Building Trust

Use Cases and Implementation of TPM 2.0 in Embedded Linux Systems

inovex

Hello,

I’m Anna-Lena Marx

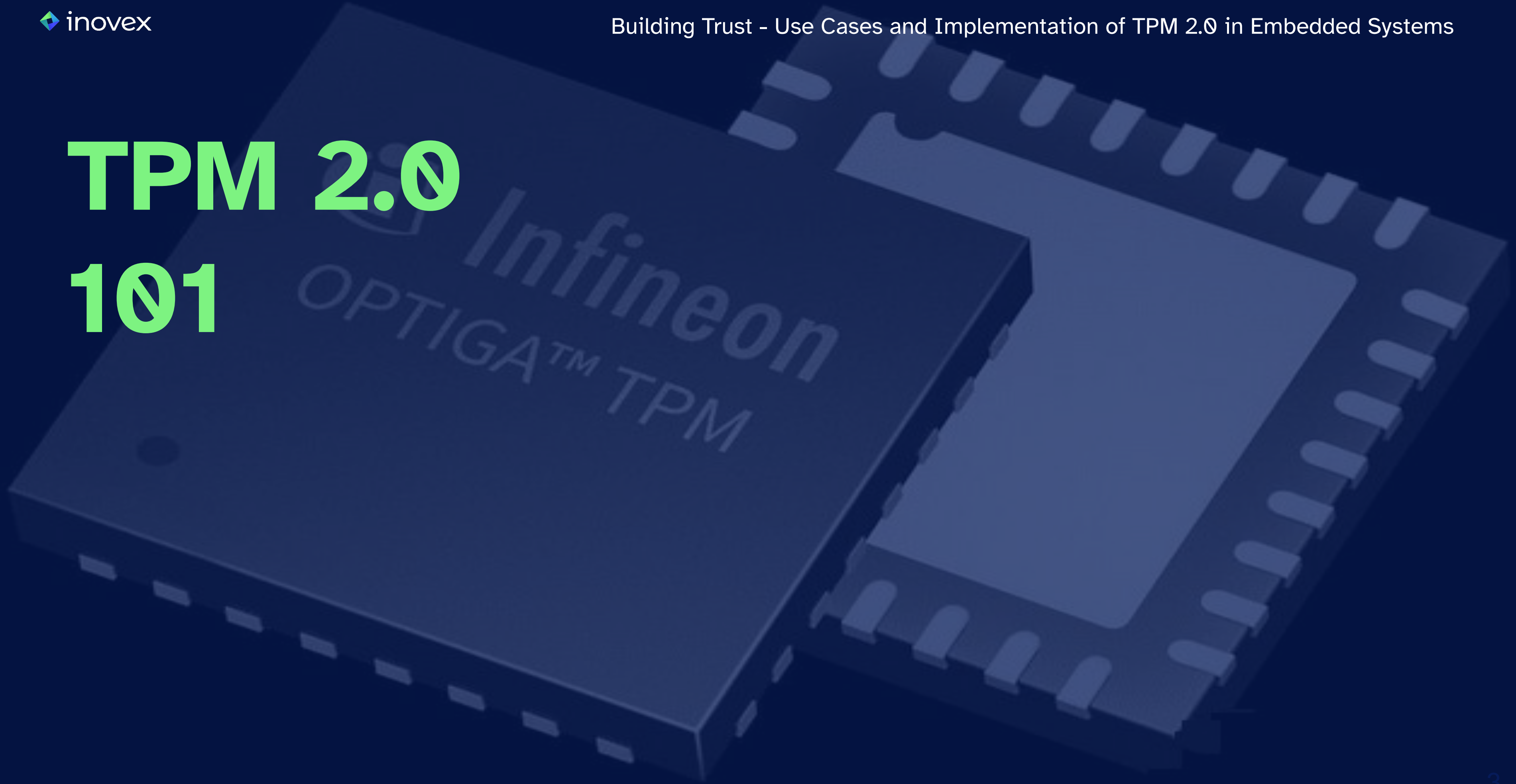
Senior Embedded Systems Engineer, Karlsruhe (Germany)

- with inovex since 2015
- has a Master’s degree in Embedded Systems
- studies Electrical Engineering as a hobby

anna-lena.marx@inovex.de
+49 1523 / 31 81 26 0
@anna-lena-marx-embedded
@Allegra42

#embeddedsystems #yocto #linux #kernel #zephyr #aosp

TPM 2.0 101



What’s a TPM?

Trusted Platform Module

specified by the **Trusted Computing Group** (TCG)
-> ISO/IEC 11889:2015

What’s a TPM?

- often referred to as
  - a cryptographic co-processor
  - a **hardware** security module (HSM)
  - a **hardware** root of trust (HRoT)

TRUST ELEMENT	SECURITY LEVEL	SECURITY FEATURES	TYPICAL APPLICATION
DISCRETE TPM	HIGHEST	TAMPER RESISTANT HARDWARE	CRITICAL SYSTEMS
INTEGRATED TPM	HIGHER	HARDWARE	GATEWAYS
FIRMWARE TPM	HIGH	TEE	ENTERTAINMENT SYSTEMS
SOFTWARE TPM	NA	NA	TESTING & PROTOTYPING
VIRTUAL TPM	HIGH	HYPERVISOR	CLOUD ENVIRONMENT

e.g. Intel Platform Trust Technology (PTT)

e.g. ARM TrustZone Application, used in Smartphones

https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf

What's a TPM?

