Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT

Trusted Firmware M in Zephyr

Kevin Townsend, Linaro

Agenda

- What is TF-M?
- How Does This Help ME?
- Secure Boot
- Secure Processing Environment
- Root of Trust
- Isolation Levels
- Secure Services
- Key Management
- Supported Boards
- Sample Applications
- Further Reading

What is TF-M?

WARNING

This presentation is a high-level overview of TF-M in Zephyr!

It tries to leave **breadcrumbs** to follow …

… but necessarily leaves some avenues as an exercise for the reader to explore.

What is Trusted Firmware-M?

- TF-M is an open source reference implementation of the **Platform Security Architecture (PSA)** IoT Security Framework
- It aims to address some of the main security concerns in IoT products
- Initiated following success of TF-A

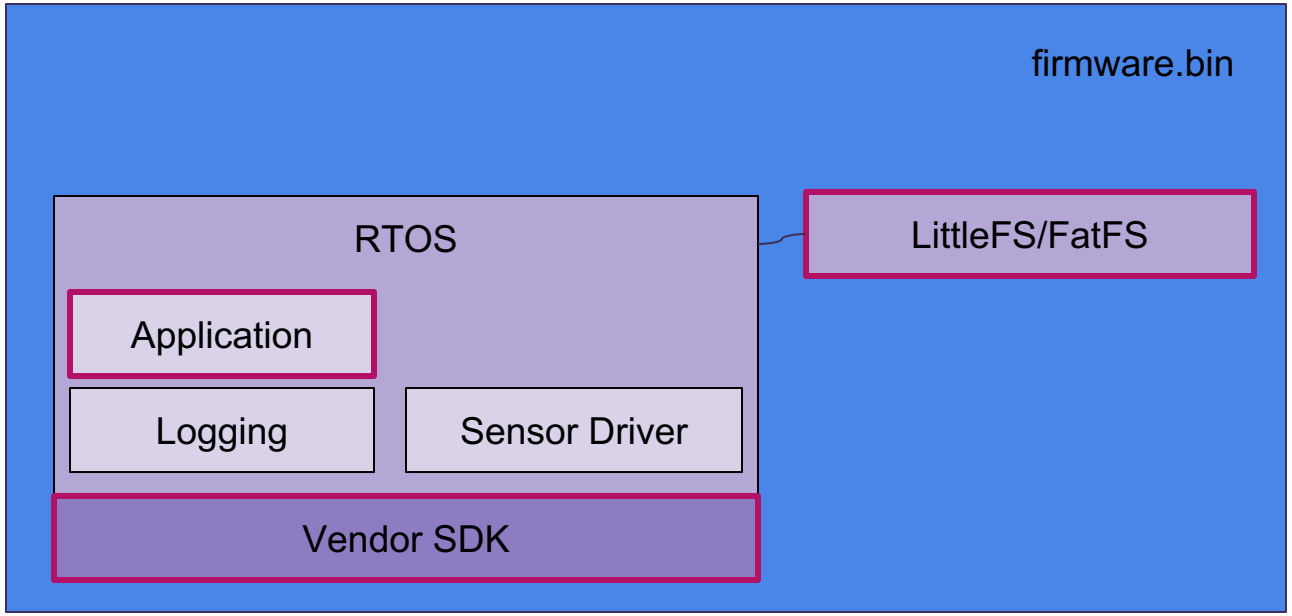
- Zephyr RTOS has been PSA Certified since Zephyr 2.0.0 with TF-M 1.0
- Re-certification planned with Zephyr 2.6.0 and TF-M 1.3.0

How Does This Help Me?

Firmware today is often monolithic, ad hoc and tends to be built like this …

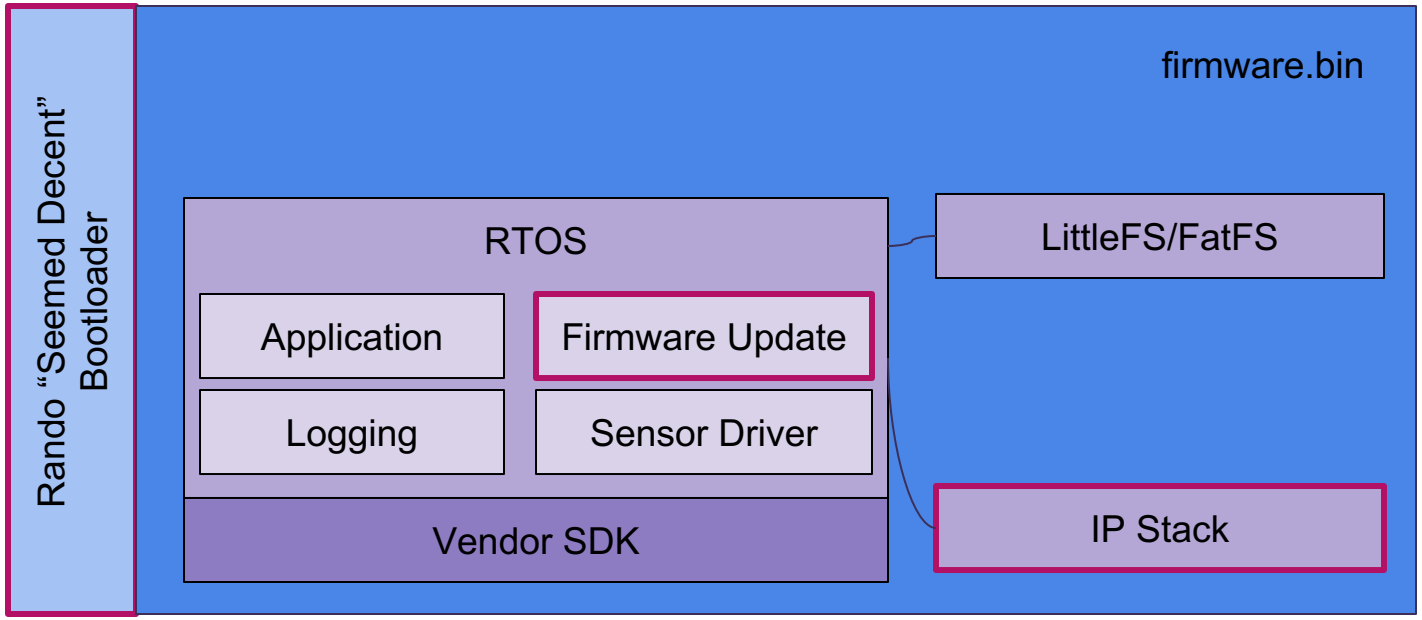
How Does This Help Me?

… You start with some requirements and manage to put various SDKs and libraries together ...



