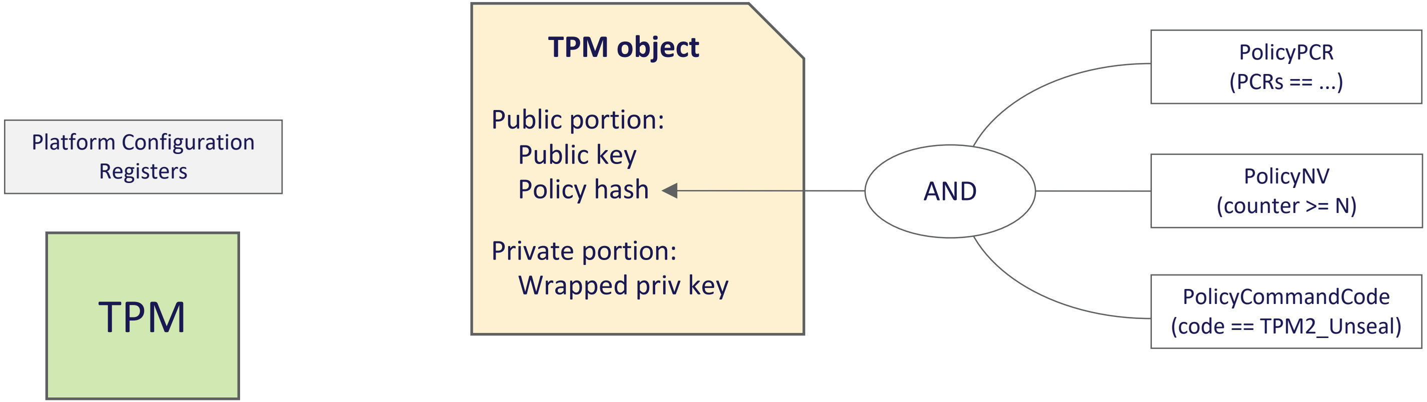
Background: TPM measurement via PCRs

- Host measures each boot layer it runs
- Host pushes measurements to the TPM as PCR extensions
- PCRs reflect the host's boot configuration



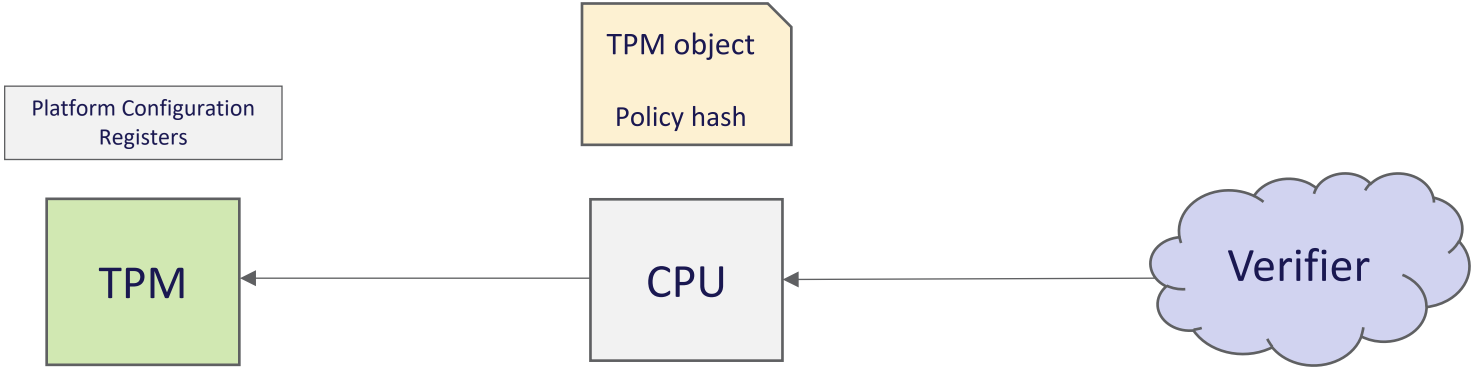
Background: TPM policies

- TPM objects can be gated by policies that the caller must satisfy
- TPM policies support various assertion types, with arbitrary logical grouping