TPM 2.0

Cryptographic Co-Processor

- Random Number Generator (RNG)
- Key Generator
- Hash-Functions and Hash Based Message Authentication Code (HMAC)
- Algorithms
  SHA-1, SHA-256, SHA-384, RSA, ECC, AES, SM4, HMAC, XOR

Permanent Memory

- Seeds
- Templates
- Persistent key slots
- Platform Configuration Register (PCR)
- Counter and indexes (NVRAM)

Endorsement Seed/Key | Platform Seed/Key | Owner Seed/Key

Volatile Memory

- Authorization sessions
- Active keys

NULL Key (random seed)

TPM Capabilities

```
$ sudo tpm2_getcap -l
- algorithms
- commands
- pcrs
- properties-fixed
- properties-variable
- ecc-curves
- handles-transient
- handles-persistent
- handles-permanent
- handles-pcr
- handles-nv-index
- handles-loaded-session
- handles-saved-session
- vendor
```

```
$ sudo tpm2_getcap algorithms
rsa:
  value:      0x1
  asymmetric: 1
  symmetric:  0
  hash:       0
  object:     1
  reserved:   0x0
  signing:    0
  encrypting: 0
  method:     0
sha1:
  value:      0x4
  asymmetric: 0
  symmetric:  0
  ...
```

```
$ sudo tpm2_getcap ecc-curves
TPM2_ECC_NIST_P256: 0x3
TPM2_ECC_NIST_P384: 0x4
```