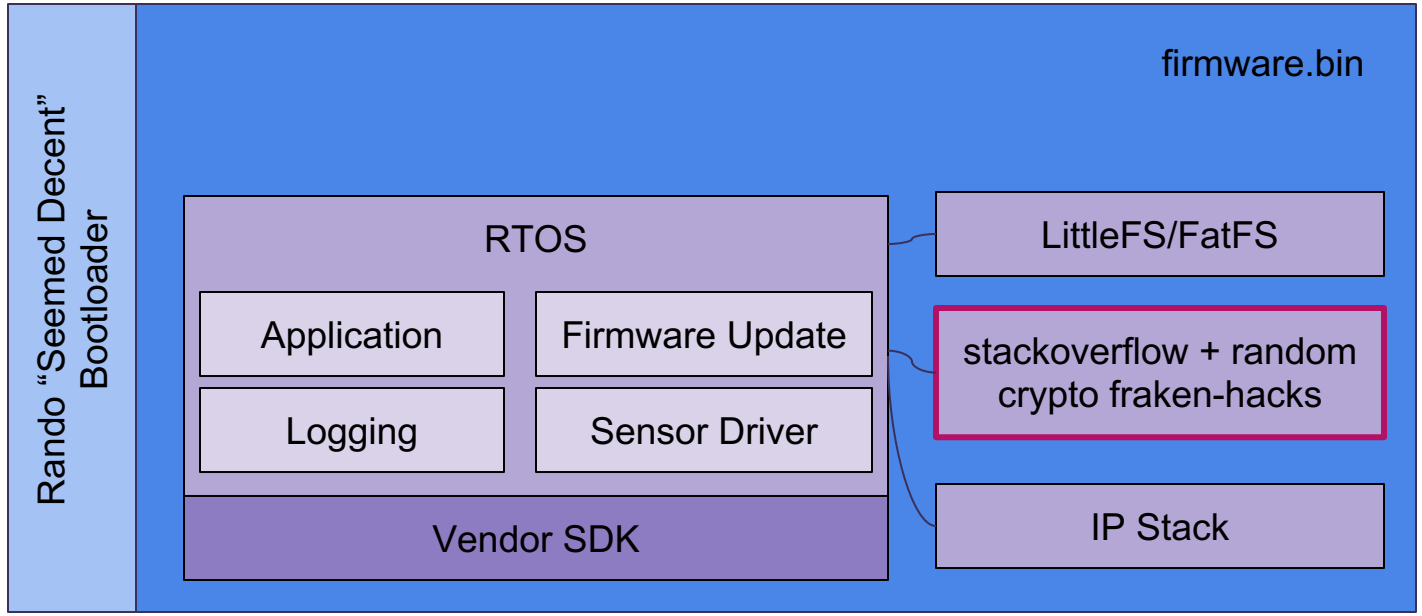
How Does This Help Me?

… 60% in you realise you need firmware updates or some other major feature ...



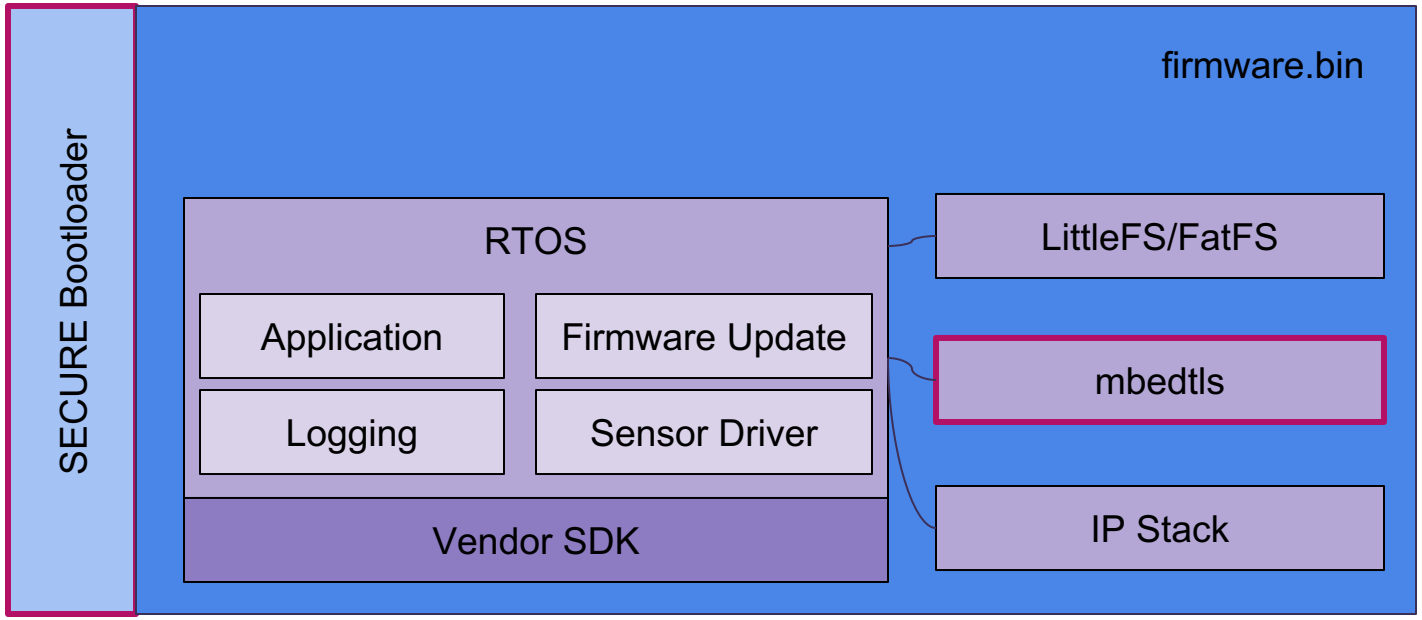
How Does This Help Me?

… 90% of the way in, you realise you care where those updates come from and implement whatever solution you can find for TLS …