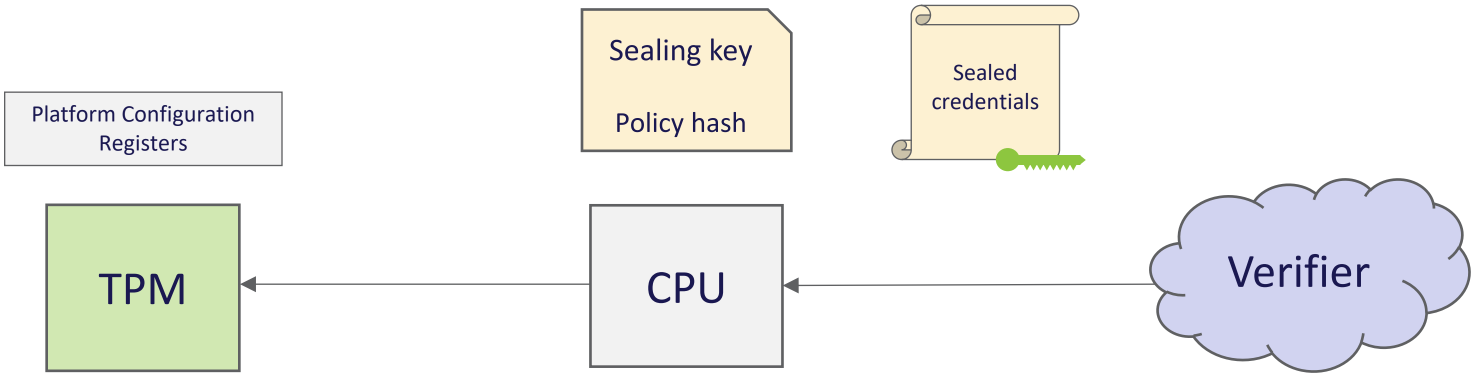
Background: TPM attestation via policies

- Host creates a TPM object with an attached policy (e.g. PolicyPCR)
- Verifier evaluates the object's policy hash
- Verifier confers privileges on the host, contingent on its satisfying that policy



Background: TPM attestation via policies

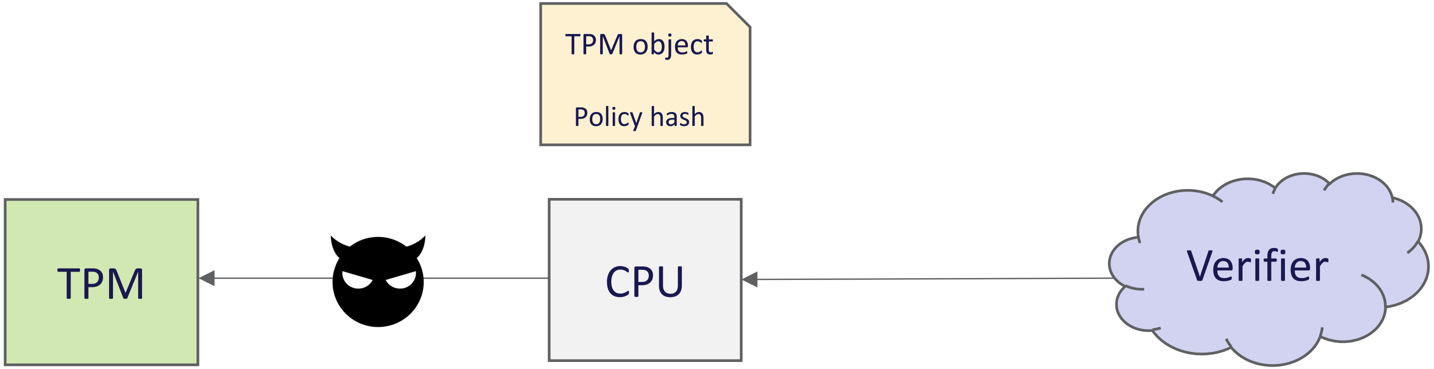
- Host creates a TPM object with an attached policy (e.g. PolicyPCR)
- Verifier evaluates the object's policy hash
- Verifier confers privileges on the host, contingent on its satisfying that policy