tpm2_getcap is part of https://github.com/tpm2-software/tpm2-tools

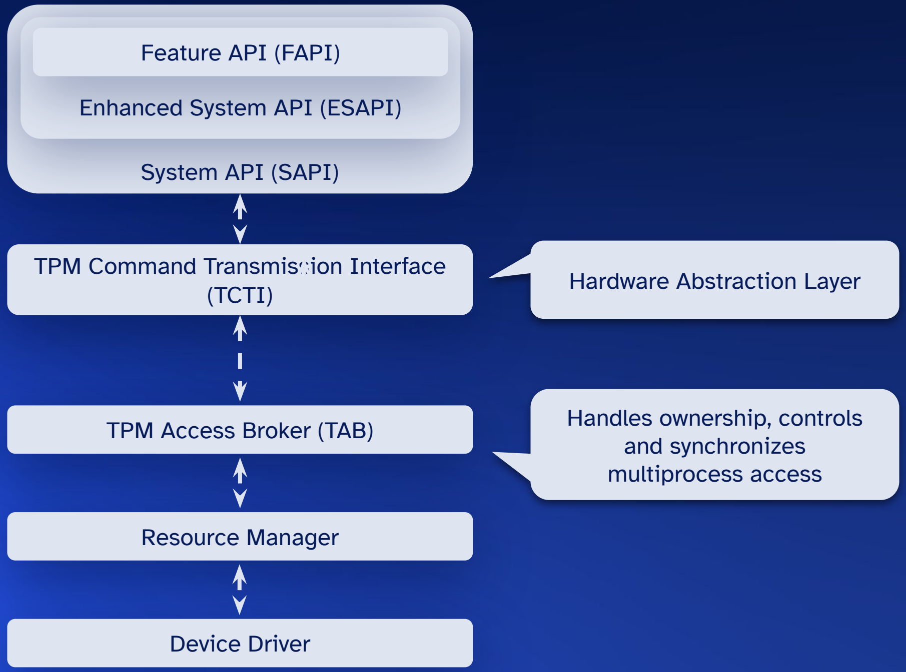
TPM 2.0 Software Implementations

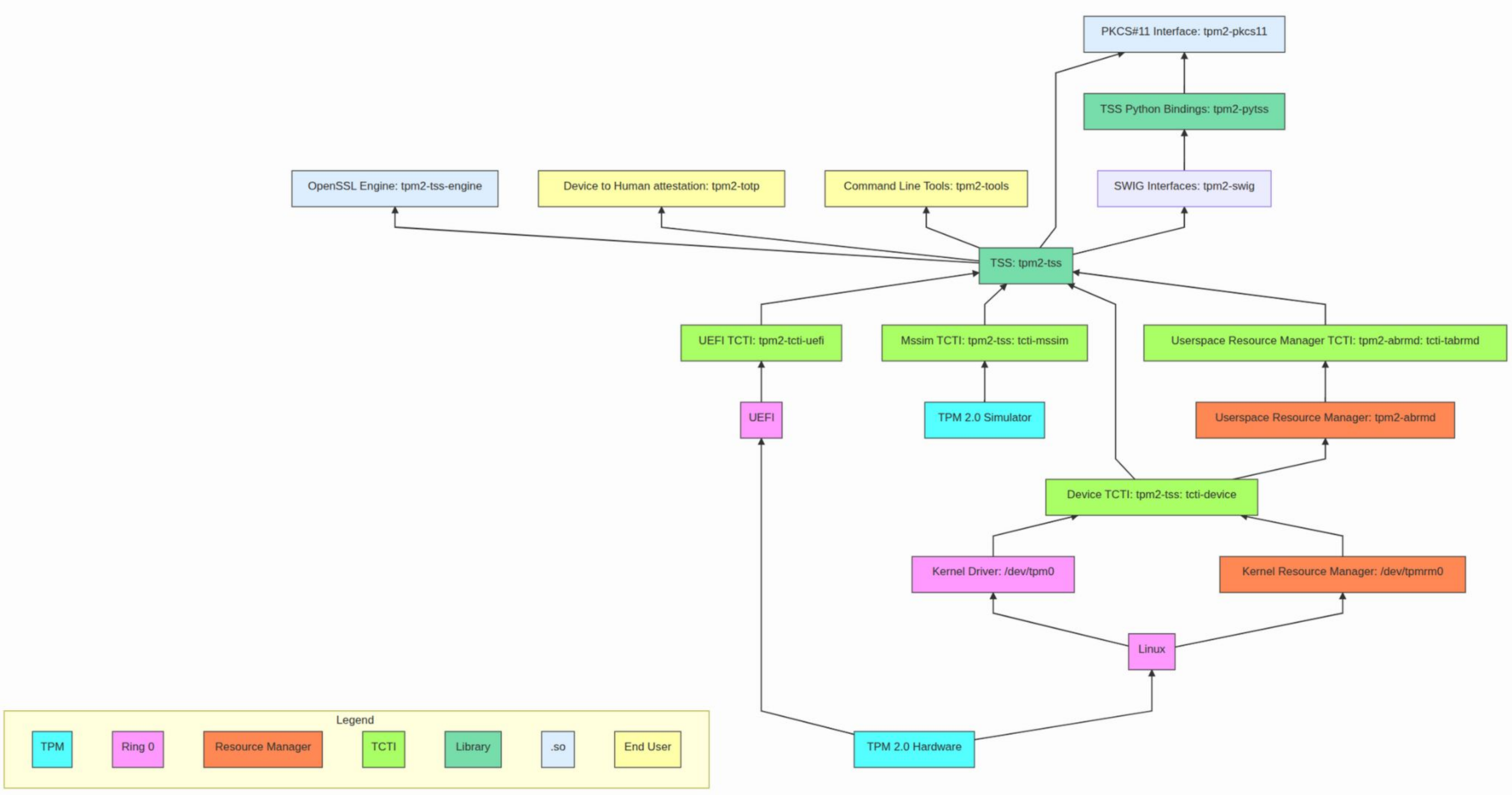
Also specified by TCG

- TPM 2.0 Library Specification
- TPM Software Stack (TSS)

Several implementations available:

- **tpm2-tss**
- wolfTPM
- Official TPM 2.0 Reference Implementation
- TSS.MSR
- ...



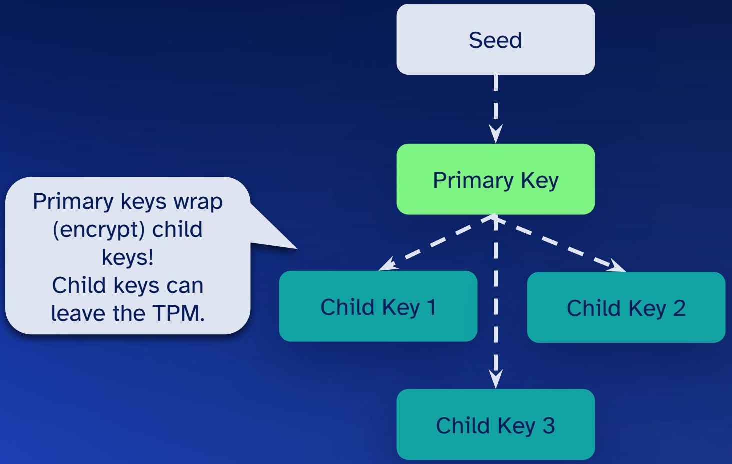


https://tpm2-software.github.io/

TPM 2.0 Terms
Keys

- Asymmetric keys with several algorithms
  - Create primary keys (hardware based RNG)
  - Load (import) external keys (public and private)
  - Encrypt / decrypt
  - Signing / verify signatures
  - Attestation tasks

- Private keys of primary key pairs never leave the TPM!
  - Primary keys are regenerated from the seed
  - Child keys are encrypted by primary keys (key wrapping)
    -> can be stored outside

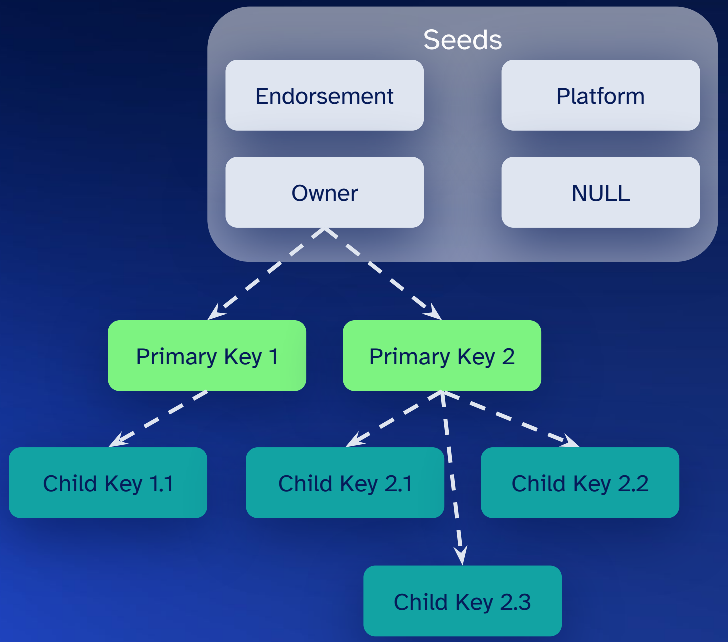


TPM 2.0 Terms
Key Hierarchies

Four hierarchies with different seeds and authorizations

- Seed acts as cryptographical root of a hierarchy (Never leaves the TPM!)
- Each hierarchy has an associated proof value
  - Derived from the seed or independently generated
  - Verify data supplied to the TPM was initially generated by itself (e.g. used for HMAC)

- **Endorsement hierarchy**
  - Controlled by TPM manufacturer / privacy owner
  - Privacy sensitive
  - Used to validate the authenticity of a TPM
- **Platform hierarchy**
  - Controlled by platform manufacturer / OEM who ships and controls the early bootcode e.g. UEFI secure boot
- **Owner hierarchy**
  - For user owned objects
- **NULL hierarchy**
  - Random seed at every power cycle
  - Used for sessions, digest and HMAC state, RNG, ...