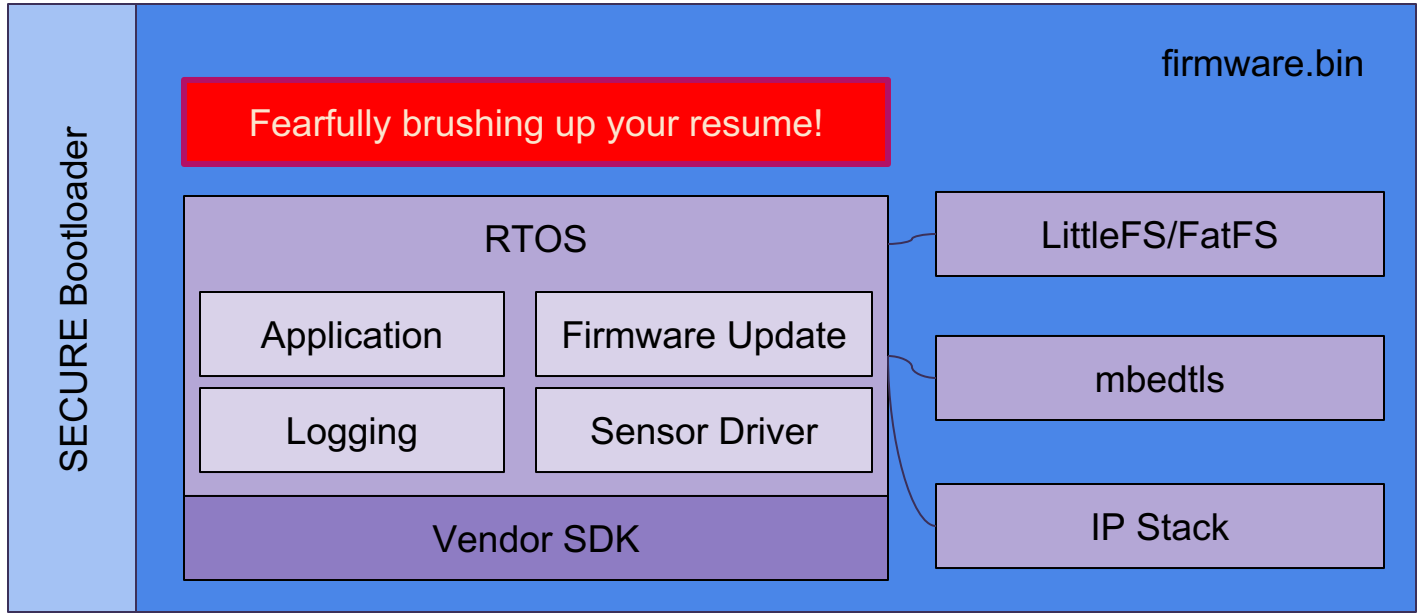
How Does This Help Me?

… 98% of the way in, you hit a brick wall testing and switch to new, more reliable key components, but with a completely untested config ...



How Does This Help Me?

… at 110% you ship a house of cards, because … deadlines!



How Does This Help Me?

What you have is …

… firmware mystery soup!

It might look decent, but …

… you couldn't test the temperature before you serve it ...

… and you don't know if those random ingredients were out of date of not!

How Does This Help Me?

Please don’t serve firmware mystery soup.

How Does This Help Me?

This approach has a number of evident problems:

- 👷 Security concerns often surface too late in the development process
- 🧠 Expertise is required to select crypto algorithms and implementations
  (Roll-your-own security!)
- 💸 No isolation of device secrets or resources
  (Any open window or doorway gives attackers the keys to the castle)
- 😓 Risky all or nothing firmware updates
- 🤪 Key management and storage often an afterthought (keys hard-coded in firmware)
- 💩 Audit logging and analysis a *distant* afterthought
- 😭 Non-security engineers may be unaware of common attack surfaces and vectors

How Does This Help Me?

What if someone could give you:

- 👮 Isolation of secure and non-secure resources
- 🔐 Modern embedded-appropriate crypto
- 📦 Reliable secret management and storage
- 🛂 Firmware verification (and encryption)
- 🏦 Protected off-chip data storage and retrieval
- 😐 Proof of device identity (Get thee behind me, Mallory!)
- 🧻 Audit logging (who accessed what and when?)
- 🙅 Enough unfamiliar acronyms to confound colleagues for days
- All free, today and out of the box with minimal learning curve?

How Does This Help Me?

TF-M brings you just what you need after that dodgy soup everyone was served last time…

How Does This Help Me?

TF-M brings you just what you need after that dodgy soup everyone was served last time…

… a balanced (**free!**) lunch

How Does This Help Me?

- TF-M is based upon a **Root of Trust** (RoT) architecture
- Unlike your mother-in-law, RoT is all about **boundaries** 🤫
- It allows for hierarchies of trust from most 👮 to less 🤪 to least 🤷 trusted
- It provides a sound **foundation** upon which other structures can be built

Secure Boot

Everything starts with a secure bootloader checking for signed images:

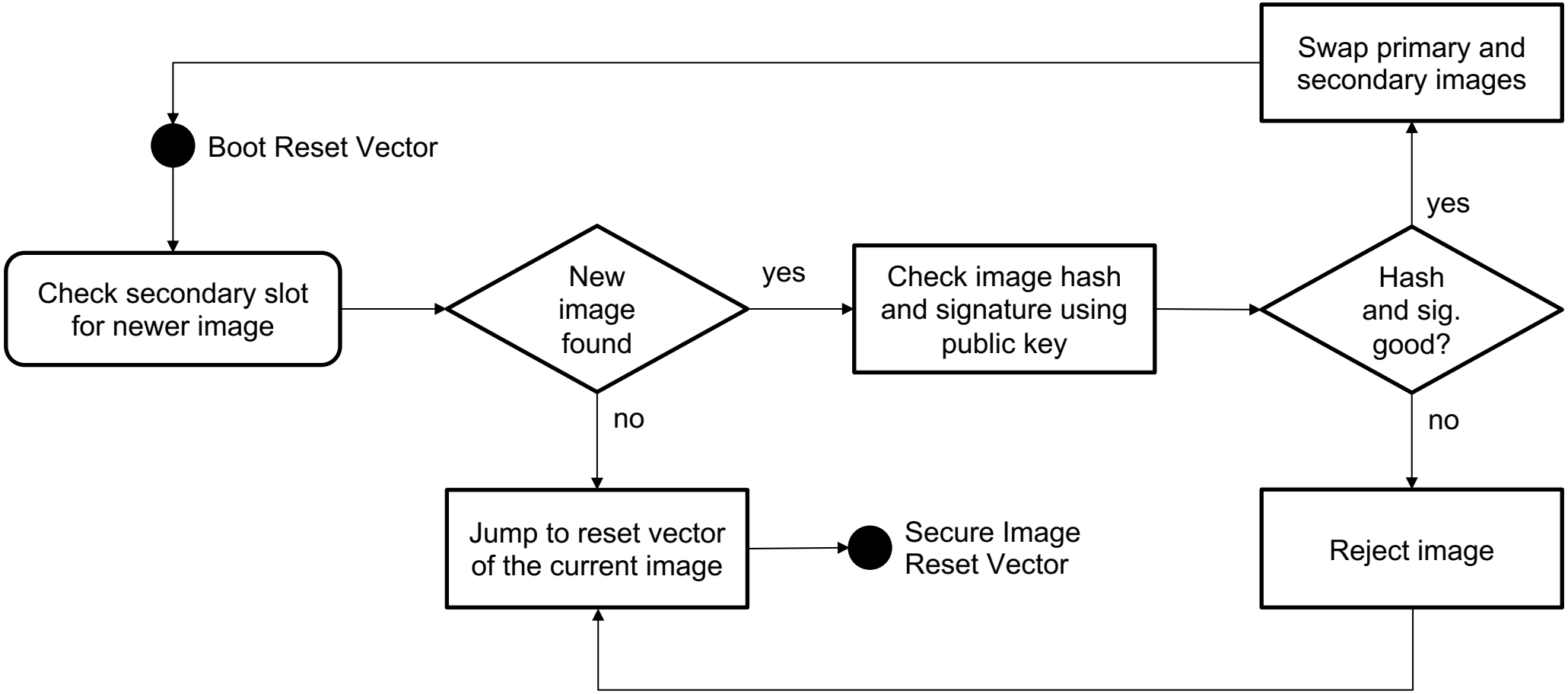
Secure Boot

- Based on https://mcuboot.com
- Public signing key baked into the bootloader (`IMAGE_TLV_KEYHASH`)
- Images are hashed and signed (tamper resistance, known provenance)
- S and NS images can be signed using different keys
- Static flash layout of two identically-sized memory regions by default
- Client software responsible for writing a new image to the secondary slot
- Optional security counter in image for rollback protection (`IMAGE_TLV_SEC_CNT`)
- Supports encryption of firmware images
- Immutable