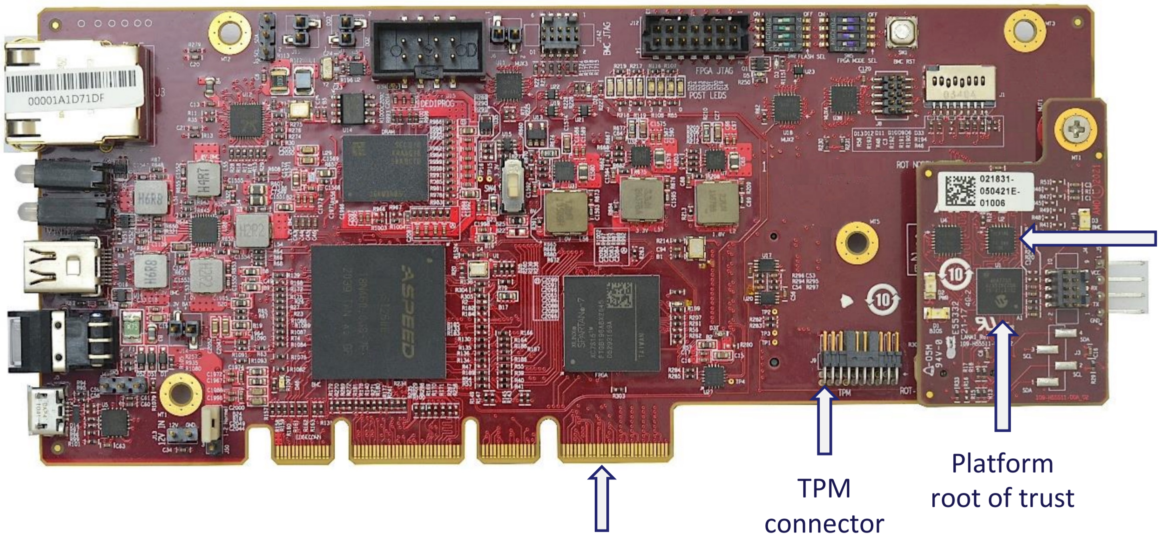
See also https://youtu.be/z0Joifl7JS0

Threat: interposers

- Passive traffic monitoring, e.g. snoop the TPM2_Unseal response
- Suppress / modify TPM commands, e.g. drop PCR extensions
- Inject arbitrary TPM commands
- Physically steal the TPM