TPM 2.0 Terms
Platform Configuration Registers (PCRs)

- 24+ dedicated registers, holding a hash digest

```
sudo tpm2_getcap pcrs
selected-pcrs:
  - sha1: [ ]
  - sha256: [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 ]
  - sha384: [ ]
```

- **Cryptographically measure and record software states**

- Content can only be cleared under strict conditions
  - Most PCRs can only be reset at system boot / power on
  - Some can only be modified in a specific TPM locality level

- PCRs are not written but **extended**
  - Preserves existing information
  - **The order of extend operations influences the resulting end digest!**
  - PCR_n = HASH(PCR_{n-1} | NewInput)

TPM 2.0 Terms
Platform Configuration Registers (PCRs)

PCR#	Used by	From Location	Measured Objects	Log
0	Firmware 💻	UEFI Boot Component	Core system firmware executable code	UEFI TPM event log
1	Firmware 💻	UEFI Boot Component	Core system firmware data/host platform configuration; typically contains serial and model numbers	UEFI TPM event log
2	Firmware 💻	UEFI Boot Component	Extended or pluggable executable code; includes option ROMs on pluggable hardware	UEFI TPM event log
3	Firmware 💻	UEFI Boot Component	Extended or pluggable firmware data; includes information about pluggable hardware	UEFI TPM event log
4	Firmware 💻	UEFI Boot Component	Boot loader and additional drivers; binaries and extensions loaded by the boot loader	UEFI TPM event log
5	Firmware 💻	UEFI Boot Component	GPT/Partition table	UEFI TPM event log
7	Firmware 💻	UEFI Boot Component	SecureBoot state	UEFI TPM event log
8	`grub` 🥧	UEFI Boot Component	Commands and kernel command line	UE ev
9	`grub` 🥧	UEFI Boot Component	All files read (including kernel image)	UE ev
	Linux kernel 🌰	Kernel	All passed initrds (when the new `LOAD_FILE2` initrd protocol is used)	UE ev
10	IMA 📐	Kernel	Protection of the IMA measurement log	IM lo
11	`systemd-stub` 🚀	UEFI Stub	All components of unified kernel images (UKIs)	UE ev
	`systemd-pcrphase` 🚀	Userspace	Boot phase strings, indicating various milestones of the boot process	Jo no
12	`systemd-stub` 🚀	UEFI Stub	Kernel command line, system credentials and system configuration images	UE ev
13	`systemd-stub` 🚀	UEFI Stub	All system extension images for the initrd	UE ev
14	`shim` 🔑	UEFI Boot Component	"MOK" certificates and hashes	UE ev

https://uapi-group.org/specifications/specs/linux_tpm_pcr_registry/

TPM 2.0 Terms
Platform Configuration Registers (PCRs)

Table 12: Localities Reserved for Platform-Specific Workgroups

Workgroup Defining the locality	Locality value	Description of the locality
PC-Client	00_{16}	The Static RTM, its chain of trust and its environment
	01_{16}	An environment for use by the Dynamic OS
	02_{16}	Dynamically Launched OS (Dynamic OS) "runtime" environment
	03_{16}	Auxiliary components
	04_{16}	Trusted hardware component

RTM
Root of Trust for Measurement

Resets all PCRs at system start
Extends hash in PCR 0-7

Operating system works in locality 4
Can only reset PCR 16 and 23

https://trustedcomputinggroup.org/wp-content/uploads/RegistryOfReservedTPM2HandlesAndLocalities_v1p1_pub.pdf

TPM 2.0 Terms
Platform Configuration Registers (PCRs)

```
$ systemd-analyze pcrs
NR NAME                 SHA256
 0 platform-code        0a2310be6f1890d486e54be738ace4e4ca6e5a42d8a340584
 1 platform-config      5e006c735eadd81dda0c3e54c18da347b1c4b2dbd937ce433
 2 external-code        3b617409aac07541af09a3dc8fa2adde0c1ffcc1416946dcc
 3 external-config      3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7
 4 boot-loader-code     f6bf3e0ef200d74c6e2aee8d2c45fc6c78937ffcc6f0f73fb
 5 boot-loader-config   e0c17ac5f024013317eec9a029e23c127c7fa8ec6e33ea5b3
 6 host-platform        3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7
 7 secure-boot-policy   51f40d32ebe8d10b28aecf9839f9f28bb2ca122d2f394a742
 8 -                    0000000000000000000000000000000000000000000000000
 9 kernel-initrd        f4ada4804f6a426a1f69c5b602e72b32dd449fd2204d4ca8b
10 ima                  0000000000000000000000000000000000000000000000000
11 kernel-boot          0000000000000000000000000000000000000000000000000
12 kernel-config        762d9730d8a38ff63d3ebfc3029d18c8bacc683edd08a7049
13 sysexts              0000000000000000000000000000000000000000000000000
14 shim-policy          0000000000000000000000000000000000000000000000000
15 system-identity      0000000000000000000000000000000000000000000000000
16 debug                0000000000000000000000000000000000000000000000000
17 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
18 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
19 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
20 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
21 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
22 -                    fffffffffffffffffffffffffffffffffffffffffffffffff
23 application-support  0000000000000000000000000000000000000000000000000
```

```
$ sudo tpm2_pcrread
  sha1:
  sha256:
    0 : 0x0A2310BE6F1890D486E54BE738ACE4E4CA6E5A42D8A3405840FF1F3956359500
    1 : 0x5E006C735EADD81DDA0C3E54C18DA347B1C4B2DBD937CE4338EDAF7B210E56E1
    2 : 0x3B617409AAC07541AF09A3DC8FA2ADDE0C1FFCC1416946DCC850FAC0439B1864
    3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    4 : 0xF6BF3E0EF200D74C6E2AEE8D2C45FC6C78937FFCC6F0F73FB41559F0B7A13E44
    5 : 0xE0C17AC5F024013317EEC9A029E23C127C7FA8EC6E33EA5B3EAAB8FF72FC9C45
    6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    7 : 0x51F40D32EBE8D10B28AECF9839F9F28BB2CA122D2F394A742369B48F5DA70CCA
    8 : 0x0000000000000000000000000000000000000000000000000000000000000000
    9 : 0xF4ADA4804F6A426A1F69C5B602E72B32DD449FD2204D4CA8BF950ABCEDA27831
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x0000000000000000000000000000000000000000000000000000000000000000
    12: 0x762D9730D8A38FF63D3EBFC3029D18C8BACC683EDD08A7049BF8AEC6C9A3265A
    13: 0x0000000000000000000000000000000000000000000000000000000000000000
    14: 0x0000000000000000000000000000000000000000000000000000000000000000
    15: 0x0000000000000000000000000000000000000000000000000000000000000000
    16: 0x0000000000000000000000000000000000000000000000000000000000000000
    17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    23: 0x0000000000000000000000000000000000000000000000000000000000000000
  sha384:
```

TPM 2.0

Cryptographic Co-Processor

- Random Number Generator (RNG)
- Key Generator
- Hash-Functions and Hash Based Message Authentication Code (HMAC)
- Algorithms
  SHA-1, SHA-256, SHA-384, RSA, ECC, AES, SM4, HMAC, XOR

Permanent Memory

- Seeds
- Templates
- Persistent key slots
- Platform Configuration Register (PCR)
- Counter and indexes (NVRAM)

Endorsement Seed/Key

Platform Seed/Key

Owner Seed/Key

Volatile Memory

- Authorization sessions
- Active keys

NULL Key (random seed)

How can we trust a TPM?

TPM manufacturer

TPM manufacturing process

System / Platform manufacturing process

Bootcode / Secure Boot provisioning

Building up a root of trust is rather about processes than a cryptographical issue.

How can we trust a TPM?
Verify a TPM's authenticity using the Endorsement Hierarchy

- The Endorsement Hierarchy is controlled by the TPM manufacturer
- Primary keys in this hierarchy are constrained to an authentic TPM attached to an authentic platform
  - Verify a TPM's authenticity
  - Identify a machine

    Privacy concern: All child keys generated from an EK can be correlated to a single TPM

- Endorsement Key (EK)
  - A primary key generated using the TCG's EK key template (available for RSA and ECC)
  - Seed is fixed, can not be cloned -> EK can be regenerated
- EK certificate
  - Pre Installed during manufacturing

How can we trust a TPM? Verify a TPM’s authenticity using the Endorsement Key

Check authenticity

1. Endorsement hierarchy on TPM
    a. Generate endorsement key pair `tpm2_createek`
    b. Read public key `tpm2_readpublic`
    c. Obtain EK certificate `tpm2_getekcertificate`

2. Obtain vendor intermediate CA for your TPM (website)

3. Verify
    a. EK public key (1. b) matches the one in the EK certificate (1. c)
    ```
    openssl rsa -pubin -in <ek-public-key>  -text -noout
    x509 -in <ek-certificate> -inform DER -noout -text
    ```
    b. EK certificate (1. c) is valid with the root/intermediate/both CA from the vendor (2.)
    ```
    openssl verify -CAfile <vendor-root-ca> -untrusted <vendor-intermediate-ca> <ek-certificate>
    openssl verify -verbose -CAfile <vendor-intermediate-ca> <ek-certificate>
    ```

TPM 2.0
in Practice with Yocto

Establish a Chain of Trust

Secure Boot on x86 Platforms with meta-secure-core/meta-efi-secure-boot and Mender