Secure Boot: Encrypted Images

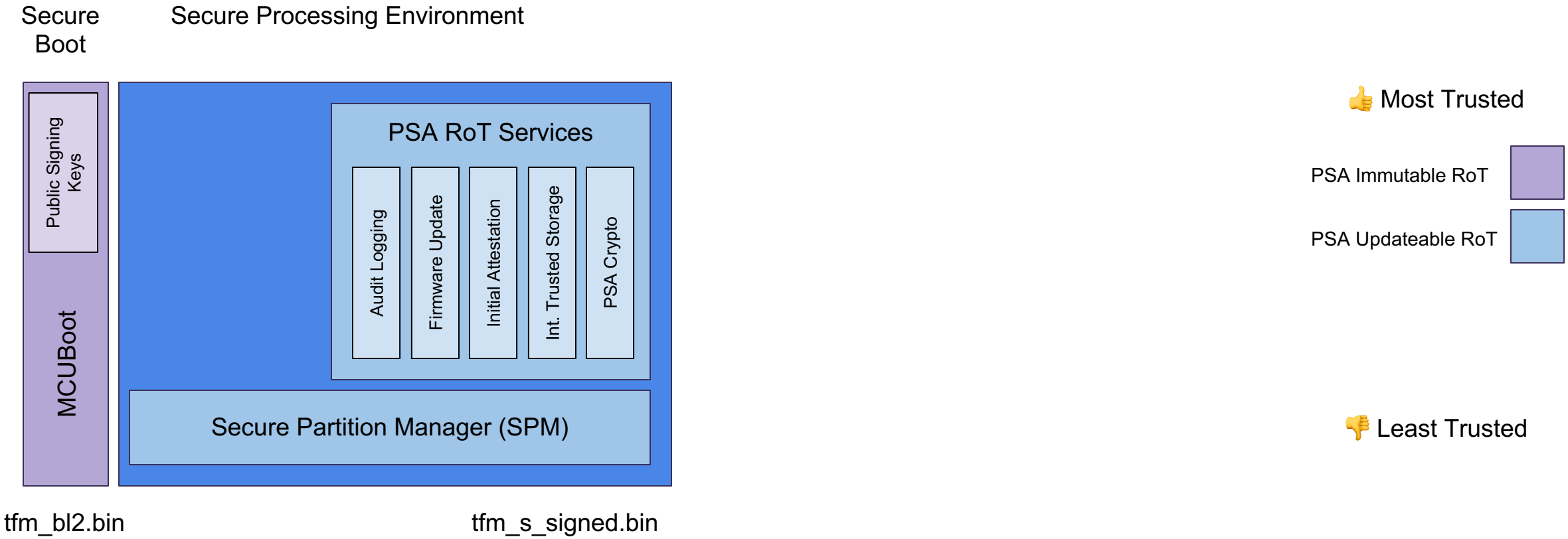
- Image encryption is optional
- Only the payload is encrypted (header, TLVs are plain text)
- Hashing and signing are applied over the un-encrypted data
- Uses `AES-CTR-128` or `AES-CTR-256` for encryption
- Encryption key randomized every encryption cycle (via imgtool)
- The `AES-CTR` key is included in the image and can be encrypted using:
  - `RSA-OAEP` (TLV value `0x30`)
  - `AES-KW` (128 or 256 bits depending on the `AES-CTR` key length) (TLV value `0x31`)
  - `ECIES-P256` (TLV value `0x32`)
  - `ECIES-X25519` (TLV value `0x33`)