The interposability of DC-SCM



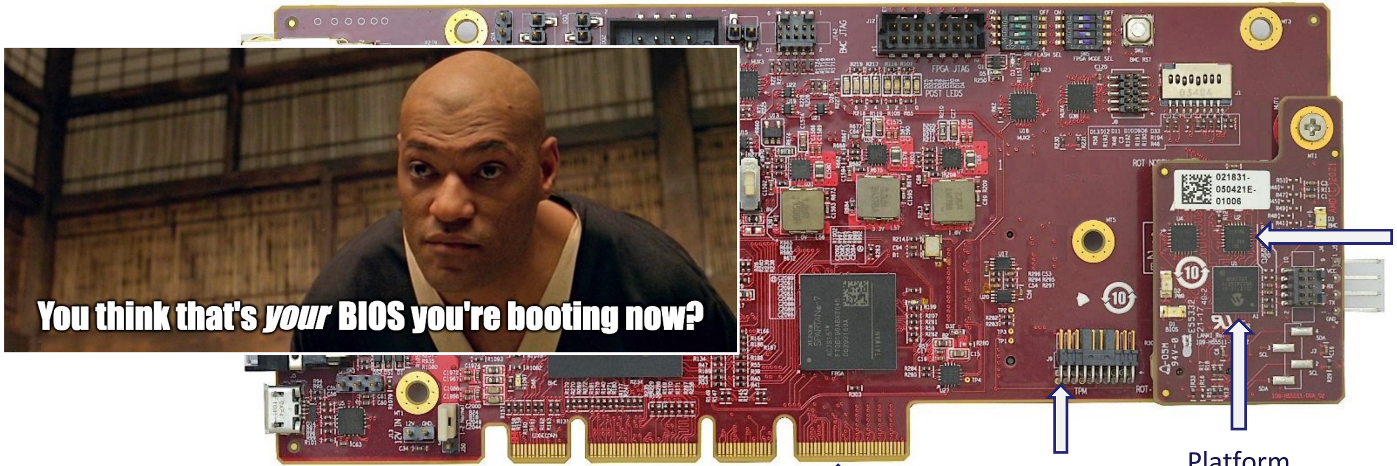
BIOS SPI chip

Platform root of trust

TPM connector

Well-documented physical interface

The interposability of DC-SCM



BIOS SPI chip

TPM connector

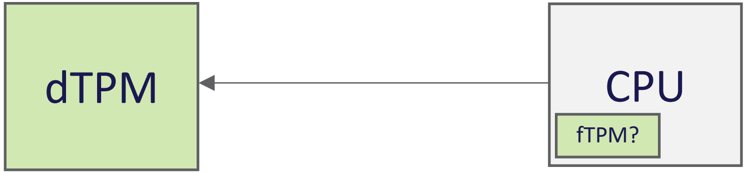
Platform root of trust

Well-documented physical interface

Why not firmware TPMs?

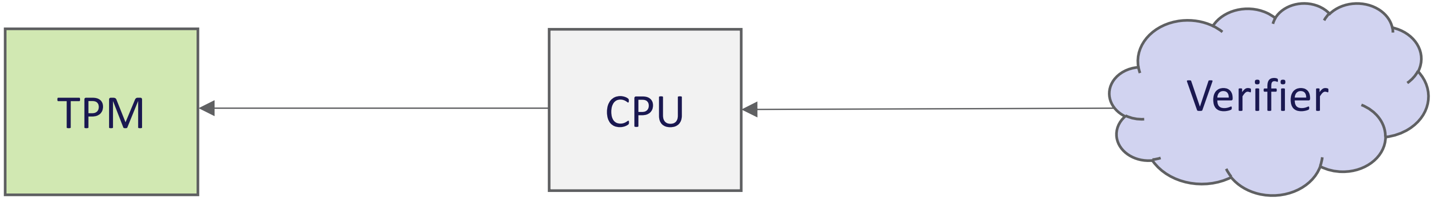
- Certification: it is far easier to Common Criteria certify discrete TPMs

- Implementation: TPMs need secure wear-resistant rewritable storage



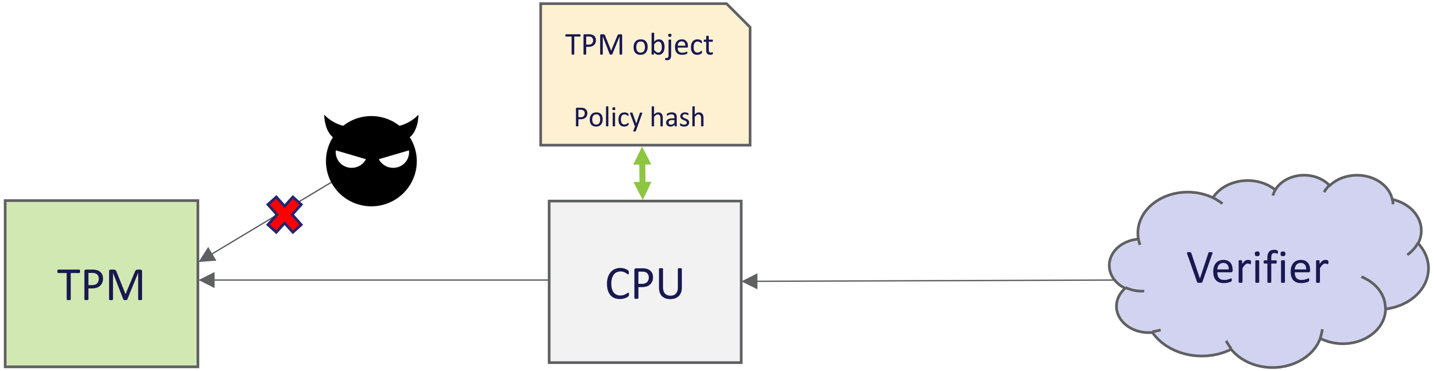
A datacenter-friendly solution

- Supports remote verification
- Supports intentional TPM part swaps
- Minimal disruption to existing TPM client logic