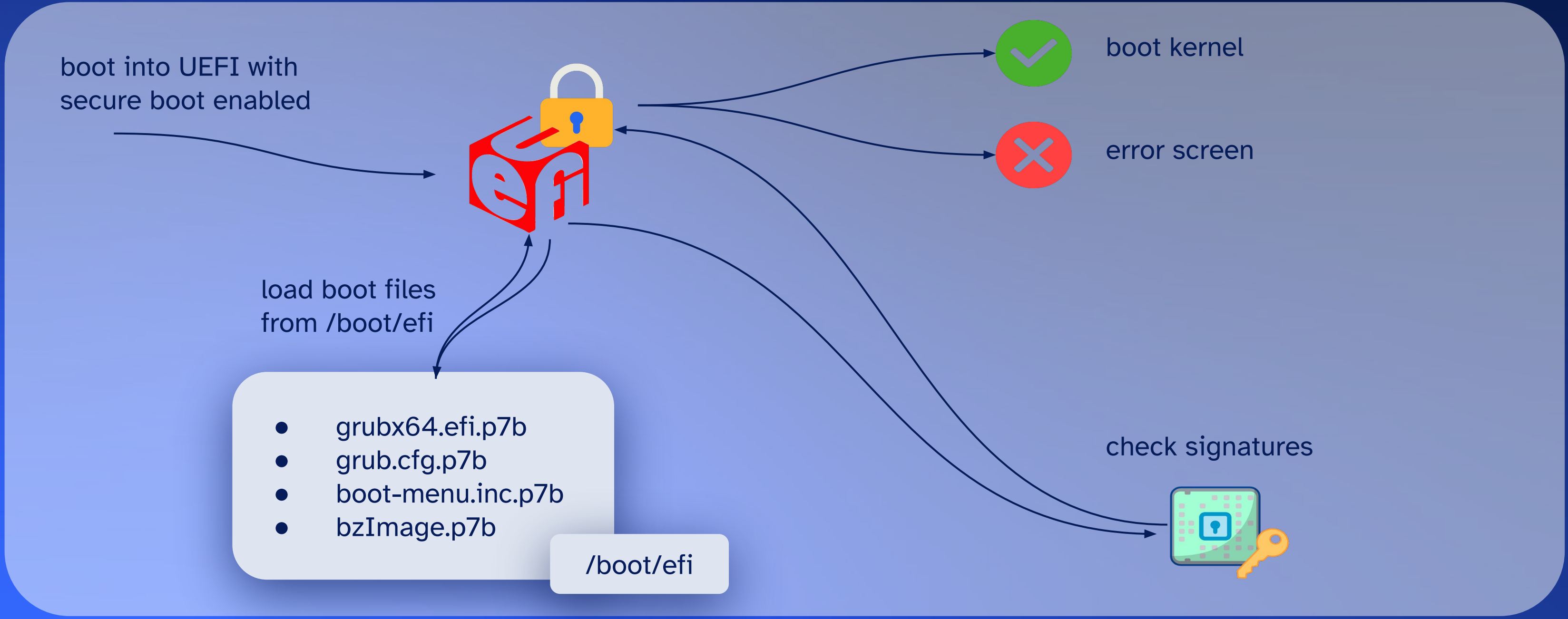
- Ensure a device runs a with a trusted, signed and untampered
  - bootloader
  - kernel
  - and similar binary blobs

- Establish a trusted environment for later (critical) applications and tasks
  - disabling secure boot can be detected during runtime

ARM devices may use some special memory areas for storing secure boot / verified boot certificates that can only be written once.

Establish a Chain of Trust

Secure Boot on x86 Platforms with meta-secure-core/meta-efi-secure-boot and Mender



Establish a Chain of Trust

Secure Boot on x86 Platforms with meta-secure-core/meta-efi-secure-boot and Mender

- Last tested with kirkstone

- Uses meta-secure-core/meta-efi-secure-boot
  - grub-efi, efi-tools, mokutils, shim, ...
  - Patches for GRUB
  - Signing tasks for kernel and bootloader
  - Utilizes TPM for keys and certificates
  - Certificate provisioning step automated during bootup

- Most documentation describes the process with initramfs
- Further hints for Mender with Secure Boot (kirkstone): https://hub.mender.io/t/mender-and-efi-secure-boot-on-intel-corei7-64/4862
- Kernel within the boot partition is not managed by Mender!
  - Use Mender state scripts to update kernels

Establish a Chain of Trust

Secure Boot on x86 Platforms with meta-secure-core/meta-efi-secure-boot and Mender

distro.conf

```
MACHINE_FEATURES:append = " \
    efi \
    tpm2 \
"
DISTRO_FEATURES:append = " \
    security \
    tpm2 \
    efi-secure-boot \
"

MOK_SB = ""
SIGNING_MODEL = "user"
```

image.bb

```
IMAGE_INSTALL:append = " \
    tpm2-tools \
    libtss2-tcti-device \
    efitools \
    seloader \
    update-signed-kernel-state-script \
"

# Deploy kernel and signature to boot partition.
# They cannot be verified on rootfs partition.
#
# Mender processes IMAGE_BOOT_FILES and has some issues with multiline parsing
IMAGE_BOOT_FILES:append = " ${KERNEL_IMAGETYPE} ${KERNEL_IMAGETYPE}${SB_FILE_EXT}"
```

Establish a Chain of Trust

Secure Boot on x86 Platforms with meta-secure-core/meta-efi-secure-boot and Mender

overwritten
90_mender_boot_grub.cfg

```
...

mender_kernel_path=""
if [ "${drop_to_grub_prompt}" = "no" ];
then
    search --no-floppy --label --set=root boot
    if linux "${mender_kernel_path}/${kernel_imagetype}" root="${mender_kernel_root}" ${bootargs};
    then
        if test -n "${initrd_imagetype}" -a test -e "${mender_kernel_path}/${initrd_imagetype}";
        then
            initrd "${mender_kernel_path}/${initrd_imagetype}"
        fi
        maybe_pause "Pausing before booting."
        boot
    fi
    maybe_pause "Pausing after failed boot."
fi
```

Make use of PCRs

Encrypt a userdata partition on first boot and use a TPM 2.0 device as key

Bind the encryption to TPM PCR(s)

- Usually PCR 7 (secure boot state) or 0-7
- Encrypted partition without the need for user interaction (entering a password)
- If PCR(s) change, data can not be unlocked with TPM
  - Stays locked if a device gets tampered
  - Can be restored with backup password

Make use of PCRs

Encrypt a userdata partition on first boot and use a TPM 2.0 device as key

systemd

LUKS
Linux Unified Key Setup

systemd

LUKS
Linux Unified Key Setup

systemd service

- triggers encryption
- runs before **`data.mount`**

encryption script

- write LUKS header on existing partition
  `cryptsetup reencrypt --encrypt`
- actual online encryption
  `cryptsetup reencrypt`

encryption script

- link TPM to LUKS device
  `systemd-cryptenroll`

Make use of PCRs

Encrypt a userdata partition on first boot and use a TPM 2.0 device as key

distro.conf

```
PACKAGECONFIG:append:pn-cryptsetup = " cryptsetup veritysetup udev luks2"
PACKAGECONFIG:append:pn-systemd = " cryptsetup tpm2"
```

systemd_%.bbappend

```
do_install:append() {
    install -d ${D}${libdir}/cryptsetup
    install -m 0755 ${WORKDIR}/build/libcryptsetup-token-systemd-tpm2.so \
  ${D}${libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so
}

FILES:${PN}:append = " \
  ${base_libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \
  ${libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \
"
```

Only needed with kirkstone!

Make use of PCRs

Encrypt a userdata partition on first boot and use a TPM 2.0 device as key

crypttab

```
data UUID=<data-uuid> none tpm2-device-auto,tpm2-pcrs=7
```

At early boot and when the system manager configuration is reloaded, /etc/crypttab is translated into systemd-cryptsetup@.service units by systemd-cryptsetup-generator(8).

fstab

```
- UUID=<data-uuid> /data ext4 rw 0 2
+ /dev/mapper/data /data ext4 rw 0 2
```

Mender also modifies fstab in a ROOTFS_POSTPROCESS_COMMAND

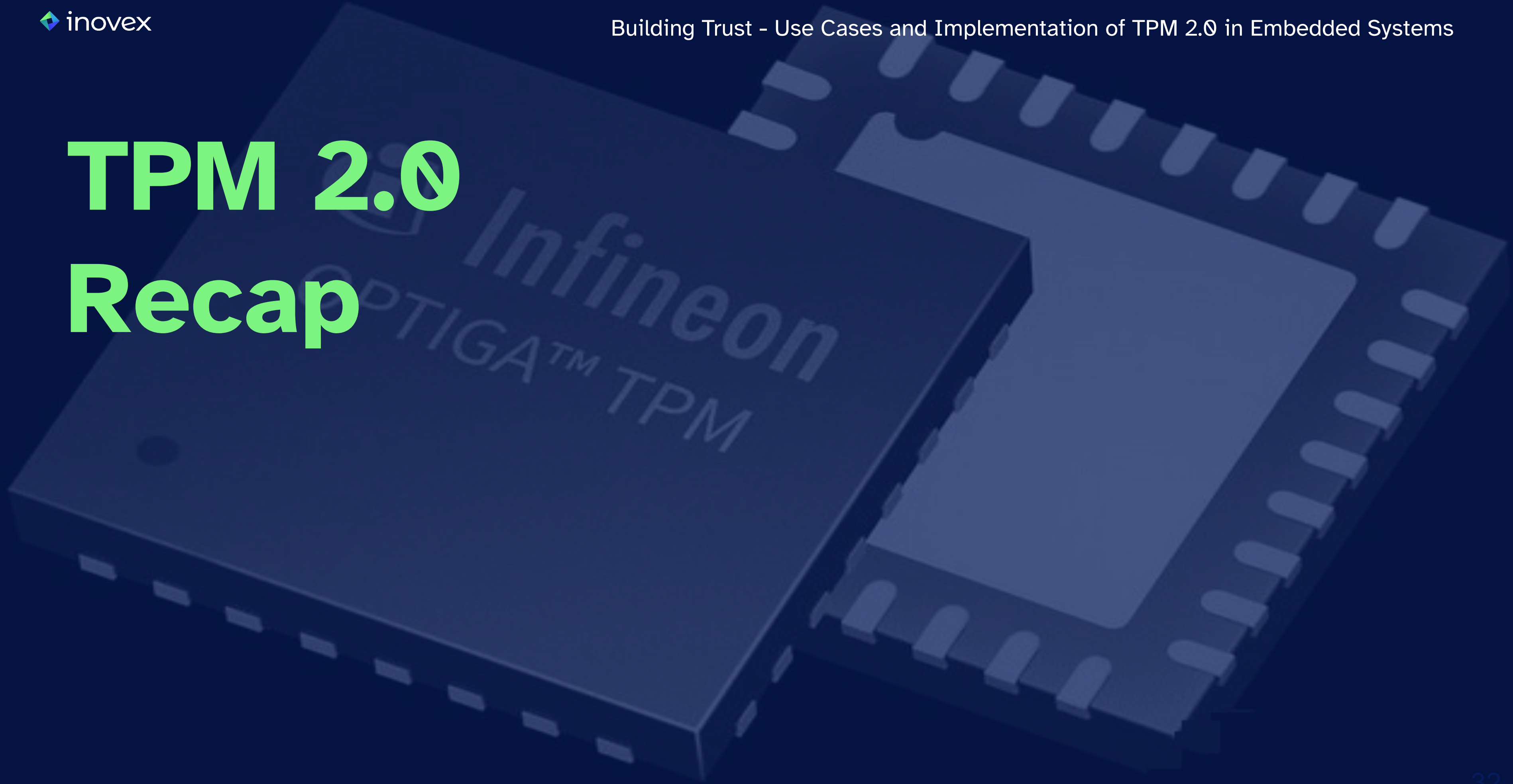
encrypt-data.sh

```
echo "Generating a new key..."
/usr/bin/openssl rand -base64 44 > ${tmp_key_file}


echo "Writing encryption headers..."
/bin/cat "${tmp_key_file}" | /usr/sbin/cryptsetup reencrypt --encrypt \
--type luks2 --key-slot=1 --batch-mode --init-only --reduce-device-size 32M
--offset="${OFFSET}" "${data_dev}" data


# Enrolling the TPM2 integration only works after the online encryption step
is finished.
echo "Encrypting the data partition..."
/bin/cat ${tmp_key_file} | /usr/sbin/cryptsetup reencrypt \
--offset="${OFFSET}" "${data_dev}"


echo "Deploying TPM2 keys..."
/usr/bin/systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 \
--unlock-key-file=${tmp_key_file} "${data_dev}"
```

More ideas

- Utilize the Linux Integrity Measurement Architecture - maybe together with keylime.dev
- Store sensitive counter or read-only information in NVRAM
- Use TPM backed keys for
  - SSH
  - SSL
  - your application
- Or does your stack already utilize a TPM but you did not know?
  - Azure IoT Edge

TPM 2.0 Recap



Recap

Chances and Risks

TPM 2.0 can help trusting devices, e.g. with

- Hardened cryptographic operations and random number generation
- Trusted kernels with secure boot (utilizes the key infrastructure on x86)
- Measure system state and integrity (PCRs)
- Secure, tamper resistant storage for (read-only) information (NVRAM)
- and a lot more

Recap
Chances and Risks

TPM 2.0 devices are affordable, available and well supported

- But integrating one is an active decision in embedded systems
- Just using a TPM is not a no-brainer that makes a device trustworthy
  - Attacks on TPM exist (e.g. I2C/SPI bus sniffing)
  - Who owns and controls the platform?
  - Keys based on the endorsement hierarchy can be correlated to a single TPM
  - Do you trust the TPM vendor, platform manufacturer and the device provisioning?

inovex

Thank you!

Questions?

Anna-Lena Marx

Embedded Systems Engineer

@anna-lena-marx-embedded

@Allegra42

Learn more about TPM 2.0

- https://trustedcomputinggroup.org/
- Course | Trusted Computing 1101: Introductory Trusted Platform Module (TPM) usage | OpenSecurityTraining2
- Course | Trusted Computing 1102: Intermediate Trusted Platform Module (TPM) usage | OpenSecurityTraining2
- https://github.com/tpm2-software/tpm2-tss
- https://tpm2-tools.readthedocs.io/en/latest/
- https://uapi-group.org/specifications/specs/linux_tpm_pcr_registry/
- https://www.freedesktop.org/software/systemd/man/255/systemd-cryptenroll.html
- https://0pointer.net/blog/authenticated-boot-and-disk-encryption-on-linux.html
- https://0pointer.net/blog/brave-new-trusted-boot-world.html
- https://git.yoctoproject.org/meta-security/tree/meta-integrity/README.md?h=scarthgap
  https://github.com/Wind-River/meta-secure-core/tree/scarthgap/meta-integrity
  https://ima-doc.readthedocs.io/en/latest/ima-concepts.html
- https://github.com/tpm2-software/tpm2-pkcs11
  https://github.com/tpm2-software/tpm2-pkcs11/blob/master/docs/SSH.md
- https://github.com/tpm2-software/tpm2-openssl

Utilize NVRAM storage

For additional device identity, persistent and tamper resistant counters, …

Establish a session

```
tpm2_startauthsession -S session.dat
tpm2_policycommandcode -S session.dat TPM2_CC_NV_Write
tpm2_policynvwritten -S session.dat -L nvwrite.policy c
tpm2_flushcontext session.dat
```

Not a typo!
c -> clear state

Define a NV space

```
tpm2_nvdefine -s 20 -a "ownerread|authread|policywrite" -L nvwrite.policy
    Output: nv-index: 0x1000000

tpm2_nvreadpublic
0x1000000:
  name: 000bbf41e8a6f49fbbdcf194a41a916126ab8ccbba68bb17ad4058ef3623b3d41bfb
  hash algorithm:
    friendly: sha256
    value: 0xB
  attributes:
    friendly: policywrite|ownerread|authread
    value: 0x60008
  size: 20 …
```

Write to NV space

```
# Establish the policy session
tpm2_startauthsession -S session.dat --policy-session
tpm2_policycommandcode -S session.dat TPM2_CC_NV_Write
tpm2_policynvwritten -S session.dat c

# Write content
tpm2_nvwrite 0x1000000 -i testWrite.txt -P session:session.dat

tpm2_nvreadpublic
0x1000000:
  name: 000bfb12a225aad716468ad1f173cfe31dd7e26e952bf4ea42d91b747e0e6ab194f0
  hash algorithm:
    friendly: sha256
    value: 0xB
  attributes:
    friendly: policywrite|ownerread|authread|written
    value: 0x20060008
  size: 20
  authorization policy:
B7AFECEE9BF7BCBD5078F264DE85F7E361DC84F745DA7EFA34E91FDAF200EE9B

tpm2_nvread -C o 0x1000000 -s 20
    Output: `hello`
```

Owner hierarchy (default)