Secure Boot Process



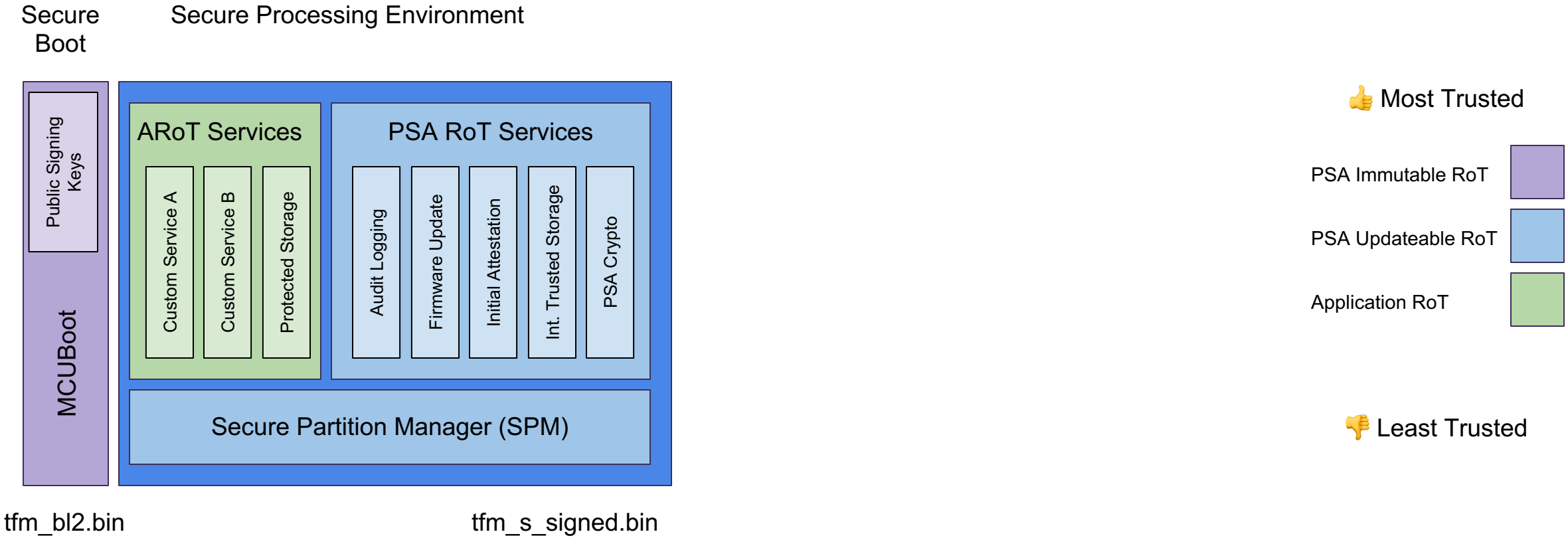
Secure Processing Environment

That starts the secure processing environment, booting up critical systems



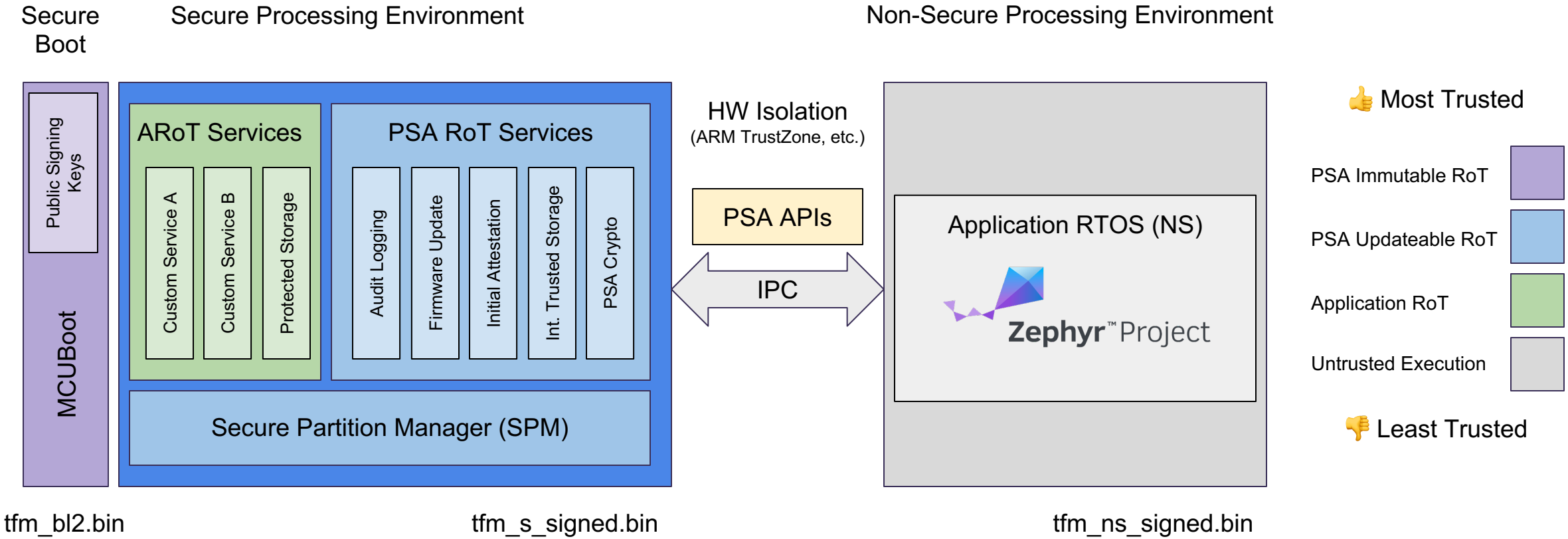
Secure Processing Environment

Which can then start up other isolated secure services with lower privilege levels



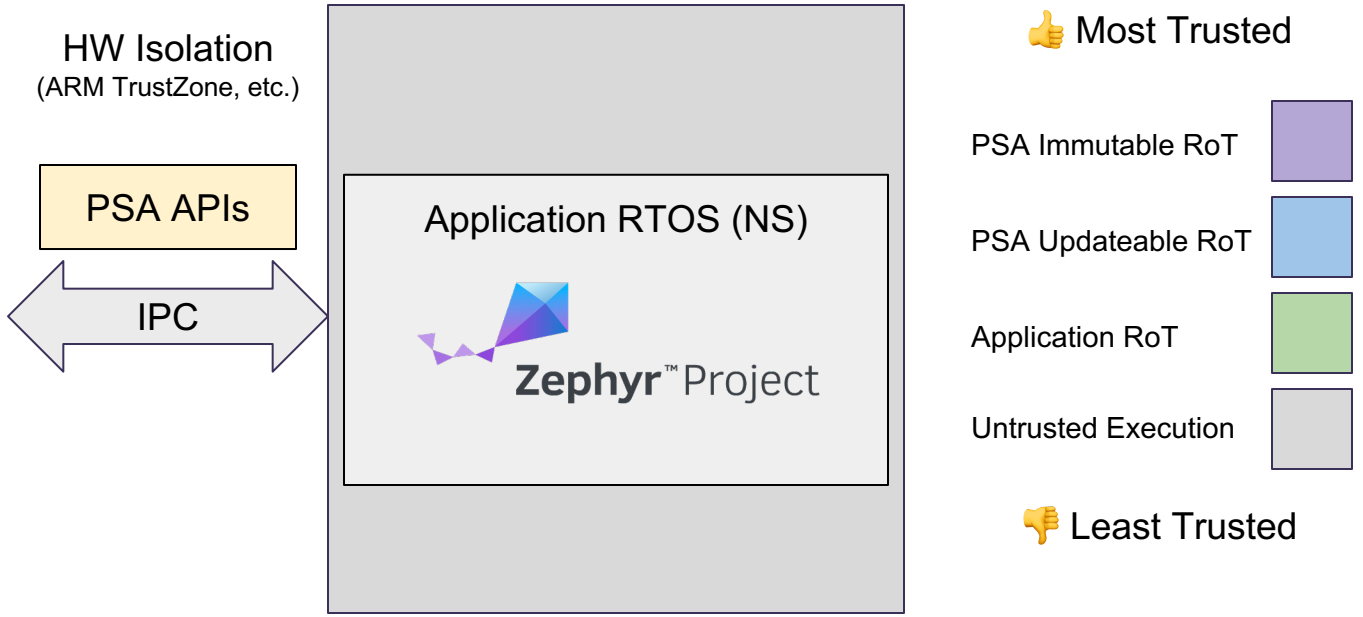
Non-Secure Processing Environment

Before handing off to the untrusted environment, windows/doors locked tight



The Free Lunch Part!

Plus, Zephyr builds, links and exposes TF-M for you, no know-how required!