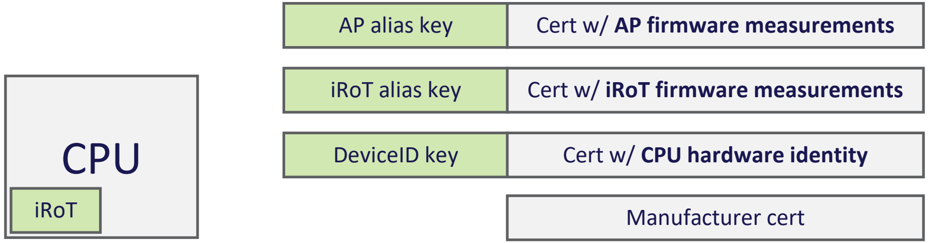
Approach: bind TPM objects to the CPU

- TPM can enforce that an object will only be usable by the intended CPU
- TPM can prove to a verifier that it will enforce the object-CPU pairing



How: CPU integrated roots of trust

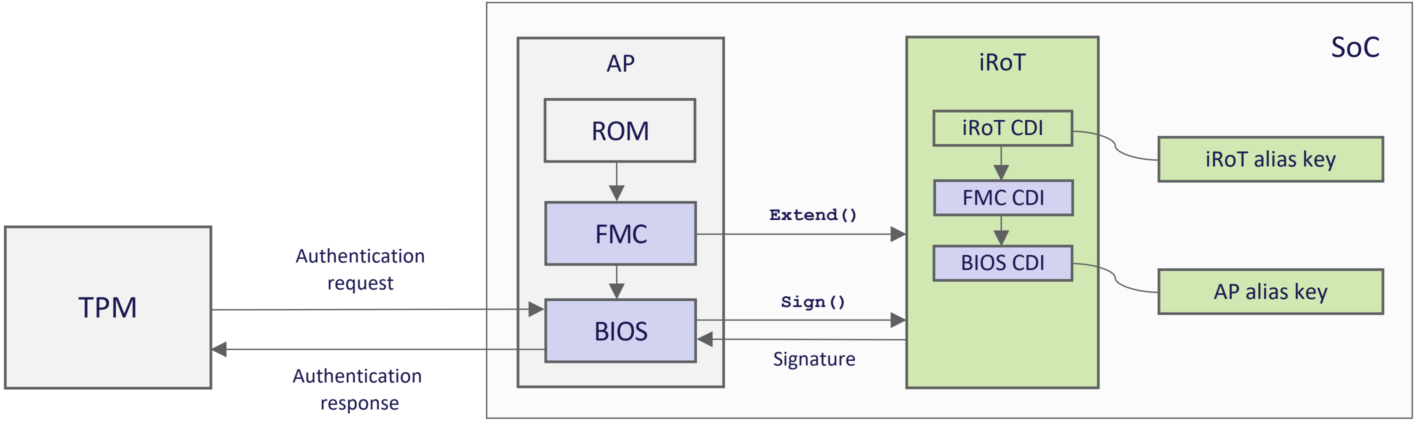
- CPU iRoT must have a cryptographic identity endorsed by the CPU vendor
- CPU iRoT must measure first-mutable-code that runs on the CPU
- **CPU iRoT must mint a DICE alias key for the host**



AP alias key
Cert w/ **AP firmware measurements**
iRoT alias key
Cert w/ **iRoT firmware measurements**
CPU
DeviceID key
Cert w/ **CPU hardware identity**
iRoT
Manufacturer cert

iRoT API: DICE Protection Environment (DPE)

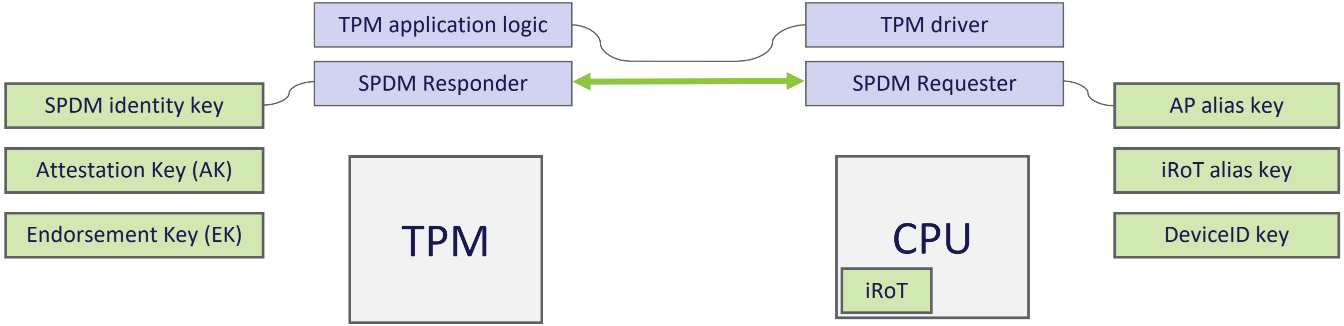
- Defined by TCG; allows one entity to defer its DICE key management to another
- DPE exposes primitives for managing DICE secrets (extend, sign, revoke)
- Clean separation: TPM logic in AP; DICE keys in iRoT



