Designing a Trusted Execution Environment in Zephyr OS

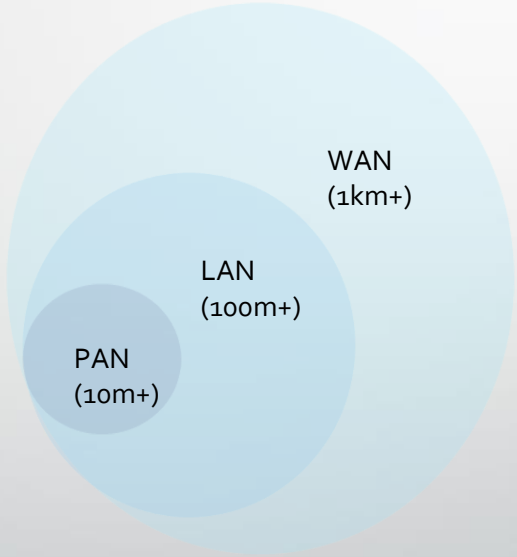
Ioannis Glaropoulos

Agenda

- Background
  - Embedded device security considerations
  - Trusted Execution and ARM TrustZone-M fundamentals
- Trusted Execution APIs and implementation in Zephyr OS
- Building Trusted Firmware in Zephyr OS
- Future plans

Nordic Semiconductor vision

A leading vendor of wireless connectivity and embedded processing solutions for internet connected things



Things we **wear**

... we **carry around**

... around us at **home**

... around us at **work**

... around us in the **city**

... around us in the **country side**

Things - not infrastructure, ... not PCs, ... not phones ... not tablets

Nordic as an IoT enabler

(…)