Summary

This architecture allows for:

- Avoiding common security issues out of the box with minimal investment
- A high degree of *trust* in the integrity of the system, and data coming off it
- Strict (usually HW-based) separation of resources
  - A security failure in the NS environment won't compromise the secure one
  - Untrusted code never has access to device secrets
- Algorithm selection (and implementation!) by security experts
  - Selecting crypto ciphers that are robust AND appropriate for embedded system is hard
- S and NS images can be individually signed and developed
- Sleeping a bit more soundly at 3:00 AM on a Sunday

The T in TF-M: Root of Trust

TF-M contains two Root of Trusts:

- **PSA Root of Trust** (PRoT)
  This is the most privileged domain
- **Application Root of Trust** (ARoT)
  Application-specific domain(s)

PSA Root of Trust (PRoT)

PRoT is the **primary** and most trusted Root of Trust, and consists of two parts:

- **PSA Immutable Root of Trust** (Secure Boot)
  - Secure boot is the ultimate root of trust for all software that follows!
  - Inherently trusted

- **PSA Updateable Root of Trust** (Secure Services, Secure Partition Manager)
  - Trusted through verification anchored to the PSA Immutable Root of Trust
  - Has access to other services in the PSA-RoT
  - Isolated from the Application Root(s) of Trust (see next slides)
  - Mutable (TF-M secure image can be updated)

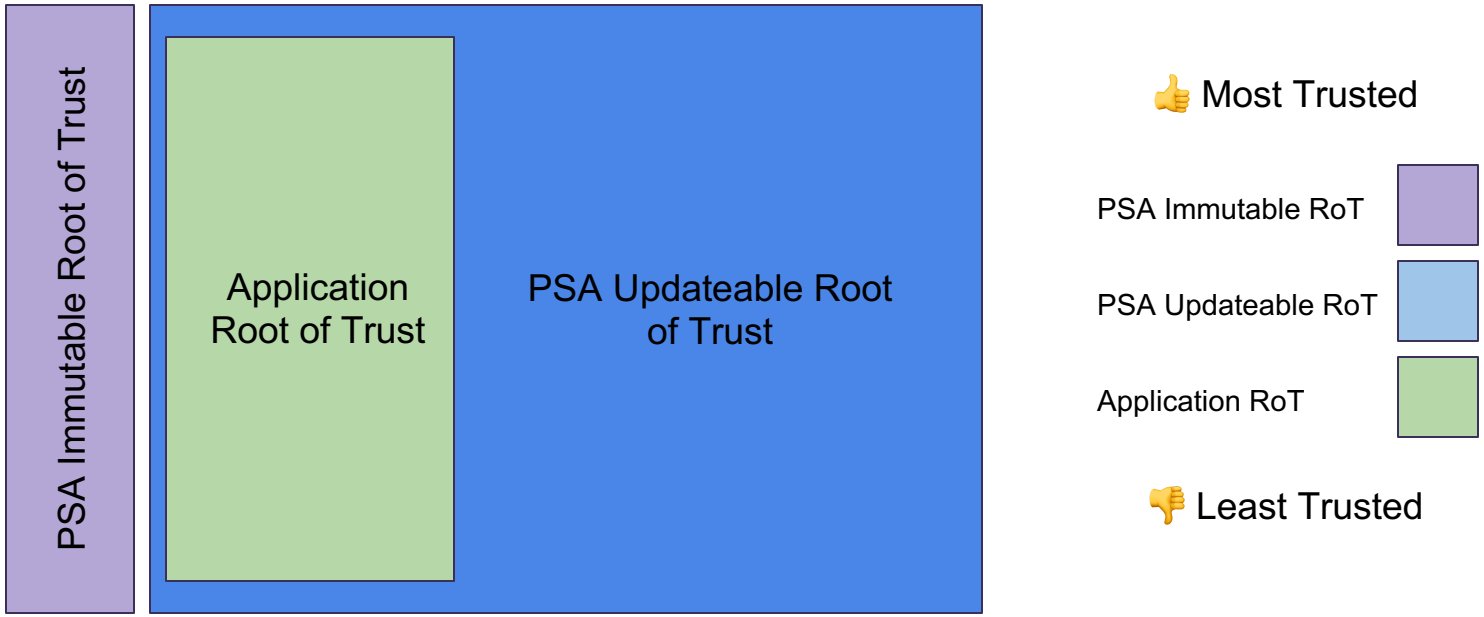
Application Root of Trust (ARoT)

The **Application RoT** has a **reduced privilege level**:

- Provides application-specific trusted services
- Has no direct access to immutable components
- Unable to access the PSA Root of Trust (Isolation Level 2)
- Unable to access other ARoTs (Isolation Level 3)
- Mutable (TF-M secure image can be updated)

Root of Trust Model

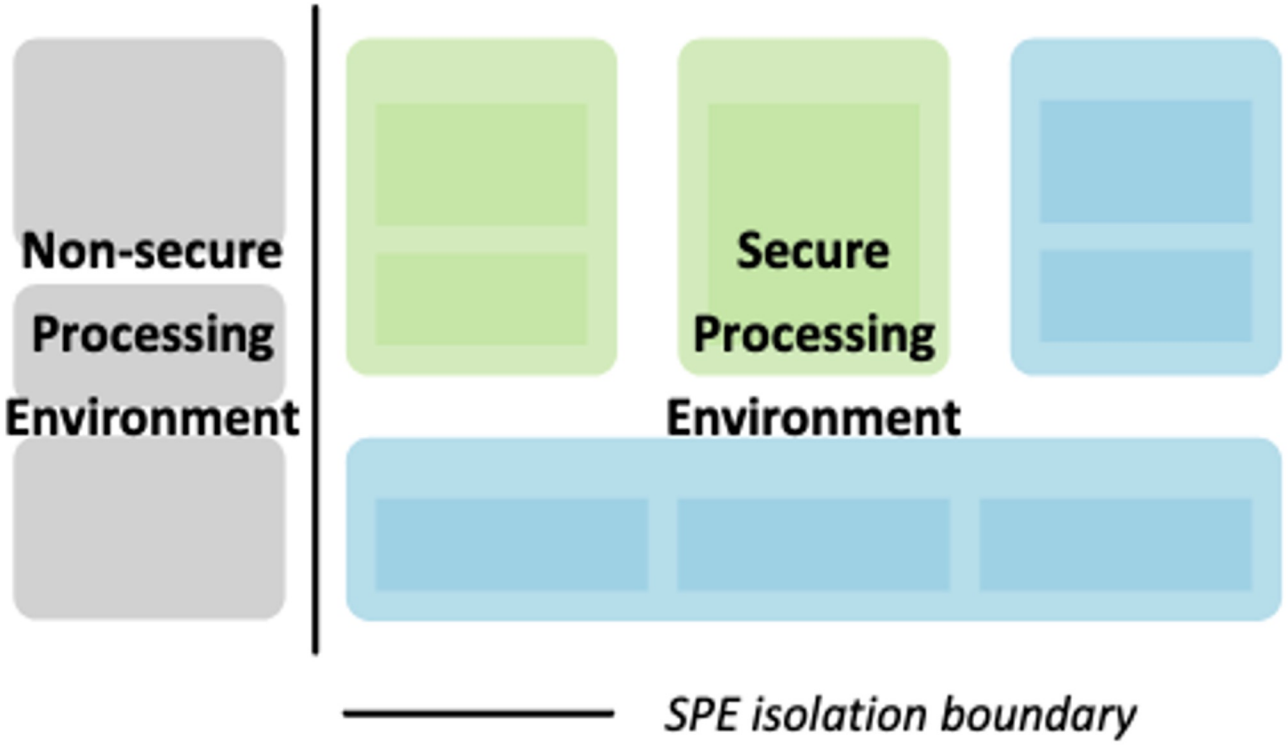
- Various '**levels**' of separation between RoTs are supported, depending in isolation requirements and available system resources:



TF-M Isolation Levels

Isolation Level 1

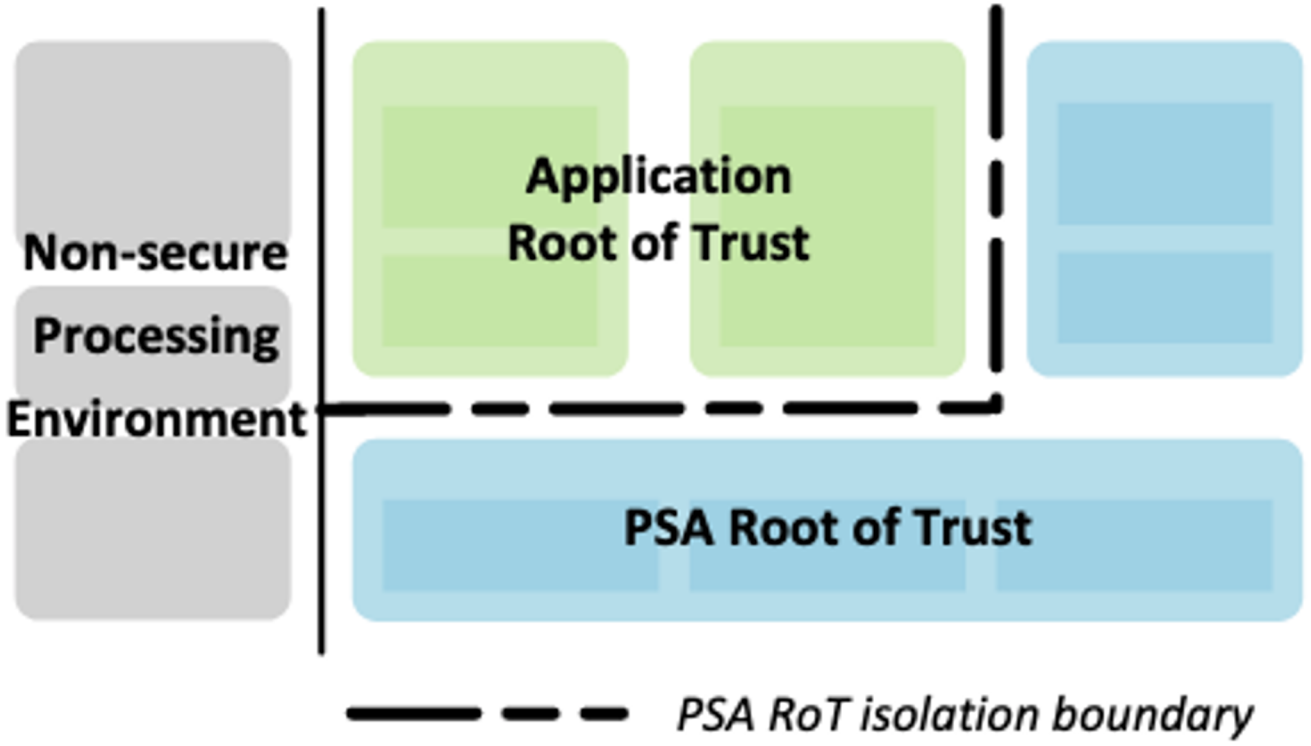
- SPE and NSPE isolated
- Secure services not isolated from each other
- ARoT meaningless here



TF-M Isolation Levels

Isolation Level 2

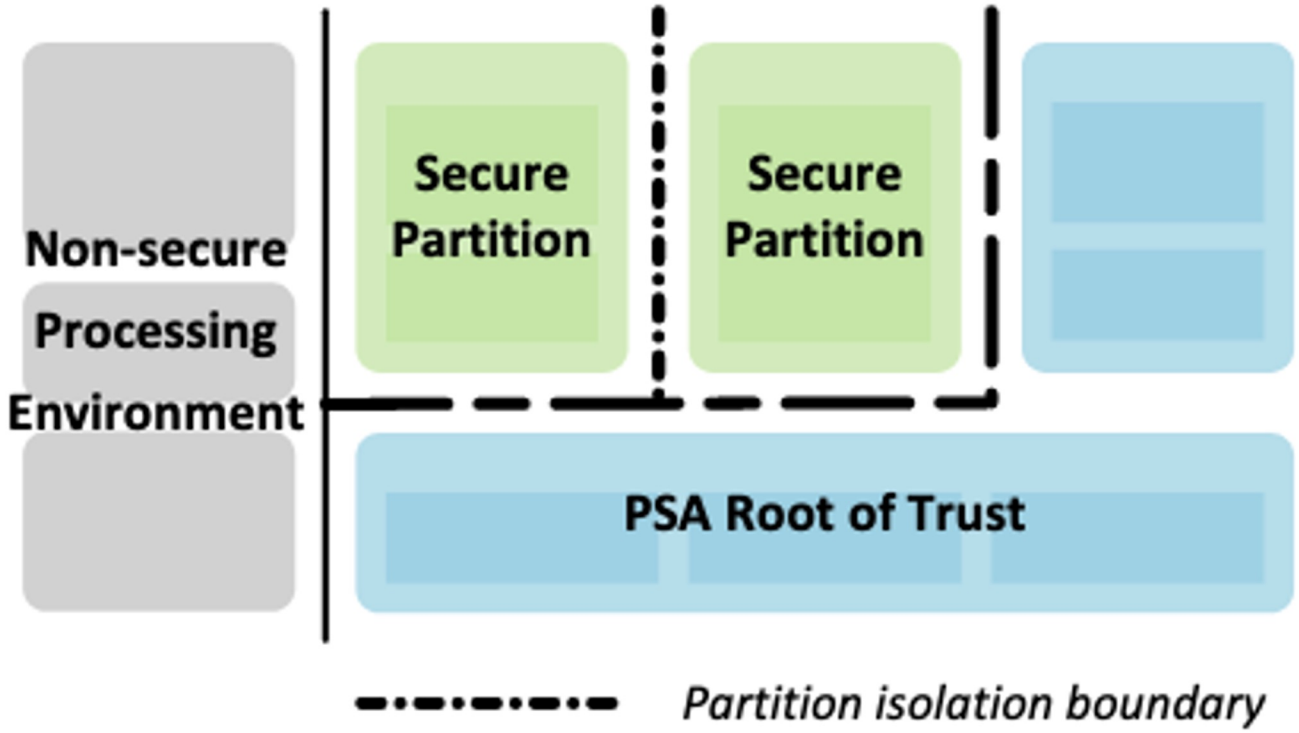
- ARoT services isolated from PRoT services
- ARoT services are NOT isolated from each other



TF-M Isolation Levels

Isolation Level 3

- ARoT services isolated from each other



Secure Services

As of TF-M 1.3.0, the following secure services are available out of the box:

- Audit Logging (Audit)
- Crypto (Crypto)
- Firmware Update (FWU)
- Initial Attestation (IAS)
- Secure Storage:
  - Internal Trusted Storage (ITS)
  - Protected Storage (PS)

- A template also exists for creating your own custom services!