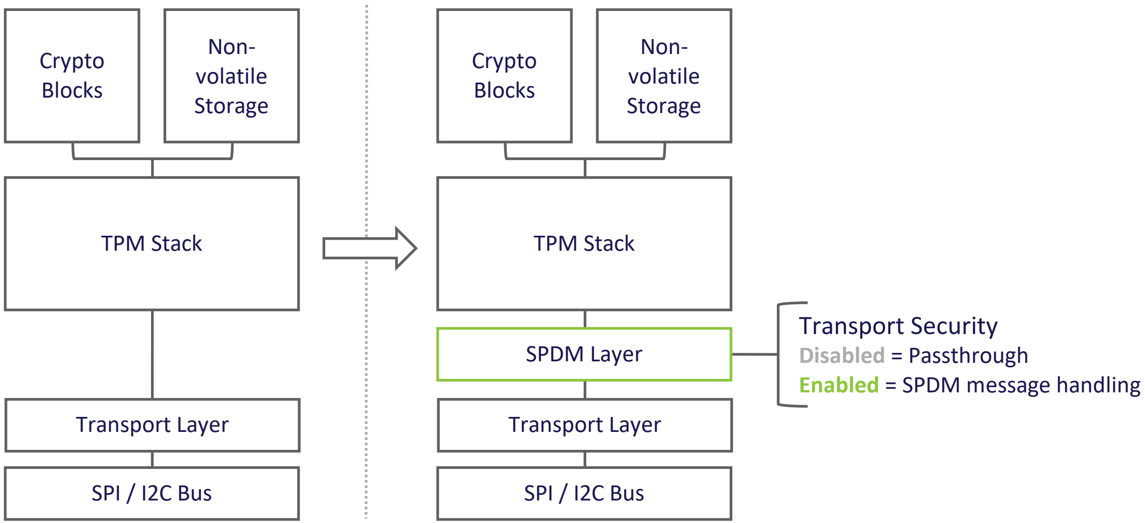
AP = application processor
FMC = first mutable code
CDI = Compound Device Identifier

End-to-end protected TPM channel

- CPU establishes a secure SPDM session with the TPM
- CPU wields the AP DICE alias key to sign the SPDM session handshake
- TPM commands are transparently tunneled over the SPDM session



TPM stack changes



Transport Security
Disabled = Passthrough
Enabled = SPDM message handling

Current

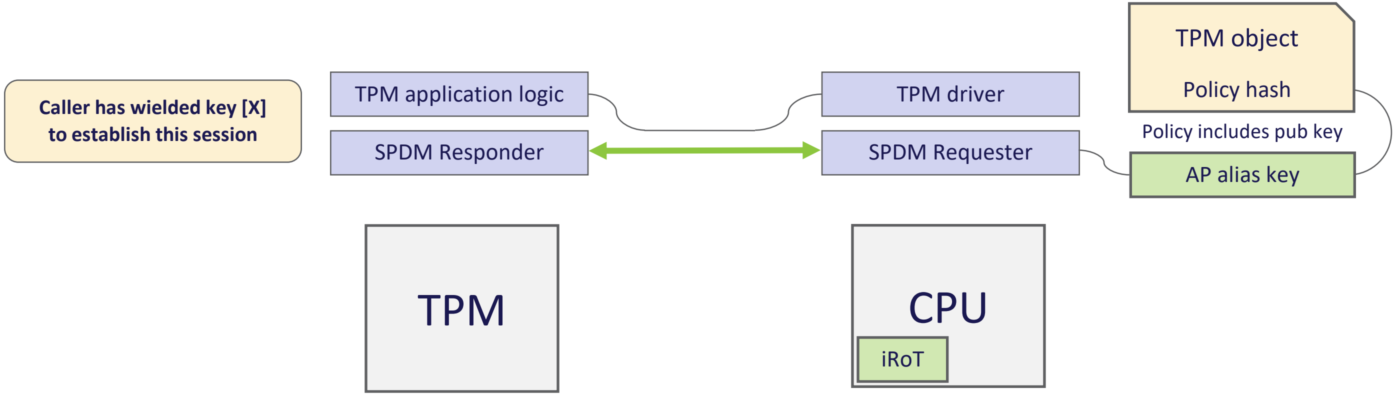
Proposal

Minimal subset of SPDM commands
Only those needed for secure sessions

- GET_VERSION
- GET_CAPABILITIES
- NEGOTIATE_ALGORITHMS
- KEY_EXCHANGE
- FINISH
- END_SESSION

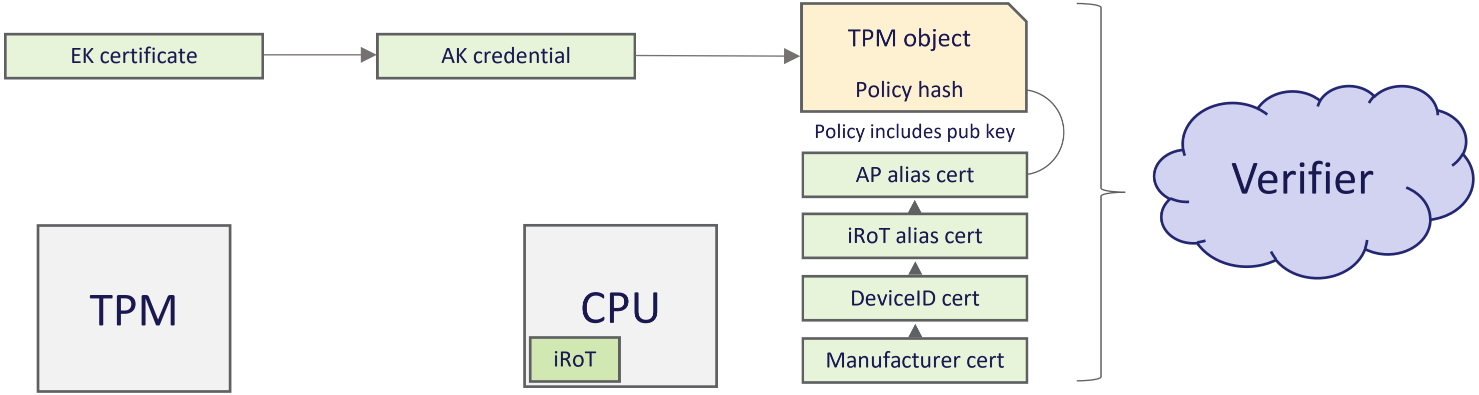
Policy enforcement of caller's SPDM key

- TPM can support a new policy assertion tied to the SPDM channel
- PolicyTransportSecurity(X) only succeeds if the caller used key X to set up the channel



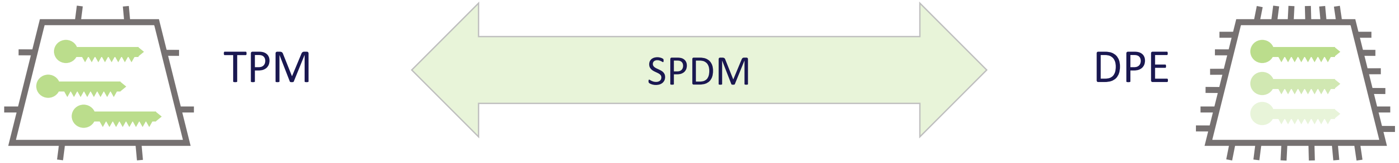
Providing evidence to a verifier

- "I'm convinced this object was made by a legit TPM"
- "I'm convinced the TPM will only allow this object to be used via SPDM pub key X"
- "I'm convinced pub key X is owned by a legit CPU running legit code"



Demo!

Summary



This standards-based flow provides strong defense against interposer attacks

Call to Action

- Standardize TPM-over-SPDM bindings
    - Join the conversation in TCG!

- Develop CPU iRoTs that support DPE
    - See Caliptra, an open iRoT specification

TRUSTED® COMPUTING GROUP

Caliptra

Thank you!

EMPOWERING OPEN.

OCP GLOBAL SUMMIT

OCTOBER 18-20, 2022
SAN JOSE, CA