Logistics/transport

Payment/ID

Enterprise automation

Industrial automation

Longer tail

Wireless Charging

Automotive

Smart Home

Beacons

Healthcare

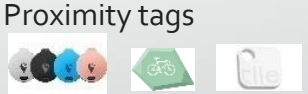
Emerging

Wearables

Remote controls

PC/phone/tablet HID

Gaming



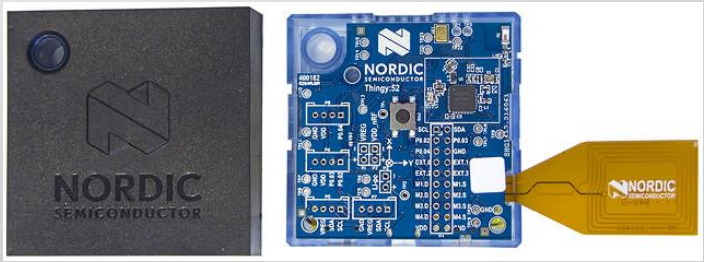
Proximity tags

Toys

Established Verticals

- SOCs

- Software
  - Protocol stacks
  - SDKs
  - Front-end APPs
- Development kits



Security in embedded devices: an important concern

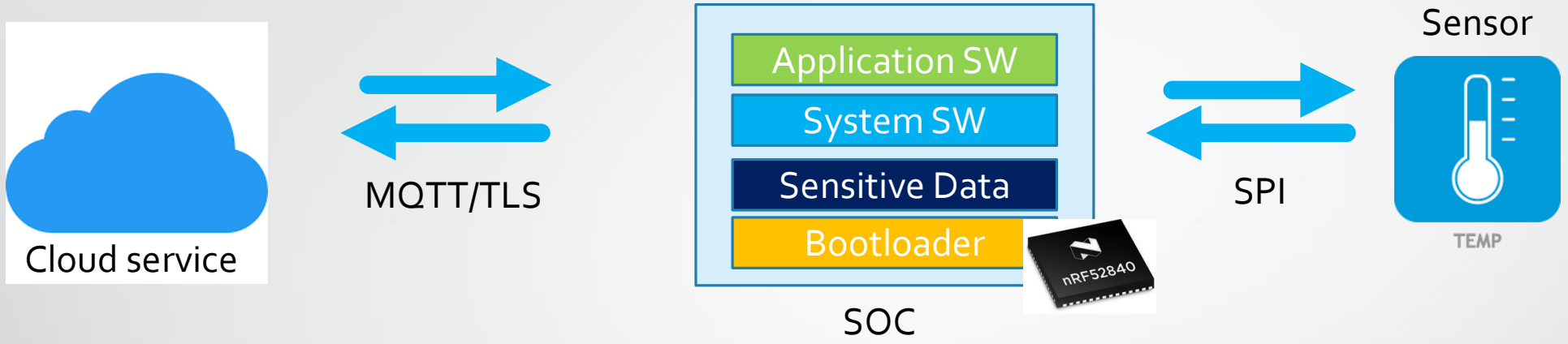
Embedded Devices

- Are **not** tiny
  - Typically ~100MHz clock, ~1MB flash, ~100kB RAM, sensors, interfaces
- Are connected & remotely accessible
  - To/from the internet, to/from each-other
- Store & share sensitive information
- May be part of (very) large systems
- May compromize system-level security (even if a single device is compromized)

Embedded Software

- Is complex, buggy, and is being updated **multiple** times
- Is developed by **multiple** entities
- Is vulnerable to all kinds of mordern tools that can read/write/corrupt/expose it

IoT sensor device threats



Example threats

- Functional abuse (e.g. stack overflows, un-defined commands, etc.)
- Expose secrets / keys to un-authorized entities
- Corrupt root-of-trust establishment during device boot
- Load an un-authorized application firmware image
- Expose sensor data to un-trusted entities
- Corrupt notion of system/current time
- Un-authorized access to device management / system logs

Trusted Execution Environments

1. Data protection

- Sensitive data, keys, secrets, authorized-access
- Avoid system compromize due to device compromize

2. Firmware protection

- E.g. readback protection
- Avoid reverse engineering

3. Operational protection

- Un-compromised execution of critical (trusted) functionality
- Protection from accidental/malicious un-trusted software

4. Communication protection

- E.g. obscure crypto operation

TrustZone-M: TEE for ARM Cortex-M

Dual processor state and resource attribution

- Secure (S), Non-Secure (NS) execution
- Address security ←→ domain security

Hardware-based isolation

- Security state transition: in (and only in) a predefined way
  - S → NS : **Non-Secure function calls** with resource stacking and clearing
  - NS → S: **Non-Secure callable entry** functions
- Security fault mechanism to detect security violations

Resource «banking» (i.e. Double instances)

- Execution stacks
- Vector Tables
- MCU control blocks (MPU, NVIC, Faults, Control, etc.)

ARM TrustZone-M: in depth

S/NS execution context orthogonal to

- Thread – Handler mode
- Privileged – Unprivileged mode

Seamless prevention of Secure content leaking upon state transition

- Domain context-switch with register clearing (compiler)
- Hiding internal domain processing state

Configurable S/NS interrupt routing

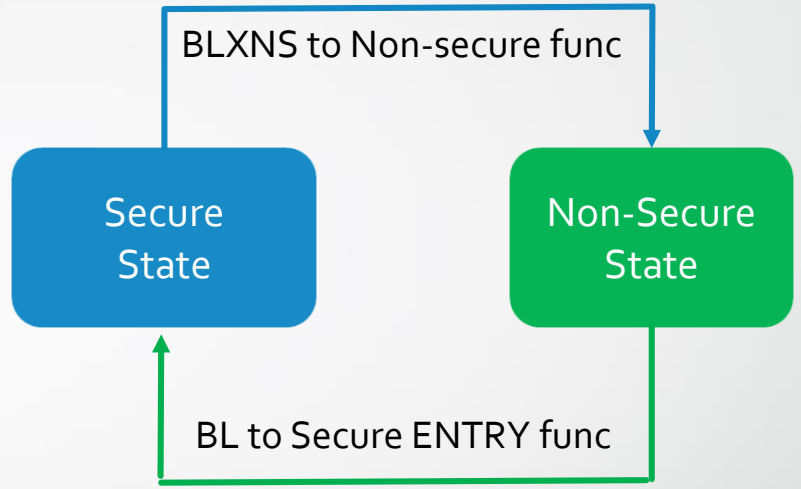
- Faults, Exceptions, HW interrupts

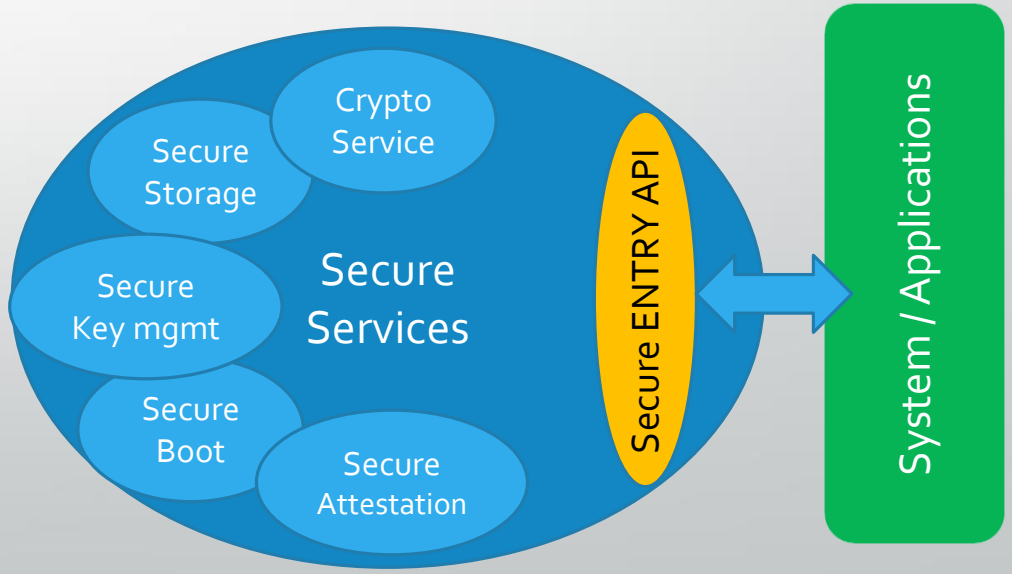
Dedicated exception for handling of security violations

- SecureFault and SecureHardFault

Dedicated IP

- Security Attribution Unit (SAU)
- ...or Implementation-Defined Attribution Unit (IDAU)





Trusted Execution for ARM in Zephyr OS

What APIs to support?

What functionality to implement?

- No reference implementation elsewhere ☹
- Begin with core-TrustZone functionality and ARMv8-M support ☺

How to use the functionality?

- How to build Zephyr applications with TEE?
- How to organize the device firmware?
- Integration with Zephyr build /configuration system (Kconfig, Device Tree)

How to abstract the APIs?

- Zephyr is a cross-architecture OS
  - At least ARM and ARC architectures have support for TEE

Zephyr TrustZone-M – implementation

NORDIC SEMICONDUCTOR

ARM-only APIs for:

- Configuring security attributions for memory areas
- Evaluating security configuration
  - Using ARMv8-M **TT** intrinsics
  - For memory addresses/ranges/objects
- Interrupts' management
  - Priority boosting, S/NS routing
- Non-Secure function pointer registration
- Secure entry functions

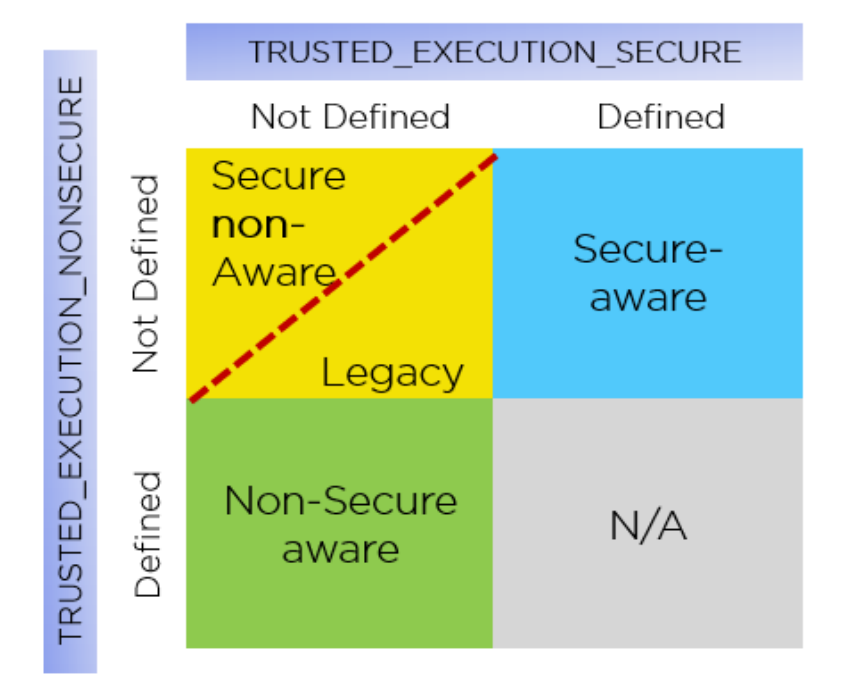
Secure fault handling

- With deep, all stacks' inspection

Integration with Zephyr build system

- Cmake (compile options configuration)
- Linker
  - Dedicated sections for Secure entry functions
  - Symbol table generation for Secure entry veneers

Building Zephyr applications with TEE



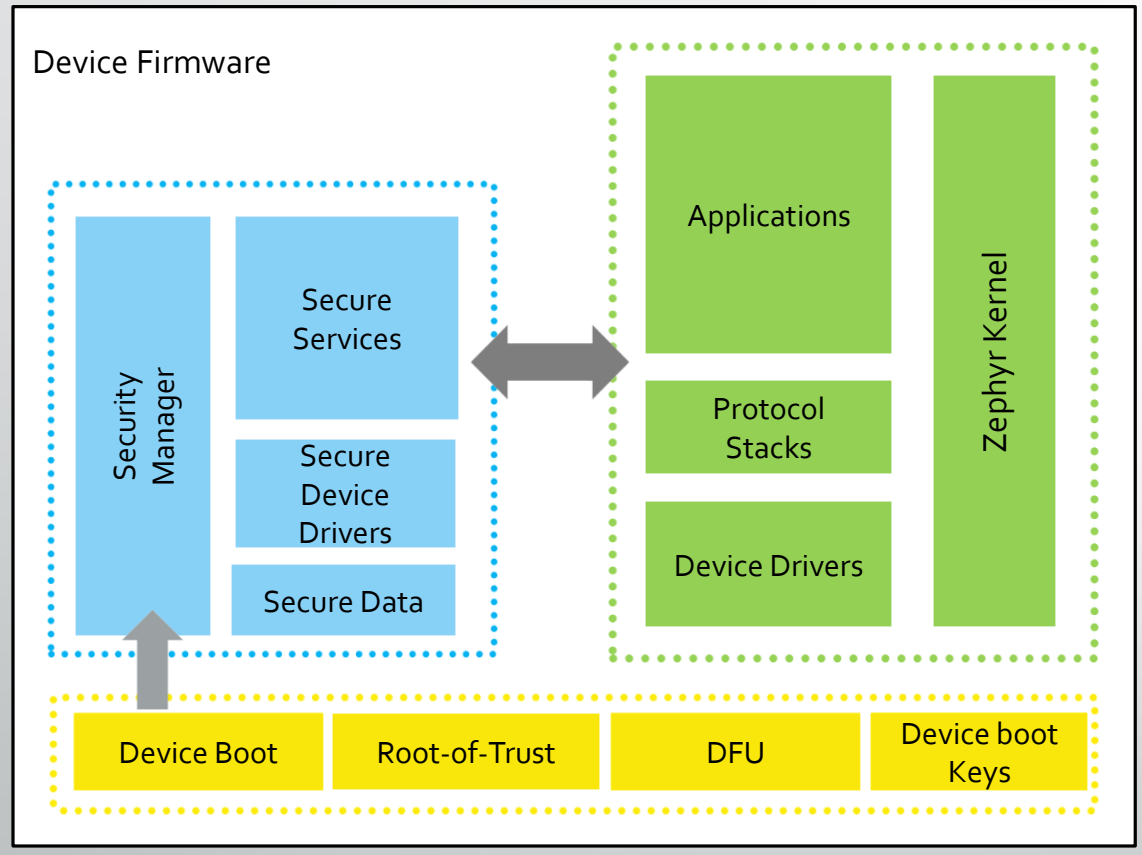
Separate Zephyr builds for Secure (S) and Non-Secure (NS) images

- Combined into single firmware image
- Secure firmware: mini-kernel, secure libraries (e.g. crypto). Typically: secure bootloader
- Non-Secure firmware: full RTOS, drivers and user applications

Firmware categorization

- **Secure "non-aware" image**
  - By default in Secure mode
  - Never transits to Non-Secure code execution
  - Typically: first-stage bootloader
- **Secure "aware" image**
  - Configures security attributions
  - Runs secure libraries
  - Allows transition to Non-Secure code execution
  - Implements SecureFault handling
- **Non-Secure "aware" image**
  - Running in Non-Secure mode
  - Shall not directly access secure MCU resources
  - May only access the Non-Secure callable API

Device firmware organization



Secure "non-aware": Bootloader-0

- Immutable
- Implementing Root-of-Trust
- Minimal (Zephyr-built or bare-metal)

Secure "aware": Bootloader-1

- Upgradeable
- Security Management
- Secure applications

Non-Secure "aware" : RTOS + Application

- Un-trusted application code
- Calling Secure services via Non-Secure Callable API

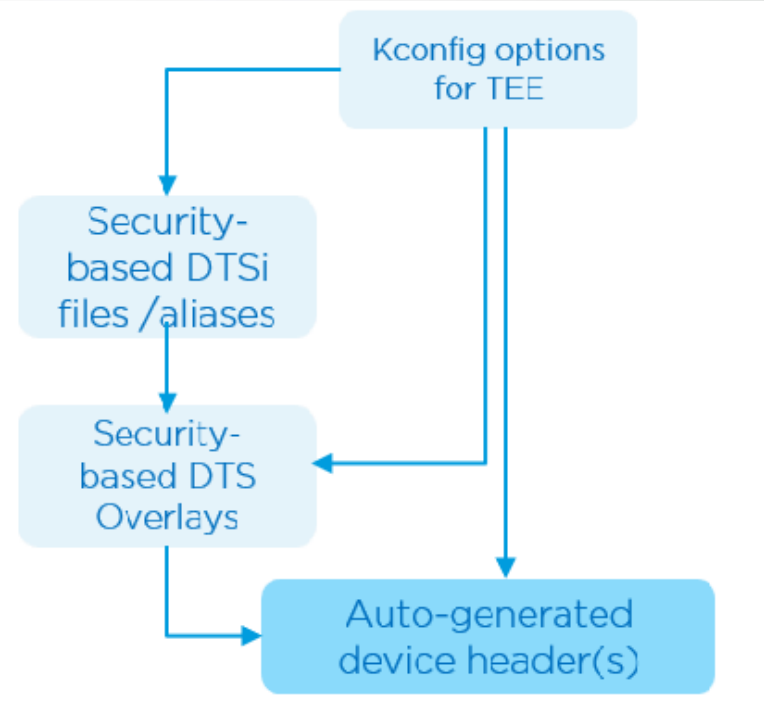
Security-aware boot-time configuration

HW description dependent on the security domain

- Affects drivers, libraries, protocol-stacks interracting with Hardware, File Systems, etc.

Made easy with Zephyr Device Tree

- Device Tree Source (DTS) files are a common convention in Zephyr to describe HW and boot time configuration
- **Separate .dtsi descriptor files** for Secure and Non-Secure firmware images
- Secure / Non-Secure firmware **image planing** in DTS (Board, SOC)
- Use of DTS **overlay** files for over-writing DTS configuration



Future plans – TEE abstractions in Zephyr

Generic, cross-architecture APIs for

- Configuring & assessing security attributions for memories and Kernel objects
- Managing Secure/Non-Secure core MCU resources
  - MCU register blocks, faults, exceptions, priority boosting, etc.
- Peripheral security
  - Access configuration
  - Managing DMA
  - Managing interrupt routing

Future plans – Secure Services in Zephyr OS

Secure applications natively supported in Zephyr

- Secure boot & Root of Trust implementation
- Key management
- Secure peripheral services

Services (in or on top of Zephyr)

- Secure data storage
- Secure firmware upgrade
- Generic secure services' API

Q/A

- Zephyr Project
  - http://zephyrproject.org/
- Zephyr Github project for Trusted Execution
  - https://github.com/zephyrproject-rtos/zephyr/projects/11
- ARM TrustZone-M
  - https://www.arm.com/why-arm/technologies/trustzone-for-cortex-m