High-end security features for low-end microcontrollers: Hardware-security acceleration on ARMv8-M systems

ARM

Milosch Meriac
Principal Security Engineer

github.com/ARMmbed/uvisor

Embedded World Conference
14 March 2017

©ARM 2017

Why is microcontroller security so hard?

©ARM 2016

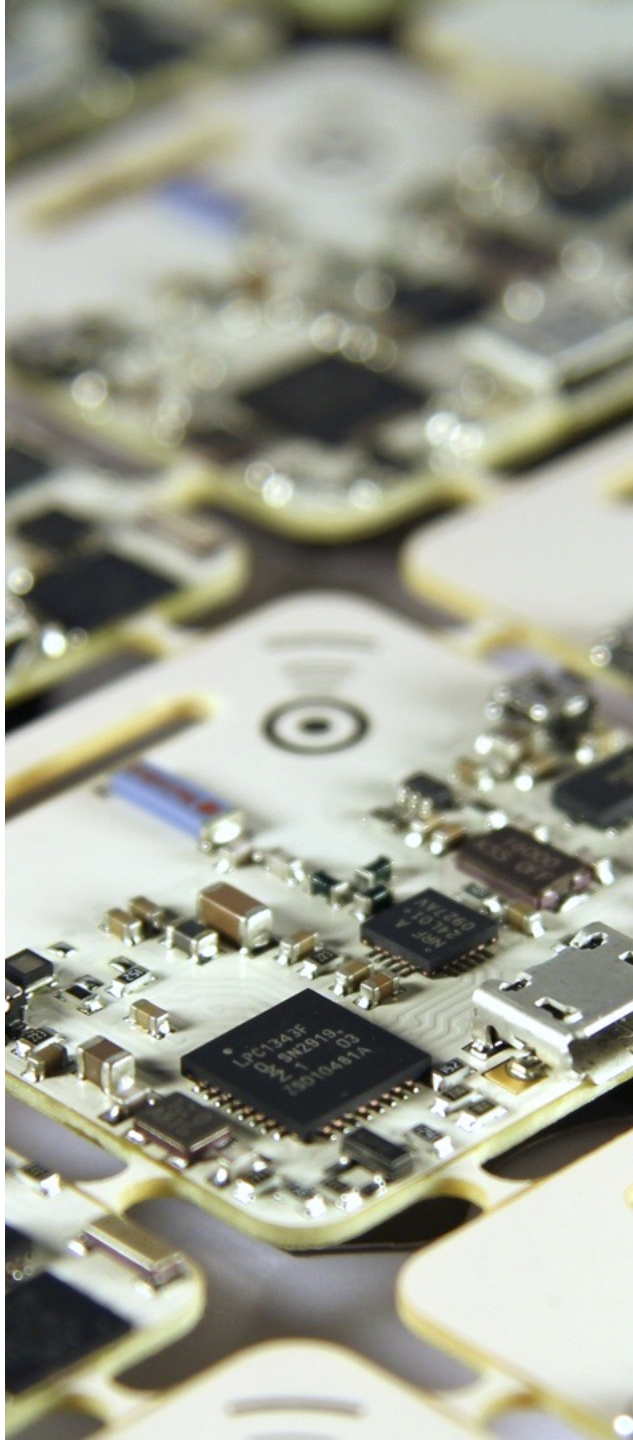
Microcontroller systems are usually…

- Simple
- Low cost
- Deployed for a long time

…. if a product is successful, it will attract hackers

The IoT revolution has increased connectivity requirements in embedded systems, but:

- Many MCU developers lack sufficient security knowledge
- Cost pressure prevents sophisticated security solutions
- Reuse of old software that was not designed for security



©ARM 2017

Microcontroller systems

Device lifetime

The security of a system is dynamic over its lifetime. Lifetimes of home automation nodes can be 10+ years

Attacks may be easily scaled

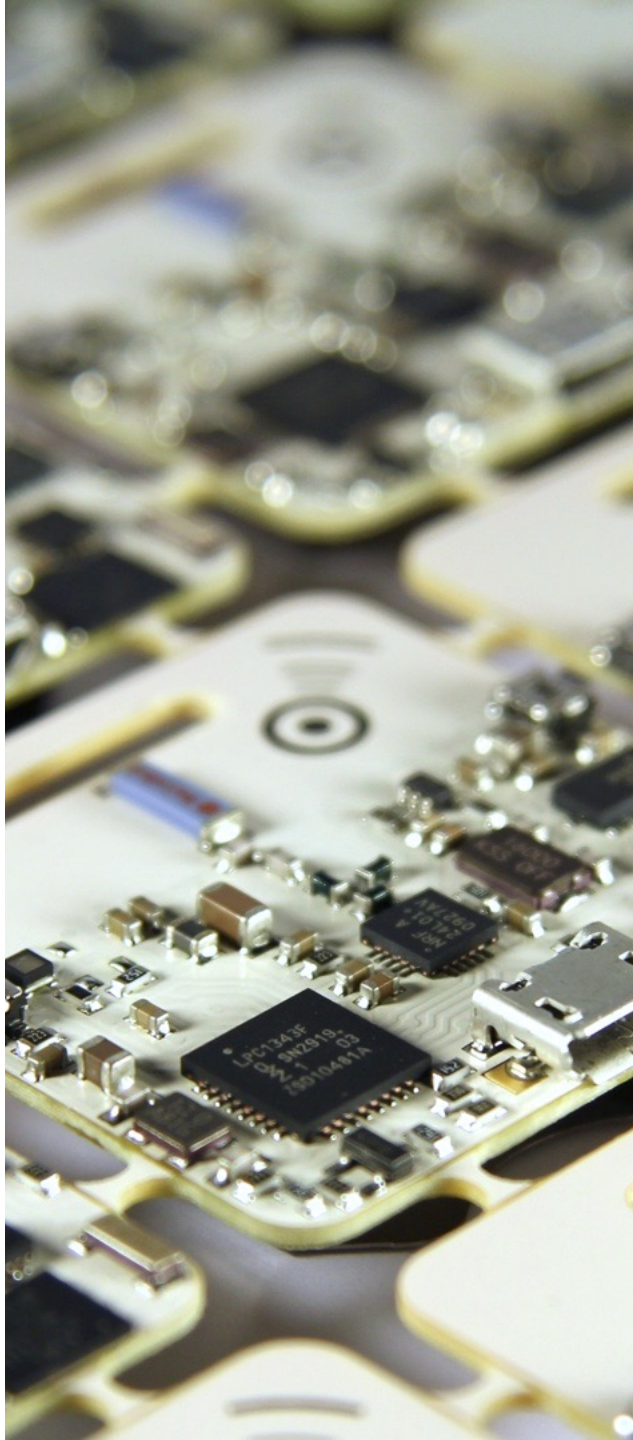
The assumption of being hacked at some point requires a solid mitigation strategy for simple, reliable and inexpensive updates

Do our defenses scale with our attackers?

Assume you can't prevent it

Value of bugs is expressed by value = number_of_installations x device_value. Increased value and large deployments drive attackers - especially in the IoT.

Massively parallelized security researchers/attackers vs. limited product development budgets and time frames.



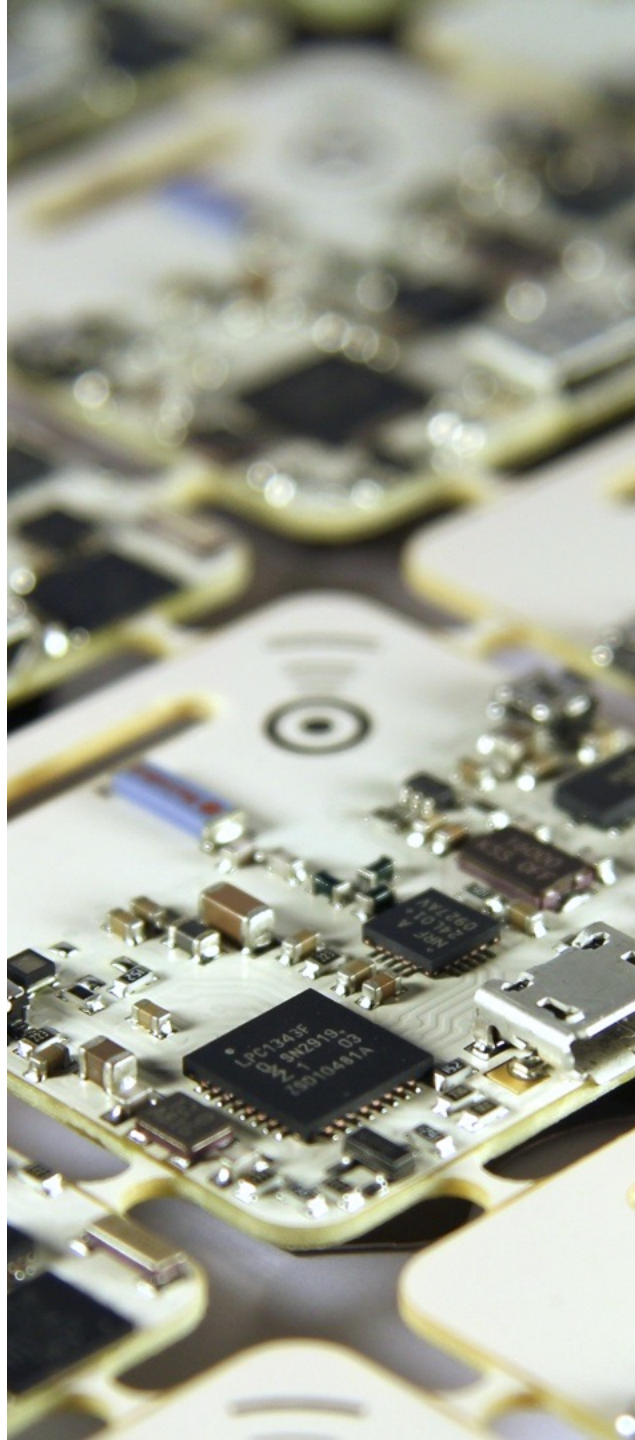
©ARM 2017

Flat memory models

A flat address space, as the result of lack of a memory management units (MMU), does not justify the absence of security.

The resulting security model is identical to ancient computer systems from the 80's – one remote exploits the whole system.

Many microcontrollers like ARM Cortex-M3/M4 provide a hardware memory protection unit (MPU) as a good alternative to a MMU.

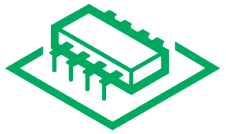


©ARM 2017

Flat memory models

No separation

Flat memory models and ignorance of MPUs prevents deployment of vital security models like “least privilege”



Escalation

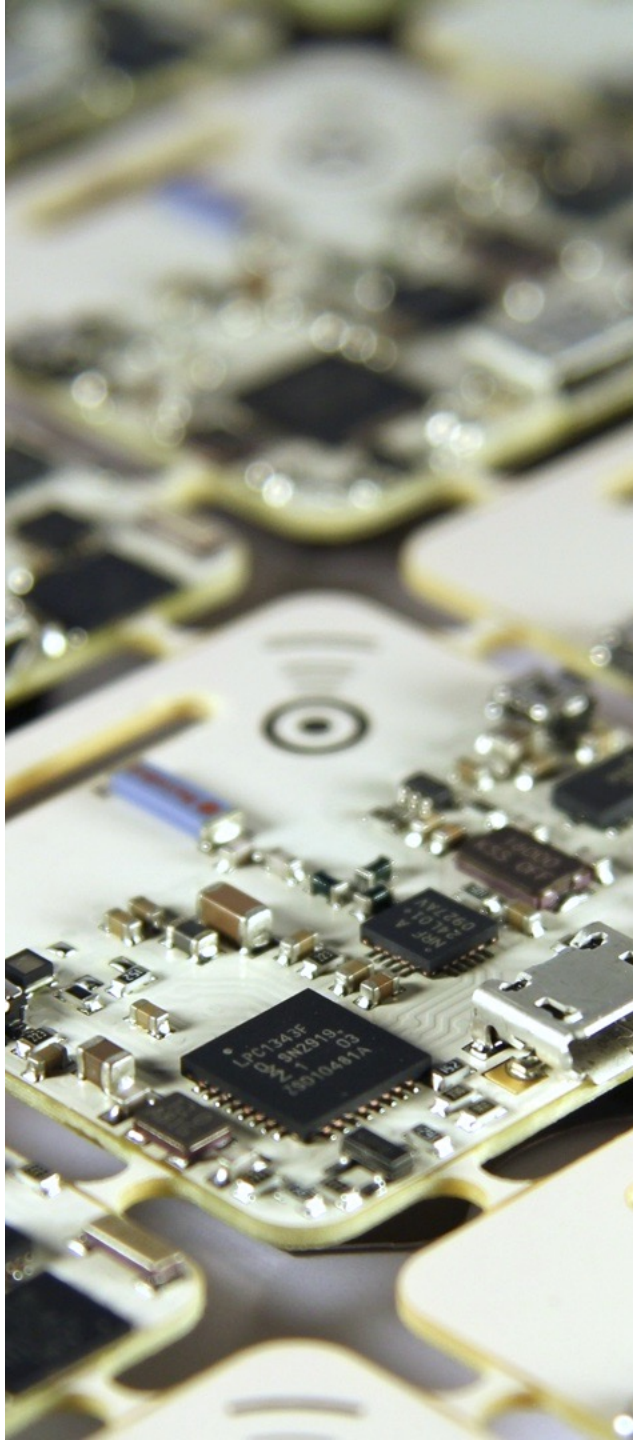
Flat memory models enable escalation and persistence of bugs by uncontrolled writing to Flash memories

Verification

Security verification impossible due to the immense attack surface and lack of secure interfaces between system components

Leakage

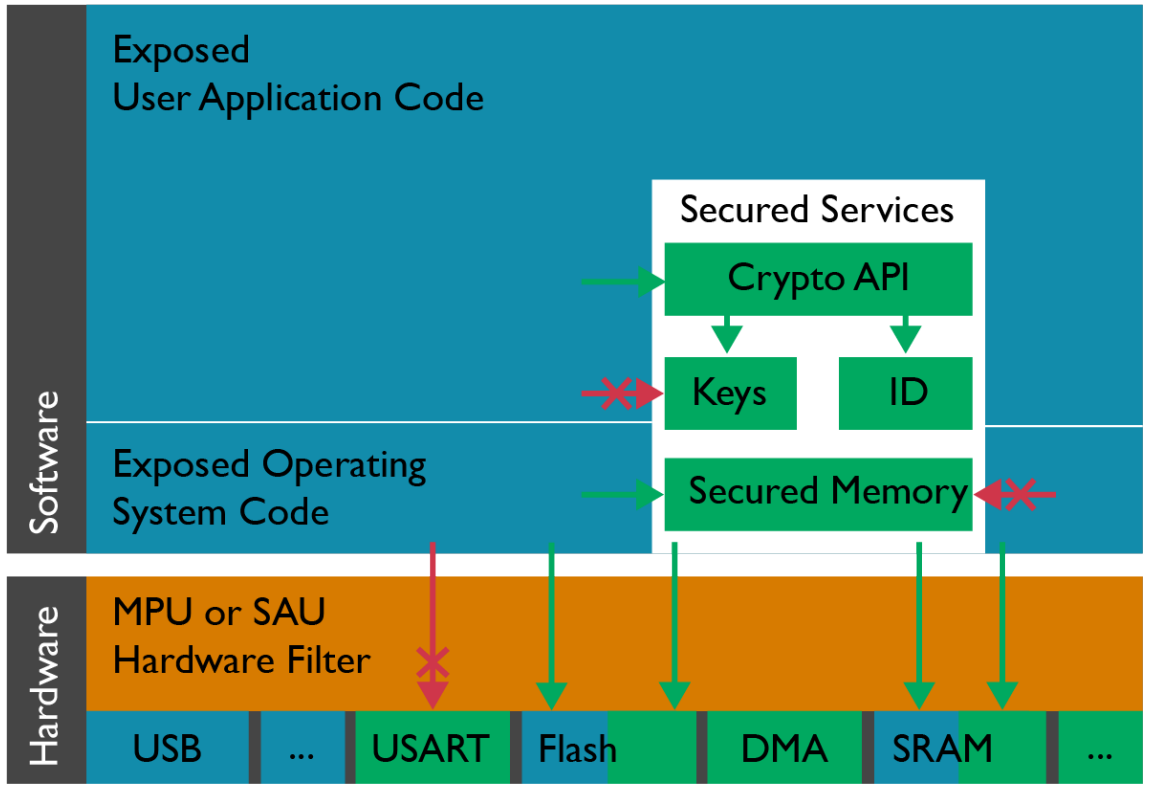
“All your bases are belong to us” – thanks to leakage of device secrets like identity keys or even class secrets.



©ARM 2017

Securing microcontrollers?

- Compartmentalization of threads and processes on microcontrollers
- Private stack and data sections
- Initialization of memory protection unit based on process permissions:
  - Whitelist approach – only required peripherals should be accessible to each box
  - Each box must have private .bss data and stack sections
- Switch execution to non-secure side, continue boot unprivileged to initialize OS and libraries to reduce the attack surface



©ARM 2017

TrustZone for ARM ARMv8-M simplifies MCU security

©ARM 2016

ARM TrustZone Technology

Bringing ARM security extensions to the embedded world

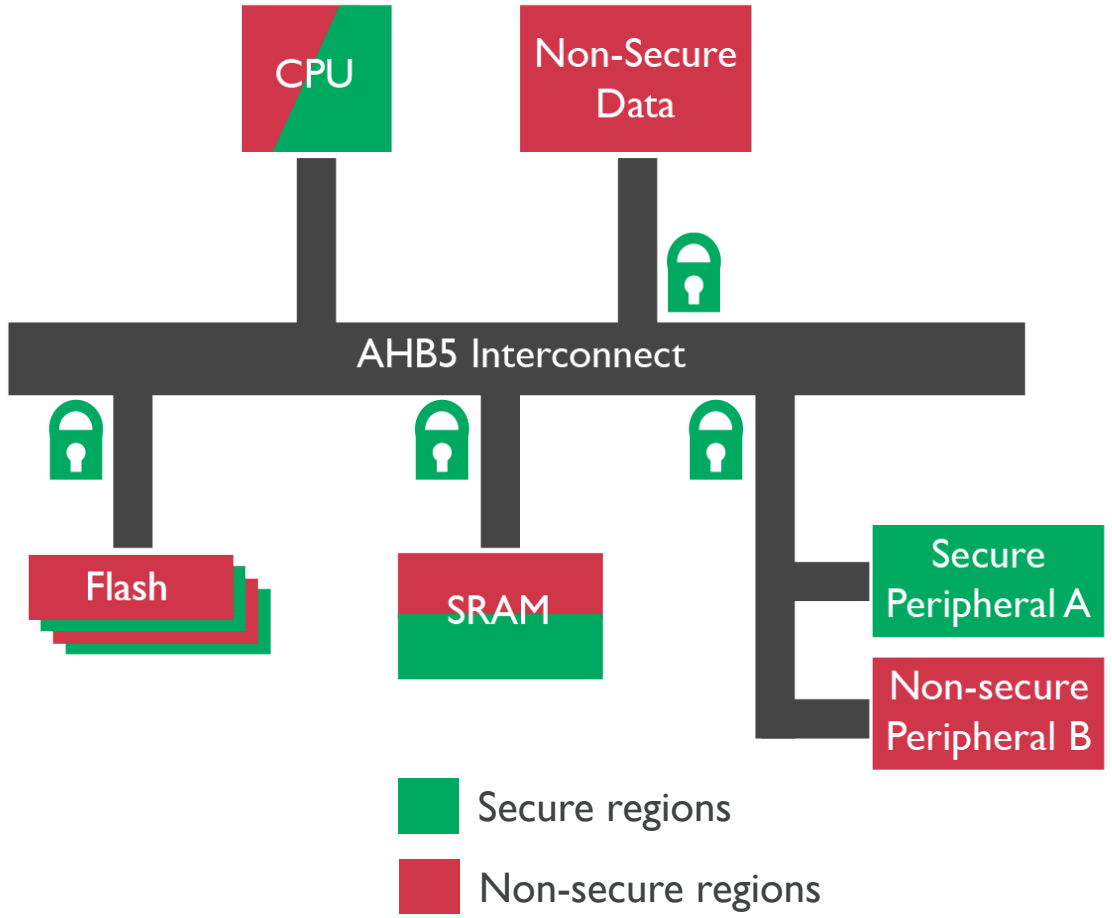
ARM TRUSTZONE
System Security

- Optional security extension for the ARMv8-M architecture
  - Security architecture for deeply embedded processors
  - Enables containerisation of software
  - Simplifies security assessment of embedded devices
- Similar and compatible to existing TrustZone technology
  - New architecture tailored for embedded devices
    - Preserves low interrupt latencies of Cortex-M
    - Provides high performance cross-domain calling

©ARM 2017

Applying ARM TrustZone to microcontrollers

- ARMv8-M architecture as used in Cortex-M23 and Cortex-M33 systems
- Introduced system level hardware security features
- Security-aware DMA peripherals and debugging
- Security extended to memories and peripherals through bus filters
  - Memory protection controllers (MPC)
  - Peripheral protection controllers (PPC)
- CPU can run in secure and in non-secure states, with visibility for all bus peripherals
  - Efficient and privacy-enabled transitions between the two security modes
  - Two instances of the interrupt vector table, one for exclusive use on the secure side.
- Stack overflow protection



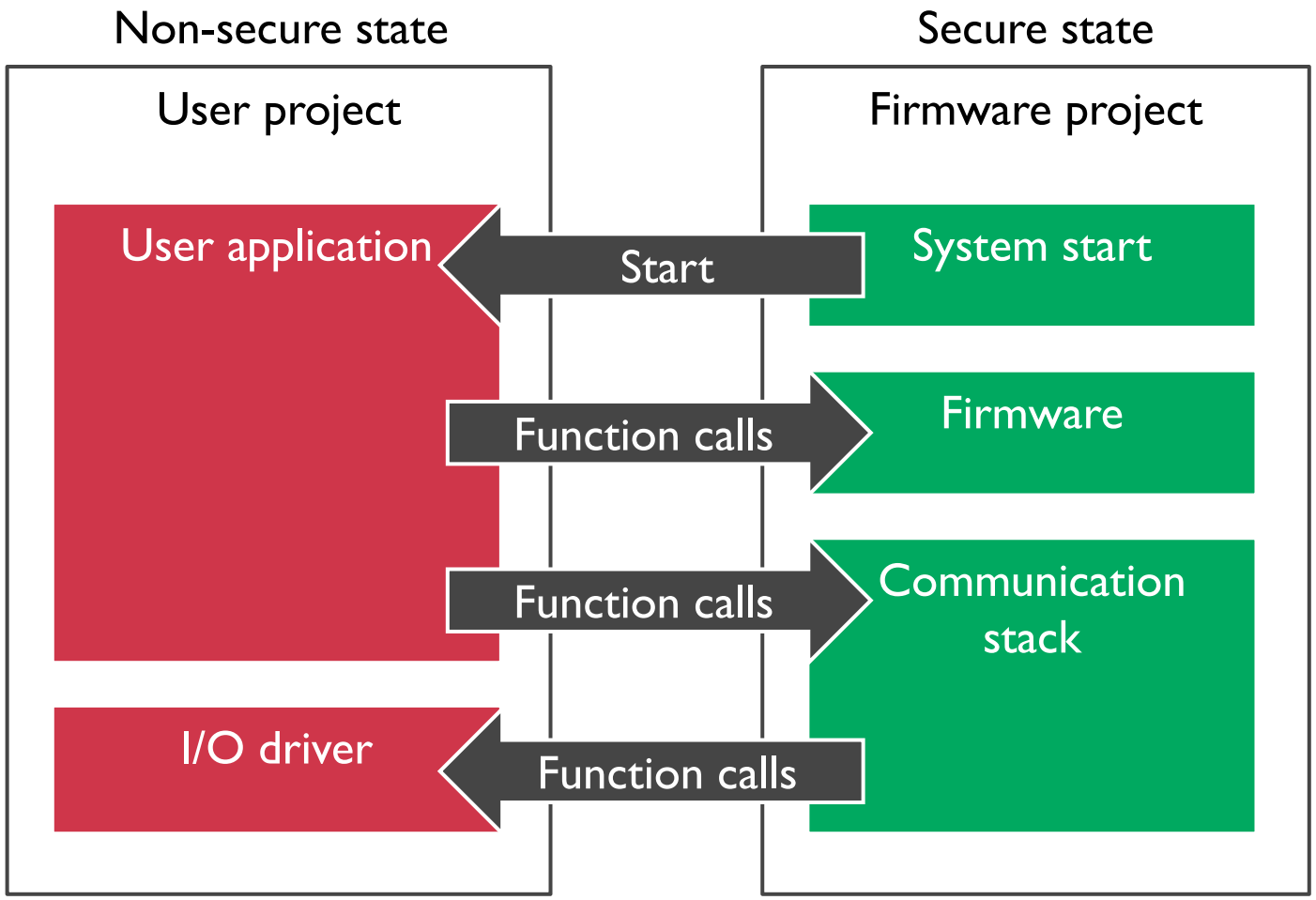
©ARM 2017

Designing system security for microcontrollers

©ARM 2016

A simplified use case

Composing systems from secure and non-secure security domains



- Non-secure side cannot access secure resources.
- Secure side has access to everything on the system.
- Secure and Non-secure processes may implement independent scheduling.

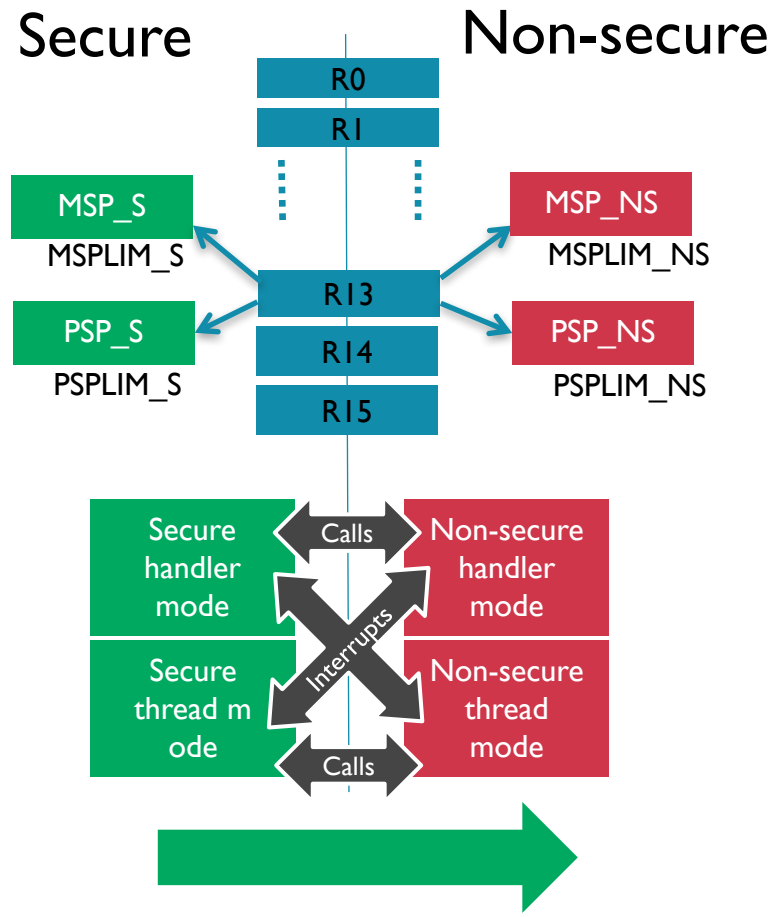
©ARM 2017

About TrustZone for ARMv8-M

Security extension for next generation Cortex-M processors

ARM TRUSTZONE
System Security

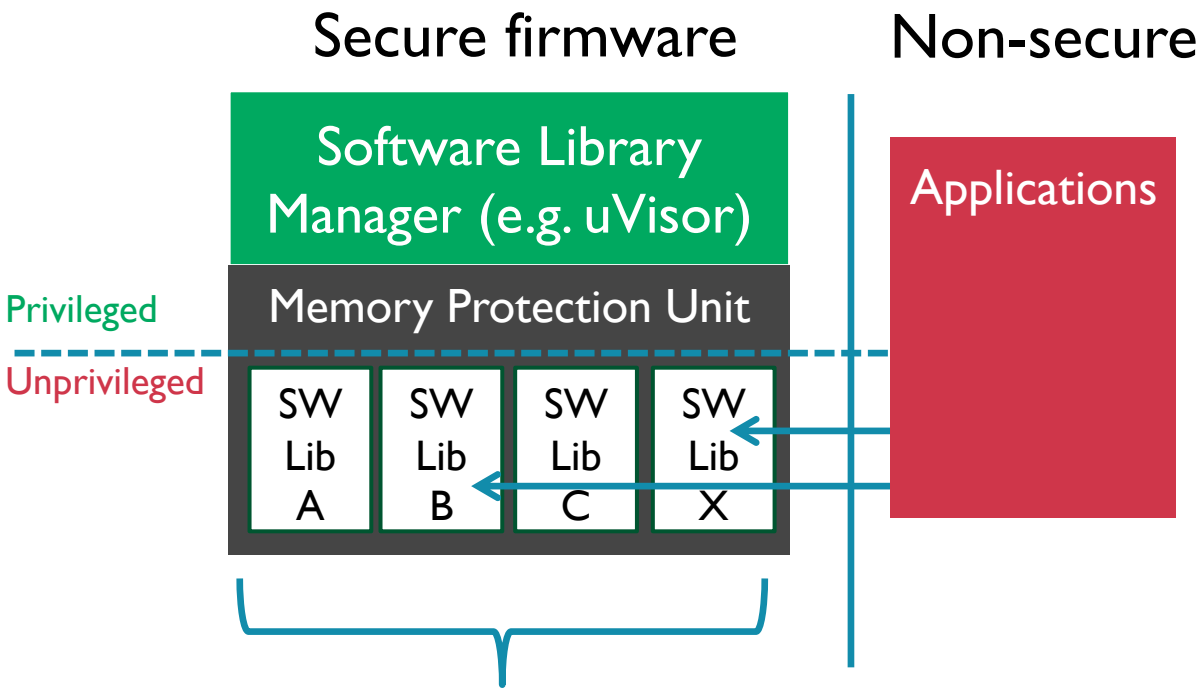
- A security barrier
  - Secure and Non-Secure code runs on the same core
  - Non-Secure code cannot access secure resources
- Allows direct function calls between domains
  - Non-Secure to Secure API functions
    - Entry points protected use Secure Gateway (SG) instructions, non-secure side can't jump past security checks on the secure side as a result
    - Entries only permitted in special secure memories with Non-Secure-callable attribute (NSC)
  - Secure to Non-Secure API functions
    - BLXNS instruction and FNC_RETURN used by the secure side to perform call-backs into the non-secure side



©ARM 2017
ARM

Support for multiple secure software libraries

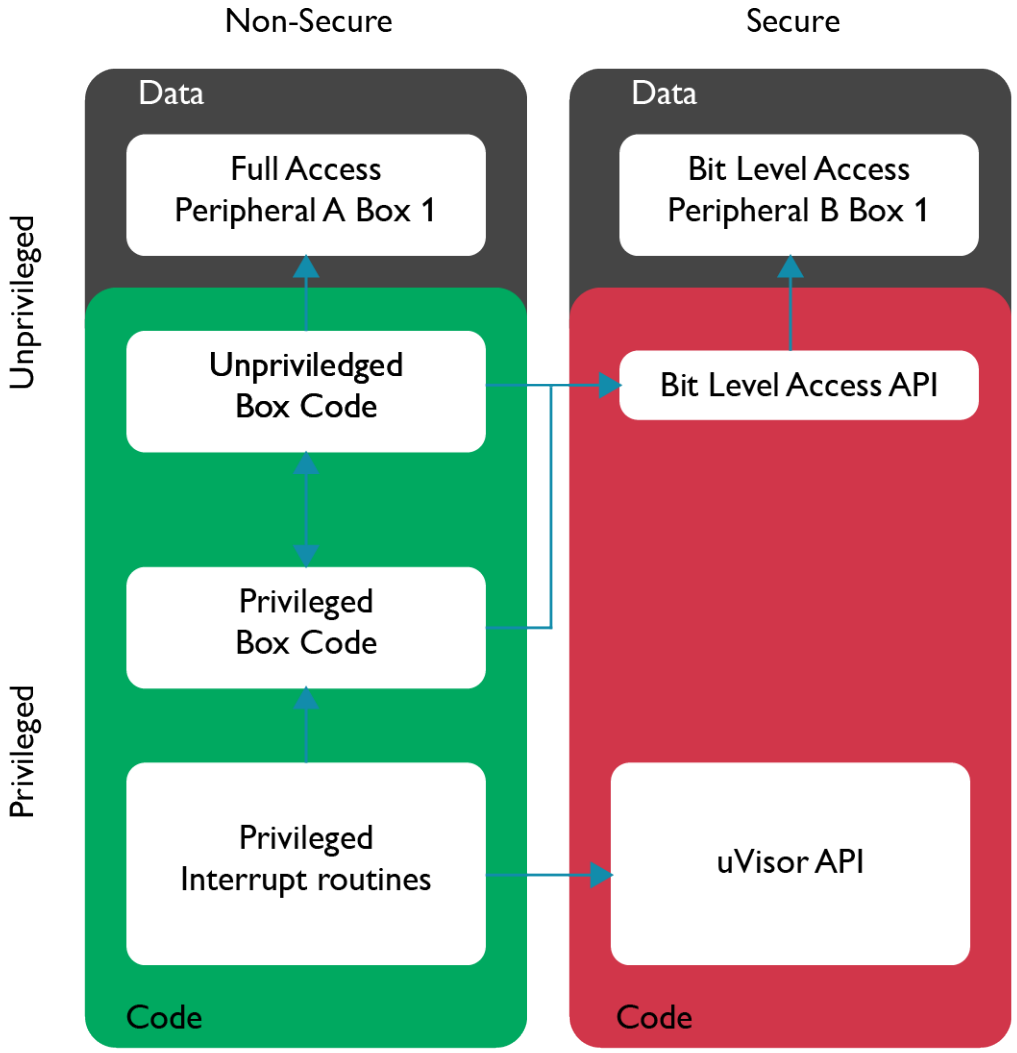
- Secure software library manager to handle context switching
  - Requires Secure privileged and unprivileged levels, and Secure MPU
- Calling secure APIs from currently selected library
  - API executes without delay
- Calling secure APIs from another Secure library
  i. Triggers Secure memory management exception
  ii. Library manager switches context to new MPU configuration
  iii. API call resumes



©ARM 2017
ARM

Secure device services on TrustZone: SDS

- Use hardware-accelerated security context switching for low-latency system services
  - Secure Interrupt management
  - Secure GPIO access (pin-wise access)
  - Register Level / Bit Level access gateway
  - IPC
  - DMA-APIs
  - Shared Crypto Accelerators / Crypto API
  - Random Entropy Pool Drivers
  - Key Provisioning / Storage
  - Configuration Storage APIs
  - … and many more



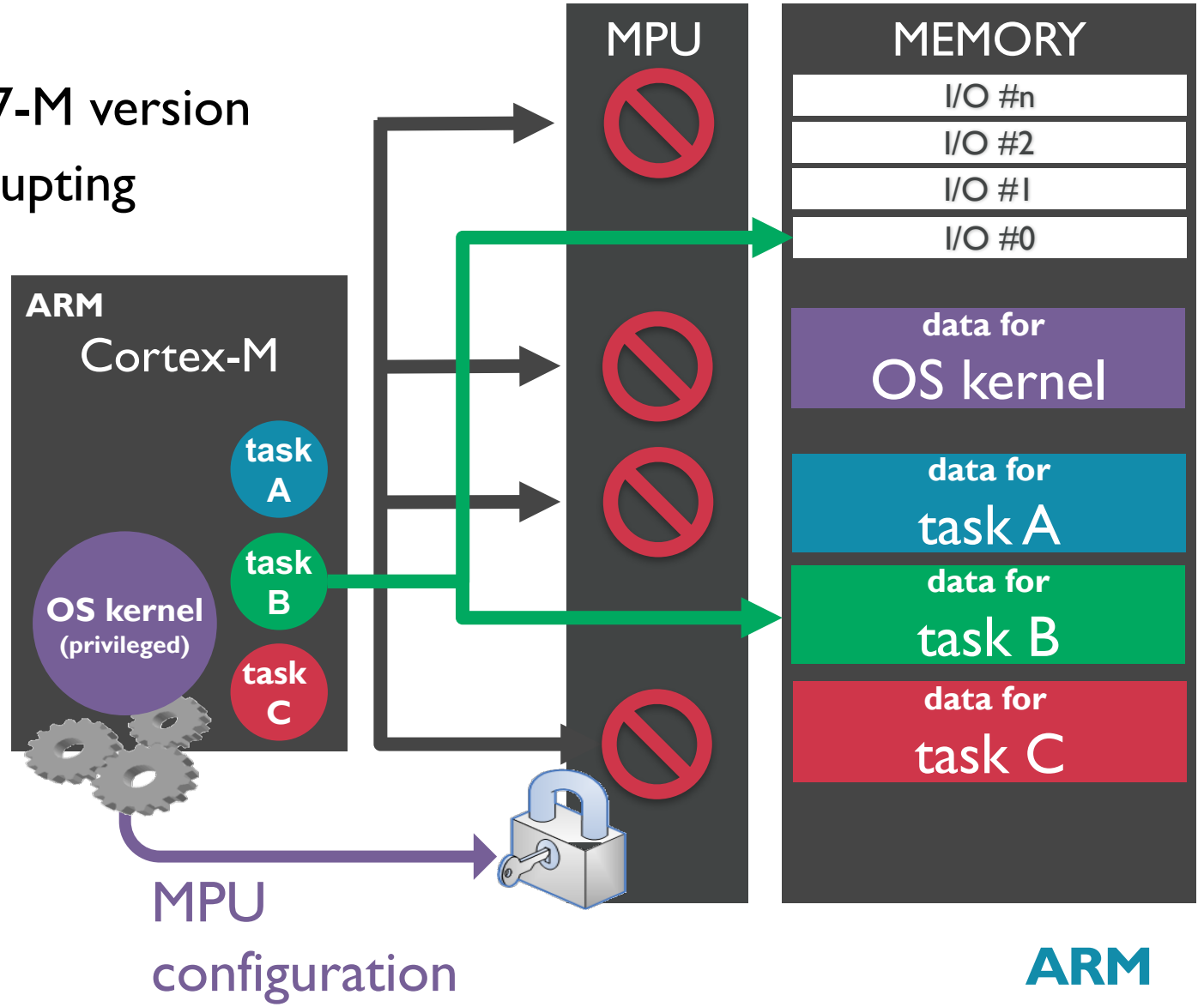
©ARM 2017
ARM

ARMv8-M provides system security

©ARM 2016

Memory protection unit (MPU)

- Major update from previous ARMv7-M version
- Prevents application task from corrupting OS or other task data
  - Improves system reliability
- User-configurable regions
  - Address
  - Size
  - Memory attributes
  - Access permissions
- Optional in all Cortex-M processors (except Cortex-M0)



©ARM 2017
ARM

ARMv8-M MPU features

- MPU configuration registers is banked between Secure/Non-secure states
  - When running Secure code – use Secure MPU settings (MPU_S)
  - When running Non-Secure code – use Non-Secure MPU settings (MPU_NS)

- Both MPUs are optional
  - Possible to have no MPU, an MPU in one domain or MPUs in both domains
  - Region count of MPUs can be different between different domains
  - Support 0 (no MPU), 4, 8, 12 and 16 regions + background regions

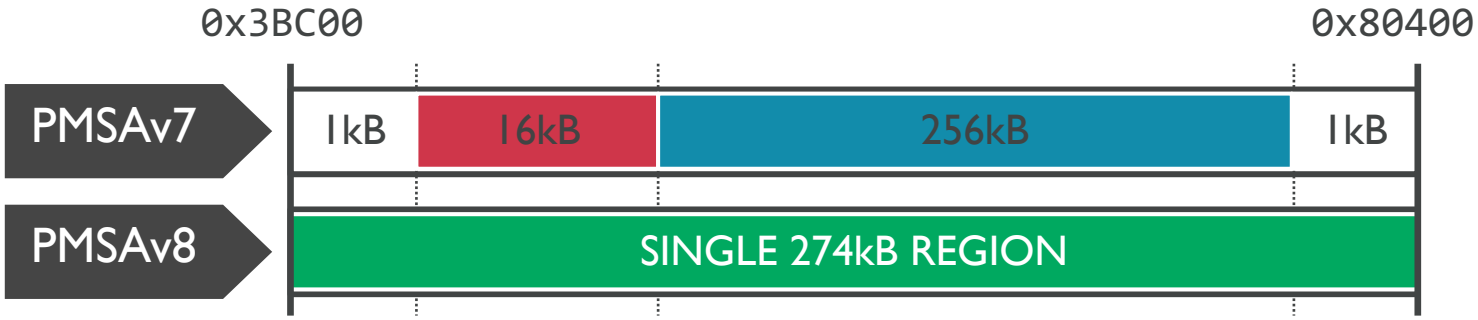
- Software can use TT (Test Target) instruction to check MPU attributes for enabling portable software security models and security-aware libraries

©ARM 2017

PMSAv8 (Protected Memory System Arch. v8)

Improved programmability and flexibility

- ARMv8-M adopts base and limit style comparators for regions
  - Replaces previous power-of-two size, sized aligned scheme
    - Simplifies software development, encouraging creation of safer software
    - Accelerates programming, potentially reducing context switch times.



  - MPU configurable down to 32-byte granularity.
  - New Device memory attribute definitions (ARMv8 alignment)

©ARM 2017

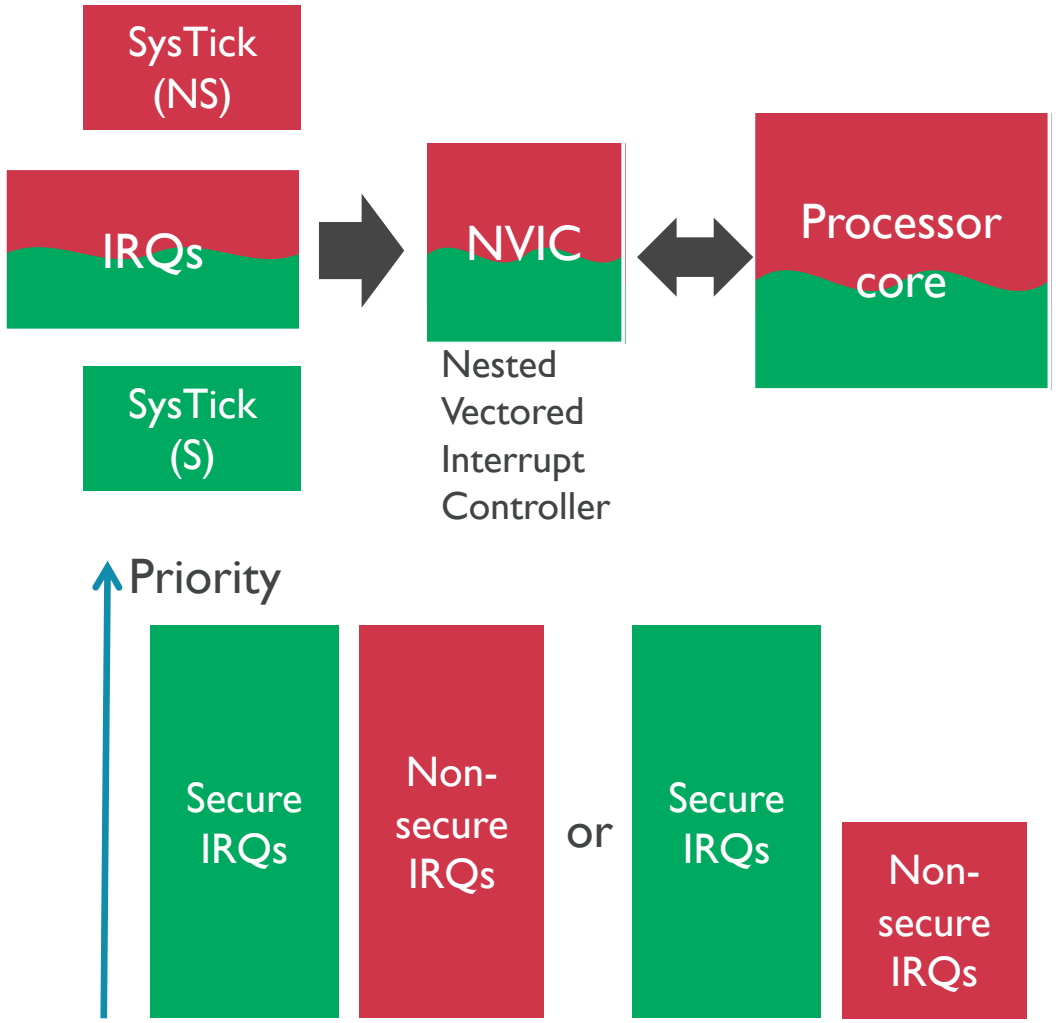
The ARMv8-M MPU: key changes to ARMv7-M

- The size of an MPU region can be any size in the granularity of 32 bytes. The previous restriction where region size must be 2^N is removed.
- Sub-region-disable removed, regions are continuous now
- Support for overlapping regions removed (apart from background region)
  - region priorities are not needed any more
- New attribute definitions for device memories
- Memory regions define memory attribute using an index value which is then looked up in a set of memory attribute registers

©ARM 2017

Interrupt handling in ARMv8-M

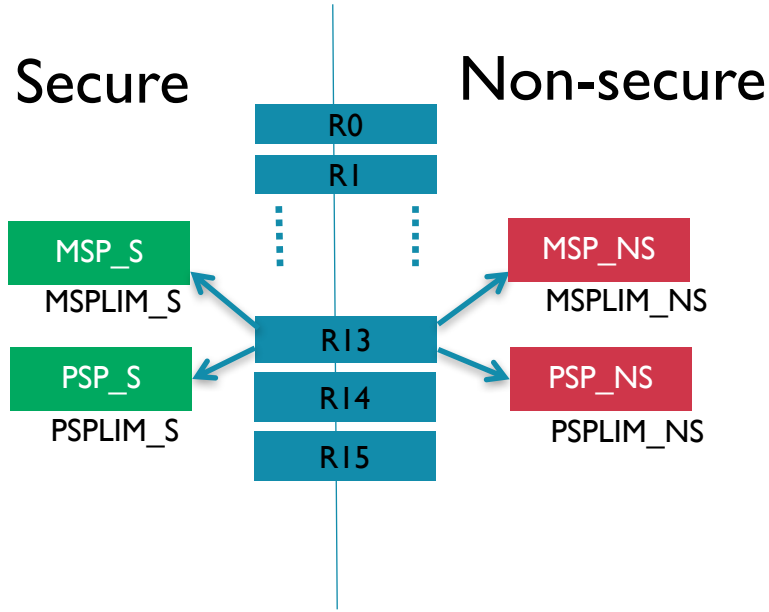
- Each interrupt can be assigned toSecure or Non-Secure side
- Some system exceptions are banked
  - e.g. SysTick timer is banked
- Secure interrupts can be programmed to have higher priority than Non-secure IRQs
- Same operations as in ARMv7-M in most cases (no extra latency)



©ARM 2017
ARM

Hardware stack protection: Stack limit registers

- Stack limit registers are special registers
  - Access using MRS, MSR
- In ARMv8-M the "Mainline" profile (Cortex-M23):
  - 4 stack limit registers
- In ARMv8-M the "Baseline" profile (Cortex-M33):
  - Stack limit for Secure SPs only
  - Prevent secure stack overflowed into NS
  - NS software can use MPU for stack limit



©ARM 2017
ARM

Defining system level security with the SAU

- The security attribution unit (SAU) and the implementation defined attribution unit (IDAU) define access permissions for Secure/Non-Secure separation:
  - MPU defines
    - Access permissions based on Privileged/Unprivileged separation
    - Memory attributes (e.g. cache policy, XN)
  - Both of them
    - Based on (pre-)configured memory regions
    - SAU/IDAU regions and MPU regions are not necessarily correlated
    - Programmable in privileged state only (word accesses only)

©ARM 2017

Defining system level security with the IDAU

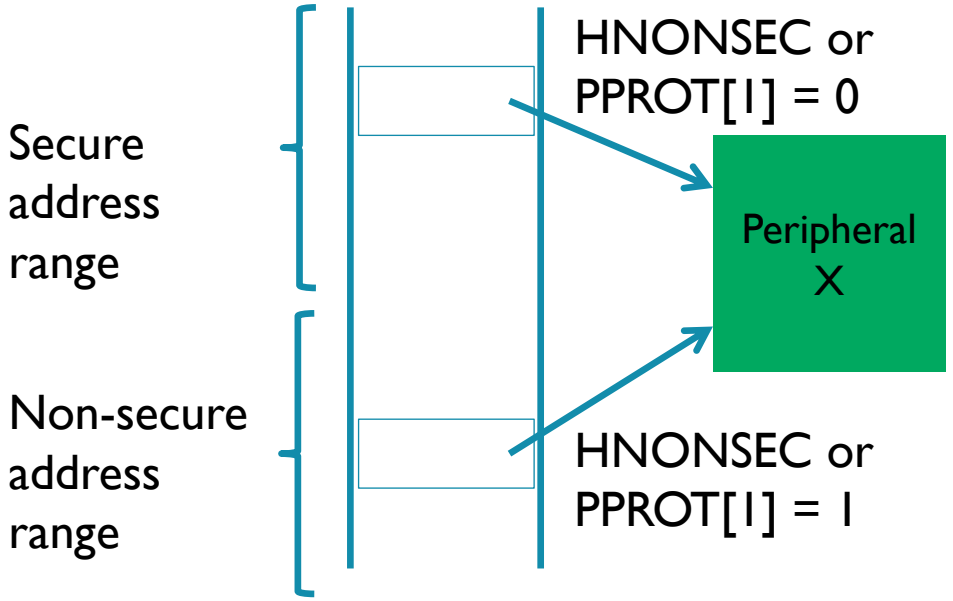
- Use IDAU for primary memory partitioning.
- Minimal TrustZone setup
  - Allows to use IDAU interface only, with zero SAU regions
  - Memory partitioning using the Memory Protection Controller (MPC)
  - Peripheral management using the Peripheral Protection Controller (PPC)
  - Fixed memory space for secure code entries: Non-Secure Callable (NSC)
- Typical MCU setup is to use the IDAU interface and 8 SAU regions
  - Memory partitioning using Memory Protection Controller (MPC)
  - Peripheral management using Peripheral Protection Controller (PPC)
  - Secure memory set to NSC by programmable registers, use SAU to override to Secure, leaving small amount of memory as NSC

©ARM 2017

Security-aware peripherals

Default design approach – Secure and Non-secure aliases

- A peripheral can be designed to have different programming views when accessed in Secure state and in Non-Secure state
  - Accessible in Secure address and Non-Secure address
  - AHB5 / APB4 is needed for TrustZone support
  - Peripherals should be placed in Non-executable address spaces (to prevent code injection)



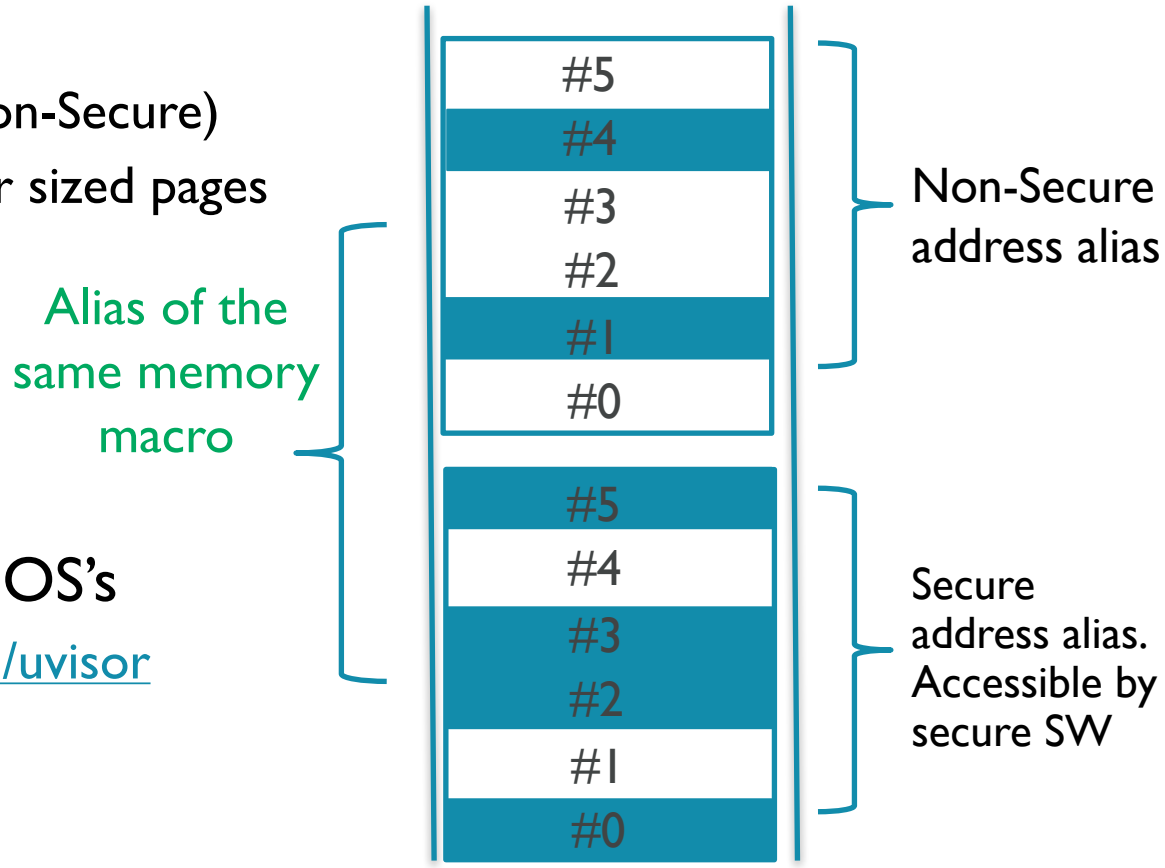
Note: HNONSEC is an AHB5 signal. PPROT is an APBv2 signal with same information.

©ARM 2017

Block base memory protection controller: MPC

- Partition address space
  - Break down memory space into pages
  - Each page visible only on one alias (Secure or Non-Secure)
  - Ideal for embedded flash - with flash erase sector sized pages
  - Only one configuration-register-bit per page to control access

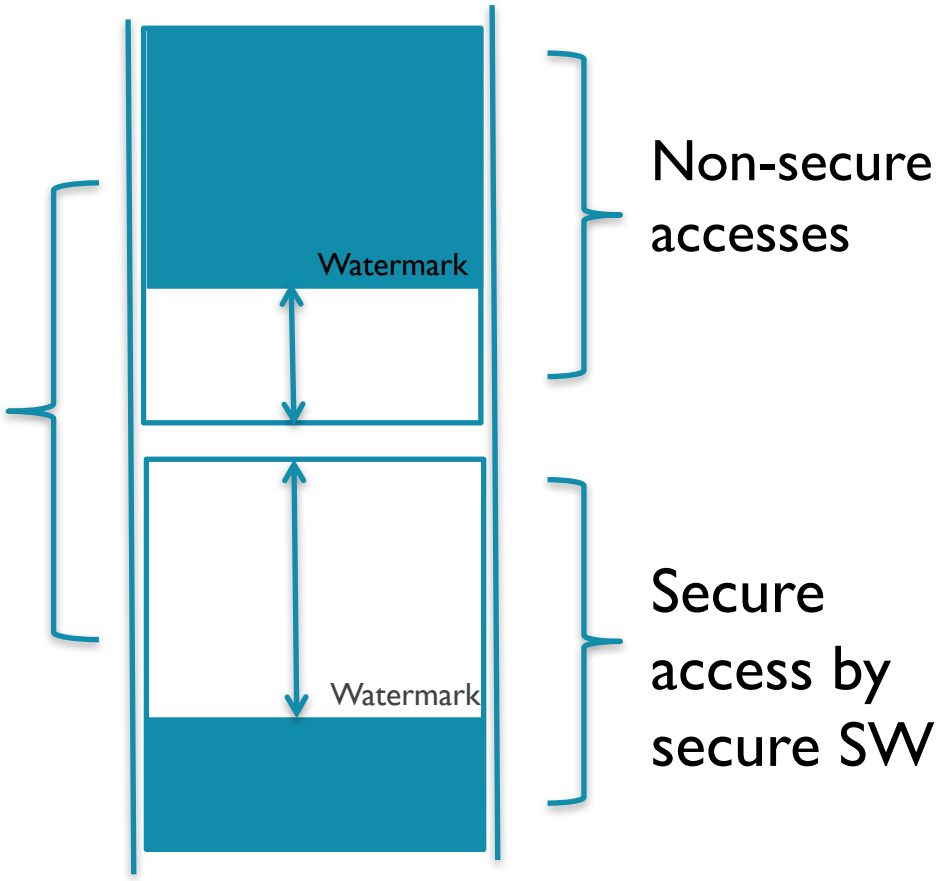
- Suitable for SRAM protection in advanced OS's
  - e.g. ARM mbedOS uVisor: github.com/ARMmbed/uvisor
  - design of gating unit can be complex



©ARM 2017
ARM

Watermark based memory protection controller

- Partition memory address space using watermark levels
  - Each memory segment is visible in one of the address aliases
  - Easy to design, low gate count
  - Less flexible in terms of memory layout
- Recommended for DDR/SDRAM
- Suitable for typical MCU software
  - Static allocation of Secure/Non-secure memories
  - Not ideal for advanced OS's

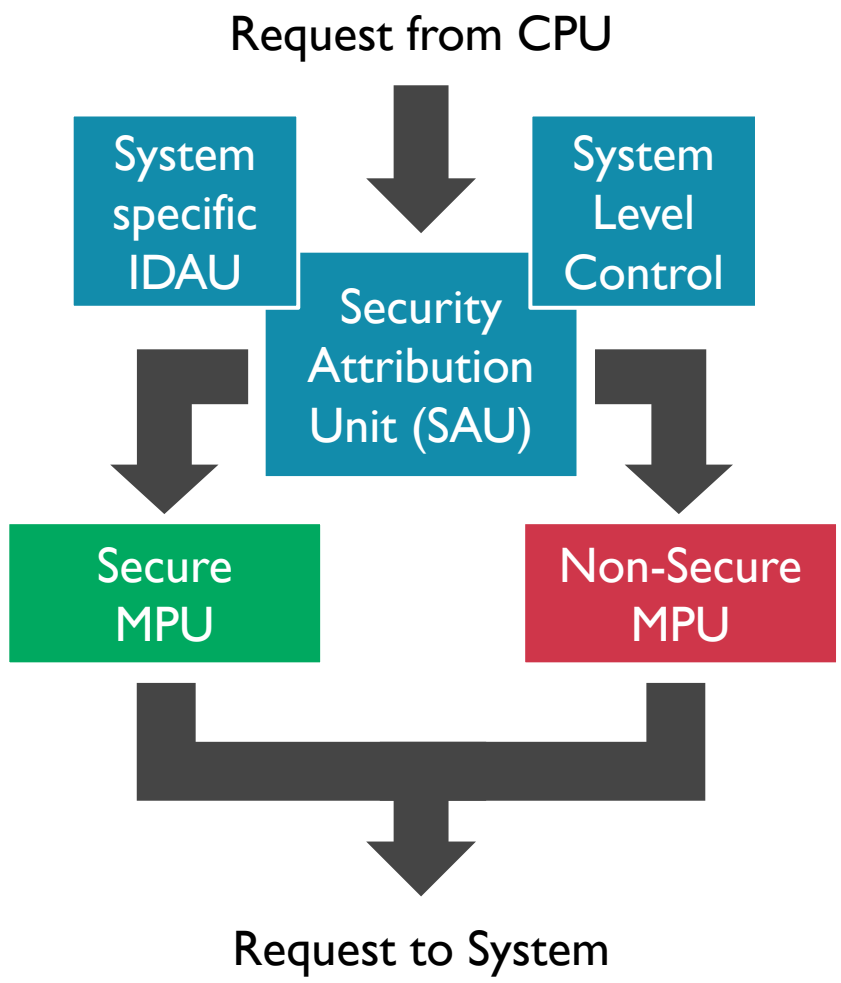


©ARM 2017

IDAU/SAU summary: Security defined by address

All transactions from core and debugger checked

- All addresses are either Secure or Non-secure.
- Policing managed by Secure Attribution Unit (SAU)
  - SAU internally similar to MPU
  - Implementation Defined Attribution Unit (IDAU) interface for adding hardware based default policing rules
  - Supports use of external system-level definitions
    - e.g. based on flash blocks or per peripheral
- Banked MPU configuration across Secure/Non-Secure
  - Independent memory protection per security state
  - Secure OS can be completely decoupled from Non-secure OS
- Load/stores acquire NS attribute based on address
  - Non-secure access attempts to a secure address results in a memory fault

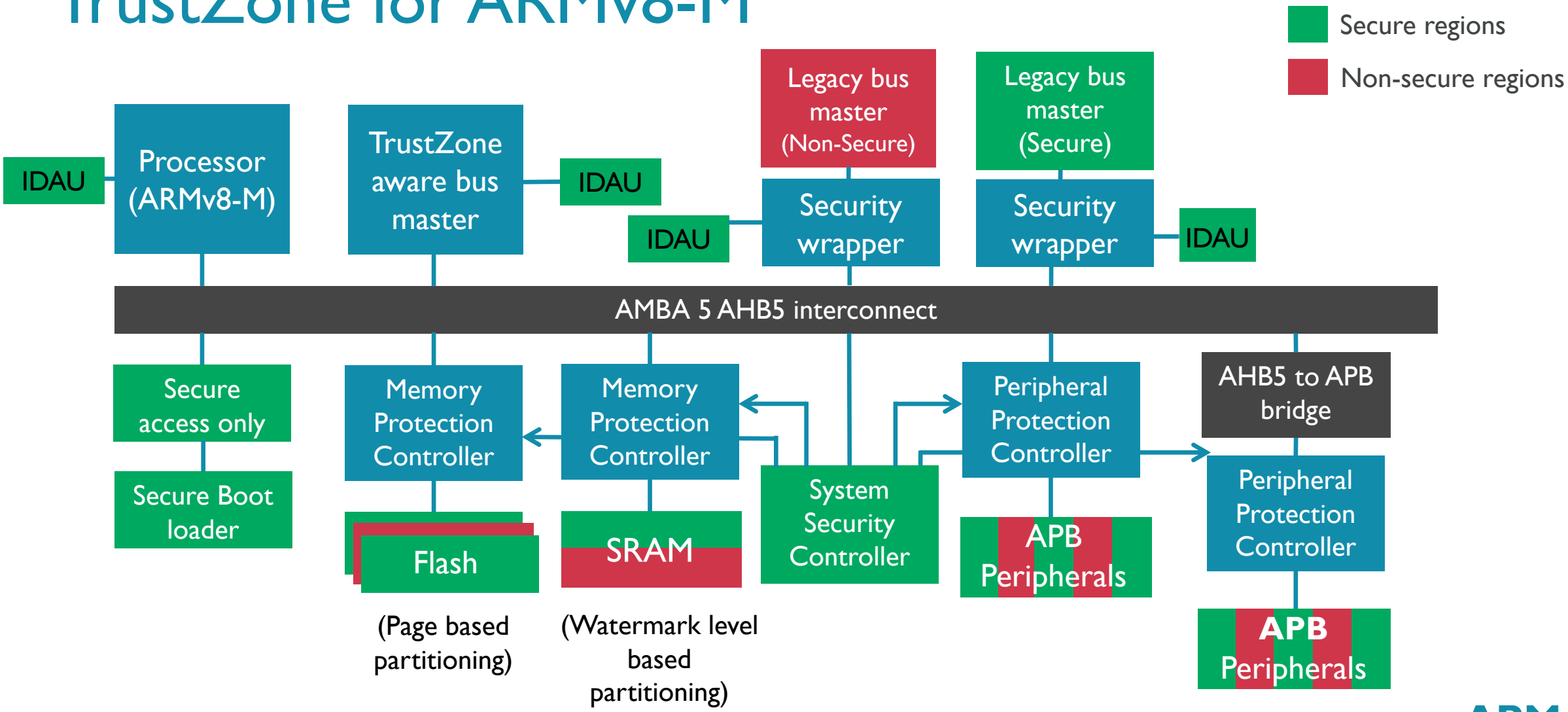


©ARM 2017

Microcontroller security: Practical use cases

©ARM 2016

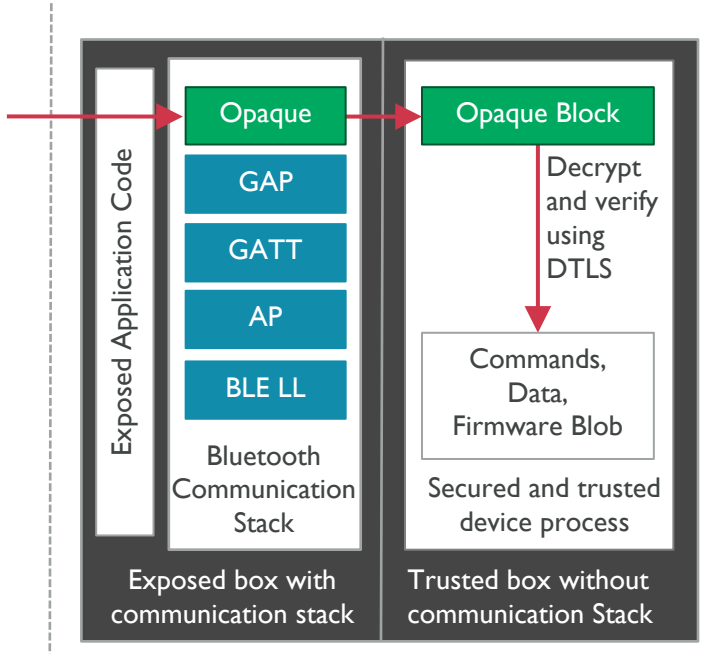
Simplified software security with TrustZone for ARMv8-M



©ARM 2017
ARM

Use case: Secure outbound communication

- Trusted messages contain commands, data or firmware parts.
- Box security not affected by communication stack exploits or infections outside of trusted box.
- Payload delivery is agnostic of protocol stack.
- Resilient box communication over the available channels:
  - Ethernet, CAN-Bus, USB, Serial
  - Bluetooth, Wi-Fi, ZigBee, 6LoWPAN

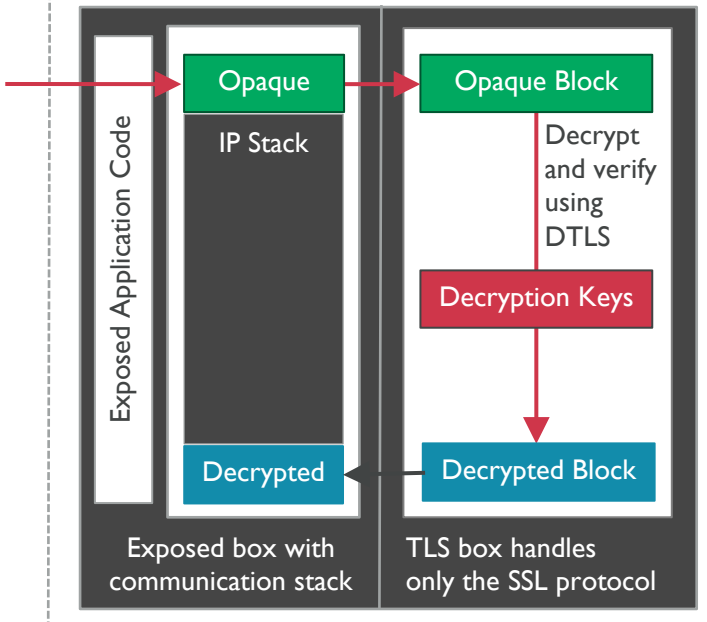


IoT Device owned by user.
Initial identity provisioned by System Integrator
Messages delivered agnostic of communication stack

©ARM 2017
ARM

Use case: Secure server communication

- Communication protected using TLS.
- Raw message payloads decrypted and verified directly by protected code:
  - TLS protocol box not exposed to communication protocol stack bugs.
  - No interference by other boxes.
  - Low attack surface.
- Authentication and encryption keys are protected against malware.
- Malware cannot interfere without knowing the encryption or signing keys.



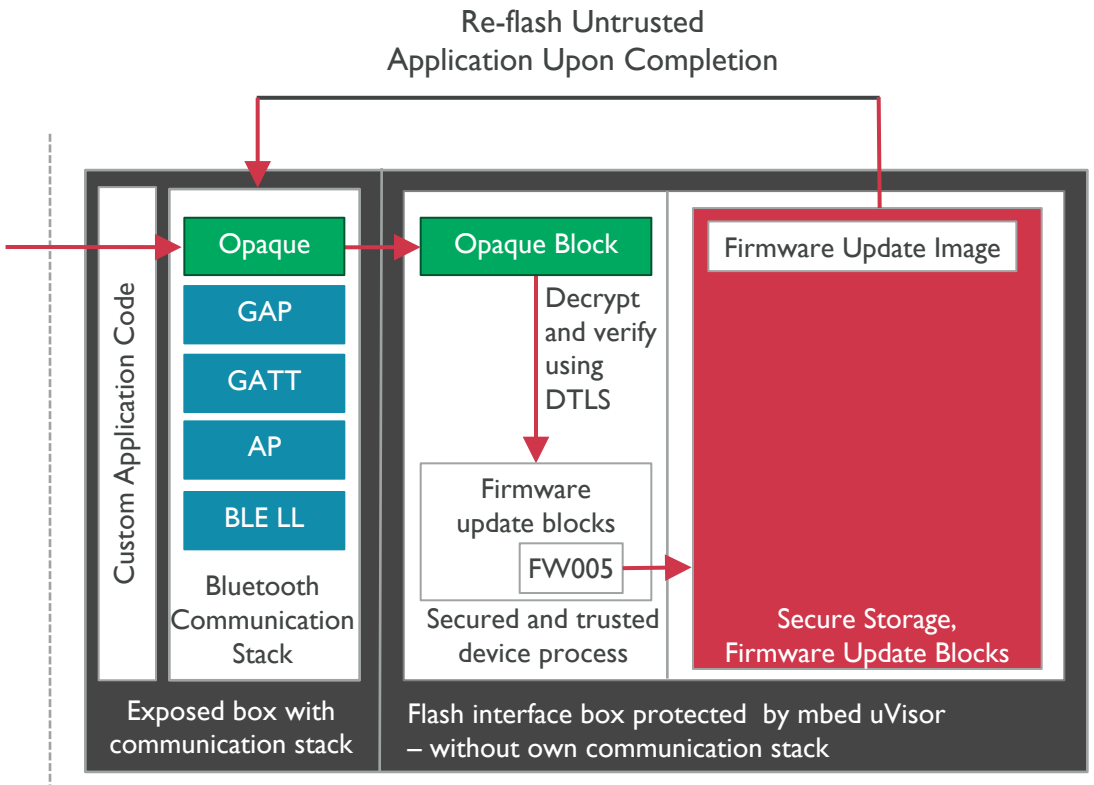
Initial keys provisioned by System Integrator. Messages decoded independent of stacks using mbed TLS in separate security context

©ARM 2017

Use case: Secure remote firmware update

Note: You can find the extended version of this slide at page 37

- Delivery of firmware update must be decoupled from a protocol-independent firmware image verification
- Bugs in communication stacks or cloud infrastructure must not compromise the firmware update
- End-to-end security for firmware updates between the firmware developer and the secure box



IoT device owned by user,
Initial identity provisioned by System Integrator,
Messages delivered independent of stacks

©ARM 2017
ARM

Use case: Controlled malware recovery

Note: You can find the extended version of this slide at page 38

- Secure box can remotely recover from malware:
  - Enforces communication through the exposed side to the server
  - Thanks to flash controller ACL restrictions, malware cannot modify monitor code or install itself into non-volatile memories
- When communication breaks with the server:
  - Parts of the device stack are reset to a known-good state
  - Disambiguation from Network failure
  - Reset prevents malware from staying on the device
  - Device switches to a safe mode to rule out network problems or to remotely update the firmware via reboot if needed

https://commons.wikimedia.org/wiki/File:Biohazard.svg

©ARM 2017

Thank you! Questions?

Download ARM's detailed paper on the topics presented https://github.com/ARMmbed/uvisor/

Follow ARMmbed uVisor development on github.com, a reference implementation for ARMv8-M security

Contact Milosch Meriac milosch.meriac@arm.com

ARM

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

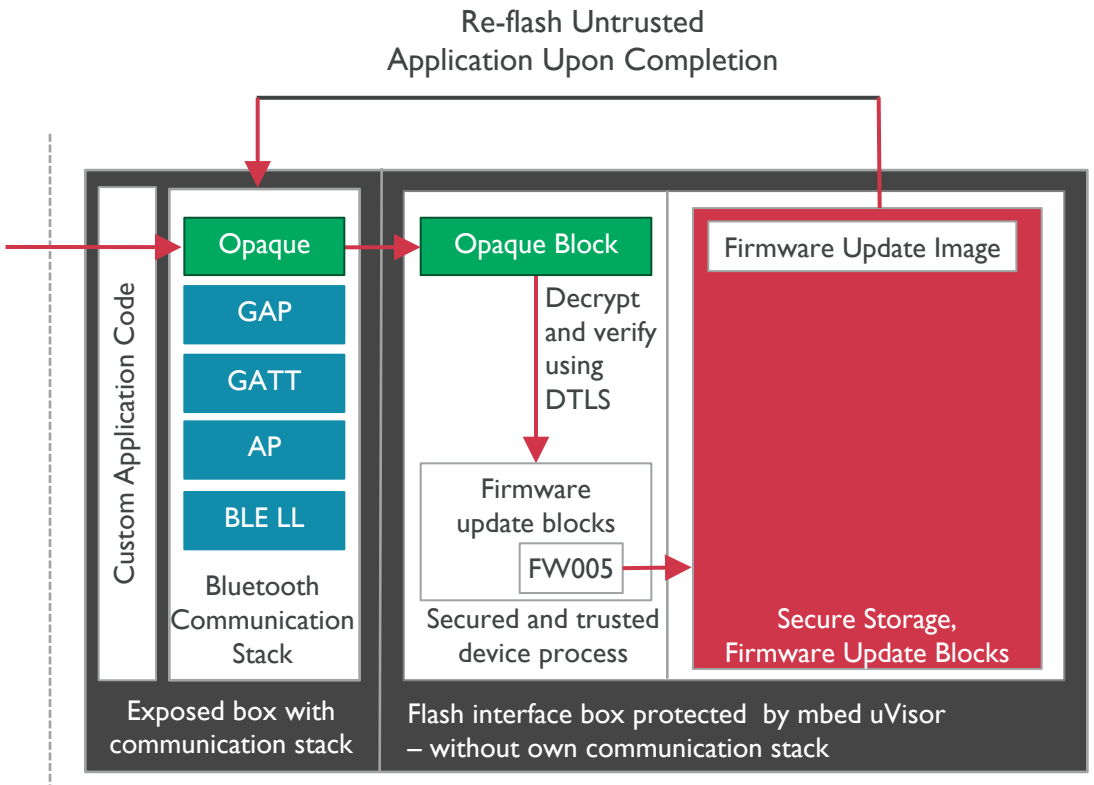
Copyright © 2017 ARM Limited

Extended Slides

©ARM 2016

Use case: Secure remote firmware update

- Delivery of firmware update must be decoupled from a protocol-independent firmware image verification
- Bugs in communication stacks or cloud infrastructure must not compromise the firmware update:
  - End-to-end security for firmware updates between the firmware developer and the secure box
  - Secure box on the device has exclusive flash-write-access
  - Box with flash controller ACLs only needs the public update key to verify validity of firmware.
  - Local malware can't forge a valid firmware signature to the firmware update box, the required private firmware signature key is not in the device.



IoT device owned by user,
Initial identity provisioned by System Integrator,
Messages delivered independent of stacks

©ARM 2017
ARM

Use case: Controlled malware recovery

- Secure box can remotely recover from malware:
  - Enforces communication through the exposed side to the server.
  - Receives latest security rules and virus behaviour fingerprints for detection.
  - Shares detected pattern fingerprint matches with control server
  - Distributed detection of viruses and live infrastructure attacks.
  - Thanks to flash controller ACL restrictions, malware cannot modify monitor code or install itself into non-volatile memories.
- When communication with the server breaks for a minimum time:
  - Disambiguation from Network failure
  - Parts of the device stack are reset to a known-good state.
  - Reset prevents malware from staying on the device.
  - Device switches to a safe mode to rule out network problems or to remotely update the firmware via reboot if needed.

https://commons.wikimedia.org/wiki/File:Biohazard.svg

©ARM 2017