Audit Logging (Audit)

Allows secure services in the system to log critical system events and information that has security implications.

- ALS integration guide
- Limited to RAM based storage (as of TF-M 1.3.0)
- Entries made up of **Audit Log Records**
- They consist of a header, and one or more TLV (type, len, value) records

Domain: PSA Root of Trust (PRoT) (Secure Privileged Domain)

Crypto (Crypto)

Provides an implementation of the **PSA Crypto API**, and is based on **mbedtls**. It's accessible to services in the secure processing environment, or to applications in the non-secure processing environment.

- PSA Crypto integration guide
- Crypto Service Design
- PSA Crypto API 1.0

Domain: PSA Root of Trust (PRoT) (Secure Privileged Domain)

Firmware Update (FWU)

Provides a shim layer between the firmware update partition in TF-M and the bootloader, allowing firmware updates to be initiated and managed from the application (NS) side. It allows you to:

- Query the current firmware version(s) for both the running and staging areas.
- Write image(s) to the staging area
- Request validation of an image in the staging area
- Trigger a reboot and update attempt

- Firmware Update Service Documentation

Domain: PSA Root of Trust (PRoT) (Secure Privileged Domain)

Initial Attestation (IAS)

IAS provides **Initial Attestation Tokens (IATs)** for a unique device, which can be use to prove **device identity**, and describe specific features of the device via the IAT payload.

- IAT Integration Guide
- Tokens are encoded according using CBOR and signed following COSE
- When requesting a token, a 32/48/64 byte **authorisation challenge** can be included in the request
- The optional challenge is included in the response token to help prevent **replay attacks**, since the final payload is also **signed** with private key

Domain: PSA Root of Trust (PRoT) (Secure Privileged Domain)

Secure Storage: Internal Trusted Storage

Internal Trusted Storage (ITS) is a PSA RoT Service for **storing the most security-critical device data** (e.g. cryptographic keys) in internal storage, which is trusted to provide data confidentiality and authenticity.

- ITS integration guide and ITS service technical reference
- Limited to internal non-volatile memory (eFlash, etc.), or optionally to RAM
- No encryption or authentication is used!
  - It is assumed that the internal flash memory used by the secure processing environment is protected in HW or via another means.
- Resistant to power failure scenarios and incomplete write cycles

Domain: PSA Root of Trust (PRoT) (Secure Privileged Domain)

Secure Storage: Protected Storage

Allows larger data sets to be stored securely in external flash, with the option for encryption, authentication and rollback protection to protect the data-at-rest.

- PSA Integration Guide
- Intended to be used with **external non-volatile memory**, but can also be redirected to internal non-volatile memory or RAM (`PS_RAM_FS`)
- Default crypto algorithm is `AES-128-GCM`
- Encryption key derived from the HUK
- Tamper resistant: malicious change attempts detected by the service

Domain: Application Root of Trust (ARoT)

Key Management: HUK

- The HUK provides the root of trust for confidentiality in the system!
- It must have at least **128 bits of entropy** (and a 128 bit data size)
- Accessible only to Trusted code, or hardware on behalf of Trusted code
- Generally stored in a crypto element (CC312, etc.) or OTP memory
- `TFM_HUK_KEY_ADDR` is a pointer to the shared HUK data (used in combination with `TFM_HUK_KEY_LEN`)
- `TFM_CRYPTO_KEY_ID_HUK` points to the key ID for the HUK when using the PSA Crypto API

Key Management: Key Derivation

- The HUK can be used to **derive new keys**, as seen in Protected Storage:
  - During startup, PS requests that the Crypto service derives a storage key from the HUK
  - PS itself does not have access to the key material, only a key handle
- Derived keys don't need to be stored since they can be regenerated from the HUK at startup, using an additional salt/seed value (depending on the key derivation algorithm used)
- `TFM_CRYPTO_ALG_HUK_DERIVATION` identifies the default algorithm used
  - If a software implementation is used, the current default algorithm is `HKDF` (RFC 5869) with a `SHA-256` hash
  - Other hardware implementations may be available on some platforms

Key Management

- PSA Crypto in combination with Internal Trusted Storage (ITS) can also be used to generate and store private keys in any format that PSA Crypto (mbedtls) supports
- Private key data can be generated via entropy, or a secure write-only process of provisioning static key data in the factory can be defined
- Private key data never leaves the secure processing environment
- All key-based operations are executed via a key ‘handle’

Supported Boards

- As of Zephyr 2.6.0, the following boards can be used with TF-M:
  - Emulated Platforms:
    - mps2_an521_nonsecure (qemu, Cortex-M33)
  - Hardware Platforms:
    - bl5340_dvk_cpuappns
    - lpcxpresso55s69_ns
    - mps3_an521_nonsecure
    - nrf5340dk_nrf5340_cpuappns
    - nrf9160dk_nrf9160ns
    - nucleo_l552ze_q_ns
    - stm32l562e_dk_ns

Sample Applications

The following samples are available in `samples/tfm_integration`:

- **psa_level_1** (rename planned to psa_crypto):
  Demonstrates how to use the PSA Crypto API for key generation, key storage, and securely signing and hashing data without exposing the keys.
- **psa_protected_storage**
  Demonstrates how to use protected storage to protect device secrets from the non-secure processing domain
- **tfm_regression_test** or **tfm_psa_test**
  Runs the TF-M regression or PSA test suites (useful to PSA Certification!)

Running Samples

- You can run any of the samples in QEMU with some variation of:

```
$ cd <ZEPHYR_ROOT>
$ west build -p -b mps2_an521_nonsecure \
  samples/tfm_integration/psa_level_1
  -t run
```

- To run on real hardware, simply reference an appropriate board

```
$ west build -p -b lpcxpresso55s69_ns \
  samples/tfm_integration/psa_level_1
```

… then flash the resulting images with the tools required by that board.

Further Reading

- TF-M Developer Documentation
- Platform Security Documentation

- Upstream TF-M repository
  https://git.trustedfirmware.org/TF-M/trusted-firmware-m.git/

- Zephyr's TF-M Module
  https://github.com/zephyrproject-rtos/trusted-firmware-m

- My technical notes to Future-Me putting this presentation together
  https://gist.github.com/microbuilder/53ac490db265e86e5840a9f752b3dc48

Zephyr™ Project
Developer Summit

“Privacy is not for the passive.”

- Jeffrey Rosen


Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT